PLAN-SPACE PLANNING PAH CV5

PAH CV5

kopriva@agents.felk.cvut.cz

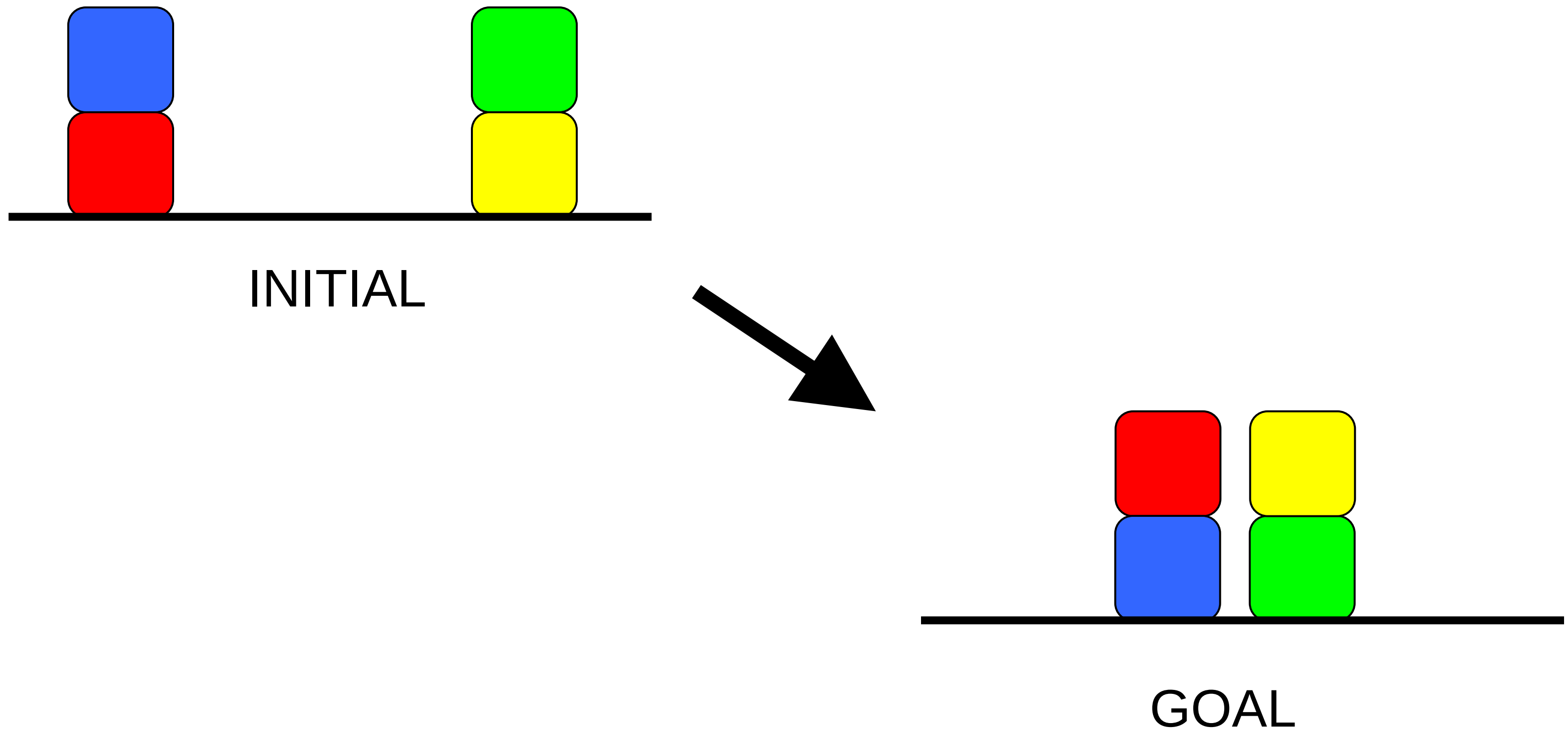
Total-order planning

The state-space planning technique produces **totally-ordered** plans, i.e. plans which consist of a strict sequence of actions.

Often, however, there are many possible orderings of actions than have equivalent effects.

Example

- Consider the planning problem:



Example

- There are many possible plans:

```
move(blue, red, table)
move(red, table, blue)
move(green, yellow, table)
move(yellow, table, green)
```

```
move(green, yellow, table)
move(yellow, table, green)
move(blue, red, table)
move(red, table, blue)
```

```
move(blue, red, table)
move(green, yellow, table)
move(red, table, blue)
move(yellow, table, green)
```

```
move(green, yellow, table)
move(blue, red, table)
move(red, table, blue)
move(yellow, table, green)
```

Example

These plans share some common structure. In fact, they are all different **interleavings** of two separate plans:

move(blue, red, table) move(red, table, blue)	move(green, yellow, table) move(yellow, table, green)

A **partial-order** plan is one which specifies only the necessary ordering information. One partial-order plan may have many total-orderings

Plan-space planning

Plan-space planning is a kind of approach to planning that produces partial-order plans.

