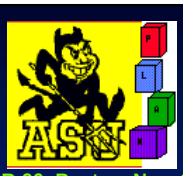
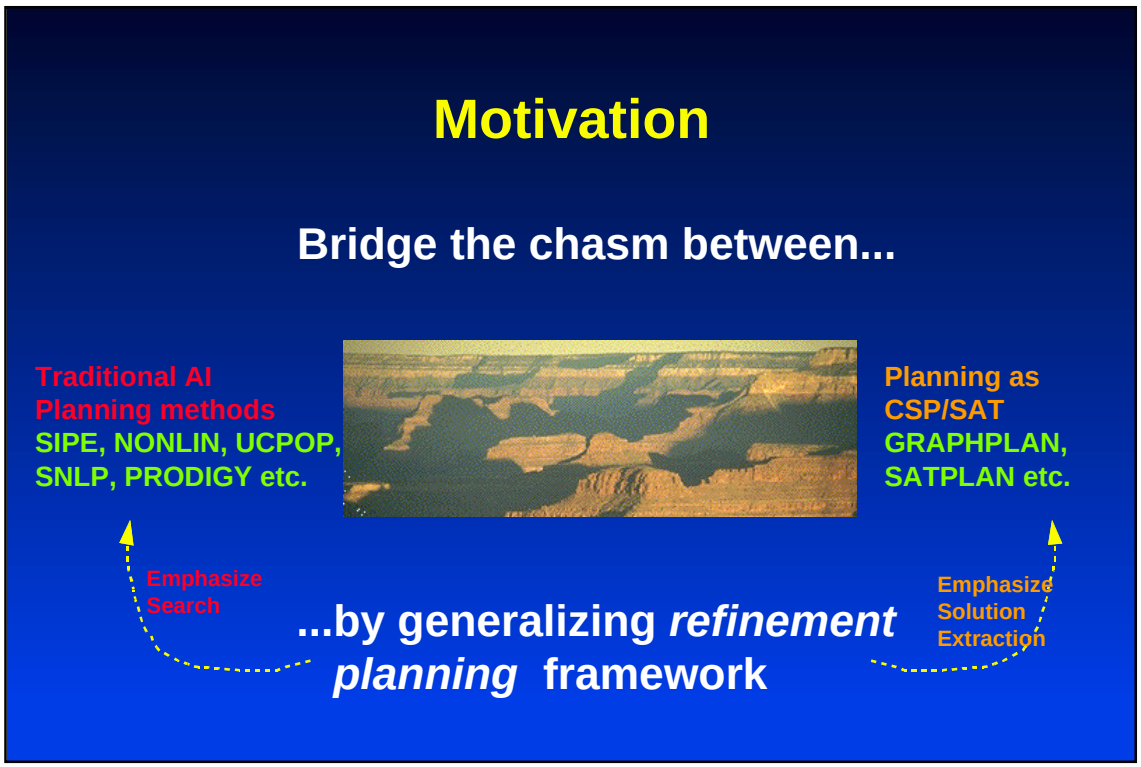


KRR-96; Boston. Nov, 96.

On the role of Disjunctive Representations and Constraint Propagation in Refinement Planning

Subbarao Kambhampati & Xiuping Yang
Arizona State University
http://rakaposhi.eas.asu.edu

Presented by Amol Mali



Most traditional approaches to plan generation, developed over the last twenty years work by searching in the space of partial plans, extending them incrementally until they become a solution, and backtracking when a plan can no longer be fruitfully extended.

More recently, several algorithms -- Graphplan, SATPLAn and Descartes -- have developed competing approaches that cast planning as a constraint satisfaction problem.

At the first glance, there appears to be huge gulf between the traditional refinement planning algorithms and these new breed of planners based on CSP/SAT techniques.

In our previous work, we have unified the traditional planning algorithms under the rubric of refinement planning paradigm.

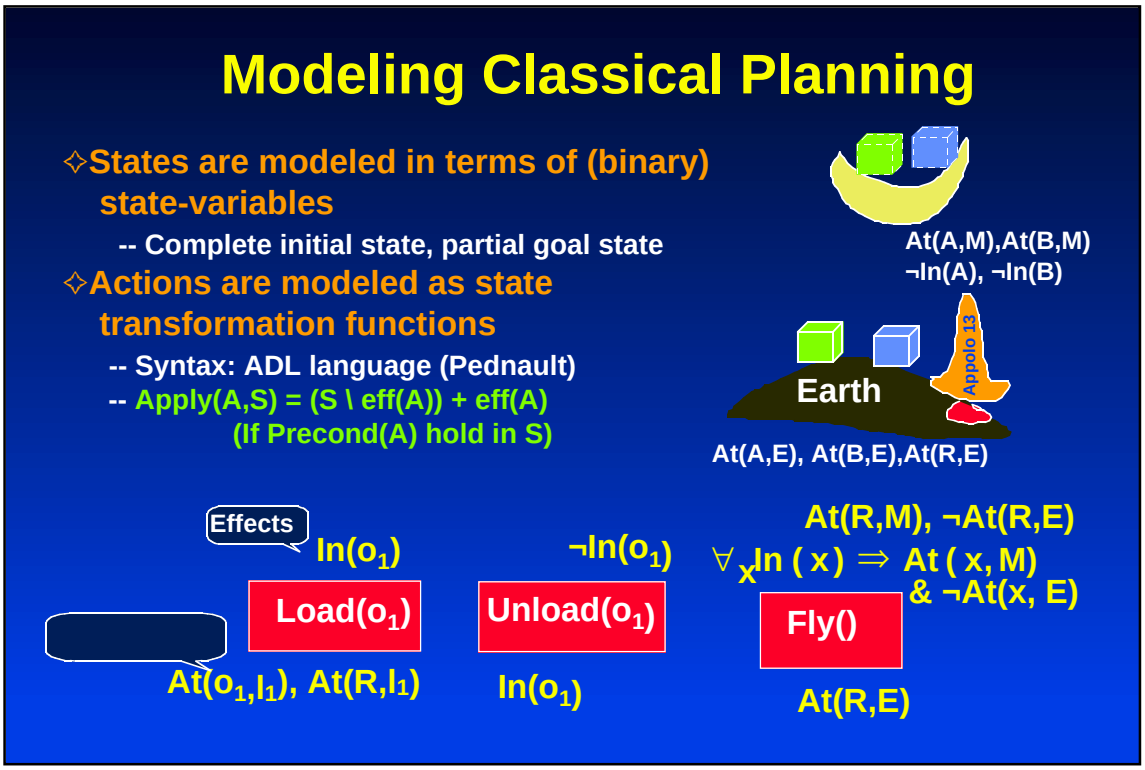
In this paper, we generalize the this framework to subsume both traditional refinement planners and the CSP based planners. We wil see that the major tradeoffs between the two types of planners is how much work is done in “how much work is done refining and backtracking through individual partial plans” vs. “extracting a solution from a set of partial plans”. Traditional planners emphasize the former while the CSP approaches emphasize the latter.

Our generalized approach will also bring to fore a continuum of approaches between these two classes of planning algorithms.

Overview

- Generalized Refinement Planning Framework
  - Role of search vs. solution extraction
- Utility of disjunctive representations
  - Refining disjunctive plans
  - Constraint propagation to focus refinements
- Putting the framework to work...
  - Graphplan
  - Planning as Satisfiability
  - UCPOP-D(*)
- Future directions and Conclusion

Here is the outline of the talk. We will start with a view of refinement planning that generalizes on the framework we presented in KR-94 and is capable of subsuming both the traditional and newer approaches to plan generation. We will show that the newer approaches refine sets of plans and keep the resulting sets together without splitting. We will then argue that disjunctive representations and constraint propagation help in handling sets of plans together. Finally, we will show how this generalized framework advances both our understanding and the state of the art by considering several new planners within our framework -- inclduing Graphplan, SATPLAN, and a variant of UCPOP called UCPOP-D. We will end with a discussion of future research directions.

