The Downward Refinement Property

Fahiem Bacchus and Qiang Yang*
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L-3G1

Abstract

Using abstraction in planning does not guarantee an improvement in search efficiency; it is possible for an abstract planner to display worse performance than one that does not use abstraction. Analysis and experiments have shown that good abstraction hierarchies have, or are close to having, the *downward refinement property,* whereby, given that a concrete-level solution exists, every abstract solution can be refined to a concrete-level solution without backtracking across abstract levels. Working within a semantics for ABSTRIPS-style abstraction we provide a characterization of the downward refinement property. After discussing its effect on search efficiency, we develop a semantic condition sufficient for guaranteeing its presence in an abstraction hierarchy. Using the semantic condition, we then provide a set of sufficient and polynomial-time checkable syntactic conditions that can be used for checking a hierarchy for the downward refinement property,

1 Introduction

Plan formation is concerned with finding sequences of operators that bring about certain goal states from certain initial states. This task is particularly difficult because of the exponential nature of the search spaces involved. Abstraction [1, 2, 3, 4) is a widely adopted strategy for lessening this computational burden.

It is well-known that if one has a good abstraction hierarchy, then search can be made exponentially more efficient. Korf [5] has shown that by using macro-operators (a form of abstraction), the average time complexity of planning search can drop from exponential to linear. Similar speed-up results have been reported by Knoblock [2] for ABSTRIPS-style abstraction. However, close examination reveals that the *downward refinement property* (DRP) is a major assumption underlying both analyses. This property simply states that if a non-abstract, concrete level solution to the planning problem exists, then *any* abstract solution can be refined to a concrete solution *without backtracking* across abstraction levels. That is, once a solution is found at the abstract level it need never be reconsidered, just refined.

Furthermore, experiments with Abstrips [1] and Abtweak [4] have shown that abstraction only increases search efficiency in hierarchies that are close to having the DRP, i.e., in hierarchies where most abstract solutions can be refined. In hierarchies where this is not the case, using abstraction can in fact decrease the efficiency of the planner. Hence, it is important to characterize those hierarchies that have the DRP. Such a characterization could be to used to *check* whether a given abstraction hierarchy has the DRP. In the presence of the DRP a modified search strategy is applicable that runs exponentially faster without giving up completeness. A characterization could also be used to *generate* an abstraction hierarchy for a given domain guaranteed to have the DRR

In this paper, we will provide a semantics for ABSTRIPS-style abstraction. We will then use this semantics to give a definition of the DRP. After examining the effect of the DRP on search efficiency, we will give a semantic condition that is sufficient to ensure the DRP. This semantic condition yields both a better understanding of the nature of the DRP and collection of syntactic conditions that are sufficient for guaranteeing its presence. To be useful, we have focused on conditions that can be tested in polynomial time, allowing a specification of a domain hierarchy to be checked quickly for the DRP.

2 The Planning Representation

We will focus on planning problems that can be described with a quantifier-free language, L, consisting of a collection of predicates, of various arities; constants; and variables, used to describe parameterized operators. Such a language can be given a semantics by a traditional first-order model, with a domain of discourse, relations over the domain, and an interpretation function mapping the symbols of the language to semantic entities. The result of such a model will be the assignment of a truth-value to every formula of the language. Abstracting away from the models we can focus on their end product: the truth-value assignments. Treating these assignments as functions from the formulas to TRUE/FALSE, we can view distinct truth-value functions as alternate

*This work is supported by grants from the Natural Science and Engineering Council of Canada. The authors' e-mail addresses are fbacchuslogos.waterloo.ca and qyangClogos.waterloo.ca.

realities or possible worlds as described by $\mathcal{L}$. We will call these distinct truth-value functions possible worlds, and will use $\mathcal{W}$ to denote the set of *all* possible worlds. We say that a possible world w *satisfies* a formula $\lambda \in \mathcal{L}$ (set of formulas S) if it assigns the truth-value TRUE to λ (to every member of S), and we will use $w \models \lambda$ to denote this.

Now consider a collection of formulas of $\mathcal{L}$, S. If this collection is consistent (i.e., if there is at least one possible world that satisfies S), then in general there will be many possible worlds which satisfy S. We will use the notation Σ_S to refer to the set of *all* worlds $w \in \mathcal{W}$ that satisfy S.[1] If we add new formulas to S then Σ_S will shrink in size, and if we remove formulas from S, Σ_S will increase.[2] Furthermore, it can be the case that a syntactically distinct set of formulas T will be such that $\Sigma_T = \Sigma_S$ (in the case where T is logically equivalent to S).

