Part IV

Plan Space Planning

Outline of Part IV

Introduction

- The search space is no more a states space but a plans space.
  - Nodes are partially specified plans.
  - Arcs are plan refinement operations intended to further complete a partial plan, i.e., to achieve an open goal or to remove a possible inconsistency.
- Solution plan definition changes. Planning is considered as two separate operations:
  1. the choice of actions
  2. the ordering of the chosen actions so to achieve the goal.

I. Plans Space Search

Plan Space Example

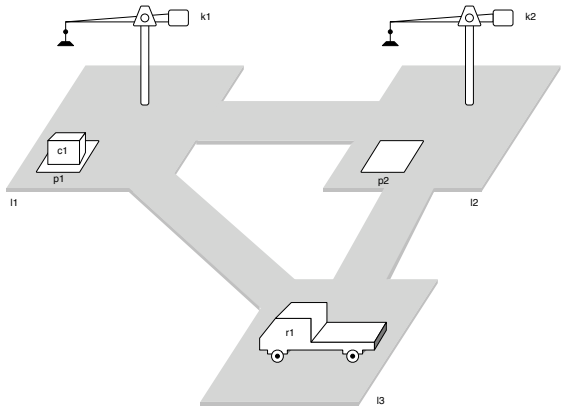


Figure 6: A robot r1 has to move a container c1 from pile p1 at location l1 to pile p2 and location l2. Initially r1 is unloaded at location l3. There are empty cranes k1 and k2 at locations l1 and l2. Pile p1 at location l1 contains only container c1; pile p2 at location l2 is empty. All location are adjacent.

Plan Space Example

Consider we have a partial plan that contains only the two following actions

- take(k1,c1,p1,l1): crane k1 picks up container c1 from pile 1 at location l1
- load(k1,c1,r1,l1): crane k1 loads container c1 on robot r1 at location l1

Let us refine it by adding a new action and let us analyse how the partial plan should be updated. We will come up with four ingredients:

1. adding actions
2. adding ordering constraints
3. adding causal relationship
4. adding variable binding constraints

Adding Actions Example

Nothing in this partial plan guarantees that robot r1 is already at location l1. Proposition at(r1,l1), required as a precondition by action load, is a subgoal in this partial plan. We need to add the following action:

- move(r1,l,l1): robot r1 moves from location l to the required location l1.

Adding Ordering Constraints

This additional move action achieves its purpose only if it is constrained to come before the load action. But should the move action come before or after the take action? Both are possible.

Least Commitment principle

Not add a constraint to a partial plan unless it is strictly needed. May permit to run actions concurrently.

Adding Causal Links

In partial plan, we have added one action and an ordering constraint to another action already in the plan. Is that enough? No quite. Because

there is no explicit notion of a current state (e.g., an ordering constraint does not say that the robot stays at location l1 until load action is performed). Hence, we will be encoding explicitly in the partial plan the reason why action move was added: to satisfy the subgoal at(r1,l1) required by action load.

This relationship between the two actions move and load with respect to proposition at(r1,l1), is called a *causal link*.

Note

The former action is called the *provider* of the proposition, the later the *consumer*. The role of a causal link is to state that a precondition is *supported* by another.

Adding Variable Binding Constraints

A final item in the partial plan that goes with refinement we are considering is that variable binding constraints.

- Operators are added in the partial plan with systematic variables renaming.
- We should make sure that the new operator move concerns the same robot r1 and the same location l1 as those in the operator take and load.
- What about l the robot will be come from? At this stage there is no reason to bind this variable to a constant. The variable l is kept unbounded.

Partial Plan Definition

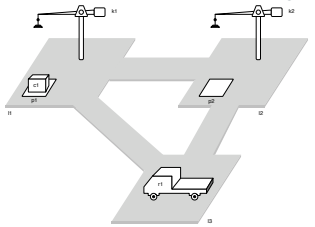
Definition (Partial Plan)

A *partial plan* is a tuple $= (A, \prec, B, L)$ where:

- $A = \{a_1, \ldots, a_k\}$ is a set of partially instantiated planning operators.
- $\prec$ is a set of ordering constraints on A of the form $(a_i \prec a_j)$.
- B is a set of binding constraints on the variables of actions in A of the form $x = y$, $x \neq y$, or $x \in D_x$, D_x being a subset of the domain of x.
- L is a set of causal links of the form $\langle a_i \xrightarrow{p} a_j \rangle$, such that a_i and a_j are actions in A, the constraint $(a_i \prec a_j)$ is in $\prec$, proposition p is an effect of a_i and a precondition of a_j, and the binding constraints for variables of a_i and a_j appearing in p are in B.

