AI Planning.
Plan-space planning

Eva Onaindia

Universitat Politècnica de València

Acknowledgements

Most of the slides used in this course are taken or are modifications from Dana Nau's lecture slides for the textbook *Automated Planning,* licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License:

http://creativecommons.org/licenses/by-nc-sa/2.0/
http://creativecommons.org/licenses/by/3.0/es/

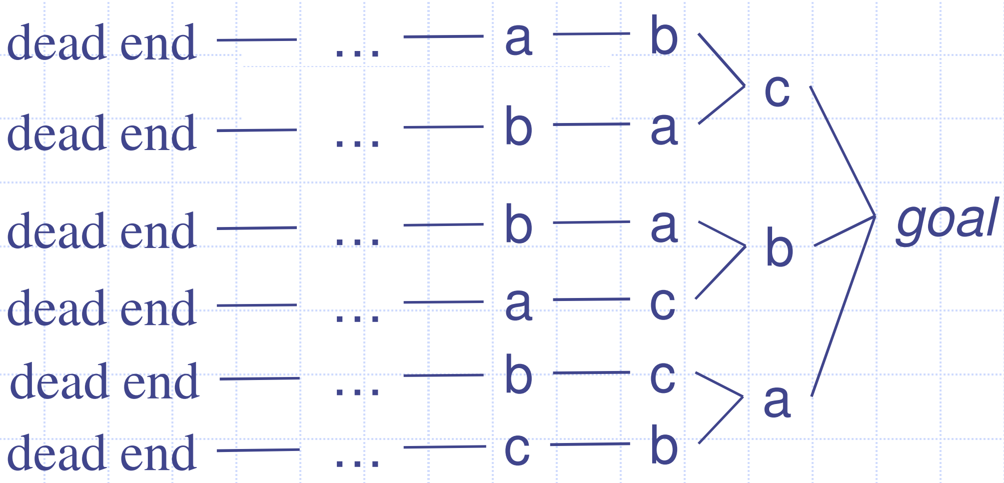
I would like to gratefully acknowledge Dana Nau's contributions and thank him for generously permitting me to use aspects of his presentation material.

Plan-space planning. Outline

- Motivation
- POP tree (plan space search)
- Structure of a partial plan
  - Example *transportation*
- The PSP algorithm
- Example
- POP and the Sussman anomaly
- Summary
- Heuristics in plan-space planning
  - Flaw-selection heuristics
  - Resolver-selection heuristics

Plan-space planning. Motivation.

- Problem with state-space search
  - In some cases we may try many different orderings of the same actions before realizing there is no solution



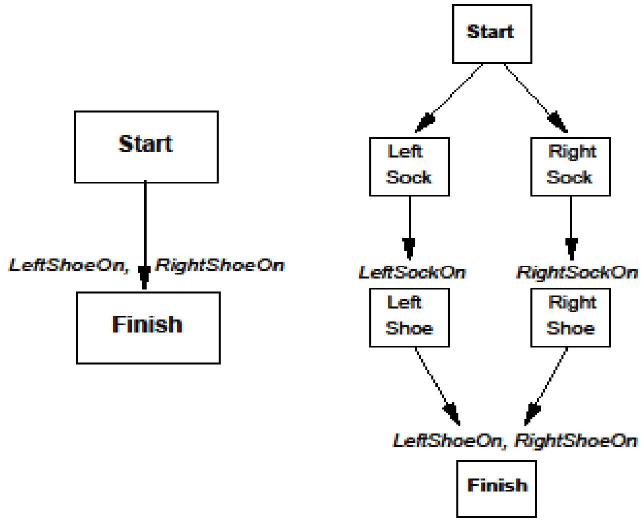
- *Least-commitment strategy:* don't commit to orderings, instantiations, etc., until necessary

Plan-space planning. Motivation.

- State-space planning:
  - Actions are planned in the same order as they will be later executed
  - Sequential plans (total order)
  - Multiple relationships among actions. Commiting to an specific ordering may yield a non-efficient planning process
- Plan-space planning:
  - Not commitment to a particular action ordering until necessary
  - Plans with parallel actions (POP: partial-order planning)

Plan-space planning. Motivation.

Partially ordered plans



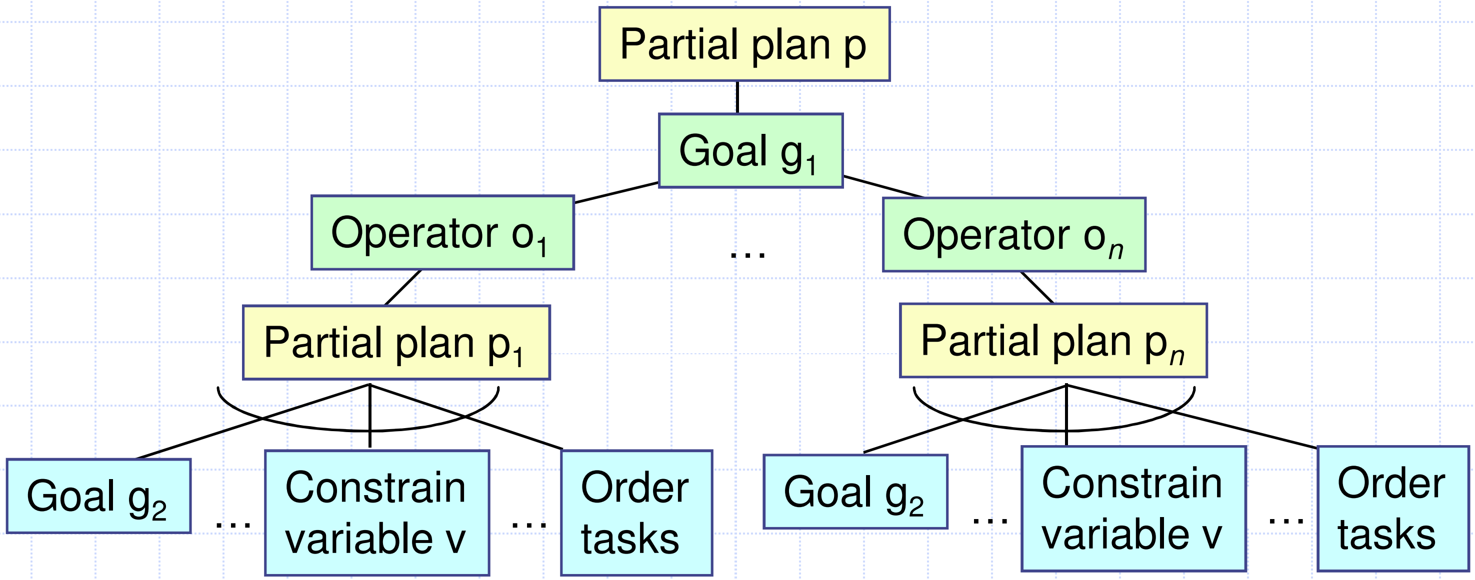
A plan is complete iff every precondition is achieved

A precondition is achieved iff it is the effect of an earlier step and no possibly intervening step undoes it

POP tree (plan space search)

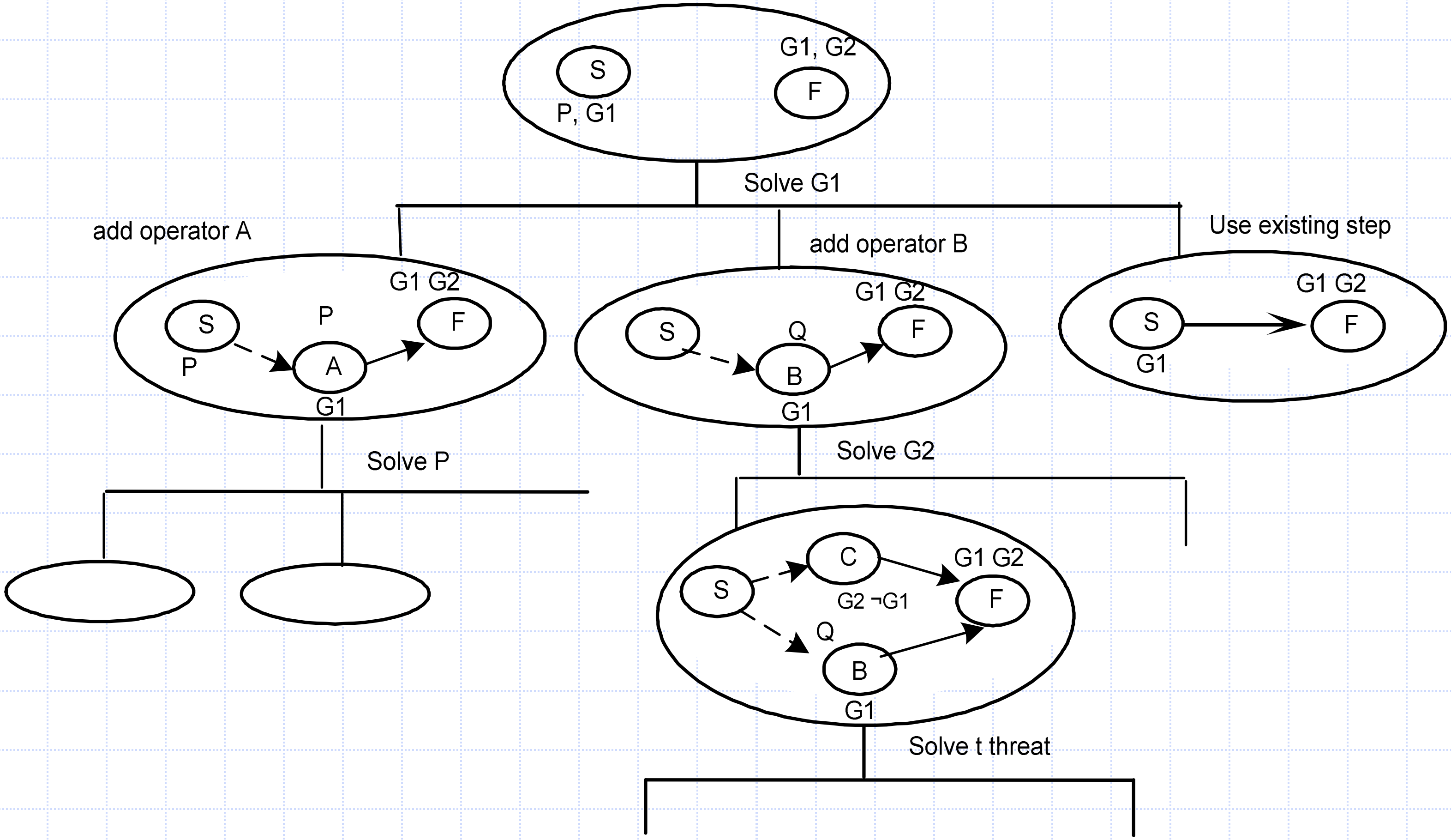
- Backward search from the goal
- Each node of the search space is a *partial plan (graph of actions)*
  - A set of partially-instantiated actions
  - A set of constraints
  - Make more and more refinements, until we have a solution

POP tree (plan space search)



Nodes in the POP tree are partial plans (yellow boxes)

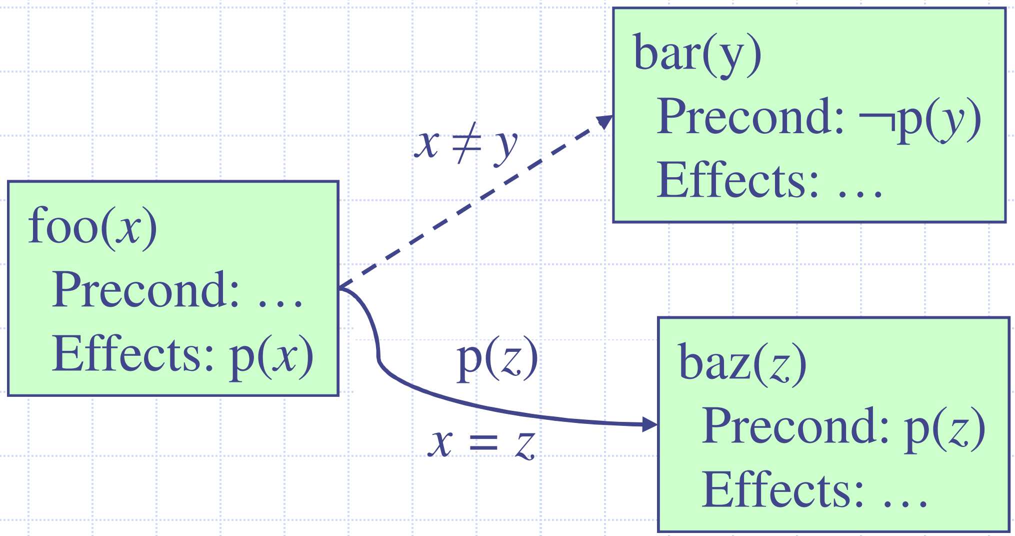
POP tree (plan space search)



Structure of a partial plan

- Plan step: totally or partially-instantiated operator (action)
  - Start step: initial state
  - Finish step: problem goals
- Types of constraints between steps:
  - *causal link*:
    - use action *a* to establish the precondition *p* needed by action *b* *<a,p,b>*
  - *precedence constraint*: *a* must precede *b* *(a < b)*
  - *binding constraints*:
    - inequality constraints, e.g., $v_1 \neq v_2$ or $v \neq c$
    - equality constraints (e.g., $v_1 = v_2$ or $v = c$) and/or substitutions

Structure of a partial plan (example 1)



Causal link: <foo(x), p(x), bar(z)>
Binding/equality constraint: x=z
Precedence constraint: foo(x) < bar(y)
Inequality constraint: x≠y

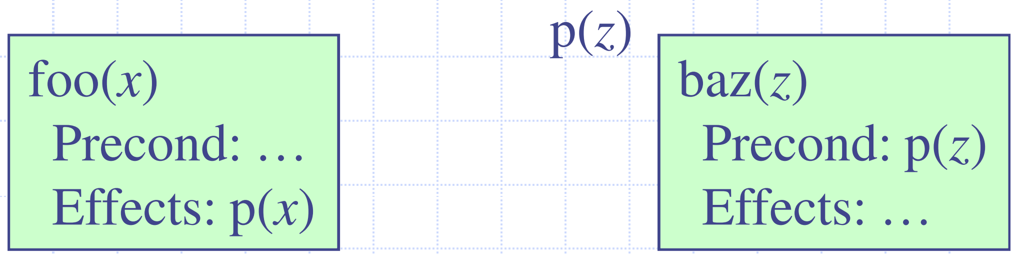
Structure of a partial plan (plan refinement)

- Plan refinement: selecting a *flaw* in the plan and branching all over ways of solving the flaw
- Flaw:
  - open goal: selecting a pending subgoal to be solved in the plan
  - binding constraint: selecting a non-instantiated variable
  - threat: selecting a step that *threatens* a causal link
    - causal link: <p1,l,p2>
    - $\exists$ p3/ l $\in$ effects$^-$(p3)
    - p3 is unordered with respect to p1 and p2
      - - promotion: p3 < p1
      - - demotion: p2 < p3
      - - separation: constraint variables to avoid the threat

