State-Space Planning

Lecture 7

Outline

1. Planning
2. Classical Planning
3. State-Space Planning

Acknowledgement

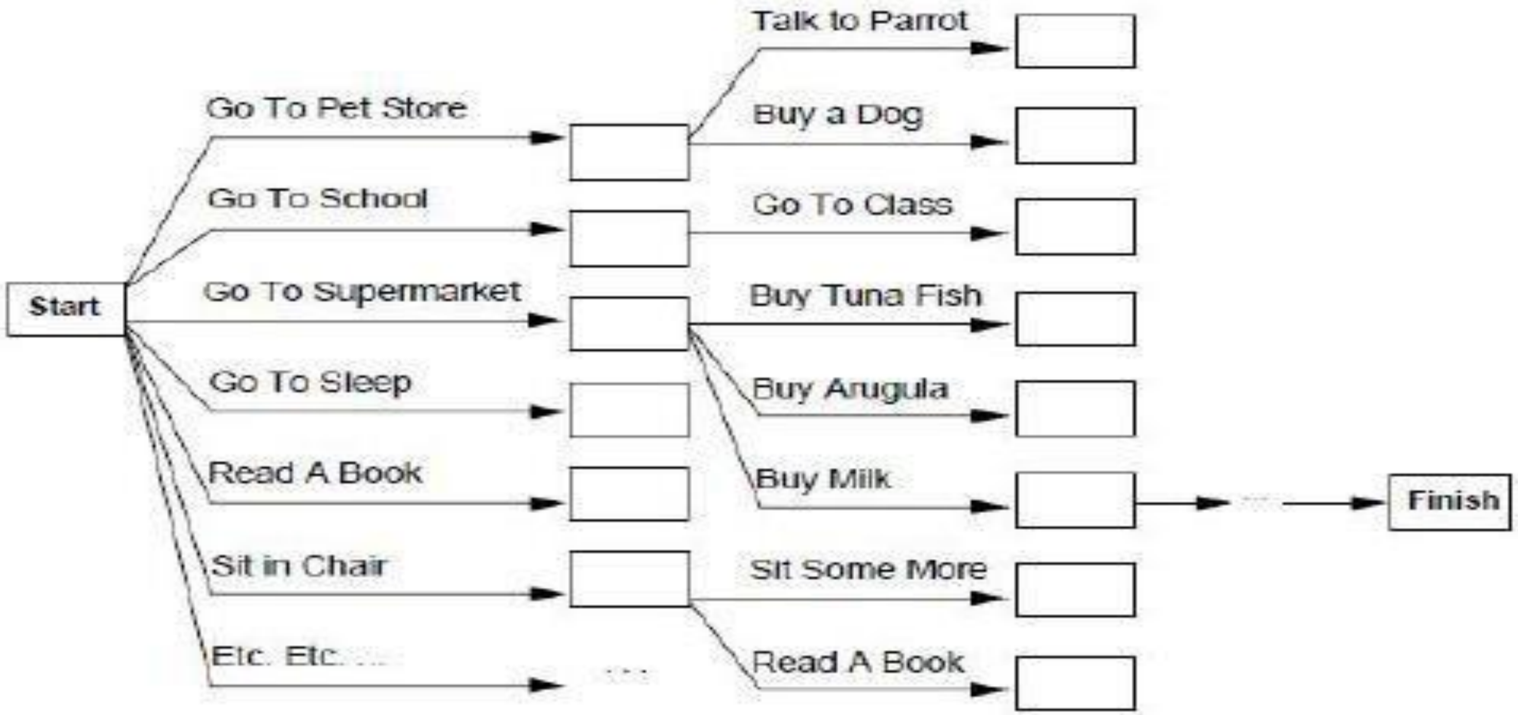
This part of the course is primarily based on material from Ghallab, Nau, and Traverso (2004). *Automated Planning: Theory and Practice*. Elsevier.
and on the slides copyrighted by Stuart Russell and Peter Norvig.

Outline

1. Planning
2. Classical Planning
3. State-Space Planning

How Far Can Search Take Us?

Consider the task *Get milk, bananas, and a cordless drill.*



Search vs. Planning

	Search	**Planning**
States	Data structures	Logical sentences
Actions	Programs	Preconditions and effects
Goal	Program for goal test	Logical sentence
Plan	Sequence from initial state	Constraints on actions

Key Ideas Behind Planning

Russell and Norvig:

1. Planning *opens up* the representation of states, goals, and actions.
   - The planner can make direct connections between actions and states.
2. The planner is free to add actions to the plan whenever they are needed, rather than in an incremental sequence starting at the initial state.
   - Decide to buy milk even before deciding how and where from.
3. Most parts of the world are independent of most other parts.
   - Divide-and-conquer to achieve conjunctive goals.

