Appears in the 2nd International Conference on AI Planning Systems (AIPS), pp. 188-193, 1994

Decomposition and Causality in Partial-Order Planning

R. Michael Young[†] and **Martha E. Pollack**[*,†] and **Johanna D. Moore**[*,†,§]
[†]Intelligent Systems Program
[*]Department of Computer Science
[§]Learning Research and Development Center
University of Pittsburgh, Pittsburgh, PA 15260
myoung+@pitt.edu, pollack@cs.pitt.edu, jmoore@cs.pitt.edu

Abstract

We describe DPOCL, a partial-order causal link planner that includes action decomposition. DPOCL builds directly on the SNLP algorithm (McAllester & Rosenblitt 1991), and hence is clear and simple, and can readily be integrated with other SNLP extensions. In addition, DPOCL is specifically designed to handle partially specified action decompositions. Plan generation in DPOCL exploits the planner's ability to fill in the missing pieces of a partially specified subplan in a way that uses the existing context of the larger plan being constructed.

Introduction

Research in AI plan generation was heavily influenced by the development of simple algorithms for partial-order causal link (POCL) planning, notably TWEAK (Chapman 1987) and SNLP (McAllester & Rosenblitt 1991).[1] These algorithms, and the systems based on them (notably UCPOP (Penberthy & Weld 1992)), have been widely accepted as capturing the key insights of a host of earlier planners, in a framework that is more amenable to rigorous analysis. However, one aspect of previous work on planning that was not adequately captured in these algorithms is *action decomposition*. In planners that support action decomposition, such as NOAH (Sacerdoti 1977), SIPE (Wilkins 1988), and Nonlin/O-Plan (Tate 1977; Currie & Tate 1991), one specifies how high-level, abstract actions can be decomposed into more primitive actions. The process of generating a plan then involves not only establishing causal connections between actions at the same level of abstraction, but also establishing decompositions of the high-level actions in the plan into more primitive ones. Planning with action decomposition is one species of hierarchical planning; the main alternative is precondition hiding (Sacerdoti 1974; Yang & Tenenberg 1990; Knoblock 1994).

Hierarchical planning has several advantages. First, it can potentially lead to a significant reduction in the amount of search needed (Yang 1990; Knoblock 1994; Barrett & Weld 1994). Second, it can make the task of encoding domain knowledge much easier, because the operator writer can re-use operators describing subactions that are common to many actions (Hobbs 1985; Wilkins 1988). Third, hierarchical planning facilitates the interleaving of planning and execution, by making it possible to fully expand only some portions of a plan—including those that need to be executed immediately, while deferring the expansion of other portions (Bratman, Israel, & Pollack 1988). Although the first of these three advantages may accrue to either form of hierarchical planning, the latter two are most fully realized when the levels of the planning hierarchy are specified via action decomposition.

Previous efforts to formalize action decomposition have used Nonlin-based algorithms (Yang 1990; Kambhampati & Hendler 1992). More recently, Barrett and Weld (1994) add action decomposition to the UCPOP algorithm, itself an extension of SNLP. They do this with a bottom-up strategy, generating partial plans at the primitive level, and then using the action hierarchy to filter out completions that cannot be "parsed" into higher-level plans. They argue that this approach allows for the inclusion of actions with universally quantified effects, and also present preliminary experimental results that suggest that it may significantly reduce the search-space size. However, by abandoning top-down decomposition, they give up the third, key advantage of hierarchical planning: their approach cannot support interleaved planning and execution, in which the decomposition of the some higher-level actions is deferred.

In this paper, we show how to incorporate action decomposition directly into the SNLP algorithm. The resulting algorithm, which we call DPOCL (Decompositional Partial-Order Causal-Link planner) is clear and simple, and can readily be integrated with other

[1]Young and Moore receive support from the Office of Naval Research (N00014-91-J-1694). Pollack receives support from the Air Force Office of Scientific Research (F49620-92-J-0422), from the Rome Laboratory of the Air Force Material Command and the Advanced Research Projects Agency (F30602-93-C-0038), and from an NSF Young Investigator's Award (IRI-9258392).

