Escola Tècnica Superior d'Enginyeria

Universitat Rovira i Virgili

AI Planning

Hatem A. Rashwan

Plan-Space Search and Hierarchical Planning

Plan-Space Search

Plan-Space Planner (PSP)
Partial Plans

Overview

- **Search States: Partial Plans**
- Plan Refinement Operations
- The Plan-Space Search Problem
- Flawless Partial Plans
- The PSP Algorithm

State-Space vs. Plan-Space Search

- state-space search:
  search through graph (tree) of nodes representing world states
- plan-space search:
  search through graph of partial plans
  - nodes: partially specified plans
  - arcs: plan refinement operations
  - solutions: partial-order plans

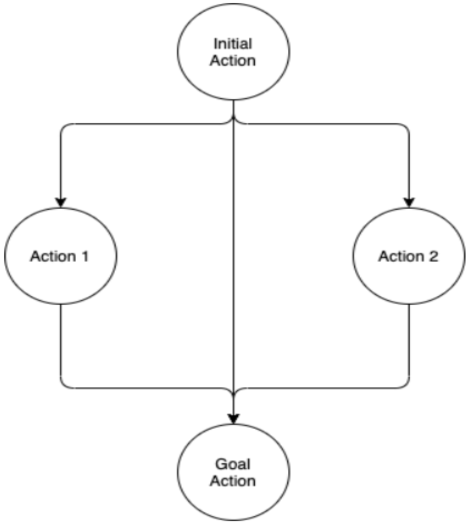
State-Space vs. Plan-Space Search

Planning as Search

	State Space		Plan Space
Algorihtm	Progression	Regression	Partial-Order causal link: UCPOP
Node	World State	Set of World States	Partial Plans
Edge	Apply Action If prec satisfied, Add adds, Delete deletes	Regress Action If a provides some g in CG: CG' = CG – effects(a) + preconditions(a)	Plan refinements: ▪ Satisfy Goals: ▪ Step addition ▪ Step reuse ▪ Resolve Threats ▪ Demotion ▪ Promotion ▪ Confrontation

Partial Plans

- plan: set of actions organized into some structure
- partial plan:
  - subset of the actions
  - subset of the organizational structure
    - temporal ordering of actions
    - rationale: what the action achieves in the plan
  - subset of variable bindings



Partially-ordered plan

Definition of Partial Plans

- A partial plan is a tuple $\pi = (A, \prec, B, L)$, 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 $x=y$, $x \neq y$;
  - L is a set of causal links of the form $\langle a_i \rightarrow [p] \rightarrow a_j \rangle$ 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 in a_i and a_j appearing in p are in B.

Overview

- Search States: Partial Plans
- **Plan Refinement Operations**
- The Plan-Space Search Problem
- Flawless Partial Plans
- The PSP Algorithm

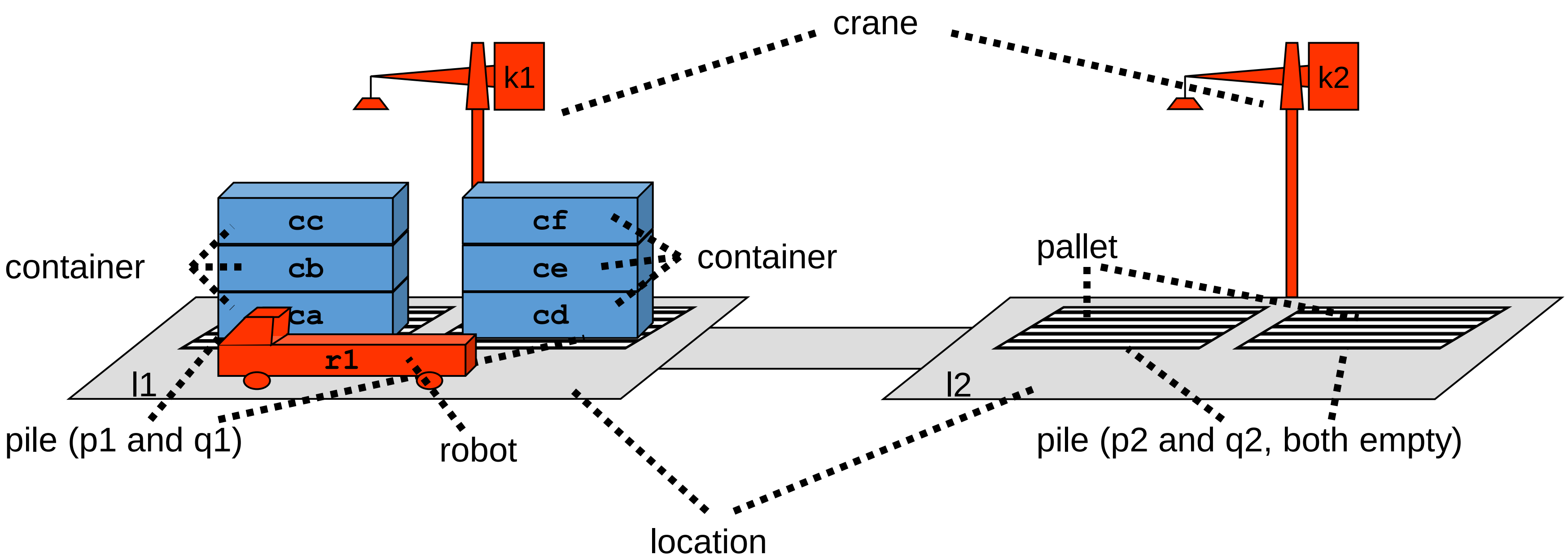
Adding Actions

- partial plan contains actions
  - initial state
  - goal conditions
  - set of operators with different variables