Dynamic Systems

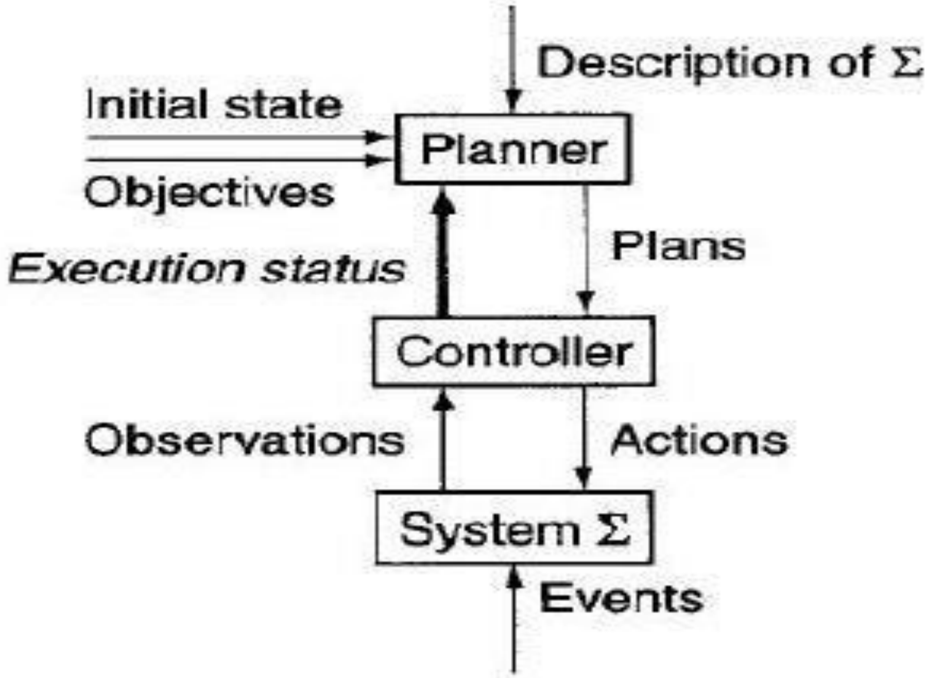
A model for planning requires a general model for a *dynamic system*.

Definition (Ghallab et al., 2004)

A state transition system is a 4-tuple $\Sigma = (S, A, E, \gamma)$, where

1. $S = \{s_1, s_2, \ldots\}$ is a finite or recursively-enumerable set of states;
2. $A = \{a_1, a_2, \ldots\}$ is a finite or recursively-enumerable set of actions, $\mathrm{no\text{-}op} \in A$ stands for no action;
3. $E = \{e_1, e_2, \ldots\}$ is a finite or recursively-enumerable set of events, $\epsilon \in E$ stands for no event; and
4. $\gamma : S \times A \times E \longrightarrow 2^S$ is a state transition function.

A Conceptual Model for Dynamic Planning

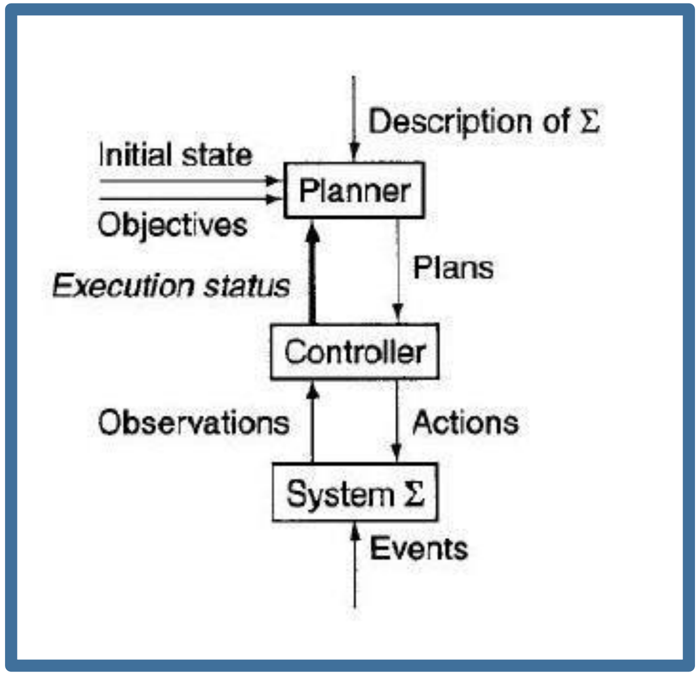


©Ghallab et al. (2004)