It follows the **least-commitment principle**:

Do not add constraints (eg action ordering) to a plan until it becomes necessary to ensure the correctness of the plan.

Planning as plan-space search

A search through the space of plans.

Nodes in this search represent **incomplete plans** – plans with some steps missing.

Edges represent **refinements** – additional actions or constraints that can be added to make new plans.

Partial Order Planning

Planning as search:

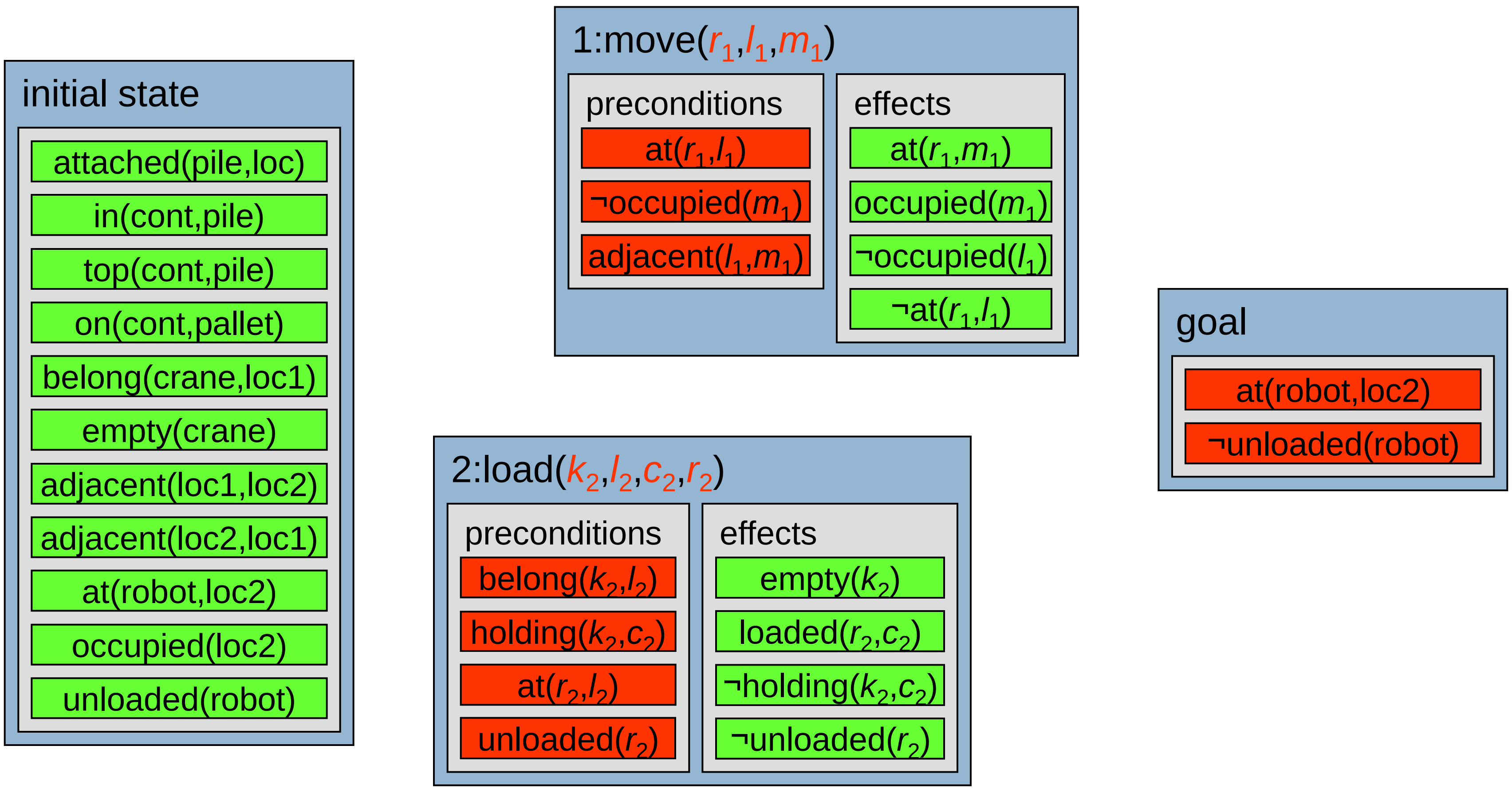
- Start with the empty plan
- While there are goals unsatisfied:
  - Pick an unsatisfied goal (**Generate**)
  - Add an action that satisfies it (**Select**)
  - Resolve conflicts (**Refine/Prune**)