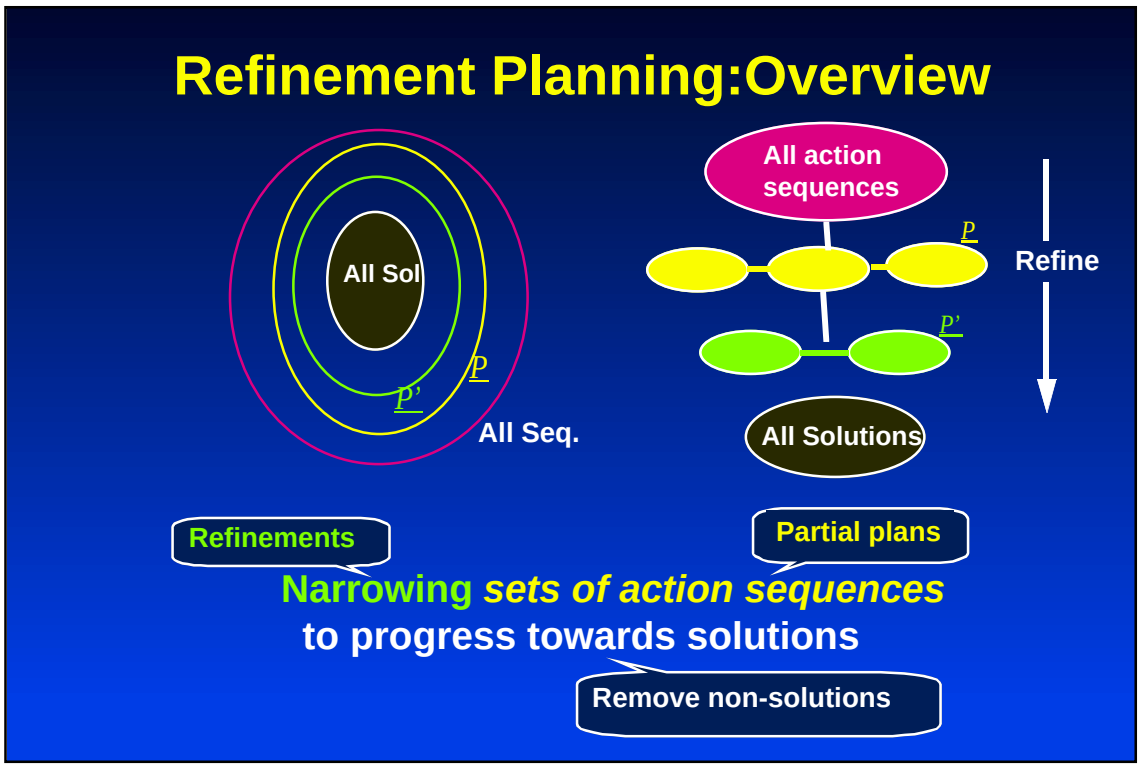


Let us start by recalling that the classical planning problem involves finding a sequence of actions which when executed will take the agent from a given initial state to a desired goal state. In this example, we are interested in transporting two packets from earth to Moon, using a single (and somewhat out-of-shape) rocket.

States of the world are modeled in terms of a bunch of binary state-variables.

Actions are modeled as state-transformation functions, with pre-conditions and effects.

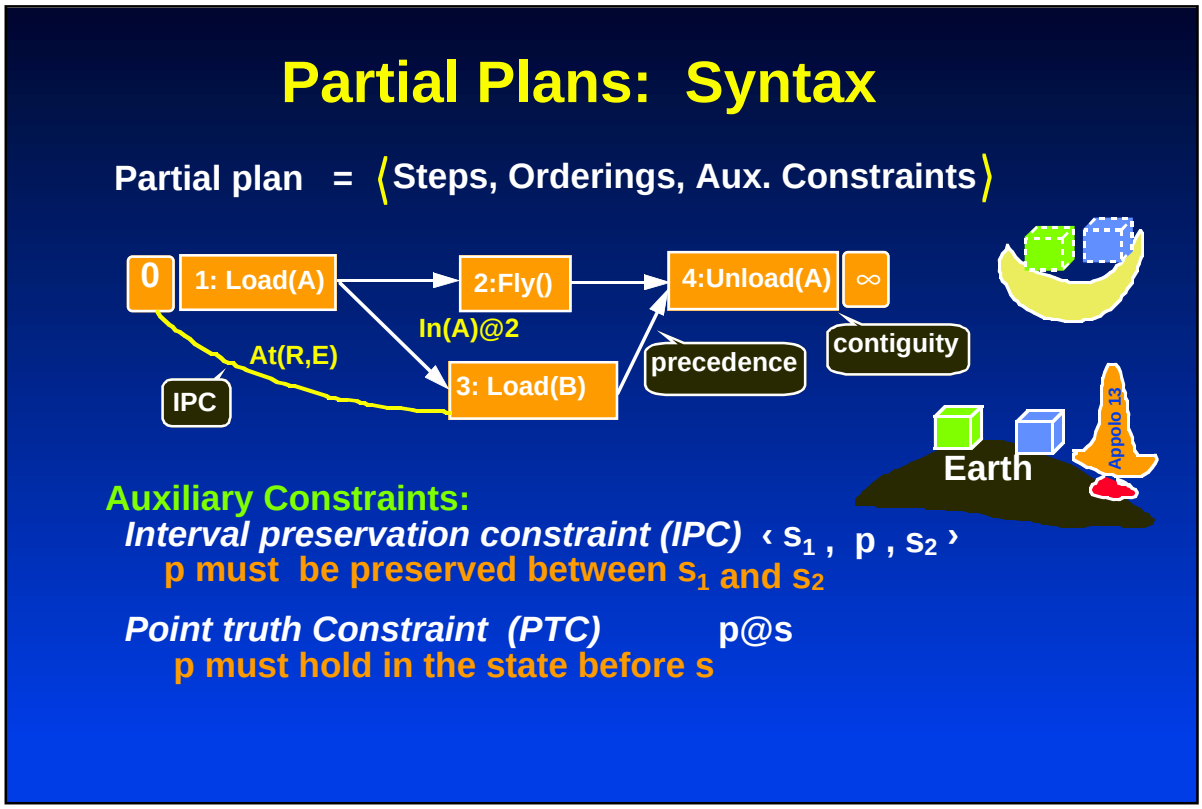
We have three actions in our rocket domain -- load which makes a package to be IN the rocket, unload, which gets it out, and Fly, which takes everything in the rocket over to the moon..



Refinement planning can be thought of as a process of gradually narrowing down the set of all actions sequences so as to progress towards the set of all solutions.

Sets of action sequences are represented as partial plans. Narrowing is done by refinement operations which add “constraints” to partial plans. Finally, progress is measured by the ability to eliminate the sequences that cannot be solutions to the problem. (if no solutions are eliminated in this process, we will gradually get closer to the set of all solutions). Termination can occur any time an action sequence capable of solving the problem can be picked up from among the sequences currently under consideration.

To make these ideas precise, we shall now look at the syntax and semantics of partial plans and refinement operations.



