Automated Planning

Theory and practice

Damien Pellier
Damien.Pellier@imag.fr
www-leibniz.imag.fr/~pellier

Master I Informatique
UFR d'Informatique et de Mathématiques Appliquées
Université Joseph Fourier

April 25, 2005

Part I

Introduction and Overview

Outline of Introduction

Intuitive Planning Definition

What is Planning ?

- Planning is the reasonning side of acting. It is an abstract, explicite deliberation process that chooses and organizes action by anticipating their outcomes.
- This deliberation aims at achieving as best as possible some pretated objectives.
- Automated planning is an area of Artificial Intelligence (AI) that studies this deliberation process computationally.

Automated Planning Motivations

1. *Practical Motivations:* Designing information processing tools that give access to affordable and efficient planning ressources.

Example

Imagine a rescue operation after a natural disaster.

- That operation involves a large number of actors and require transportation infrastuctures.
- It relies on careful planning and assessment of several alternate plans.
- It is also time constrainted and it demands immediate decisions that must be supported with a planning tool.

2. *Theorical Motivations:* Planning is an important component of rational behavior.

Path and Motion Planning

Path and Motion Planning is concerned with the synthesis of a geometric path from a starting position in space to a goal and a control trajectory along that path that specifies the state variables in the configuration space of mobile systems, such as a truck, a mechanical arm, a robot, *etc.*

Motion planning takes into account:

- the model of the environment
- the kinematic constraints
- the dynamic contraints

Perception Planning

Perception Planning is concerned with plans involving senseing actions for gathering informations. It arises in tasks such as modeling environements or objects, identifying objects, localizing through sensing a mobile system, or more generally identifying the current state of the environment.

- Perception planning addresses question such as information needed and when it is needed, where to look for it, which sensors are most adequate for a particular task and how to use them.
- It relies on decision theory for problems of which and when information is needed, on mathematical programming and constraint satisfaction for viewpoint selection and the sensor modalities.

Navigation Planning

Navigation Planning combines the two previous problems of motion and perception planning in order to reach a goal or to explore an area. the purpurse of navigation planning is to synthetize a policy that combines localization primitives and sensor-based motion primitives.

Example

- visually following a road until reaching some landmark
- moving along some heading while avoiding obstacles
- *etc.*

Manipulation Planning

Manipulation Planning is concerned with handling objects, *e.g.*, to build assemblies.

- The actions include sensory information.
- A plan might involve picking up an object from its marked sides, returning it if needed, inserting it into an assembly, and pushing lightly till it clips mechanically into position.

Manipulation Planning

Manipulation Planning arises in dialog and in cooperation problems between several agents, human or artificial. It addresses issues such as when and how to query needed information and which feedback should be provided.

Domain Specific Approaches

Domain specific approaches to specific forms of planning are certainly well justified. However, they are frustrating for several reasons.

1. Some commonalities to all these forms of planning are not adressed in the domain specific approaches. The study of these commonalities is needed for understanding the process of planning.
2. It is more costly to adress each planning problem anew instead of relying on and adapting some general tools.
3. Domain specific approaches are not satisfactory for studying and designing an autonomous intelligent machine. Its deliberative capabilities will be limited to areas for which it has a domain specific planner.

Domain Independent Approaches

Domain independent approaches relies on abstract, general models of actions. These models range from simple ones that allow only for limited forms of reasoning to models with richer prediction capabilities. There are in particular the following forms of models and planning capabilities.

1. *Project Planning* in which models of actions are reduced mainly to temporal and precedence constraints, *e.g.*, the earliest and the latest start times of an action or its latency with respect to another action. Project planning is used for interactive plan edition and verification.
2. *Scheduling and resources allocation* in which the action models include the above types of constraints plus constraints on the resources to be used by each action.
3. *Plan synthesis* in which the action models enrich the precedent models with the conditions needed for applicability of an action and the effects of the action on the state of the world.

State Transition System

The conceptual model of planning can be represented as a *state transition system*. Formally, a state transition system is a 4-tuple $\Sigma = (S, A, E, \gamma)$, where:

- $S = \{s_1, s_2, \ldots, s_n\}$ is a finite or recursively enumerable set of states
- $A = \{a_1, a_2, \ldots, a_n\}$ is a finite or recursively enumerable set of actions
- $E = \{e_1, e_2, \ldots, e_n\}$ is a finite or recursively enumerable set of events
- $\gamma : S \times A \times E \rightarrow 2^S$ is a state transition function

A state transition system may be represented by a directed graph whose nodes are the state in S

Planning Objectives

Given a state transition Σ, the purpose planning is to find which actions to apply to which states in order to achieve some objective when starting from a given situation. A *plan* is a structure that gives the appropriate actions. The objective can be specified in several different ways.

1. The simplest specification consists of a *goals state* s_g or a set of goal states S_g. In this case, the objective is achieved by any sequence of state transition that ends at one of the goal states.
2. The objective can be also expressed by the satisfaction of some conditions over the sequence of state followed by the system.
3. The objective can be expressed by an utility function attached to each states, with penalties and rewards. The goal is to optimize some compound function of these utilities.
4. The objective can be expressed as a tasks that the system should perform.

Graphical Representation of Planning Model

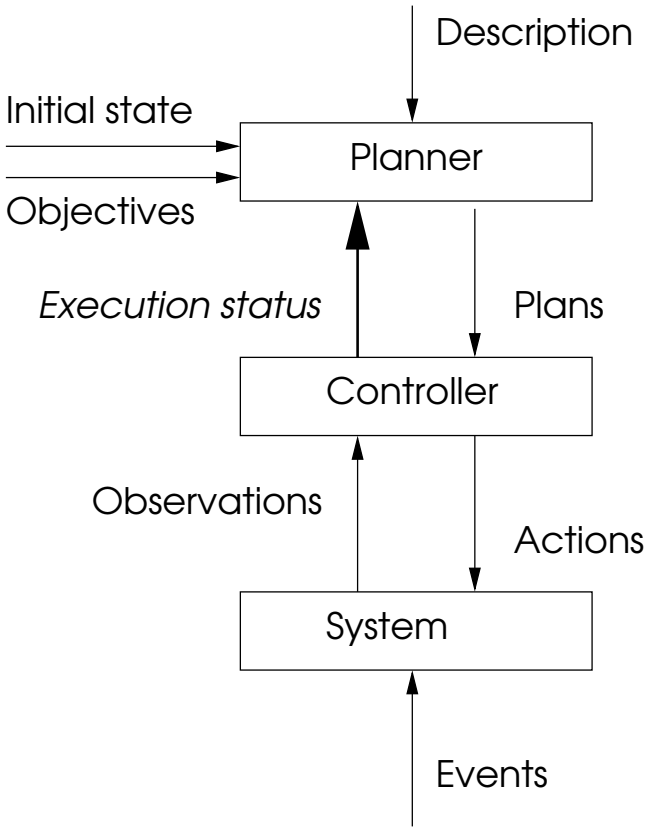

Figure: Conceptual model of planning

It is convenient to depict conceptual planning model through the interaction between three components:

- A *state transition system* Σ evolves as specified by its state transition function γ, according to the events and actions that it receives.
- A *controller*, given as input the state s of the system, provides as output an action a according to some plan.
- A *planner*, given as input a description of the system Σ, an initial situation, and some objective, synthesizes a plan for the controller in order to achieve the objectives.

