Automated Planning

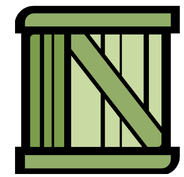
Plan-Space Planning / Partial Order Causal Link Planning

Jonas Kvarnström
Automated Planning Group
Department of Computer and Information Science
Linköping University

Partly adapted from slides by Dana Nau

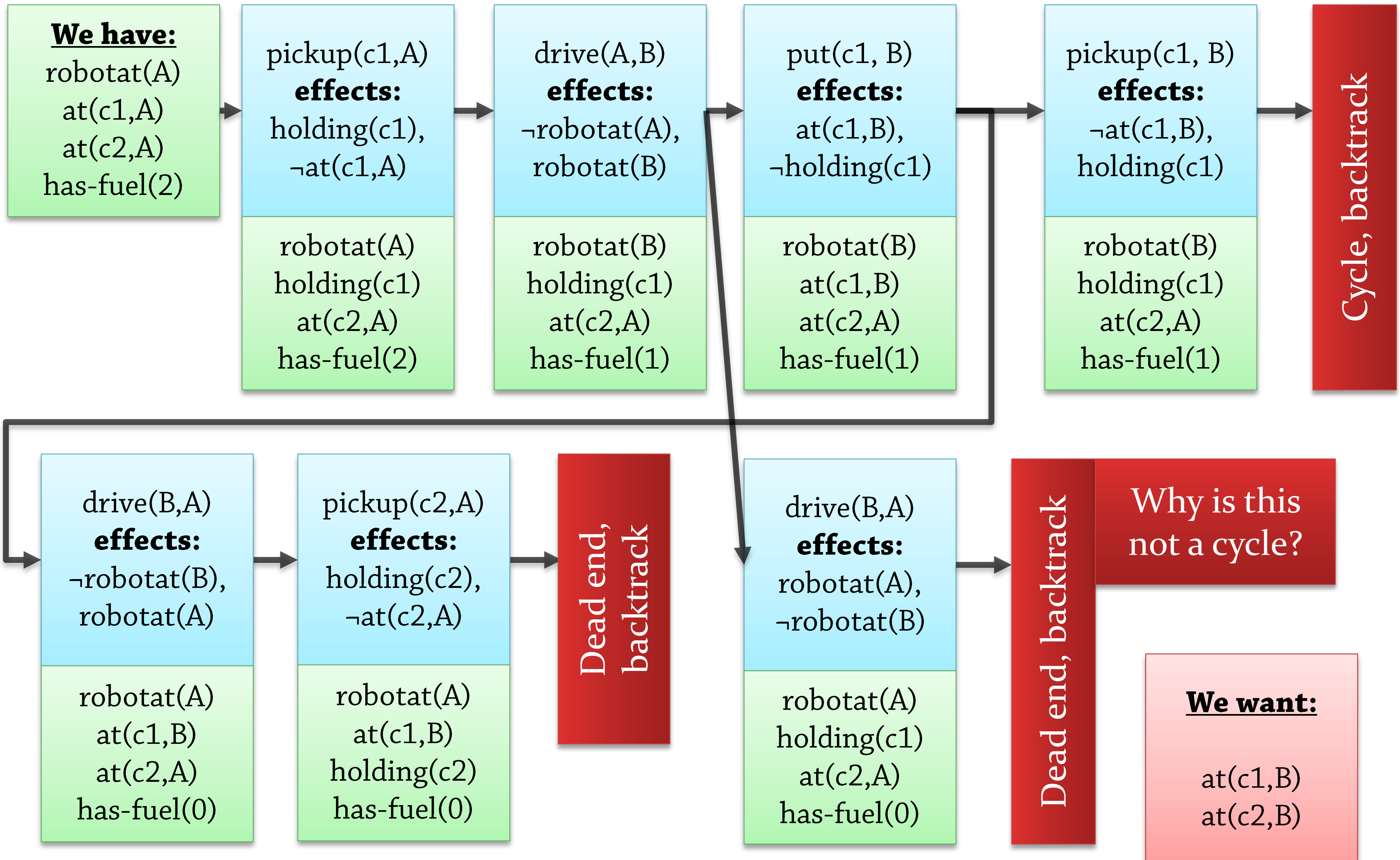
Motivating Problem

- Simple planning problem:
  - Two **crates**
    - Both are at A
    - Both should be at B

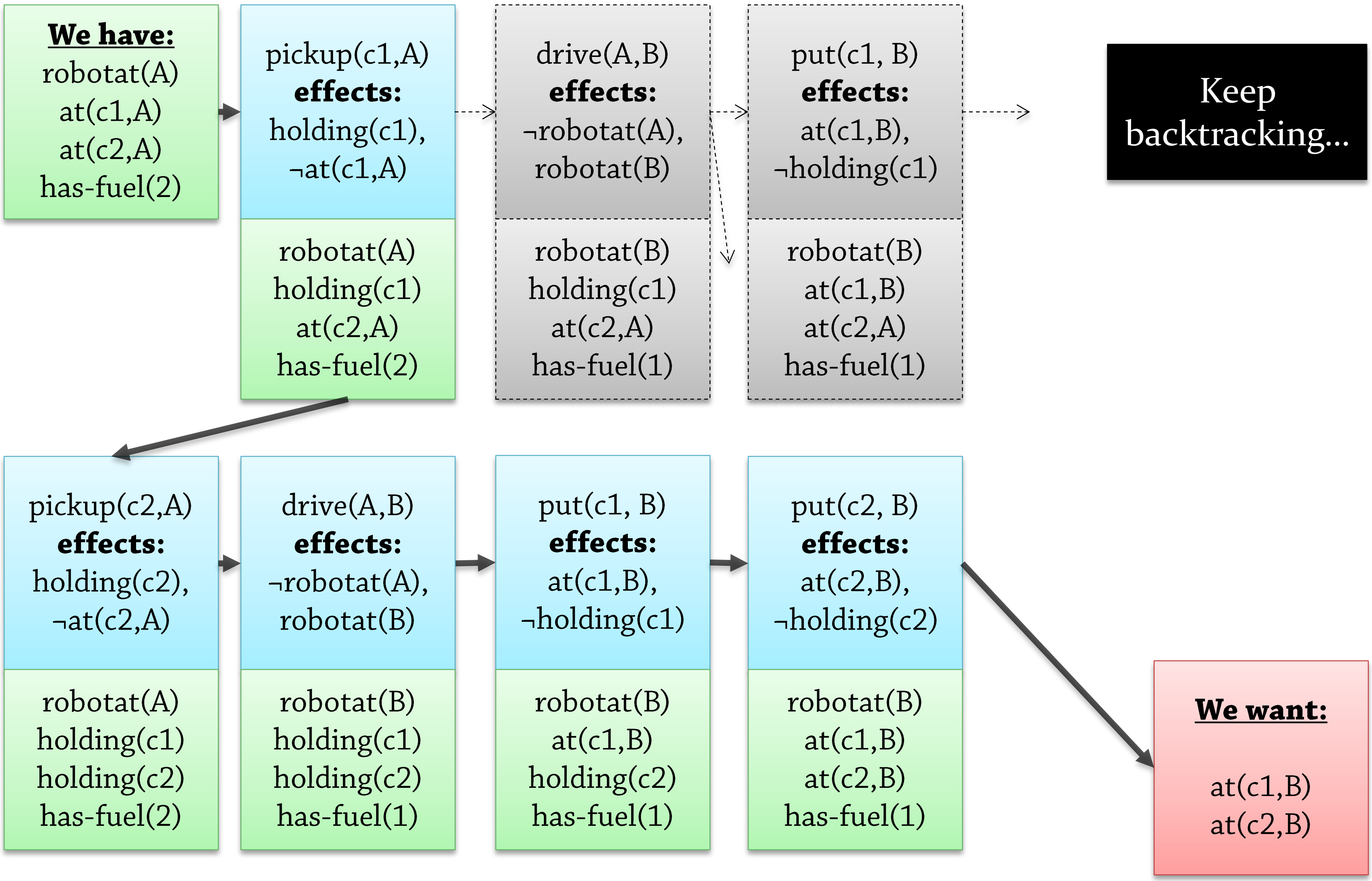


  - One **robot**
    - Can **carry** up to two crates
    - Can **move** between locations, which requires one unit of **fuel**
    - Has only two units of fuel

Motivating Problem 2: Forward Search



Motivating Problem 3



Motivating Problem 4

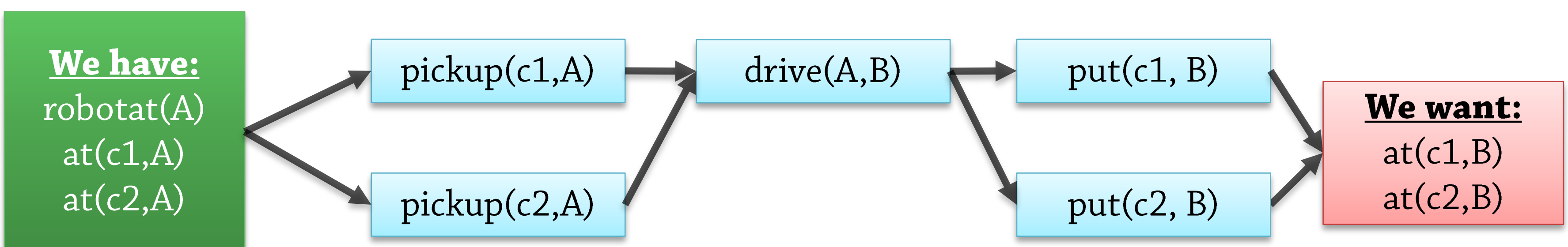
- Observations:
  - Most actions we added before backtracking were **useful** and **necessary**!

pickup(c1,A)	drive(A,B)	put(c1, B)		pickup(c2,A)

  - At first, we added them in the **wrong order**
    - State-space planning **commits** immediately to action order (in backwards search as well)
    - Puts each action in its **final place** in the plan
  - ➔ A great deal of backtracking

POCL 1: Intuitions

- Partial Order Causal Link (POCL) planning:
  - As in backward search:
    - Add **relevant** actions to achieve necessary conditions
    - Keep track of what remains to be achieved
  - But use a **partial order** for actions!
    - Insert actions ”at any point” in a plan
    - **Least/late commitment** to ordering

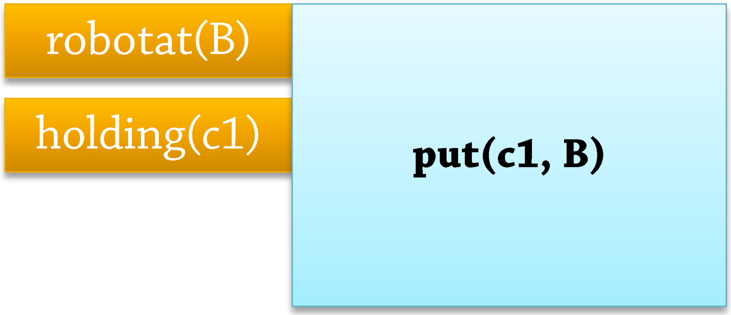


More sophisticated ”bookkeeping” required!

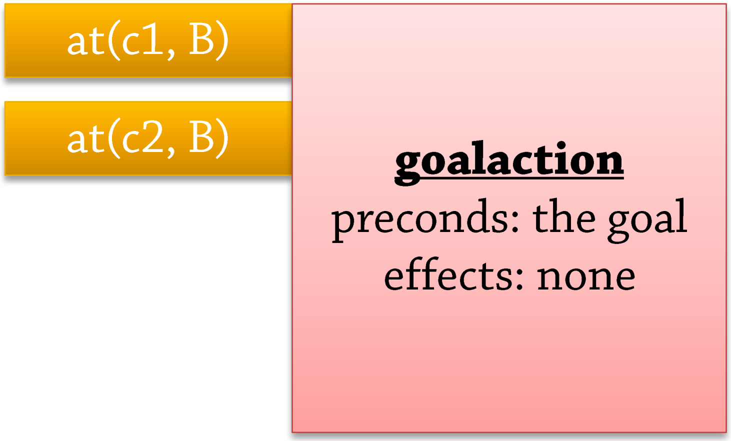
POCL 2: Goal Action

- Must keep track of **<u>propositions to be achieved</u>**
  - May come from **<u>preconditions</u>** of actions in the plan

> Simplified (non-standard) graphical representation: Preconditions on the left/top side



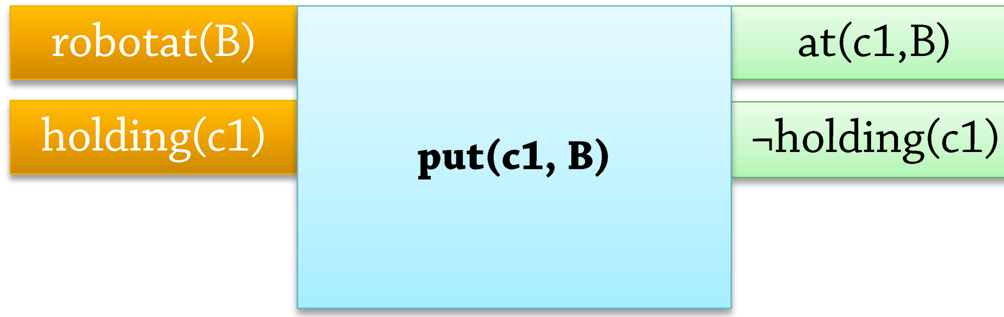
  - May come from the **<u>problem goal</u>** as in backward search
    - Let's use a **<u>uniform representation</u>**
    - Add a "fake" **<u>goal action</u>** to every plan, with the goals as preconditions!



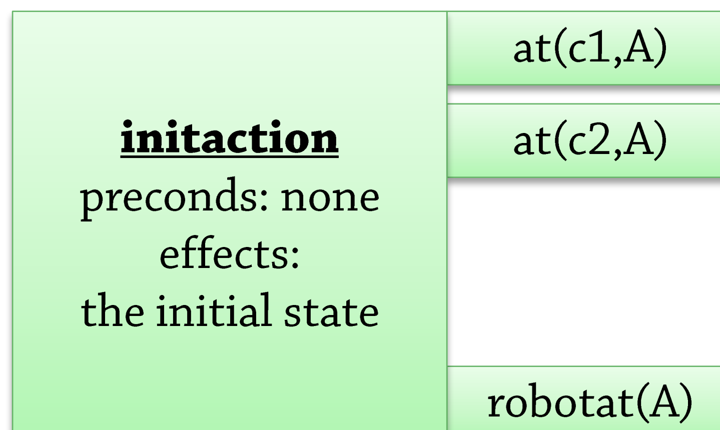
POCL 3: Initial Action

- Must keep track of **propositions that are achieved**
  - May come from **effects** of actions in the plan

Effects on the right/bottom side



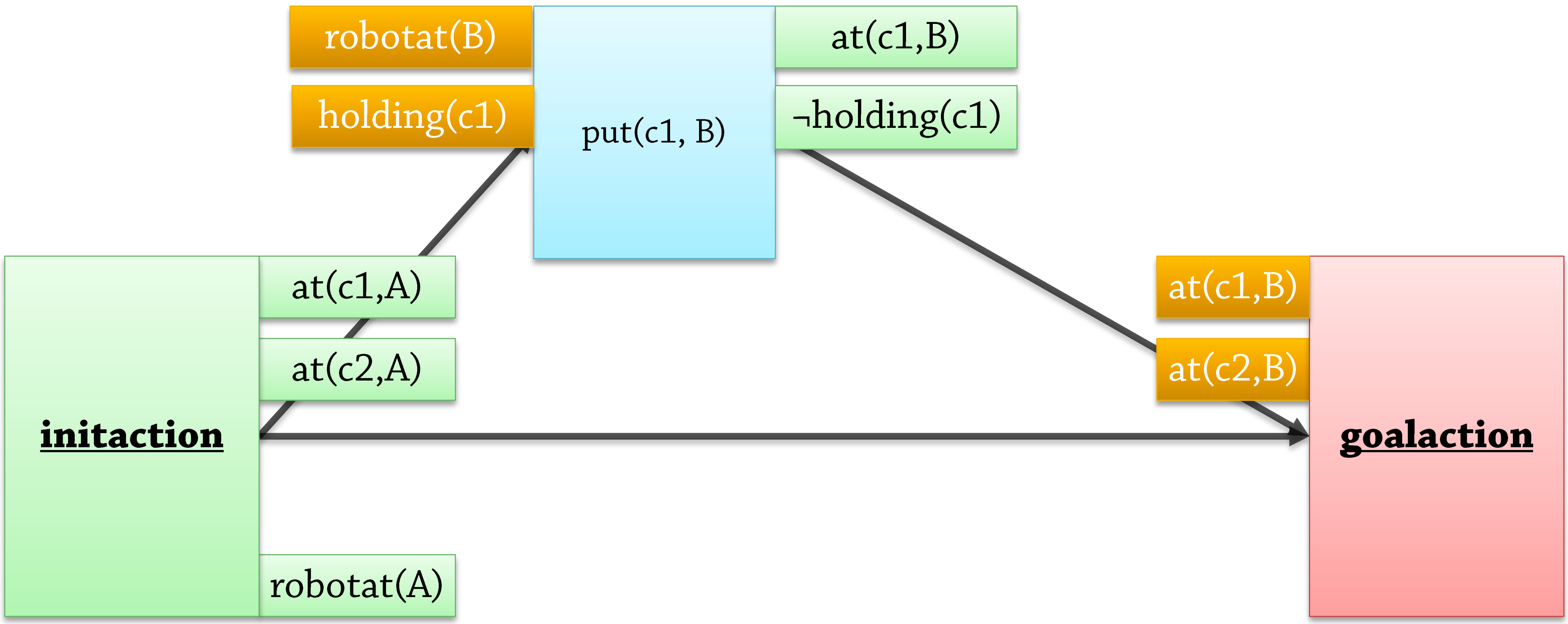
  - May come from the **initial state**
    - Add a "fake" **initial action** to every plan, with the initial state as effects!