A Simple Planning Agent

function PLANNING-AGENT(*percept*) **returns** action
  static: *KB*
    t //A counter, initially 0
    p //A plan, initially *NoPlan*
    q //A plan, initially *NoPlan*
    G //A goal

  TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))
  current ⟵ STATE-DESCRIPTION(*KB*, *t*)
  if *p* = *NoPlan* **then**
    p ⟵ PLANNER(*current*, *G*, *KB*)
    q ⟵ *p*
    if *p* = *NoPlan* **or** *p* is empty **then return** *NoOp*
  if *p* is not valid given *KB* **then**
    p ⟵ REPLAN(*current*, *p*, *q*)
    q ⟵ *p*
  action ⟵ FIRST(*p*)
  p ⟵ REST(*p*)
  TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))
  t ⟵ *t* + 1
  return *action*



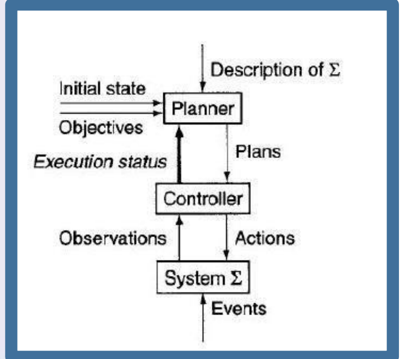
Outline

1. Planning
2. Classical Planning
3. State-Space Planning

The Restricted Model

We make the following assumptions.

1. Σ is finite. S, A, and E are finite.
2. Σ is fully observable. Observations provide complete knowledge of the actual state.
3. Σ is deterministic. The range of γ is S, not 2^S.
4. Σ is static. $E = \{\epsilon\}$; we drop the third argument of γ.
5. Restricted goals. The objective is a goal to be achieved.
6. Sequential plans. A solution plan is a linearly ordered finite sequence of actions.
7. Implicit time. Actions and events have no duration.
8. Offline planning. The planner is not concerned with any change that may occur in Σ while it is planning.



Classical Planning Problems

- In classical planning, we consider a restricted state transition system $\Sigma = (S, A, \gamma)$.
  - What happened to E?
- A planning problem is a triple $\mathcal{P} = (\Sigma, s_0, g)$, where $\Sigma = (S, A, \gamma)$, $s_0 \in S$ is the initial state, and g is the goal.
- A solution to $\mathcal{P}$ is a sequence of actions $(a_1, a_2, \ldots, a_{k-1}, a_k)$ such that

$$\gamma(\gamma(\ldots\gamma(\gamma(s_0, a_1), a_2), \ldots, a_{k-1}), a_k) \text{ satisfies } g$$

Issues in Classical Planning

1. How to represent the states and the actions in a way that does not explicitly enumerate S, A, and γ.
2. How to perform the search for a solution efficiently: which search space, which algorithm, and what heuristics and control techniques to use for finding a solution.

Classical Representation: States

We consider a first-order language $\mathcal{L}$ with finitely-many predicate symbols, finitely-many constants, countably infinite variables, and *no* function symbols.

Definition

- A state is a set of ground atoms of $\mathcal{L}$.
- An atom p holds in a state s if $p \in s$.
- An atom p does not hold in s if $p \notin s$.
  - This is known as the the closed world assumption.
- If g is a set of literals (atoms and negated atoms), we say that s satisfies g (denoted $s \models g$) if there is a substitution σ such that, for every positive literal $p \in g$, $\textsc{Subst}(\sigma, p) \in s$ and, for every negative literal $\neg n \in g$, $\textsc{Subst}(\sigma, n) \notin s$.