Crane and robot transportation example I

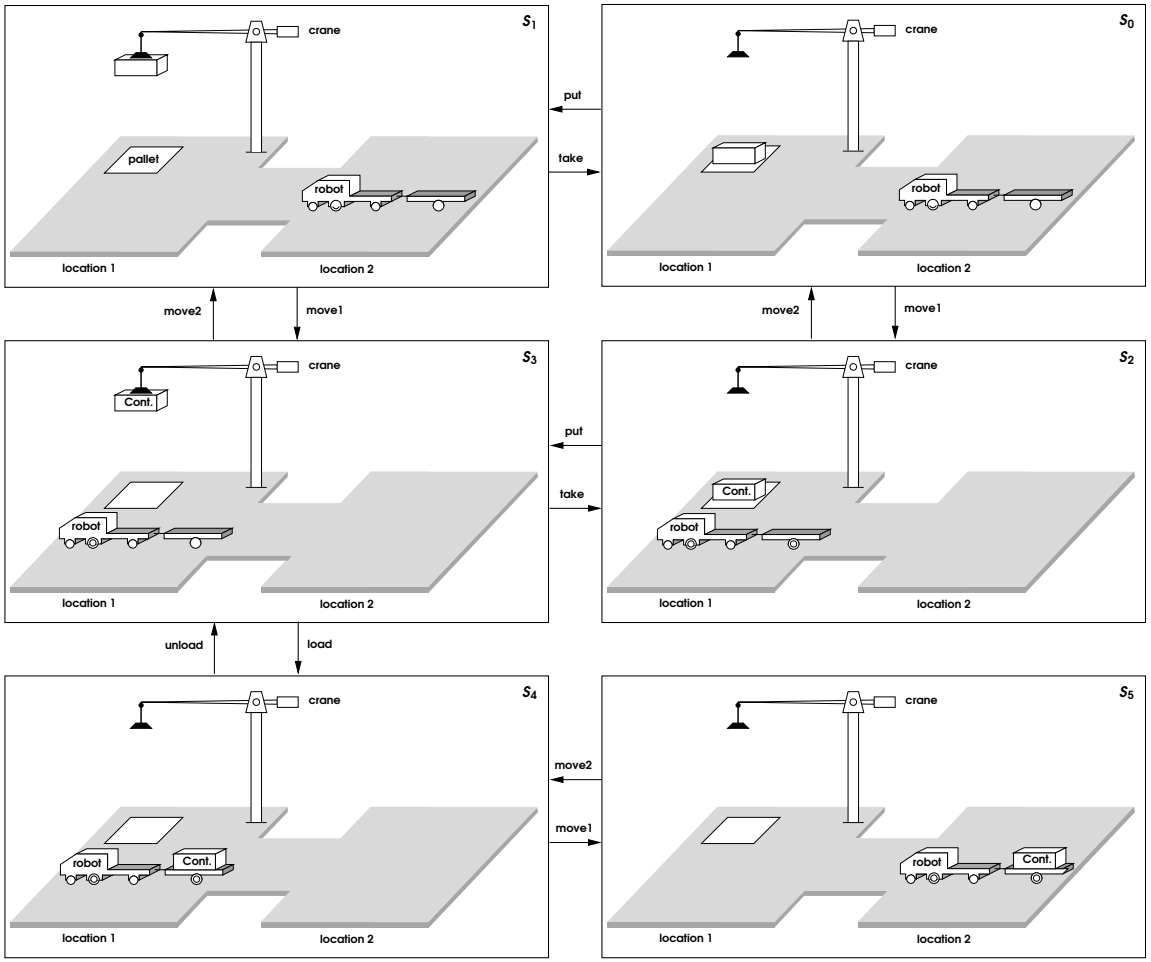

Figure: shows a state transition system involving a container in a pile, a crane that pick up and put down the container and a robot that can carry the container and move it from one location to another.

Crane and robot transportation example II

In this example:

- the set of states is $S = \{s_0, s_1, s_2, s_3, s_4, s_5\}$
- the set of actions is $A = \{take, put, load, unload, move1, move2\}$
- the set of events E is empty
- the transition function γ is defined by: if a is an action and $\gamma(s, a)$ is not empty then a is *applicable* to state s

Restricted Model

Planning model puts forward various restrictive assumptions, particularly the following ones.

- *Finite* Σ. The system Σ has a finite set of states.
- *Fully Observable* Σ. The system Σ is *fully observable*, *i.e.,* one has complete knowledge about the state of Σ.
- *Deterministic* Σ. The system Σ is *deterministic*, *i.e.,* , for every states s and for every event of action u, $|\gamma(s, u)| \leq 1$. If an action is applicable to a state, its application brings a deterministic system to a single other state.
- *Static* Σ. The system Σ is *static*, *i.e.,* the set of event E is empty. Σ has no internal dynamics.

Restricted Model

- *Restricted Goals*. The planner handles only *restricted goals* that are specified as an explicit goal state s_g or a set of goal states S_g.
- *Sequential Plans*. A solution plan to a planning problem is a linearly ordered finite sequence of actions.
- *Implicit time*. Actions and events have no duration. They are instantaneous, state transitions. This assumptions is embedded in state transition systems, a model that does not represent time explicitly.
- *Offline Planning*. The planner is not concerned with any change that may occur in Σ while it is planning. It plans for a given initial and goal states regardless of the current dynamics, if any.

Part II

Classical Representation for Planning

Outline of Part II

Introduction

We discuss three different ways to represent classical planning problems. Each of them is equivalent in expressive power.

- *Set theoric representation*, each state of the world is a set of propositions and each action is a syntactic expression specifying which propositions belong to the state in order for the action to be applicable and which propositions the action will add or remove to change the state of the world.
- *Classical representation*, the states and the actions are like the ones described for set theoric representation except that first order literals and logical connectives are used instead propositions.
- *State variable representation*, each state is represented by a tuple of value n state variables $\{x_1, \ldots, x_n\}$ and each action is represented by a partial function that map this tuple into some other tuple of values of the n states.

Planning Domains, Problems and Solutions

A *set theoric representation* relies on a finite set of proposition symbols that are intented to represent various propositions about the world. We need to define the basic notion of

- Planning Domain
- Planning Problem
- Planning Solution

Planning Domains Definition

Definition (Set Theoric Planning Domain)

Let $L = \{p_1, \ldots, p_n\}$ be a finite set of *proposition symbols*. A *set theoric planning domain* on L is a restricted state transition system $\Sigma = (S, A, \gamma)$ such that:

- $S \subseteq 2^L$, *i.e.,* each state s is a subset of L. If $p \in s$ then p holds in s. Otherwise p does not hold in s (Closed World Assumption).
- Each action $a \in A$ is a triple of subset of L written $a = (\text{precond}(a), \text{effect}^-(a), \text{effect}^+(a))$ and $\text{effect}^-(a)$ $\text{effect}^+(a)$ are disjoint.
- S has the property that if $s \in S$, then, for every action a that is applicable to s, the set $(s$ - $\text{effect}^-(\text{a})) \cup \text{effect}^+(a) \in S$.
- The state transition function is $\gamma(s, a) = (s$ - $\text{effect}^-(\text{a})) \cup \text{effect}^+(a)$ if $a \in A$ is applicable to $s \in S$.

Planning Problem Definition

Definition (Set Theoric Planning Problems)

A *set theoric planning problem* is a triple $\mathcal{P} = (\Sigma, s_0, g)$ where

- s_0, the *initial state*, is a member of S
- $g \subseteq L$ is a set of propositions called *goal propositions* that give the requirements that a state must be satisfy in order to be a goal state. The set of goal states is $S_g = \{s \in S \mid g \subseteq s\}$.

Planning Problem Example

Here is one possible set theoric representation of the domain described in figure 2.