  - Effects are sometimes omitted from the slides, due to lack of space…

POCL 4: Precedence Constraints

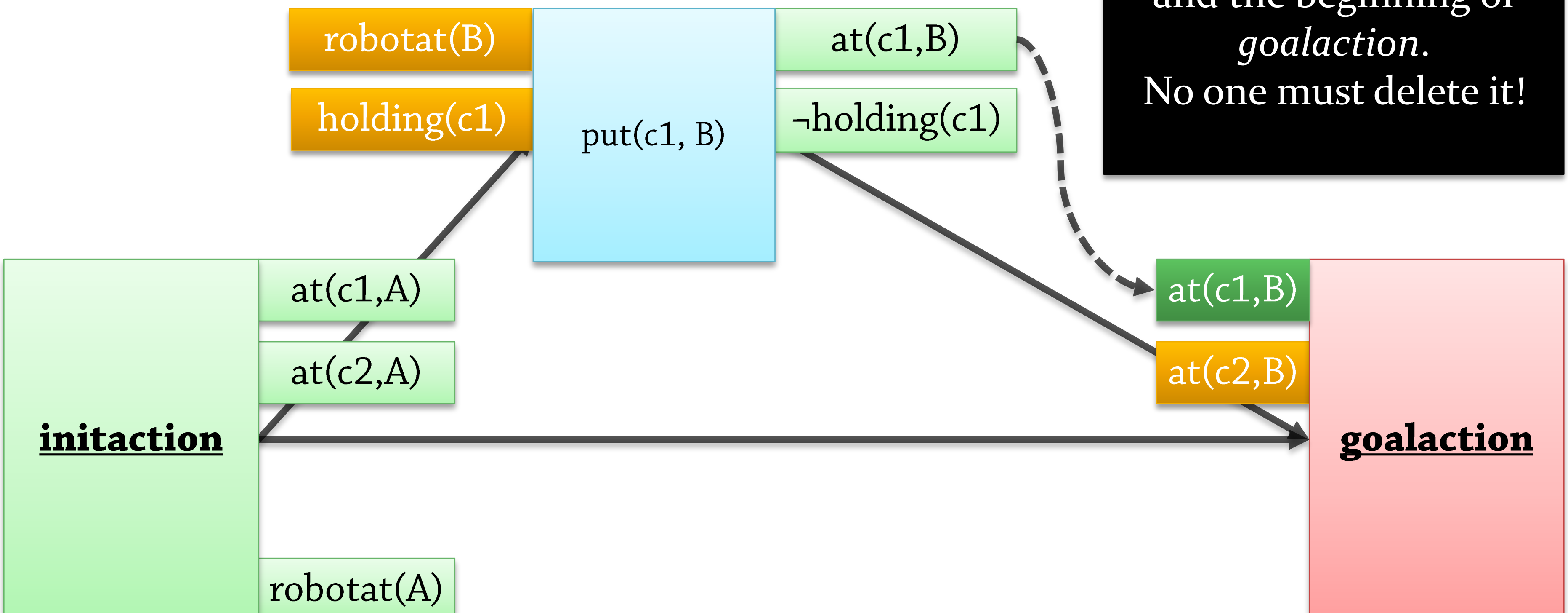
- Must keep track of **precedence constraints**
  - Stating that one action must end before another action can start
  - We will represent this using solid arrows



POCL 5: Causal Links

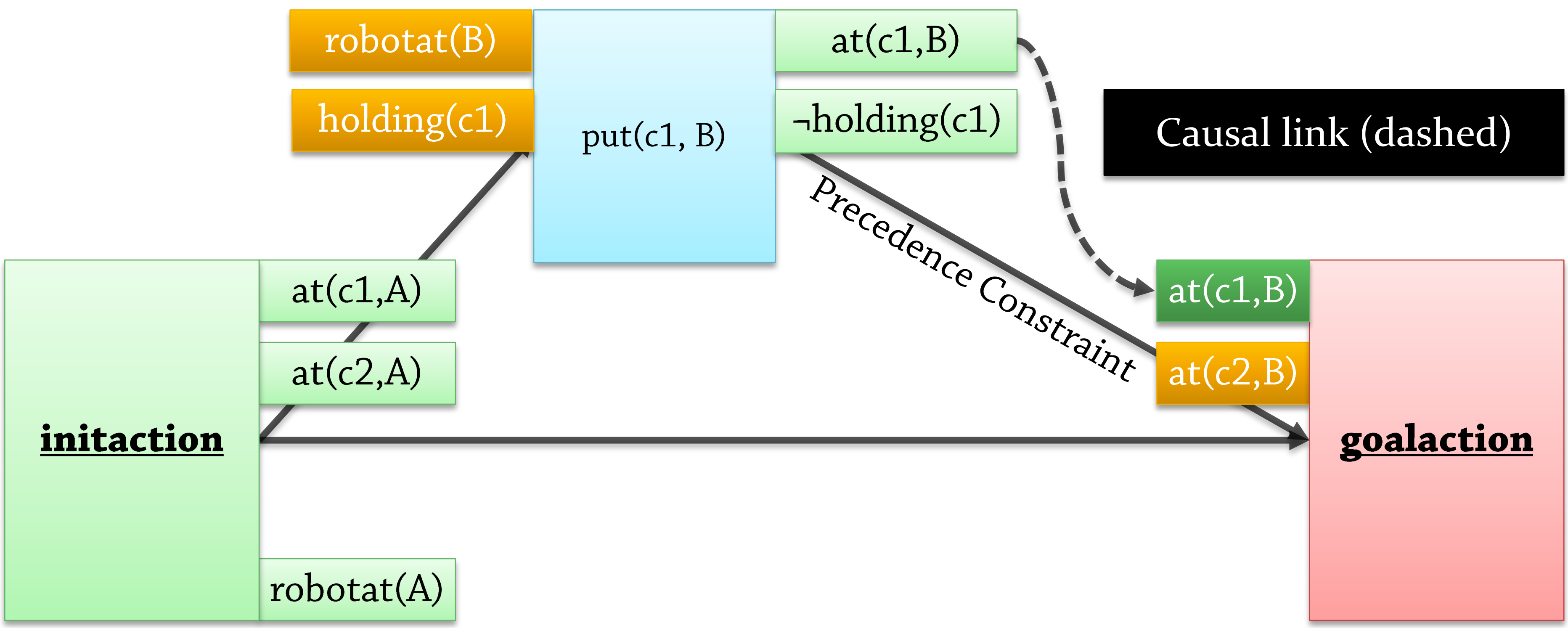
- Must keep track of **which** action **achieves** which precondition
  - **Causal links**

Causal link (dashed):
at(c1,B) must
remain true
between the end of *put*
and the beginning of
goalaction.
No one must delete it!



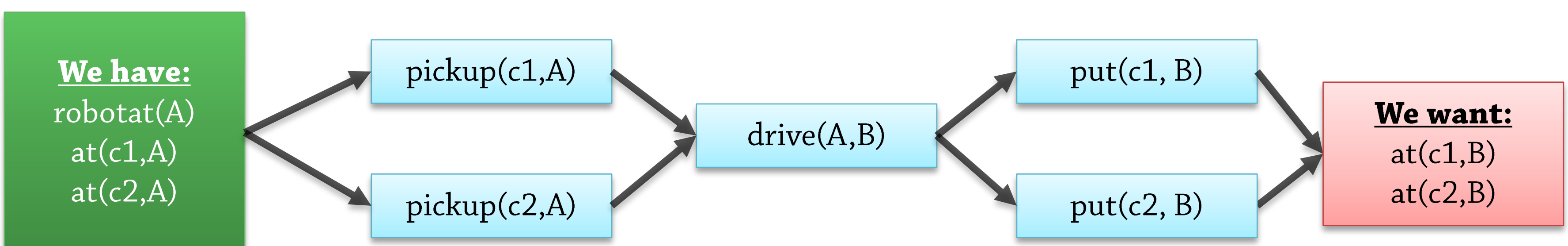
Partial-Order Plans

- To summarize, a ground **partial-order plan** consists of:
  - A set of **actions**
  - A set of **precedence constraints**: *a* must precede *b*
  - A set of **causal links**: action *a* establishes the precond *p* needed by *b*



Partial-Order Solutions

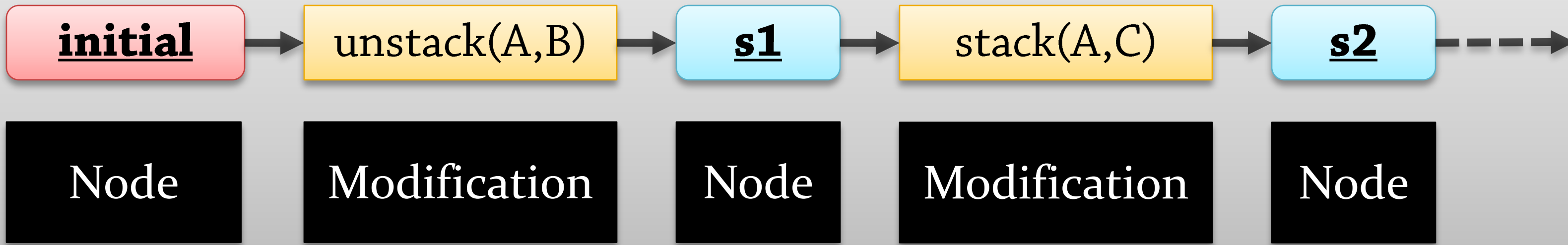
- Original motivation: **performance**
  - Therefore, a partial-order plan is a **solution**
    iff **all** **sequential** plans satisfying the ordering are solutions
    - Similarly, **executable** iff corresponding sequential plans are executable
    - <pickup(c1,A), pickup(c2,A), drive(A,B), put(c1,B), put(c2,B)>
    - <pickup(c2,A), pickup(c1,A), drive(A,B), put(c1,B), put(c2,B)>
    - <pickup(c1,A), pickup(c2,A), drive(A,B), put(c2,B), put(c1,B)>
    - <pickup(c2,A), pickup(c1,A), drive(A,B), put(c2,B), put(c1,B)>
  - Can be **extended** to allow **concurrent execution**
    - Requires a new formal model:
      Our state transition model says nothing about what happens
      if c1 and c2 are picked up simultaneously!



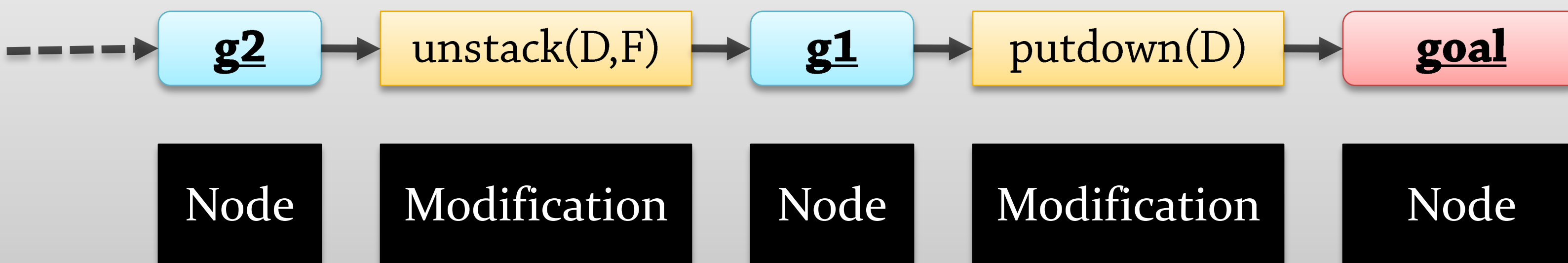
Generating Partial-Order Plans

Context: Forward, Backward

Forward search: A search node is a "current state"

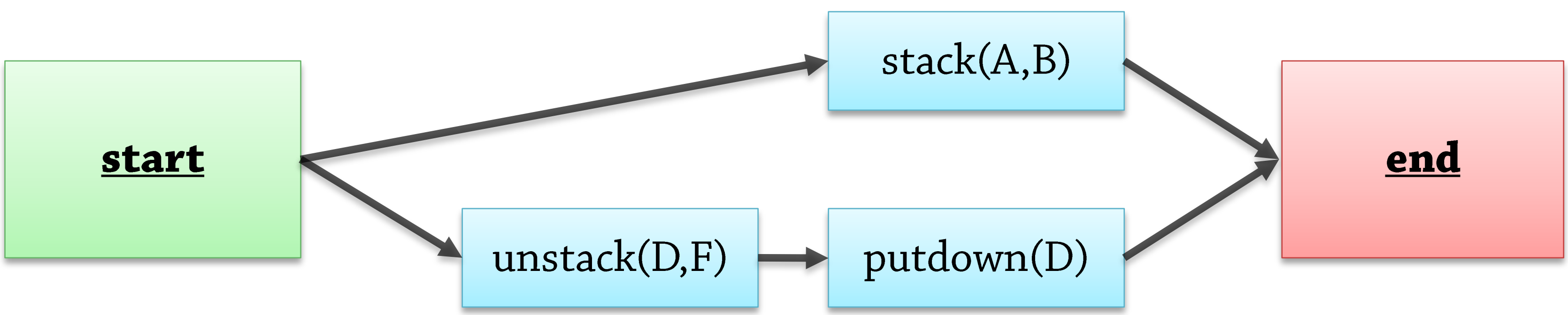


Backward search: A search node is a "current goal"



No Current State during Search!

- With **partial-order plans**: No "current" state or goal!
  - What is true after stack(A,B) below?
    - **Depends** on the order in which **other** actions are executed
    - **Changes** if we insert **new** actions **before** stack(A,B)!



A search node can't correspond to a state or goal!

Search Nodes are Partial Plans

- A node has to contain **more information**: **The entire plan**!
  - The **initial** search node contains the **initial plan**
    - The special **initial** and **goal actions**
    - A **precedence constraint**

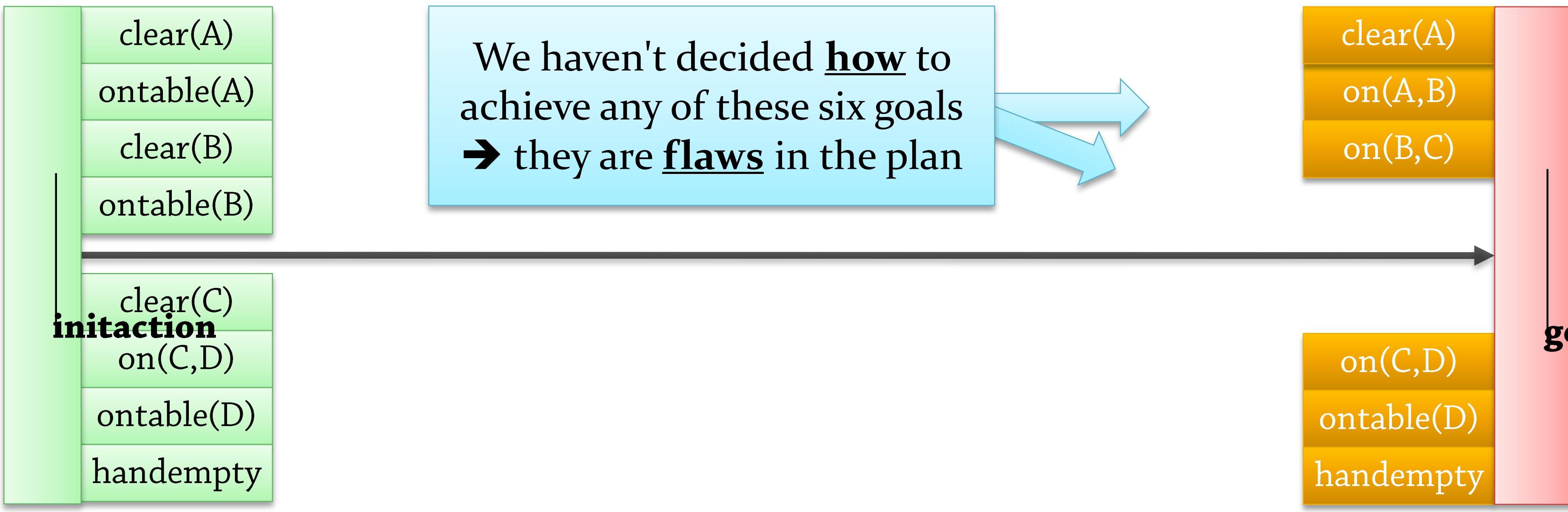
Therefore, this is one form of "plan-space" planning!

Branching Rule

- We need a **branching rule** as well!
  - Forward planning: One successor per action **applicable** in *s*
  - Backward planning: One successor per action **relevant** to *g*
  - POCL planning: One successor for every way that a **flaw in the plan** (**open goal** or **threat**) can be **repaired**

Flaw Type 1: Open Goals

- Open goal:
  - An **action** *a* has a **precondition** *p* with **no incoming causal link**



clear(A) **is** already true in s0, but there is no causal link…

Adding one from s0 means clear(A) **must never be deleted**!
We need other alternatives too: Delete clear(A), then re-achieve it for goalaction…

Flaw Type 1: Open Goals

- To resolve an open goal :
  - Find an action *b* that causes *p*
    - Can be a *new* action
    - Can be an action *already* in the plan, if we can *make* it precede *a*
  - Add a **causal link**

Partial order! This was not possible in backward search…

Essential:
Even if there **is already** an action that causes p, you can still add a **new** action that **also** causes p!