Operators indicate what action to perform and with which variables.

- reason for adding new actions
  - to achieve unsatisfied preconditions
  - to achieve unsatisfied goal conditions

Dock-Worker Robots (DWR) Example State



Predicates in the DWR Domain

(loaded ?r -robot ?c -container); robot r loaded with container c

(unloaded ?r -robot); robot r without loading

(at ?r -robot ?l -location) ; robot r in a location l

(belongs ?k -crane ?l -location); carne k belongs to a location l

(attached ?p -pile ?l -location); pile p "attached" to a location l

*(adjacent ?l1 ?l2 -location);*location l1 is adjacent to location l2

(occupied ?l -location); location is full (the robot cannot come)

(in ?c -container ?p -pile); container c on a pile p

(on ?c ?cc -container); container c on container cc

(top ?c -container ?p -pile); container c is at the top of a pile p

(empty ?k -crane); empty crane k

(holding ?k -crane ?c -container); crane k holds a container c

Actions in the DWR Domain

- **`Move(r,l,l')`** robot *r* from location *l* to some adjacent and unoccupied location *l'*
- **`Take(c,k,p,l)`** container *c* with empty crane *k* from the top of pile *p*, all located at the same location *l*
- **`Put(k,l,c,p)`** container *c* held by crane *k* on top of pile *p*, all located at location *l*
- **`Load(k,l,c,r)`** container *c* held by crane *k* onto unloaded robot *r*, all located at location *l*
- **`Unload(k,l,c,r)`** container *c* with empty crane *k* from loaded robot *r*, all located at location *l*

Adding Actions: Example

initial state

- attached(pile,loc1)
- in(cont,pile)
- top(cont,pile)
- on(cont,pallet)
- belong(crane,loc1)
- empty(crane)
- adjacent(loc1,loc2)
- adjacent(loc2,loc1)
- at(robot,loc2)
- occupied(loc2)
- unloaded(robot)

1:move(r_1,l_1,m_1)

preconditions	effects
at(r_1,l_1)	at(r_1,m_1)
¬occupied(m_1)	occupied(m_1)
adjacent(l_1,m_1)	¬occupied(l_1)
	¬at(r_1,l_1)

2:load(k_2,l_2,c_2,r_2)

preconditions	effects
belong(k_2,l_2)	empty(k_2)
holding(k_2,c_2)	loaded(r_2,c_2)
at(r_2,l_2)	¬holding(k_2,c_2)
unloaded(r_2)	¬unloaded(r_2)