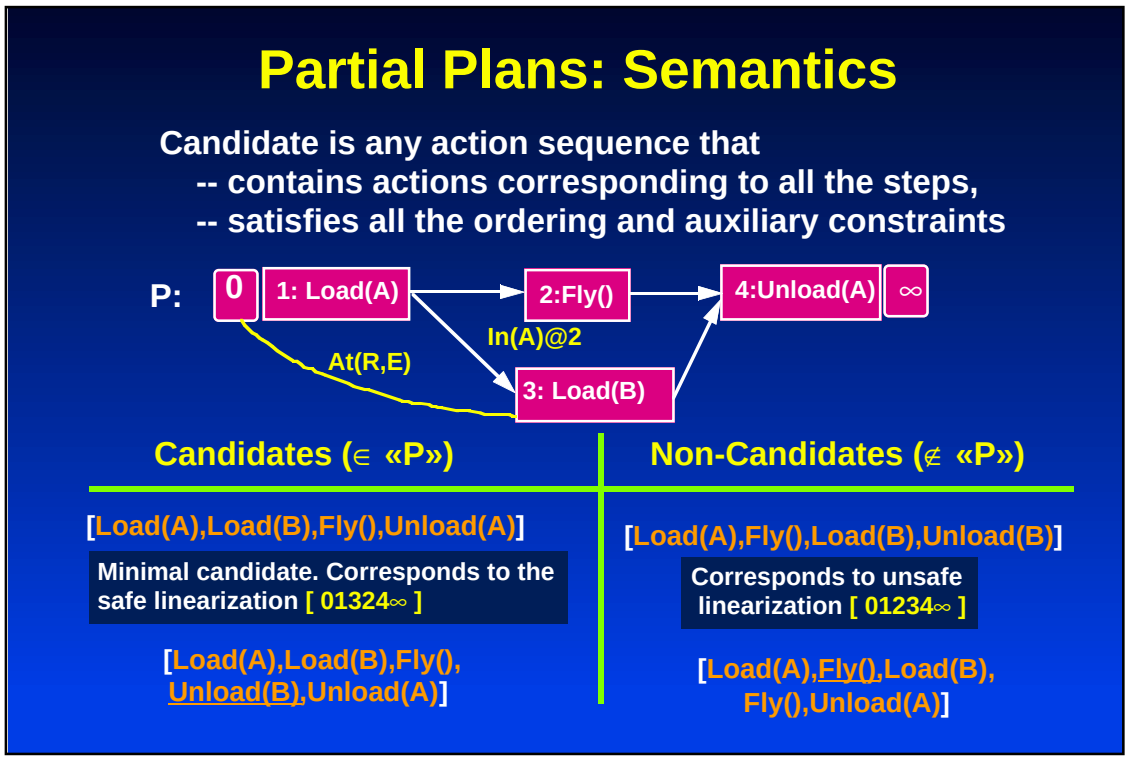
A partial plan can be seen as any set of constraints that together delineate which action sequences belong to the plan's candidate set and which do not.

One representation that is sufficient for our purposes models partial plans as a set of steps, ordering constraints between the steps, and auxiliary constraints. Each plan step corresponds to an action. There are two types of ordering constraints -- precedence and contiguity constraints. The latter require that two steps come immediately next to each other.

Auxiliary constraints involve statements about truth of certain conditions over time intervals. These come in two important types -- *interval preservation constraints* which require preservation of a condition over an interval, and *point truth constraints* that require the truth of a condition at a particular time point.

Here is an example plan from our rocket domain in this representation. The steps 0 and 1 are contiguous, 2 precedes 4, and the condition At(R,E) must be preserved between 0 and 3.

Finally, the condition In(A) must hold in the state preceding the execution of step 2. (*This is in addition to the constraint that all preconditions of an action must hold in the state preceding the action.)*



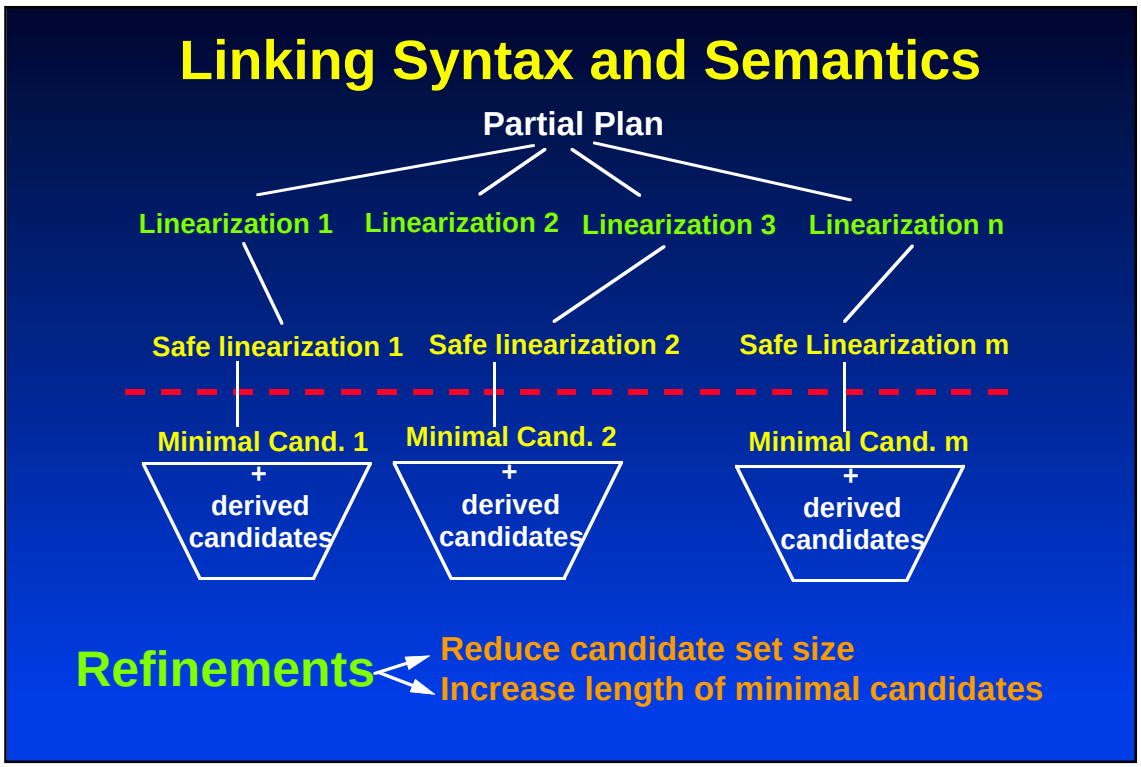
The semantics of the partial plans are given in terms of candidate sets. An action sequence belongs to the candidate set of a partial plan if it contains the actions corresponding to all the steps of the partial plan, in an order consistent with the ordering constraints on the plan, and it also satisfies all auxiliary constraints.

For the example plan shown here, the sequences on the left are candidates while those on the right are non-candidates. Notice that the candidates may contain more actions than are present in the partial plan.

Candidates that only contain the actions in the plan are called “minimal candidates”. These correspond to the syntactic notion of safe linearizations.

Safe linearizations are linearizations (or topological sorts) of the plan steps that also satisfy the auxiliary constraints. The linearization 0-1-3-2-4-infty is a safe one while the linearization 0-1-2-3-4-infty is not (since step 2 will violate the IPC on At(R,E).

The sequences on the right are non-candidates because both of them fail to satisfy the auxiliary constraints



Here then is the connection between the syntax and semantics of a partial plan. Each partial plan has at most exponential number of linearizations, some of which are safe with respect to the auxiliary constraints.

Each safe linearization corresponds to a minimal candidate of the plan. Thus, there are at most exponential number of minimal candidates. A potentially infinite number of additional candidates can be derived from each minimal candidate by padding it with new actions without violating auxiliary constraints.