State Descriptions. In planning, $\mathcal{L}$ is used to write state descriptions: collections of formulas that are partial descriptions of the state of the world. We restrict our attention to planning problems where the state descriptions are finite, consistent collections of ground literals (a typical situation for STRIPS-style planners). By the above, each such state description, S, corresponds semantically to a set of possible worlds Σ_S. Furthermore, because of our restriction to literals we have that $\Sigma_{S1} = \Sigma_{S2}$ if and only if $S1 = S2$.[3] This follows from the fact that for each set of ground literals there exists a possible world that satisfies those literals and no other literals.

Operators. In addition to state descriptions, planning systems contain operators that, in the case of STRIPS-style planners, are partial functions between state descriptions. A STRIPS-style operator α is defined by three sets of literals, its preconditions, $\mathsf{Pre}(\alpha)$, its add list, $\mathsf{Add}(\alpha)$, and its delete list, $\mathsf{Del}(\alpha)$. Applying an operator α to a state description S results in a new state description $\alpha(S)$, which is obtained by removing from S all the literals in $\mathsf{Del}(\alpha)$, and then adding to S the literals in $\mathsf{Add}(\alpha)$. That is, $\alpha(S) = (S \setminus \mathsf{Del}(\alpha)) \cup \mathsf{Add}(\alpha)$, where '$\setminus$' is the set difference operator.

Semantically, operators correspond to partial functions between sets of possible worlds. More precisely, consider the power set of $\mathcal{W}$. A subset of this power set is the set of *describable* sets of worlds, $\mathcal{DW}$. This collection is defined by the condition that for each $\Sigma \in \mathcal{DW}$ there exists a legal state description S (finite collection of literals by the above definition) such that $\Sigma = \Sigma_S$.[4] Furthermore, as we noted above, the set of literals S is unique. Thus, an operator α corresponds semantically to a partial function $\hat{\alpha}$ from $\mathcal{DW}$ to $\mathcal{DW}$, such that:

1. $\Sigma \in \mathcal{DW}$ is in the domain of $\hat{\alpha}$ if and only if $\Sigma \subseteq \Sigma_{\mathsf{Pre}(\alpha)}$.
2. $\hat{\alpha}(\Sigma) = \Sigma_{\alpha(S)}$, where S is the unique state description corresponding to Σ.[5]

In general, a planning system will contain operator templates instead of operators. These templates contain variables which specify parameterized versions of the actual operators. The operators themselves are generated by instantiating the variables in the template. We will often refer to these operator templates as if they were actual operators. In these cases we are implicitly referring to all of the template's instantiations.

Planning Problems and Plans A plan II is a sequence of operators $\alpha_1, \ldots, \alpha_n$. A planning problem is a pair of state descriptions (J,G), where I is the initial state, and *G* is the goal state. If we apply the plan II to the initial state *I* the operators in Π will define a sequence of state descriptions *SQ*....S_n resulting from the application of the operators: $S_0 \overset{\alpha_1}{\mapsto} S_1 \overset{\alpha_2}{\mapsto} \ldots \overset{\alpha_n}{\mapsto} S_n$, where *SQ = 1.* A plan Π is a *solution* a planning problem {/,G), or is *correct* with respect to (/,G), if the sequence of state descriptions generated by applying II to I satisfies two conditions: (1) $\mathsf{Pre}(\alpha_i) \subseteq S_{i-1}$ and (2) $G \subseteq S_n$. That is, the plan is correct if the preconditions of each operator are satisfied in the state to which it is applied, and the final state satisfies the goal *G-*

Semantically, a planning problem corresponds to a pair of sets from DW, $\langle \Sigma_I, \Sigma_G \rangle$. A correct plan II for *{I,G)* corresponds to a semantic solution II consisting of a sequence of functions $\hat{\alpha}_1, \ldots, \hat{\alpha}_n$ that traverse through *VW* such that Σ_I is in the domain of a_1, each intermediate set of possible worlds, $\Sigma_{S_{i-1}}$, is in the domain of the function next applied, $\hat{\alpha}_i$, and the final set of possible worlds Σ_{S_n} is a subset of Σ_G. It is not difficult to see that every syntactic solution has a corresponding semantic solution and vice versa.

3 Abstraction

The type of abstraction we consider here is of the ABSTRIPS-style, where abstract operators are generated by eliminating preconditions. This type of abstraction has been widely used in planning research [1, 4, 9, 10]. Every literal *L* in the language *C* is assigned one of a finite number of integer criticality values denoted by *crit(L).* The number of levels of abstraction is equal to the number of distinct criticality values. In particular, let there be k+1 different criticality values corresponding to the integers {0,1,...,k}, where the highest level of abstraction is *k* and level 0 corresponds to the concrete level where no abstraction occurs.

[1]Hence, Σ_S satisfies "All I know is S" in the sense of Levesque [6].

[2]These notions mirror the possible worlds notion of knowledge, where more knowledge corresponds to a smaller set of accessible worlds [7].

