Universal Classical Planner: An algorithm for unifying State-space and Plan-space planning

Subbarao Kambhampati* & **Biplav Srivastava**
Department of Computer Science and Engineering
Arizona State University, Tempe AZ 85287-5406
email: {rao,biplav}@asu.edu
WWW: http://rakaposhi.eas.asu.edu:8001/yochan.html

Appears in Current Trends in AI Planning: EWSP '95, IOS Press

Abstract

We present a plan representation and a generalized algorithm template, called UCP, for unifying the classical plan-space and state-space planning approaches within a single framework. UCP models planning as a process of refining a partial plan. The plan-space and state-space planning approaches are cast as complementary refinement strategies operating on the same partial plan representation. UCP has the freedom to arbitrarily and opportunistically interleave plan-space and state-space refinements within a single planning episode. This allows it reap the benefits of both state-space and plan-space planning approaches. We discuss the coverage, completeness and systematicity of UCP. We also present some preliminary empirical results that demonstrate the utility of combining state-space and plan-space approaches.

1 Introduction

Domain independent classical planning techniques fall into two broad categories-- state space planners which search in the space of states, and plan space planners which search in the space of plans. Although most current research in planning is rooted in plan-space planning [1, 15, 10], it is believed that the ''state'' information provided by the state-space planners could be advantageous in some situations either for improving planning performance (c.f. [18]), or for combining general purpose planners with specialists [13]. Unfortunately, however, there has not been a systematic analysis of when state-space planning could be more advantageous over plan-space planning. This is partly due to the fact that plan-space and state-space planners have been seen as ''competing'' (rather than ''complementary'') approaches involving search in fundamentally different search spaces [1, 18]. A more fruitful way of understanding the relative tradeoffs between plan-space and state-space planners would be to develop a unifying planning framework that encompasses them both.

*Authors names are listed in alphabetical order. This research is supported in part by NSF research initiation award (RIA) IRI-9210997, NSF young investigator award (NYI) IRI-9457634 and ARPA/Rome Laboratory planning initiative grant F30602-93-C-0039. We thank Laurie Ihrig and Craig Knoblock for helpful comments.

Previously, we have shown that viewing planning as a process of refinement search provides a powerful framework for unifying the large variety of plan-space planning approaches [10, 8, 9]. In this paper, we show that the same framework can be extended to unify state-space planning and plan-space planning approaches. In particular, we present UCP, a generalized algorithm for classical planning. UCP covers classical planning approaches through three complementary refinement strategies, corresponding respectively to the plan-space, forward state-space and backward state-space planning approaches. If it chooses to always use plan-space refinements, it becomes a pure plan-space planner, and if it chooses to always use state-space refinements, it becomes a pure state-space planner. Since all three refinement strategies operate on the same general partial plan representation, UCP also facilitates opportunistic interleaving of the plan-space and state-space refinements within a single planning episode. Such interleaving could produce a variety of hybrid planners (including the traditional means-ends analysis planners such as STRIPS and PRODIGY [4]), and can thus help us reap the benefits of both state-space and plan-space approaches in a principled manner. The traditional question ''when should a plan-space planner be preferred over state space planners?'' [1, 18] can now be posed in a more sophisticated form: ''*when should a plan-space refinement be preferred over a state-space refinement (or vice versa) within a single planning episode?*.''

The rest of this paper is organized as follows. In Section 2, we review the preliminaries of refinement planning. Section 3 describes the representation and semantics for partial plans used in UCP. In Section 4, we describe the UCP algorithm and illustrate its operation through an example. Section 5 discusses the coverage, completeness and systematicity of the UCP algorithm. Section 6 describes several heuristic control strategies for UCP. Section 7 empirically demonstrates the utility of opportunistic interleaving of refinements in UCP. Section 8 discusses the related work and Section 9 presents the conclusions.

2 Planning as Refinement Search: Overview

A planning problem is a 3-tuple $\langle I, G, \mathcal{A} \rangle$, where I is the description of the initial state, G is the (partial) description of the goal state, and $\mathcal{A}$ is the set of actions (also called ''operators''). An action sequence (also referred to as ground operator sequence) S is said to be a solution for a planning problem, if S can be executed from the initial state of the planning problem, and the resulting state of the world satisfies all the goals of the planning problem.

Refinement planners [10, 8, 9] attempt to solve a planning problem by navigating the space of *sets of potential solutions (ground operator sequences)*. The potential solution sets are represented and manipulated in the form of ''partial plans.''[1] Syntactically, a partial plan $\mathcal{P}$ can be seen as a set of constraints (see below). Semantically, a partial plan is a shorthand notation for the set of ground operator sequences that are consistent with its constraints. The latter set is called the *candidate set* of the partial plan, and denoted by $\langle\langle \mathcal{P} \rangle\rangle$.

Refinement planning consists of starting with a ''null plan'' (denoted by $\mathcal{P}_\emptyset$), whose candidate set corresponds to all possible ground operator sequences, and successively refining the plan (by adding constraints, and thus splitting their candidate sets) until a solution is reached. Semantically, a *refinement operator* $\mathcal{R}$ maps a partial plan $\mathcal{P}$ to a set of partial plans $\{\mathcal{P}'_i\}$ such that the candidate sets of each of the children plans are proper subsets of the candidate set of $\mathcal{P}$ (i.e., $\forall_{\mathcal{P}'_i} \langle\langle \mathcal{P}'_i \rangle\rangle \subset \langle\langle \mathcal{P} \rangle\rangle$). Refinement planning involves repeatedly applying refinement operators to a partial plan until a solution can be picked up from the candidate set

[1] For a more formal development of the refinement search semantics of partial plans, see [8, 10]

of the resulting plan.

A refinement operator $\mathcal{R}$ is said to be **complete** if every solution belonging to the candidate set of $\mathcal{P}$ belongs to the candidate set of at least one of the children plans. $\mathcal{R}$ is said to be **systematic** if the candidate sets of children plans are non-overlapping (i.e. $\forall_{\mathcal{P}'_i,\mathcal{P}'_j,i\neq j}\ \langle\langle \mathcal{P}'_i \rangle\rangle \cap \langle\langle \mathcal{P}'_j \rangle\rangle = \emptyset$). It is easy to see that as long as a planner uses only refinement operators that are complete, it never has to backtrack over the application of a refinement operator. Similarly, if all the refinement operators are systematic, the search space of the planner will be systematic in that no ground operator sequence will belong to the candidate sets of plans in more than one branch of the search tree [12].

In what follows, we will show that state-space planning and plan-space planning approaches can essentially be modeled as two different varieties of refinement operators operating on the same partial plan representation.

3 Representation of partial plans in UCP

In this section, we will develop the syntactic and semantic representation of partial plans that is adequate to support both state-space and plan-space refinements.