SNLP extensions. In addition, DPOCL is specifically designed to handle partially specified action decompositions. Plan generation in DPOCL exploits the planner's ability to fill in the missing pieces of a partially specified subplan in a way that uses the existing context of the larger plan being constructed.

In the next section, we discuss some issues that must be addressed in an account of action decomposition. The third section provides some preliminary definitions that we use in the DPOCL algorithm, which is stated in the fourth section. The fifth section briefly sketches the formal properties of DPOCL and the final section summarizes the work.

Action Decomposition

Plan generation involves (at least) two different kinds of reasoning. First, it sometimes involves deciding what actions to use to achieve certain effects (or goals). For example, if you want to achieve the goal of being at the airport, you might decide to perform the action of taking a taxi there. Second, it sometimes involves deciding what action(s) to perform as a way of performing some higher-level action.[2] For example, if you want to perform the action of taking a taxi to the airport, you might first call the taxi company to reserve a taxi, then open your front door so you'll hear it pull up, and so on. Nonhierarchical planners, like SNLP, perform only the first kind of reasoning; others, like Nonlin, perform largely the second kind. Nonlin begins with a single, very high-level description of a task to be performed, and iteratively expands—or decomposes—that task. At each iteration, harmful interactions may be introduced; these must be resolved before the next level of expansion.

One important type of interaction involves actions in one subplan whose effects are achieved by actions in other subplans. Suppose that you have two top-level goals: visiting a friend in Santa Fe, and bringing him a gift. You may decompose the action of visiting the friend into a set of actions including taking a taxi to the airport, flying to Santa Fe, and taking a taxi to your friend's house. There are alternative possible decompositions for getting a gift: one involves making a gift, and another involves buying a gift. Suppose you settle on the second, i.e., you decompose the top-level task of getting a gift into subtasks of going to a store and buying a gift. During the process of expansion, you may observe that the step of going to a store exists to achieve the condition of being at a store, which is a precondition of buying the gift—and that the condition of being at a store was achieved by another step in your plan, namely, the step of taking a taxi to the airport. (At least this is true if you live in Pittsburgh, which has a number of good stores at its airport.) What you might well want to do in these circumstances is to use the single act of going to the airport to establish both the precondition of boarding the plane and the precondition of buying the gift.[3]

The idea behind this example is not new. Nonlin, for instance, includes a process called goal phantomization, which does basically what we have just described: it allows a step already in a plan to be used to achieve effects elsewhere in the plan. In our view, the idea behind phantomization is key to the appropriate performance of planners that perform action decomposition. To produce efficient plans, these systems must be able to reason about the context of the larger plan in which any particular decomposition is being inserted. If a planner has the capability to do this, then not only can it choose not to include unneeded steps that may be part of some action-decomposition specification, but it can also fill in pieces of subplans that are only partially specified. For instance, we might imagine that the "get gift" operator is written with multiple effects: the agent performing it has a gift and the gift is wrapped. It might also have several alternative decompositions. In one, the agent makes the gift and then wraps it; in another, all that is specified is that the agent goes to a store and buys a gift there. The higher-level action will not be complete until the condition of having the gift is wrapped has been achieved, and consequently, somewhere in the decomposition this condition must be established. However, the operator writer may choose not to specify the means by which this condition is brought about, believing that it should instead be determined by what else is occuring in the larger plan. Nonlin's use of unsupervised conditions is related to the idea of partiality in action decomposition specifications. However, the two notions are not identical, because the achievement of unsupervised conditions is constrained to be outside the scope of the current plan.