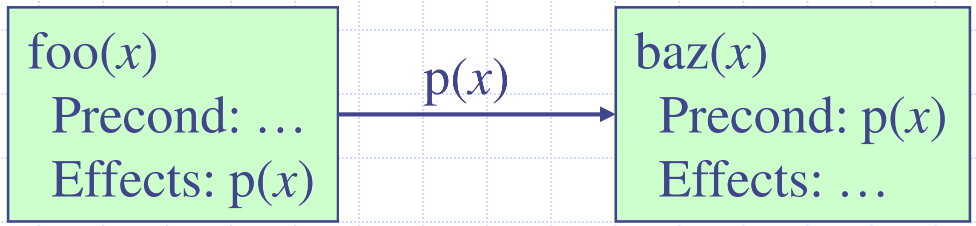
- How to tell we have a solution: no more *flaws* in the plan

Flaws: open goals

- Open goal:
  - An action/step *a* has a precondition *p* that we haven't decided how to establish

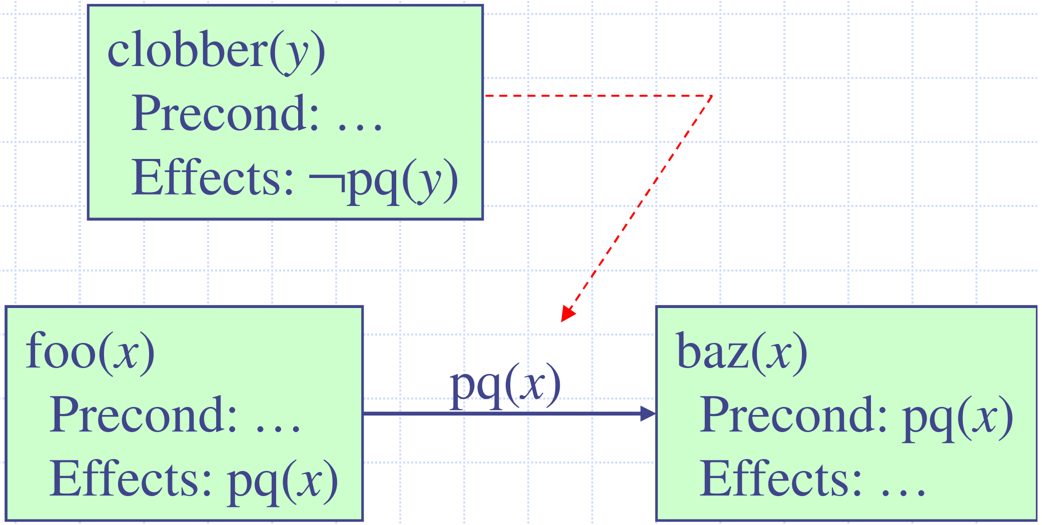


- Resolving the flaw:
  - Find an action/step *b*
    - (either already in the plan, or insert it)
  - that can be used to establish *p*
    - can precede *a* and produce *p*
  - Instantiate variables and/or constrain variable bindings
  - Create a causal link



Flaws: threats

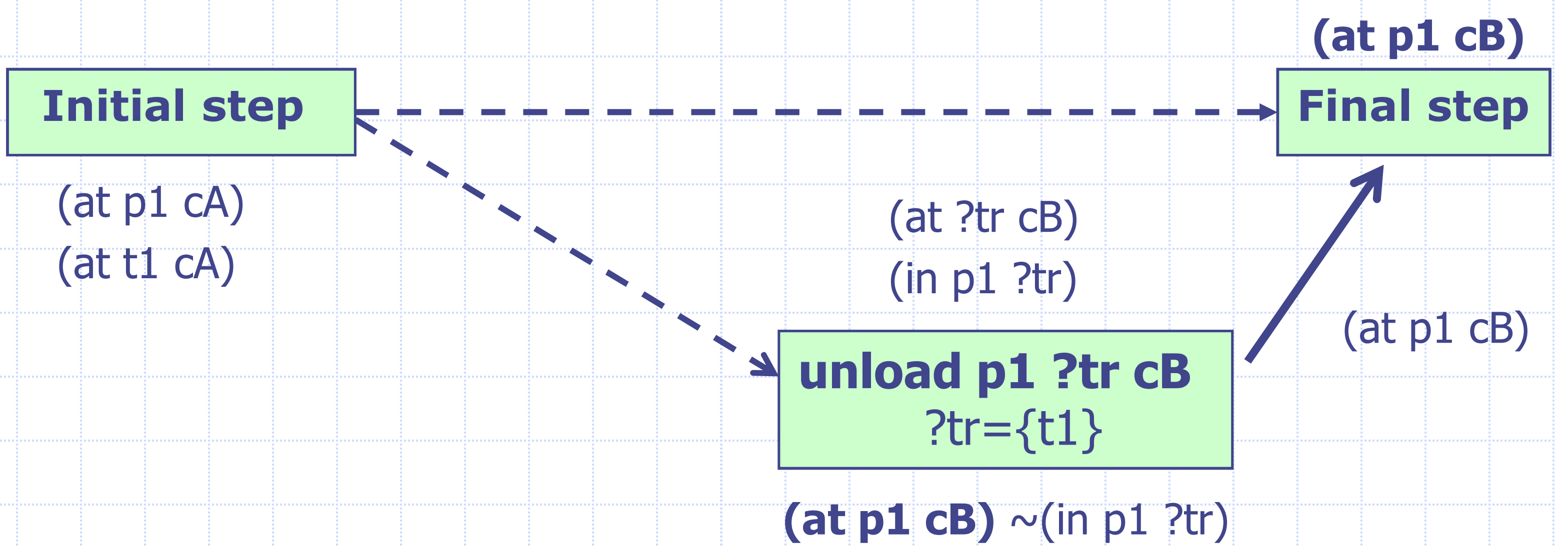
- Threat: a deleted-condition interaction
  - Action/step *a* establishes a precondition (e.g., pq(*x*)) of action *b*
  - Another action/step *c* is capable of deleting *p*
- Resolving the flaw:
  - impose a constraint to prevent *c* from deleting *p*
- Three possibilities:
  - Make *b* precede *c*
  - Make *c* precede *a*
  - Constrain variable(s) to prevent *c* from deleting *p*



Example *transportation* (partial plans)

Plan 1: initial plan

Plan 2: refinement plan over plan 1



Example *transportation* (POP tree)

Plan 1

flaw: goal (at p1 cB)

resolvers of the flaw: operator (unload p1 ?tr cB)

Plan 2

flaws:
- variables: ?tr
- open goals: (at ?tr cB), (in p1 ?tr)

flaw: variable ?tr

resolvers of the flaw: ?tr=t1

Plan 2.1

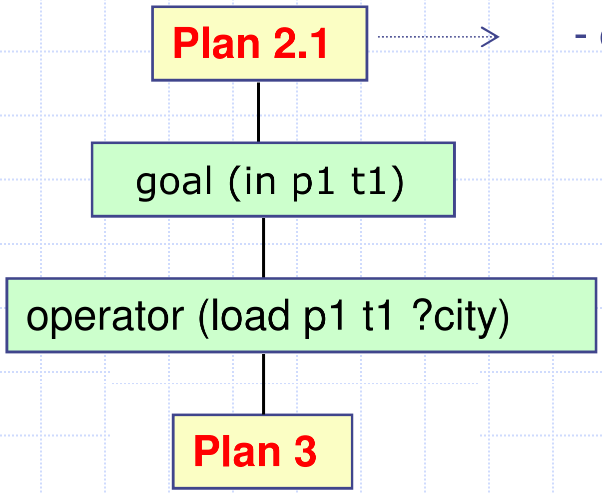
Flaws:
- open goals: (at t1 cB), (in p1 t1)

Example *transportation* (POP tree cont.)

Flaws:

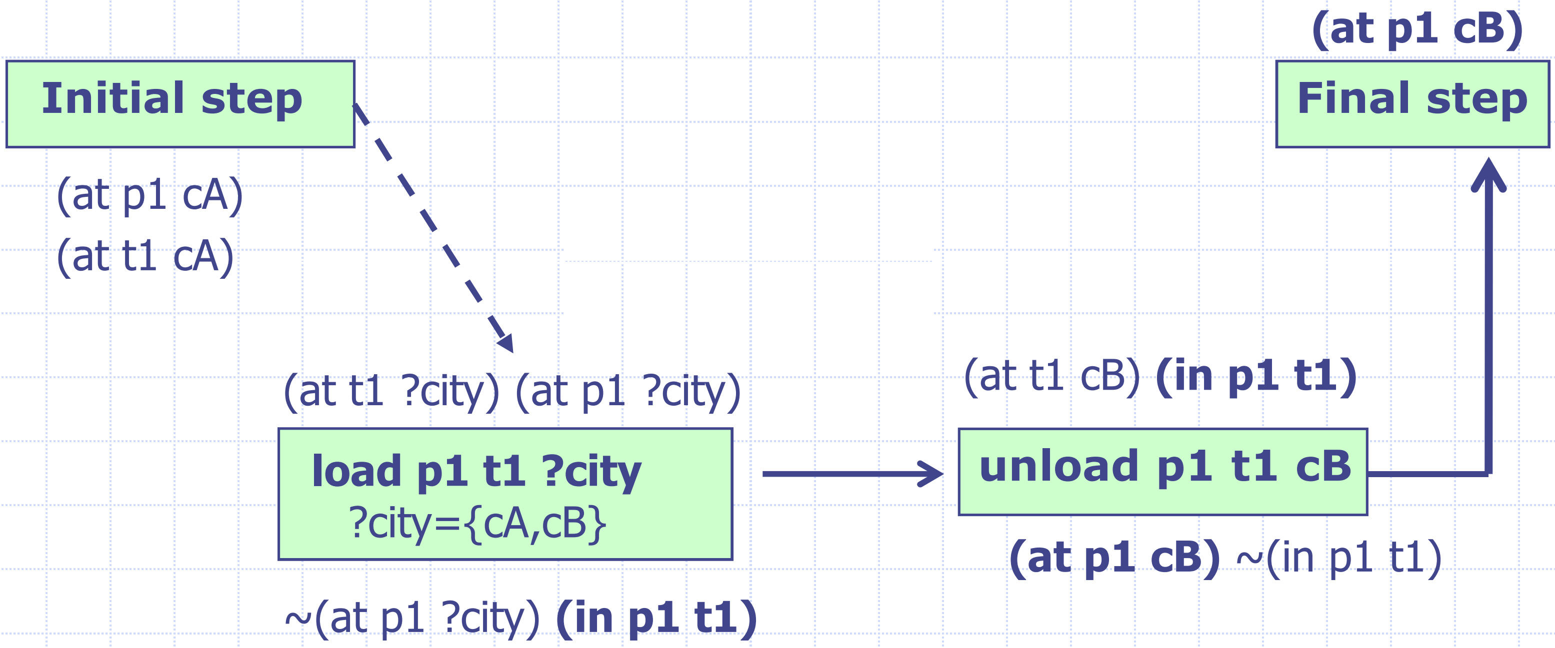
- open goals: (at t1 cB), (in p1 t1)

resolvers of the flaw:



Example *transportation* (partial plans cont.)

Plan 3



Example *transportation* (POP tree cont.)

Plan 2.1

Flaws Plan 2.1:
- open goals: (at t1 cB), (in p1 t1)

goal (in p1 t1)

resolvers of the flaw:

operator (load p1 t1 ?city)

Plan 3

Flaws Plan 3:
- variables: ?city
- open goals: (at t1 cB), (at t1 ?city), (at p1 ?city)

goal (at t1 cB)

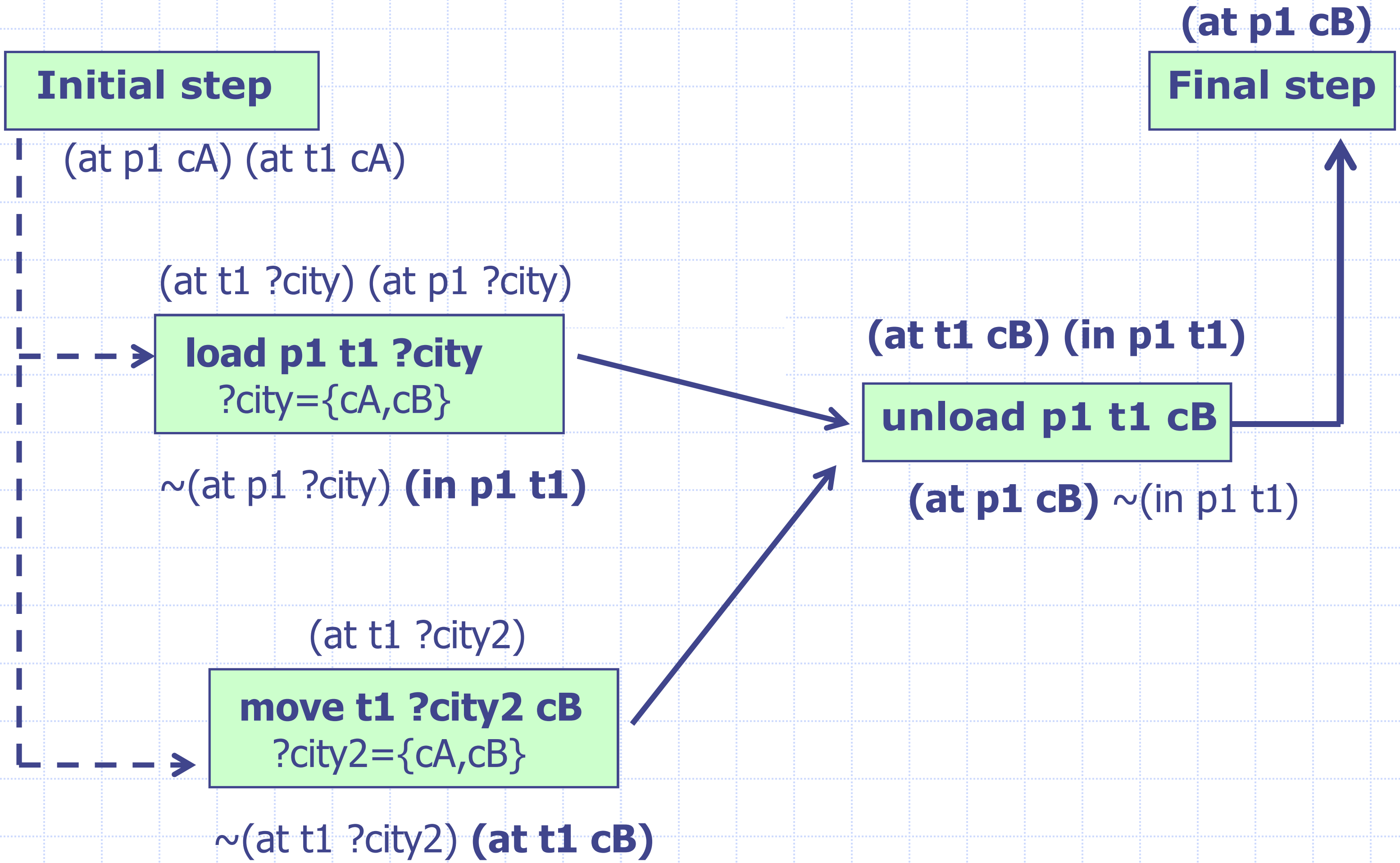
resolvers of the flaw:

operator (move t1 ?city2 cB)

Plan 4

Example *transportation* (partial plans cont.)