Adding Actions

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

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

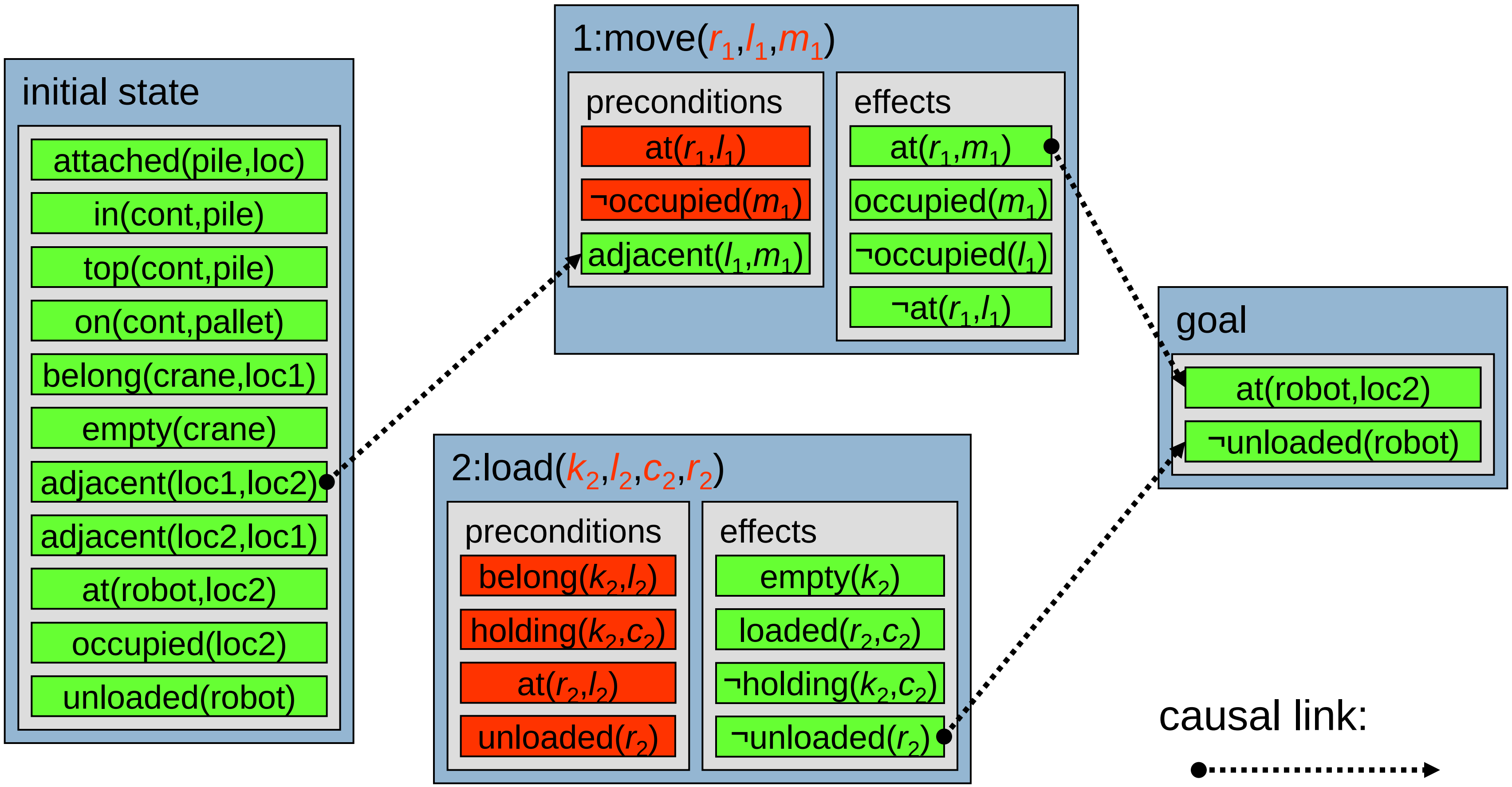
Adding Actions: Example