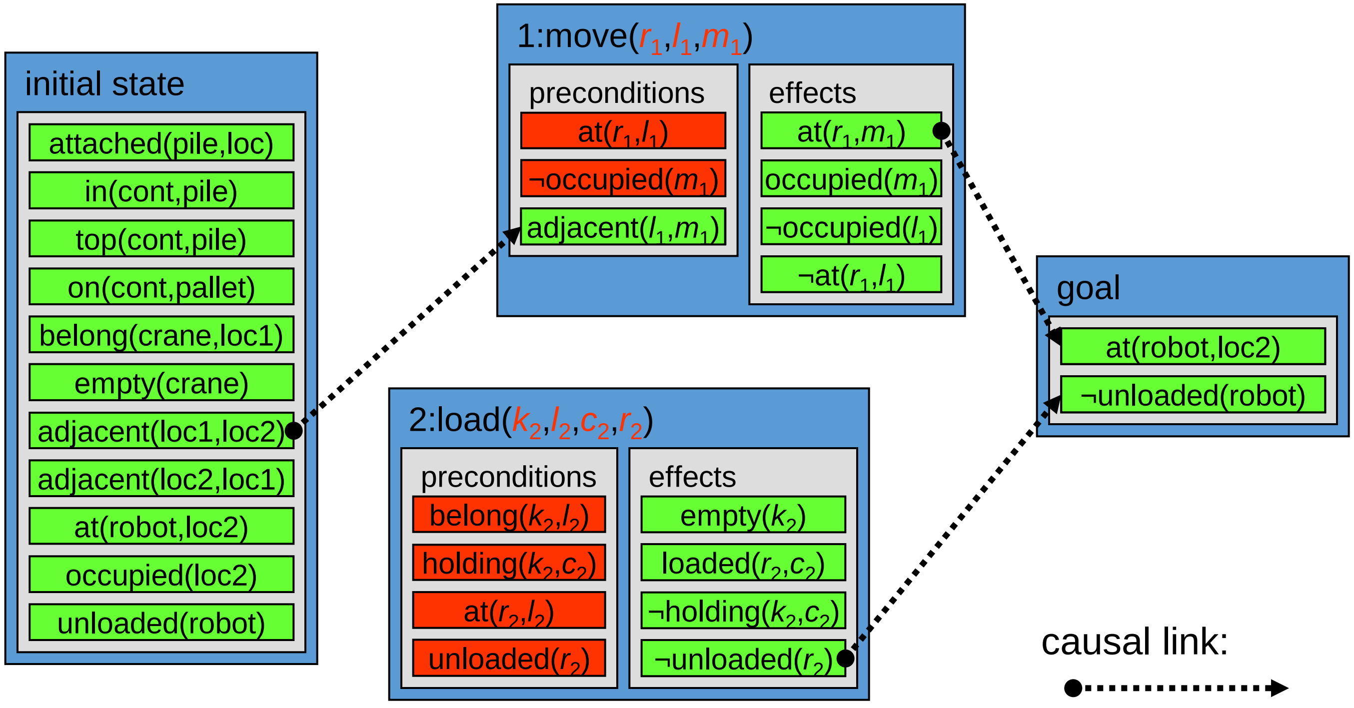
goal

- at(robot,loc2)
- ¬unloaded(robot)

Adding Causal Links

- partial plan contains causal links
  - links from the provider
    - an effect of an action or
    - an atom that holds in the initial state
  - to the consumer
    - a precondition of an action or
    - a goal condition
- reasons for adding causal links
  - prevent interference with other actions

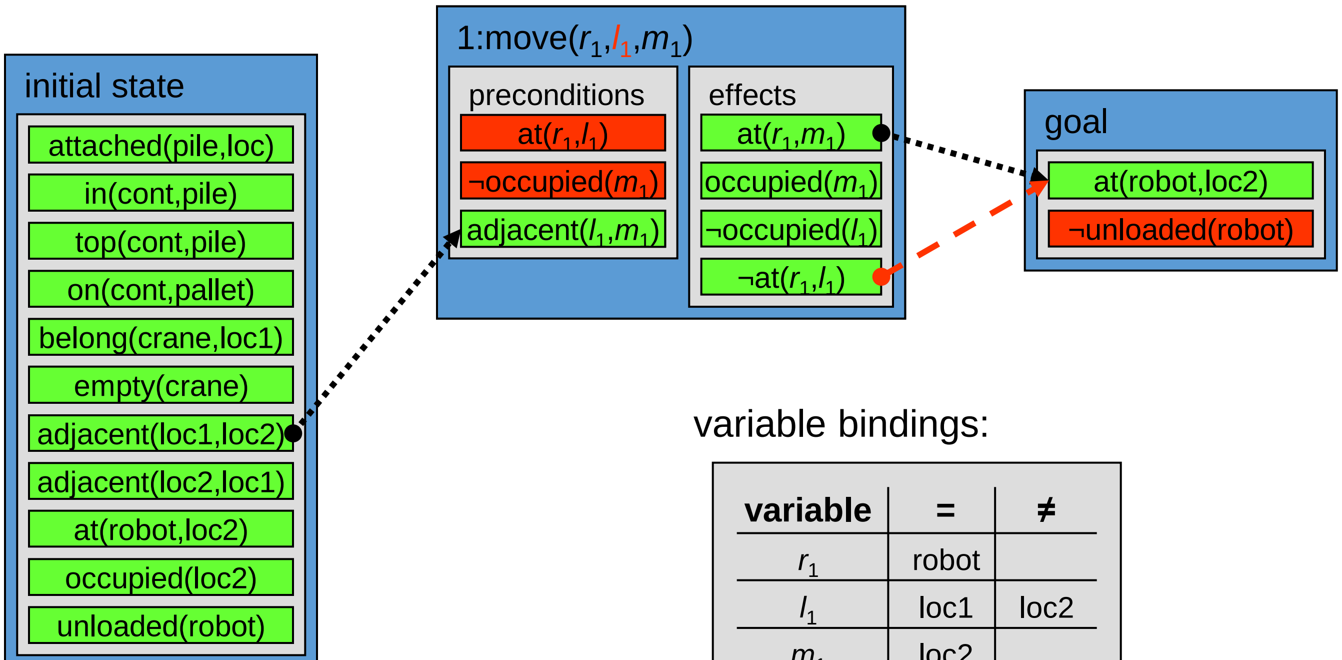
Adding Causal Links: Example



Adding Variable Bindings

- partial plan contains variable bindings
  - new operators introduce new (copies of) variables into the plan
  - solution plan must contain actions
  - variable binding constraints keep track of possible values for variables and co-designation
- reasons for adding variable bindings
  - to turn operators into actions
  - to unify and effect with the precondition it supports