Plan 4

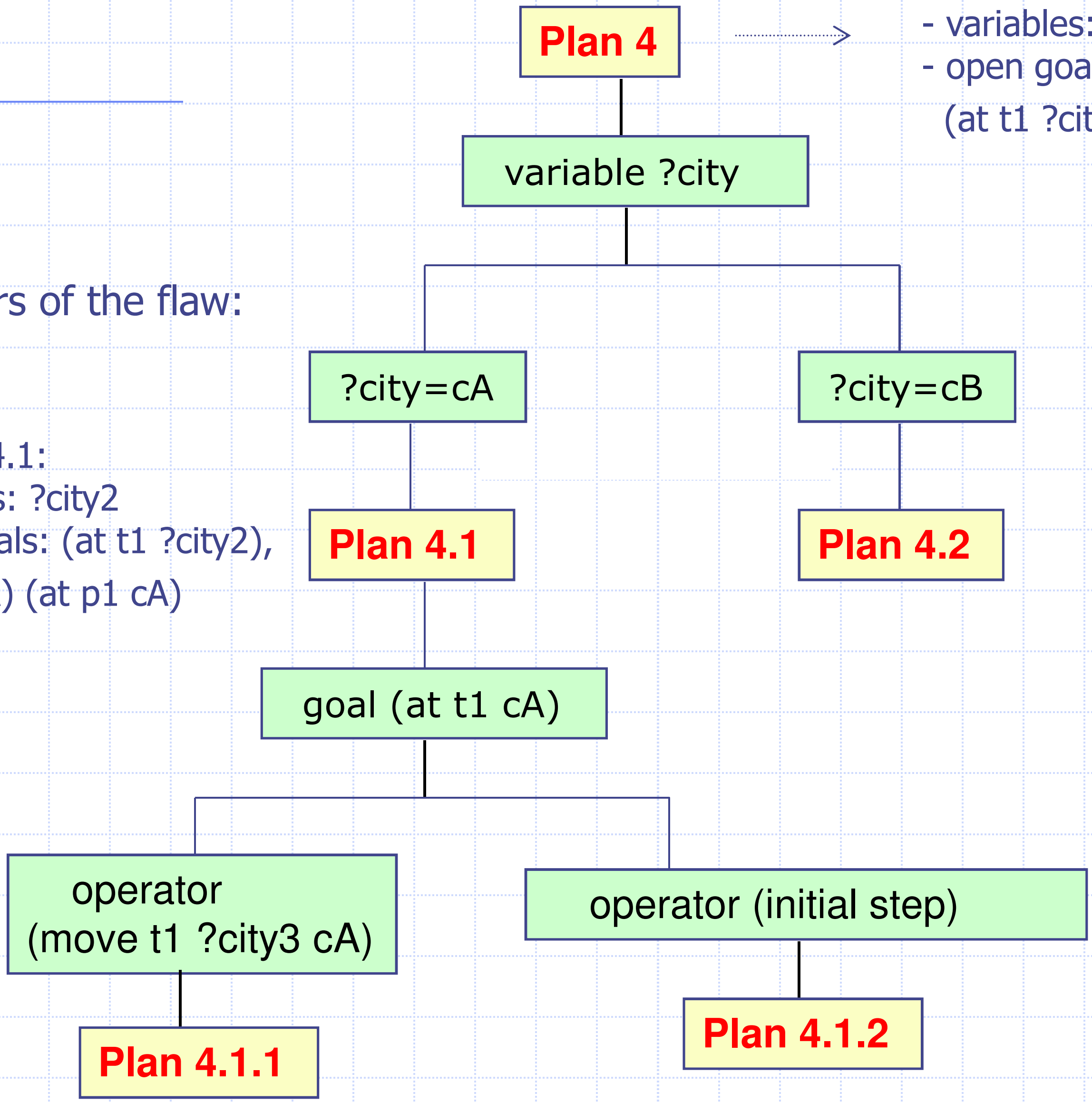


Example *transportation* (POP tree cont.)

Flaws Plan 4:
- variables: ?city, ?city2
- open goals: (at t1 ?city2), (at t1 ?city) (at p1 ?city)

resolvers of the flaw:

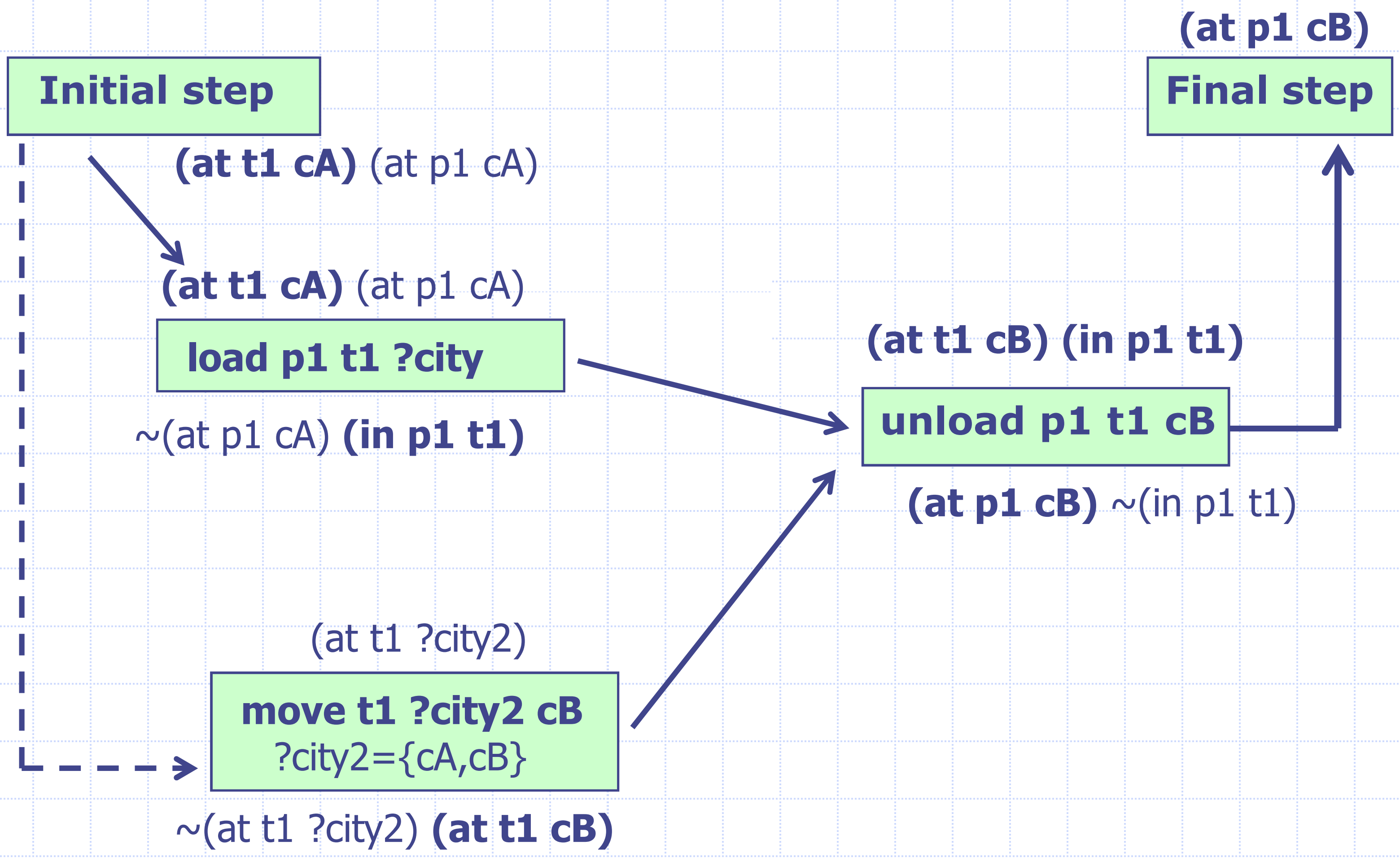
Flaws Plan 4.1:
- variables: ?city2
- open goals: (at t1 ?city2), (at t1 cA) (at p1 cA)



resolvers of the flaw:

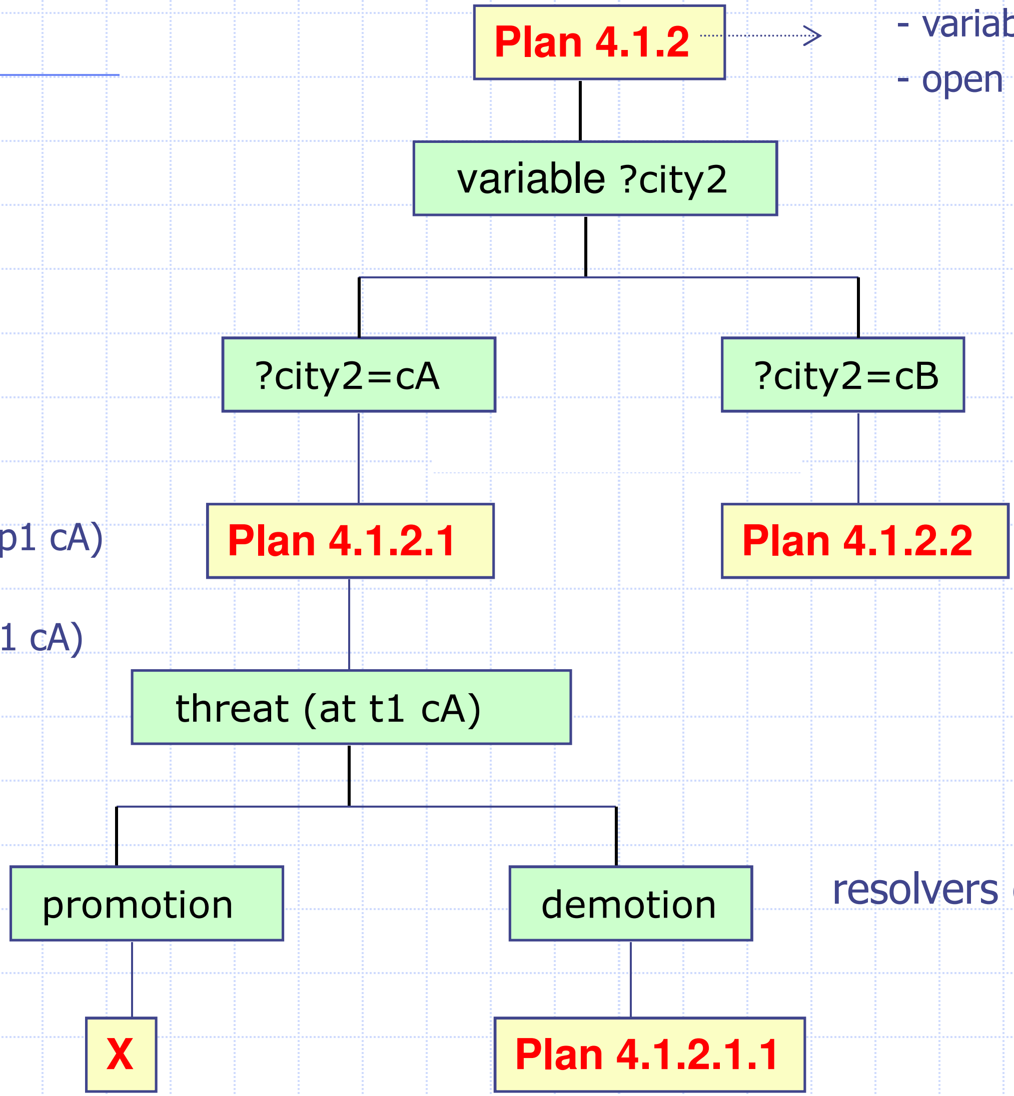
Example *transportation* (partial plans cont.)

Plan 4.1.2



Example *transportation* (POP tree cont.)

Flaws Plan 4.1.2:
- variables: ?city2
- open goals: (at p1 cA), (at t1 ?city2)



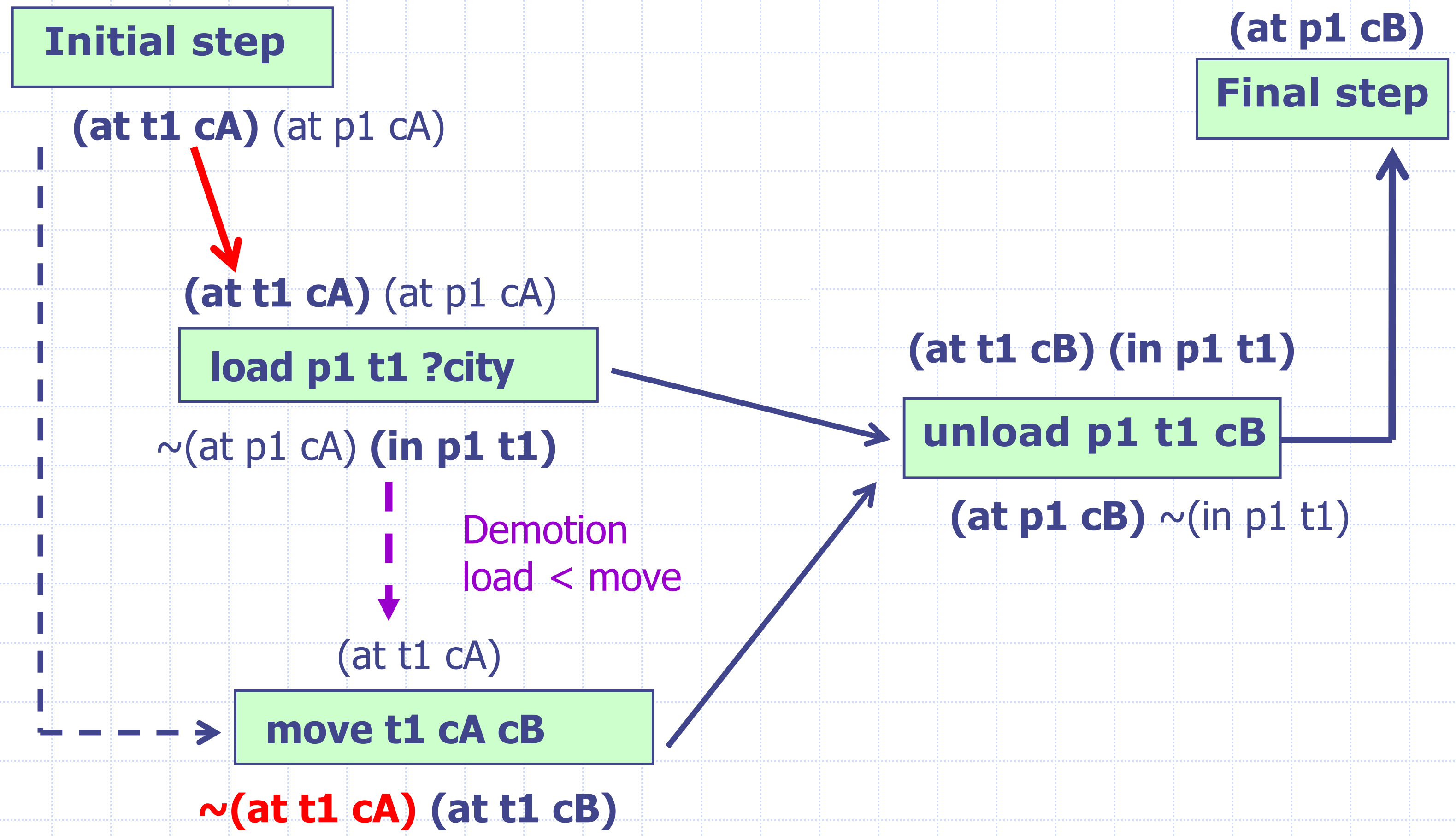
resolvers of the flaw

Flaws Plan 4.1.2.1:
- open goals: (at p1 cA)
  (at t1 cA)
- threat over (at t1 cA)