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

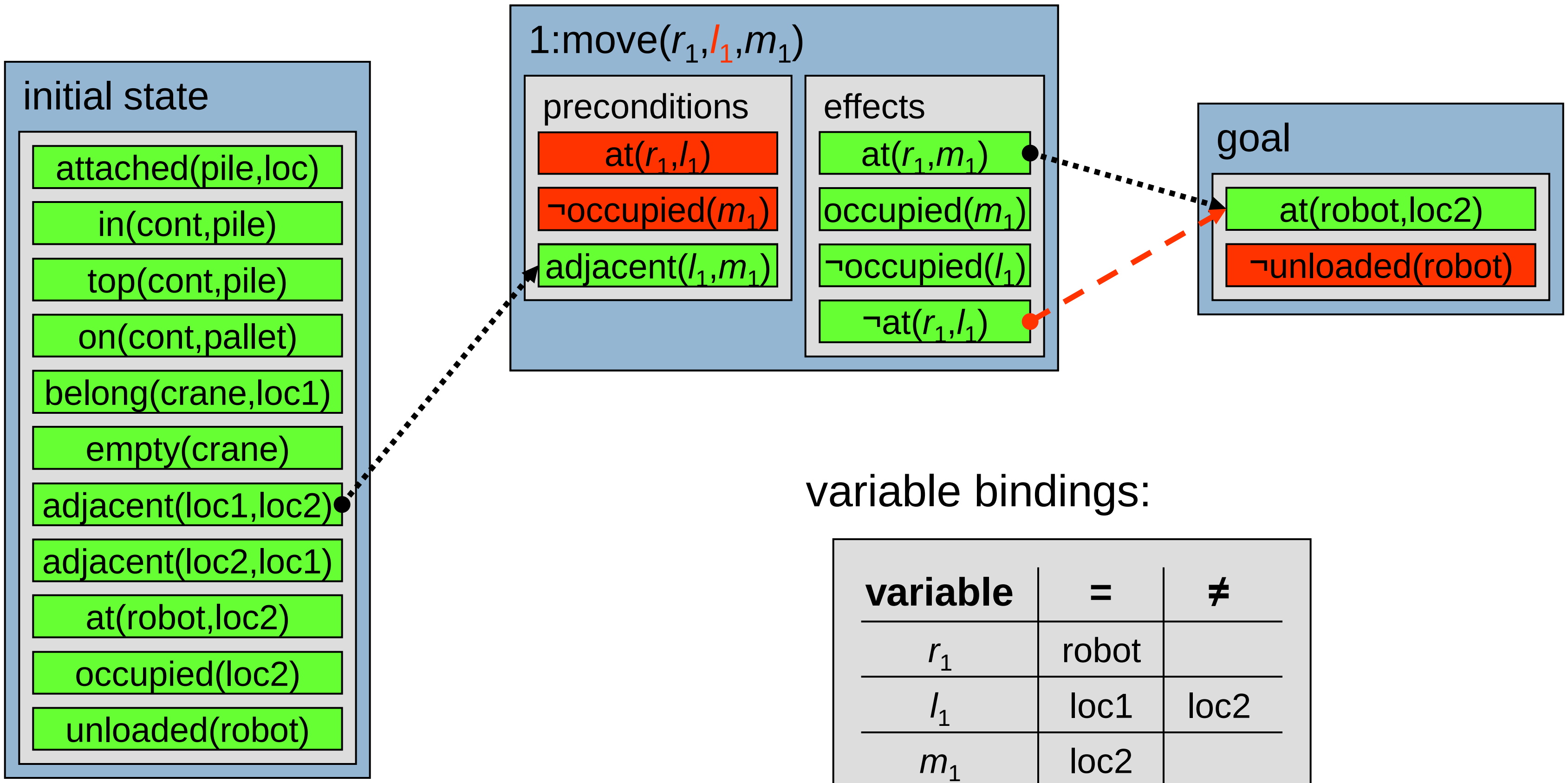
Adding Causal Links: Example



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



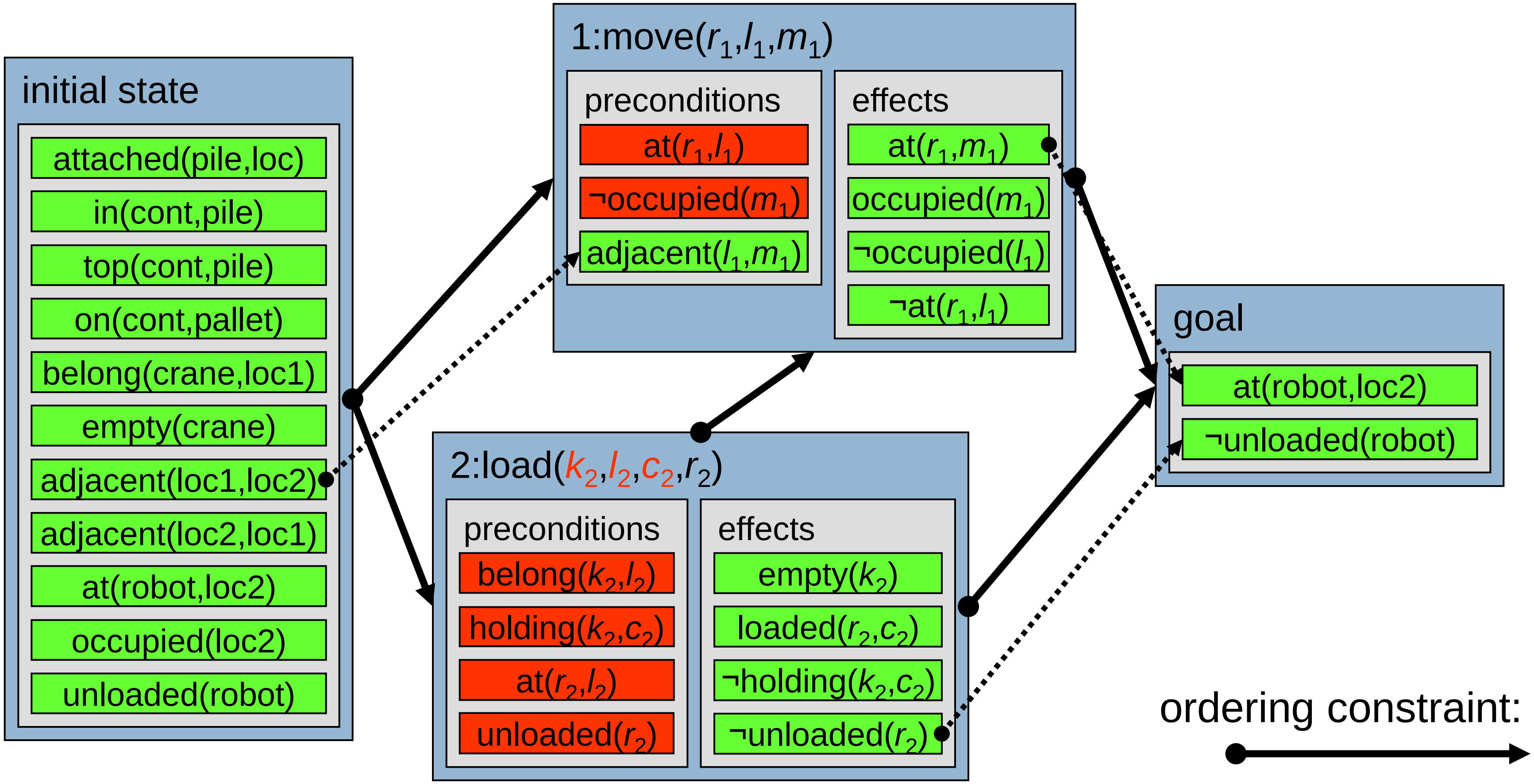
variable bindings:

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

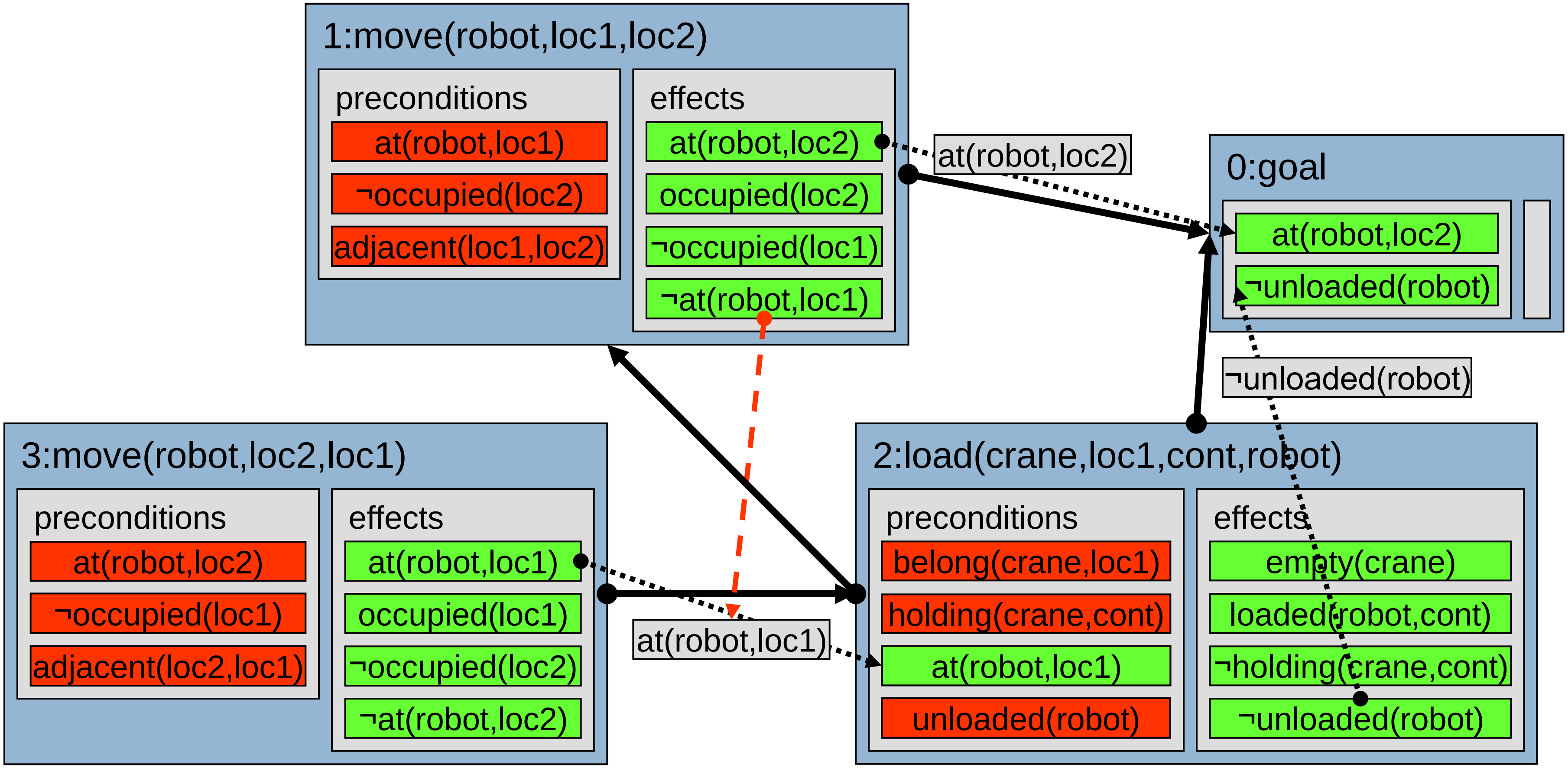
Adding Ordering Constraints

- 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



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.

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≺goal)},{},{})