Example (Set of propositions)

$L = \{$onground, onrobot, holding,at1,at2$\}$ where

- onground means that the container is on the ground
- onrobot means that the container is on the robot
- holding means that the crane is holding the container
- at1 means that the robot is at location1
- at2 means that the robot is at location2

Planning Problem Example

Example (Set of states)

$S = \{s_0, \ldots, s_5\}$ where

- $s_0 = \{onground, at2\}$; $s_1 = \{holding, at2\}$; $s_2 = \{onground, at1\}$
- $s_3 = \{holding, at1\}$; $s_4 = \{onrobot, at1\}$; $s_5 = \{onrobot, at2\}$

Example (Set of actions)

$A = \{$take, put, load, unload, move1, move2$\}$ where

- take = ({onground}, {onground}, {holding})
- put = ({holding}, {holding}, {onground})
- load = ({holding, at1}, {holding}, {onrobot})
- unload = ({onrobot, at1}, {onrobot}, {holding})
- move1 = ({at2}, {at2}, {at1})
- move2 = ({at1}, {at1}, {at2})

Plan Definition

Definition (Plan)

A *plan* is any sequence of action $\pi = \langle a_1, \ldots, a_k \rangle$, where $k \geq 0$. The *length* of the plan $|\pi| = k$, the number of actions. If $\pi_1 = \langle a_1, \ldots, a_k \rangle$ and $\pi_2 = \langle a'_1, \ldots, a'_j \rangle$ are plans, then their *concatenation* is a plan $\pi_1 \cdot \pi_2 = \langle a_1, \ldots, a_k, a'_1, \ldots, a'_j \rangle$.

The state produced by applying π to a state s is the state that is produced by applying the action of π in the order given. We will denote this by extending the state transition function γ as follows:

$$\gamma(s, \pi) = \begin{cases} s & \text{if} k = 0 \\ \gamma(\gamma(s, a_1, \langle a_2, \ldots, a_k \rangle) & \text{if} k > \text{and } a_1 \text{ is applicable to } s \\ \text{undefined} & \text{otherwise} \end{cases}$$

Plan Solution Definition

Definition (Plan Solution)

Let $\mathcal{P} = (\Sigma, s_0, g)$ be a planning problem. A plan π is a *solution* for $\mathcal{P}$ if $g \subseteq \gamma(s_0, \pi)$.

A solution can have two proprieties:

1. A solution plan π is *redundant* if there is a proper subsequence of π that is also a solution of $\mathcal{P}$.
2. A solution plan π is *minimal* if no other solution plan for $\mathcal{P}$ contains fewer actions that π.

Plan Solution Example

Example

In the planning domain described previously, suppose the initial state is s_0 and $g = \{onrobot, at2\}$. Let

- $\pi_1 = \langle move2, move2 \rangle$
- $\pi_2 = \langle take, move1 \rangle$
- $\pi_3 = \langle take, move1, put, move2, take, move1, load, move2 \rangle$
- $\pi_4 = \langle take, move1, load, move2 \rangle$
- $\pi_5 = \langle move1, take, load, move2 \rangle$

Then π_1 is not a solution because it is not applicable to s_0; π_2 is not a solution because although it is applicable to s_0, the resulting state is not a goal state; π_3 is a redundant solution; π_4 and π_5 are the only minimal solutions.

Properties of the Set Theoric Representation

1. **Readability.** On advantage of the set theoric representation is that it provides a more concise and readable representation of the state transition system than we would get by enumerating all of the states and transition explicitly.
2. **Computation.** A propositions in a state s is assumed to *persist* in $\gamma(s, a)$ unless explicitly mentioned in the effects of a. The effects are defined with two subsets: effect$^-(a)$ and effect$^+(a)$. Hence, the transition function γ and the applicability conditions of actions rely on very early computable set operations: if precond$(a) \subseteq s$;, then $\gamma(s, a) = (s -$ effect$^-(a)) \cup$ effect$^+(a)$.
3. **Expressibility.** A significant problem is that not every state transition system Σ has a set theoric representation.

Classical Representation

The classical representation scheme generalize the set theoric representation scheme using notation derived from first order logic.

- *States* are represented as set of logicals atoms that ere true or false within some interpretation.
- *Actions* are represented by *planning operators* that change the truth values of theses atoms.

States Representation

The classical planning language is built on a first order language $\mathcal{L}$.

Definition (State)

A state is a set of ground atoms of $\mathcal{L}$. $\mathcal{L}$ has no function symbols. Thus the set S of all possible states is guaranteed to be finite. As in the set of theoric representation scheme, an atom p holds in s iff $p \in s$. If g is a set of literals, we will say that s satisfies g (denoted $s \models g$) when there is a substitution σ such that every positive literal of $\sigma(g)$ is in s and no negated literal of $\sigma(g)$ is in s.

States Representation Example

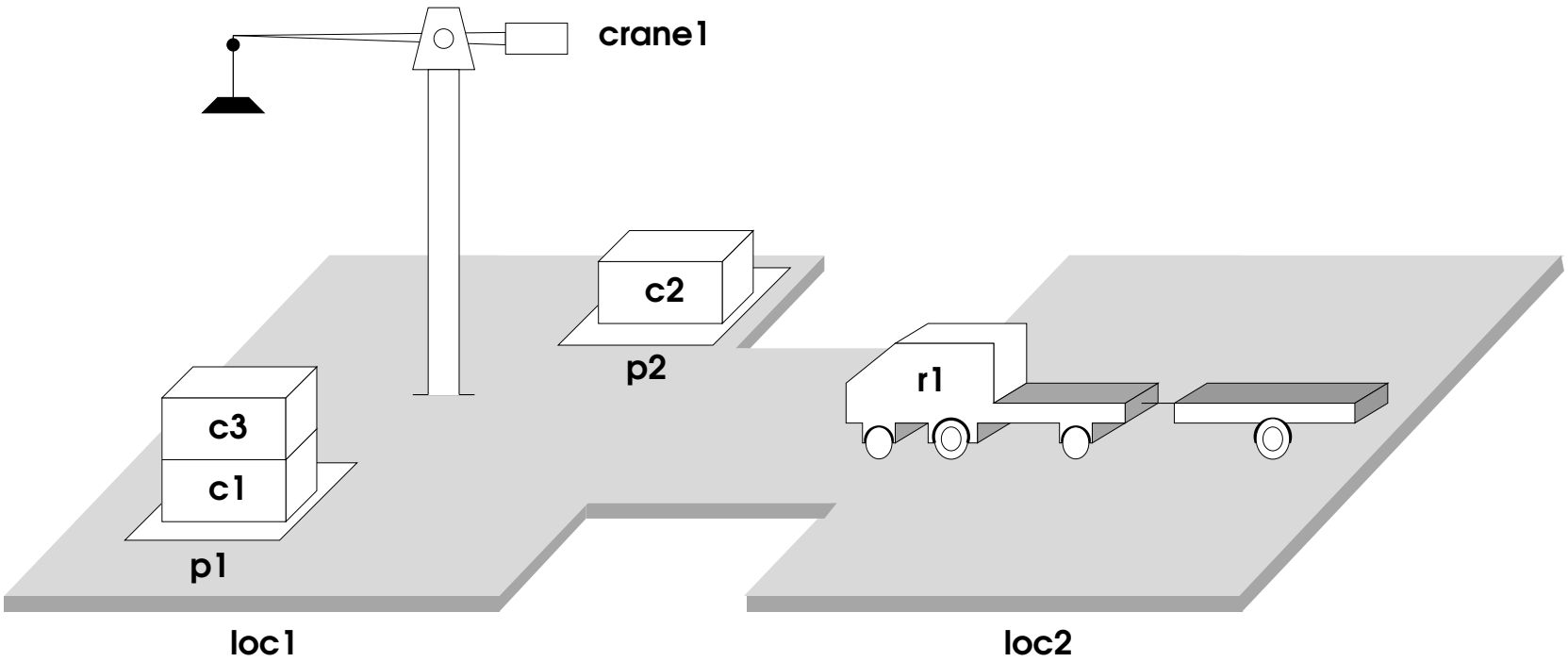

Figure: Initial state $s_0 = \{$ attached(p1, loc1), attached(p2, loc1); in(c1, p1, in(c3, p1), top(c3, p1), on(c3, c1), on(c1, pallet) in(c2, p2), top(c2, p2), on(c2,pallet), belong(crane1, loc1), empty(crane1), adjacent(loc1, loc2), adjacent(loc2, loc1), at(r1, loc2), occupied(loc2), unloaded(r1)$\}$.