Resolving Open Goals 1

- In this initial Blocks World plan we have **six** open goals
  - We could choose to **find support for clear(A)**:
    - From initaction
    - From a new unstack(B,A), unstack(C,A), or unstack(D,A)
    - From a new stack(A,B), stack(A,C), stack(A,D), or putdown(A)
  - Or we could choose to **find support for on(A,B)**:
    - Only from a new instance of stack(A,B)
  - …

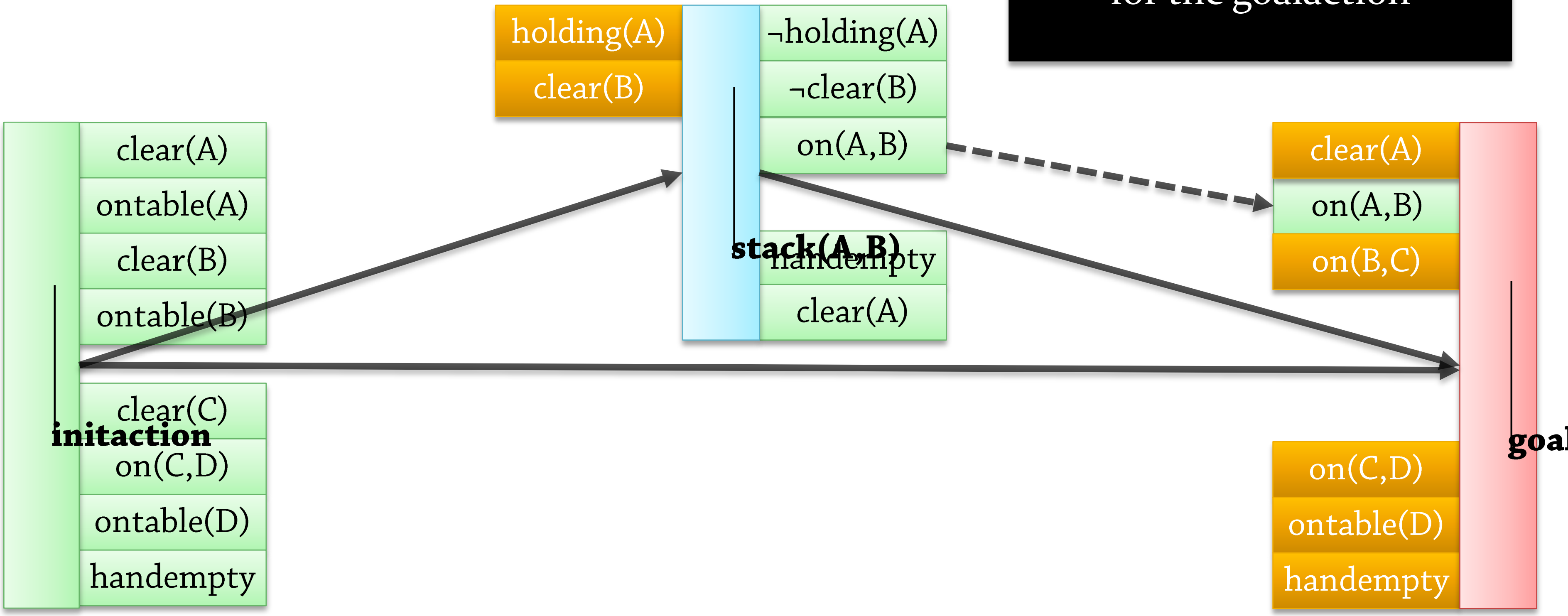
8 distinct successors

1 successor

Resolving Open Goals 2

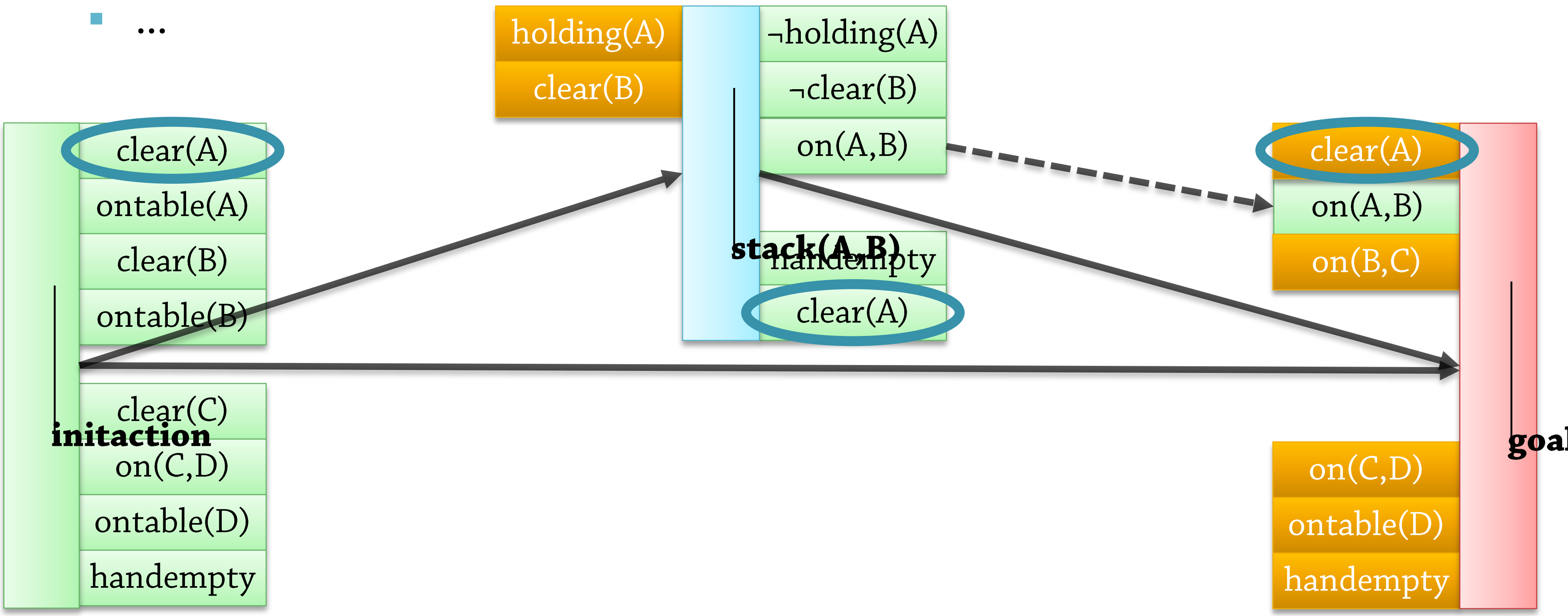
- Suppose we **add stack(A,B)** to **support (achieve) on(A,B)**
  - Must add a **causal link** for **on(A,B)**
    - Dashed line
  - Must **also** add precedence constraints
  - The plan *looks* totally ordered
    - Because it actually only has one "real" action…

Causal link says:
This instance of stack(A,B)
is responsible for
achieving on(A,B)
for the goalaction



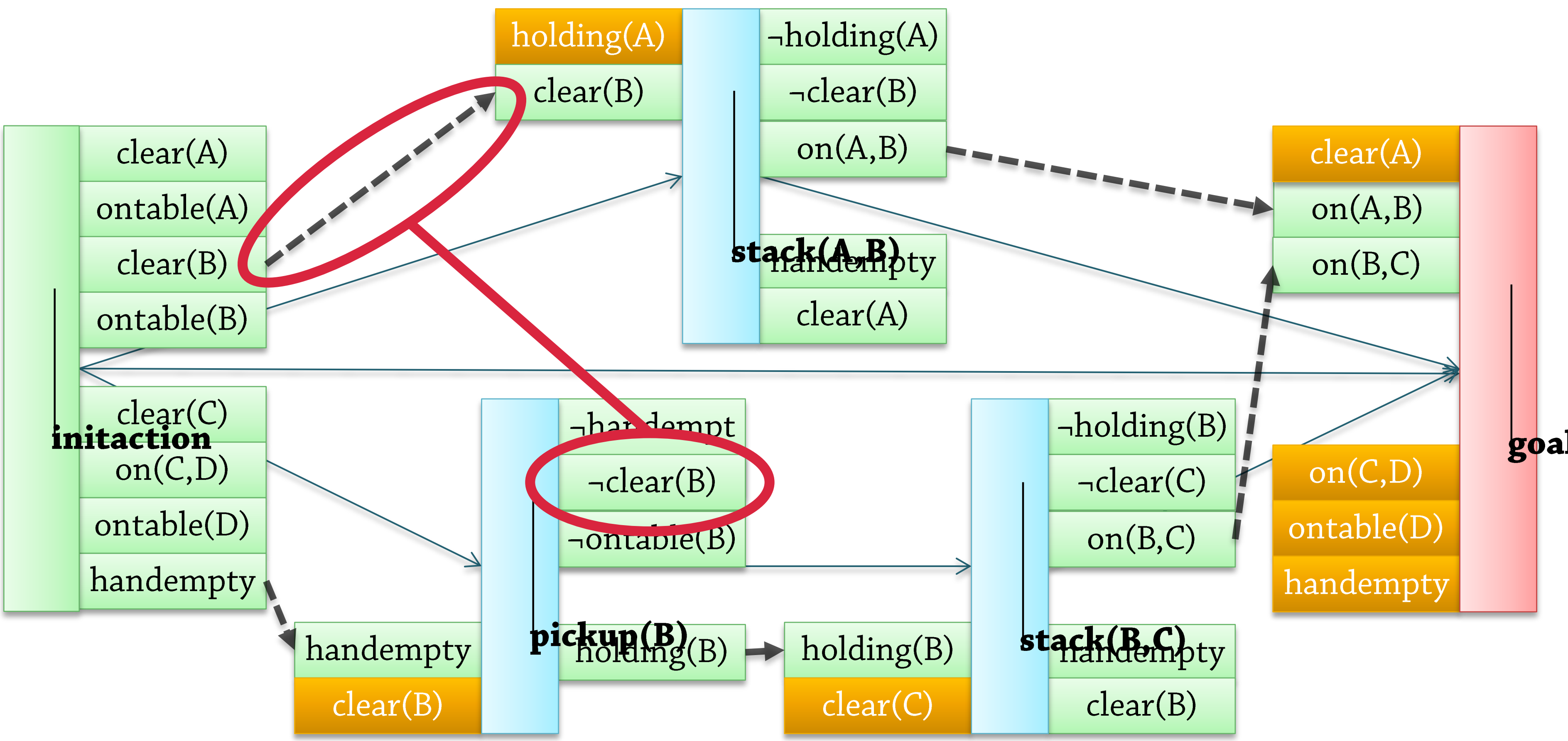
Resolving Open Goals 3

- Now we have 7 open goals (one more!)
  - We can choose to find support for clear(A):
    - From the initaction
    - From the instance of **stack(A,B)** that we just added
    - From a **new** instance of **stack(A,B)**, stack(A,C), stack(A,D), or putdown(A)
    - From a **new** instance of unstack(B,A), unstack(C,A), or unstack(D,A)
  - …



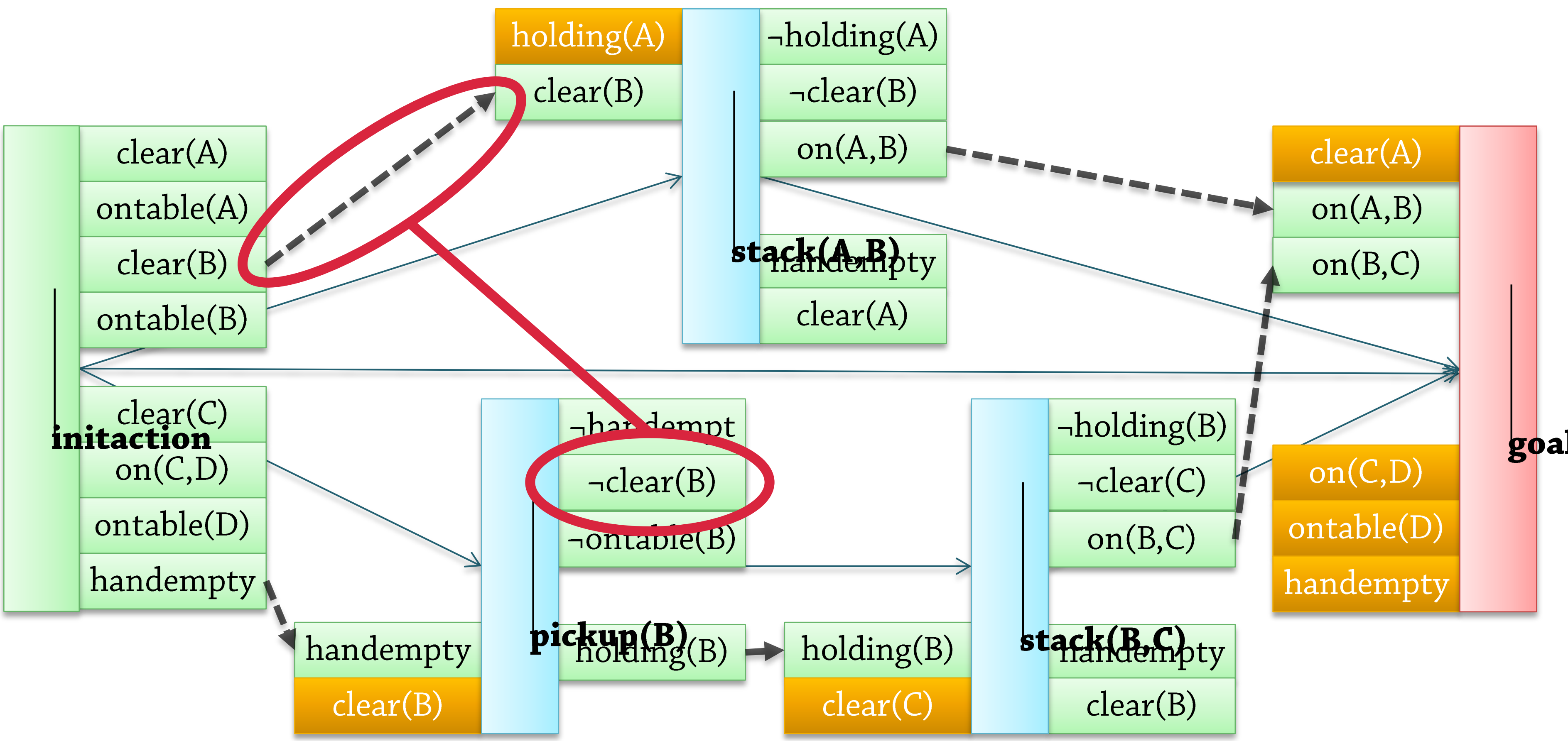
Flaw Type 2: Threats

- Second flaw type: A **threat**
  - initaction **supports** clear(B) for stack(A,B) – there's a causal link
  - pickup(B) **deletes** clear(B), and may occur **between** initaction and stack(A,B)
  - So we can't be certain that clear(B) still holds when stack(A,B) starts!



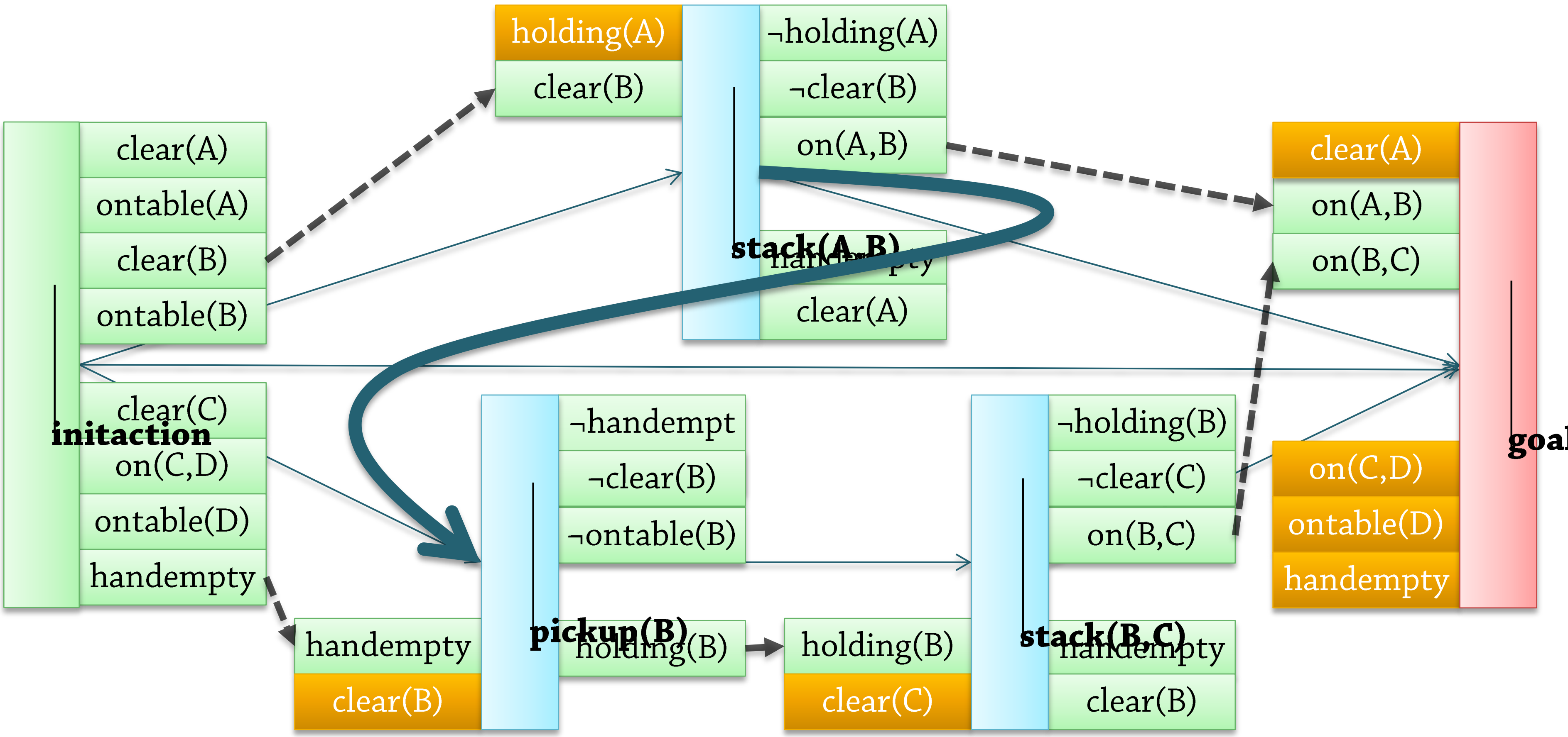
Flaw Type 2: Threats (2)

- Some possible **execution orders**:
  - <…, stack(A,B), pickup(B), …> -- preconditions of stack(A,B) OK
  - <…, pickup(B), stack(A,B), …> -- preconditions of stack(A,B) not satisfied



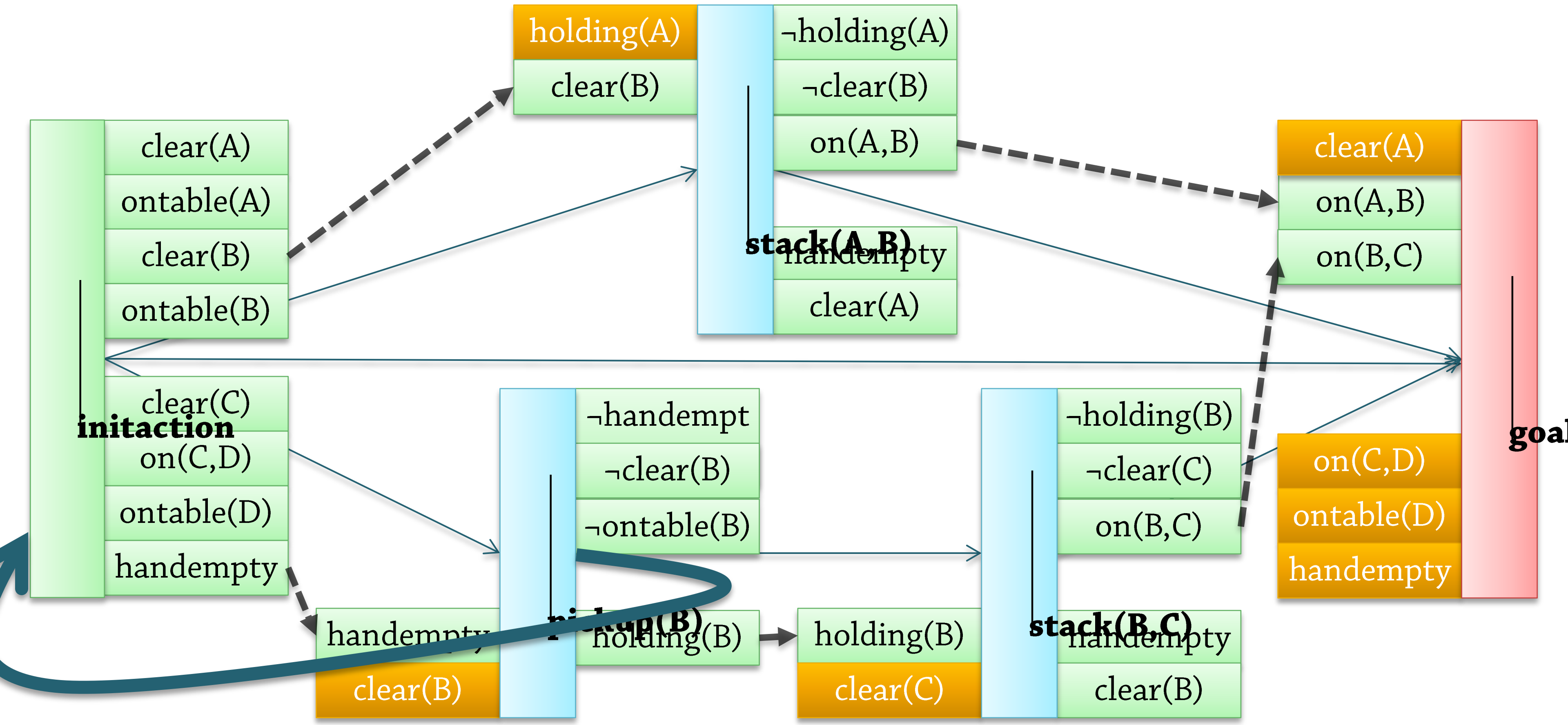
Resolving Threats 1

- How to **make sure** that clear(B) holds when stack(A,B) starts?
  - Alternative 1: The action that **disturbs** the precondition is placed **after** the action that **has** the precondition
    - Only possible if the resulting partial order is consistent (acyclic)!