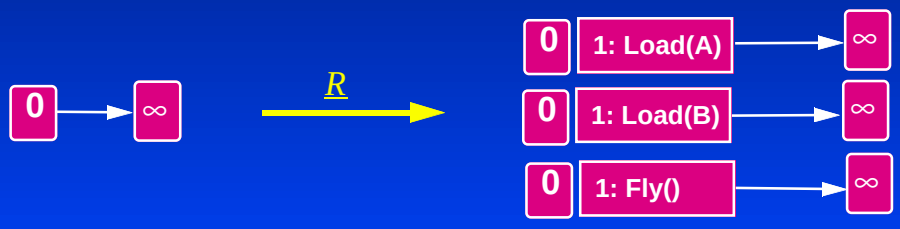
Refinements add new constraints to a partial plan. They thus simultaneously shrink the candidate set of the plan, and increase the length of its minimal candidates.

Thus, one incremental way of exploring the candidate set of a plan for solutions is to check through its minimal candidates after refinements.

Refinement Strategies

§ A *plan set* P is a set of partial plans $\{P_1, P_2 \ldots P_m\}$

✧A refinement strategy $R : P \rightarrowtail P'$ («P' » a subset of «P»)
- –R is complete if «P' » contains all the solutions of «P »
- –R is progressive if «P' » is a proper subset of «P »
- –R is systematic if components of P' don't share candidates



We will now formally define a refinement strategy. Refinement strategies operate on sets of partial plans. We thus define a plan-set as a set of partial plans, with the understanding that the candidate set of the planset is the union of the candidate sets of its component plans.

A refinement strategy R maps a plan set P to another plan set P' such that the candidate set of P' is a subset of the candidate set of P. R is said to be complete if P' contains all the solutions of P. It is said to be progressive if the candidate set of P' is a strict subset of the candidate set of P. It is said to be systematic if no action sequence falls in the candidate set of more than one component of P'.

Completeness ensures that we don't lose solutions by the application of refinements. Progressiveness ensures that refinement narrows the candidate set. Systematicity ensures that we never consider the same candidate more than once.

At the bottom is an example refinement, for our rocket problem, which takes the null plan, corresponding to all action sequences and maps it to a plan set containing 3 components. (In this case, the refinement is complete since no solution can start with any other action, progressive since it eliminated action sequences beginning with unload(A) etc, and systematic since all the candidates of the three components will have different prefixes.)

Refinement Planning Template

Refine (*P* : Plan set)

0*. If «*P* » is empty, Fail.
1. If a minimal candidate of *P* is a solution, return it. End
2. Select a refinement strategy *R*
Apply *R* to *P* to get a new plan set *P'*
3. Call Refine(*P'*)

State-space, Plan-space, HTN, Tractability

--Termination ensured if *R* is complete and progressive

-- Solution extraction (step 2) involves checking exponentially many minimal candidates
-- Can be cast as propositional model-finding (satisfaction)

We are now in a position to present the general refinement planning template. It has three main steps.

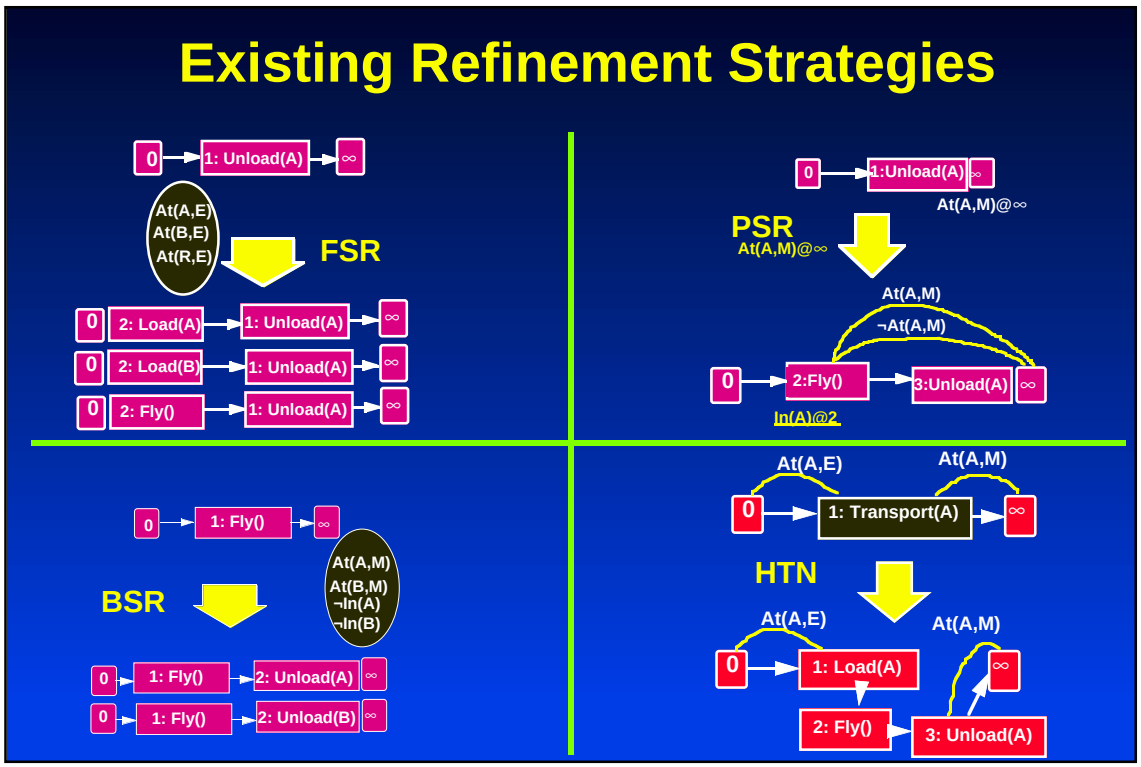
If the current plan-set has an extractable solution -- which is found by inspecting its minimal candidates, we terminate.

If not, we select a refinement strategy R and apply it to the current plan set to get a new plan set.

As long as the selected refinement strategy is complete, we will never lose a solution. As long as the refinements are progressive, for solvable problems, we will eventually reach a planset one of whose minimal candidates will be a solution (the figure on the top right illustrates this).

The solution extraction process involves checking an exponential number of minimal candidates (corresponding to safe linearizations). This can be cast as a model-finding or satisfaction process.

Some recent planners like Graphplan and Satplan can be seen as instantiations of this general refinement planning template. However, most earlier planners use a specialization of this template that we shall discuss next.



Let us briefly describe the various known refinement strategies.

The forward state space refinement refines a partial plan by extending its prefix with each of the actions that can be applied in the state of the world following the prefix. The backward state space refinement works analogously by extending the suffix of the partial plan.

THe plan space refinement adds steps into a partial plan without constraining their absolute position. For example, here, the step unload is added to the plan to achieve At(A,M) without specifying when exactly it will take place.

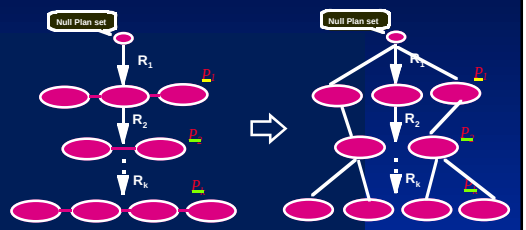
Finally, the task reduction (or HTN) refinements replace an abstract action by its (user-specified) reductions.

All these refinements are complete and progressive. They can all be made systematic.

Combining Refinement with Search

Refine (P : Plan)

0*. If «P » is empty, Fail.
1. If a minimal candidate of P is a solution, terminate.
2. Select a refinement strategy R .
Appply R to P to get a new plan set P'
3. Split P' into k plansets
4. Simplify plansets by introducing disjunction and constraint propagation.
5. Non-deterministically select one of the plansets P'$_i$
Call Refine(P'$_i$)



The algorithm template in the previous slide does not have any search in the foreground All the search is pushed into the solution extraction function.

It is possible to add "search" to the refinement process in a straightforward way. The algorithm template shown here introduces search into refinement planning. It does this by partitioning a plan set into k smaller plan sets, and handling each of these in a diferent search branch. As k increase, the cost of handling plansets is reduced by pushing complexity into the search space size.