Other approaches to formalizing action decomposition have tended to make overly strong assumptions about the completeness of decomposition specifications. For example, Yang requires that decomposition specifications be nearly complete "miniature plan[s] free of any conflicts." (Yang 1990, p. 14). The requirement of near completeness entails, amongst other things, that every effect of the parent action be asserted by some step in the decomposition specification, and that, moreover, there is no possibly subsequent action in the decomposition specification that clobbers that effect. Kambamphati and Hendler have similarly strong requirements (Kambhampati & Hendler 1992). What we show in the rest of this paper is that such near completeness in action-decomposition specification is not necessary. Action decomposition specifications can be suggestions, sometimes partial, about how to per-

[2] Many papers discuss the foundations of this relation between "high-level" and lower-level actions, (Allen 1984; Pollack 1986; Israel, Perry, & Tutiya 1991).

[3] In fact, you may want the fact that you can *overload* your action in this way to influence your choice of decomposition (Pollack 1991).

form a higher-level action. We also show that this can be achieved within the clean framework of the SNLP algorithm.

Definitions

We begin with some definitions that are needed to give the DPOCL algorithm. A planning system forms plans using information about the actions that can be performed. In DPOCL, the representation of each action is separated into two parts: the action schema and a possibly empty set of decomposition schemata.

Definition 1 (Action Schema) *An* action schema *is a tuple* $\prec A, V, P, E, B \succ$ *where* A *is an action type,* V *is the list of free variables,* P *is the set of preconditions for the action,* E *is the set of effects for the action and* B *is the set of binding constraints on the variables in* V*.*

In this paper, we restrict the preconditions and effects of an action to be sets of literals. Binding constraints may include requirements of codesignation or noncodesignation of pairs of variables in V, as is standard in POCL algorithms. During plan generation, DPOCL (again, like all POCL algorithms) manipulates *steps*, which are uniquely named instances of action schemata.

Every action also has a set of decomposition schemata—the "specifications" of alternative decompositions to which we referred above—although this set may sometimes be empty. We assume that there is a distinguished subset of action types that are identified as being *primitive*, i.e. are directly executable by the planning agent. Primitive actions types have empty sets of decomposition schemata. Any action type that is not primitive is *composite*. Steps in a plan are primitive or composite according to whether their action type is.

Decomposition schemata specify how composite actions can be decomposed into more primitive actions. A decomposition is really a partial plan involving more-primitive actions for achieving a composite action. The partiality is critical: because a decomposition schema may only sketch an expansion of a particular composite action, DPOCL can fill in the remaining details of the expansion during planning in a way that exploits the larger context of the plan in which the composite action is being used. In this sense, a decomposition schema represents an abstraction of many decomposition instances in a manner similar to the abstraction of many actions by an action schema.

Each decomposition schema represents a single-layer expansion of a composite step: decomposition schemata resemble the non-hierarchical partial plans of other POCL planners. Of course, any of the actions within the schema may themselves be composite, and thus subject to further decomposition.

Definition 2 (Decomposition Schema) *A* decomposition schema *is a tuple* $\prec A, S, B, O, L \succ$ *where* A *is an action type,* S *is a set of pseudo-steps,* B *is a set of binding constraints on the free variables in* S *and* A*,* O *defines a partial order on the elements of* S*, and* L *is a set of causal links between the members in* S*.*

Because decomposition schemata specify partial single-level plans, they include several elements that are standard in POCL planners. In particular, causal links are tuples relating a step s, one of its effects e, and another step, t that has a precondition e' that can unify with e. The interpretation of the causal link $\prec s, e, e', t \succ$ is that s is intended to achieve the relevant precondition (e') of t.

One difference between a standard POCL plan and a decomposition schema involves the elements of S. In a POCL plan, these are steps, i.e., they denote particular instances of certain action types, each of which is uniquely named. The unique names are needed because a plan may include more than one instance of the same action type; the step names make it possible to distinguish between, and refer to, each instance, for example in binding constraints. A similar requirement exists for decomposition schemata: the same action type may occur more than once in a decomposition schema, and so individual names must be assigned to each occurrence. In addition, the same decomposition schema may be used more than once in a plan. Thus the names given to the steps in a decomposition schema are only temporary; hence, we refer to them as pseudo-steps. During plan generation, each pseudo-step will either be associated with another step already in the plan, and given that step's name, or else it will be instantiated as a new step, and given a unique name.