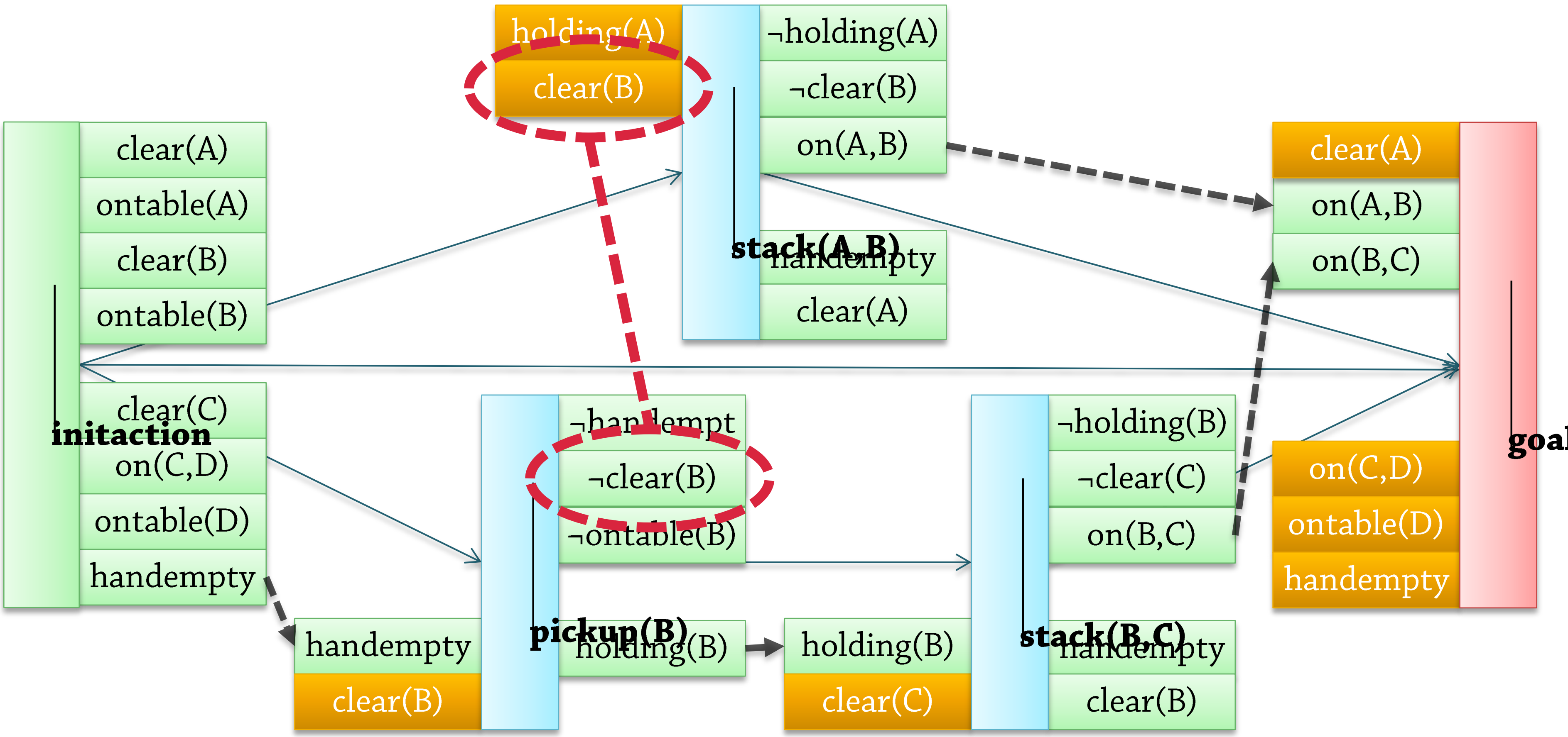
Resolving Threats 2

- Alternative 2:
  - The action that **<u>disturbs</u>** the precondition is placed **<u>before</u>** the action that **<u>supports</u>** the precondition
  - Only possible if the resulting partial order is consistent – not in this case!



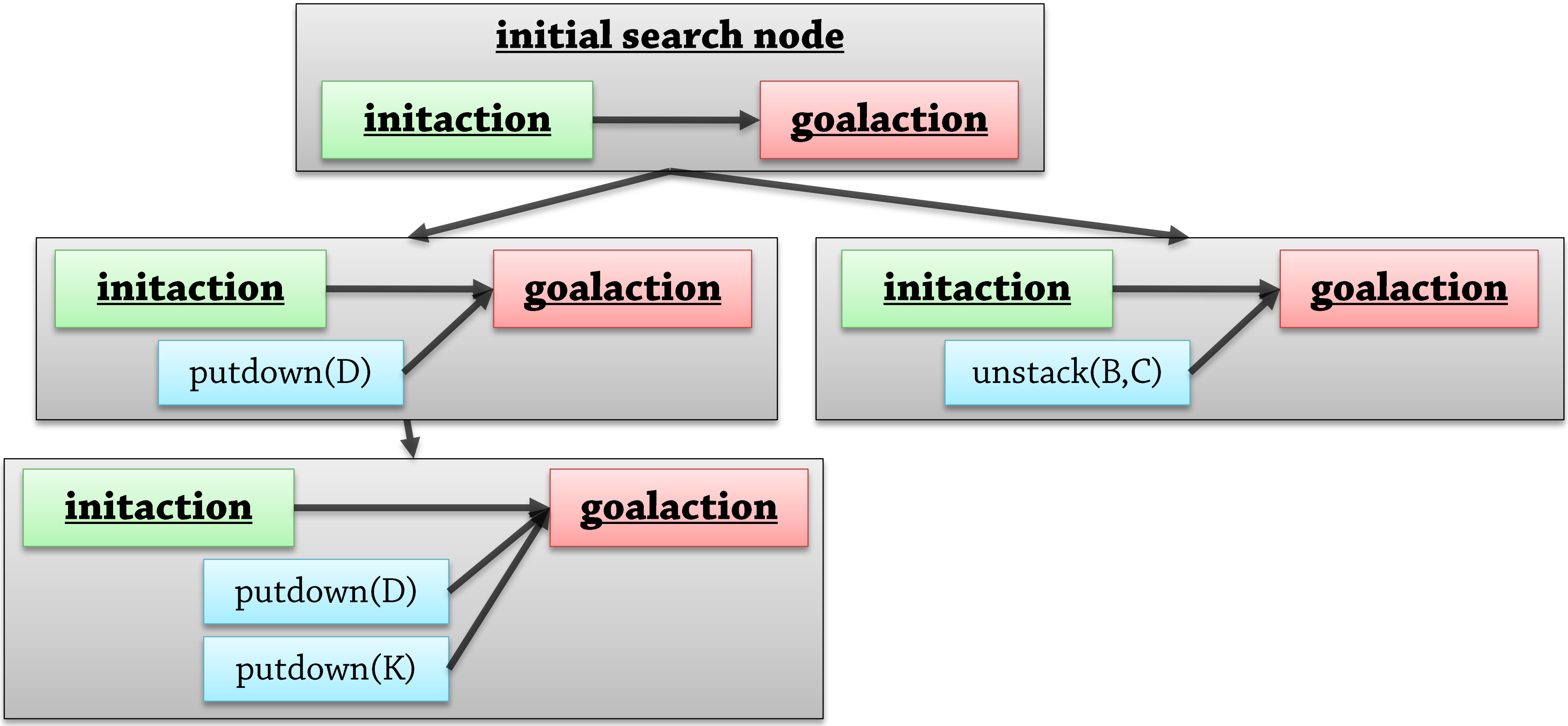
Resolving Threats 3

- Only **causal links** can be threatened!
  - Below, pickup(B) does **not** threaten the precond clear(B) of stack(A,B)
    - We haven't decided yet **how** to achieve clear(B): No incoming causal link
    - So we can't claim that its achievement is threatened!



Search Space

- Gives rise to a **search space**
  - Use search strategies, backtracking, heuristics, … to search **this** space!



The PSP Procedure

- **Plan-Space Planning**:
  - PSP(π)
    flaws ← OpenGoals(π) ∪ Threats(π)
    if *flaws* = ∅ then return π

    select any flaw φ ∈ *flaws*
    resolvers ← FindResolvers(φ, π)
    if *resolvers* = ∅ then return failure // *Backtrack*

    nondeterministically choose a resolver ρ ∈ *resolvers*
    π' ← Refine(ρ, π) // *Actually apply the resolver*
    return PSP(π')
    end
  - Call PSP(the initial plan)
  - PSP is both sound and complete
  - It returns a partially ordered solution plan
    - Any total ordering of this plan will achieve the goals

The plan is complete exactly when there are no remaining flaws (no open goals, no threats)

Not a backtracking point! Resolving one flaw cannot prevent us from resolving other flaws.

Requires heuristics!

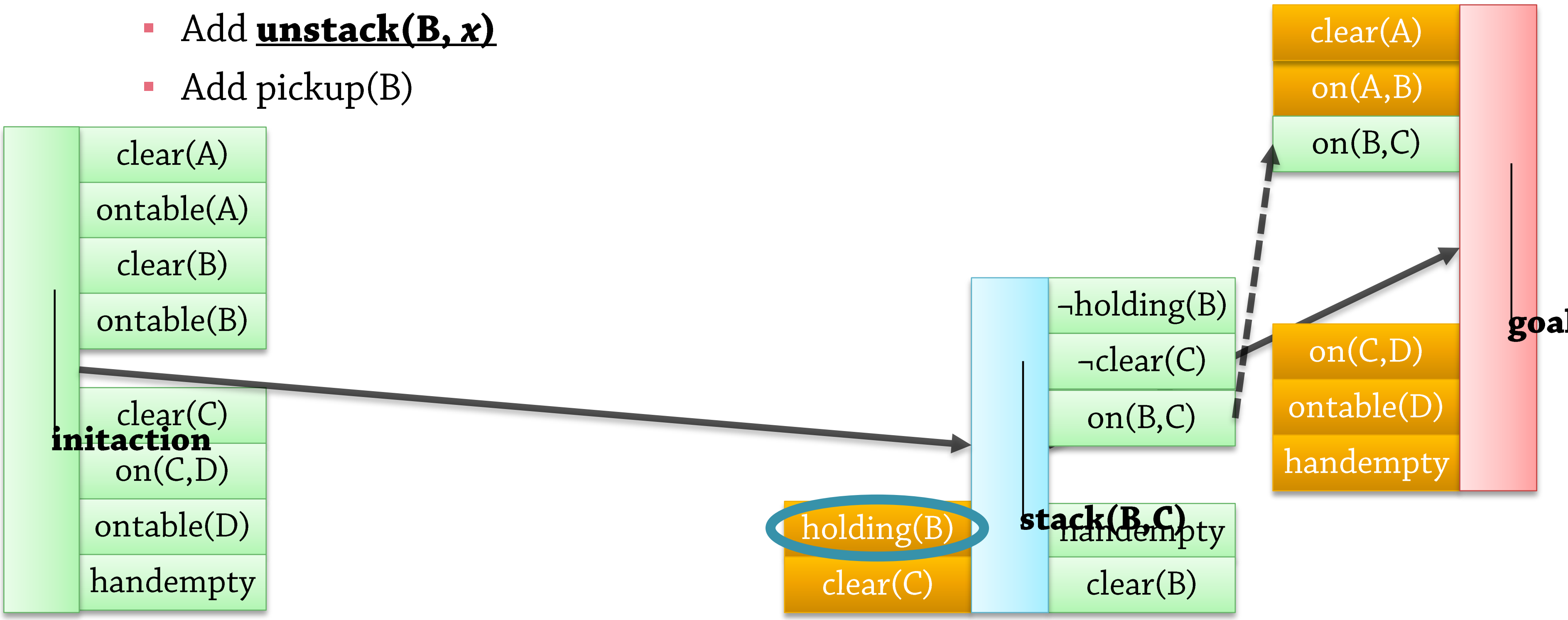
This **is** a backtracking point. For example, a resolver might add an action that solves *this* local flaw, but that cannot be part of a solution.

Partial Action Instantiation

Partial Instantiation

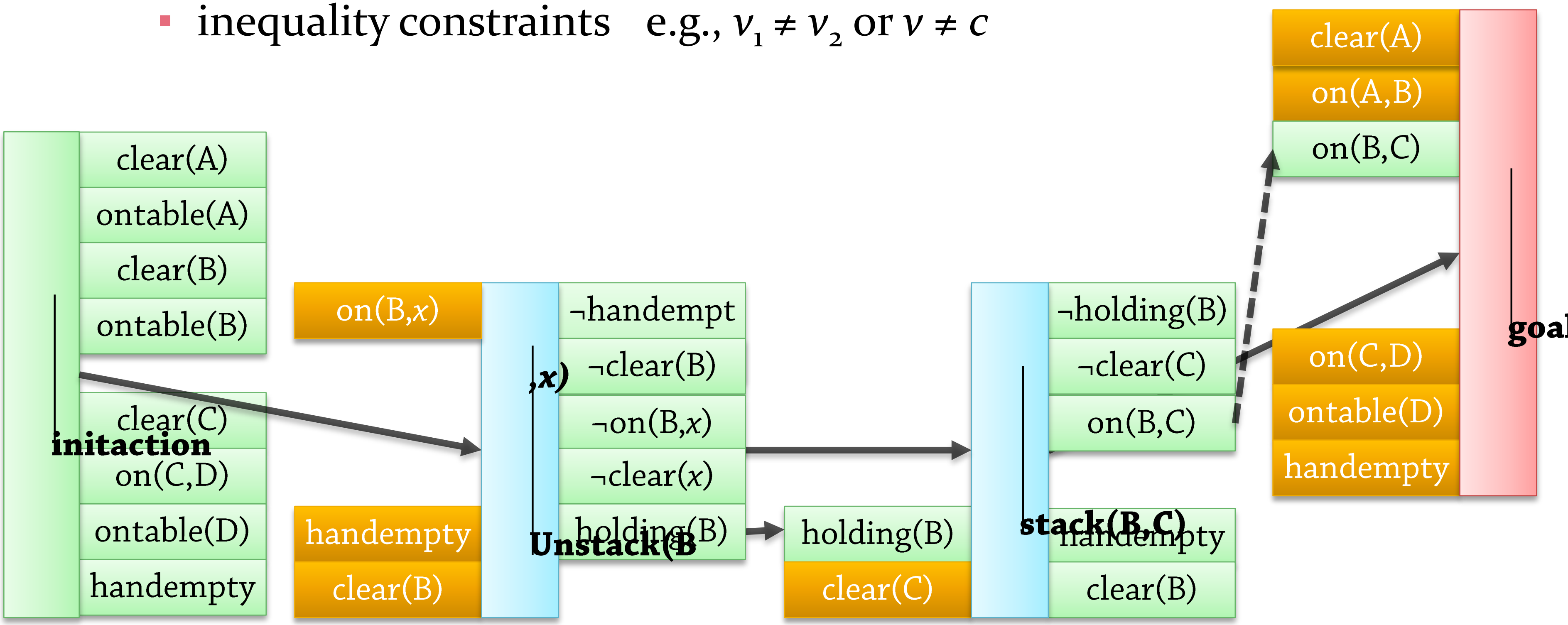
- Suppose we want to achieve holding(B)
  - **Ground** search generates **many** alternatives
    - Add unstack(B,A), unstack(B,F), unstack(B,G), …
    - Add pickup(B)
  - Let's take the idea of **least commitment** one step further
  - **Lifted** search generates two **partially instantiated** alternatives
    - Add **unstack(B, *x*)**
    - Add pickup(B)

So far, we see no reason why we should unstack B from any **specific** block!