(This can be said wrt the next slide) All traditional refinement planners, such as UCPOP, Prodigy, SNLP, etc. correspond to complete partitioning -- ie. k= # component sof the plan. We shall see that planners such as GRAPHPLAN can be seen as doing no partition -- ie.e k=1. It sis also possible to do medium levels f partitioning as we shall see when discussing UCPOP-D.

A Spectrum of Refinement Planners

Refine (P_i: Plan)

0*. If «P_i» is empty, Fail.
1. If a minimal candidate of P_i is a solution, terminate.
2. Select a refinement strategy R_i. Appply R_i to P_i to get a new plan set P_i'
3. Split P_i' into k plansets
4. Simplify plansets by introducing disjunction and constraint propagation.
5. Non-deterministically select one of the plansets P_i'. Call Refine(P_i')

Planner	Refinement	Splitting (k)
UCPOP, SNLP	PSR	k = #Comp
TOPI	BSR	k = #Comp
Graphplan	FSR	k = 1
SATPLAN	FSR, PSR	k = 1
UCPOP-D Descartes	PSR	1 < k < #comp

(This slide is a continuation of the previous one.)

The algorithm in the previous slide can cover both traditional refinement planners and the recent planners such as Graphlan based on the value of splitting factor k, and the type of refinement strategy selected.

For example, UCPOP and SNLP correspond to the choice of plan-space refinement with full splitting (k = # components of the plan)

TOPI corresponds to selection of BSR refinement with full splitting.

Graphplan and SATPLAN can be seen as selecting FSR with no splitting

There are also planners such as UCPOP-D which we shall describe that can allow “medium” level of splitting

Issues in handling plansets without splitting

- ✣ Reduced commitment
- ✣ Separation of action selection and action sequencing
- – Unwieldy plan sets
  - » Use disjunctive representations
    - • Refining disjunctive plans
- – Transfer of complexity to solution-extraction
  - » Use efficient SAT solvers
  - » Use incremental constraint propagation

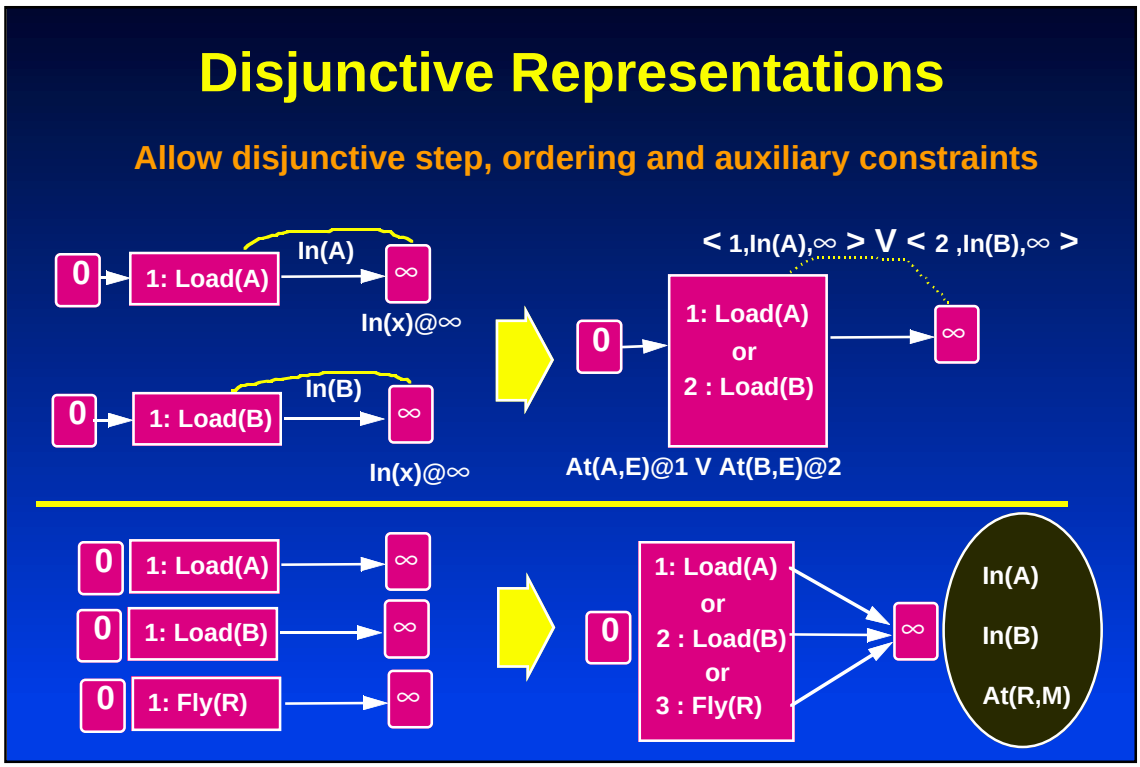
Although the framework we presented allows handling plan set components separately or together, most existing planners handle them separately. We shall now explicitly consider the issues involves in handling plan sets together without splitting.

So, let us look at what happens if we don't split plan sets. Of course, we reduce the search space explosion and avoid the premature commitment to specific plans. We also separate the action selection and action sequencing phases, so that we can apply scheduling techniques for the latter.

There can be two potential problems however. First off, keeping plan sets together may lead to very unwieldy data structures. The way to get around this is to "internalize" the disjunction in the plan sets so that we can represent them more compactly. The second potential problem is that we may just be transfering the complexity from one place to another -- from search space size to solution extraction. This may be true. However, there are two reasons to believe that we may still win.

First, as we mentioned earlier, solution extraction can be cast as a model-finding activity, and there have been a slew of very efficient search strategies for propositional model finding. Second, we may be able to do even better by using constraint propagation techniques that simplify plans and reduce refinement possibilities.

Let me illustrate these ideas.

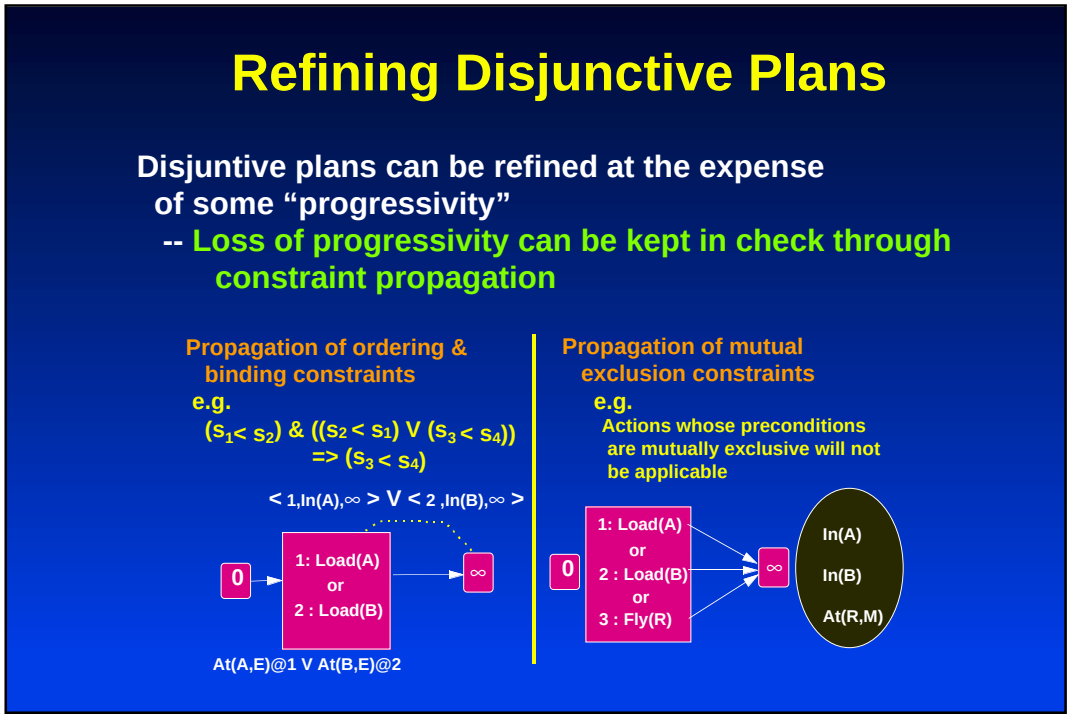


The general idea of disjunctive representations is to allow disjunctive step, ordering, and auxiliary constraints into the plan. Here are two examples that illustrate the compaction we can get through them.