Planning Operator Definition

The planning operators define the transition function γ of the state transition system.

Definition (Planning Operator)

A planning operator is a triple $o = (\text{name}(o), \text{precond}(o), \text{effects}(o))$ whose elements are follows:

- $\text{name}(o)$, the name of the operator, is a syntactic expression of the form $n(x_1, \ldots, x_k)$ where n is a symbol called an operator symbol (n is unique in $\mathcal{L}$) and $x_1, \ldots, x_k$ are all variable symbols that appear anywhere in o.
- $\text{precond}(o)$ and $\text{effects}(o)$, the preconditions and effects of o, respectively are generalizations of the preconditions and the effects of the set theory action, *i.e.,* instead of being sets of proposition they are sets of literals.

Planning Operator Example

Example (Take operator)

The planning operator take(k,l,c,d,p) can be defined as follow:

;; crane k at location l takes c off of d in pile p

take(k,l,c,d,p)

precond: belong(k,l), attached(p,l), empty(k), top(k), on(c,d)

effects: holding(k,c), $\neg$empty(k), $\neg$in(c,p), $\neg$top(c,p), $\neg$on(c,d), top(d,p)

Action Definition

Definition (Action)

An *action* is any ground instance of planning operator. If a is an action and s is a state such that $\text{precond}^+(a) \subseteq s$ and $\text{precond}^-(a) \cap s = \emptyset$, then a is applicable to s, and the result of applying a to s is the state:

$$\gamma(s, a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a)$$

Thus, like in set theoric planning, state transitions can easily be computed using set operations.

Action Example

Example

The action take(*crane*1,*loc*1,*c*3,*c*1,*p*1) is applicable to the state s_0 of the figure 2. The result is the state $s_5 = \gamma(s_0,$ take(*crane*1,*loc*1,*c*3,*c*1,*p*1)) shown by the figure below.

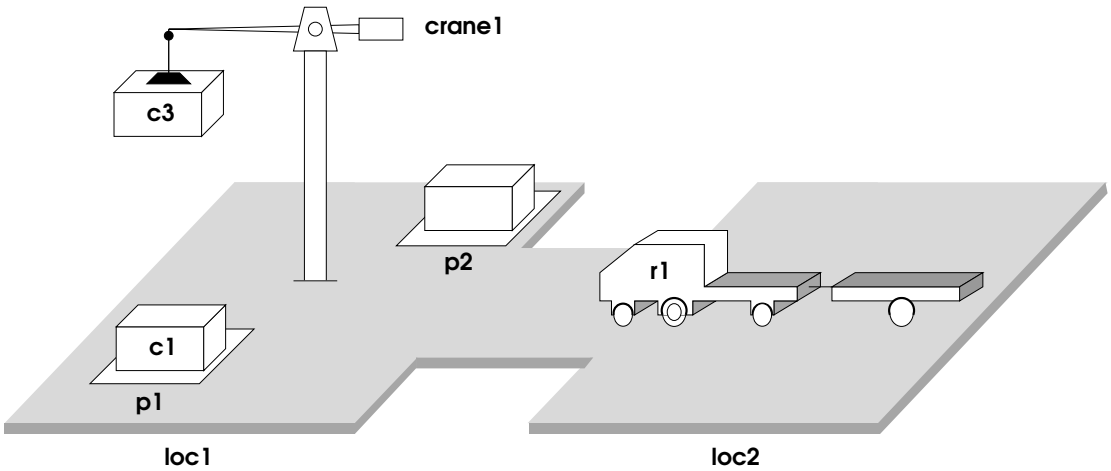

Figure: $s_5 = \{$ attached(p1, loc1), in(c1, p1), top(c1,p1), on(c1, pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,pallet), belong(crane1,loc1), holding(crane1,c3), adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc2), occupied(loc2), unloaded(r1)$\}$.

Classical Planning Domains Definition

Definition (Classical Planning Domain)

Let $\mathcal{L}$ be a first order language that has finitely many predicate symbols and constraint symbols. A classical planning domain in $\mathcal{L}$ is a restricted state transition system $\Sigma = (S, A, \gamma)$ such that:

- $S \subseteq 2^{\text{all ground atoms of} \mathcal{L}}$
- $A = \{\text{all ground instances of the operators in } \mathcal{O}\}$ where $\mathcal{O}$ is a set of operators as defined earlier
- $\gamma(s, a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a)$ if $a \in A$ is applicable to $s \in S$ and otherwise $\gamma(s, a)$ is undefined
- S is closed under γ, *i.e.*, if $s \in S$, then for every action a that is applicable to s, $\gamma(s, a) \in S$.

Classical Planning Problems Definition

Definition (Classical Planning Problem)

A classical planning problem is a triple $\mathcal{P} = (\mathcal{O}, s_0, g)$ where:

- $\mathcal{O}$ is the set of planning operators
- s_0, the initial state, is any state in S
- g, the goal, is any set of ground literals
- $S_g = \{s \in S \mid s \text{ satisfies } g\}$

Plan Example

Example

Consider the following plan:

$$\pi_1 = \langle \text{ take(crane1, loc1,c3,c1,p1),}$$
$$\text{move(r1,loc2,loc1),}$$
$$\text{load(crane1,loc1,c3,r1) } \rangle$$

This plan is applicable to the state s_0 shown in figure 2 producing the state s_6. We verify that

$$g_1 = \{\text{loaded(r1,c3), at(r1,loc1)}\}$$

is included in s_6.