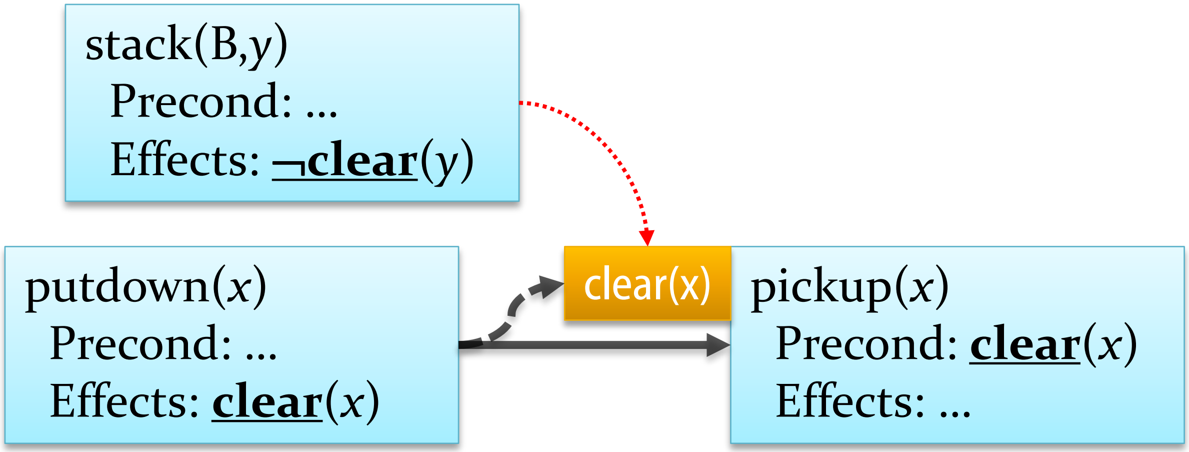
Partial-Order Plans

- A **lifted partial-order plan** consists of:
  - A set of **possibly unground actions**
  - A set of **precedence constraints**: *a* must precede *b*
  - A set of **causal links**: action *a* establishes the precond *p* needed by *b*
  - A set of **binding constraints**:
    - equality constraints e.g., $v_1 = v_2$ or $v = c$
    - inequality constraints e.g., $v_1 \neq v_2$ or $v \neq c$



Resolving Threats

- Another way of **resolving threats** for lifted plans:
  - For partly uninstantiated actions, we may find **potential** threats
    - stack(B,y) **may** threaten the causal link, but only if x=y
    - Can be resolved by adding a constraint: x != y



Complete Example

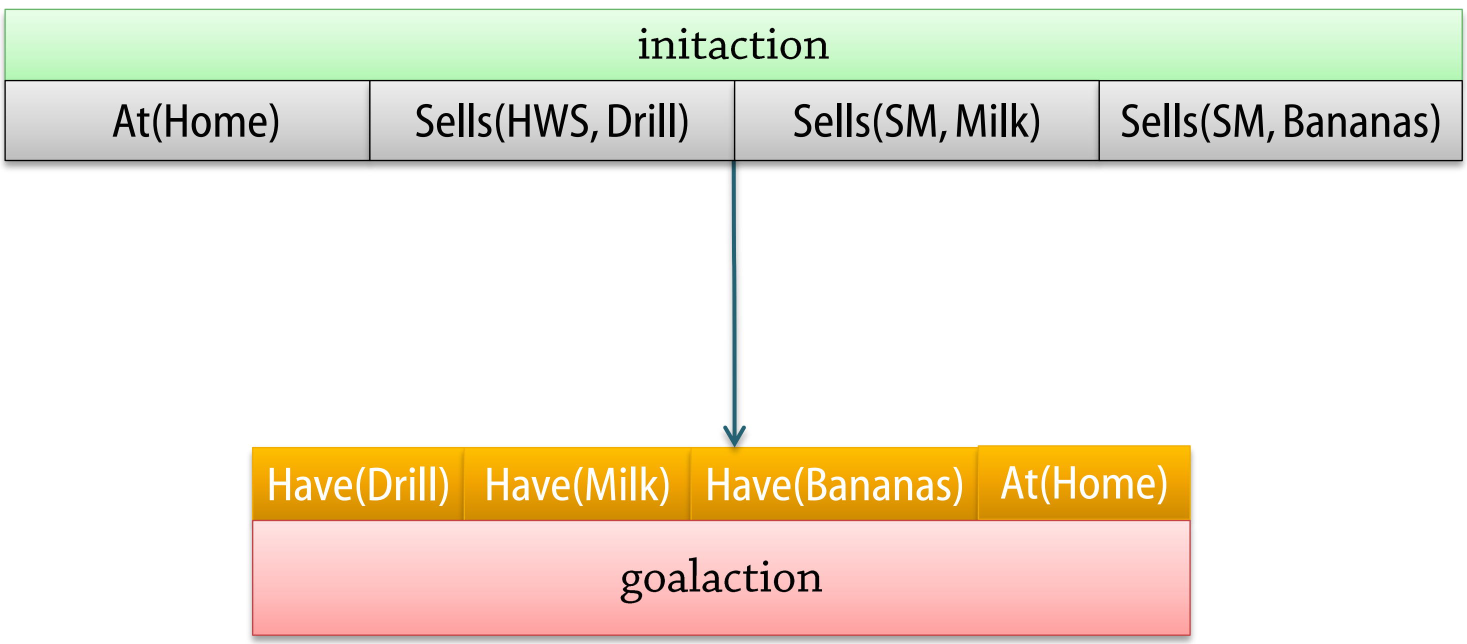
Example

- Running Example: Similar to an example in AIMA
  - Russell and Norvig's *Artificial Intelligence: A Modern Approach* (1st ed.)

  - Operator **Go(from,to)**
    - Precond: At(from)
    - Effects: At(to), ¬At(from)
  - Operator **Buy(product, store)**
    - Precond: At(store), Sells(store, product)
    - Effects: Have(product)
  - **Initial state**
    - At(Home), Sells(HWS, Drill), Sells(SM, Milk), Sells(SM, Bananas)
  - **Goal**
    - At(Home), Have(Drill), Have(Milk), Have(Bananas)

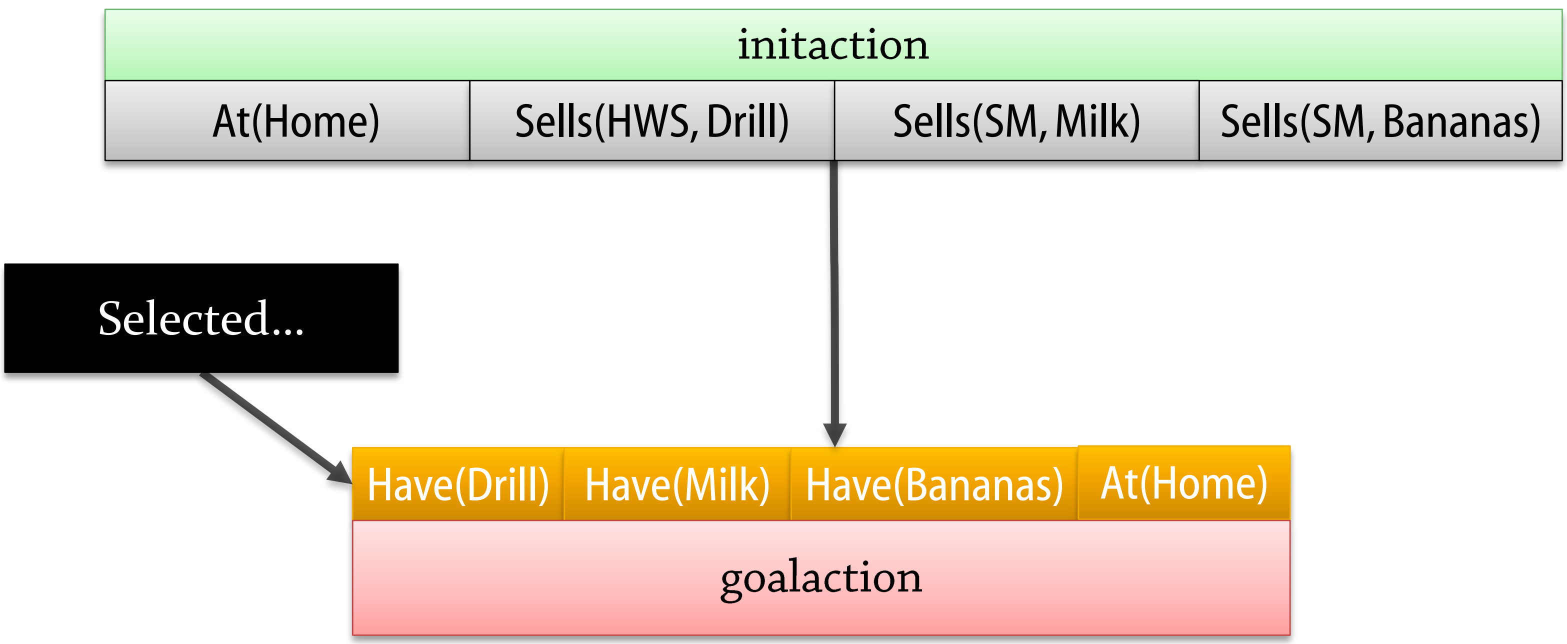
Example (continued)

- PSP takes a plan π as its argument
  - Initial plan: **initaction**, **goalaction**, and an ordering constraint



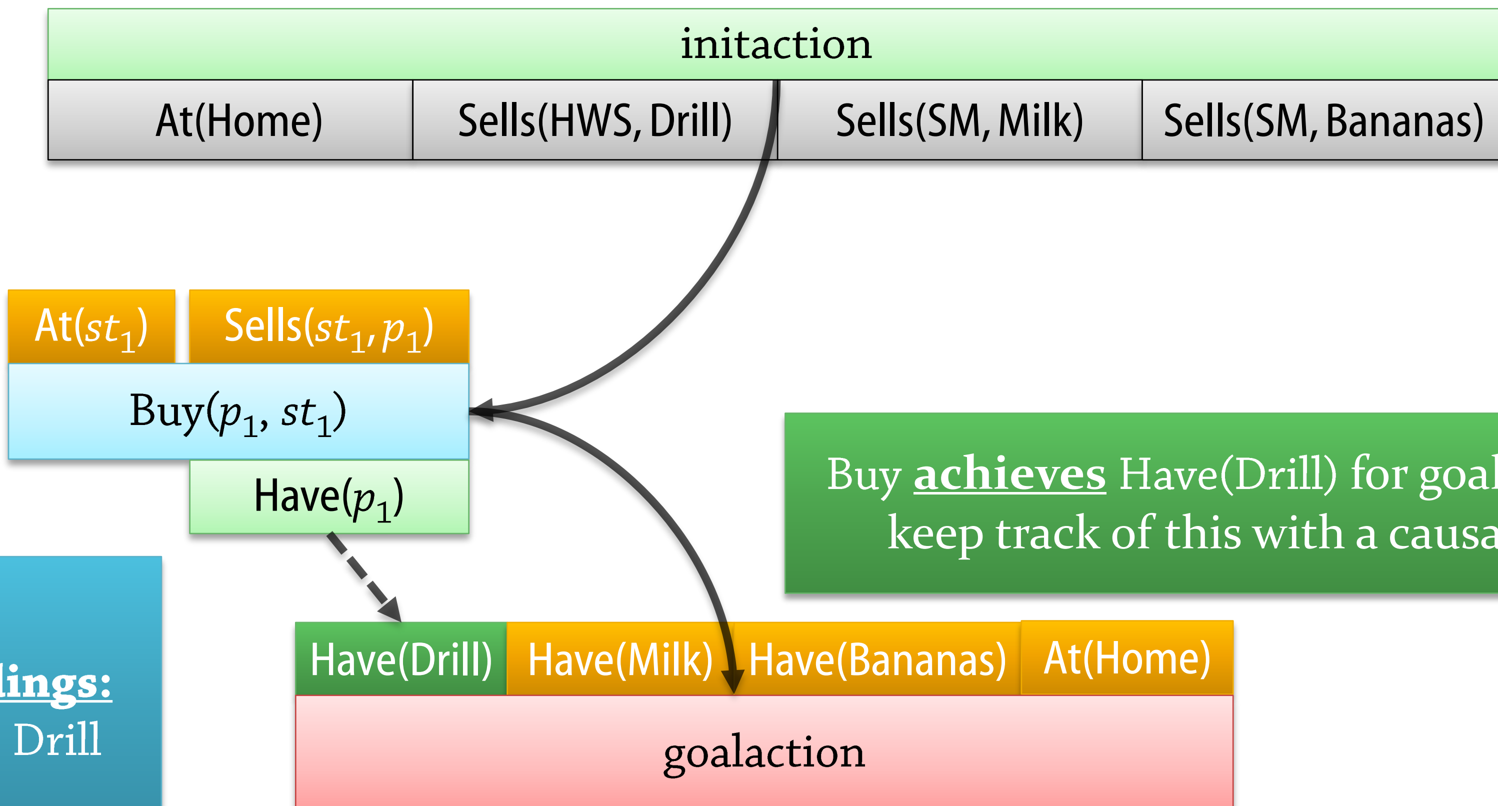
Example (continued)

- Four **flaws** exist: Open goals
  - Suppose our heuristics tell us to resolve Have(Drill) first



Example (continued)

- Have(drill) is not achieved by any action in the current plan
- But **Buy(product, store)** achieves Have(product)
  - Partially instantiate:
    Buy(Drill, store) (right now we don't care **where** we buy it)

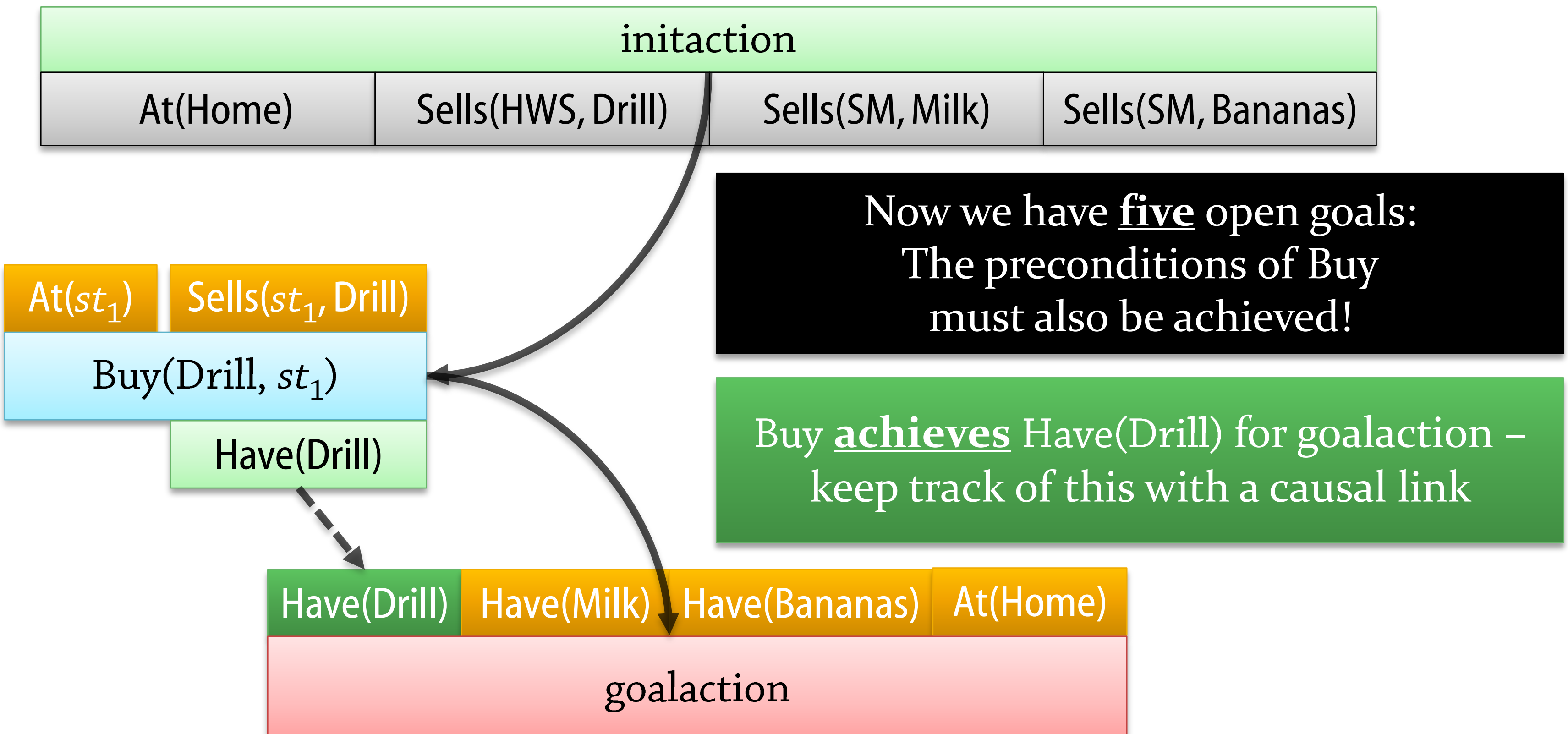


Buy **achieves** Have(Drill) for goalaction – keep track of this with a causal link

