Course "Automated Planning: Theory and Practice"
Chapter 06: The Partial Order Causal Link Search Space

Teacher: Marco Roveri - marco.roveri@unitn.it

M.S. Course: Artificial Intelligence Systems (LM)
A.A.: 2025-2026
Where: DISI, University of Trento
URL: https://shorturl.at/A81hf

Last updated: Wednesday 8th October, 2025

Terms of Use and Copyright

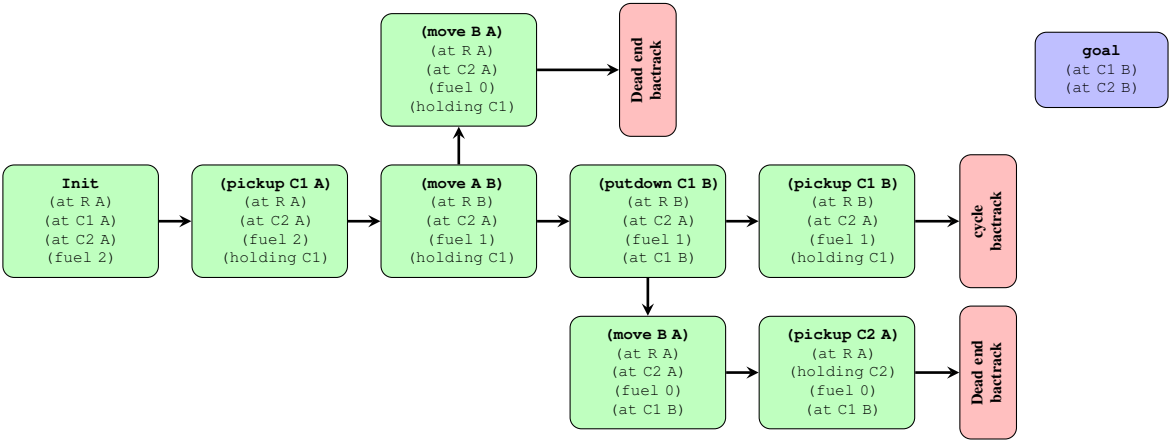
The material (text, figures) in these slides is authored by Jonas Kvarnström and Marco Roveri.

Motivating Problem

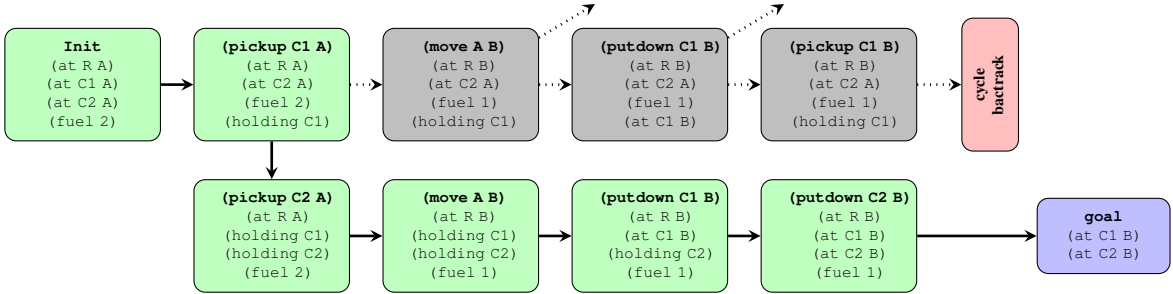
- Let's consider a simple planning problem:
  - Two crates `C1`, `C2`, two positions `A`, `B`, and one robot `R`
  - The robot:
    - can `carry` up to two crates
    - can `move` between locations, consuming one unit of fuel
  - Initially crates and robot are at `A`, and the robot has 2 unit of fuel
  - Both crates shall be moved to `B`

Let's see what forward-chaining planning might do (depending on heuristics)...