Action Example

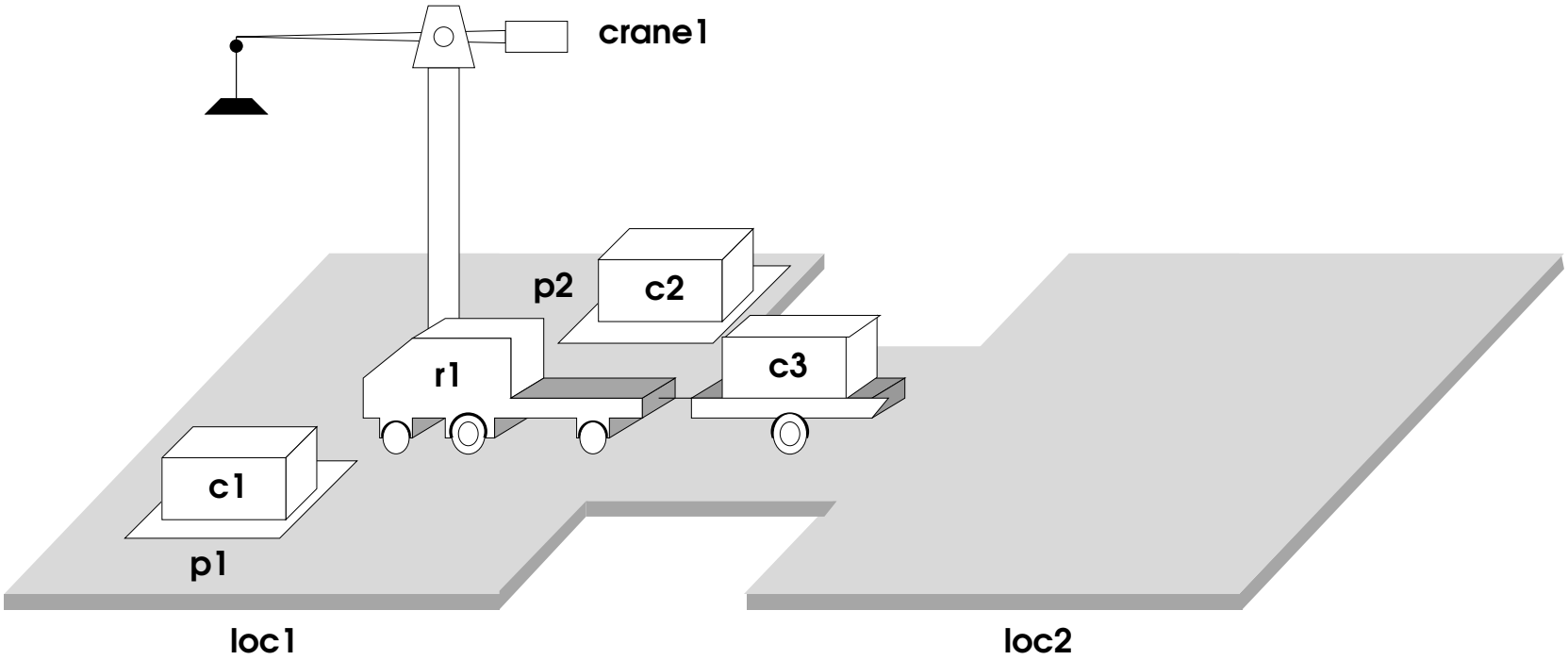

Figure: $s_6 = \{$ attached(p1, loc1), in(c1, p1), top(c1,p1), on(c1, pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,pallet), belong(crane1,loc1), empty(crane1), adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc1), occupied(loc1), loaded(r1)$\}$.

Extending the Classical Representation

Classical planning formalism is very restricted, extensions to it are needed in order to describe interesting domains. The most important extensions are :

- Typing variables
- Conditional Planning Operators
- Quantified expression
- Disjunctive preconditions
- Axiomatic Inference
- *etc.*

A planning langage, called PDDL, has been developed to express all these extensions (PDDL stands for Planning Domain Description Langage).

PDDL Example

Example (Crane robot transportation domain)

(**define** (domain dwr)
  (:**requirements** :**strips** :**typing**)
  (:**types** location pile robot crane container)
  (:**predicates**
    (adjacent ?l1 ?l2 - location) (attached ?p - pile ?l -location)
    (belong ?k - crane ?l - location) (at ?r - robot ?c container)
    (occupied ?l - location) *etc.*)
  (:**action** move
    (:**parameters** (?r - robot ?from ?to - location))
    (:**precondition** (and (adjacent .from ?to) (at ?r ?from)
      (not (occupied ?to))))
    (:**effect** (and (at ?r ?to) (not (occupied ?from)) (occupied ?to)
      (not (at ?r ?from)))))
  (:**action** load
    etc.))

Further readings

V. Lifschitz

On the semantics of STRIPS.

Reasoning about actions and plans 1-9, Morgan Kaufmann, 1987

B. Nebel

On the compatibility and expressive power of propositional planning formalism.

Journal of Artificial Intelligence Research 12:271-315, 2000

D. McDermott

PDDL, the Planning Domain Definition Language.

Technical report. Yale Center for Computational Vision and Control, 1998

ftp://ftp.cs.yale.edu/pub/mcdermott/software/pddl.tar.gz

Part III

State Space Planning

Outline of Part IV

Introduction

What is State Space Planning ?

- The simplest classical planning algorithms.
- Search algorithms in which the search space is a subset of the state space:
  - Each node corresponds to a state of the world.
  - Each arc corresponds to a state transition.
  - The current plan corresponds to the current path in the seach space.

Forward Search Principle

- The forward search algorithm is nondeterministic
- The forward search algorithm is sound and complete
- The forward search algorithm takes as input the statement $P = (\mathcal{O}, s_0, g)$ of a planning problem $\mathcal{P}$. If $\mathcal{P}$ is solvable, then Forward-search($\mathcal{O}$, s_0, g) returns a solution plan. Otherwise it returns failure.
- The plan returned by each recursive invocation of the algorithm is called a *partial solution* because it is part of the final solution returned by the top level invocation.

Forward Search Algorithm

Algorithm (ForwardSearch($\mathcal{O}$, s_0, g))

if s *satisfies* g **then return** *an empty plan* π ;
$active \leftarrow \{a \mid a$ *is a ground instance of an operator* $\mathcal{O}$
and precond(a) is true in $s\}$;
if $active = \emptyset$ **then return** *Failure* ;
nondeterministically choose an action $a_1 \in active$;
$s_1 \leftarrow \gamma(s, a_1)$;
$\pi \leftarrow ForwardSearch(\mathcal{O}, s_1, g)$;
if $\pi \neq$ *Failure* **then return** $a_1 \cdot \pi$;
else return *Failure* ;

Forward Search Example

Take the state s_5 defined in figure 4:

$s_5 = \{$ attached(p1, loc1), in(c1, p1), top(c1,p1), on(c1, pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,pallet), belong(crane1,loc1), holding(crane1,c3), adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc2), occupied(loc2), unloaded(r1)$\}$.

and the goal:

$g = \{$ at(r1,loc1), loaded(r1,c3)$\}$.

If the ForwardSearch algorithm chooses the action $a =$ move(r1,loc2,loc1) in the first invocation and $a =$ load(crane1, loc1,c3,r1) in the second invocation producing the state s_6. s_6 staisfies g, the execution returns:

$\pi = \langle$move(r1,loc2,loc1), load(cran1,loc1,c3,r1)$\rangle$

Forward Search Example

Warning

There are many other execution traces, some of which are infinite. For instance, one of them makes the following infinite sequence of choices for a:

- move(r1,loc2,loc1)
- move(r1,loc1,loc2)
- move(r1,loc2,loc1)
- move(r1,loc1,loc2)
- *etc.*

Backward Search Principle and Algorithm

The idea is to start at the goal and apply inverses of the planning operator to produce subgoals, stopping if we produce a set of subgoals satisfied by the initial state. The backward search algorithm is sound and complete.

Algorithm (BackwardSearch($\mathcal{O}$, s_0, g))

if s_0 *satisfies* g **then return** *an empty plan* π ;
$revelant \leftarrow \{a \mid a$ *is a ground instance of an operator* $\mathcal{O}$
that is revelant for $g\ \}$;
if $revelant = \emptyset$ **then return** *Failure* ;
nondeterministically choose an action $a_1 \in revelant$;
$s_1 \leftarrow \gamma^{-1}(s, a_1)$;
$\pi \leftarrow BackwardSearch(\mathcal{O}, s_1, g)$;
if $\pi \neq$ *Failure* **then return** $a_1 \cdot \pi$;
else return *Failure* ;

Backward Search Example

Recall that the initial state is the state s_5:

$s_5 = \{$ attached(p1, loc1), in(c1, p1), top(c1,p1), on(c1, pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,pallet), belong(crane1,loc1), holding(crane1,c3), adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc2), occupied(loc2), unloaded(r1)$\}$.

and the goal:

$g = \{$ at(r1,loc1), loaded(r1,c3)$\}$.

which is a subset of the state s_6.