Bindings:
p1 = Drill

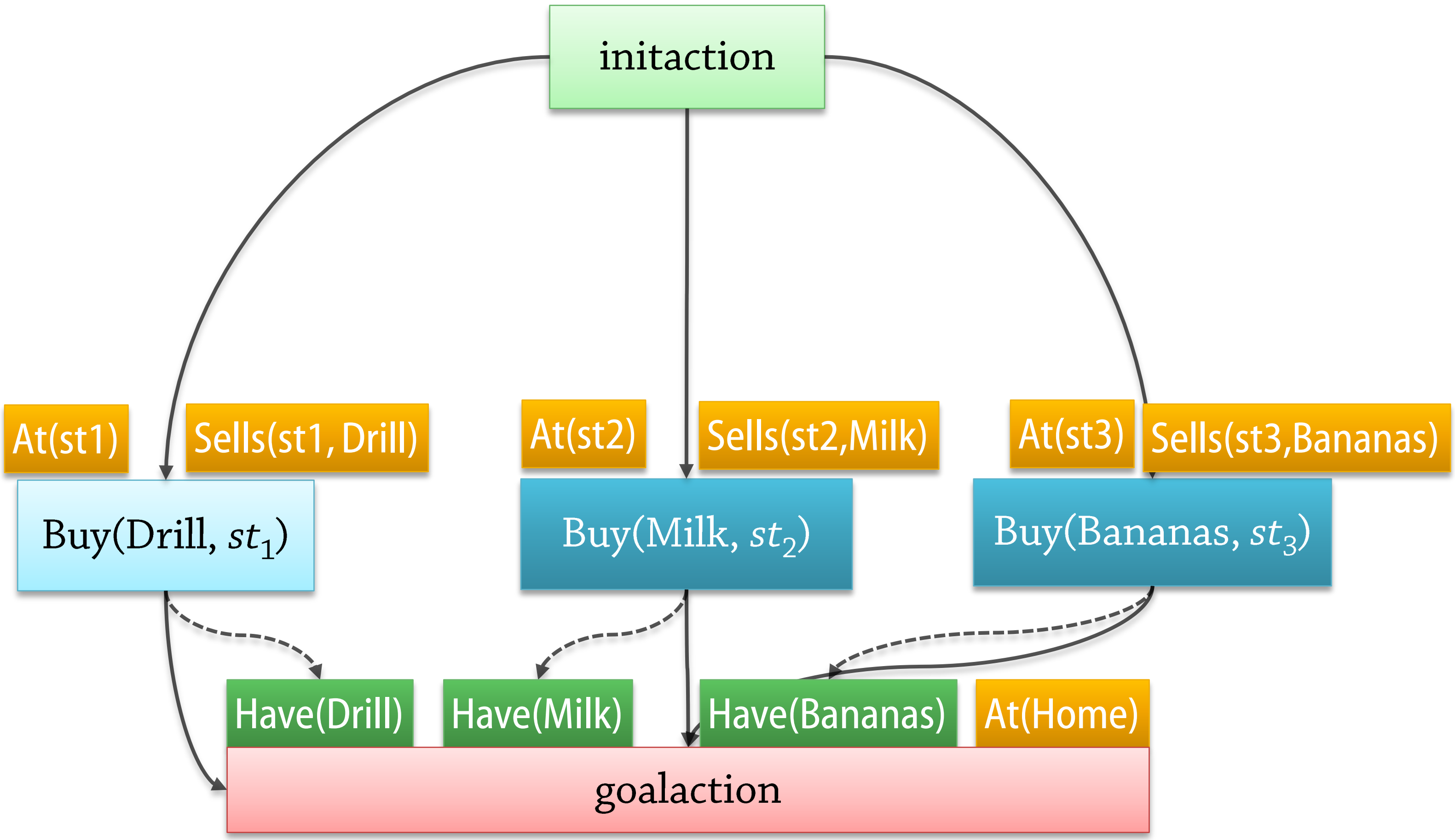
Example (continued)

- **<u>Alternative Notation</u>** for simplicity
  - Variable bindings are implicit in the diagram



Example (continued)

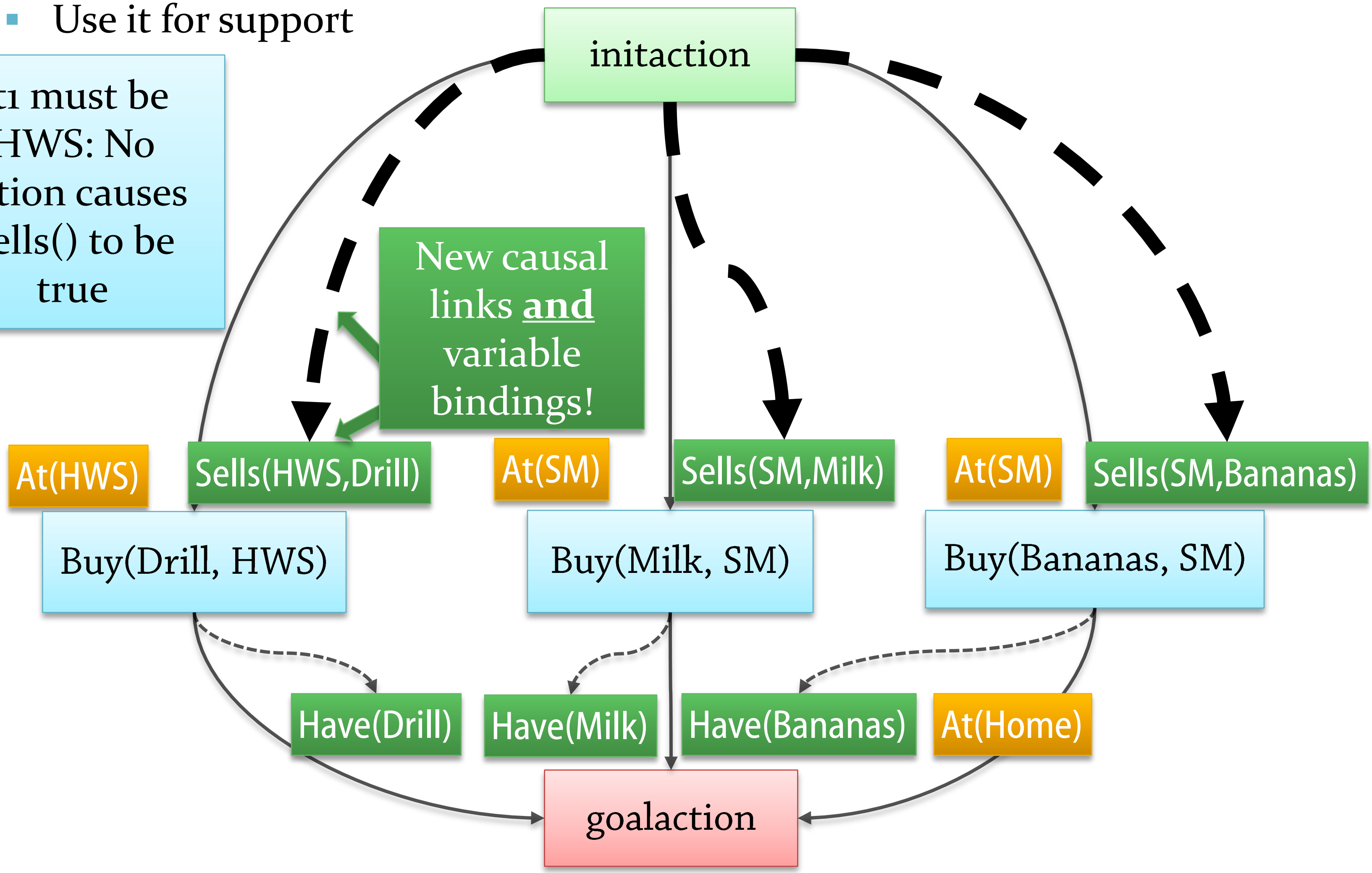
- The first **three** refinement steps
  - These are the only possible ways to establish the Have preconditions
  - We don't care in which **order** we buy things!



Example (continued)

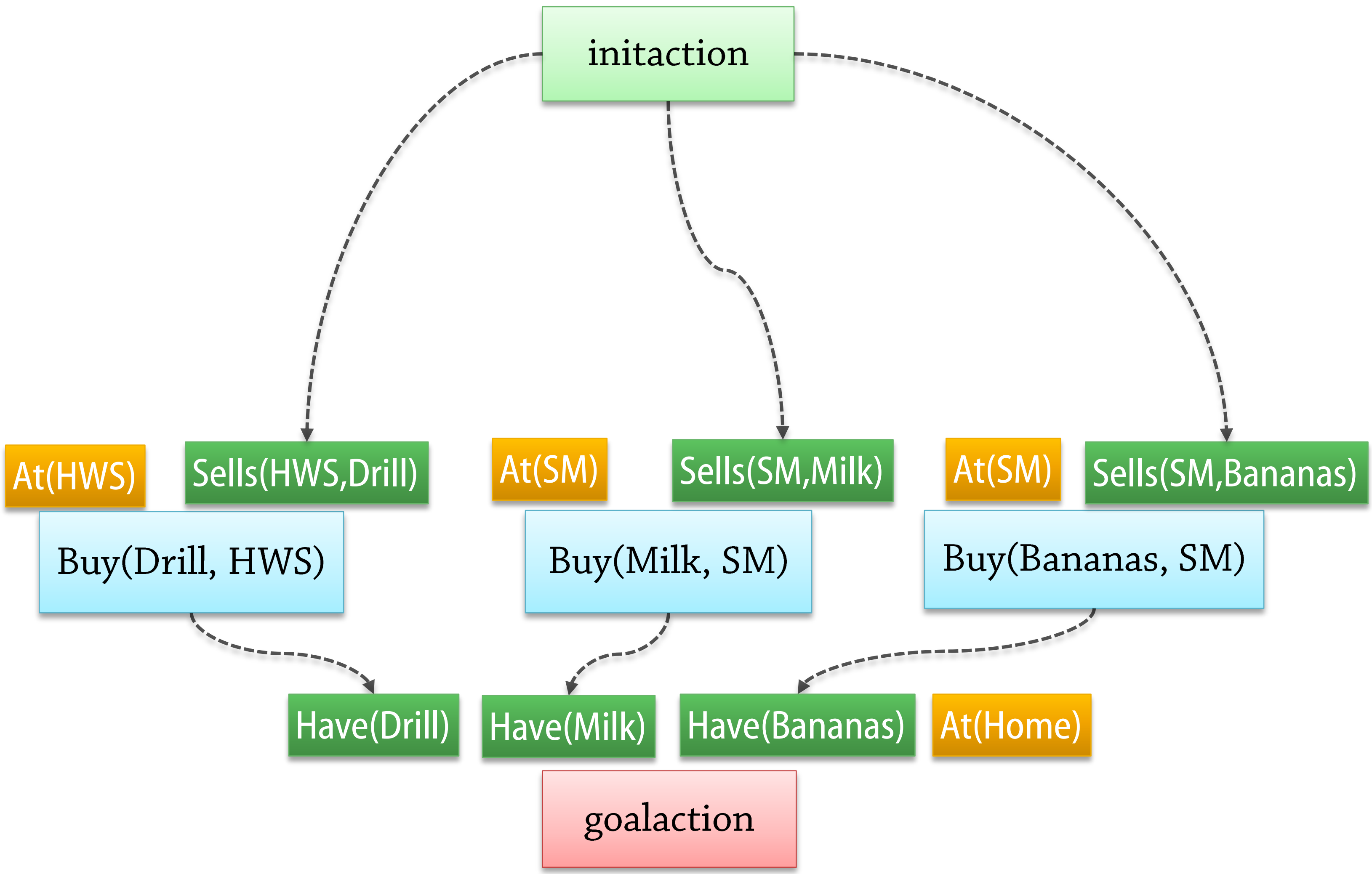
- Three more refinement steps
  - No action causes Sells(...) to be true – except the “fake” initial action!
  - Use it for support

st1 must be HWS: No action causes Sells() to be true



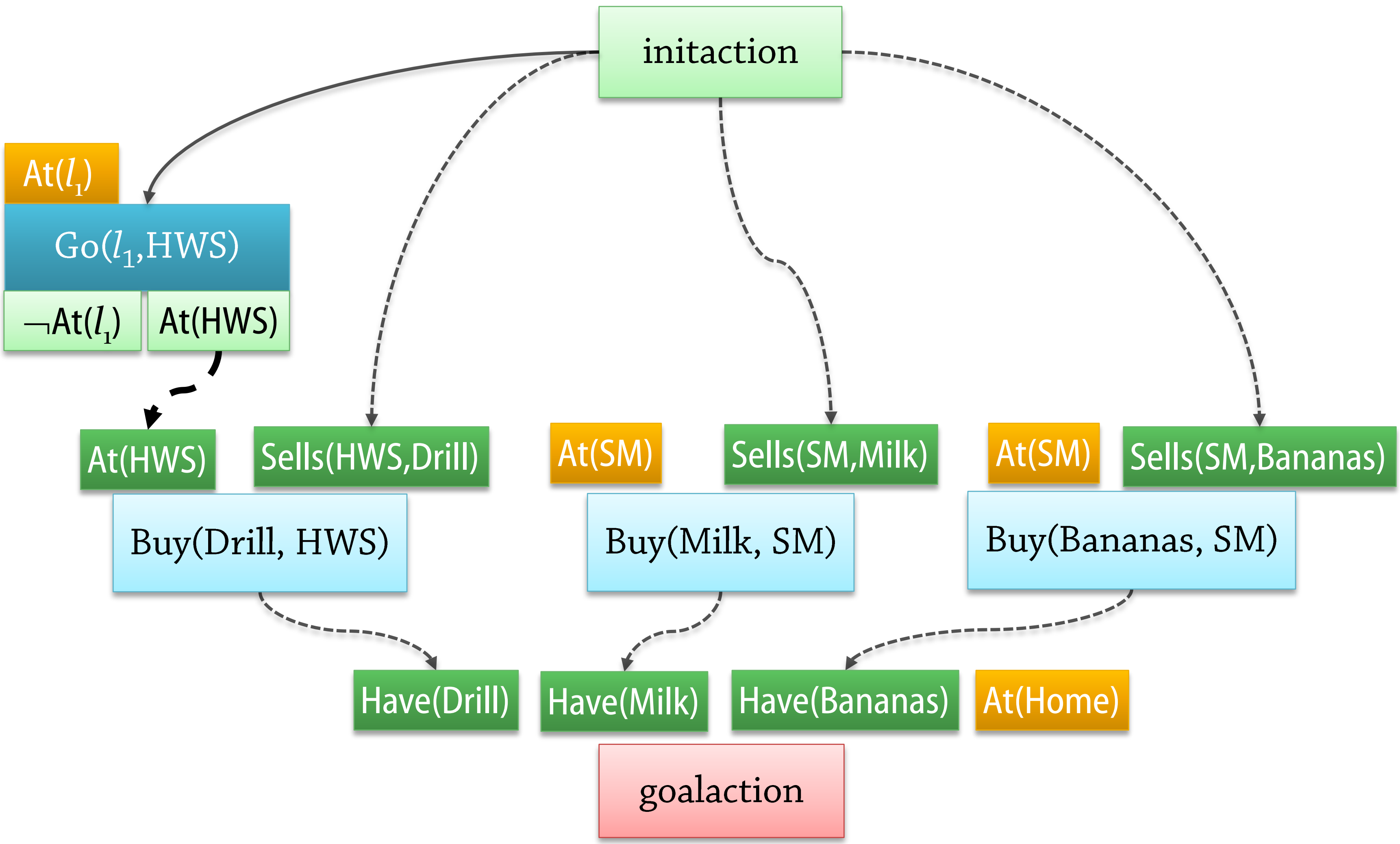
Example (continued)

- It's getting messy!
  - Let's omit the precedence constraints that are implicit in causal links…



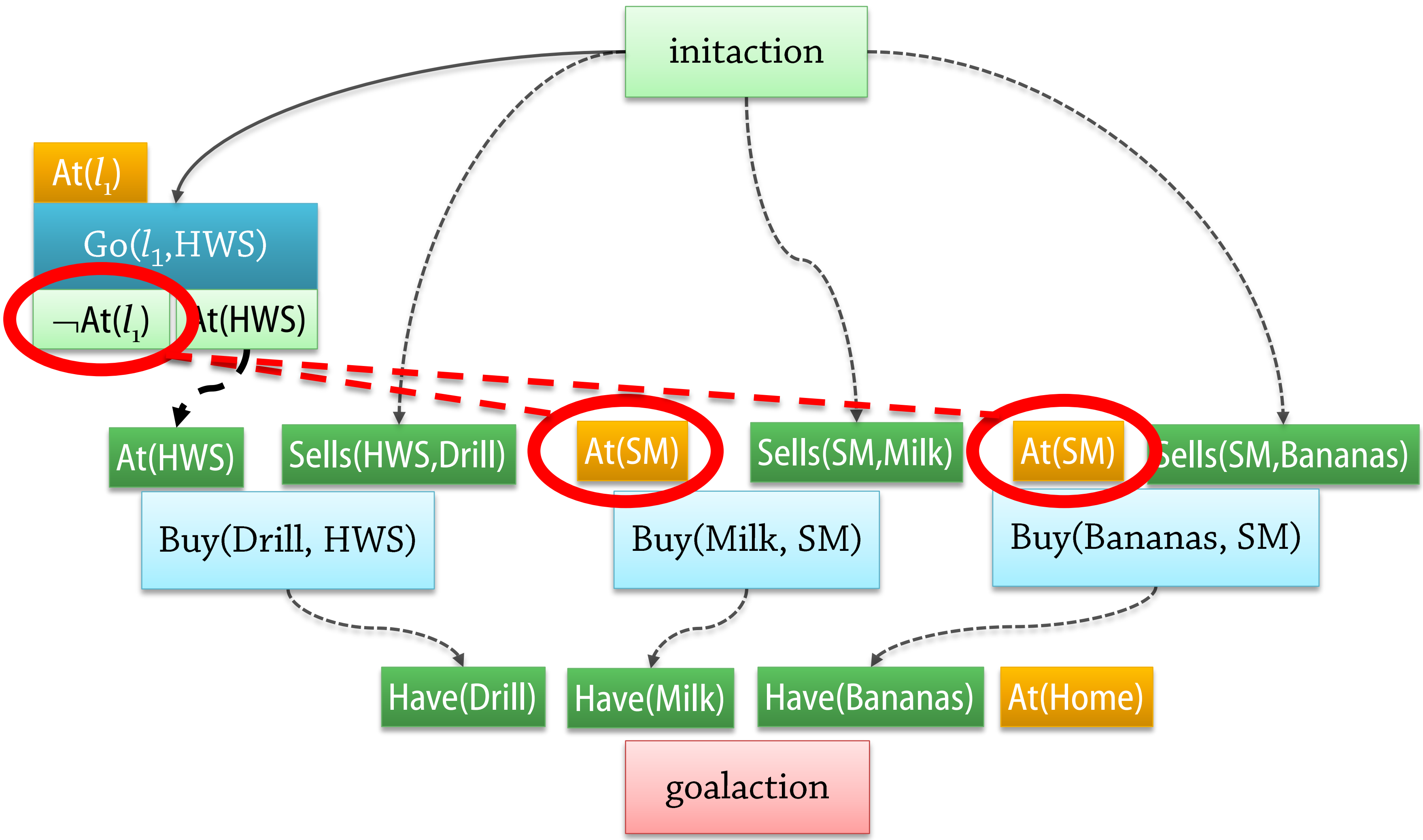
Example (continued)