Constraints on Decomposition Schemata

Even though they can be partial, decomposition schemata cannot be arbitrary plans; rather, there are certain constraints that hold between a composite action type and any valid decomposition schema for it. Most systems that have used action decomposition, e.g., NOAH, SIPE, and NONLIN have not been explicit or uniform about these constraints. As we mentioned above, Yang (1990) presents a set of constraints that he suggests "capture the formal aspects of action or goal expansions in [these systems]" (p. 14). Whether or not he is right about his constraints capturing the practice in earlier systems, we believe that those constraints are too strong to capture a natural interpretation of action decomposition. For example, he requires that every effect of the parent action be asserted by some step in the decomposition schema, and requires that no decomposition schema include potential threats.

Although we want to allow partiality in the decomposition schema, we do not superfluous planning to occur at a higher level of planning. We therefore include a requirement, similar to one of Yang's, that each precondition of the parent action in a decomposition schema should be needed by some step in the subplan,

which itself contributes, through a chain of causal links, to the establishment of one of the parent action's effects. Intuitively, the preconditions of an action at a particular level of abstraction must contribute to the establishment of its effects no matter which way the action is ultimately performed, i.e., regardless of which decomposition is selected during planning.

To represent the use of the preconditions of the parent action in the decomposition, we employ the standard POCL technique of having a null initial step s_i in the decomposition schema, whose effects are those conditions true in the "initial state" of the subplan, that is, precisely the set of preconditions of the parent action. Analogous to our use of a null initial step, we also include a null final step s_f in decomposition schemata, which, as in all POCL formalisms, has as its preconditions the intended effects (goals) of the plan. Specifically, we require that each final step in a subplan include as preconditions the effects of the parent action. In addition, for each of s_i's effects, we require that there be a path of causal links beginning at s_i, passing through other steps in the decomposition, and ending with a precondition of the final step s_f.

Furthermore, we allow the decomposition schema to specify which conditions in a subplan must be established by the same step that establishes those conditions for the parent. Intuitively, a causal link from one of s_i's effects to a step s with precondition c indicates that c is established for s by the same step that established c as a precondition of the subplan's parent. At the time the parent step is expanded there may be many steps already in the plan that assert effects that can unify with c. When the decomposition schema does not include a causal link to c, the planner may create a link to c from any of these steps—or may insert a new step to establish c.

Decomposition schemata may include causal links that specify how some or all of the preconditions of their steps are to be established. When the schema includes a causal link between two steps, then the first step will be used in the plan to establish the indicated precondition of the second step. Sometimes, however, the schema will include preconditions without incoming causal links; in this case the planner is free, during plan expansion, to use *any* step in the plan to establish the condition in question. In addition, the schema may include steps without any outgoing causal links. Such steps are included as suggestions of actions that may be selected, during plan expansion, to achieve some condition of the subplan. If these steps are not selected, they are termed "unused", and must be pruned from the final plan.

We can now state the constraints on decomposition schemata. Each decomposition schema: (1) contains a a dummy initial step s_i whose effects are the preconditions of the parent step; (2) contains a dummy final step s_f whose preconditions are the effects of the parent step; (3) has ordering constraints ensuring that s_i precedes all other steps in the subplan, and that s_f follows all other steps in the subplan; and (4) each effect of s_i, has a path of causal links that terminates in a precondition of s_f.[4]

Planning with Decompositions

The process of decomposition is one of creating a subplan from a valid decomposition schema. Each step named in the decomposition schema is added to the plan, either by choosing an existing step of the same action type as the step named in the schema or by instantiating a new step from the library of action operators. Ordering and binding constraints for each step are added and causal links created between steps where specified by the decomposition.

During plan generation, the planner needs to keep track of the decompositional decisions that it makes. Whereas a causal link is used to record the fact that the purpose of some step s is to establish the preconditions of some other step (or the goal), a *decomposition link* is used to record the fact that the purpose of some step s is to be part of a more-primitive realization of some other step.