Adding Variable Bindings: Example



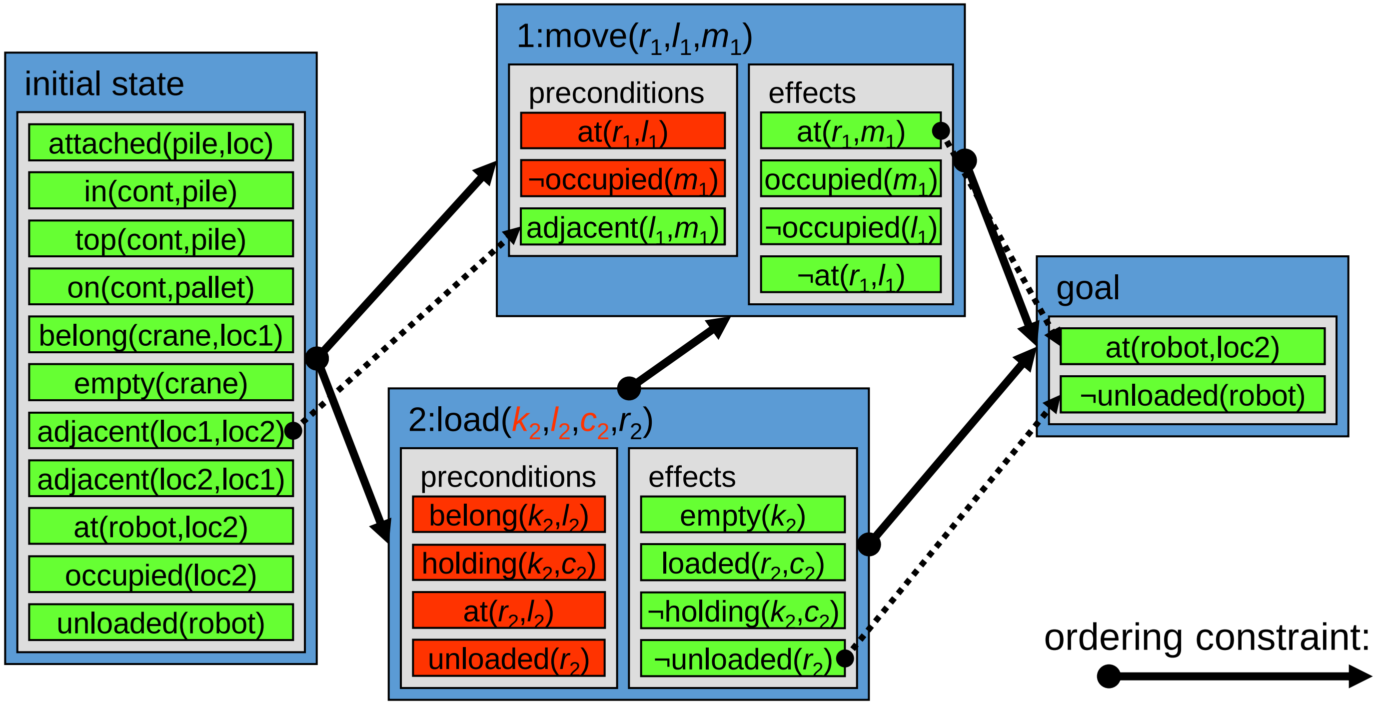
variable bindings:

variable	**=**	**≠**
r_1	robot	
l_1	loc1	loc2
m_1	loc2	

Adding Ordering Constraints

- partial plan contains ordering constraints
  - binary relation specifying the temporal order between actions in the plan
- reasons for adding ordering constraints
  - all actions after initial state
  - all actions before goal
  - causal link implies ordering constraint
  - to avoid possible interference