[3]This would not be true if we allow arbitrary formulas in our state descriptions. For example, $\{P(a), P(a) \rightarrow Q(a)\}$ and $\{P(a), Q(a)\}$ are distinct sets of formulas but they have identical sets of satisfying worlds.

[4]Not every set of possible worlds is definable by a finite collection of literals. For example, a set consisting of a single world might require an infinite number of literals in its definition.

[5]These semantics differ in important ways from the semantics for STRIPS provided by Lifschitz [8], Future work will explore these semantics in more detail, and will treat the more general case where the restriction to literals is removed.

Abstract Operators and Plans Given these abstraction levels we can define an abstraction operator Abs which maps a state description S to its i-th level abstraction $\mathsf{Abs}(i, S)$, where $0 \le i \le k$. This abstract state is defined by removing from S all literals with criticality value less than i. We can extend Abs so that it can be applied to an operator α to yield an abstract operator $\mathsf{Abs}(i, \alpha)$. The abstract operator has the same add and delete lists as α but it has a abstracted precondition list. That is, $\mathsf{Add}(\mathsf{Abs}(i,\alpha)) = \mathsf{Add}(\alpha)$ and $\mathsf{Del}(\mathsf{Abs}(i,\alpha)) = \mathsf{Del}(\alpha)$ but $\mathsf{Pre}(\mathsf{Abs}(i,\alpha)) = \mathsf{Abs}(i, \mathsf{Pre}(\alpha))$. Extending Abs further we can apply it to plans. If Π is a plan, then $\mathsf{Abs}(i, \Pi)$ is an i-th level abstract plan where every operator $\alpha \in \Pi$ has been replaced by its i-th level abstraction $\mathsf{Abs}(i, \alpha)$. Note that the ordering of the operators has not been disturbed.

Finally, say that an i-th level abstract plan $\mathsf{Abs}(i, \Pi)$ is a i-th level *abstract solution* to the planning problem $\langle I, G\rangle$ if it is correct with respect to the problem with an abstracted goal. That is, if $\mathsf{Abs}(i, \Pi)$ is a solution to $\langle I, \mathsf{Abs}(i, G)\rangle$. A 0-th level solution is called a *concrete* solution: it involves no abstractions.

It should be noted that as a consequence of our notation $\mathsf{Abs}(0, \alpha) = \alpha$, $\mathsf{Abs}(0, S) = S$, and $\mathsf{Abs}(i, S) \subseteq \mathsf{Abs}(i-1, S)$. Further, if Π is correct with respect $\langle I, G\rangle$ then $\mathsf{Abs}(i, \Pi)$ will be a solution to $\langle I, \mathsf{Abs}(i, G)\rangle$, i.e., an i-level abstract solution. However, if $\mathsf{Abs}(i, \Pi)$ is an abstract solution, Π will probably not be a solution as some of its operators might have unsatisfied lower level preconditions.

Semantically, this style of abstraction has an easy description. *The abstraction of a state description will* correspond to a larger set of possible worlds, i.e., $\Sigma_S \subseteq S_{\mathsf{Abs}(i,S)}$, $i > 0$. The semantic function corresponding to the abstract operator has the same definition as the concrete level function, it simply has a (potentially) larger domain. In particular it now includes all $\Sigma \in \mathcal{DW}$ such that $\Sigma \subseteq \Sigma_{\mathsf{Abs}(i,\mathsf{Pre}(\alpha))}$. The semantic function denoting $\mathsf{Abs}(i, \alpha)$ is $\mathsf{Abs}(i, \hat{\alpha})$. Similarly, if S is the set of literals that defines $\Sigma \in \mathcal{DW}$, then $\mathsf{Abs}(i, \Sigma)$ is simply $\Sigma_{\mathsf{Abs}(i,S)}$.

Since abstraction preserves the property that the state descriptions are collections of literals, we continue to have a one-to-one correspondence between abstract solutions (plans) and sequences of semantic function applications, where the semantic functions now correspond to abstract operators,

To simplify our subsequent discussion, we further restrict our attention to a special case where criticalities are assigned to predicates only. That is, an atomic formula and its negation always have the same criticality value.

4 The Downward Refinement Property

Let us fix an arbitrary planning problem (I,G). AH of our subsequent discussion will be about plans, at various levels of abstraction, that are intended as solutions to this fixed problem. For simplicity, we will augment every such plan with special initial and terminal operators. The initial operator ao, has no preconditions and has / as its add list. The terminal operator a_{n+1} has G as its preconditions and an empty add list. Both have empty delete lists. Semantically, $\hat{\alpha}_0$ is a function whose domain is all of $\mathcal{DW}$ and whose range is Σ_I; $\hat{\alpha}_{n+1}$ is an identity function whose domain is $\{\Sigma | \Sigma \subseteq \Sigma_G\}$. Hence, Π will have the form $\overset{\alpha_0}{\mapsto} S_0 \ldots S_n \overset{\alpha_{n+1}}{\mapsto} S_{n+1}$, where $S_0 = I$ and $S_n = S_{n+1}$. The advantage of this convention is that all of the states in the sequence of states defined by the non-augmented Π are sandwiched by a pair of operators in the augmented Π. Note also that α_0 is unaffected by abstraction, as it has no preconditions, but α_{n+1} is affected, as it has the goal as its preconditions and the goal is altered by abstraction.