Backward Search Example: first invocation

In the first invocation of the BackwardSearch algorithm, it chooses $a =$ load(crane1, loc1,c3,r1) and then assigns:

First Invocation

$$
\begin{aligned}
g &\leftarrow \gamma^{-1}(g, a) \\
&= (g - \text{effects}^{+}(a)) \cup \text{precond}(a) \\
&= (\{\text{at(r1,loc1), loaded(r1,c3)}\} - \{\text{empty(crane1), loaded(r1,c3)}\}) \\
&\cup \{\text{belong(crane1,loc1), holding(crane1,c3), at(r1,loc1),} \\
&\quad \text{unloaded(r1)}\} \\
&= \{\text{at(r1,loc1), belong(crane1,loc1), holding(crane1,c3),} \\
&\quad \text{unloaded(r1)}\}
\end{aligned}
$$

Backward Search Example: second invocation

In the second invocation of the BackwardSearch algorithm, it chooses a = move(r1, loc2,loc1) and then assigns:

Second Invocation

$$
\begin{aligned}
g &\leftarrow \gamma^{-1}(g, a) \\
&= (g - \text{effects}^{+}(a)) \cup \text{precond}(a) \\
&= (\{\text{at(r1,loc1), belong(crane1,loc1), holding(crane1,c3),} \\
&\quad \text{at(r1,loc1), unloaded(r1)}\} - \{\text{at(r1,loc1), occupied(loc1)}\}) \\
&\cup \{\text{adjacent(loc2,loc1), at(r1,loc2), } \neg\text{occupied(loc1)}\} \\
&= \{\text{belong(crane1,loc1), holding(crane1,c3), unloaded(r1),} \\
&\quad \text{adjacent(loc2,loc1), at(r1,loc2), occupied(loc1)}\}
\end{aligned}
$$

Backward Search Example: result

This time g is satisfied by s_5, so the execution trace terminates and returns the plans:

$\pi = \langle$(move(r1,loc2,loc1), load(crane1,loc1,c3,r1)$\rangle$

Warning

Like ForwardSearch algorithm, there are many other execution traces, some of which are infinite. For instance, one of them makes the following infinite sequence of choices for a:

- load(crane1,loc1,c3,r1)
- unload(crane1,loc1,c3,r1)
- load(crane1,loc1,c3,r1)
- unload(crane1,loc1,c3,r1)
- *etc.*

STRIPS Algorithm Principle

- The biggest problem of the previous approaches is how improve efficiency by reducing the size of the search space.
- STRIPS is somewhat similar to the BackwardSearch but differs from it in the following ways:
  1. In each recursive call of the STRIPS algorithm, the only subgoals eligible to be worked on are the preconditions of the last operator added to the plan. This reduce the branching factor substantially. However, it makes STRIPS incomplete.
  2. If the current state satisfies all of on operator's preconditions, STRIPS commits to executing that operator and will not backtrack over this commitment. This prune a large portion of the search space but again make STRIPS incomplete.

STRIPS Algorithm

Algorithm (STRIPS($\mathcal{O}$, s, g))

$\pi \leftarrow$ *the empty plan ;*
while *true* **do**
    if s *satisfies* g **then return** π ;
    $revelant \leftarrow \{a \mid a$ *is a ground instance of an operator* $\mathcal{O}$
                a *that is revelant for* $g\ \}$;
    if $revelant = \emptyset$ **then return** *Failure* ;
    nondeterministically choose an action $a \in revelant$;
    $\pi' \leftarrow STRIPS(\mathcal{O}, s, precond(a))$;
    if $\pi' =$ *Failure* **then return** *Failure* ;
    ;; *if we get here, then* π' *achieves* $precond(a)$ *from* s ;
    $s \leftarrow \gamma(s, \pi')$;
    ;; s *now satisfies* $precond(a)$;
    $s \leftarrow \gamma(s, a)$;
    $\pi' \leftarrow \pi \cdot \pi \cdot a$;
end

Sussman Anomaly

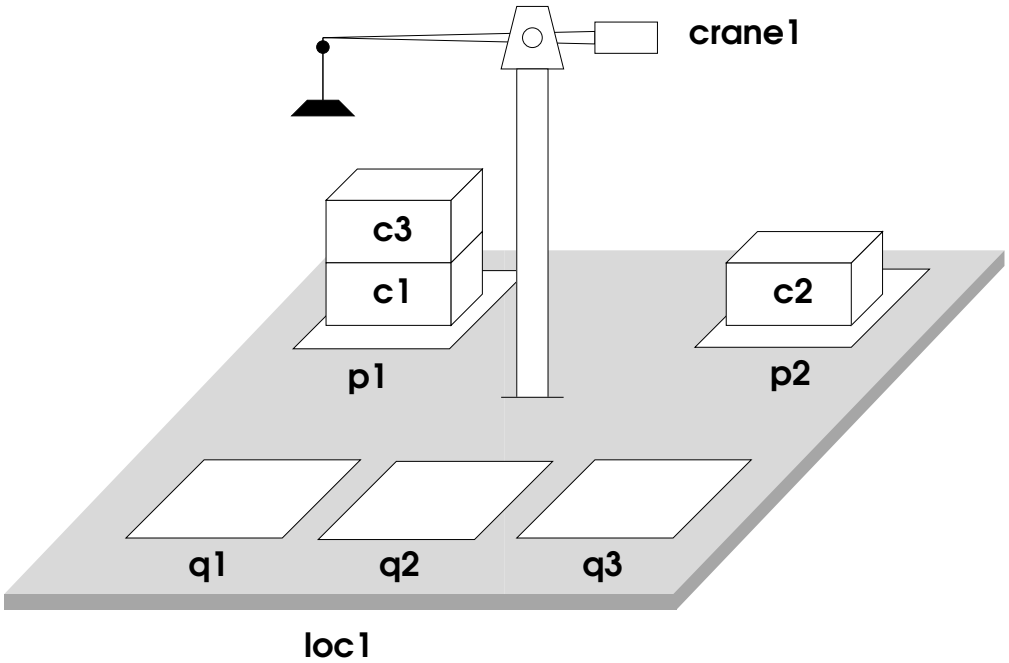

Figure: $s_0 =$ { in(c3,p1), top(c3,p1), in(c1,p1), on(c3, c1), on(c1,pallet), in(c2,p2), top(c2,p2), on(c2,pallet), top(pallet,q1), top(pallet,q2), top(pallet, q3), empty(crane1) }

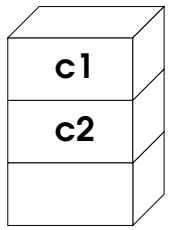

Figure: $g =$ { on(c1,c2), on(c2,c3) }

STRIPS result for the Sussman Anomaly

The shortest solutions that STRIPS can find are all similar to the following:

take(c3,loc1,crane1,c1)	
put(c3,loc1,crane1,q1)	
take(c1,loc1,crane1,p1)	
put(c1,loc1,crane1,c2)	*STRIPS has achieved on(c1,c2)*
take(c1,loc1,crane1,c2	
put(c1,loc1,crane1,p1)	
take(c2,loc1,crane1,p2)	
put(c2,loc1,crane1,c3)	*STRIPS has achieved on(c2,c3)*
	but needs to reachieved on(c1,c2)
take(c1,loc1,crane1,p1)	
put(c1,loc1,crane1,c2)	*STRIPS has now achieved both goals*

STRIPS result for the Sussman Anomaly

- STRIPS does not always find the best solution.
- STRIPS's difficulty involves *deleted condition interaction*.

Example

The action take(c1,loc1,crane1,c2) is necessary in order to help achieve on(c2,c3) but it deletes the previous acheived condition on(c1,c2).

- One way to find the shortest plan for Sussman anomaly is to interleave plans for different goals.

Note

This observation such as these led to the development of a technique called *plan space planning*, in which the planning system searches thought a space whose nodes are partial plans rather that states of the world.