  - goal test for plan state *p*: *p* has no flaws
  - path cost function for plan *π*: |*π*|
  - successor function for plan state *p*: refinements of *p* that maintain ≺ 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*)

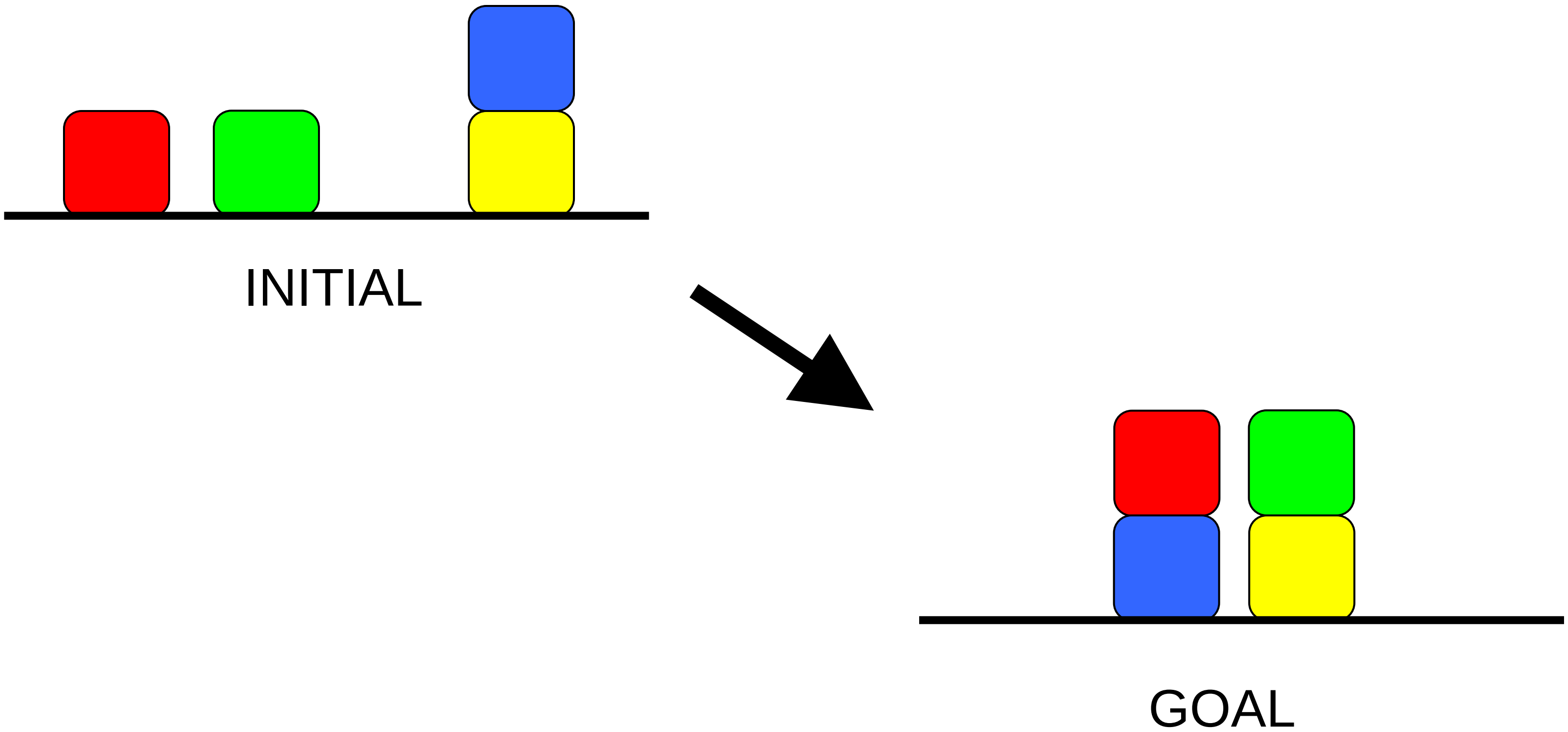
return PSP(*newPlan*)