We can now give a semantic definition of the DRP. Let $\hat{\Pi}$ be a semantic solution to a planning problem $\langle\Sigma_I, \Sigma_G\rangle$. Using the convention of augmented plans, this solution is a sequence of function applications with a corresponding sequence of semantic states: $\overset{\hat{\alpha}_0}{\mapsto} \Sigma_{S_0} \overset{\hat{\alpha}_1}{\mapsto} \ldots \overset{\hat{\alpha}_n}{\mapsto} \Sigma_{S_n} \overset{\hat{\alpha}_{n+1}}{\mapsto}$, where $\Sigma_{S_0} = \Sigma_I$, $\Sigma_{S_n} \subseteq \Sigma_G$, and each $\Sigma_{S_i} \in \mathrm{Domain}(\hat{\alpha}_{i+1})$.

Forward Justification A plan may contain extraneous operators. Justification is a notion that characterizes some of the redundant operators in II; i.e., operators that can be removed without affecting its correctness. In particular, in the version of justification we define, an operator is called justified in II if it is the last operator in the sequence to guarantee that a particular literal is satisfied before it is required as a precondition for a subsequent operator or in the goal state.

Justification plays an important role in determining what is a "good" refinement of an abstract plan.

Definition 1 (Forward Justified P(ans) Let L be a literal. We say that n is *correct with respect to L,* if and only if, whenever $\mathrm{Domain}(\hat{\alpha}_i) \subseteq \Sigma_L$, then $\Sigma_{i-1} \subseteq \Sigma_L$. That is, correctness with respect to *L* ensures that whenever *L* is required as a precondition, it is supplied.

The first operator in the plan $\hat{\alpha}_1$ is forward justified with respect to *L* if and only if upon removing it from II, II is no longer correct with respect to *L.*

Inductively, let Q_i be the set of operators in II that precede $\hat{\alpha}_i$ and that are *not* forward justified with respect to *L.* Then $\hat{\alpha}_i$ is forward justified with respect to *L* if and only if upon removing $\hat{\alpha}_i$ and all of the operators in Q_i from n, II is no longer correct with respect to *L.*

Now, we can define II to be *forward justified* if and only if for every operator $\hat{\alpha}_i$ in II there is some literal *L* such that a_i is forward justified with respect to L. ■

This definition of forward justification is equivalent to a syntactic definition given in [11]. It is not, perhaps, the most natural definition of non-redundant plans from a semantic point of view, and we have examined a number of alternatives. However, for the purposes of this particular paper our definition has the advantage that it corresponds with concepts previously defined in [11].

If a plan II is not forward justified, then it can be converted to a forward justified plan by simply removing the non-justified operators. Furthermore, it is not difficult to demonstrate that if Ft is a correct plan (with respect to our fixed planning problem) then its forward justified version will also be correct.

Mono tonic Refinement The idea behind monotonic refinement is that it defines a "good" refinement of an abstract plan. It captures the intuition that when an abstract plan at level t is refined to level i-1 we want to preserve as much of the work done at the abstract level as is possible. That is, we would not be gaining much advantage from abstraction if we continually replanned achievements of the higher level at the lower level. One way such an undesirable effect could occur is to refine the abstract plan by reversing all of the high-level operators and then plan at the lower level from scratch. Restricting the legal refinements to be monotonic guarantees that such wasteful behavior will not occur.

Definition 2 (Monotonic Refinement) Let II_1 and II_2 be forward justified abstract solutions, with II_1 being an i-th level solution and II_2 an i—1-th level solution, $0 < i < k$.

We say that II_2 is a *monotonic refinement* of II_1, if when we forward justify Abs(i,II_2) we obtain II_1. ■

Note that a monotonic refinement must be a correct plan, i.e., a solution. When we refine an abstract plan we push the operators in that plan to a lower level of abstraction, generating additional preconditions for those operators. Monotonic refinement essentially means that when we add new operators to refine the plan, i.e., to make it correct again, they are added solely to achieve these new preconditions or lower level goal conditions, not to achieve previously satisfied conditions. Hence, when we drop the new conditions again, by abstraction, all of the operators we added during refinement will become redundant. Thus, rejustifying the plan will yield the original abstract plan.