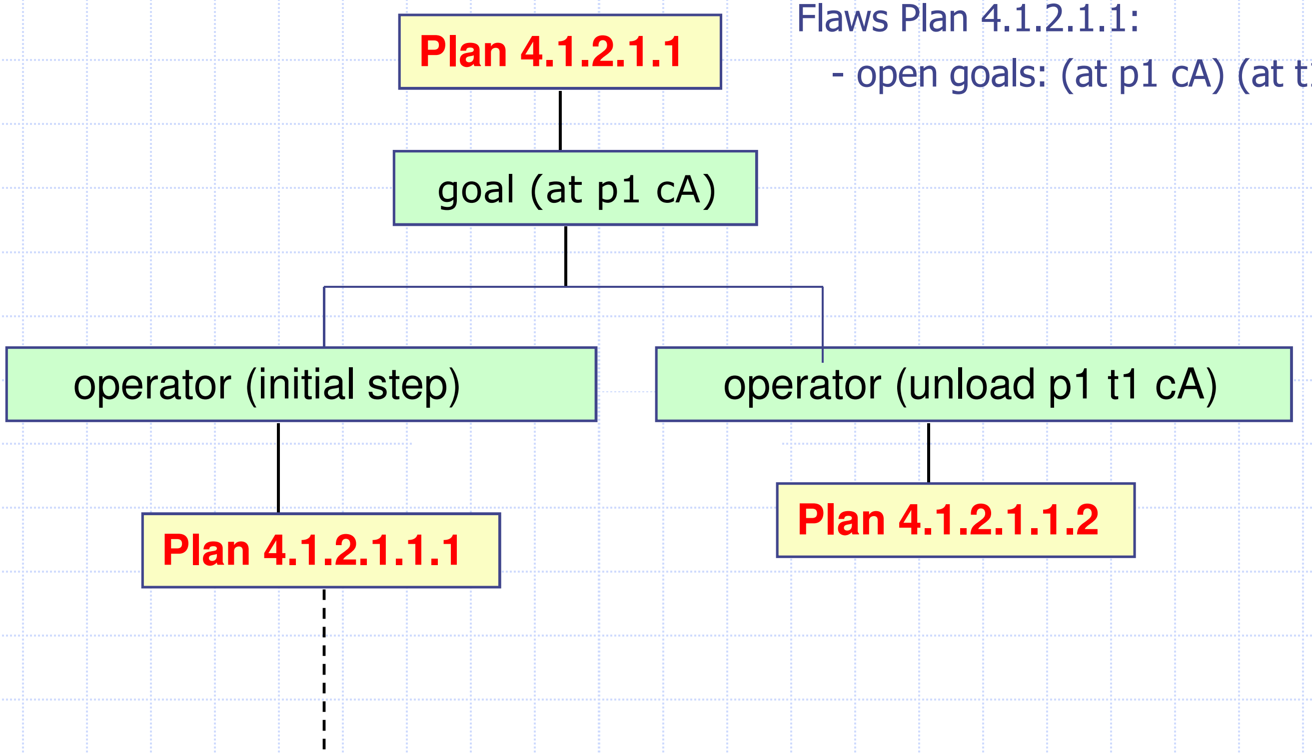
resolvers of the flaw

Example *transportation* (partial plans cont.)

Plan 4.1.2.1.1



Example *transportation* (POP tree cont.)

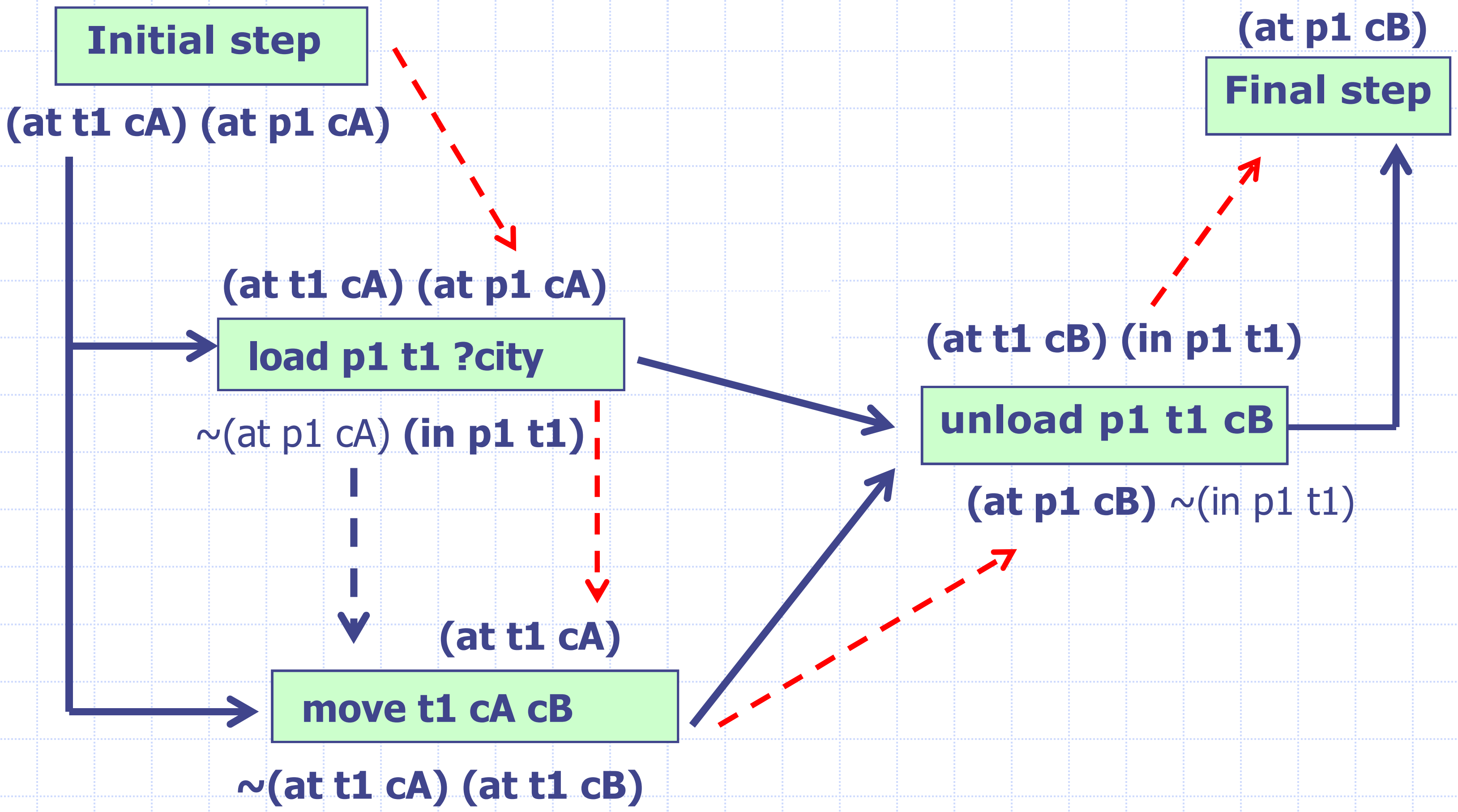


Flaws Plan 4.1.2.1.1:

- open goals: (at p1 cA) (at t1 cA)

Example *transportation* (partial plans cont.)

Plan 4.1.2.1.1.1



The PSP Procedure

```
PSP(π)
    flaws ← OpenGoals(π) ∪ Threats(π)
    if flaws = ∅ then return(π)
    select any flaw φ ∈ flaws
    resolvers ← Resolve(φ, π)
    if resolvers = ∅ then return(failure)
    nondeterministically choose a resolver ρ ∈ resolvers
    π′ ← Refine(ρ, π)
    return(PSP(π′))
end
```

- PSP is both sound and complete
- It returns a partially ordered solution plan
  - Any total ordering of this plan will achieve the goals
  - Or could execute actions in parallel if the environment permits it

Example

- Similar (but not identical) to an example in Russell and Norvig's *Artificial Intelligence: A Modern Approach* (1st edition)
- Operators:
  - **Start**

    Precond: none

    Effects: At(Home), sells(HWS,Drill), Sells(SM,Milk), Sells(SM,Banana)
  - **Finish**

    Precond: Have(Drill), Have(Milk), Have(Banana), At(Home)

    Effects: none
  - **Go(*l*,*m*)**

    Precond: At(*l*)

    Effects: At(*m*), ¬At(*l*)
  - **Buy(*p*,*s*)**

    Precond: At(*s*), Sells(*s*,*p*)

    Effects: Have(*p*)

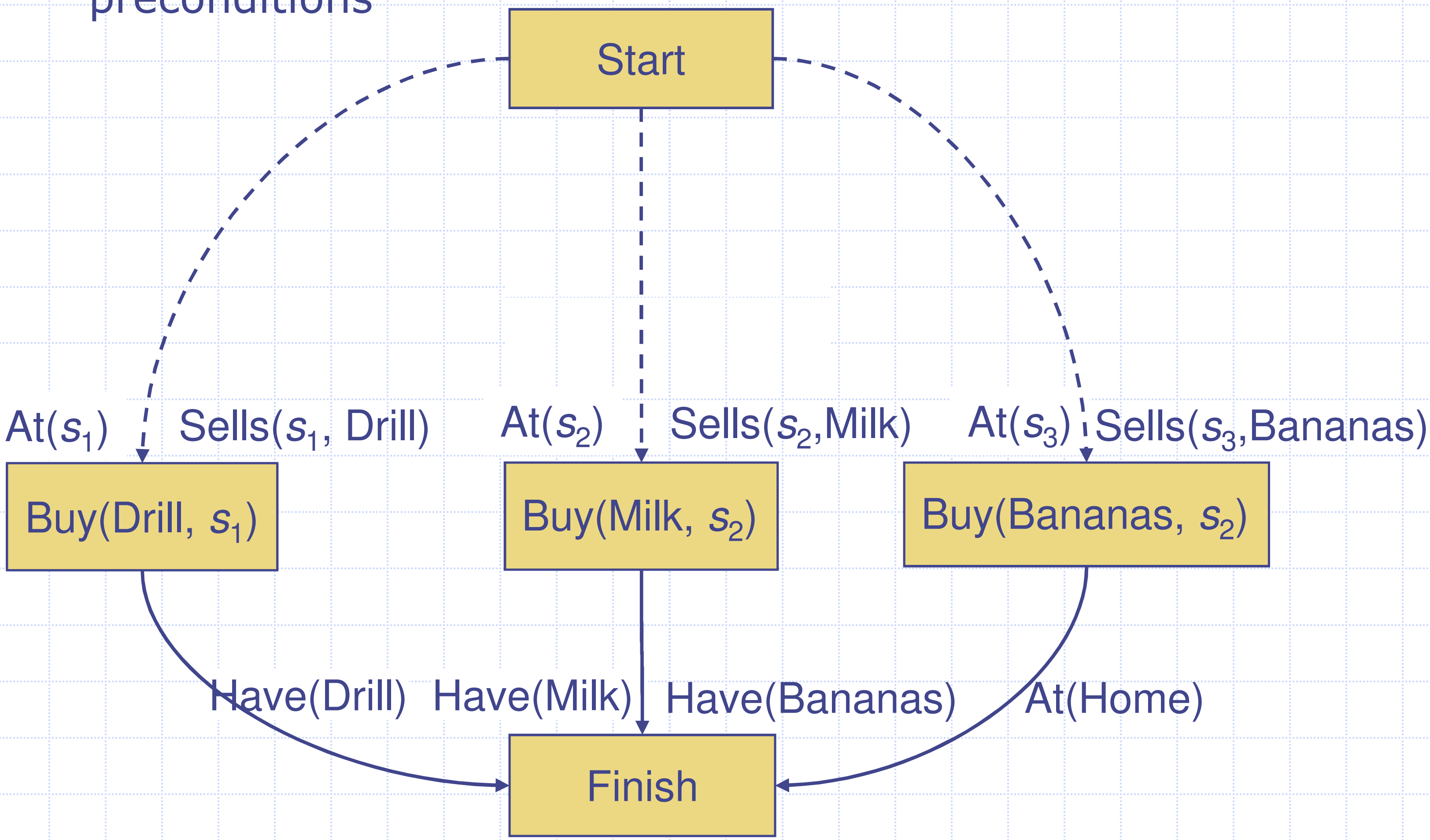
Start and **Finish** are dummy actions that we'll use instead of the initial state and goal

Example (continued)

- Need to give PSP a plan π as its argument
  - Initial plan: **Start**, **Finish**, and an ordering constraint

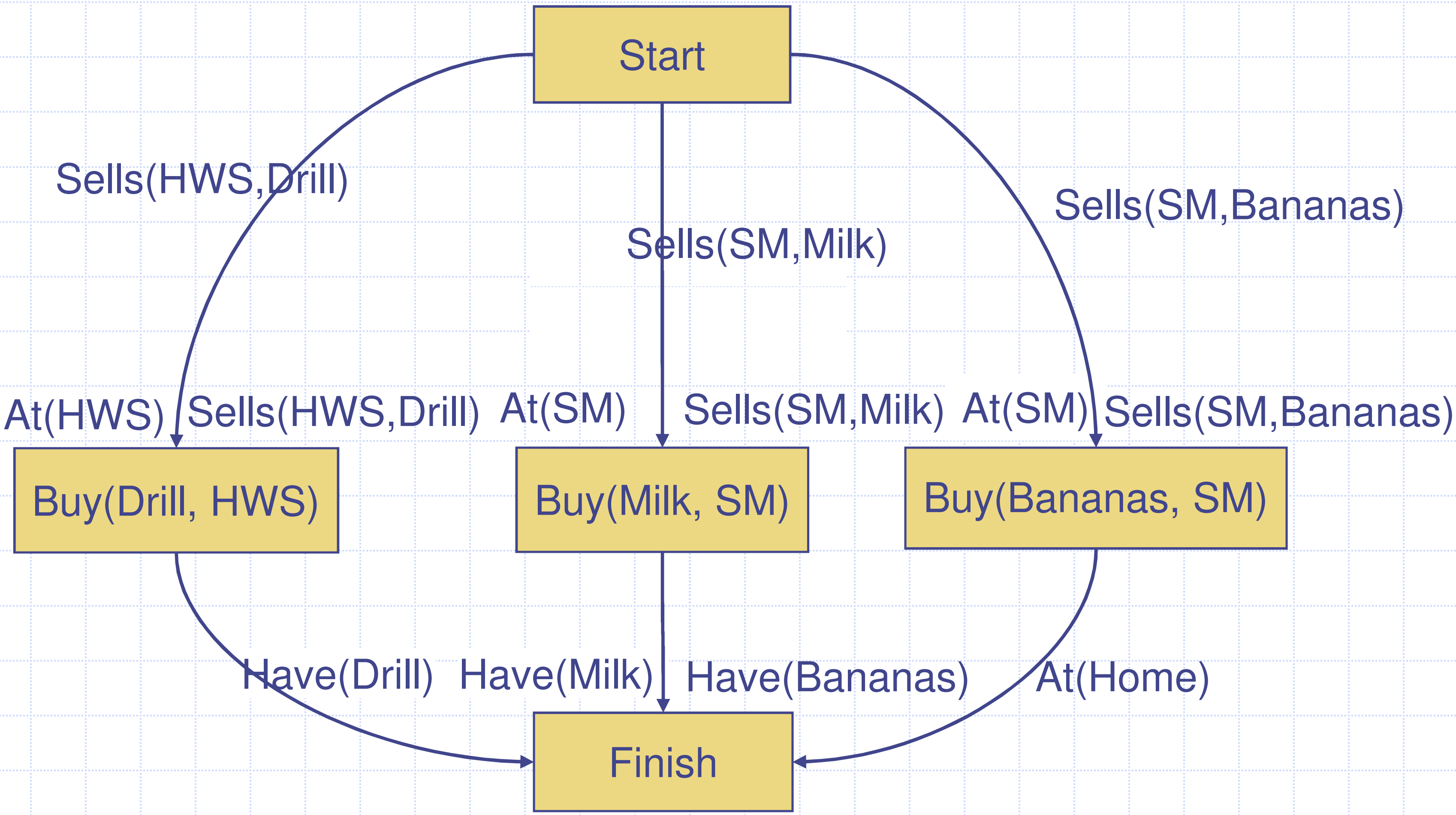
Example (continued)