A partial plan is a 5-tuple $\langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle$ where: T is the set of steps in the plan; T contains two distinguished step names t_0 and t_∞. $\mathcal{ST}$ is a symbol table, which maps step names to actions. The special step t_0 is always mapped to the dummy operator `start`, and similarly t_∞ is always mapped to `finish`. The effects of `start` and the preconditions of `finish` correspond, respectively, to the initial state and the desired goals of the planning problem. O is a partial ordering relation over T. $\mathcal{B}$ is a set of codesignation (binding) and non-codesignation (prohibited binding) constraints on the variables appearing in the preconditions and post-conditions of the operators. A ground linearization of $\mathcal{P}$ is a permutation on its steps that is consistent with O, with all variables instantiated to values that are consistent with $\mathcal{B}$. $\mathcal{L}$ is a set of auxiliary constraints that restrict the allowable orderings and bindings among the steps. Three important types of auxiliary constraints are:

Interval Preservation Constraints: An *interval preservation constraint* (IPC) is specified as a 3-tuple: $\langle t, p, t' \rangle$. Syntactically, it demands that the condition p be preserved between t and t' in every ground linearization of the plan. Semantically, it constrains the candidates of the partial plan such that in each of them, p is preserved between the operators corresponding to steps t and t'.

Point Truth Constraints: A *point truth constraint* (PTC) is specified as a 2-tuple: $\langle p, t \rangle$. Syntactically, it demands that the condition p be necessarily true [2] in the situation before the step t. Semantically, it constrains all solutions of the partial plan to have p in the state in which the operator corresponding to t is executed.

Contiguity Constraints: A contiguity constraint is specified as a relation between two steps: $t_i * t_j$. Syntactically, it demands that no step intervene between t_i and t_j in any ground linearization of $\mathcal{P}$. Semantically, it constrains all candidates of the partial plan to have no operators intervening between the operators corresponding to t_i and t_j. From the definition, if t_i is immediately before t_j, it also means that $t_i \prec t_j$. Further, since t_i and t_j are contiguous, there cannot be any step t' such that $t_i \prec t' \prec t_j$, or $t_i * t'$ or $t' * t_j$.

Readers who are familiar with the refinement search view of partial order planning developed in [10, 8, 9] may note that *the only extension to the plan representation is the addition of the new type of auxiliary constraints called contiguity constraints.* We will see that this extension is enough to handle state space refinements. The contiguity constraints are first described by Ginsberg in [5].

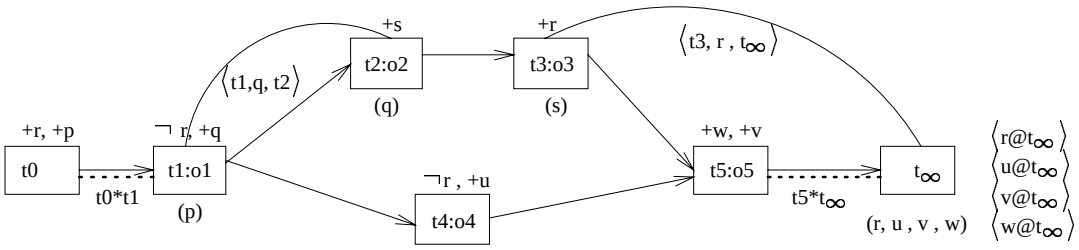


Figure 1: Example Partial Plan $\mathcal{P}_E$. The effects of the steps are shown above the steps, while the preconditions are shown below the steps in parentheses. The ordering constraints between steps are shown by arrows. The interval preservation constraints are shown by arcs, while the contiguity constraints are shown by thick dotted lines. The PTCs are used to specify the goals of the plan

Safe Ground Linearizations: A ground linearization is said to be a **safe ground linearization** if it syntactically satisfies all the contiguity constraints, and the interval preservation constraints [10]. The semantic notion of the candidate set of the partial plan is tightly related to a syntactic notion of safe ground linearization [10, 8]. Specifically, safe ground linearizations correspond to minimal length candidates of the partial plan [10]. If a partial plan has no safe ground linearizations, it has an empty candidate set.

Example: Figure 1 illustrates these definitions through an example plan $\mathcal{P}_E$, which contains the steps $\{t_0, t_1, t_2, t_3, t_4, t_5, t_\infty\}$, the symbol table $\{t_0 \rightarrow \texttt{start}, t_\infty \rightarrow \texttt{end}, t_1 \rightarrow o_1, t_2 \rightarrow o_2, t_3 \rightarrow o_3, t_4 \rightarrow o_4, t_5 \rightarrow o_5\}$, the ordering constraints $\{t_1 \prec t_2, t_2 \prec t_3, t_3 \prec t_5, t_1 \prec t_4, t_1 \prec t_5, t_4 \prec t_5\}$, the contiguity constraints $\{t_0 * t_1, t_5 * t_G\}$, the interval preservation constraints $\{\langle t_1, q, t_2\rangle, \langle t_3, r, t_\infty\rangle\}$, and the point truth constraints $\{\langle r, t_\infty\rangle, \langle u, t_\infty\rangle, \langle v, t_\infty\rangle, \langle w, t_\infty\rangle\}$. The ground operator sequence $o_1o_2o_4o_3o_2o_2o_5$ is a candidate of the plan $\mathcal{P}_E$, while the ground operator sequences $o_3o_1o_2o_3o_3o_5$ and $o_1o_2o_3o_4o_5$ are not candidates of $\mathcal{P}_E$ (the former violates the contiguity constraint $t_0 * t_1$, and the latter violates the interval preservation constraint $\langle t_3, r, t_\infty\rangle$). $t_0t_1t_2t_4t_3t_5t_\infty$ is a safe ground linearization, while $t_0t_1t_2t_3t_4t_5t_\infty$ is not a safe ground linearization (since the interval preservation constraint $\langle t_3, r, t_\infty\rangle$ is not satisfied by the linearization). The safe ground linearization corresponds to the minimal candidate (ground operator sequence) $o_1o_2o_4o_3o_5$.

We will now define some derived attributes of a partial plan, which will be useful in describing the UCP algorithm. Given a plan $\mathcal{P}$, a step $t_\mathcal{H}$ is said to be the **head step** of the plan if there exists a sequence of steps $t_1 \cdots t_n$ such that $t_0 * t_1 * t_2 \cdots t_n * t_\mathcal{H}$ and there is no step t' such that $t_\mathcal{H} * t'$. The sequence of steps $t_0, t_1, \cdots t_n, t_\mathcal{H}$ is called the **header** of the plan. The state resulting from the application of the operators corresponding to the header steps, in sequence, to the initial state, is called the **head state**. The **head fringe** of a plan $\mathcal{P}$ is the set of all steps t that can possibly come immediately after the $t_\mathcal{H}$ in some ground linearization of the plan (i.e., all t such that $t_\mathcal{H} \prec t$ and there is no step t' such that $\Box(t_\mathcal{H} \prec t' \prec t)$).