Our definition of monotonic refinement is the semantic version of a syntactic definition given in [11]. Hence, by results from that work it can be shown that in every ABSTRIPS-type abstraction hierarchy, completeness is preserved even when search is restricted by considering only monotonic refinements of abstract solutions [4].

The Downward Refinement Property

Definition 3 (DRP) An abstraction hierarchy has the *downward refinement property* if and only if for every forward justified i-th level abstract solution ft, there is a monotonic refinement of II at level i-1, for $0 < i < k$. *m*

In other words, a hierarchy has the DRP if every solution at an abstract level can be monotonically refined to a solution at the next lower level of abstraction.

Example 1 Consider the Towers of Hanoi domain with 3 pegs, *Peg_1*, Peg_2, *Peg3* and 2 disks *Big, Small.* Using predicates *OnBig(x)* and OnSmall(x), where *x* is a peg, we can represent the location of the two disks. The initial state is *{OnBig(Peg_1) OnSmaU(Pegi)}* and the goal state *{OnBig(Peg_3)*, *OnSmall(Peg_3)*}. We can define two operators *MoveBig{x,y)* which moves the big disk from peg x to y, and MoveSmall(x, y) which similarly moves the small disk. We have that Pre(MoveSmali(x,y)) is {JsPeg(x), Is*Peg(y), OnSmall(x)}* Add(MoveSmall(x,y)) is {*OnSmall(y)*}; and Del(MoveSmall(x,y)) is {*OnSmall(x)}.* For the other operator we have *Pre(MoveBig(x,y))* is {JsPeg(x), *IsPeg{y), ^OnSm&ll(y), ^OnSmail(x), OnBig{x)}* Add(MoveBi£(x,y)) is *{OuBig{y)};* and Del(MoveBi£(x,y)) is *{OnBig(x)}.*

Predicates that are not modifiable by any action, like Ispeg, act as constraints at all levels of abstraction. For example, the *Ispeg* preconditions insure that the parameters can only refer to pegs regardless of the level of abstraction. Hence, we always place these 'Hype" predicates at the highest level of abstraction, i.e., level *k.*[6]

There are two ways to construct an abstraction hierarchy. We can put *OnBig* on the abstract level and *OnSmall* on the concrete level, or we can assign the criticalities the other way around. As we will demonstrate later, the first choice produces a hierarchy that has the DRP, while the second choice produces one that does not.

5 Effects on Search Efficiency

The main reason for being concerned with the DRP is its profound effect on search efficiency.

In planning, it is reasonable to require that completeness is preserved when an one-level planning system is extended to a multiple-level system. Hence, if a non-abstract solution to a problem exists, the abstract planner should also be able to find a concrete solution.

To preserve the completeness of abstract planning, a search method that is complete *across* abstraction levels has to be adopted [4]. Typically, this means that one cannot abandon alternate abstract paths in the search tree when a solution is found at an abstract level. Instead, all paths have to be kept in the open list to be explored later. This can greatly hamper the efficiency of search.

On the other hand, one can do considerably better in hierarchies that satisfy the downward refinement property. At any level of abstraction, search for alternate abstract plans can be terminated once a single correct plan has been found. In other words, search never backtracks to a previously explored abstract level. The DRP guarantees that any abstract solution can be refined to a concrete level one. Thus, if there is a concrete level solution, search in this manner will find it without having to considering alternate abstract plans.

Let *b* be the average branching factor of the search, and *d* be the length of an optimal solution. Then breadth-first search without abstraction will take on average *$0(b^d)$*. On the other hand, say we have *k* levels of abstraction and assume that the branching factor is constant and equal to b in each abstract level. Further assume that at each level of refinement approximately *k/d* new operators are inserted.[7] Then, if the DRP is not satisfied and we assume that there is only a 50% chance that an abstract solution can be refined to a solution at the next lowest level, it can be shown that the average case complexity remains *$0(b^d)$.* With the DRP,

[6]In actuality, since they are not changed by any action, type predicates do not play a role in determining the DSP.

7These assumptions are similar to those made in [12].

however, it can be shown that the average case complexity becomes $O(k \times b^{(d/k)})$. That is, the search is exponentially, in k, more efficient.

The downward refinement property has been implemented as an option in the ABTWEAK planner [4], And experiments done in Towers of Hanoi domain with three disks have shown that there can be as much as a 10 fold improvement in search time and space when using the DRP. This often makes the difference between termination and non-termination.

6 Semantic Conditions for the DRP

Given an abstraction hierarchy, one would like to tell whether it has the DRP. Alternatively, from a domain specification one may want to generate a hierarchy with the DRP guaranteed. We will now provide a set of conditions sufficient to guarantee that the DRP holds. This is done by first providing a semantic condition, and then, in the next section, we will consider various syntactic realizations of this condition.

First we provide a sufficient condition for monotonic refinement.