- To establish At(HWS): **Must** go there from **somewhere**



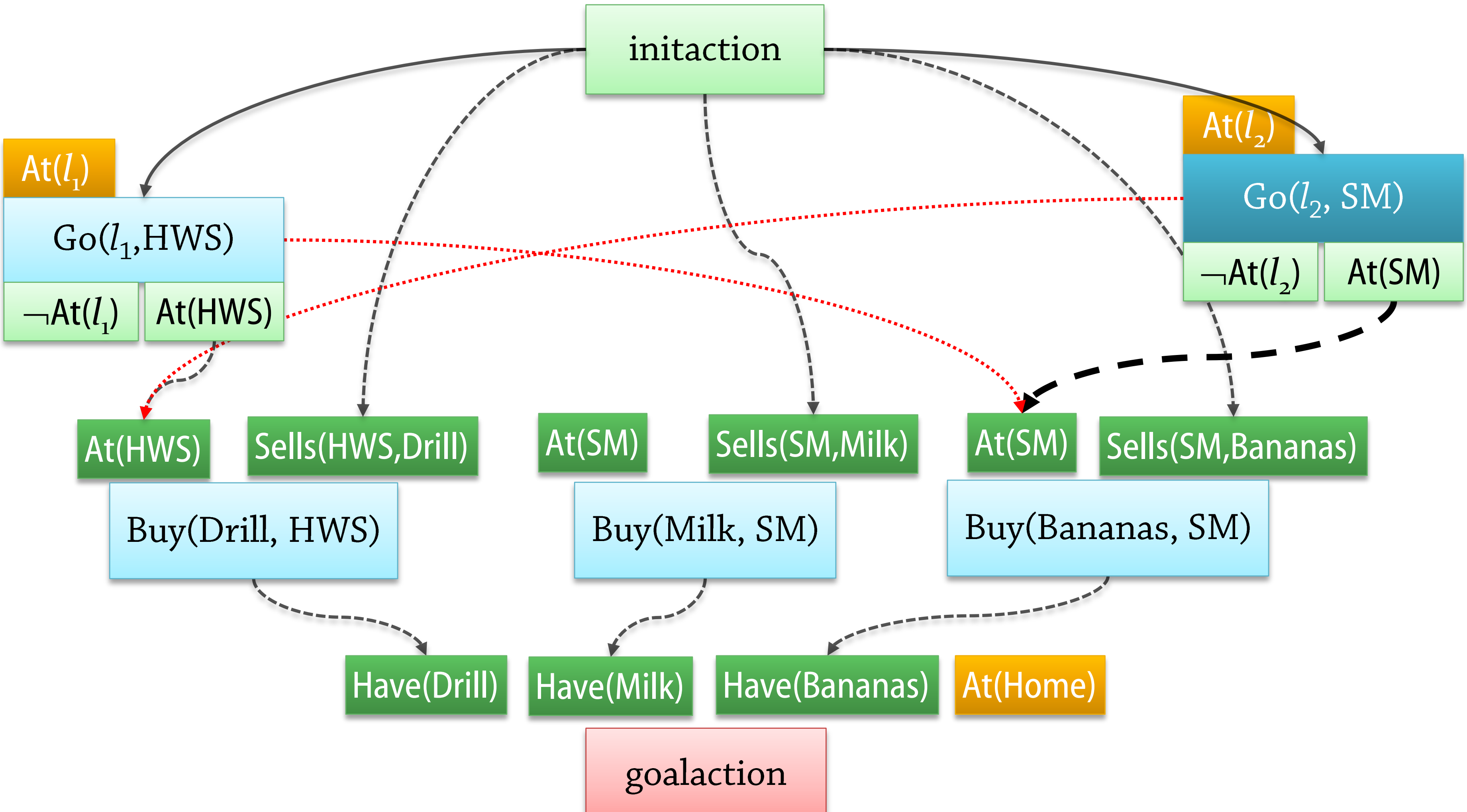
Example (continued)

- Does $\neg$at(l_1) threaten At(SM)?
  - No! Only a **causal link** to At(SM) can be threatened



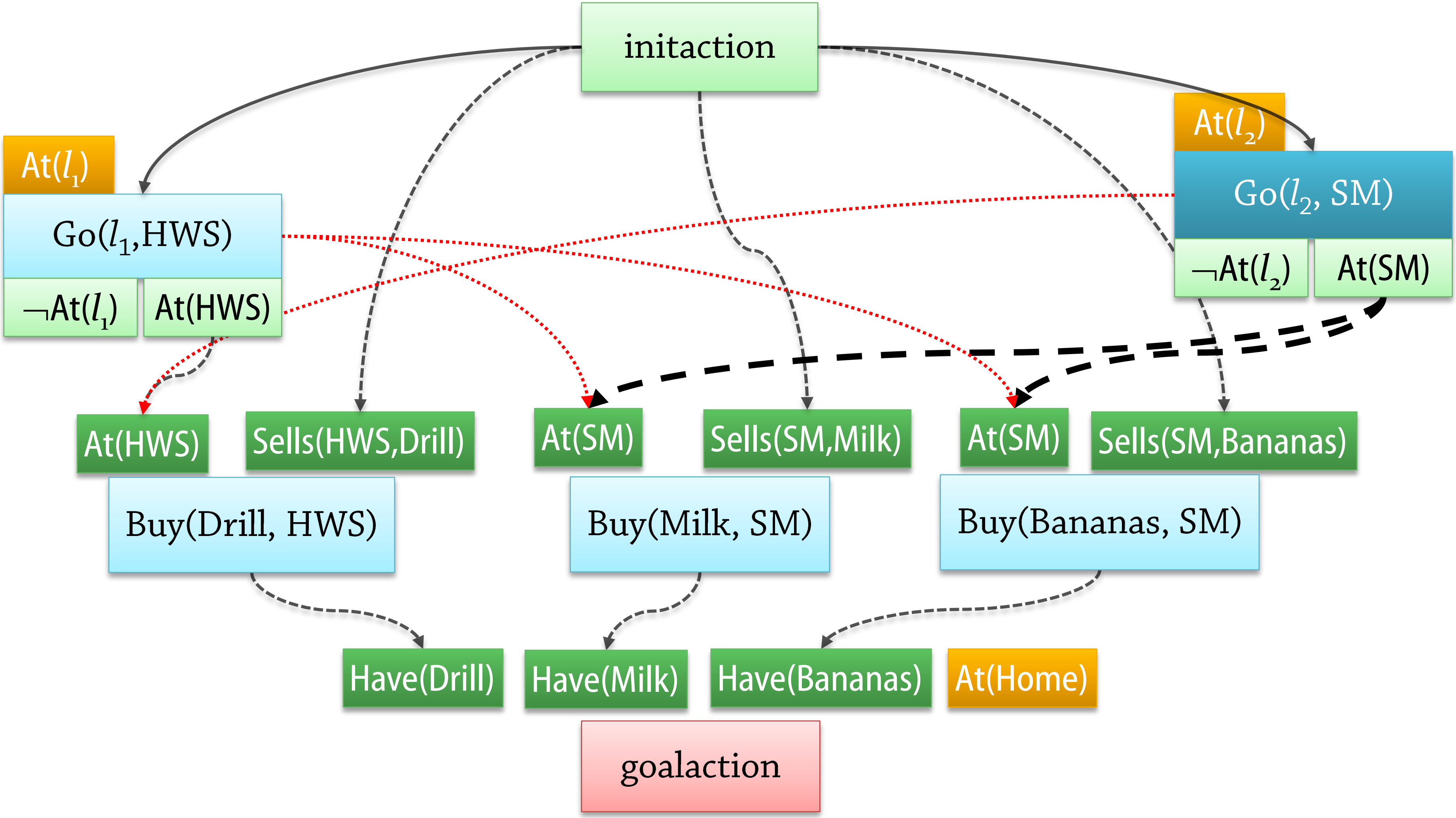
Example (continued)

- To establish At(SM): Must go there from **somewhere**
  - Mutual **threats**...



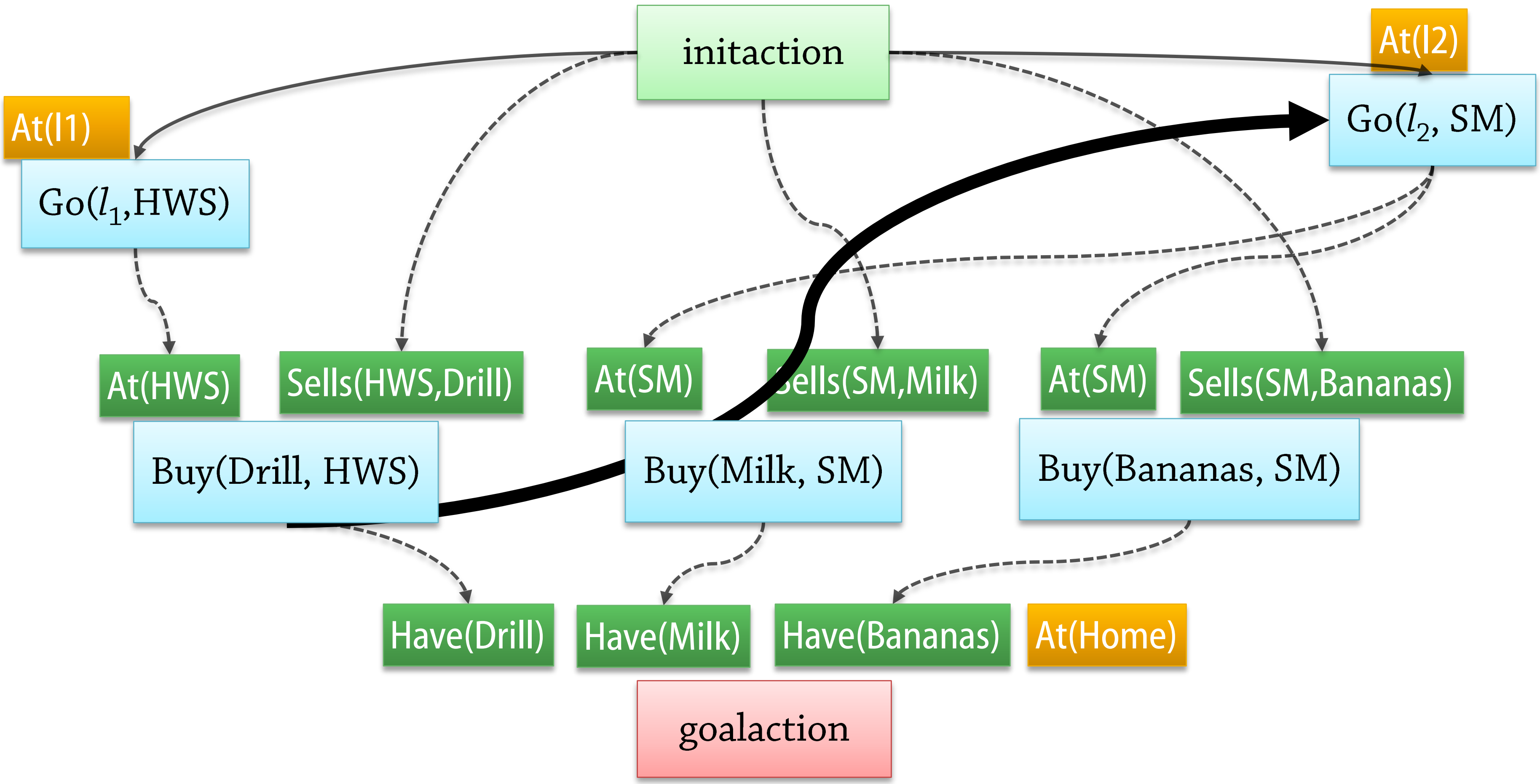
Example (continued)

- Let's use the same action for **both** At(SM) preconditions...
  - More threats – could deal with them now or wait



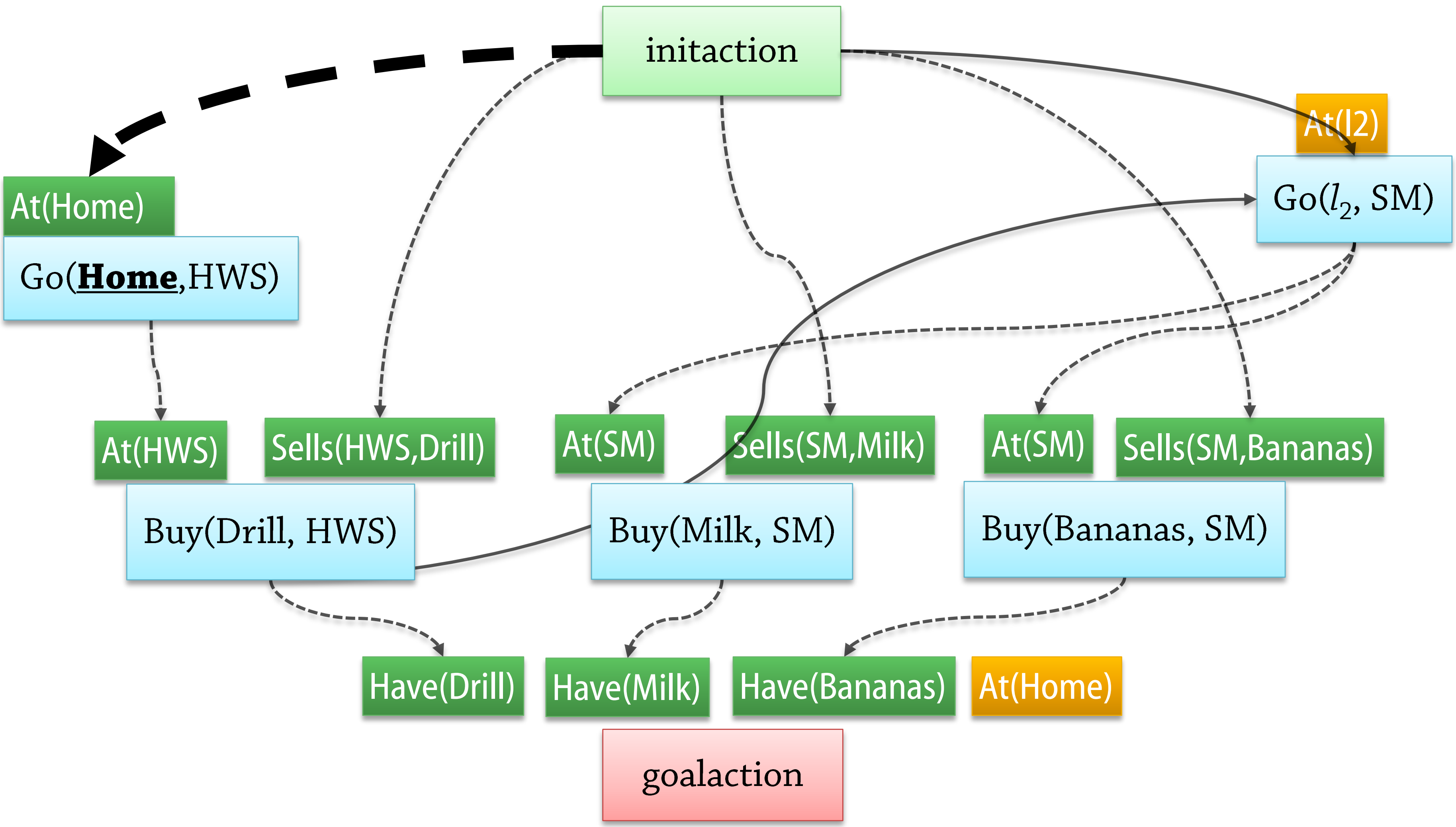
Example (continued)

- **Nondet. choice**: how to resolve the threat to At(HWS)?
  - Our choice: make the “requirer” Buy(Drill) precede the “threatener” Go(l2, SM)
  - Also happens to resolve the other two threats
    - “Threatener” Go(l1, HWS) before “achiever” Go(l2, SM)



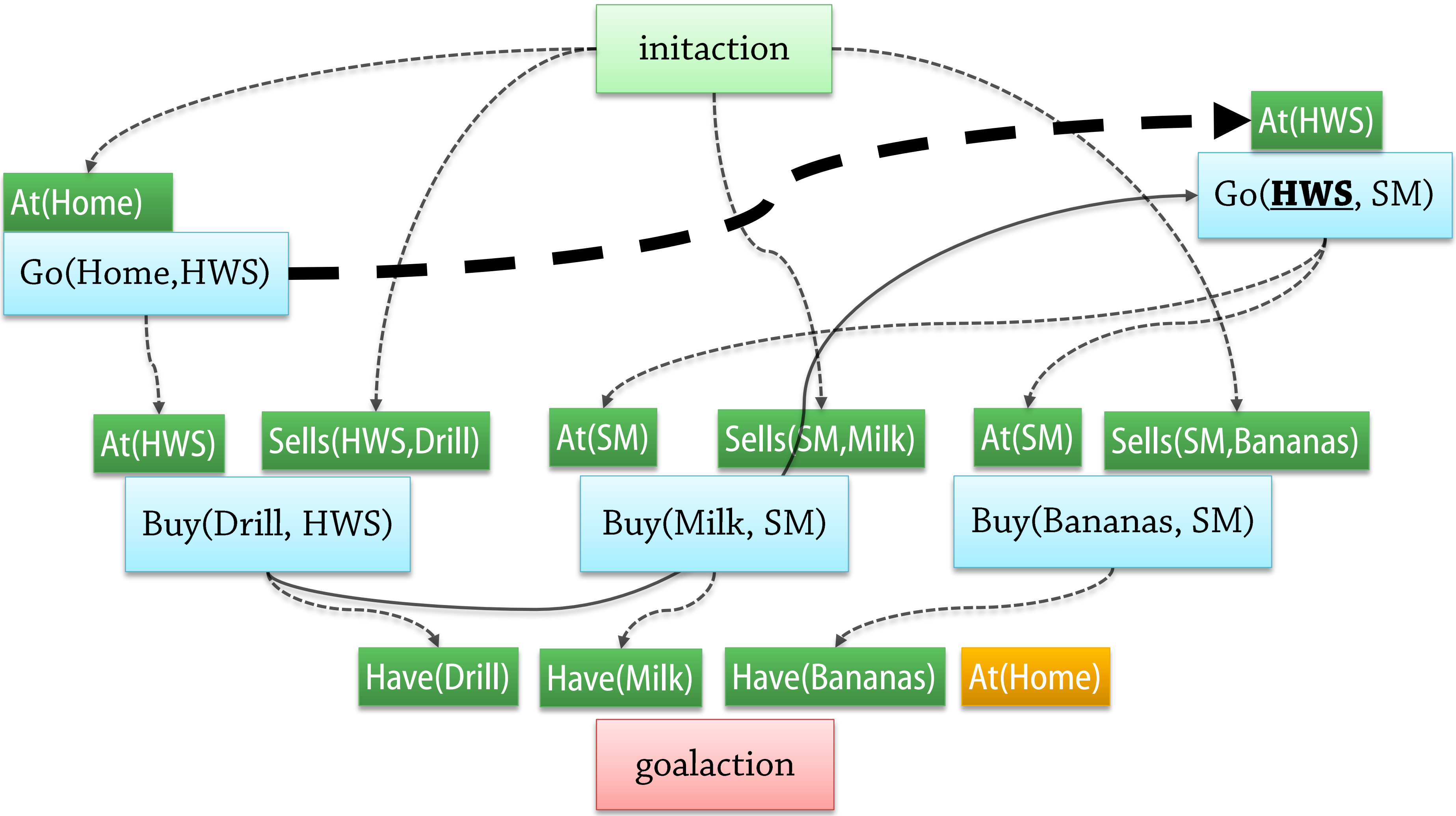
Example (continued)