Definition 3 (Decomposition Link) *A* decomposition link *is a tuple* $\prec s, s_i, s_f \succ$ *where s is a composite step, s_i is the initial step of some decomposition of s and s_f is the final step of that decomposition.*

We can now define a DPOCL plan:

Definition 4 (Plan) *A plan is a tuple* $\prec \mathcal{S}, \mathcal{B}, \mathcal{O}, \mathcal{L_C}, \mathcal{L_D} \succ$, *where $\mathcal{S}$ is a set of steps, $\mathcal{B}$ is a set of binding constraints on free variables in $\mathcal{S}$, $\mathcal{O}$ is a set of ordering constraints on steps in $\mathcal{S}$, $\mathcal{L_C}$ is a set of causal links between steps in $\mathcal{S}$, and $\mathcal{L_D}$ is a set of decomposition links among steps in $\mathcal{S}$.*

The sets $\mathcal{S}$, $\mathcal{B}$, and L_C are defined in the standard way; L_D has been specified just above. Given the more general representation of plans in DPOCL, we need to extend the standard POCL step-ordering relation so that we can compare steps at different levels of the plan hierarchy. In DPOCL, ordering constraints capture temporal precedence by representing direct ordering constraints as well as ordering arising from one step belonging to the decomposition of another, i.e., ordering constraints inherited from parent steps. Details are found in the longer version of this paper.

Finally, we also need to modify the termination conditions of non-hierarchical POCL planners. DPOCL halts once a complete plan has been constructed, i.e., once the partial plan under construction has all of its outstanding goals established by causal links, there are no threats to any of these causal links, and all the composite steps have been expanded to the level of primitive actions.

[4] There are also certain restrictions on the binding constraints that exists in the parent action and those in the decomposition; these are detailed in the longer version of this paper.

When DPOCL halts, there may be remaining unused steps in the plan. This can occur when a decomposition schema that was selected during plan expansion includes steps that have no outgoing causal links in the schema. Such steps are included in the schema as suggestions for use by the planner, but it is free to ignore them during the remainder of the planning process. Specifically, a step s in a plan is used only if s is the top-level initial or goal state, or there is a causal link from s to some other used step, or s is the initial or final step in a subplan. Unless a step is incorporated into the plan, it plays no subsequent role in the planning process or in determining when the process can terminate successfully. Plan completeness is thus defined relative only to used steps.

Definition 5 (Plan Completeness) *A plan is* complete *if and only if*

1. **All Steps Are Established.** *For every step $s \in \mathcal{S}$, if s is* used*, then for every precondition p of s, there is a causal link $\prec s', q, p, s \succ \in \mathcal{L_C}$.*
2. **All Threats Are Resolved.** *For any used steps s and t and link $\prec s, q, p, t \succ \in \mathcal{L_C}$ there is no used step S_{threat} that possibly comes between s and t and has effect $\neg e$, where e can unify with the most-general unifier of p and q.*
3. **All Composite Steps are Expanded.** *For every step $s \in \mathcal{S}$, if s is used, then either s is associated with a primitive action or there is a decompositional link $\prec s, s_i, s_f \succ$ in $\mathcal{L_D}$.*

The Algorithm

We can now provide the DPOCL algorithm, shown in Figure 1. Standard POCL planners iterate through a loop in which they first check for a completed plan, then perform plan refinement by adding causal links for open conditions, and finally resolve threats to existing causal links created by recent plan modifications. DPOCL differs from this approach by providing an additional option for the plan refinement phase: the planner may either do causal planning or may do decompositional planning. Either of these options is followed by a threat resolution phase. The algorithm terminates when there are no open conditions and when all abstract steps have been decomposed into primitive actions.

In its general form, the decision about whether to perform causal or decompositional planning at each iteration is left open: the two forms of plan refinement can be fully interleaved. Traditional hierarchical planning has first done complete causal planning at a single layer of the hierarchy, and only then done decomposition. Control rules that enforce this ordering could be added to DPOCL. However, we believe that in general it is advantageous to allow for interleaving of causal and decompositional planning, especially in situations in which planning and execution must be interleaved.