Lemma 1 **Let $\hat{\Pi}_1$ and $\hat{\Pi}_2$ be forward justified abstract solutions, with $\hat{\Pi}_1$ being an i-th level solution and $\hat{\Pi}_2$ an $i-1$-th level solution. $\hat{\Pi}_2$ is a monotonic refinement of $\hat{\Pi}_1$ if the following two conditions hold.**

1. **$\hat{\Pi}_1$ is a subsequence of $\mathsf{Abs}(i, \hat{\Pi}_2)$.**
2. **Let $\hat{\alpha}_1$ and $\hat{\alpha}_2$ be any pair of adjacent operators in $\hat{\Pi}_1$, and let Σ be the semantic state between these two operators in $\hat{\Pi}_1$. Now by condition (1) there are two corresponding operators $\hat{\alpha}_1^-$ and $\hat{\alpha}_2^-$ in $\hat{\Pi}_2$ that are the $i-1$ level refined versions of $\hat{\alpha}_1$ and $\hat{\alpha}_2$. Let Σ' be any semantic state in $\hat{\Pi}_2$ lying between $\hat{\alpha}_1^-$ and $\hat{\alpha}_2^-$. The second condition is then stated as the requirement that $\Sigma' \subseteq \mathsf{Abs}(i, \Sigma)$. ■**

Now we can give a condition for the downward refinement property.

Theorem 1 **Let $\hat{\Pi}_1$ be any forward justified i-th level abstract solution to any planning problem that has a concrete solution. Let $\hat{\alpha}_1$ and $\hat{\alpha}_2$ be any pair of adjacent operators in $\hat{\Pi}_1$, and let Σ be the semantic state between these two operators in $\hat{\Pi}_1$. Let $\hat{\alpha}_1^-$ and $\hat{\alpha}_2^-$ denote the $i-1$ level refined versions of $\hat{\alpha}_1$ and $\hat{\alpha}_2$. Suppose that for every semantic state Σ_1 such that $\Sigma_1 \in \mathsf{Range}(\hat{\alpha}_1^-)$ and $\Sigma_1 \subseteq \mathsf{Abs}(i, \Sigma)$, there exists a state Σ_2, such that**

1. **$\Sigma_2 \subseteq \mathsf{Abs}(i, \Sigma)$, and $\Sigma_2 \in \mathsf{Domain}(\hat{\alpha}_2^-)$, and**
2. **a solution $\hat{\Pi}_2$ consisting of $i-1$ level operators exists for the problem $\langle \Sigma_1, \Sigma_2 \rangle$, such that for every state Σ' in $\hat{\Pi}_2$, $\Sigma' \subseteq \mathsf{Abs}(i, \Sigma)$.**

Then the DRP is satisfied by the hierarchy, ■

Intuitively, when we refine an abstract solution it may no longer be a solution because of newly introduced and as yet unsatisfied low level preconditions. Hence, looking at the situation between two adjacent operators in the abstract solution we see that the semantic state that is the result of the first operator may no longer be in the domain of the second operator, because of unsatisfied low level preconditions. Our DRP condition states that the hierarchy has the property that a sequence of new operators can be found that will reconnect the two operators. Furthermore, this sequence has the property that it does not affect any higher level conditions; i.e., at the t-th level nothing is changed.

7 Syntactic Checks for The DRP

In this section, we present some sufficient syntactic conditions for the DRP. These conditions are presented in order of increasing sophistication, and range of applicability. Each of these conditions is a syntactic realization of Theorem 1.

Complete Independence We start by considering the most trivial case for the DRP. Suppose that the k+1 criticality levels decompose the operators into k+1 disjoint subsets, $O_o, \ldots, O_k$, so that all of the literals in the precondition, add and delete lists of each operator in O_i have the same criticality value, i. This condition, which we call *complete independence* insures that no operator used to achieve a condition in an i-th level solution will have any effect on any other levels, and furthermore it will not require any preconditions from other levels either.

It is not hard to see that complete independence ensures the DRP. On the semantic level, the k+l abstraction levels define k+1 independent subproblems. Any concatenation of the solution paths at two adjacent levels of abstraction returns a monotonic refinement of the abstract solution. Thus, we have

Theorem 2 Complete independence is sufficient to guarantee the DRP.

Example 2 Consider again the Tower of Hanoi domain. Assume that there are two copies of the 2-disk version of the problem. Given the restriction that no disk for one problem is allowed to be moved to a peg for the other problem, a complete-independence hierarchy can be trivially built by putting one copy of the problem on the abstract level, and the other one on the concrete level.

Complete low-level connectivity The second condition we can place on the preconditions is more useful than complete independence.