- The first three refinement steps
  - These are the only possible ways to establish the Have preconditions



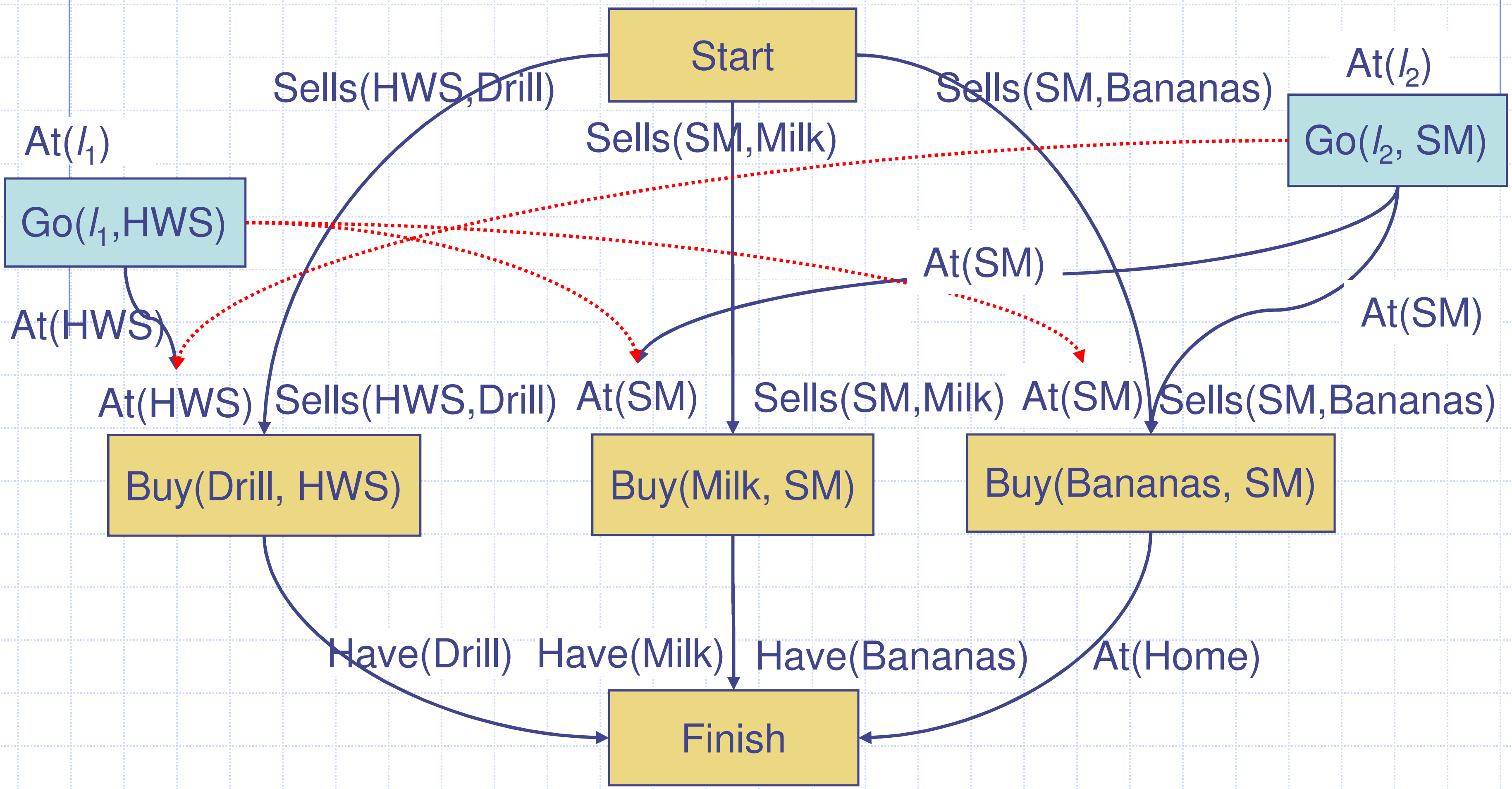
Example (continued)

- Three more refinement steps
  - The only possible ways to establish the Sells preconditions

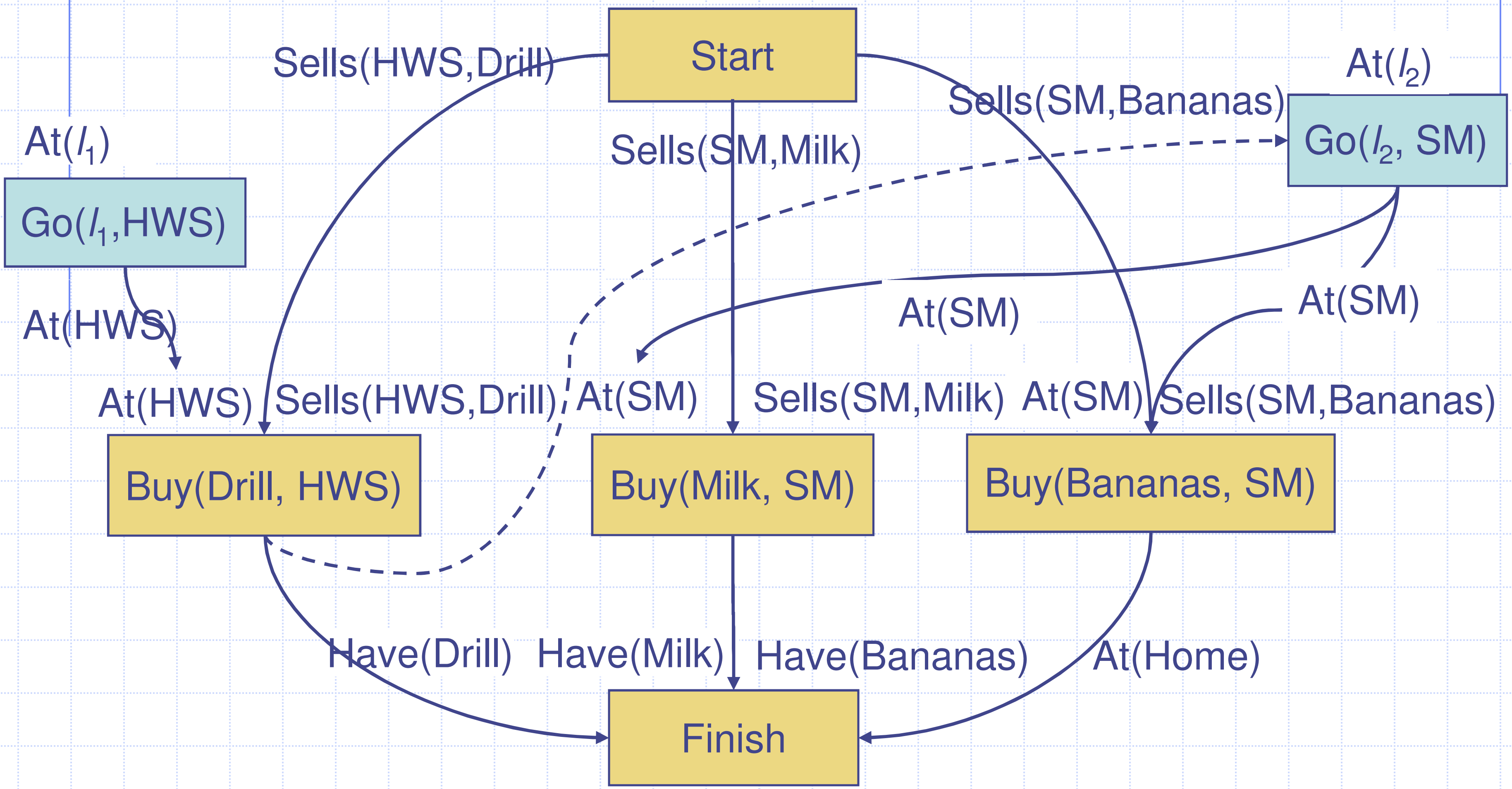


Example (continued)

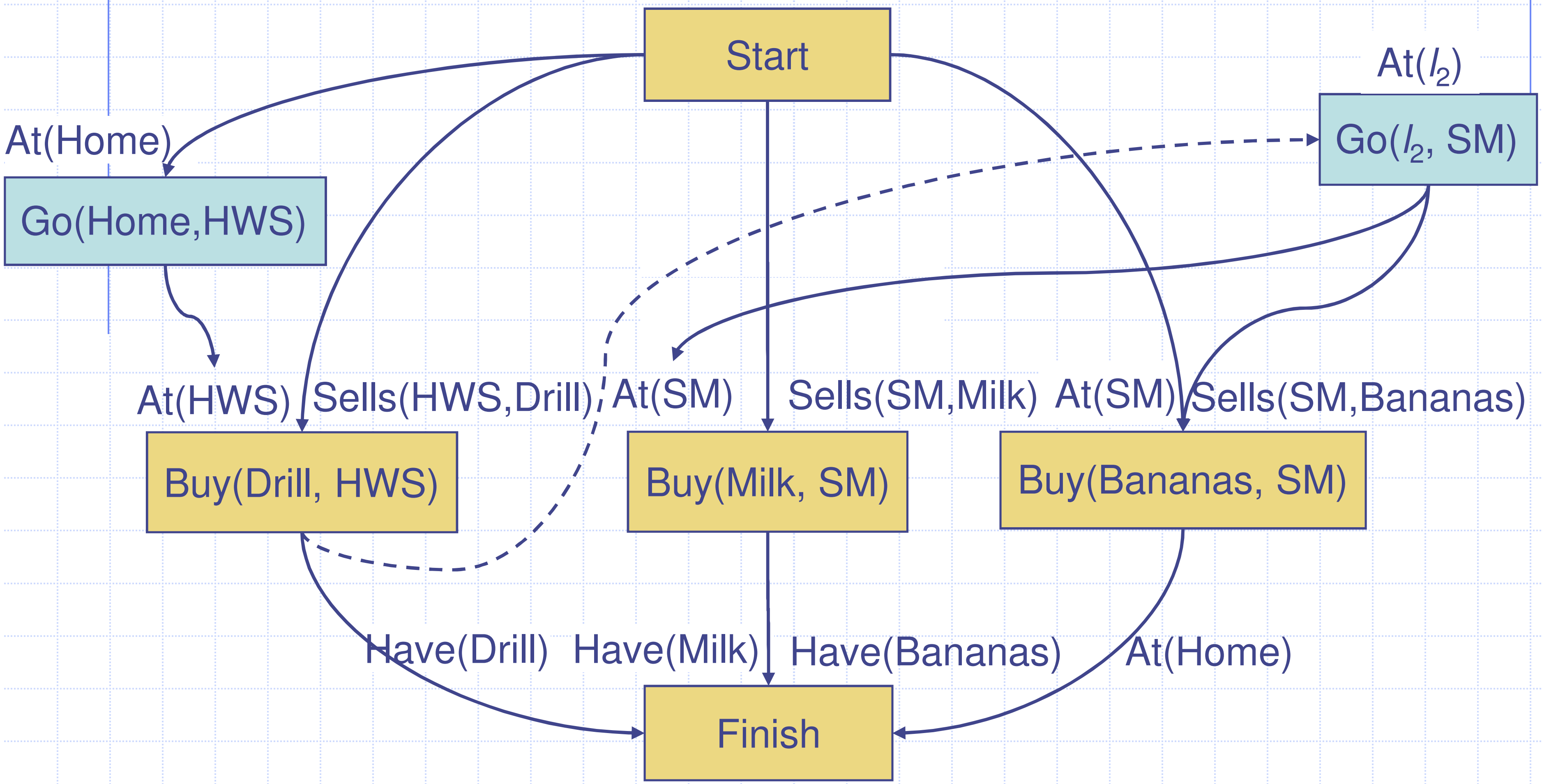
- Two more refinements: the only ways to establish At(HWS) and At(SM)
  - This time, several threats occur