$DPOCL(\prec \mathcal{S}, \mathcal{B}, \mathcal{O}, \mathcal{L_C}, \mathcal{L_D} \succ, \Delta, \Lambda)$

On the initial call to DPOCL, there are only two steps in $\mathcal{S}$ —the dummy initial and final steps—and a single ordering constraint between them in $\mathcal{O}$. $\mathcal{B} = \mathcal{L_C} = \mathcal{L_D} = \{\}$. Δ is the set of decomposition schemata and Λ is the set of action schemata.

I. Termination If $\mathcal{O}$ or $\mathcal{B}$ is inconsistent, fail. Otherwise, if $\prec \mathcal{S}, \mathcal{B}, \mathcal{O}, \mathcal{L_C}, \mathcal{L_D} \succ$ is complete, prune unused steps from $\mathcal{S}$ and return the result.

II. Plan Refinement Nondeterministically do one of the following:

- **Causal Planning**
  - **Goal Selection** Select a goal (i.e., a used step S_{need} with precondition p and no causal link $\prec t, q, p, S_{need} \succ \in \mathcal{L_C}$).
  - **Operator Selection** Let S_{add} be a step that adds an effect e which can be unified with p (to create S_{add}, nondeterministically either choose a step S_{old} already in $\mathcal{S}$ or instantiate an operator from Λ). If no such step exists, backtrack. Otherwise, let $\mathcal{S}' = \mathcal{S} \cup \{S_{add}\}$, $\mathcal{O}' = \mathcal{O} \cup \{S_{add} < S_{need}\}$, $\mathcal{B}' = \mathcal{B} \cup$ the set of variable bindings needed to make S_{add} add e, including the bindings of S_{add} itself, $\mathcal{L}'_{\mathcal{C}} = \mathcal{L_C} \cup \prec S_{add}, e, p, S_{need} \succ$, and $\mathcal{L}'_{\mathcal{D}} = \mathcal{L_D}$.
- **Decompositional Planning**
  - **Action Selection** Nondeterministically select a used, unexpanded composite step S_{parent} from $\mathcal{S}$.
  - **Decomposition Selection** Nondeterministically select a decomposition schema $D = \prec T_D, S_D, B_D, O_D, L_{C_D} \succ$ from Δ that has a type T_D matching the action type associated with S_{parent}. Replace each step name occurring in S_D with actual step names, either instantiated from Λ or already existing in $\mathcal{S}$, such that each new step name is of the same action type as the step name it replaces. Replace each reference to the old steps from S_D with the corresponding new step names in B_D, O_D and L_{C_D}. Then let $\mathcal{S}' = \mathcal{S} \cup S_D$, $\mathcal{O}' = \mathcal{O} \cup O_D$, $\mathcal{B}' = \mathcal{B} \cup B_D$, $\mathcal{L}'_{\mathcal{C}} = \mathcal{L_C} \cup L_{C_D}$, and $\mathcal{L}'_{\mathcal{D}} = \mathcal{L_D} \cup \{\prec S_{parent}, S_{parent_i}, S_{parent_f} \succ\}$.

III. Threat Resolution A step S_{threat} threatens a causal link $\prec S_j, e, p, S_k \succ$ when it occurs between S_j and S_k and it asserts $\neg e$. For every used step S_{threat} that might threaten a causal link $\prec S_j, e, p, S_k \succ \in \mathcal{L_C}$, nondeterministically do one of the following: *Promotion* If S_k possibly precedes S_{threat}, let $\mathcal{O}' = \mathcal{O} \cup \{S_k < S_{threat}\}$. *Demotion* If S_{threat} possibly precedes S_j, let $\mathcal{O}' = \mathcal{O} \cup \{S_{threat} < S_j\}$. *Separation* Let $\mathcal{O}' = \mathcal{O} \cup \{S_j < S_{threat}, S_{threat} < S_k\}$ and let $\mathcal{B}' = \mathcal{B} \cup$ the set of variable prohibitions needed to ensure that S_{threat} won't assert $\neg e$.