The two plans on the top left corner can be compacted by using a single disjunctive step constraint, a disjunctive precedence constraint, a disjunctive IPC and a disjunctive PTC.

Similarly, the three plans in the bottom right can be compacted into a single disjunctive step, with disjunctive contiguity constraints.

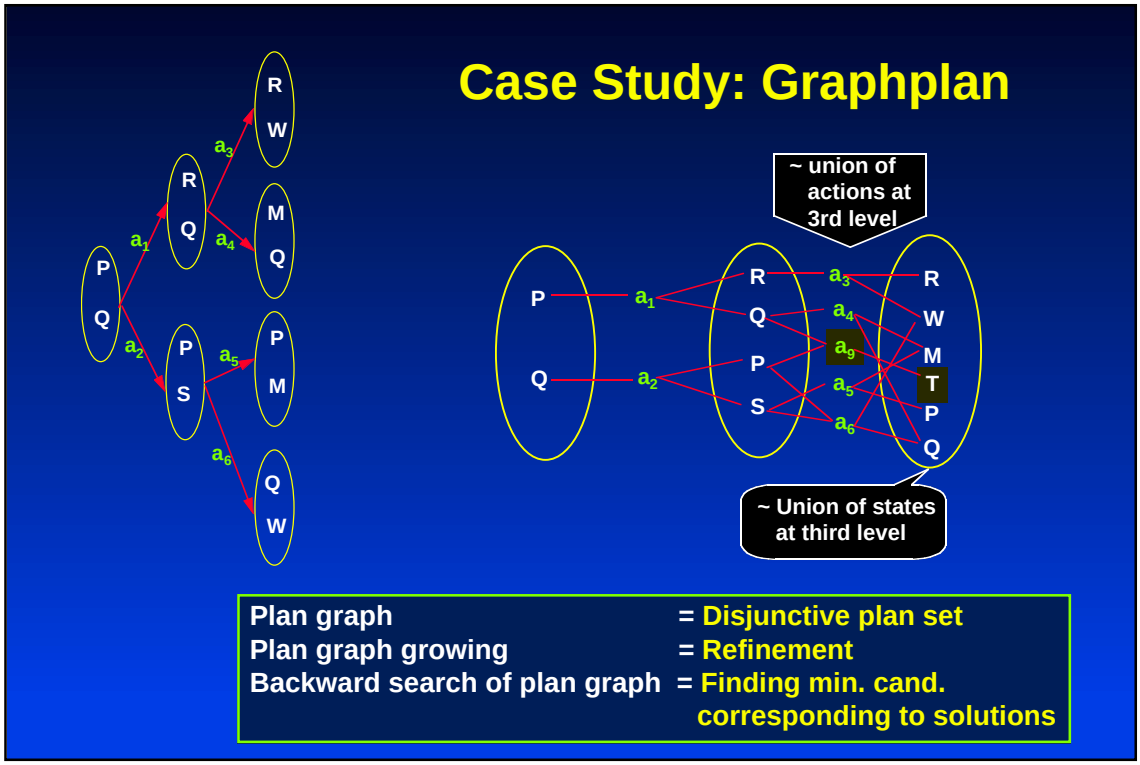
Candidate set semantics can be given naturally from the interpretation of disjunctive constraints.



Disjunctive representations necessitate some generalizations to the specifics of refinement strategies. Notice that the syntactic specifics of refinement strategies are described clearly only for partial plans without disjunction, and thus cannot be applied in a straightforward way to non-disjunctive ones. For example, in the left plan, we don't know whether steps 1 or 2 or both will be present in the eventual plan. Thus a PSR refinement won't know whether we should work on At(A,E) precondition or the At(B,E) precondition or both. Similarly, in the right hand side we don't know which of the steps will be coming next to 0 and thus we don't quite know what the state of the world will be after the disjunctive step. Thus, the FSR refinement will not know which actions are should be applied to the plan prefix next.

One way of refining such plans is to handle the uncertainity in a conservative fashion. For example, in the plan on the right, although we do not know the exact state after the first (disjunctive) step, we know that it can only be a subset of the union of literals in the effects of the three steps. We can thus consider a variant of FSR that adds only those actions whose preconditions are subsumed by the union of the effects of the three steps. *it is of course possible that even though the preconditions of an action are in the union of effects, there is no real way fo r that action to take place. For example, although the preconditions of "unload at moon" action may seem satisfied, it is actually never going occur as the second step in any solution because load and fly cannot be done at the same time.* ***This brings up an important issue-- disjunctive plans can be refined at the expese of some of the "progressivity" of the refinement.***

Although the loss of progressivity cannot be avoided, it can be reduced to a significant extent by doing constraint propagation along with refinements. For example, by marking the pair-wise exclusivity of the actions load / Fly in the first step, we can realize that the effects they give cannot both be true at the next level, and thus remove actions such as unload from consideration. {*BTW, the fact that Graphplan's efficiency depends on mutex propagation means that the ease of solution extraction depends on the progressivity of the refinement!*}



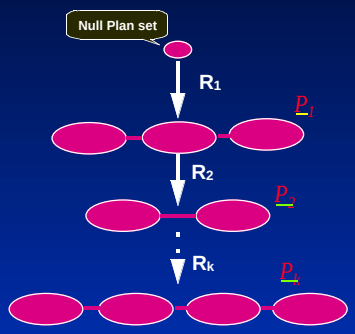
We noted earlier that Graphplan can be seen as using FSR refinements without splitting. We elaborate this relation now in light of our discussion of disjunctive representations.

On the left is a state tree generated by a forward state space planner that employs full splitting. On the right is the "plan graph" structure generated by Graphplan for the same problem(*) Note that plan graph can be seen roughly as a "disjunction" of the branches of the tree on the left.

Specifically, the ovals representing the "plan graph" proposition lists at a particular level can be seen as approximate union of the states in the state space tree at that level. Similarly, the actions at a given level in the plan graph canbe seen as the union of actions on the various transitions at that level in the state tree.

It is important to note that the relation is only approximate--for example the action and the proposition high ligted in brown at the third level do not have any correspondence with the search tree. This is part of the price we pay for doing refinements over disjunctive representations. However, the propagation of of mutual exclusion constraints allows Graphplan to keep as close a correspondence to the state-space search tree as possible.

Relation to Planning as Satisfiability



Is there a minimal candidate of P_k that is a solution to the problem?