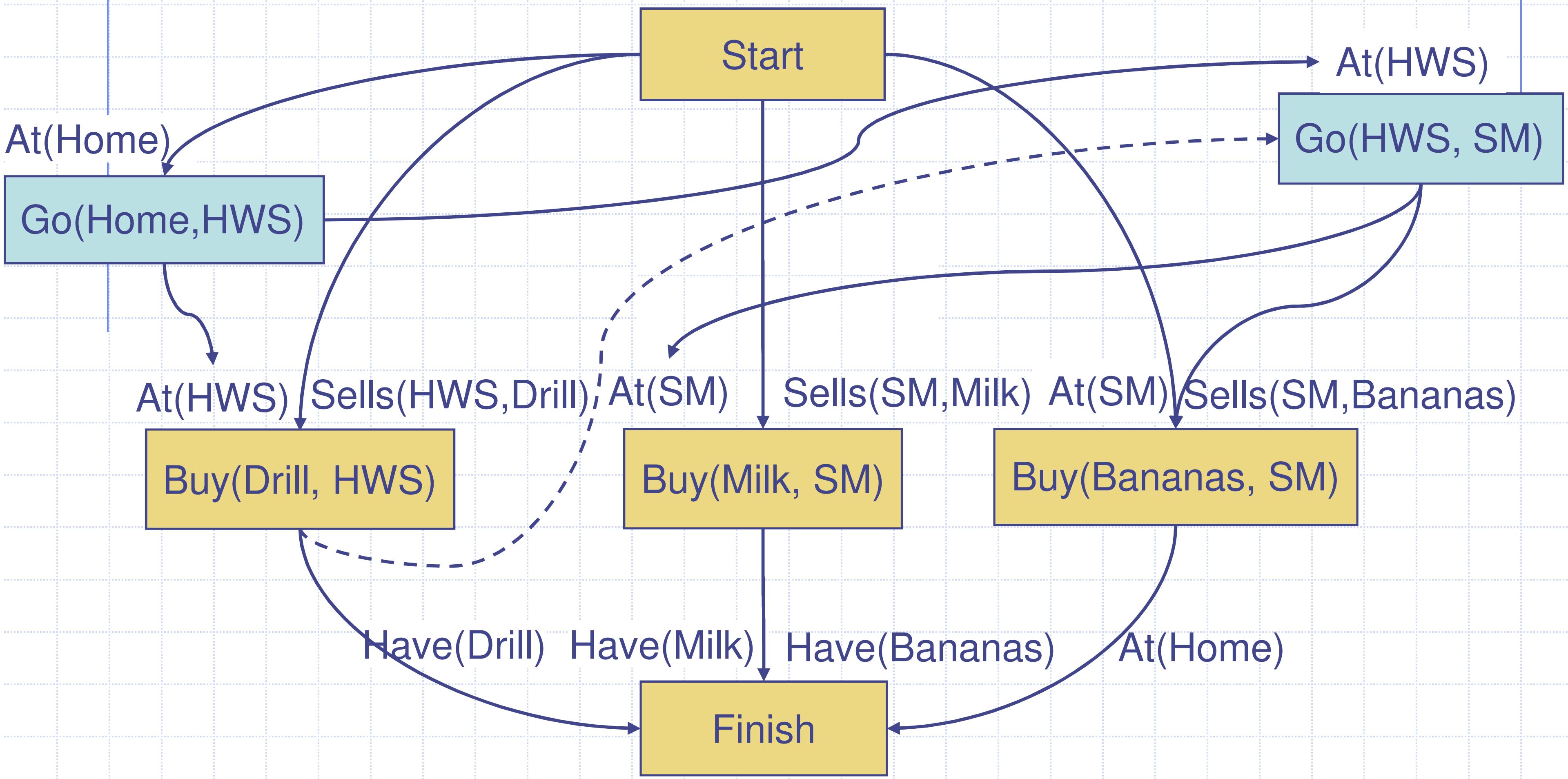
- Finally, a nondeterministic choice: how to resolve the threat to $At(s_1)$?
  - Our choice: make Buy(Drill) precede Go(SM)
  - This also resolves the other two threats



- Nondeterministic choice: how to establish At(l_1)?
  - We'll do it from Start, with l_1=Home

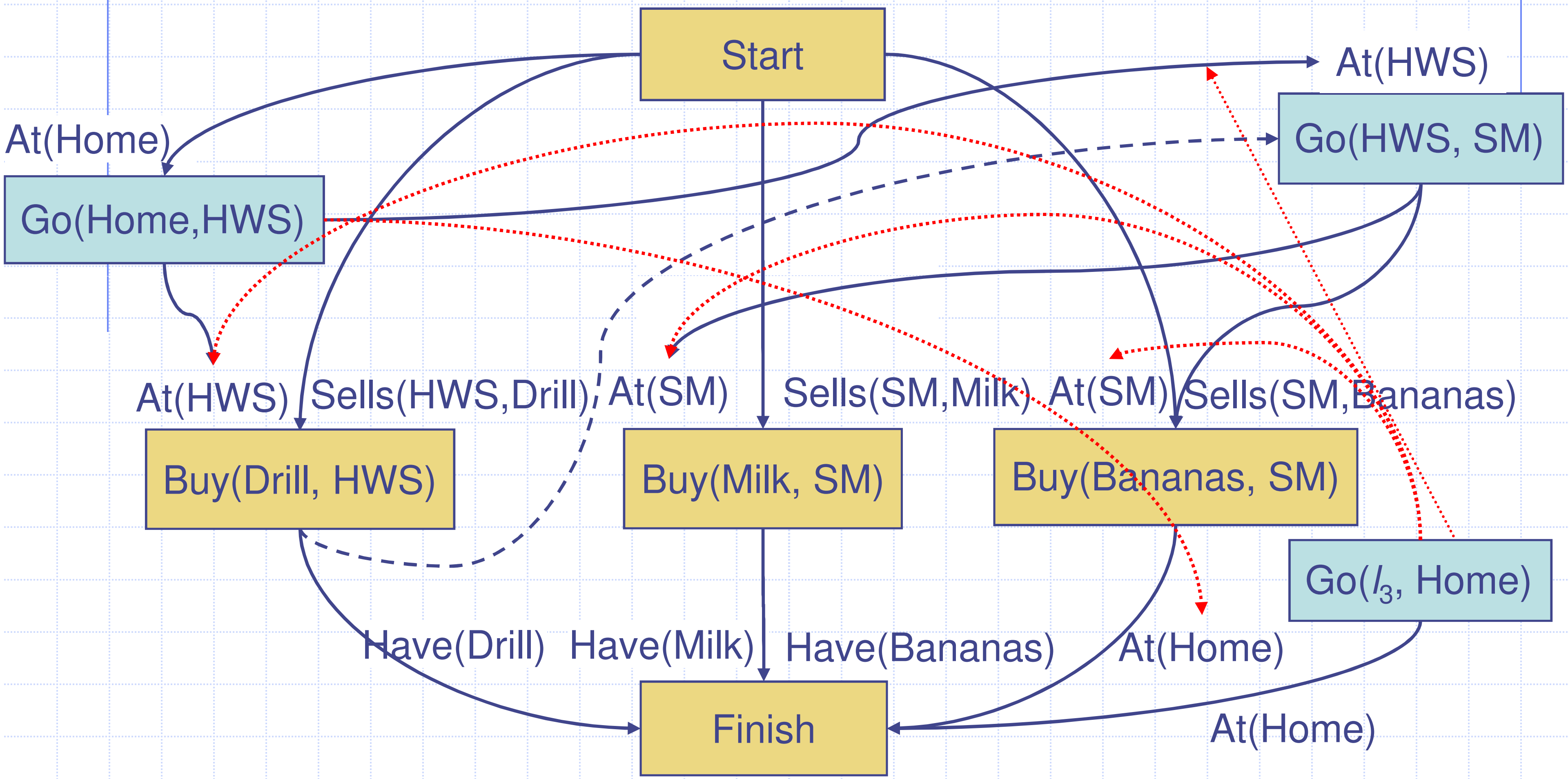


- Nondeterministic choice: how to establish At(l_2)?
  - We'll do it from Go(Home,HWS), with l_2= HWS

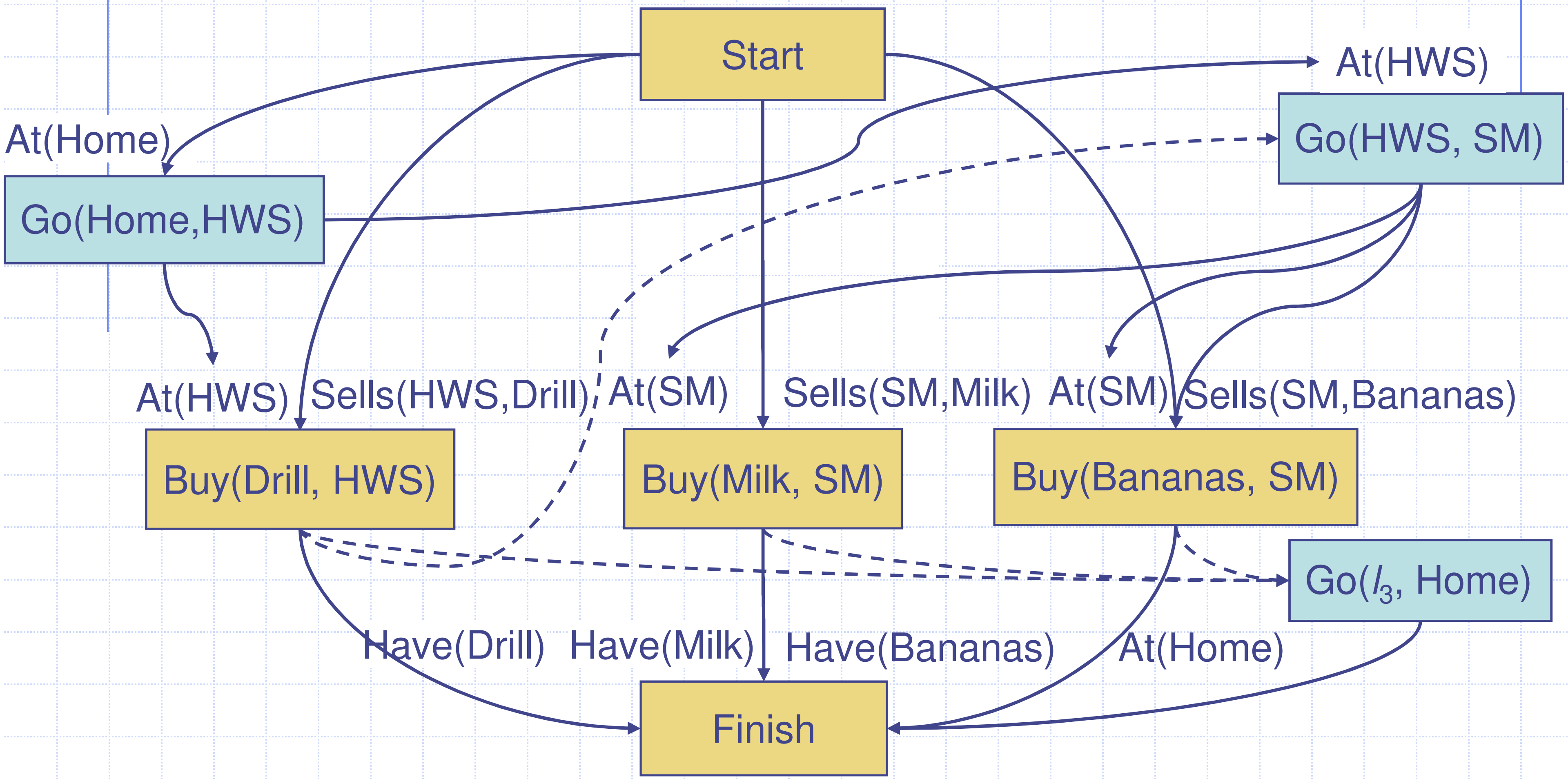


Example (continued)

- The only possible way to establish At(Home) for Finish
  - This creates a bunch of threats

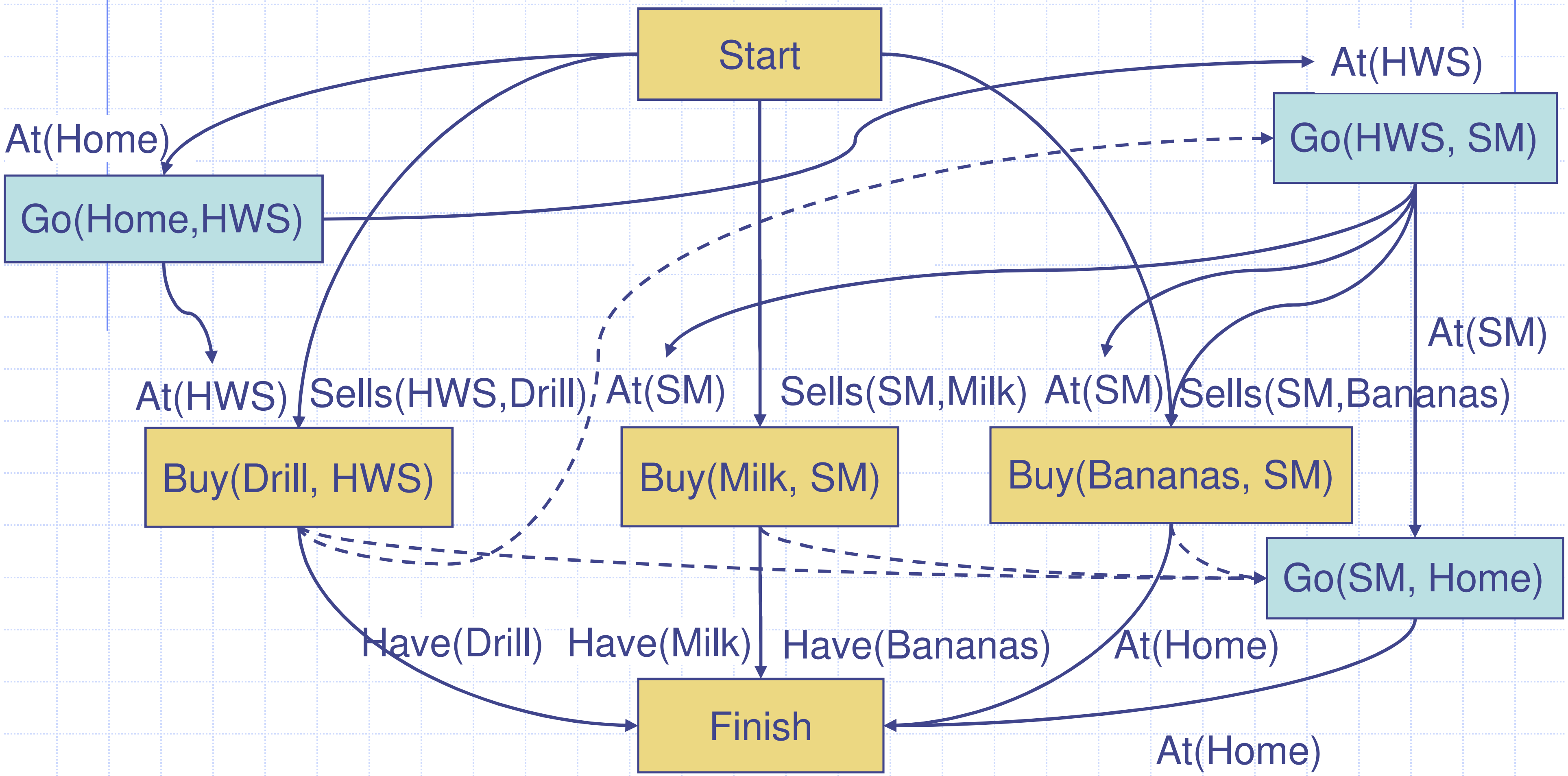


- To remove the threats to At(SM) and At(HWS), make them precede Go(l_3,Home)
  - This also removes the other threats