Adding Ordering Constraints: Example



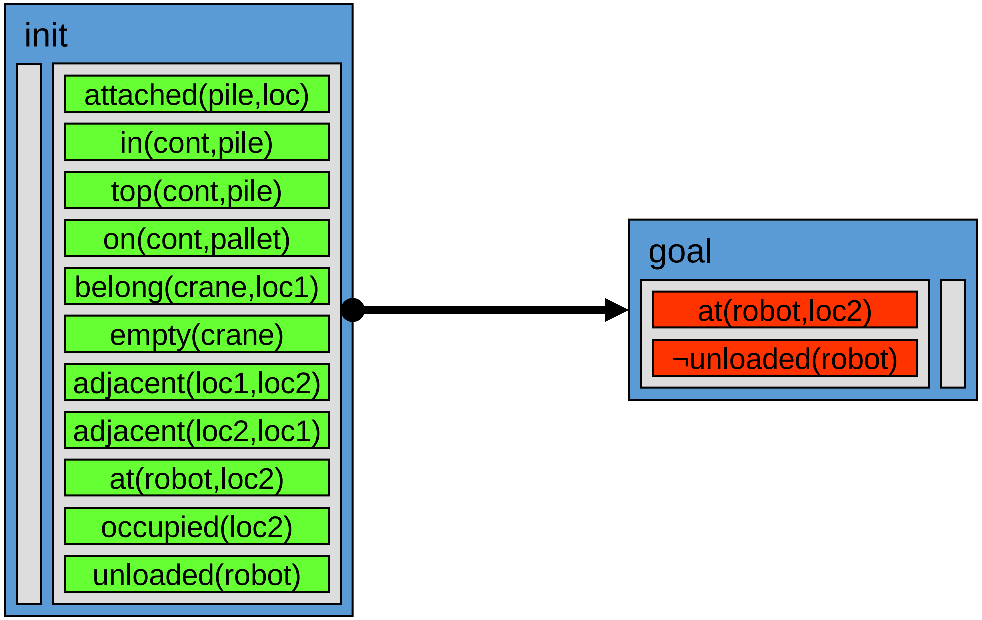
Overview

- Search States: Partial Plans
- Plan Refinement Operations
- **Plan-Space Search Problem**
- Flawless Partial Plans
- The PSP Algorithm

Plan-Space Search: Initial Search State

- represent initial state and goal as dummy actions
  - init: no preconditions, initial state as effects
  - goal: goal conditions as preconditions, no effects
- empty plan π_0 = ({init, goal},{(init$\prec$goal)},{},{}):
  - two dummy actions init and goal;
  - one ordering constraint: init before goal;
  - no variable bindings; and
  - no causal links.

Plan-Space Search: Initial Search State Example



Plan-Space Search: Successor Function

- states are partial plans
- generate successor through plan refinement operators (one or more):
  - adding an action to A
  - adding an ordering constraint to $\prec$
  - adding a binding constraint to B
  - adding a causal link to L

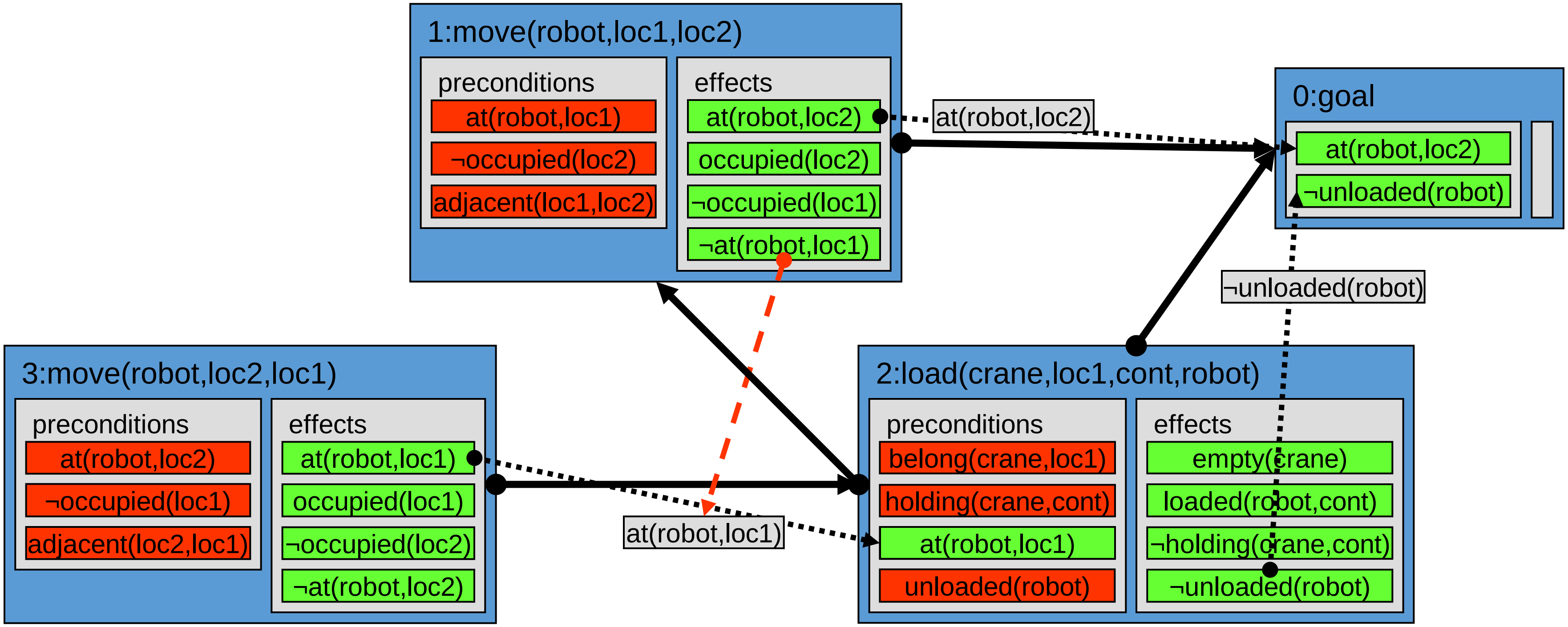
Partial Order Solutions

- Let $\mathcal{P}=(\Sigma,s_i,g)$ be a planning problem. A plan $\pi = (A,\prec,B,L)$ is a (partial order) solution for $\mathcal{P}$ if:
  - its ordering constraints $\prec$ and binding constraints B are consistent; and
  - for every sequence $\langle a_1,\ldots,a_k\rangle$ of all the actions in A-{init, goal} that is
    - totally ordered and grounded and respects $\prec$ and B
    - $\gamma(s_i, \langle a_1,\ldots,a_k\rangle)$ must satisfy g.