Similarly, a step $t_\mathcal{T}$ of a plan $\mathcal{P}$ is said to be the **tail step** of the plan if there exists a sequence of steps $t_1 \cdots t_n$ such that $t_\mathcal{T} * t_1 * t_2 \cdots t_n * t_\infty$ and there is no step t' such that $t' * t_\mathcal{T}$. The sequence of steps $t_\mathcal{T}, t_1, \cdots t_n, t_\infty$ is called the **trailer** of the plan. The state resulting from the backward application of the operators corresponding to the trailer steps, in sequence, to the goal state, is called the **tail state**. The **tail fringe** of a plan $\mathcal{P}$ is the set of all steps t that can possibly come immediately before the tail step $t_\mathcal{T}$ in some ground linearization of the plan (i.e., all t such that $t \prec t_\mathcal{T}$ and there is no step t' such that $\Box(t \prec t' \prec t_\mathcal{T})$).

Example: In the example plan $\mathcal{P}_E$ shown in Figure 1, t_1 is the head step and t_5 is the tail

Algorithm UCP($\mathcal{P}$) /*Returns refinements of $\mathcal{P}$ */

Parameters: `sol`: the routine for picking solution candidates from the candidate set of the partial plan
`pick-refinement`: a strategy for picking refinements

0. Termination Check: If `sol`($\mathcal{P}$) returns a ground operator sequence that `solves` the problem, return it and terminate.

1. Refinement: Using `pick-refinement` strategy, do any one of (*not* a backtrack point):

- Refine-plan-forward-state-space($\mathcal{P}$)
- Refine-plan-backward-state-space($\mathcal{P}$)
- Refine-plan-plan-space($\mathcal{P}$) {Corresponds to a class of refinements }

2. Consistency Check: (Optional) If the partial plan is inconsistent (i.e., has no safe ground linearizations), or non-minimal (e.g. has state loops) prune it.

3. Recursive Invocation: Call `UCP` on the the refined partial plan (if it is not pruned).

Figure 2: UCP: A generalized algorithm template for classical planning

Algorithm Refine-plan-Forward-State-space ($\mathcal{P}$) /*Returns refinements of $\mathcal{P}$ */

1.1 Operator Selection: Nondeterministically select one of the following (choice point):

1. Nondeterministically select a step t_{old} from head-fringe of $\mathcal{P}$, such that all preconditions of the operator $\mathcal{ST}(t_{old})$ are satisfied in head-state of $\mathcal{P}$. (If t_{old} is the tail step, then select it only if the tail state is a subset of head-state.) **or**
2. Nondeterministically select an operator o from the operator library, such that all preconditions of the operator o are satisfied in head-state of the plan. Make a step name t_{new}, and add the mapping $[t_{new} \rightarrow o]$ to $\mathcal{ST}$.

1.2 Operator Application: Let t_{sel} be the step selected above. Add the auxiliary constraint $t_{\mathcal{H}} * t_{sel}$. (This implicitly updates the head step to be t_{sel}, and head state to be the result of applying $\mathcal{ST}(t_{sel})$ to head state).

Figure 3: Forward State Space Refinement

step. $t_0 t_1$ is the header and $t_5 t_\infty$ is the trailer. $\{t_2, t_4\}$ is the head fringe, and $\{t_3, t_4\}$ is the tail fringe. Head state is $p \wedge q$ while the tail state is $r \wedge u$.

4 The UCP planning algorithm

The UCP algorithm uses the plan representation developed in the previous section to combine plan-space and state-space approaches under one framework. Figure 2 shows the top level control strategy of UCP. In step 1, UCP chooses among three different refinement strategies. corresponding, respectively, to the forward state-space planning, backward state-space planning and plan-space planning approaches. We shall refer to them as FSS, BSS and PS refinements respectively. The refinements themselves are described in Figures 3, 4 and 5. UCP accepts an arbitrary control strategy `pick-refinement` as a parameter. In each iteration, this control strategy is used to select one of the three refinement strategies (see below). The selected refinement strategy is applied to the partial plan to generate the refinements. Since all three refinements are complete (see Section 5), *UCP never has to backtrack on the choice of the refinement strategy*.

Informally, the FSS refinement involves advancing the header state by applying an operator to it (which involves putting a contiguity constraint between the current head step and the new step corresponding to the operator that we want to apply). A step is considered

Algorithm Refine-plan-backward-state-space ($\mathcal{P}$) /*Returns refinements of $\mathcal{P}$ */

1.1 Operator Selection: Nondeterministically select one of the following (choice point):

1. Nondeterministically select a step t_{old} from tail-fringe of the plan, such that (a) none of the effects of the operator $\mathcal{ST}(t_{old})$ are negating the facts in the tail state and (b) at least one effect of the operator $\mathcal{ST}(t_{old})$ is present in the tail state. (If t_{old} is the head step, then select it only if the tail state is a subset of head-state.) **or**
2. Nondeterministically select an operator o from the operator library, such that (a) none of the effects of the operator o are negating the facts in the tail state and (b) at least one effect of the operator $\mathcal{ST}(t_{old})$ is present in the tail state of $\mathcal{P}$. Make a step name t_{new}, and add the mapping $[t_{new} \rightarrow o]$ to $\mathcal{ST}$.

1.2 Operator Application: Let t_{sel} be the step selected above. Add the auxiliary constraint $t_{sel} * t_{\infty}$. (This implicitly updates the tail step and tail state).

Figure 4: Backward State Space Refinement

Algorithm Refine-plan-plan-space ($\mathcal{P}$) /*Returns refinements of $\mathcal{P}$ */
Parameters: `pick-open`: the routine for picking open conditions.
`pre-order`: the routine which adds orderings to the plan to make conflict resolution tractable.
`conflict-resolve`: the routine which resolves conflicts with auxiliary constraints.

1.1 Goal Selection: Using the `pick-open` function, pick an open prerequisite $\langle C, t \rangle$ (where C is a precondition of step t) from $\mathcal{P}$ to work on. *Not a backtrack point.*

1.2. Goal Establishment: Non-deterministically select a new or existing establisher step t' for $\langle C, t \rangle$. Introduce enough constraints into the plan such that (i) t' will have an effect C, and (ii) C will persist until t. *Backtrack point; all establishers need to be considered.*

1.3. Bookkeeping: (Optional) Add interval preservation constraints noting the establishment decisions, to ensure that these decisions are not violated by latter refinements. This in turn reduces the redundancy in the search space.

2. Tractability Refinements: (Optional) These refinements help in making the plan handling and consistency check tractable. Use either one or both:

2.a. Pre-Ordering: Use some given static ordering mechanism, `pre-order`, to impose additional orderings between steps of the partial plans generated by the establishment refinement. *Backtrack point; all interaction orderings need to be considered*.