If g is a set of literals (atoms and negated atoms), we say that s satisfies g (denoted $s \models g$) if there is a substitution σ such that, for every positive literal $p \in g$, $\textsc{Subst}(\sigma, p) \in s$ and, for every negative literal $\neg n \in g$, $\textsc{Subst}(\sigma, n) \notin s$.

$$s = \{P(a,b), Q(b,c)\}$$
$$g = \{P(x,b), \neg Q(x,c)\}$$

$$\theta = \{a/x\}$$
$$\textsc{Subst}(\theta, g) = \{P(a,b), \neg Q(a,c)\}$$

Classical Representation: Rigid Relations

- Rigid relations do not change with time.
- Examples include categories of objects, locations of unmovable objects, etc.
- Rigid relations are assumed to hold in every state.

Classical Representation: Operators

Definition

A planning operator is a triple $o = (\text{name}(o), \text{precond}(o), \text{effects}(o))$, where

- $\text{name}(o)$ is a syntactic expression of the form $n(x_1, x_2, \ldots, x_k)$, where $x_1, x_2, \ldots, x_k$ are variables and n is a unique symbol;
- $\text{precond}(o)$ and $\text{effects}(o)$ are sets of literals, denoting the preconditions and effects of o, respectively; and
- $\text{precond}^+(o)$ and $\text{precond}^-(o)$ are the sets of literals appearing, respectively, positively and negatively in $\text{precond}(o)$. Similarly for $\text{effects}^+(o)$ and $\text{effects}^-(o)$.

Classical Representation: Actions

Definition

- An action is any ground instance of a planning operator.
- An action a is applicable to a state s if $\text{precond}^+(a) \subseteq s$ and $\text{precond}^-(a) \cap s = \varnothing$.
- If a is applicable to s, we let

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

This way, we can define a planning problem as (O, s_0, g), where O is the set of planning operators, instead of (Σ, s_0, g).

Classical Representation: Goals and Solutions

Definition

- A goal is a set g of $\mathcal{L}$ literals.
  - Note that $g \not\subseteq S$.
- A plan $P = (a_1, a_2, \ldots, a_k)$ is a solution of a planning problem (O, s_0, g) if
  1. a_i is a ground instance of some $o \in O$; and
  2. the state

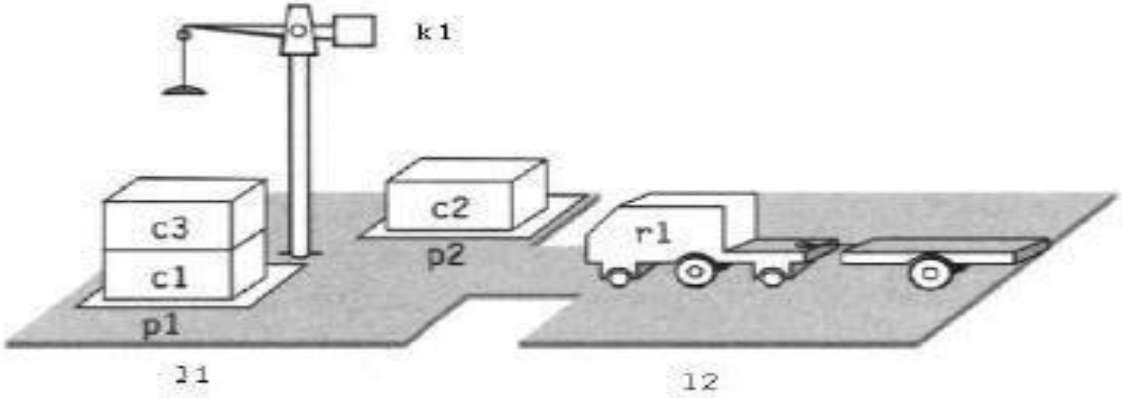
     $$\gamma(\gamma(\ldots\gamma(\gamma(s_0, a_1), a_2), \ldots, a_{k-1}), a_k)$$

     satisfies g (as indicated before).

Running Example: Dock-Worker Robots

- The domain of Dock-worker robots (DWR) from Ghallab et al. (2004) represents a harbor served by robots to load, unload, and move containers. In particular, we have
  1. a set of locations $\{l1, l2, \ldots\}$;
  2. a set of robots $\{r1, r2, \ldots\}$, each can be loaded and can transport one container at a time to any adjacent location;
  3. a set of cranes $\{k1, k2, \ldots\}$, which cannot move;
  4. a set of containers $\{c1, c2, \ldots\}$, which can be loaded and unloaded using cranes and can be transported using robots; and
  5. a set of piles $\{p1, p2, \ldots\}$, which are fixed areas attached to locations each containing a stack of containers.