Partial Plan Example

- Let us illustrate two partial plans corresponding to



- The goal of having container c1 in pile p2 can be expressed simply as in(c1,p2).
- The initial state is:
  - { adjacent(l1,l2), adjacent(l1,l3), adjacent(l2,l3), adjacent(l2,l3), adjacent(l3,l1), adjacent(l3,l2), attached(p1,l1), attached(p2,l2), belong(k1,l1), belong(k2,l2), empty((k1), empty(k2), at(r1,l3), unloaded(r1), occupied(l3), in(c1,p1), on(c1,pallet), top(c1,p1), top(pallet,p2) }

Partial Plan Example

A graphical representation of the initial plan π_0 is shown in figure 102.
Each box is an action preconditions above and effects below the box.
Solid arrows are ordering constraints; dashed arrows are causal links; and
binding constraint are implicit or shown directly in the arguments.

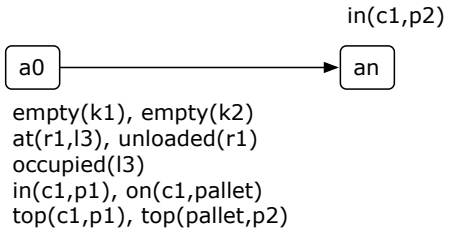
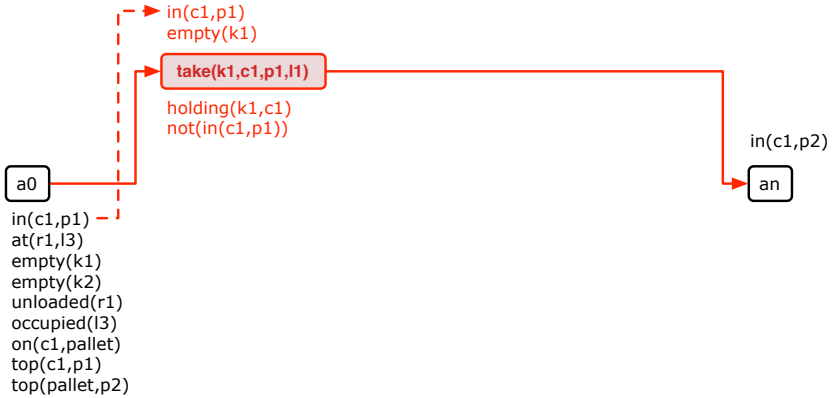
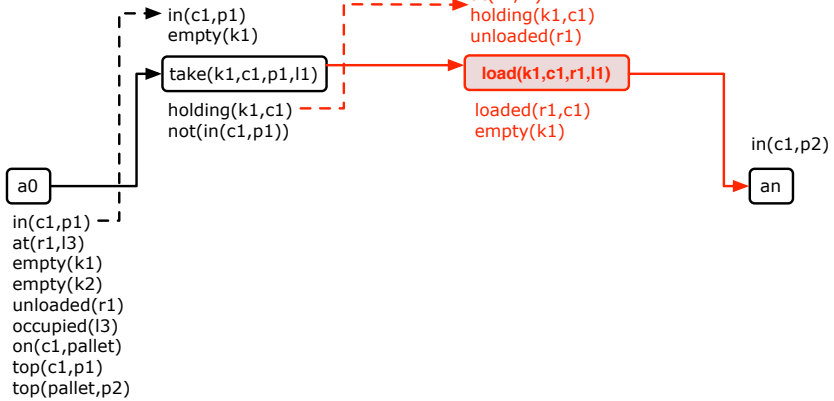


Figure 7: Initial plan π_0.