2.b. Conflict Resolution: Add orderings and bindings to resolve conflicts between the steps of the plan, and the plan's auxiliary constraints. *Backtrack point; all possible conflict resolution constraints need to considered*.

Figure 5: Plan Space Refinement

to be applicable if all its preconditions are satisfied in the head state of the partial plan. Completeness is ensured by considering both the steps in the head fringe of the plan, and the operators in the operator library. One exception arises in the case when the tail step of the plan is one of the steps on the head fringe. In this case, the tail step is applied to head state only when all the conditions in the tail state (rather than the preconditions of the tail step) are present in the head state. This is because, once the tail step is introduced into the header, no further steps can be added to the partial plan.

BSS refinement (Figure 4) is very similar to the FSS refinement, except that it regresses the tail state by backward-applying new operators to the tail state. An operator is considered to be backward-applicable to a state if it does not delete any conditions in the state, and adds at least one condition in the state (this latter part makes it goal-directed). The state resulting

from the backward application of the operator O contains all the conditions of the previous state, minus the conditions added by O, plus the preconditions of O.

Finally, the PS refinement, given in Figure 5, is more involved than the two state space refinements. As we showed in [8, 9, 10], a general PS refinement involves an establishment refinement phase, and an optional tractability refinement phase. In the establishment phase, a precondition p of a step s is selected, and a sufficient number of step, ordering and binding constraints are added to ensure that p will be necessarily true at s. The choice of which precondition to achieve does not have to be backtracked over, but all possible ways of establishing the selected precondition must be considered. Thus *PS refinement corresponds to a family of complete refinements*, each distinguished by the precondition that is considered for establishment. An optional bookkeeping step can add auxiliary (interval preservation) constraints to *protect* the establishment decisions. In the presence of interval preservation constraints, checking whether the plan is consistent (i.e., contains ground linearizations that are safe with respect to all the interval preservation constraints) is in general intractable. To make this tractable, the optional tractability refinement phase introduces additional orderings between the steps of the plan. Pre-ordering and conflict resolution are two common strategies used in this regard. For a more elaborate discussion of the details of the PS refinement, the reader is urged to consult [10, 9, 8].

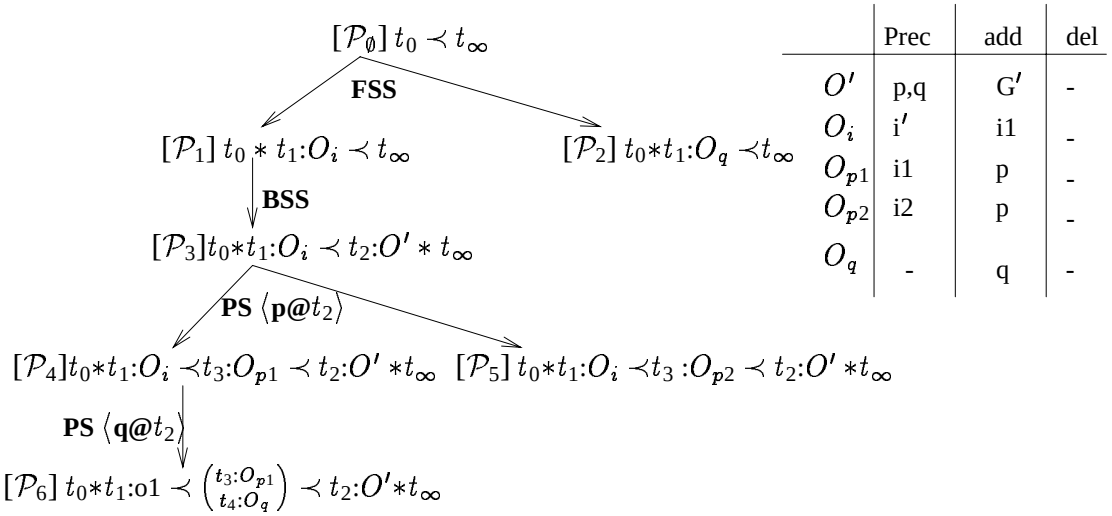
Termination Check: At each refinement cycle, the UCP algorithm checks to see if the search can be terminated successfully. As we mentioned earlier, termination can occur on a partial plan whenever we can pick a solution from the candidate set of the partial plan. Since enumerating the full candidate set is infeasible, most planners restrict their attention to the minimal candidates corresponding to the safe ground linearizations of the partial plan, and check if any of them are solutions for the planning problem. Notice that this termination check is enough to successfully terminate UCP whether it uses only state space refinements, only plan-space refinements, or a combination of both. There may of course be more specialized realizations of the termination check that are more efficient for specific instantiations of UCP. For example, pure plan space planners using causal links can use a causal link based termination check used in SNLP [12, 10]. Similarly, if we are using only the FSS and BSS refinements, then the plan can terminate as soon as the head step is introduced into the trailer, or vice versa.

Consistency Check: At each refinement cycle, UCP uses an optional consistency check to prune out unpromising refinements. One important type of unpromising refinements are those partial plans which have no safe ground linearizations (and thus have empty candidate sets). In addition, in the presence of FSS and BSS refinements, we can also check for and prune plans containing *state looping*. Forward state looping occurs when there are two steps t_1 and t_2 in the header of the plan such that t_1 precedes t_2, and the state after t_1 contains all the conditions that are present in the state after t_2. Similarly, backward sate looping occurs when there are two steps t' and t'' in the trailer of the plan such that t' precedes t'' and the state preceding t' is contains all the conditions that are present in the state preceding t''. In either case, it can be shown that the candidate sets of the corresponding partial plans will not contain any minimal solutions[2], and the partial plans can thus be pruned.

4.1 An example planning trace of UCP

Figure 6 illustrates UCP's planning process on a simple example. The problem has the initial state $\{i'\}$ and the goal state $\{G'\}$. The operators in the domain are described in the table

[2] A solution is minimal, if no ground operator sequence derived by deleting some elements from it is also a solution.