Definition 4 (Complete Low-Level Connectivity) Let S_1 and S_2 be any states such that $\mathrm{Abs}(i, S_1) = \mathrm{Abs}(i,S_2)$. That is, S_1 and S_2 only differ with respect to literals at criticality levels less than i. Complete low-level connectivity is satisfied if for any two such states at any level of abstraction i, there is an i-1 level solution, II to the planning problem (S_1,S_2) such that no operators in II add or delete a literal with criticality higher than level *i-l*

Intuitively, this condition states that we can always achieve lower level preconditions without violating the higher level effects. According to the condition, every pair of states S_1 and S_2 are connected by a path, provided that $\mathrm{Abs}(i,S_1) = \mathrm{Abs}(i,S_2)$. Furthermore, the condition requires that no operators on this path affect a higher level literal. This is equivalent to saying that the path never travels out of a state defined by higher level literals. Thus, the conditions of Theorem 1 are satisfied. Hence, we have the following theorem:

Theorem 3 Complete low-level connectivity is sufficient to guarantee the DRP.

Example 3 Consider again the 2-disk Tower of Hanoi domain. Construct an abstraction hierarchy by placing the predicate *OnBig* on the abstract level, and *OnSmall* on the concrete level. If we fix the position of the big disk we can freely move the small disk from any peg to another. The hierarchy satisfies the complete low-level connectivity condition. Now consider the opposite hierarchy where the predicate *OnSmall* is placed on the abstract level, while *OnBig* is on the concrete level. This hierarchy does not satisfy the complete low-level connectivity restriction, since whether or not the big disk can be moved to another peg depends on where the small disk is located. In this case it is also possible to show that the hierarchy does not have the DRP. For example, there is no monotonic refinement of the abstract plan *MoveSmall* (Peg_1, Peg_3).

Unfortunately the complete connectivity condition is difficult to check syntactically. However, special cases can be constructed which can be checked. We give one example.

Definition 5 (Checkable Low-Level Connectivity) Complete low-level connectivity holds if for all levels *i* we have the following:

> For any pair of literals L_1 and L_2, such that $crit(L_1) = crit(L_2) = i-1$, there exists α with $\mathsf{Pre}(\alpha) = \{L_1\}$, $\mathsf{Abs}(i-1, \mathsf{Add}(\alpha)) = \{L_2\}$, and $\mathsf{Abs}(i-1, \mathsf{Del}(\alpha)) = \emptyset$.

Necessary Connectivity The third condition we can place on the preconditions is the most genera). The previous connectivity condition requires that all low level states be connected. This, however, is too restrictive. All that is actually required is low level connectivity between pairs of states that could be generated during the refinement of a higher level plan. We call this condition *necessary connectivity.*

In particular we have:

Definition 6 (Necessary Connectivity) Let $O_{\geq i}$ be the set of operators whose add and delete lists contain at least one literal with criticality value greater than or equal to i. The condition of necessary connectivity is satisfied if for every level of abstraction i we have the following.

Let α_1 and α_2 be operators from $O_{\geq i}$ such that

1. $\mathsf{Abs}(i, \alpha_1(\mathsf{Pre}(\alpha_1))) \cup \mathsf{Abs}(i, \mathsf{Pre}(\alpha_2))$ does not contain both a literal and its negation, where $\alpha_1(\mathsf{Pre}(\alpha_1))$ is the application of α_1 to its own preconditions, and
2. $\mathsf{Abs}(i-1, \mathsf{Pre}(\alpha_2)) \neq \emptyset$.

By these restrictions α_1 and α_2 are operators that could appear in sequence in an i-th level plan, i.e., the weakest postcondition state of α_1 does not contradict the preconditions of α_2 at level t; and α_2 depends on preconditions at level i-1 so that there is the possibility of a problem when refining to this level. For every such pair α_1 and α_2 there must exist an operator β such that:

1. β does not add or delete a literal with criticality higher than $i-1$,
2. $\mathsf{Abs}(i-1, \mathsf{Pre}(\beta)) \subseteq \alpha_1(\mathsf{Pre}(\alpha_1))$, i.e., the preconditions of β at abstraction level $i-1$ are satisfied in weakest the postcondition state of α_1, and
3. $\mathsf{Abs}(i-1, \mathsf{Pre}(\alpha_2)) \subseteq \beta(\mathsf{Pre}(\beta))$; i.e., operator β achieves the $i-1$ preconditions of α_2.

Intuitively, this condition is saying the following: for every pair of operators that might be sequenced in a plan at level i, there exists an operator β whose preconditions at abstraction level i—1 are satisfied in any state that results from applying α_1, and that be used to achieve the i—1-th level preconditions of α_2. Moreover, the operator β does not change any literal whose criticality value is greater than i - 1 . Therefore, on the semantic level, the conditions of Theorem 1 are satisfied, Thus, we have the following:

Theorem 4 Necessary connectivity are sufficient to guarantee the DRP.

Example 4 Consider the following extended Tower of Hanoi domain with two disks, where there is an additional peg, Peg_4 that is completely disconnected from the other three pegs. That is, although initially a disk can be placed on Peg4, no disk can be moved to or away from it.