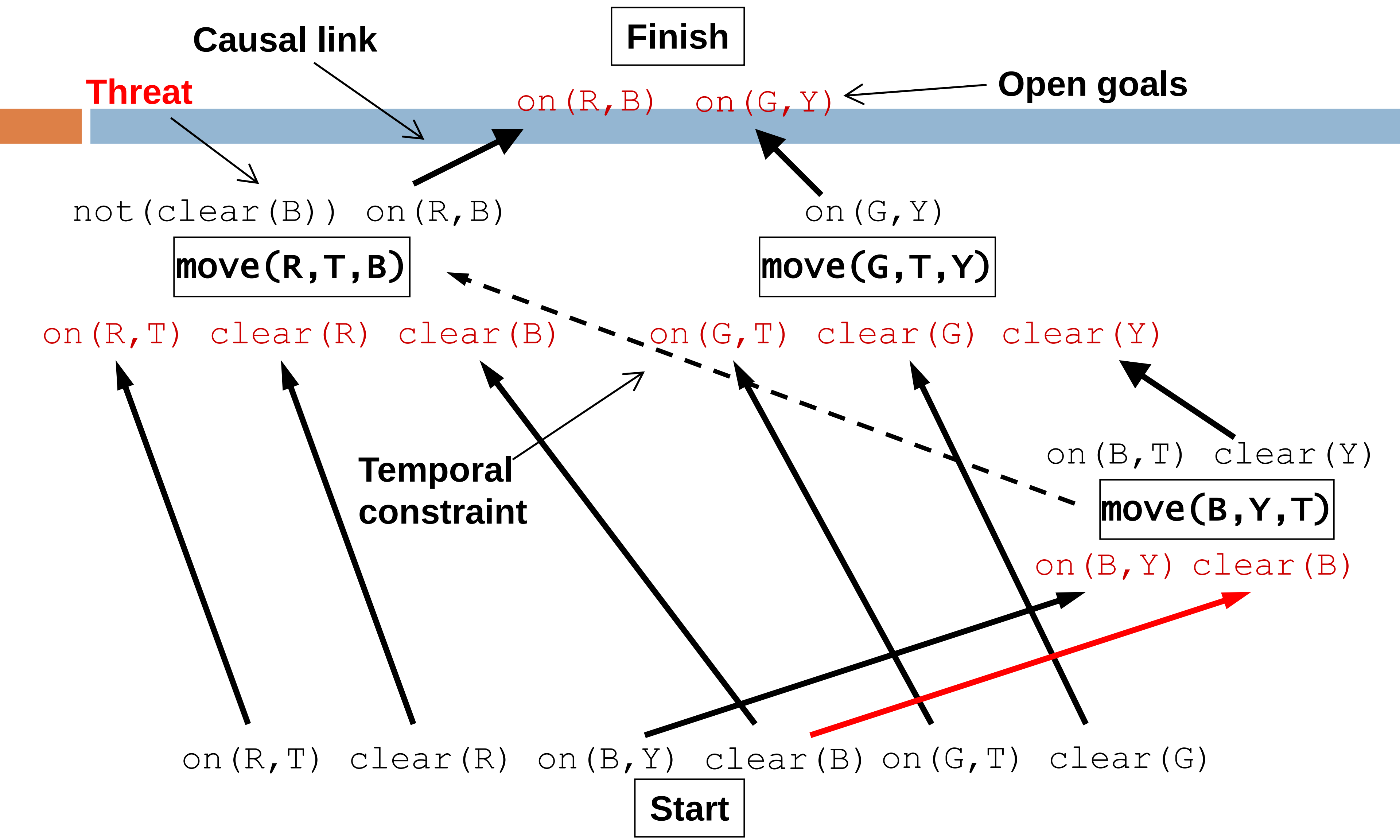
PSP: Choice Points

- *resolver* ← *allResolvers*.chooseOne()
  - non-deterministic choice
- *flaw* ← *allFlaws*.selectOne()
  - deterministic selection
  - all flaws need to be resolved before a plan becomes a solution
  - order not important for completeness
  - order is important for efficiency