	Prec	add	del
O'	p,q	G′	-
O_i	i′	i1	-
O_{p1}	i1	p	-
O_{p2}	i2	p	-
O_q	-	q	-

Figure 6: An example illustrating the refinement process of UCP. The domain description is provided in the table on the right. The problem is specified with the initial state $\{i'\}$ and the goal state $\{G'\}$. The partial plans are shown in terms of their steps, ordering and contiguity constraints.

on the right in the figure. The search starts with the null plan $\mathcal{P}_\emptyset$ in which t_0 precedes t_∞. Suppose the `pick-refinement` function suggests FSS refinement in the first iteration (such a suggestion may be based on the expected cost of the refinement; see below). There are only two library operators that are applicable to the header state (the tail fringe consists only of the tail step t_∞ which is not applicable). Two refinements, $\mathcal{P}_1$ and $\mathcal{P}_2$, each corresponding to the application of the respective operator to the header of $\mathcal{P}_\emptyset$, are produced. Next, suppose UCP picks $\mathcal{P}_1$ from the search queue. At this point, a BSS refinement strategy is chosen. This produces a single refinement $\mathcal{P}_3$, involving the application of the library operator O' to the trailer. Next, $\mathcal{P}_3$ is chosen from the search queue, and the PS refinement is selected, with the precondition $\langle p, t_2\rangle$ to be established. (To make discussion simple, we assume a PS refinement that does not impose any optional bookkeeping constraints.) Two refinements, $\mathcal{P}_4$ and $\mathcal{P}_5$ result. UCP chooses $\mathcal{P}_4$ from the search queue and refines it further with another PS refinement with $\langle q, t_2\rangle$ as the precondition to be established. This results in a single refinement, $\mathcal{P}_6$ which satisfies the termination test (it has a single safe ground linearization which is a solution for the problem). The search ends successfully when $\mathcal{P}_6$ is picked up from the search queue in the next iteration (note that in $\mathcal{P}_6$, the steps t_3 and t_4 are unordered with respect to each other).

5 Coverage, Completeness and Systematicity of UCP

Coverage: It is easy to see that by choosing the `pick-refinement` function appropriately, we can model the pure state space planners, as well as the pure plan-space planners as instantiations of UCP. In particular, in [10, 8, 9], we showed that the plan space refinement template, given in Figure 5 covers the complete spectrum of plan space planners, including UA, TO [15], SNLP [12], and TWEAK [2]. What is more interesting, as shown in Section 4.1, the UCP algorithm also allows hybrid planners that can opportunistically apply plan-space as well as state-space refinements within a single planning episode. For example, the classical means-ends analysis planners such as STRIPS [3], or their descendents such as PRODIGY [4] can be modeled by a `pick-refinement` such as the following: *If there is a step in*

the head-fringe of the plan that is applicable to the head-state, pick forward state space refinement. Else, pick plan-space refinement. Finally, it is also possible to use a more ambitious `pick-refinement` strategy: *pick the refinement that has the least expected cost* (see below) [6].

Completeness: It is fairly straightforward to show that the three refinement strategies FSS, BSS are individually complete in that they do not lose any potential solutions. The completeness of PS refinement follows from the results of Pednault [16, 10]. Since all three refinements are individually complete, any instantiation of UCP that uses these refinements will also be complete (see Section 2).[3]

Systematicity: It is possible to prove that *any instantiation of UCP that uses a systematic PS refinement is systematic, regardless of the way it chooses to interleave the FSS, BSS and PS refinements*. Recall that as pointed out by McAllester (see [12, 10]), PS refinement is systematic as long as the book-keeping step uses contributor protections (that is, whenever a precondition p of a step t is established through the effects of another step t', two IPCs $\langle t', p, t\rangle$ and $\langle t', \neg p, t\rangle$ are added to the list of auxiliary constraints of the plan [10]). Since the systematicity claim may not be obvious at first glance, we will provide a proof sketch in Appendix A.

6 Controlling UCP

In this section, we will consider the types of control strategies (heuristics, pruning techniques, selection strategies etc.) that are appropriate for UCP. In general, UCP requires heuristic guidance in selecting a partial plan to be refined next, and in picking the refinement strategy to apply to the selected partial plan. Additionally, if a plan-space refinement strategy is chosen, UCP needs guidance regarding which goal to select for establishment. The first is a backtrackable decision, while the latter two don't need to be backtracked.

For plan selection, in addition to the heuristics that are applicable to pure plan space and state space planners, UCP can also use hybrid heuristics tailored to its partial plan representation. For example, its plan selection heuristics could prefer FSS refinements that correspond to applying head fringe operators (rather than new operators from library) to the header; or evaluate the promise of the plan in terms of the set difference between the tail state and the head state.

The selection of refinement strategy can be done in many ways, and the tradeoffs offered by the various strategies is still an open question. In our preliminary studies (reported in the next section), we experimented with three hybrid strategies. The first, called UCP-MEA prefers FSS whenever a head fringe step is applicable to the head state, and PS otherwise. UCP-MEA has a generalized means-ends analysis flavor and thus simulates planners such as STRIPS and PRODIGY. The second, called UCP-MBA is similar to UCP-MEA, with the exception that before picking PS, it checks to see if a step on tail-fringe is applicable to the tail state, and if so, picks BSS. The third one, called UCP-LCFR, estimates the number of refinements generated by each of the three refinement strategies and selects the one that has the least number of refinements. This strategy is inspired by the least cost flaw refinement strategy, that was recently suggested by Joslin and Pollack [6].

If a PS refinement is selected, UCP still faces the decision of which goal to achieve (each choice corresponds to a complete PS refinement with respect to that goal). In addition to the

[3]This is true as long as UCP uses a termination criterion that effectively checks to see if *a* safe ground linearization of the plan corresponds to the solution (see [10]).

goal selection strategies used by pure plan space planners, UCP could also use the head state information. For example, it might prefer those goals that are not already true in the head state, or give preference to the preconditions of operators on the head fringe that have least number of unsatisfied preconditions.

7 Preliminary empirical studies on the utility of interleaving refinements

We have implemented the UCP algorithm on top of the Refine-Plan implementation described in [10]. Since UCP provides a framework to interleave the three different refinements within a single problem episode, we conducted several preliminary experiments to evaluate the advantages of such interleaving.

Experimental Setup: We considered six different instantiations of UCP. The first three, UCP-PS, UCP-FSS and UCP-BSS, always pick the same type of refinement, and correspond respectively to plan-space, forward state-space and backward state-space planners. The other three, UCP-MEA, UCP-MBA and UCP-LCFR correspond to the three hybrid refinement strategies described in the previous section.

Although the plan space refinement can have considerable variation [10] based on the protection strategies and goal selection strategies used, in our experiments, we kept it constant. We used a simple LIFO strategy for selecting the open-condition to be established, contributor protections for bookkeeping, and conflict resolution for tractability refinements. This is equivalent to the refinement strategy used by SNLP [12, 10]. (In [10] we discuss the performance tradeoffs offered by the other ways of instantiating the plan-space refinement).

All the experiments used best-first search, with the ranking function defined as the sum of number of steps, open conditions, unsafe links, and the number of conditions of tail state that are not present in the head state. Additionally, partial plans that are inconsistent, or contain state looping (see Section 4) are pruned. Each planner was given a cpu time limit of 120 seconds for solving any problem. The time limit was increased to 300 seconds in the case of UCP-LCFR as our simple implementation estimates the branching factors of each refinement by actually simulating the refinement (other more efficient approximate estimation methods are of course possible; see [6]).

Domains and Results: We conducted experiments both in blocks world and in a variety of artificial domains designed by various researchers to showcase the advantages of one planning approach over another. Our intent was to show that appropriate hybrid instantiations of UCP may do well in all such domains.