Overview

- Search States: Partial Plans
- Plan Refinement Operations
- Plan-Space Search Problem
- **Flawless Partial Plans**
- The PSP Algorithm

Threat: Example



Threats

- An action a_k in a partial plan $\pi = (A,\prec,B,L)$ is a threat to a causal link $\langle a_i -[p]\rightarrow a_j\rangle$ iff:
  - a_k has an effect $\neg q$ that is possibly inconsistent with p, i.e. q and p are unifiable;
  - the ordering constraints $(a_i \prec a_k)$ and $(a_k \prec a_j)$ are consistent with $\prec$; and
  - the binding constraints for the unification of q and p are consistent with B.

Flaws

- A flaw in a plan $\pi = (A, \prec, B, L)$ is either:
  - an unsatisfied sub-goal, i.e. a precondition of an action in *A* without a causal link that supports it; or
  - a threat, i.e. an action that may interfere with a causal link.

Flawless Plans and Solutions

- **Proposition: A partial plan π = (A,$\prec$,B,L) is a solution to the planning problem $\mathcal{P}$=(Σ,s_i,g) if:**
  - π has no flaw;
  - the ordering constraints $\prec$ are not circular; and
  - the variable bindings B are consistent.

Overview

- Search States: Partial Plans
- Plan Refinement Operations
- The Plan-Space Search Problem
- Flawless Partial Plans
- **The PSP Algorithm**

Plan-Space Planning as a Search Problem

- given: statement of a planning problem $P=(O,s_i,g)$
- define the search problem as follows:
  - initial state: π_0 = ({init, goal},{(init$\prec$goal)},{},{})
  - goal test for plan state *p*: *p* has no flaws
  - path cost function for plan π: $|\pi|$
  - successor function for plan state *p*: refinements of *p* that maintain $\prec$ and *B*

PSP Procedure: Basic Operations

- PSP: Plan-Space Planner
- main principle: refine partial π plan while maintaining $\prec$ and B consistent until π has no more flaws
- basic operations:
  - find the flaws of π, i.e. its sub-goals and its threats
  - select one of the flaws
  - find ways to resolve the chosen flaw
  - choose one of the resolvers for the flaw
  - refine π according to the chosen resolver

PSP: Pseudo Code

function PSP(*plan*)

allFlaws ← *plan*.openGoals() + *plan*.threats()

if *allFlaws*.empty() **then return** *plan*

flaw ← *allFlaws*.selectOne()

allResolvers ← *flaw*.getResolvers(*plan*)

if *allResolvers*.empty() **then return** failure

resolver ← *allResolvers*.chooseOne()

newPlan ← *plan*.refine(*resolver*)

return PSP(*newPlan*)

Hierarchical Planning

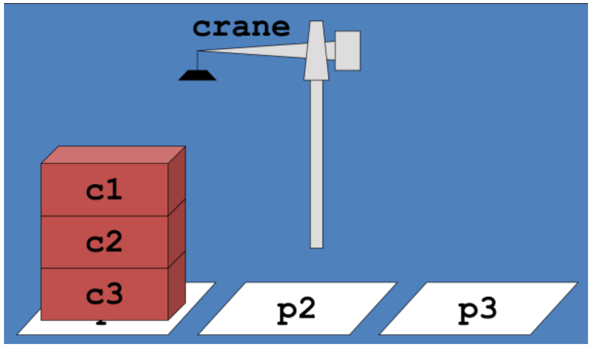
Hierarchical Task Network Planning (HTN)

STN Planning

- STN: Simple Task Network (is a simple version of HTN)
- what we know so far:
  - Terms, predicates, actions, state transition function, plans
- what's new:
  - tasks to be performed
  - methods describing ways in which tasks (subtasks) can be performed
  - organized collections of tasks (subtasks) called task networks

DWR Stack Moving Example

- task: move stack of containers from pallet p1 to pallet p3 in a way that preserves the order



- (informal) methods:
  - move topmost: take followed by put action
  - move stack: repeatedly move the topmost container until the stack is empty
  - move via intermediate: move stack to intermediate pile (reversing order) and then to final destination (reversing order again)

Tasks

- task symbols: $T_S = \{t_1,\ldots,t_n\}$
  - operator names $\subsetneq T_S$: primitive tasks
  - non-primitive task symbols: T_S - operator names
- task: $t_i(r_1,\ldots,r_k)$
  - t_i: task symbol (primitive or non-primitive)
  - $r_1,\ldots,r_k$: terms, objects manipulated by the task
  - ground task: are ground
- action $a=op(c_1,\ldots,c_k)$ accomplishes ground primitive task $t_i(r_1,\ldots,r_k)$ in state s iff
  - name(a) = t_i and $c_1 = r_1$ and … and $c_k = r_k$ and
  - a is applicable in s

Simple Task Networks

- A <u>simple task network</u> *w* is an acyclic directed graph (U,E) in which
  - the node set $U = \{t_1,\ldots,t_n\}$ is a set of tasks and
  - the edges in E define a partial ordering of the tasks in U.