If we place *OnBig* at the abstract level and *OnSmall* at the concrete level, then the extended Tower of Hanoi domain satisfies the necessary connectivity condition. However, since *Peg4* is disconnected from the rest of the pegs, the concrete-level states are not completely connected- Therefore, the hierarchy does not satisfy the complete low-level connectivity condition.

Example 5 Consider the robot planning domain described by Sacerdoti [l]. A robot can travel between several rooms, where each pair of rooms is connected by one or more doors. A door can be either opened or closed by the robot. In addition, a number of boxes also exist, which can be either pushed around by the robot. To be more complex, we can also allow the robot to carry a box from one location to another. To carry a box, the robot has to first pick it up.

The domain is described by a number of predicates (see *[\]* for a complete description). An abstraction hierarchy can be built by assigning criticality values to the predicates in the following way. The highest level of abstraction consists of all type-preconditions, such as *Connects,* which asserts that a door connects two adjacent rooms. The next level down are predicates *Inroom, At* and *Nextto.* Further down are predicates that describe the status of the door, *Status(dx,Open)* or *St&tus(dx,Closed).* The lowest level are the predicates *Holding* and *Handempty.*

Suppose that all the goals are described by predicates *Inroom* and/or *Nextto.* Then the above hierarchy satisfies the DRR In particular, the hierarchy satisfies the complete low-level connectivity condition. For example, the robot can either close or open a door once it is next to the door, without changing any location predicates. Similarly, the robot can pick up or put down a box without changing any door status or location predicates.

Example 6 Consider a problem of inter-city traveling by an agent. Assume that the agent can travel between cities by bus, train, or plane. Also assume that within each city, the agent can travel between a number of key locations by means of public transportation- Finally, the agent can travel to particular addresses in the local vicinity of each key location by walking. An abstraction hierarchy for this problem domain can be built by placing city-level locations at the highest level, the key locations at the next level, and the addresses at the concrete level. This hierarchy satisfies the DRP: once the agent is in a particular city he will never have to leave the city to travel between key locations in the city, and once he is at the nearest key location he never travel to another key location to travel between local addresses.

8 Conclusions and Future Work

In this paper, we have formalized the downward refinement property for ABSTRIPS-type of abstraction hierarchies. The DRP guarantees completeness of planning even when search refrains from backtrack across abstraction levels. Our analysis shows that an exponential amount of savings in search can be achieved. We have presented a general semantic condition for guaranteeing the DRP, and have derived a number of syntactic conditions from it. Although we lack the space it can be demonstrated that these conditions can be checked in polynomial time. Another topic we have not addressed is the use of our syntactic checks in the automatic generation of abstraction hierarchies that are guaranteed to have the DRR It is possible to specify an algorithm that accomplishes this task.

This work is a step towards understanding abstract problem-solving in general. A number of extensions will be made in future work. One such extension is to design a richer set of conditions that allows for a broader range of application. Another is to explore the use of statistical information in the verification and generation of abstraction hierarchies. As stated above, all of our conditions are sufficient for guaranteeing the DRP. In practice, it may be the case that a large portion of a domain satisfies the property, but not all of it. This might cause the statistical phenomena that most, but not all, abstract solutions can be monotonically refined. We intend to explore the ramifications of this type of behavior. For example, if we have additional statistical information, then we may be able to make inferences as to how close the hierarchy is to having the DRP and the effect this would have on search efficiency.

Acknowledgements Thanks to Craig Knoblock and Josh Tenenberg for helpful comments.

References

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

[2] 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.

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

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

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

[6] Hector J. Levesque. All I Know: A study in autoepistemic logic. Technical Report KRR-TR 89-3, University of Toronto, Toronto, Ont., Canada M5S 1A4, 1989.

[7] Joseph Y. Halpern and Yorham Moses. A guide to the modal logics of knowledge and belief. In *Proc. International Joint Conference on Artifical Intelligence (IJCAI),* pages 480-490, 1985.

[8] Vladimir Lifschitz. On the semantics of strips. In *Proceedings of the Workshop on Reasoning about Actions and Plans,* Timberline, Oregon, 1986.

[9] Jens Christensen. A hierarchical planner that generates its own abstraction hieararchies. In Proceedings *of the 8th AAAI,* pages 1004-1009, Boston, MA., 1990.

[10] Craig A. Knoblock. Learning abstraction hierarchies for problem solving. In *Proceedings of Eighth National Conference on Artificial Intelligence,* Boston, MA, 1990.

[11] Craig Knoblock, Josh Tenenberg, and Qiang Yang. Characterizing abstraction hierarchies for planning. In *Proceedings of Ninth National Conference on Artificial Intelligence,* Anaheim, CA., 1991.

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