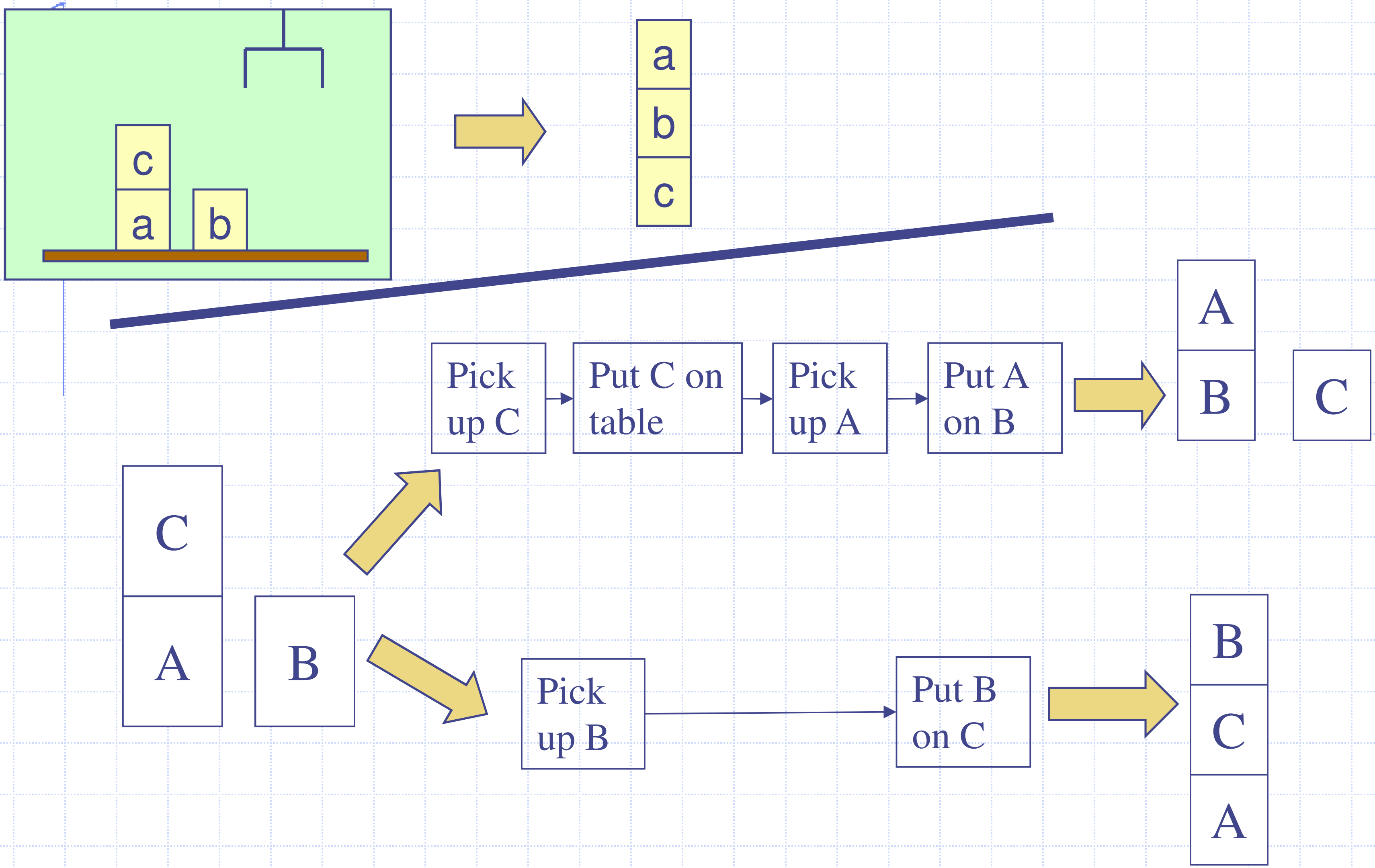


Final Plan

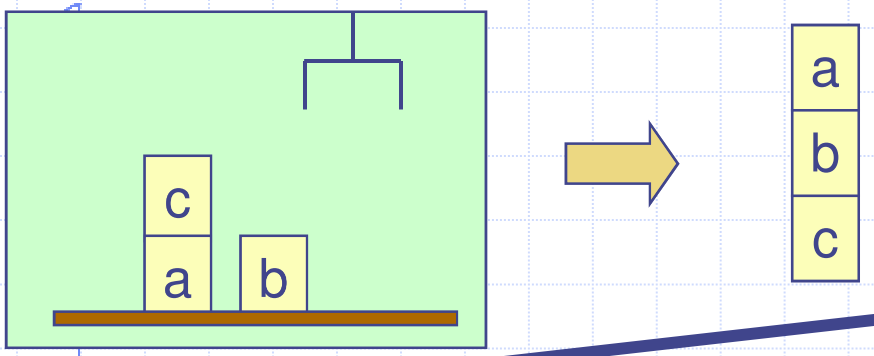
- Establish At(l_3) with l_3=SM

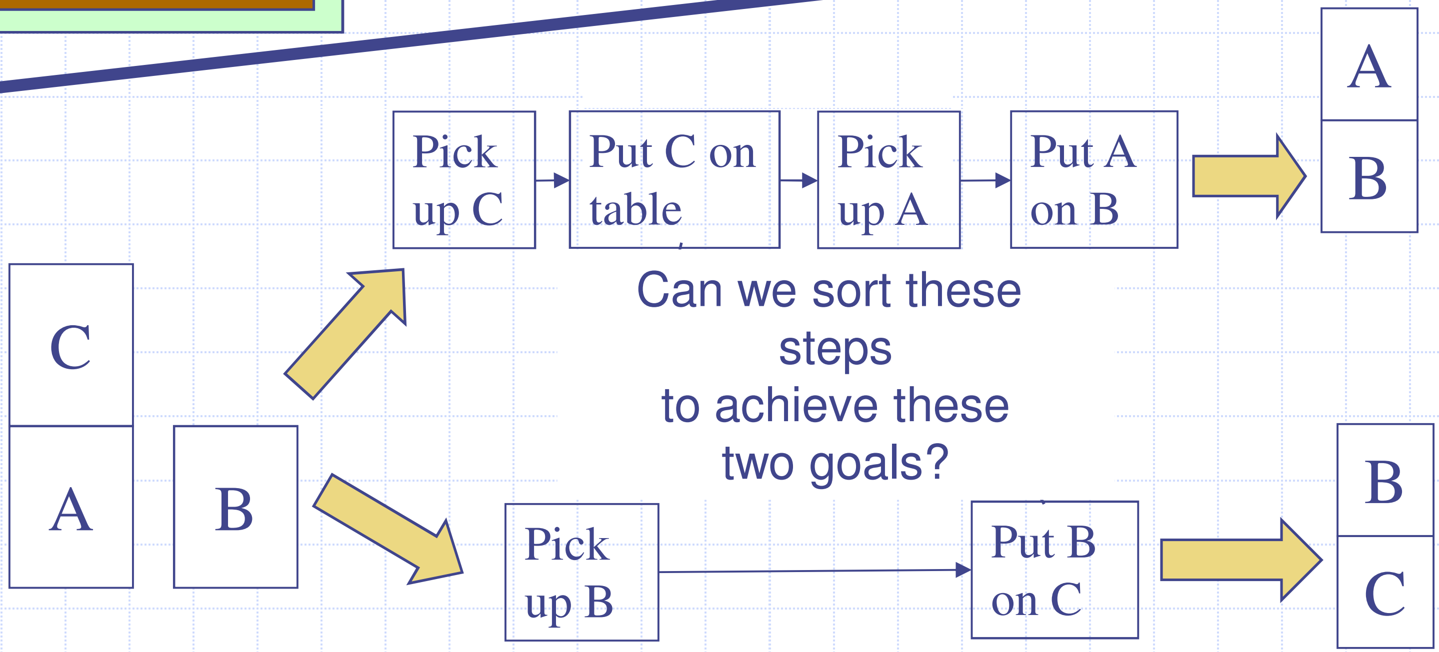


Reminder: STRIPS and the Sussman Anomaly



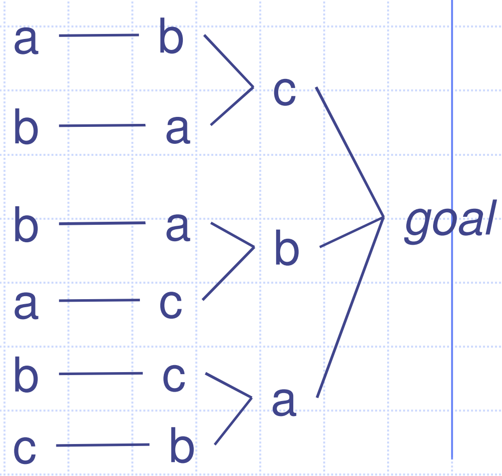
POP and the Sussman Anomaly





Can we sort these steps
to achieve these two goals?

Summary



- How to choose which flaw to resolve first and how to resolve it?
  - Heuristics
- PSP doesn't commit to orderings and instantiations until necessary
  - Avoids generating search trees like this one:
- Problem: how to prune infinitely long paths?
  - Loop detection is based on recognizing states we've seen before
  - In a partially ordered plan, we don't know the states
- Can we prune if we see the same *action* more than once?

  ... — go(b,a) — go(a,b) – go(b,a) — at(a)

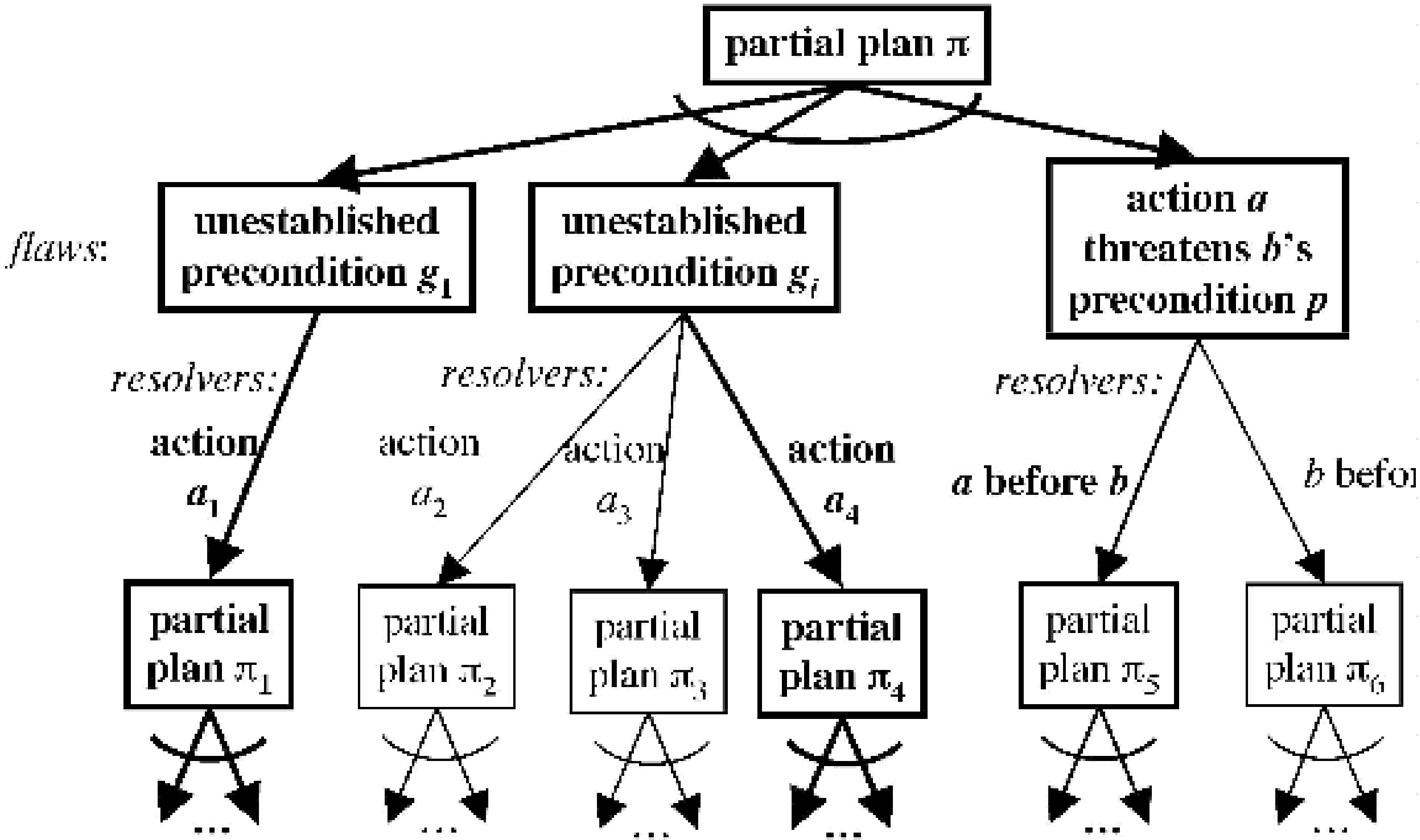
  - No. Sometimes we might need the same action several times in different states of the world.

Heuristics in plan-space planning

```
PSP(π)
    flaws ← OpenGoals(π) ∪ Threats(π)
    if flaws = ∅ then return(π)
    select any flaw φ ∈ flaws
    resolvers ← Resolve(φ, π)
    if resolvers = ∅ then return(failure)
    nondeterministically choose a resolver ρ ∈ resolvers
    π′ ← Refine(ρ, π)
    return(PSP(π′))
end
```

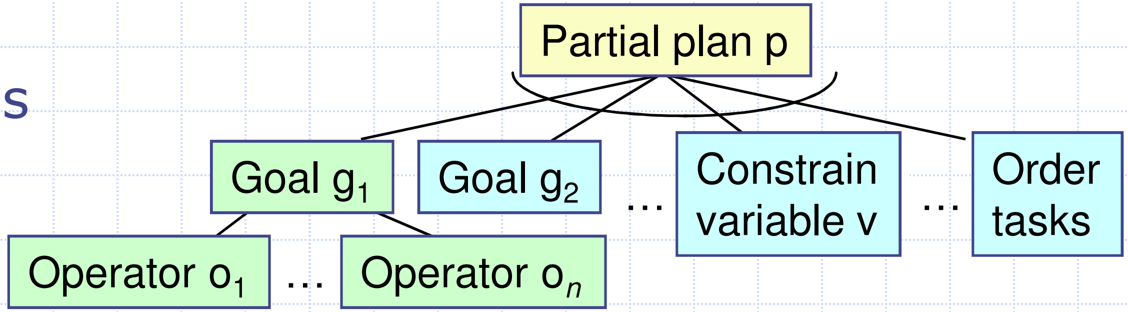
- Flaw-selection heuristic
- Resolver-selection heuristic