Can be encoded as a SAT/CSP instance

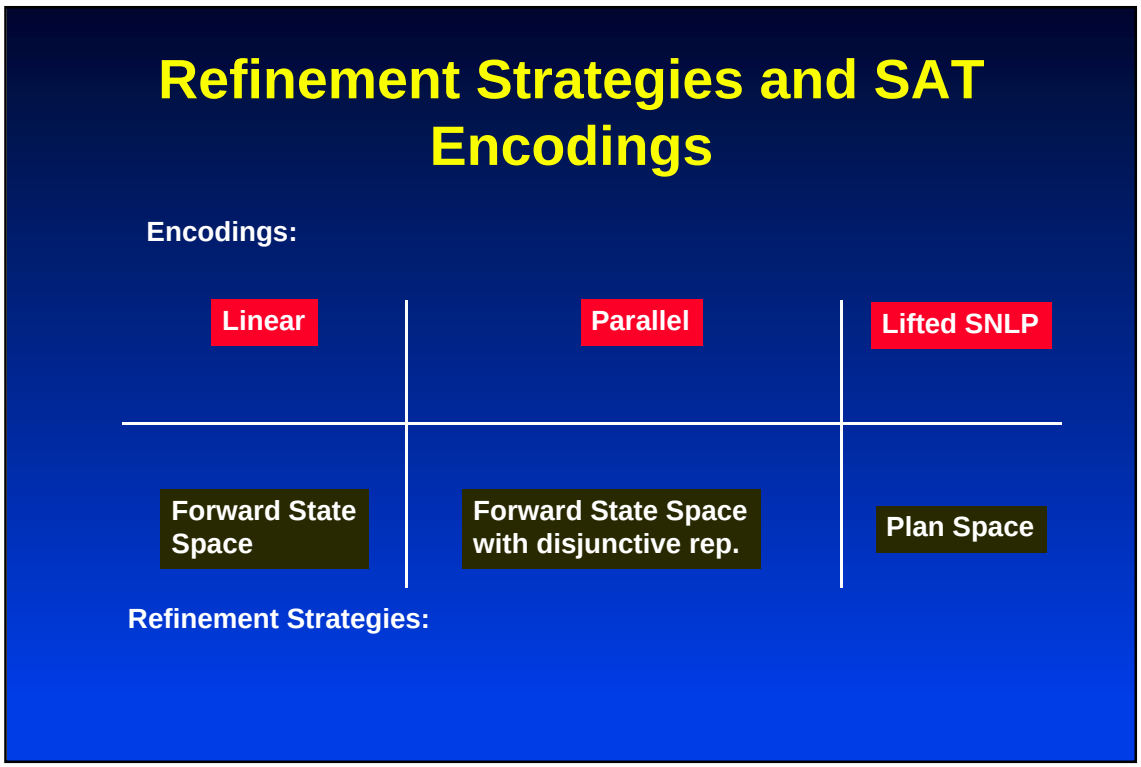
If R_1, R_2,...,R_k are all complete (and progressive)
Then,
Minimal candidates of P_k will contain all k-length solutions

Shape of the encoding depends on
-- Refinements R_i
-- Representations for plansets
-- Disjunctive/non-disjunctive

Recently, Kautz and Selman have advocated solving planning problems by encoding them first as SAT problems and then using efficient SAT solvers like GSAT to solve them. Their approach involves generating a a SAT encoding all models of which will correspond to k-length solutions to the problem (for some fixed k). Model-finding is done by the SAT solvers. They start with some arbitrary value of k, and increase it if they do not find solutions of that length. They have considered a variety of ways of generating the encodings--some described in their AAAI-96 paper, and some to be described in their paper here.

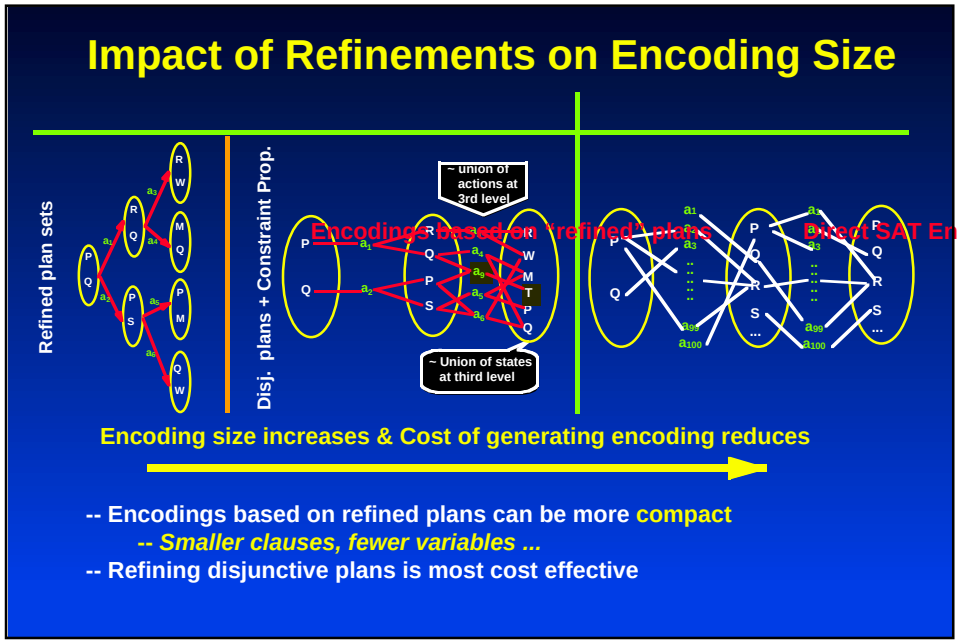
In the context of the general refinement planning framework, we can offer a rational basis on which the encodings can be generated. Specifically, the natural place where SAT solvers can be used is in the "solution extraction phase"-- specifically, after doing k "complete" and "progressive" refinements on a null plan, we get a plan set whose minimal candidates contain all k-length solutions to the problem. So, picking a solution boils down to searching through the minimal candidates-- which can be cast as a SAT problem.

This account naturally relates the character of the encodings to the type of refinements used in coming with the k-length plan-set and how the plansets themselves are represented (recall that disjunctive representations can reduce the progressivity of refinements).



Indeed, we can make rough comparisons between the different encodings explored by Kautz et. al., and the refinement strategies and representations of plan sets that could give rise to them. For example, linear and parallel encodings correspond to the use of forward state-space refinements, with the latter corresponding to disjunctive plan-set representations. Similarly, the SNLP encodings correspond to the use of plan-space refinements.

Given that interleaving refinement strategies has been shown to be a good idea in improving the cost of refinement planning, we can explore the utility of encodings based on interleaved refinements.



One question that needs to be answered is whether the relation between k-length encodings and the minimal candidates of k-level plan sets is just a matter of theoretical curiosity, or whether it has any practicial significance.

We believe that basing encodings on k-level plan sets, derived by the application of k complete refinements, leads to SAT instances that are “smaller” on the whole. Specifically, both the number of variables in the SAT as well as the size of the individual clauses sizes can reduce by starting from k-level plansets.

We illustrate this point with an example involving forward state space refinements. Here we have three different ways of generating encodings based on FSR. On the left, we do FSR refinements on individual componenets of the plan sets, generating all legal k-length prefixes (which can be searched to see if any of them correspond to a solution). On the right handside, we avoid refinements and generated the encoding directly using the methods used by Kautz et. al. The middle picture corresponds to doing FSR on the disjunctive plan representations (specifically, the structure here is similar to the k-level plan graph-- which can be seen as the disjunctive representation of k-level planset). It is interesting to note that as we go from left to right, the amount of uncertainity increases. For example, if we ask the question--what can be the actions at level 2, the left most encoding will say they can only be one of 5. The right most one says they can be one of any available actions, while the middle one says that they can be one of six.