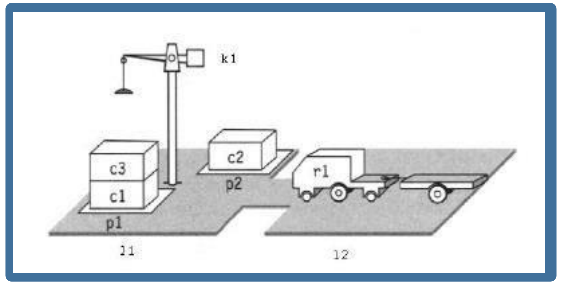
DWR: Sample State



$s_1 =$
$\{attached(p1, l1), attached(p2, l1), in(c1, p1), in(c3, p1), top(c3, p1), on(c3, c1), on(c1, pallet), in(c2, p2), top(c2, p2), on(c2, pallet), belong(k1, l1), empty(k1), adj(l1, l2), adj(l2, l1), at(r1, l2), occupied(l2), unloaded(r1)\}$

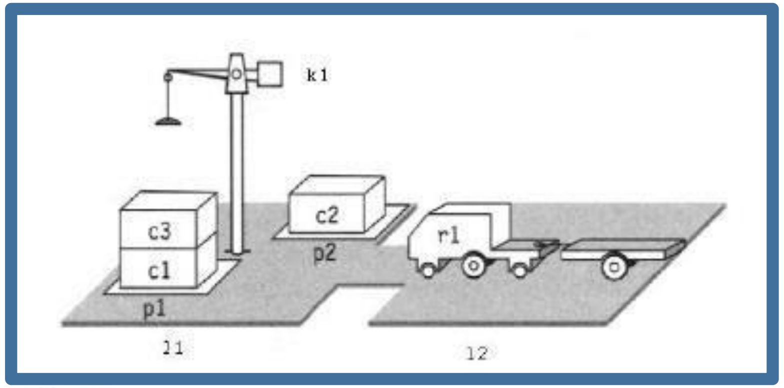
Note that $belong(k1, l1)$, $adj(l1, l2)$, and $adj(l2, l1)$ are rigid.

DWR: Sample Operators



- $move(r, l, m)$: robot r moves from location l to an adjacent location m
  - Preconditions: $adj(l, m), at(r, l), \neg occupied(m)$
  - Effects: $at(r, m), occupied(m), \neg occupied(l), \neg at(r, l)$
- $load(k, l, c, r)$: crane k at location l loads container c onto robot r.
  - Preconditions: $belong(k, l), holding(k, c), at(r, l), unloaded(r)$
  - Effects: $empty(k), \neg holding(k, c), loaded(r, c), \neg unloaded(r)$
- $unload(k, l, c, r)$: crane k at location l takes container c from robot r.
  - Preconditions: $belong(k, l), at(r, l), loaded(r, c), empty(k)$
  - Effects: $\neg empty(k), holding(k, c), unloaded(r), \neg loaded(r, c)$
- $put(k, l, c, d, p)$: crane k at location l puts c onto d in pile p.
  - Preconditions: $belong(k, l), attached(p, l), holding(k, c), top(d, p)$
  - Effects: $\neg holding(k, c), empty(k), in(c, p), top(c, p), on(c, d), \neg top(d, p)$
- $take(k, l, c, d, p)$: crane k at location l takes c off of d in pile p.
  - Preconditions: $belong(k, l), attached(p, l), empty(k), top(c, p), on(c, d)$
  - Effects: $holding(k, c), \neg empty(k), \neg in(c, p), \neg top(c, p), \neg on(c, d), top(d, p)$

DWR: Sample Action



name(a_1): $take(k1, l1, c3, c1, p1)$.

precond(a_1): $belong(k1, l1), attached(p1, l1),$
$empty(k1), top($ c3, p1 $), on($ c3, c1 $)$

effects(a_1): $holding(k1, c1), \neg empty(k1), \neg in(c1, p1),$
$\neg top(c1, p1), \neg on($ c3, c1 $), top($ c1, p1 $)$

- Note that a_1 is applicable to s_1.
- What is $\gamma(a_1, s_1)$?

Outline

1. Planning
2. Classical Planning
3. State-Space Planning

Progression State-Space Planning

- Progression planning reduces a classical planning problem, in a straightforward way, to a search problem from the initial state to a goal state.
- Typically, planning algorithms are described as non-deterministic algorithms.
- Not efficient, in general.
  - Recall the milk, banana, and cordless drill example.