Heuristics in plan-space planning

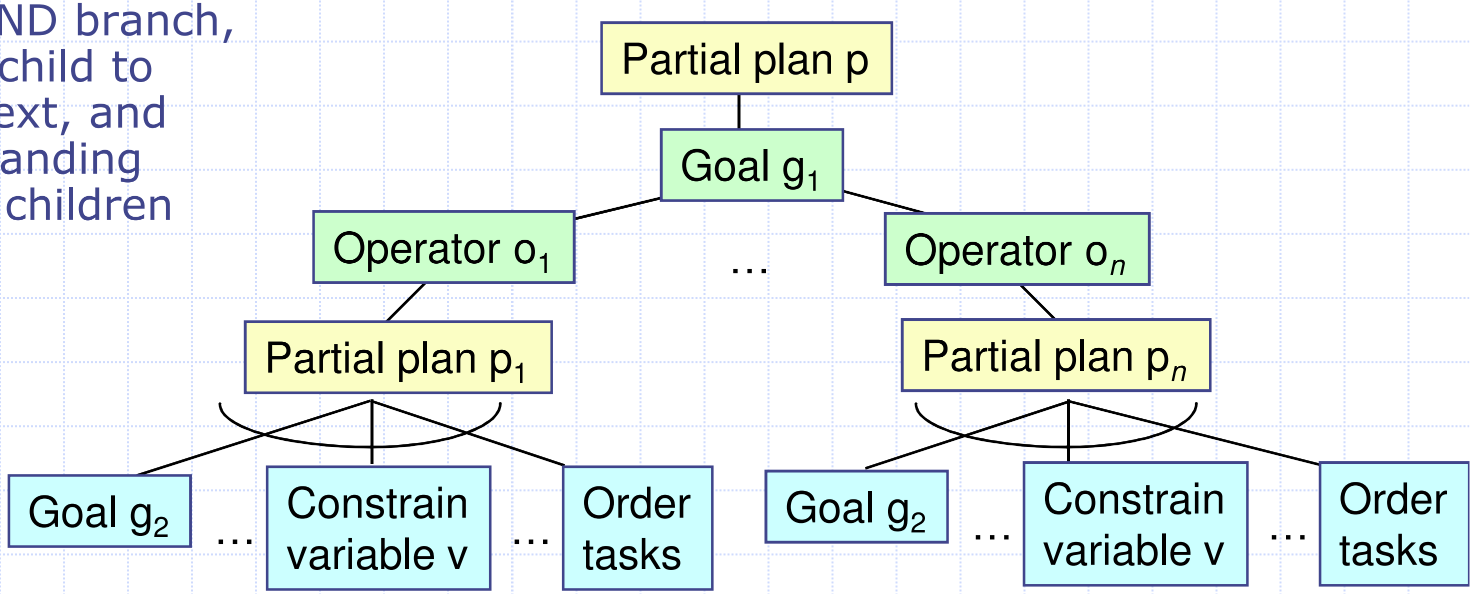


Serializing and AND/OR Tree

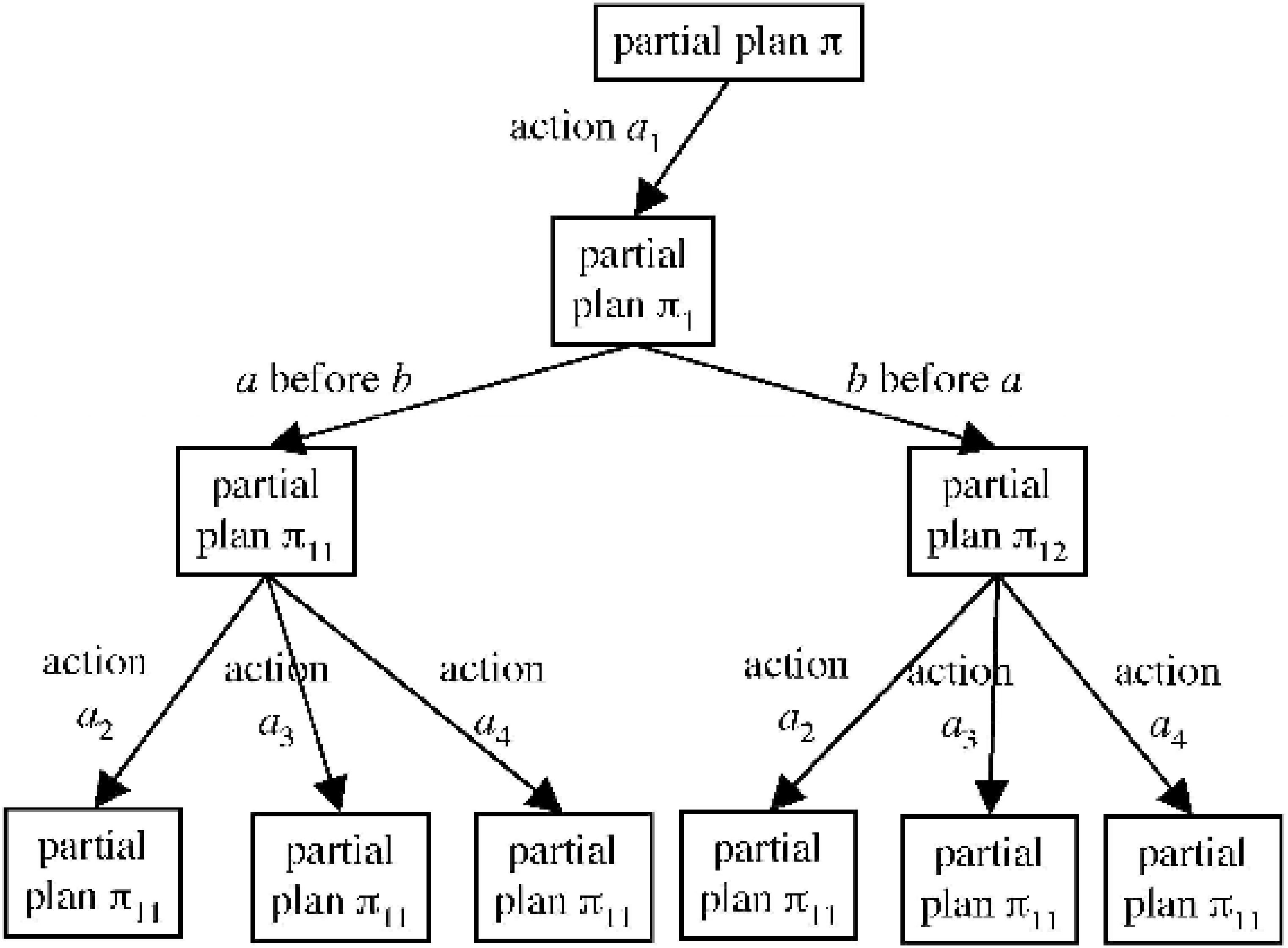
- The search space is an AND/OR tree



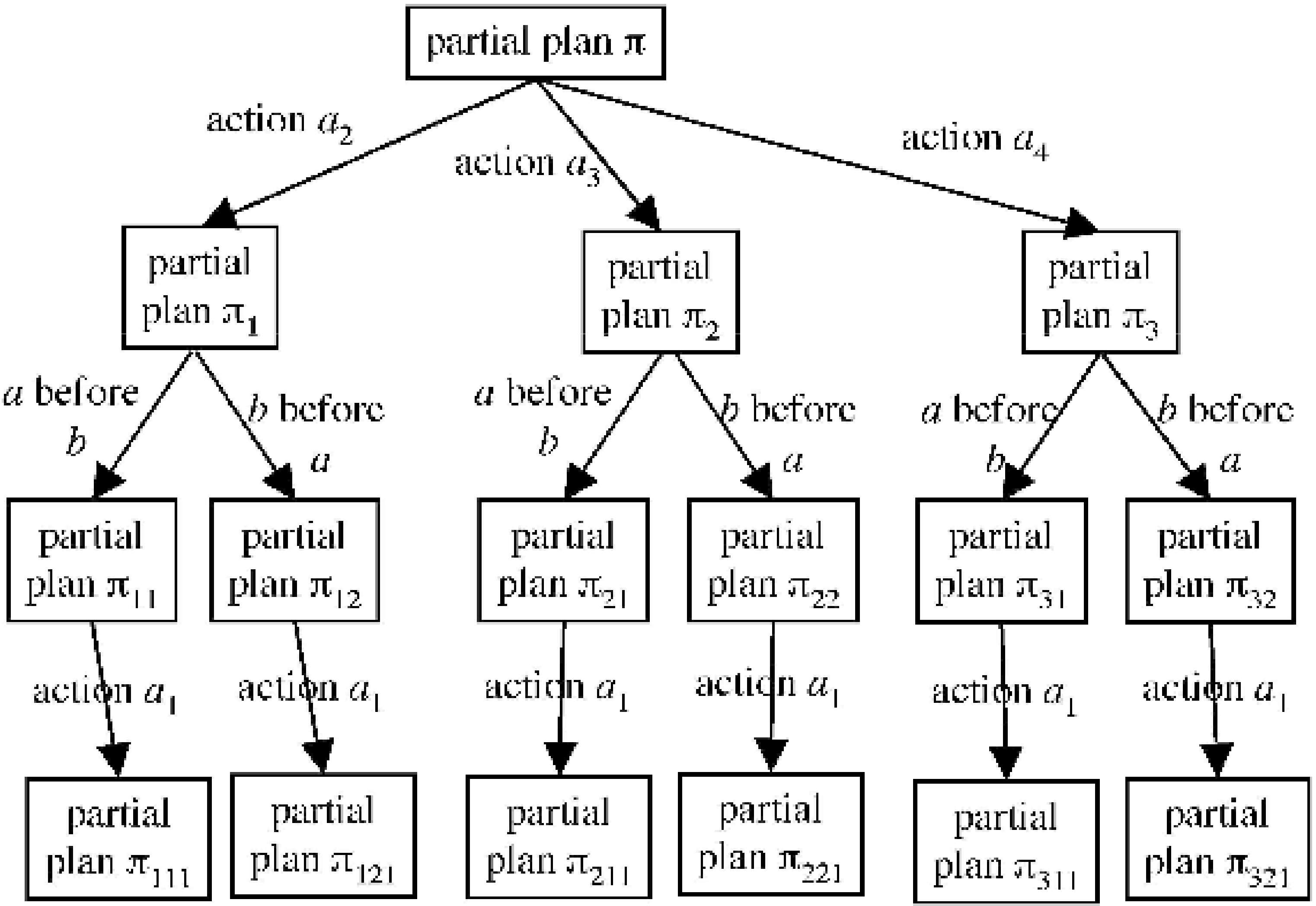
- Deciding what flaw to work on next = *serializing* this tree (turning it into a plan-space tree)
  - at each AND branch, choose a child to expand next, and delay expanding the other children



One Serialization (order of resolution: g1,threat,g2)



Another Serialization (order of resolution: g2, threat, g1)



Why Does This Matter?

- Different refinement strategies produce different serializations
  - the search spaces have different numbers of nodes
- In the worst case, the planner will search the entire serialized search space
- The smaller the serialization, the more likely that the planner will be efficient

Why Does This Matter?

- A particular goal 'g':
  - 'B' resolvers → 'B' partial plans
  - 'a' threats in each plan, 'm' resolvers for each threat
  - Effective branching factor of search by POP: $B_g = B*m^a$
- Time complexity of a solution plan with 'n' steps/actions and 'p' preconditions/goals per step:
  - $O(B_g)^{np}$
- Exponential number of nodes, so in order for PSP to be efficient we need:
  - Good flaw-selection heuristics
  - Good resolver-selection heuristics

Flaw-selection heuristics

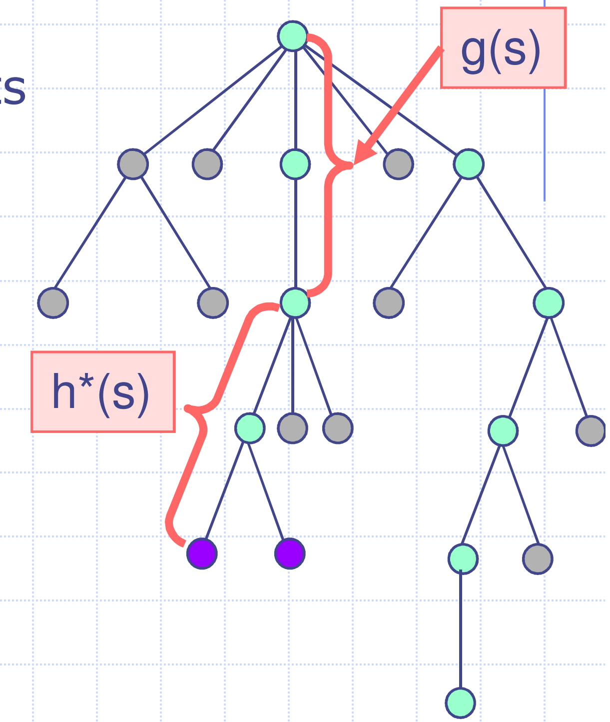
- Fewest Alternatives First (FAF)
  - Choose the flaw that has the smallest number of alternatives
  - In this case, choose to resolve precondition g_1
- Last-In, First-out (LIFO)
- Delay potential threats
- Zero-LIFO (ZLIFO)

Resolver-selection heuristics

- Select the least cost partial plan ($\Pi_1, \Pi_2, \ldots, \Pi_n$)
- Use an A* algorithm $f(\Pi)=g(\Pi)+h(\Pi)$ where:
  - $g(\Pi)$ is the cost of the path from the root node (initial empty plan) to plan Π
  - $h(\Pi)$ is an estimate of the least cost of the path from Π to a solution plan (goal state)

Node-Selection Heuristic

- Suppose we're searching a **tree** in which each edge (s,s') has a cost $c(s,s')$
  - If p is a path, let $c(p)$ = sum of the edge costs
  - For classical planning, this is the length of p



- For every state s, let
  - $g(s)$ = cost of the path from s_0 to s
  - $h^*(s)$ = least cost of all paths from s to goal nodes
  - $f^*(s) = g(s) + h^*(s)$ = least cost of all paths from s_0 to goal nodes that go through s

- Suppose $h(s)$ is an estimate of $h^*(s)$
  - Let $f(s) = g(s) + h(s)$
    - $f(s)$ is an estimate of $f^*(s)$
  - h is *admissible* if for every state s, $0 \le h(s) \le h^*(s)$
  - If h is admissible then f is a lower bound on f^*

The A* Algorithm

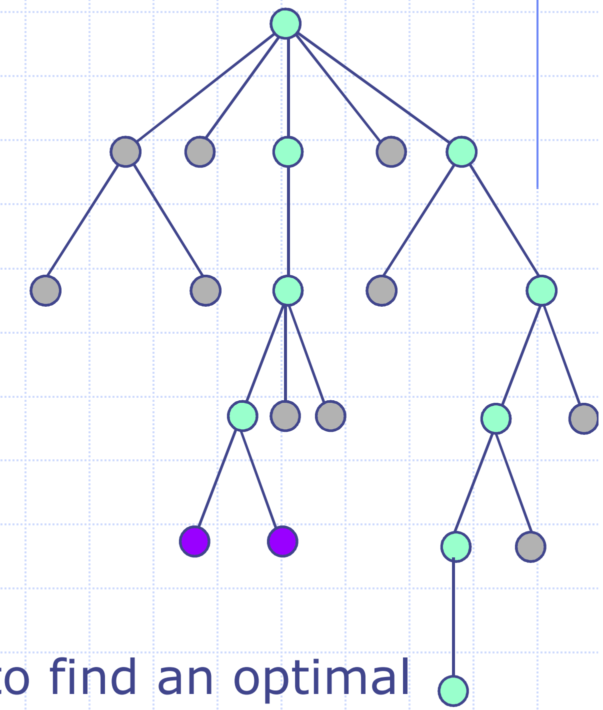
- A* on trees:

  loop

  choose the leaf node s such that $f(s)$ is smallest

  if s is a solution then return it and exit

  expand it (generate its children)



- On graphs, A* is more complicated
  - additional machinery to deal with multiple paths to the same node
- If a solution exists (and certain other conditions are satisfied), then:
  - If $h(s)$ is admissible, then A* is guaranteed to find an optimal solution
  - The more "informed" the heuristic is (i.e., the closer it is to h^*), the smaller the number of nodes A* expands
  - If $h(s)$ is within c of being admissible, then A* is guaranteed to find a solution that's within c of optimal

Resolver-selection heuristics

- Elements of a plan Π that can take part of the function $f(\Pi)=g(\Pi)+h(\Pi)$:
  - $N(\Pi)$: number of steps in plan Π
  - $OP(\Pi)$: open goals/preconditions in plan Π
  - $T(\Pi)$: threats in plan Π

- Usually $f(\Pi)=N(\Pi)+OP(\Pi)$:
  - Does this 'f' function guarantee to find an optimal solution, i.e., is it an A* algorithm?