IV. Recursive Invocation Call
$DPOCL(\prec \mathcal{S}', \mathcal{B}', \mathcal{O}', \mathcal{L}'_{\mathcal{C}}, \mathcal{L}'_{\mathcal{D}} \succ, \Delta, \Lambda)$

Figure 1: The DPOCL Algorithm

Formal Properties

In the longer version of this paper we show that DPOCL both is sound and satisfies a restricted form of completeness. The proof of soundness is similar in structure to that given for the UCPOP algorithm (Penberthy & Weld 1992). Like that proof, we define a loop invariant, which we prove holds at various points in the planning process. This has the soundness of DPOCL as a consequence. The UCPOP loop invariant states that if the current subgoals can be achieved by the current plan, then the plan is a solution, i.e., there exists an executable extension of that plan. In Penberthy and Weld's proof, the invariant is shown to hold for causal-link introduction. Our proof shows that similar invariants hold when either causal or decompositional links are introduced, i.e, performing action decomposition preserves soundness.

In addition, we prove that DPOCL is primitive complete. That is, for every solution S to a planning problem α where S contains only primitive steps, DPOCL is guaranteed to produce a plan whose primitive steps are S.

Summary

Action decomposition is generally accepted as essential to plan generation. Although many previous planning systems have employed action decomposition, the development of algorithms like SNLP make it possible for the first time to give a careful statement of the decomposition process. The algorithm we present in this paper, DPOCL, is a sound and primitive complete algorithm for the creation of plans with decompositional as well as causal structure. By incorporating decompositional planning directly into a POCL framework we have been able to specify a precise relationship between abstract steps and the subplans that achieve them. Furthermore, the constraints we place on the specification of decomposition schemata in DPOCL are less restrictive than previous formal models. By taking advantage of the context of the larger plan under construction during composite step expansion, DPOCL can generate plans that may be more efficient than those produced by planners that require more constrained specifications of decomposition schemata.

References

Allen, J. F. 1984. Towards a general theory of action and time. *Artificial Intelligence* 23(2):123–154.

Barrett, A., and Weld, D. 1994. Schema parsing: Heirarchical planning for expressive languages. Unpublished manuscript.

Bratman, M. E.; Israel, D. J.; and Pollack, M. E. 1988. Plans and resource-bounded practical reasoning. *Computational Intelligence* 4:349–355.

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32(3):333–378.

Currie, K., and Tate, A. 1991. O-plan: The open planning architecture. *Artificial Intelligence* 52:49–86.

Hobbs, J. R. 1985. Granularity. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 432–435.

Israel, D.; Perry, J.; and Tutiya, S. 1991. Actions and movement. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, 1060–1065.

Kambhampati, S., and Hendler, J. 1992. A validation structure based theory of plan modification and reuse. *Artificial Intelligence* 55(2):193–158.

Knoblock, C. A. 1994. Automatically generating abstractions for planning. *Artificial Intelligence*. In press.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 634–639.

Penberthy, J., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, 103–114.

Pollack, M. E. 1986. Inferring domain plans in question-answering. Technical Report 403, Artificial Intelligence Center, SRI International, Menlo Park, CA. Also appears as a University of Pennsylvania PhD thesis.

Pollack, M. E. 1991. Overloading intentions for efficient practical reasoning. *Noûs* 25(4):513–536.

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

Sacerdoti, E. D. 1977. *A Structure for Plans and Behavior*. New York: American Elsevier.

Tate, A. 1977. Generating project networks. In *Proceedings of IJCAI-77*, 888–893.

Wilkins, D. E. 1988. *Practical Planning: Extending the Classical AI Paradigm*. San Mateo, CA: Morgan Kaufmann.

Yang, Q., and Tenenberg, J. D. 1990. ABTWEAK: Abstracting a nonlinear, least commitment planner. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 204–209.

Yang, Q. 1990. Formalizing planning knowledge for a hierarchical planner. *Computational Intelligence* 6:12–24.