Partial Order Planning Example

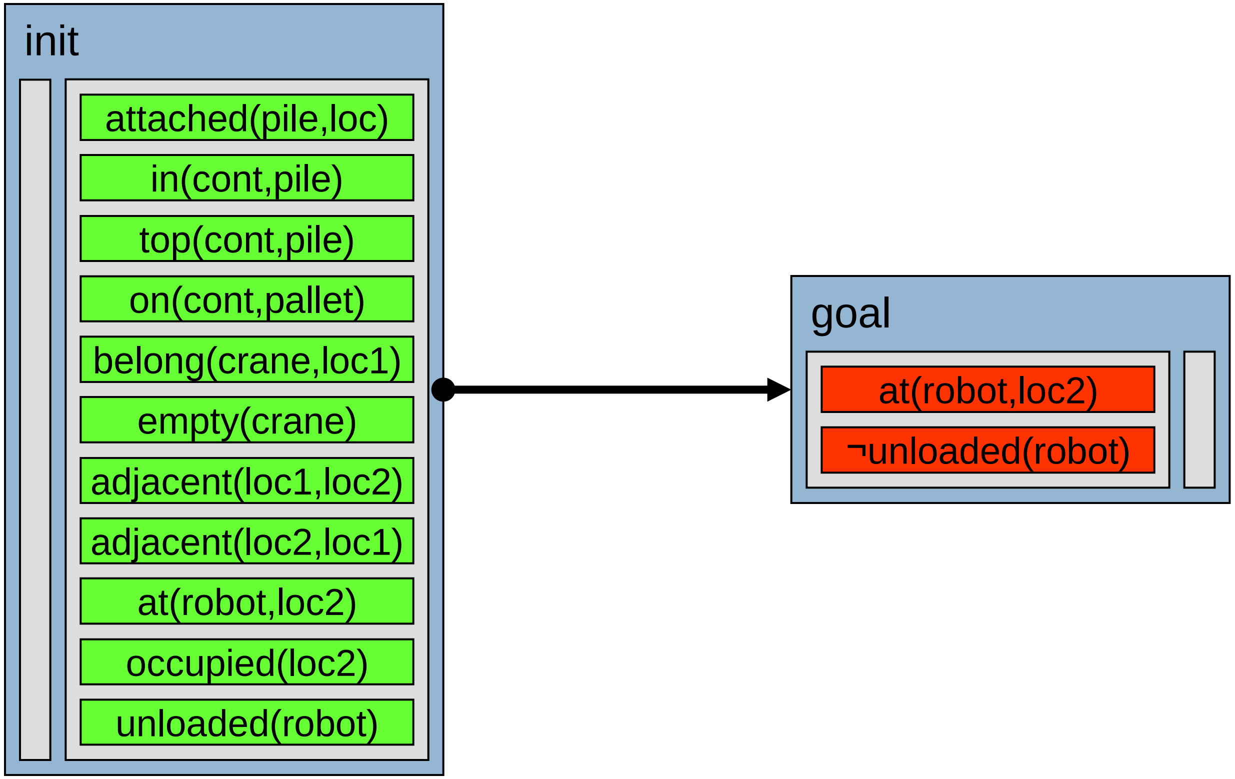
- Four block problem:





Finish
Causal link
Threat
on(R,B) on(G,Y)
Open goals
not(clear(B)) on(R,B)
on(G,Y)
move(R,T,B)
move(G,T,Y)
on(R,T) clear(R) clear(B)
on(G,T) clear(G) clear(Y)
on(B,T) clear(Y)
Temporal
constraint
move(B,Y,T)
on(B,Y) clear(B)
on(R,T) clear(R) on(B,Y) clear(B) on(G,T) clear(G)
Start

Plan-Space Search: Initial Search State Example



PSP Implementation: PoP

- extended input:
  - partial plan (as before)
  - agenda: set of pairs (*a*,*p*) where *a* is an action an *p* is one of its preconditions
- search control by flaw type
  - unachieved sub-goal (on agenda): as before
  - threats: resolved as part of the successor generation process

PoP: Pseudo Code (1)

function PoP(*plan*, *agenda*)

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

  $(a_g, p_g) \leftarrow$ *agenda*.selectOne()

  agenda $\leftarrow$ *agenda* - (a_g, p_g)

  relevant $\leftarrow$ *plan*.getProviders(p_g)

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

  $(a_p, p_p, \sigma) \leftarrow$ *relevant*.chooseOne()

  plan.L $\leftarrow$ *plan.L* $\cup \langle a_p -[p]\rightarrow a_g \rangle$

  plan.B $\leftarrow$ *plan.B* $\cup\ \sigma$

PoP: Pseudo Code (2)

if $a_p \notin plan.A$ **then**
    $plan$.add(a_p)
    $agenda \leftarrow agenda + a_p.preconditions$
$newPlan \leftarrow plan$
for each *threat* on $\langle a_p -[p] \rightarrow a_g \rangle$ **or** due to a_p **do**
    allResolvers ← *threat*.getResolvers(*newPlan*)
    if *allResolvers*.empty() **then return** failure
    resolver ← *allResolvers*.chooseOne()
    newPlan ← *newPlan*.refine(*resolver*)
return PSP(*newPlan*,*agenda*)