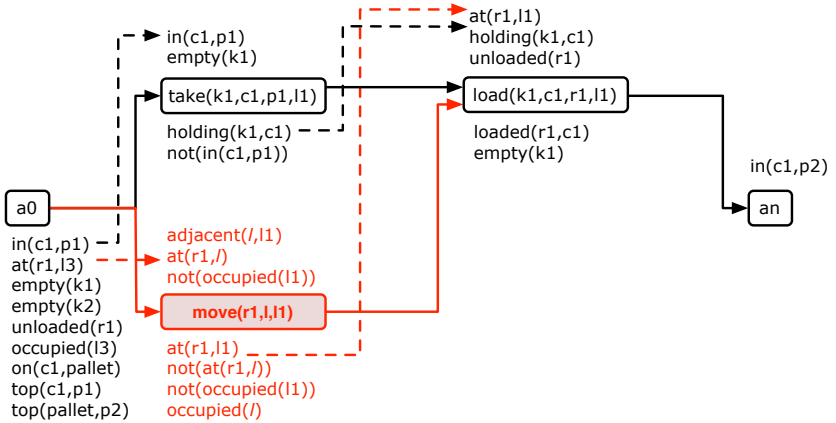
Partial Plan Example



Partial Plan Example



Partial Plan Example



To summurize

- A partial plan is a structured collection of actions that provides the causal relationships for its actions, as well intrinsic ordering and variable bindings
- A partial plan provides subgoals as preconditions without causal links
- A plan-space planner refines a partial plan by adding actions, ordering contraints, binding constraints or causal links
- It is convenient to see a partial plan as set of plans

II. Solution Plans in Plan-Space

Solution Plan Definition

To define planning algorithm we have to define formally what is a solution plan in plan-space.

Definition (Solution Plan)

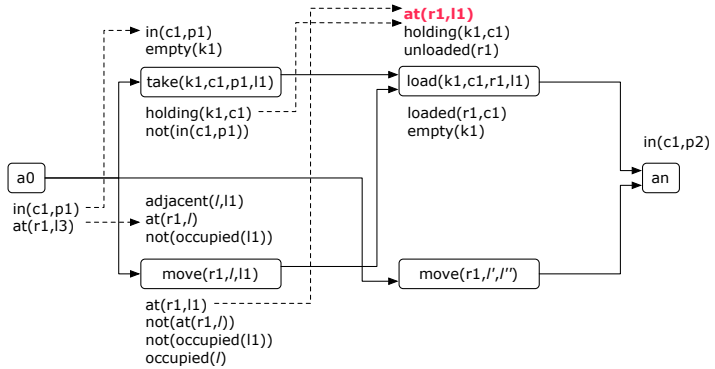
A partial plan $\pi = (A, \prec, B, L)$ is a *solution plan* for a problem $P = (\Sigma, s_0, g)$ if:

- its ordering constraints $\prec$ and binding constraints B are consistent.
- every sequence of totally ordered and totally instantiated actions of A satisfying $\prec$.
- B is a sequence that defines a path in the state transition system Σ from the initial state s_0 corresponding to effects of the action a_0 to state containing all goal proposition in g given by preconditions of a_n.

Problem

- this definition does not provide a computable test for verifying plans
- It is not feasible to check all instantiated sequences of actions of A.

Example: Plan with incorrect sequence



at(r1,l1) is not satisfied in the state preceding load due to action move(r,l',l'') in the case where $l' = l1$ for instance

Flaw and Threat

To propose a computable definition we need to introduce two notions:

Definition (Threat)

An action a_k in a plan π is a *threat* on a causal link $(a_i \xrightarrow{p} a_j)$ iff:

- a_k has an effect $\neg q$ that is possible inconsistent with p.
- the ordering constraints $(a_i \prec a_k)$ and $(a_k \prec a_j =$ are consistent with B.
- the binding constraints from the unification of q and p are consistent with B.

Definition (Flaw)

A flaw in a plan $\pi = (A, \prec, B, L)$ is either:

- a subgoal, i.e., a precondition of an action in A with out a causal link
- a threat, i.e., an action that may interfere with causal link.

Solution Plan Proposition

Proposition (Solution Plan)

A partial plan $\pi = (A, \prec, B, L)$ is a *solution plan* for a problem $P = (\Sigma, s_0, g)$ if:

- π has no flow and
- the set of ordering contraints $\prec$ is consistent and
- the set of binding constraints B is consistent.

- This proposition can be prove by induction on the number of actions in A.

Example: Solution Plan

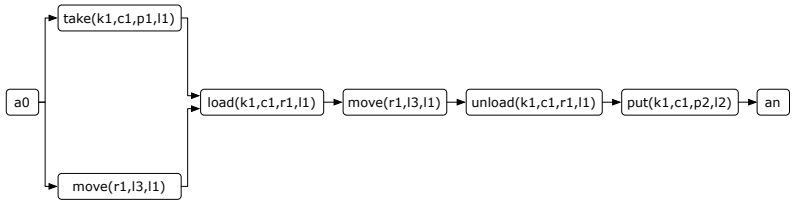
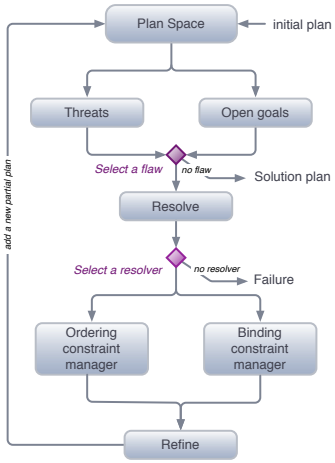