- A task network w is <u>ground/primitive</u> if all tasks $t_u \in U$ are ground/primitive, otherwise it is unground/non-primitive.

Totally Ordered STNs

- ordering: $t_u \prec t_v$ in $w=(U,E)$ iff there is a path from t_u to t_v
- STN w is totally ordered iff E defines a total order on U
  - w is a sequence of tasks: $\langle t_1,\ldots,t_n \rangle$
- Let $w = \langle t_1,\ldots,t_n \rangle$ be a totally ordered, ground, primitive STN. Then the plan $\pi(w)$ is defined as:
  - $\pi(w) = \langle a_1,\ldots,a_n \rangle$ where $a_i = t_i$; $1 \leq i \leq n$

STNs: DWR Example

- tasks:
  - t_1 = take(crane1,loc1,c1,c2,p1): primitive, ground
  - t_2 = take(crane1,loc1,c2,c3,p1): primitive, ground
  - t_3 = move-stack(p1,q): non-primitive, unground
- task networks:
  - $w_1 = (\{t_1,t_2,t_3\}, \{(t_1,t_2), (t_1,t_3)\})$
    - partially ordered, non-primitive, unground
  - $w_2 = (\{t_1,t_2\}, \{(t_1,t_2)\})$
    - totally ordered: $w_2 = \langle t_1,t_2 \rangle$, ground, primitive
    - $\pi(w_2)$ = ⟨take(crane1,loc1,c1,c2,p1),take(crane1,loc1,c2,c3,p1)⟩

STN Methods

- Let M_S be a set of method symbols. An STN method is a 4-tuple m=(name(m),task(m),precond(m),network(m)) where:
  - name(m):
    - the name of the method
    - syntactic expression of the form $n(x_1,\ldots,x_k)$
      - $n \in M_S$: unique method symbol
      - $x_1,\ldots,x_k$: all the variable symbols that occur in m;
  - task(m): a non-primitive task;
  - precond(m): set of literals called the method's preconditions;
  - network(m): task network (U,E) containing the set of subtasks U of m.

STN Methods: DWR Example (1)

- move topmost: take followed by put action

- take-and-put(c,k,l,p_o,p_d,x_o,x_d)
  - task: move-topmost(p_o,p_d)
  - precond: top(c,p_o), on(c,x_o), attached(p_o,l), belong(k,l), attached(p_d,l), top(x_o,p_o), top(x_d,p_d)
  - subtasks: ⟨take(k,l,c,x_o,p_o),put(k,l,c,x_d,p_d)⟩

STN Methods: DWR Example (2)

- move stack: repeatedly move the topmost container until the stack is empty
- recursive-move(p_o,p_d,c,x_o)
  - task: move-stack(p_o,p_d)
  - precond: top(c,p_o), on(c,x_o)
  - subtasks: ⟨move-topmost(p_o,p_d), move-stack(p_o,p_d)⟩
- no-move(p_o,p_d)
  - task: move-stack(p_o,p_d)
  - precond: top(pallet,p_o)
  - subtasks: ⟨⟩

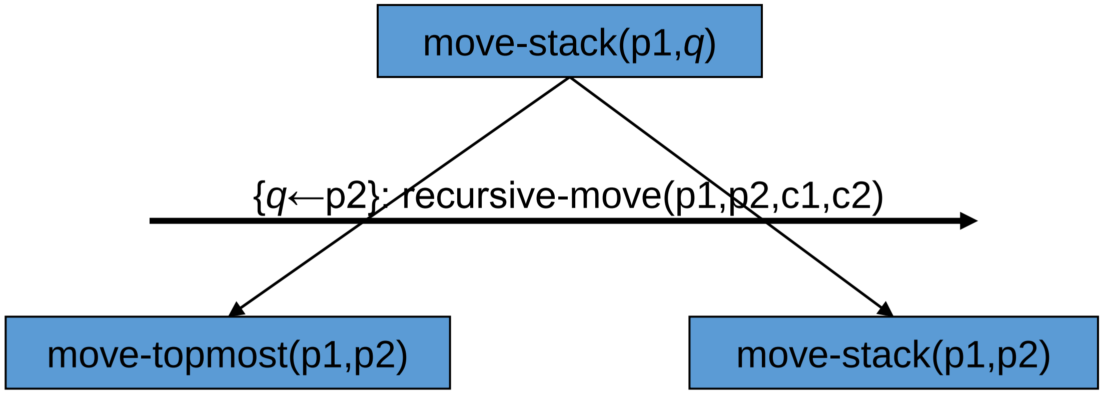
STN Methods: DWR Example (3)