- **<u>Nondet. choice</u>**: how to establish At(l_1)?
  - We'll do it from initaction, with l_1=Home



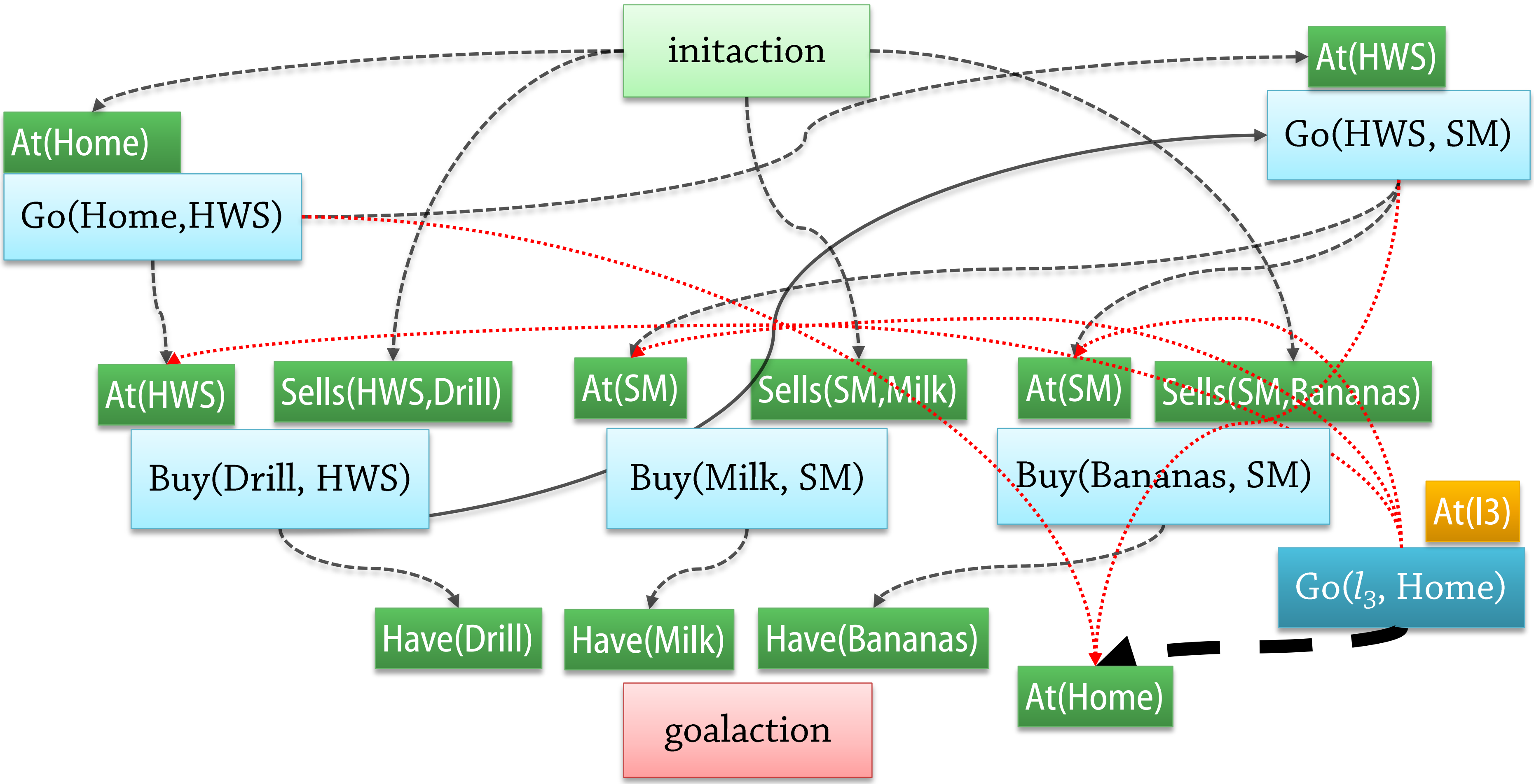
Example (continued)

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



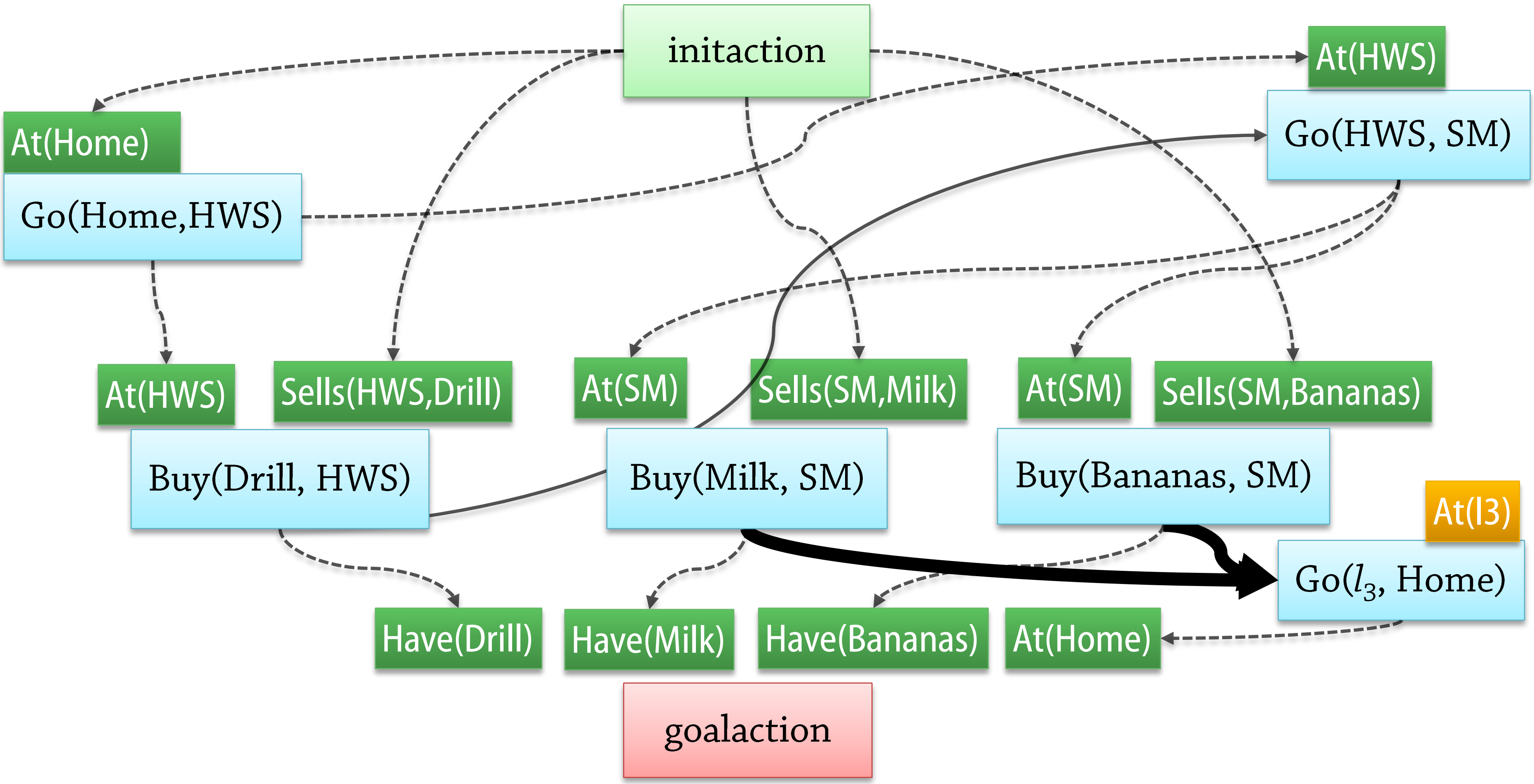
Example (continued)

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



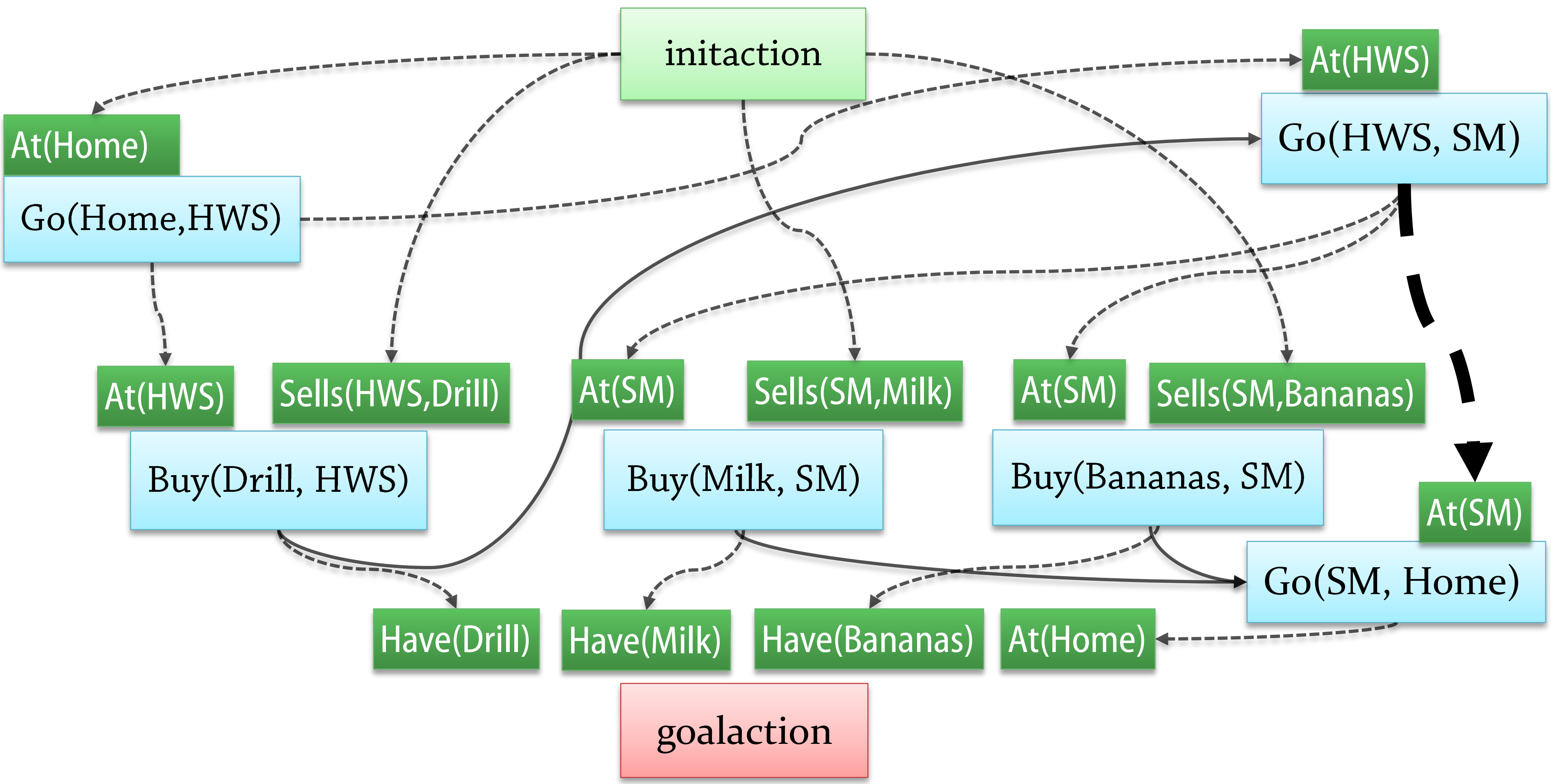
Example (continued)

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



Final Plan

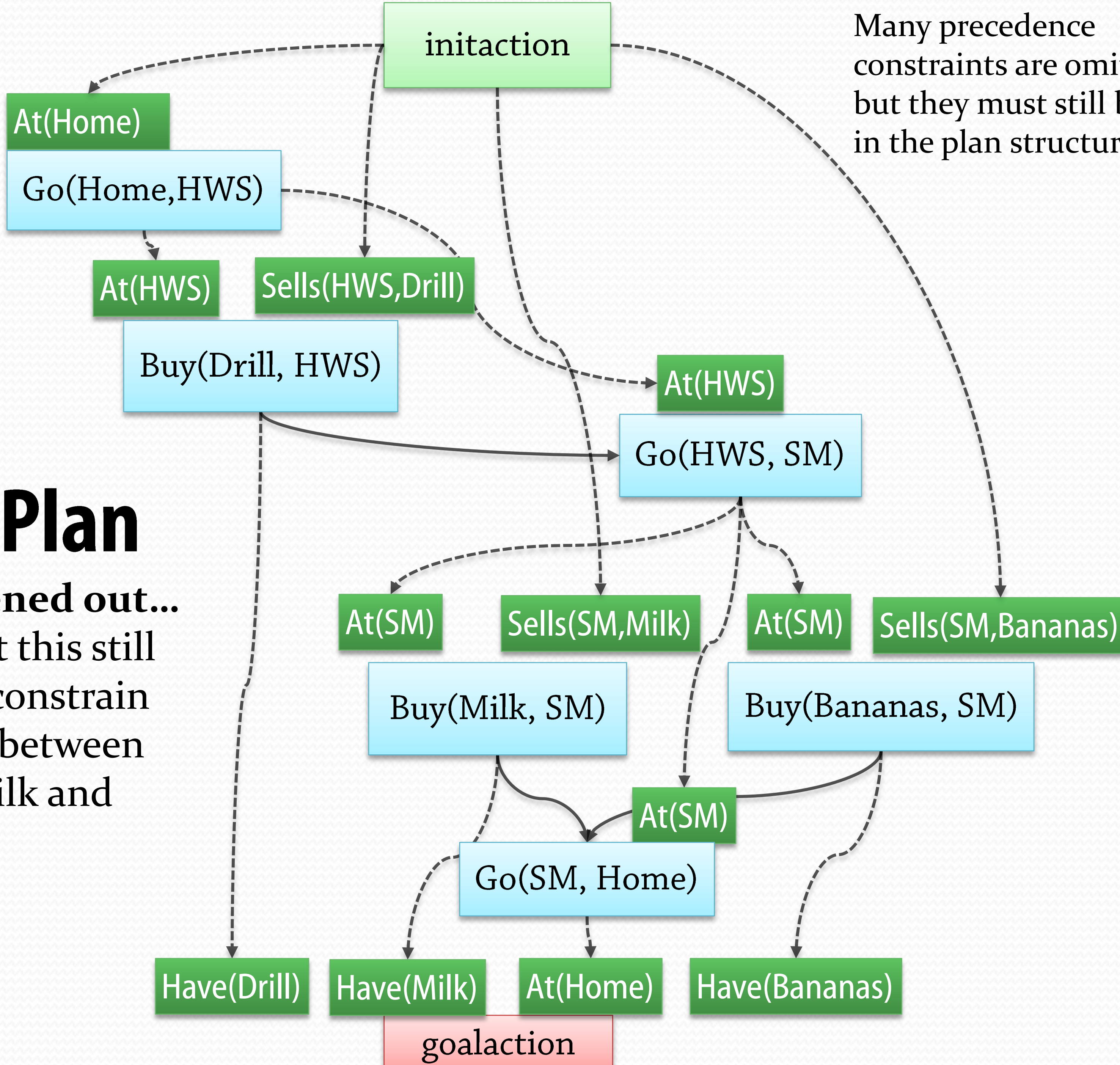
- Establish At(l_3) with l_3=SM



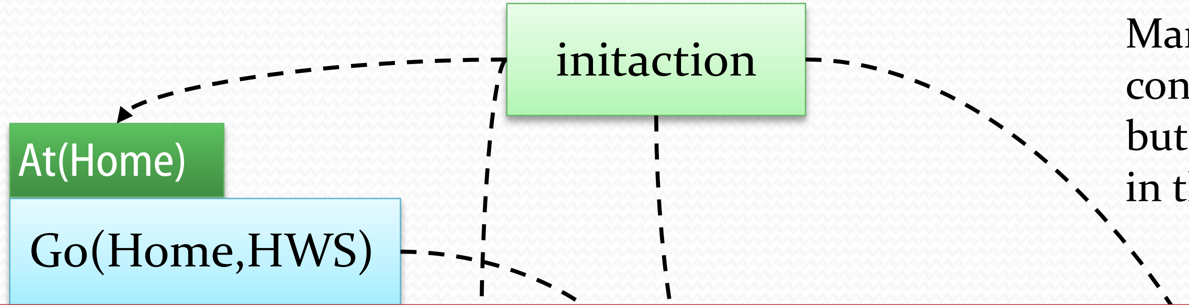
Final Plan

Straightened out…
(Note that this still does not constrain the order between buying milk and bananas)

Many precedence constraints are omitted – but they must still be there in the plan structure!



Many precedence constraints are omitted – but they must still be there in the plan structure!



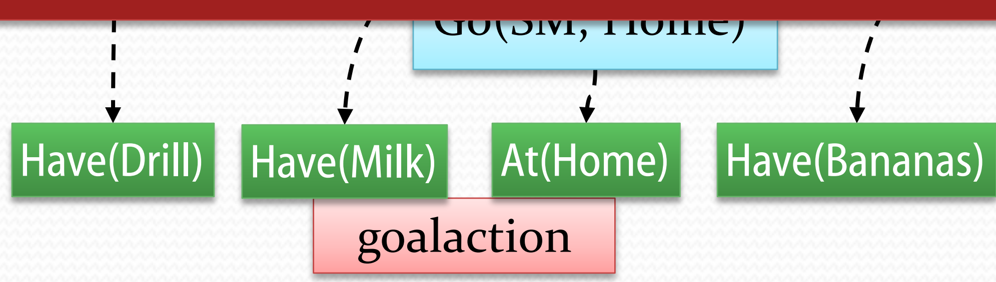
This sequence assumed optimal choices!

<u>Heuristics do exist…</u>

Simple example:

Preferring flaws with **<u>few resolvers</u>** keeps the branching factor down

Still, planners try **<u>many</u>** other alternatives, dead ends, etc…



Conclusions

- Partial-order planning **delays commitment** to action ordering
  - Lower branching factor
  - More efficient in some situations
- Many POP planners still **assume sequential execution**
  - The intention was to find plans quickly,
    not to find partially constrained plans
- Forward-chaining planners **currently** have the advantage
  - Due to strong domain-dependent heuristics