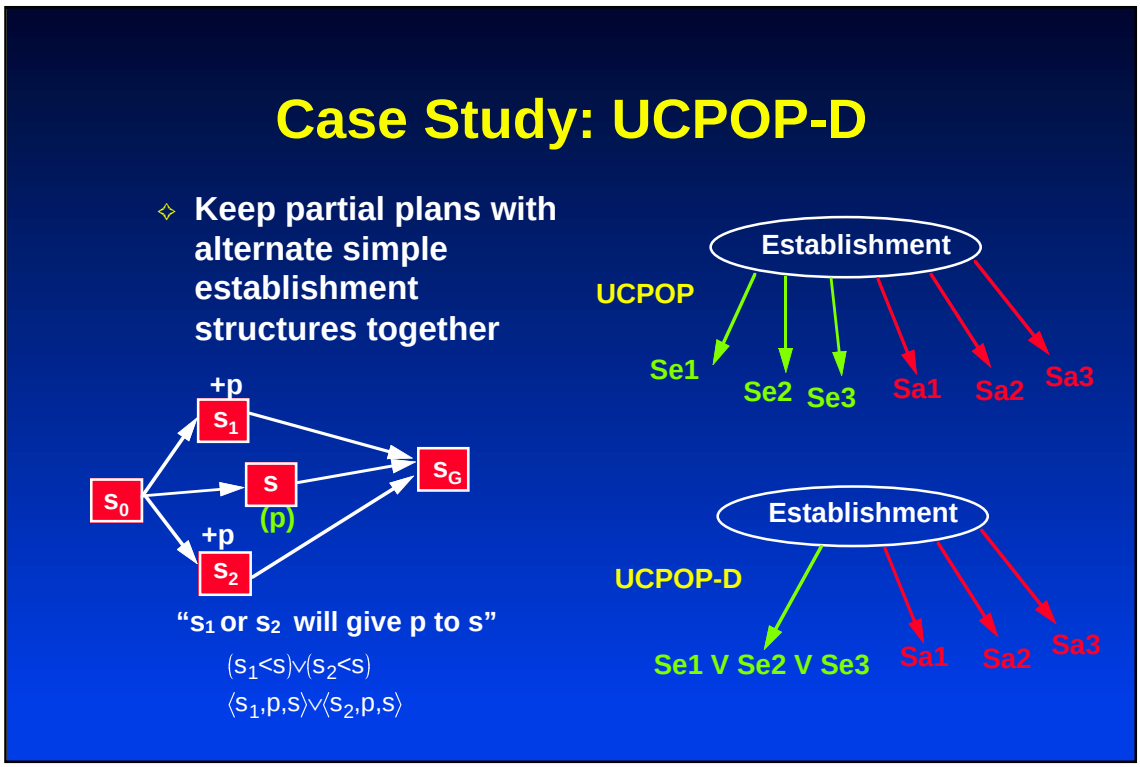
Clearly, the encodings sizes will be largest for the left and smallest for the right. At the same time, the cost of generating the encoding is lowest on right and highest on left. Thus, a happy medium is likely to be reached in the middle--ie. encodings based on disjunctive refined plans.

In addition to explining the relation between Graphplan, SATPLAn approaches and the traditional planning approaches, our treatment brings into high relief the tradeoffs involved in handling plansets without partitioning.

For example, until now, the planners we saw do either no partitioning or full partitioning. research in constraint satisfaction literature shows that propagation and refinement can have synergistic interactions. (*A case in point is 8-queens problem*. .) This raises the possibility that best planners may be doing controlled splitting of plan sets (rather than no splitting at all) to facilitate further constraint propagation. (*Extent of propagation could depend on the amount of shared sub-structure between the disjoined plans.)*

Another issue is the relative support provided by various types of refinements for planning with disjunctive representations . The old analyses based on least commitment etc. are mostly inadequate when we don't split plan set components.

Next, we will discuss a variant of UCPOP called UCPOP-D that provides a reference point for planning with limited partitioning. UCPOP-D uses the ideas of disjunctive representations and constraint propagation.



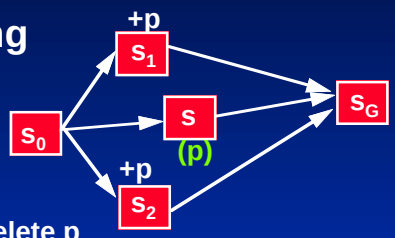
We will now discuss a variant of UCPOP planner which uses the ideas of disjunctive representation and constraint propagation to improve its performance.

Unlike UCPOP which does full splitting, UCPOP-D keeps the plans that differ only in simple establishment possibilities together. Specifically, consider the plan on left. In establishing p@S, UCPOP will consider two different plans corresponding to simple establishments with s1 or s2. UCPOP-D keeps these plans together by maintaining disjunctive IPCs (causal links). {*This can be seen as a generalization of the multi-contributor causal links idea*}.

Thus, instead of several simple establishment branches, of UCPOP (as shown on top right) UCPOP-D will have only one simple establishment branch (as shown on bottom right)

Plan-space refinements in the presence of disjunctive causal commitments

Involves maintaining and propagating disjunctive orderings.



$\langle s_1,p,s\rangle \vee \langle s_2,p,s\rangle$

Implies for every step t that can delete p

$$\left\{\begin{array}{l}(t<s)\vee\\ [(t<s_1)\wedge(s_1<s)]\vee\\ [(t<s_2)\wedge(s_2<s)]\end{array}\right\}$$

Maintaining consistency of disjunctive orderings is NP-complete

-- Constraint propagation can help

$$(s_1<s_2)\wedge[(s_2<s_1)\vee(s_3<s_4)] \Rightarrow (s_3<s_4)$$

Supporting plan-space refinement on plans containing disjunctive IPCs involves maintaining and propagating disjunctive ordering constraints.

Specifically, the precense of the disjunctive IPC saying that C is given by either S1 or S2 implies that (a) s1 or s2 must precede s and (b) for every step t that can delete p, either we must have t precede s or, t precede s1 and s1 precede s or t precede s2 and s2 precede s.

Although it is hard to exploit disjunctive orderings to reduce control plan-space refinement, we can improve the situation by simplifying them through constraint propagation whenever a new ordering is added to the plan.

[The slide describing the comparison between UCPOP and UCPOP-D and emonstrating the superiority of the latter goes here. See the paper]

Summary

- A general refinement planning framework that teases apart the notions of "refinement" and "search"
  - Explicates the relations between traditional planners and Graphplah/SATPLAN approaches
  - Foregrounds the issues involved in managing search and solution-extraction processes
    - » Disjunctive representations and constraint propagation help in managing large plansets.
    - » Existence of planners like UCPOP-D that combine refinement search and CSP methods

By teasing apart the hitherto closely intertwined notions of "refinement" and "search" we were able to present a model of refinement planning that not only includes the traiditional planners, but also supports a large variety of planners that transfer seach to solution extraction phase by handling sets of partial plans together.

Disjunctive representations and constraint propagation techniques facilitate efficient management of large plansets.

Through this, we explicated the rich relations between the traditional refinement planners and the newer planners such as Graphplan and Satplan.

Although the existing planners fall at the extremes of the search vs. solution extraction cost tradeoff, our framework suggests that planners in the middle might also provide better computational tradeoffs. As an example, we discussed a variant of UCPOP called UCPOP-D that reduces search by handling plans with differing "simple establishment structures" together.

Future Directions

- Empirical and analytical exploration of encodings based on various refinements
  - Task Reduction Refinements
  - Plan sets generated by interleaving multiple refinements
- Exploring the tradeoffs between search and solution extraction
  - Keep plan set components with shared structure together

Our work also opens up a variety of avenues for further research. Given the rich relations between refinements and encodings, it is worth investigating encodings based on other refinements such as task reduction refinements, which have enjoyed significant popularity in refinement planning. [While we are at it, the work to be presented by Ginsberg can for example be seen as using a differnt partial plan representation and refinement strategy.]

Similarly, some of our recent work shows that interleaving refinement strategies, rather than using a single refinement strategy all the time, could lead to better performance in traditional refinement planners. It is worth investigating encodings based on interleaving of multiple refinements.

Another importnat area of research will be understanding the tradeoffs involved in trading off search and solution extraction or vice versal. Specifically, it is worth understanding the utility of planners which employ a controlled partitioning to keep plan set components with shared sub-structure togehter.