Exercice

Consider the Sussman anomaly shown in figures 1 and 1. The shortest plan π_1 for achieving on(c1,c2) from the initial state is:

$$\begin{aligned} \pi_1 \quad = \quad & \langle \text{take(c3,loc1,crane1,c1)} \\ & \text{put(c3,loc1,crane1,q1)} \\ & \text{take(c1,loc1,crane1,p1)} \\ & \text{put(c1,loc1,crane1,c2)} \rangle \end{aligned}$$

and the the shortest plan π_2 for achieving on(c2,c3) from the initial state is:

$$\begin{aligned} \pi_2 \quad = \quad & \langle \text{take(c2,loc1,crane1,p2)} \\ & \text{put(c2,loc1,crane1,c3)} \rangle \end{aligned}$$

How to interleave π_1 and π_2 to find the shortest plan for the Sussman anomaly ?

Further readings


R. Fikes and N. Nilsson
STRIPS: A new approach to the application of theorem proving to problem solving.
Artificial Intelligence 2(3-4):189-208, 1971


G. Sussman
A computational model of skill acquisition.
New York: Elsevier, 1975


J. Hoffmann
FF: The fast forward planning system.
Artificial Intelligence Magazine 22(3):57-62, 2001

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.

Plan Space Example

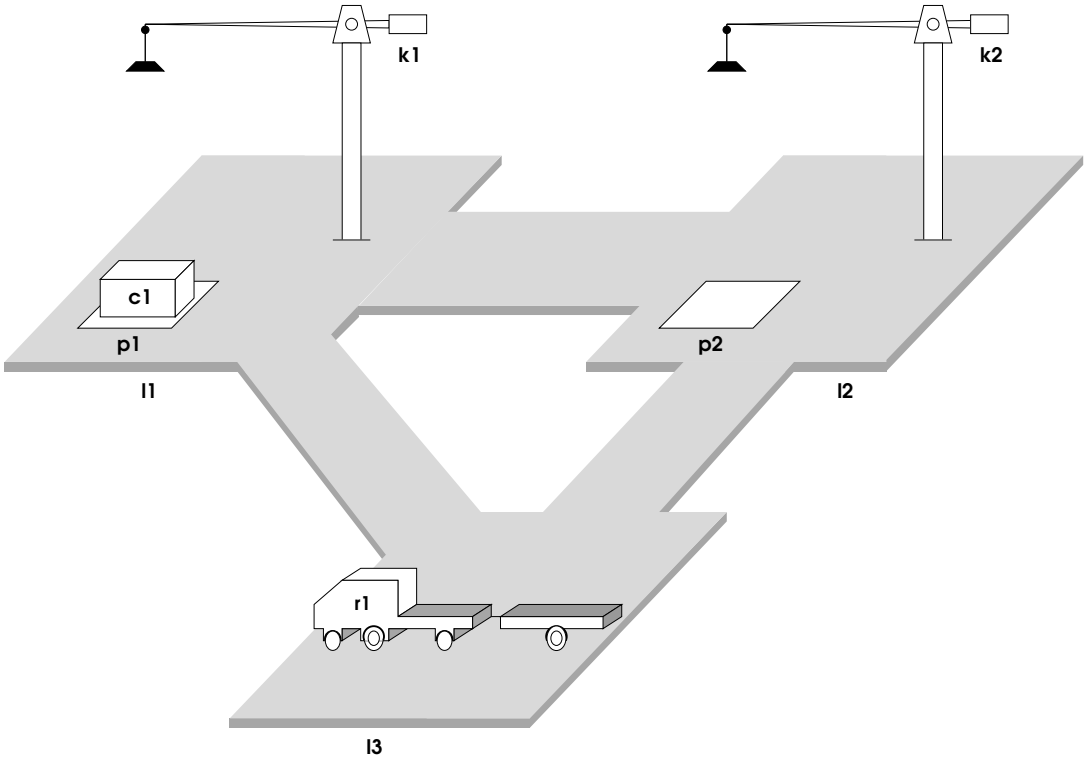

Figure: A robot r1 has to move a container c1 from pile p1 at location l1 to pile p2 and location l2. Initially r1 is unloaded at location l3. There are empty cranes k1 and k2 at locations l1 and l2. Pile p1 at location l1 contains only container c1; pile p2 at location l2 is empty. All location are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

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 figure 1. 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 3. 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.

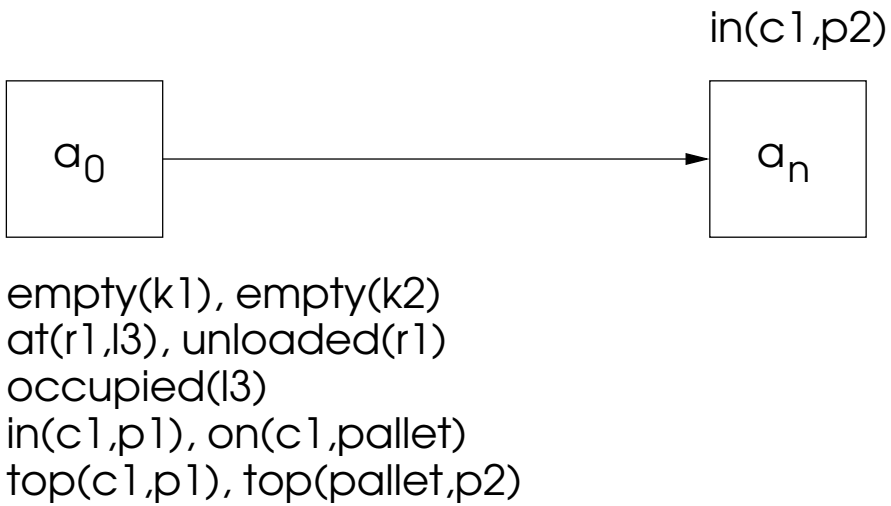

Figure: Initial plan π_0.