- move via intermediate: move stack to intermediate pile (reversing order) and then to final destination (reversing order again)

- move-stack-twice(p_o,p_i,p_d)
  - task: move-ordered-stack(p_o,p_d)
  - precond: -
  - subtasks: ⟨move-stack(p_o,p_i),move-stack(p_i,p_d)⟩

Method Decomposition: DWR Example

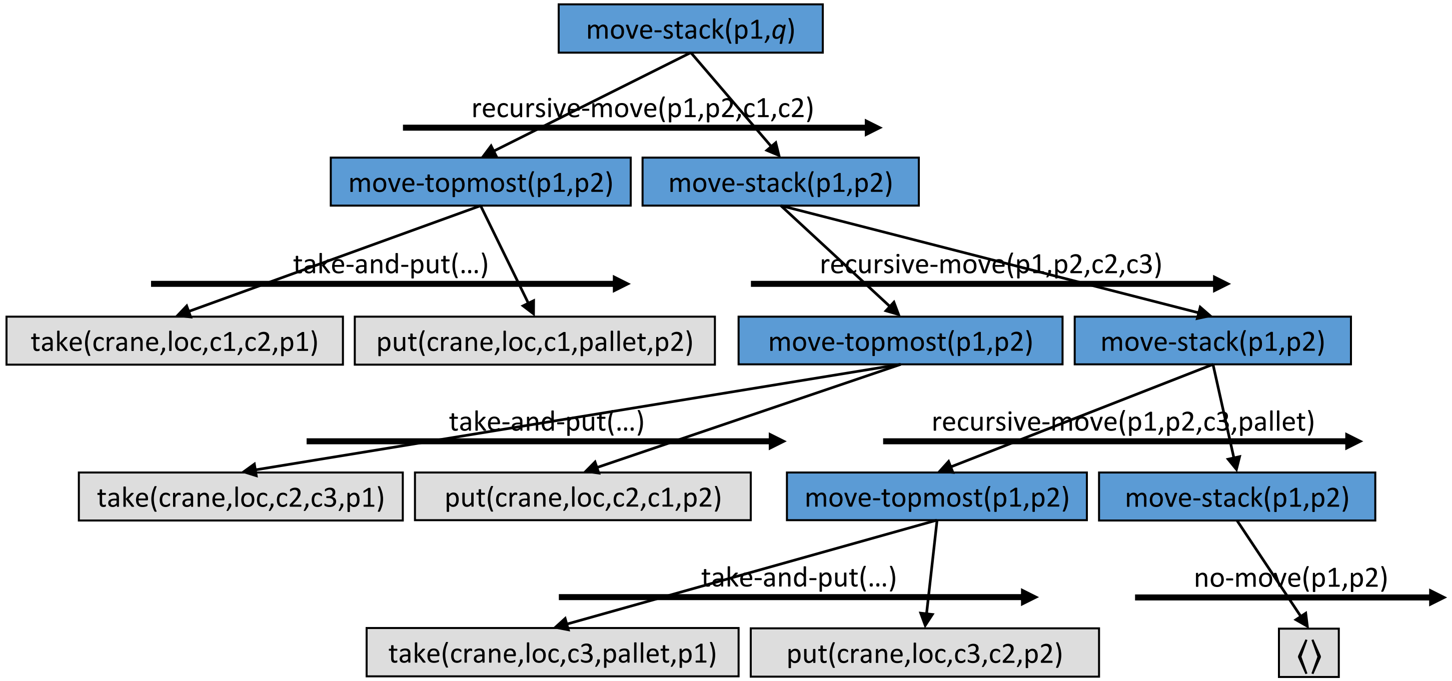
$\delta(t, m_i, \sigma) = \langle$move-topmost(p1,p2), move-stack(p1,p2)$\rangle$



Decomposition of Tasks in STNs

- Let
  - $w = (U,E)$ be a STN and
  - $t \in U$ be a task with no predecessors in w and
  - m a method that is relevant for t under some substitution σ with network(m) = (U_m,E_m).
- The <u>decomposition of t in w by m under σ</u> is the STN $\delta(w,t,m,\sigma)$ where:
  - t is replaced in U by $\sigma(U_m)$ and
  - edges in E involving t are replaced by edges to appropriate nodes in $\sigma(U_m)$.

Decomposition Tree: DWR Example



HTN vs. STRIPS Planning

- Since
  - HTN is generalization of STN Planning, and
  - STN problems can encode undecidable problems, but
  - STRIPS cannot encode such problems:
- **STN/HTN formalism is more expressive**
- non-recursive STN can be translated into equivalent STRIPS problem
  - but exponentially larger in worst case
- “regular” STN is equivalent to STRIPS

End