The first domain, called *link-chain* domain, was designed by Veloso & Blythe [18] to showcase the advantages of state space means-ends analysis planners over plan-space planners. This domain contains ten actions A_1 to A_{10}. Each action A_i requires the preconditions $G_1, G_2, \cdots G_{i-1}$, adds G_i, and $G_1, G_2, \cdots G_{i-2}$ and deletes G_{i-1}. The leftmost plot in Figure 7 shows the results of our experiments in this domain. We note that UCP-MBA and UCP-LCFR outperform all the other planners including UCP-MEA. (the plots of UCP-MBA, UCP-LCFR and UCP-FSS are all together as their performance was very close). The first plot in Figure 8 shows the fraction of times the three individual refinements were employed by UCP-LCFR during planning. We note that as the problem size increases the relative frequency of FSS increases with respect to BSS. This correlates well with the fact that UCP-LCFR tracks the performance of UCP-FSS in terms of number of partial plans refined, while UCP-BSS worsens its performance as the problem size increases.

The second domain, called $\theta_2 D^m S^1$, is one of the domains designed by Barrett & Weld

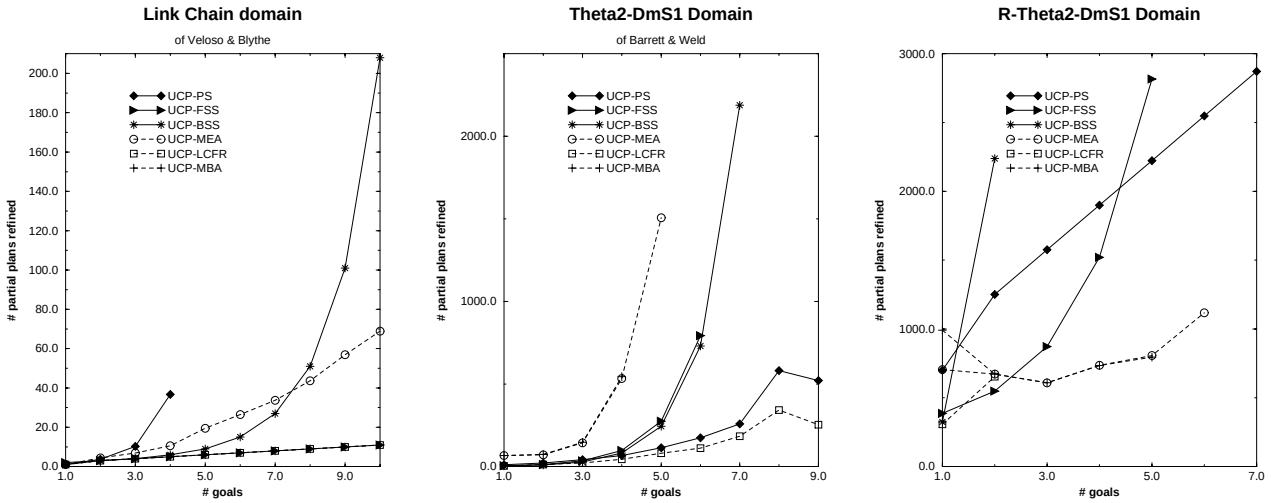


Figure 7: Plots illustrating the performance of various instantiations of UCP (measured in terms of the number of refinements made). Each point in the plot corresponds to the average over ten problems of a given number of goals. Missing points in a plot signify that the planner could not solve some of the ten problems within the allotted cpu time.

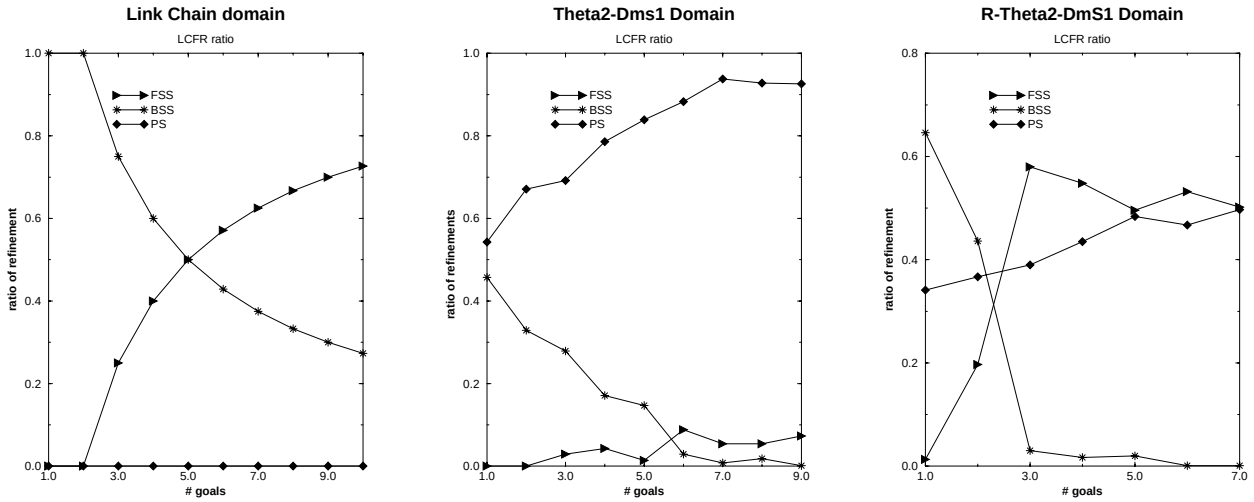


Figure 8: Plots showing the fractions of various refinements used by UCP-LCFR

[1] to demonstrate the advantages of plan space planning over state space planning. In this domain, each top level goal G_i can be achieved by either of two actions A_i^1 or A_i^2. All actions of type A_i^1 require P_α while all actions of type A_1^2 require P_β as a precondition. The initial state contains either P_α or P_β (but not both). The results of our experiments in this domain are shown in the second plot in Figure 7. Once again, we note that a hybrid instance of UCP, viz., UCP-LCFR, outperforms UCP-PSS. An analysis of the pattern of refinements used by UCP-LCFR, shown in the second plot in Figure 8 reveals that it outperforms UCP-PSS by opportunistically using BSS and FSS refinements in a small percentage of iterations. In other words, even a domain that motivates pure plan-space planning over pure state space planning [1], can benefit from a proper mix of the two types refinements!