Partial Plan Example

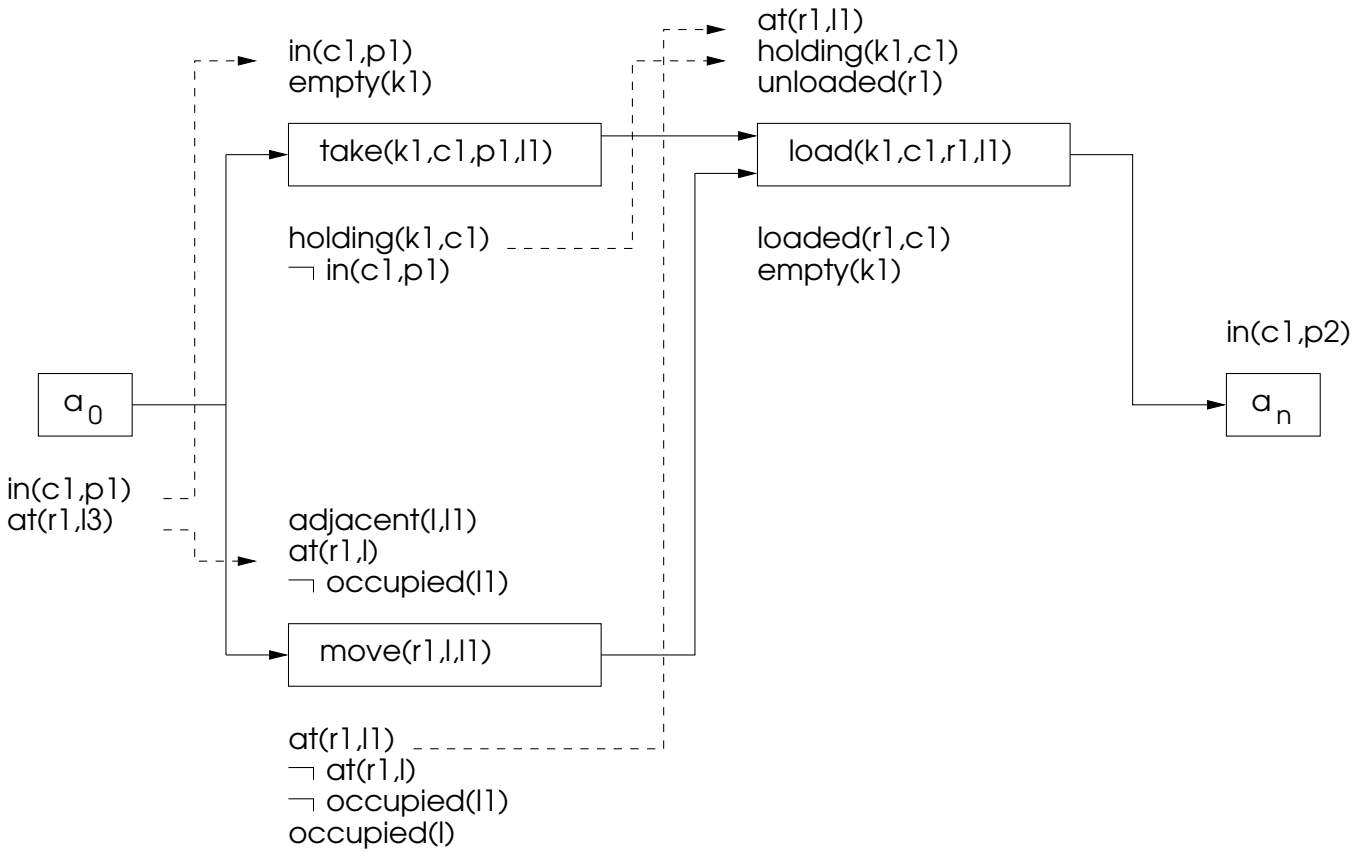

Figure: A partial plan

Solution Plan Definition

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.

Example: Plan with incorrect sequence

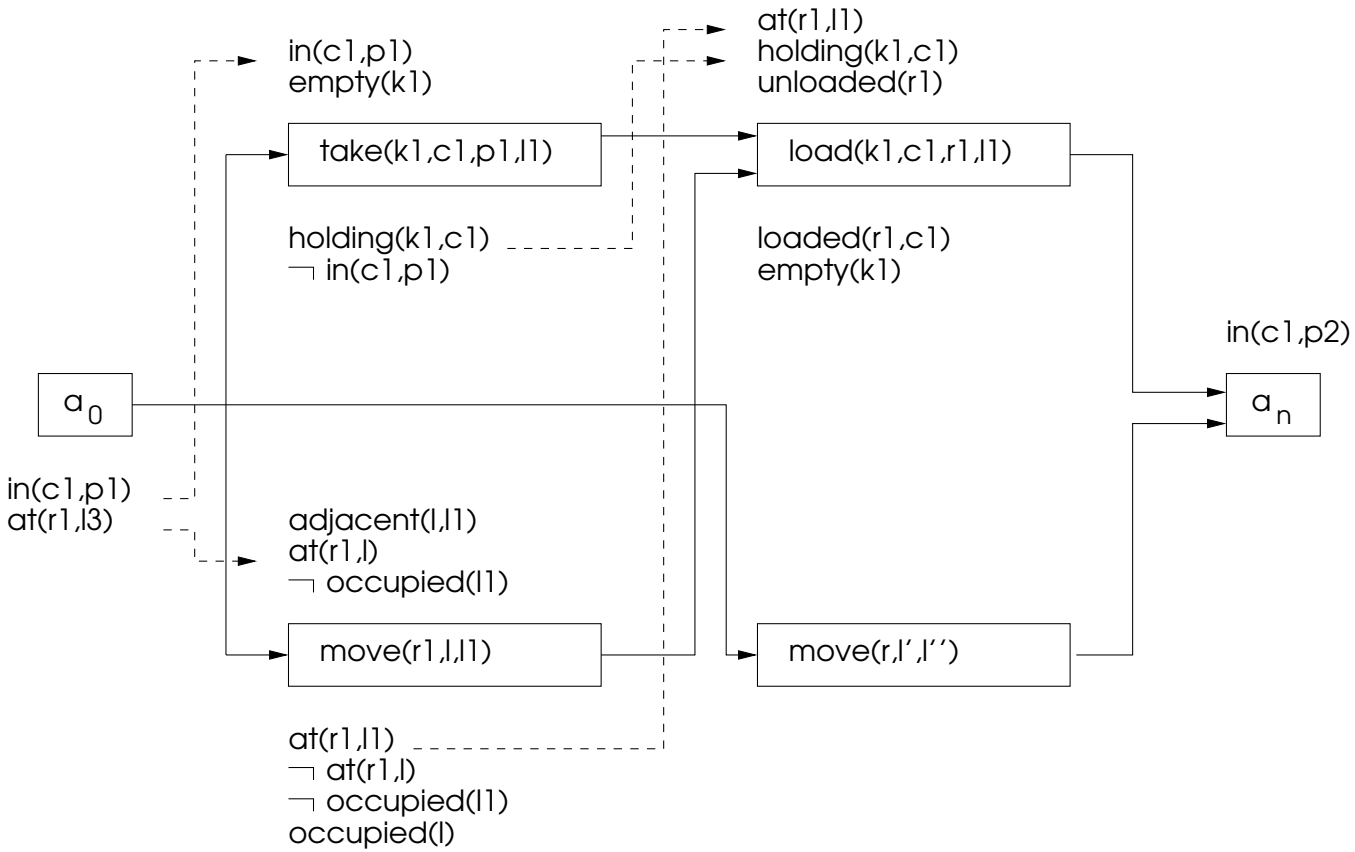

Figure: A plan containing an incorrect sequence

Flaw and Threat

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.

Example: Solution Plan

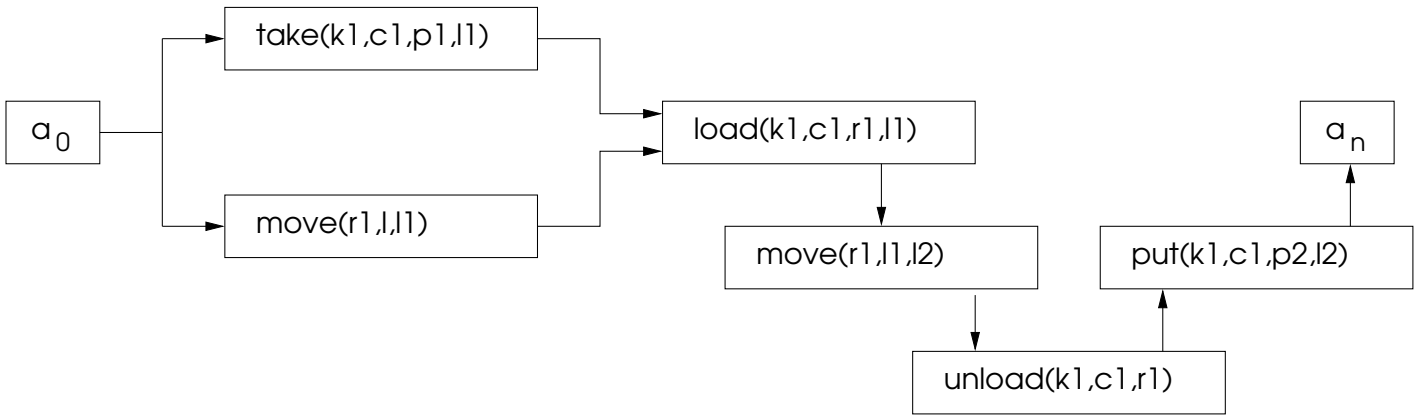

Figure: A solution plan

PSP Principle

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.

Exercice

1. Trace the PSP procedure step-by-step on the Sussman anomaly (see figure 1).
2. Draw the complete graph to compute the solution plan of the figure 4:
   - 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