Figure 8: A solution plan

III. Algorithms for Plan Space Planning

PSP procedure

- PSP procedure is a generic and simple planning alpgorithm to plan in plan-space
- A plan π is a solution when it has no flaw, the main principle is to refine π, while maintaining $\prec$ and B consistent, until it has no flaw. The basic operations for refining a partial plan π toward a solution plan are the following:
  - Find the flaws of π, i.e., its subgoals and its threats.
  - Select on such flaw.
  - Find ways to resolve it.
  - Choose a resolver for the flaw.
  - Refine π according to that resolver.



PSP Algorithm

Algorithm (PSP(π))

flaws $\leftarrow$ `OpenGoals`(π) $\cup$ `Threat`(π)
if flaws $= \emptyset$ **then return** π

select any flaw sigma $\in$ flaws
resolvers $\leftarrow$ `Resolve`(σ,π)
if resolvers $= \emptyset$ **then return** Failure

nondeterministically choose a resolver $\rho \in$ resolvers
$\pi' \leftarrow$ `Refine`(ρ, π)
return `PSP`(π')

Attached Procedures

OpenGoals(π). This procedure find all subgoals in π.

Threat(π). This procedure find every action a_k that is a threat on some causal link $(a_i \xrightarrow{p} a_j)$.

Resolve(σ, π). This procedure finds all ways to solve a flaw σ.

Refine(ρ, π). This procedure refines the partial plan π with le elements in the resolver, adding to π on ordering constraint, on or several binding constraints, a causal link, and/or a new action.

Important remarks

- Even with the restrictive assumption of finit transition system the plan space is not finite
- A deterministic implementation of the PSP procedure will maintain the completness only if it garantees to explore all partial plans, up to a some length
- It is necessary to use search strategy such as IDA* or A*.

PoP Algorithm

Algorithm (PoP(π, agenda))

if agenda $= \emptyset$ **then return** π
select any pair (a_j, p) *in and remove it from* agenda
relevant $\leftarrow$ `Providers(`p`,`π`)`
if relevant $= \emptyset$ **then return** failure
nondeterministically choose an action $a_i \in$ relevant
$L \leftarrow L \cup \{\langle a_i \xrightarrow{p} a_j \rangle\}$
update B *with the bindig constraints of this causal link*
if a_i *is a new action in* A **then**
  update A *with* a_i
  update $\prec$ *with* $(a_i \prec a_j)$, $(a_0 \prec a_i \prec a_\infty)$
  update agenda *with all preconditions of* a_i
foreach *threat on* $\langle a_i \xrightarrow{p} a_j \rangle$ *or due to* a_i **do**
  resolvers $\leftarrow$ *set of resolvers for this threat*
  if resolvers $= \emptyset$ **then return** failure
  nondeterministically choose a resolvers
  add that resolver to $\prec$ *or to* B
return `PoP(`π, agenda`)`

PoP Algorithm

- PoP algorithm is a simplified version of a planner called UCPOP.
- The main difference between PSP and PoP is that PSP processes the two types of flaws in a similar way
  - at each recusion, it selects heuristically a flaw from any type to pursue the refinement
- PoP has a distinct control for subgoals and for threart.

Plan-Sapce vs. State-Space Planning

Plan-Sapce vs. State-Space Planning

- The state space is finite, while the plan space is not
- Intermediate states are explicit in state space, while there is no explicit state in plan space
- A partial plan separates the choice of the actions that need to be done from how to organize them into a plan, e.g, just keep the ordering constraints over the chosen actions
- The plan structure an the rationale for plan's component are explicit in plan space (causal link)
- Structure use in plan space are more complex and refinements take significantly more time to compute

Avantages of Plan-Space Planners

- They build partially ordered and partially instantiated plans that are more explicite and flexible for execution than plans created by state space planners
- They provide an open planning approach for handling severals extensions to classical planning, such as time, resources, etc.
- They allow distributed planning and multi-agent planning t be adressed to due plan structure and no explicite state representation.

To go further

Exercice

1. Trace the PSP procedure step-by-step on the Sussman anomaly introduce in chapter Plan Space Planning.
2. Draw the complete graph to compute the solution plan show in the slide "Example of Solution Plan":
   - How many threats are there ?
   - How many plans can be found ?

Further readings


E. Sacerdoti
Planning in a hierarchy of abstraction spaces.
Artificial Intelligence 5:115-135, 1974


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