Finally, Figure 9 compares the performance of pure and hybrid instantiations of UCP in the standard blocks world domain (which, unlike the three artificial domains, is a non-propositional domain). The problems were generated using the random blocks world problem generator described in [14]. Each data point represents the average over 10 random problems

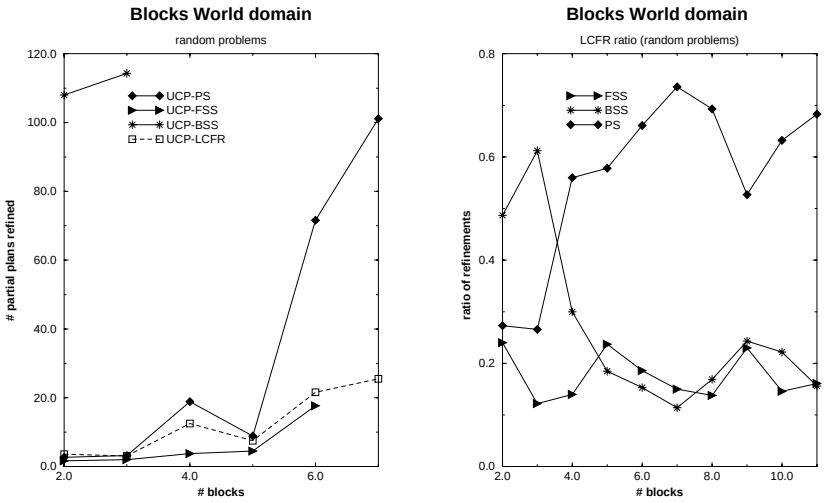


Figure 9: Performance of the instantiations of UCP in blocks world

containing a specified number of blocks. The data points in the graph correspond to situations where all 10 random problems were solved by the specific instantiation of UCP. UCP-BSS failed to achieve 100% solvability anywhere in this problem population, while UCP-FSS does quite well. Moreover, the performance of UCP-LCFR is close to that of UCP-FSS. The second plot shows the distribution of individual refinements used by UCP-LCFR. We note that UCP-LCFR achieves its performance by judiciously combining a small fraction of FSS and BSS refinements with a majority of PS refinements.

Discussion: Our results show that hybrid instances of UCP, that opportunistically interleave state space and plan space refinements, can do better than either the pure plan space or the pure state space refinements. They thus demonstrate the potential benefits of finding appropriate strategies for interleaving different refinements. Although our hybrid control strategies showed some promise, we believe that *the question of right control strategy is still an open one*. In particular, even though UCP-LCFR did well in both *link-chain* and $\theta_2 D^m S^1$ domains, we found that it is by no means infallible. Our experiments with a variant of $\theta_2 D^m S^1$, called $R\theta_2 D^m S^1$, results of which are shown in the third plots in Figures 7 and 8 demonstrate that UCP-LCFR can be mislead under some circumstances[4] as the branching factor of a refinement is not a fool-proof indicator of its heuristic utility. We are currently exploring other features that are may be more indicative of the cost of a refinement.

8 Related Work

Earlier work on unifying classical planning approaches includes Rosenchien's work [17] on *bigression* planner, which combines a forward state space search and a backward state space search; and our own more recent work [10, 8, 9] unifying a variety of plan-space planning frameworks into one algorithm template. The plan-space refinement in Figure 5 is directly taken from this latter work. To our knowledge, this paper is the first to rationally place the plan-space and state-space refinements in one unifying framework.

Fink and Veloso [4] describe the PRODIGY 4.0 planner which does an interesting combination of forward state space refinement combined with means-ends analysis. There are several differences between the PRODIGY 4.0 algorithm and the approach presented

[4]In this domain, the initial state contains either Q_α or Q_β. There are two sets of five actions each of which, when done in sequence will convert Q_α into P_α and Q_β into P_β. Finally, there are a set of dummy actions which add P_α or P_β but their preconditions are never satisfied.

here. To begin with, PRODIGY 4.0 does not employ a complete plan-space refinement. It has to backtrack on the decision to apply an operator to the header, and it will terminate only when the header state reaches the goal state. The tail part of the PRODIGY 4.0's plan does not have any well-defined semantics and does not constrain the solutions represented by the partial plan. For example, the fact that a set of steps occurs in the tail of a partial plan does not mean that any eventual solution under that partial plan will contain those steps. In contrast, steps that are part of the UCP partial plan must be present in any candidate of that partial plan. Further, based on the set of refinements it selects, UCP can act as a pure plan-space, pure state-space or a hybrid planner.

9 Conclusion and Future Work

In this paper, we presented a generalized algorithm template, called UCP, which allows plan-space and state-space refinements on a single partial plan representation. UCP provides a parameterized planner template whose completeness and systematicity are ensured by the corresponding properties of the individual refinements. It thus provides a framework for opportunistically combining plan-space and state-space refinements within a single planning episode. Depending upon the control strategy used, instantiations of UCP correspond to pure state-space, pure plan-space as well as hybrid planners that facilitate strategies such as means-ends analysis. We have discussed the issues of coverage, completeness and systematicity of the instantiations of UCP. Apart from its significant pedagogical advantages, our unified framework also promises considerable algorithmic advantages. To begin with, our implementation of UCP provides a normalized substrate for comparing the various refinement strategies. Our experiments with this implementation also demonstrate the potential benefits of opportunistically interleaving the plan-space and state-space refinements.

In the immediate future, we plan to concentrate on designing better control strategies for interleaving the different refinements in UCP. Several other extensions are planned for the long term. First, although we concentrated on unifying state space and plan-space planning approaches in this paper, our framework is powerful enough to also include the task reduction planning approach. In particular, in [11], we describe how task reduction planning and plan-space planning can be put in a common refinement planning framework. That approach can be used to extend UCP in a straightforward way to cover task reduction planners. Finally, although our original motivation has been to find a framework that unifies plan-space and state-space refinements, and allows them to be interleaved, the representation of partial plans that we discussed in this paper also facilitates novel forms of refinements other than PS, FSS and BSS. For example, we could refine partial plans by ''blocking'' some steps in the plan together so that they are constrained to come immediately next to each other.[5] Our preliminary studies show that such a refinement strategy can be used to reduce some of the wasteful conflict detection and resolution effort expended by the plan-space refinements.

References

[1] A. Barrett and D. Weld. Partial Order Planning: Evaluating Possible Efficiency Gains. *Artificial Intelligence*, Vol. 67, No. 1, 1994.

[2] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333--377, 1987.

[3] R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. In *Readings in Planning*. Morgan Kaufmann, 1990.

[4] E. Fink and M. Veloso. Formalizing the Prodigy Planning Algorithm. CMU CS Tech. Report, Fall 1994.

[5] FSS and BSS refinements can be seen as blocking steps to the prefix and suffix of the plan

[5] M. Ginsberg. Approximate Planning. *Artificial Intelligence*, Special Issue on Planning, Scheduling and Control. 1995.

[6] D. Joslin and M. Pollack. Least-cost flaw repair: A plan refinement strategy for partial order planning. *Proceedings of AAAI-94*, 1994.

[7] S. Kambhampati and B. Srivastava. Universal Classical Planner: The details ASU CSE Tech. Report., 1995 (in preparation).