Progression Planning: The Algorithm

function $\textsc{Progression-Planner}(O, s_0, g)$ **returns** a plan

$s \longleftarrow s_0$

$\pi = ()$

loop

if s satisfies g, **then return** π

$app = \{a | a$ is a ground instance of some $o \in O$ which is applicable to $s\}$

if $app = \varnothing$ **then return** *failure*

nondeterministically choose $a \in app$

$s \longleftarrow \gamma(s, a)$

$\pi \longleftarrow \pi \cdot a$

Regression State-Space Planning

- Regression planning reduces the planning problem to a search problem from a goal state back to the initial state.
- It depends on the following notions.

Definition

1. Let g be a set of literals. An operator-substitution pair (o, σ) is relevant for g if
   1. $\textsc{Subst}(\sigma, g) \cap \text{effects}(\textsc{Subst}(\sigma, o)) \neq \varnothing$;
   2. $\textsc{Subst}(\sigma, g^{+}) \cap \text{effects}^{-}(\textsc{Subst}(\sigma, o)) = \varnothing$; and
   3. $\textsc{Subst}(\sigma, g^{-}) \cap \text{effects}^{+}(\textsc{Subst}(\sigma, o)) = \varnothing$.
2. If (o, σ) is relevant for g, let

$$\gamma^{-1}(g, (o, \sigma)) = (\textsc{Subst}(\sigma, g) - \text{effects}(\textsc{Subst}(\sigma, o)))$$

$$\cup \text{precond}(\textsc{Subst}(\sigma, o))$$

Regression as Search

Regression planning defines a search problem for $P = (O, s_0, g)$ where

- a state is a set of literals;
- the initial state is g;
- the goal test on a state g' checks if s_0 satisfies g';
- the operators used to expand a state g' are all operator-substitution pairs relevant to g'; and
- the state-successor function is γ^{-1}.

Regression Planning: The Algorithm

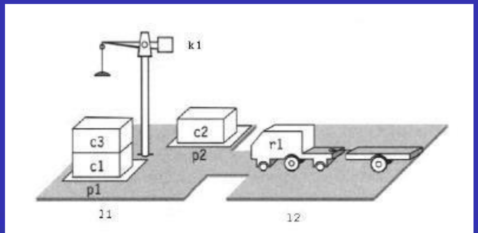
function REGRESSION-PLANNER(O, s_0, g) **returns** a plan
  $\pi = ()$
  loop
    if s_0 satisfies g with substitution σ, **then return** SUBST(σ, π)
    $rel = \{(o, \sigma) | (o, \sigma)$ is relevant for $g\}$
    if $rel = \varnothing$ **then return** *failure*
    nondeterministically choose $(o, \sigma) \in rel$
    $g \longleftarrow \gamma^{-1}(g, (o, \sigma))$
    $\pi \longleftarrow$ SUBST(σ, o) $\cdot$ SUBST(σ, π)

DWR Regression

Example

Use regression planning to solve the DWR problem where the initial state is s_1 ( ) and the goal is $\{loaded(r1, c3)\}$

DWR Regression: Solution



Example

State	$\text{SUBST}(\sigma, \mathbf{o})$
$\{loaded(r1, c3)\}$	$load(k, l, c3, r1)$
$\{belong(k, l), holding(k, c3), at(r1, l), unloaded(r1)\}$	$move(r1, m, l)$
$\{belong(k, l), holding(k, c3), at(r1, m), unloaded(r1)$ $adj(m, l), \neg occupied(l)\}$	$take(k, l, c3, d, p)$
$\{belong(k, l), at(r1, m), unloaded(r1), adj(m, l),$ $\neg occupied(l), attached(p, l), empty(k),$ $top(c3, p), on(c3, d)\}$	

- With $\sigma = \{k1/k, p1/p, l1/l, c1/d, l2/m\}$, s_1 satisfies the last state.
- Hence, $\pi = (take(k1, l1, c3, c1, p1), move(r1, l2, l1), load(k1, l1, c3, r1))$.

To Be Safe

When applying a relevant operator, always equip the operator with a fresh set of variables.

Example

- $g = \{at(r,l), at(r',l'), \ldots\}$
- Applying $move(r', m, l')$ replaces (among other things) $at(r', l')$ with $at(r', m)$.
- Now, applying $move(r, m, l)$ replaces (among other things) $at(r, l)$ with $at(r, m)$.
- But the locations of r and r' need not be the same!