[8] S. Kambhampati. Refinement search as a unifying framework for analyzing planning algorithms. In *Proc. KR-94*, May 1994.

[9] S. Kambhampati. Design Tradeoffs in Partial Order (Plan Space) Planning. In *Proc. 2nd Intl. Conf. on AI Planning Systems (AIPS-94)*, June 1994.

[10] S. Kambhampati, C. Knoblock and Q. Yang. Planning as Refinement Search: A Unified framework for evaluating design tradeoffs in partial order planning. *Artificial Intelligence* special issue on Planning and Scheduling. Vol. 76. 1995.

[11] S. Kambhampati. A comparative analysis of partial-order planning and task reduction planning. ACM SIGART Bulletin, Special Section on Evaluating Plans, Planners and Planning agents, Vol. 6., No. 1, January, 1995.

[12] D. McAllester and D. Rosenblitt. Systematic Nonlinear Planning. In *Proc. 9th AAAI*, 1991.

[13] D. McDermott. Invited talk, AIPS-94, June 1994.

[14] S. Minton. *Learning Effective Search Control Knowledge: An Explanation-Based Approach.* PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, 1988.

[15] S. Minton, J. Bresina and M. Drummond. Total Order and Partial Order Planning: a comparative analysis. Journal of Artificial Intelligence Research 2 (1994) 227-262.

[16] E.P.D. Pednault. Generalizing nonlinear planning to handle complex goals and actions with context dependent effects. In *Proc. IJCAI-91*, 1991.

[17] S. Rosenchien. Plan Synthesis: A logical perspective. *Proc. IJCAI-81*, 1981.

[18] M. Veloso and J. Blythe. Linkability: Examining causal link commitments in partial-order planning. *Proceedings of AIPS-94*, 1994.

[19] D. Weld. Introduction to Partial Order Planning. *AI Magazine*, Vol. 15, No. 4, 1994.

A Proof of Systematicity of UCP

In this appendix, we sketch the proof of systematicity of any instantiation of UCP that uses a PS refinement with contributor protections. To make exposition simple, we will concentrate on the systematicity of FSS refinement both in isolation, and in the presence of other refinements. Similar arguments hold for the systematicity of BSS and PS refinements.

If UCP uses only FSS refinement, its head fringe will always consist only of the goal step t_∞, and thus the only way FSS refines the plan is by introducing operators from library into the header. Since each FSS refinement will have a different library operator instance added to the end of the header, and since the operators in the header of a partial plan will form the prefix of each candidate of that partial plan, the candidate sets of different FSS refinements will be disjoint.

The above argument about differing prefixes may not hold when FSS is being used in conjunction with BSS and PS since in such cases FSS refinements will involve transferring steps from both the library and the *head fringe* into the header. It is possible in such cases to have the same operator instance (say O) introduced into the header in more than one refinement-- once from the head fringe and once from the library. We will now show that systematicity still holds as the candidates of different refinements differ in terms of the

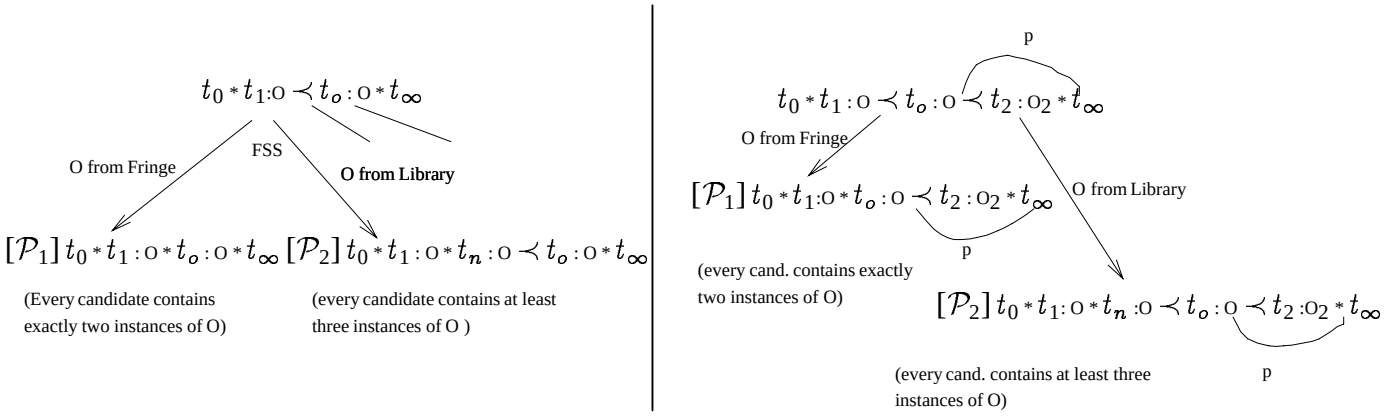


Figure 10: Examples illustrating the systematicity of UCP. The curved lines on the right correspond to IPC constraints on the plan.

number of instances of the operator O. We need to consider two cases -- one in which the instance of O on header fringe has been introduced by a previous PSS refinement, and the other in which it is introduced by a previous BSS refinement.

Case 1 O **introduced by BSS:** The illustration on the left hand side of Figure 10 shows a scenario where the step $t_o : O$ on the head fringe has been introduced by BSS (and thus it is part of the trailer). Now consider two refinements $\mathcal{P}_1$ and $\mathcal{P}_2$ generated by FSS by introducing $t_o : O$ from the fringe, and $t_n : O$ from library, respectively into the header. It is easy to see that every candidate of $\mathcal{P}_1$ will have at least one instance of O less than every candidate of $\mathcal{P}_2$ (in this example, candidates of $\mathcal{P}_1$ will have exactly two instances of O, while candidates of $\mathcal{P}_2$ will have three or more instances of O.

Case 2 O **introduced by PS:** The illustration on the right hand side of Figure 10 shows a scenario where the step $t_o : O$ on the head fringe has been introduced by PSS (using contributor protection, as per the premises of the theorem). Suppose without loss of generality that $t_o : O$ was introduced to give some condition p to t_∞. This means that the partial plan contains two IPCs $\langle t_o, p, t_\infty \rangle$ and $\langle t_o, \neg p, t_\infty \rangle$. Now consider two refinements $\mathcal{P}_1$ and $\mathcal{P}_2$ generated by FSS by transferring an instance of O from the fringe, and from the library, respectively to the header. Once again the candidates of $\mathcal{P}_1$ and $\mathcal{P}_2$ will differ in terms of the number of instances of O. In the example plan shown in Figure 10, every candidate of $\mathcal{P}_1$ will have exactly two instances of O (no new instances O can come after the header since they will violate the IPC $\langle t_o, \neg p, t_\infty \rangle$), whereas every candidate of $\mathcal{P}_2$ will at least have three instances of O. ■