Motivating Problem: Forward Search



Motivating Problem: Forward Search



Motivating Problem: Observations

- Most of the actions added before backtracking were useful and necessary

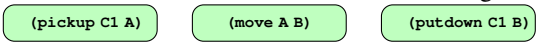
- At first, we added them in the wrong order

- Forward and backward planning commit immediately to action order!
  - Puts each action in its final place in the plan!
- State space heuristics must be "smart enough" to tell us:
  - Which actions are useful
  - When to add them in the plan

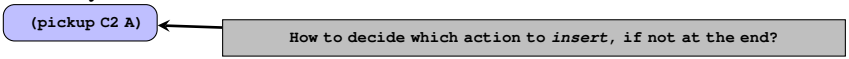
What if we could "rearrange" actions?

First step: Insertion

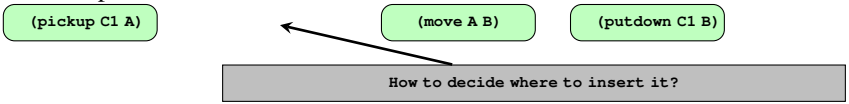
- Sequences with arbitrary insertion: Useful?
  - Most of the actions added before backtracking were useful and necessary



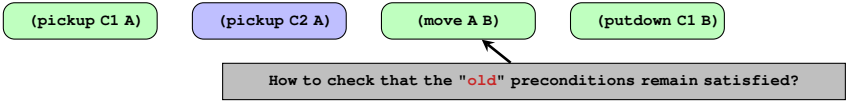
  - Realize you need another one...



  - Make a space...

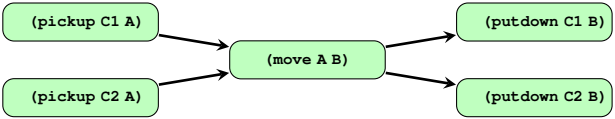


  - ... and place the action there



Second Step: Partial Order

- If we must deal with this complexity:
  - We can "get more from the same price"
- Let's skip sequences completely - a plan could be partially ordered

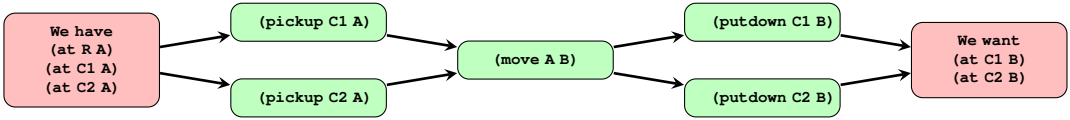


  - A set of actions $A = \{a_1, a_2, a_3, ...\}$
  - A set of precedence constraints $\{a_1 < a_2, a_1 < a_3, ...\}$
    - a_1 must finish before a_2 starts, a_1 must finish before a_3 starts
    - We represent them graphically with solid arrows

How do we generate such plans?

POCL: Introduction

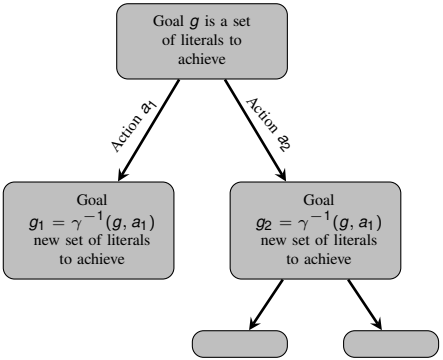
- Partial Order Causal Link (POCL) planning
  - Use a partial order as described before
    - Not when executing the plan
    - Only to delay commitment to action ordering
  - As in backward search:
    - Add useful actions to achieve necessary conditions
    - Keep track of what remains to be achieved
    - Insert actions "at any point" in the plan



More sophisticated "bookkeeping" required!

POCL: Comparison to Backward Search

- Search tree for backward search (as seen earlier)



The goal is a set of literals – simple!

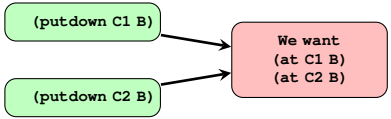
Every step takes you to a new set of literals to achieve

From a search node, you know how to reach the goal using a sequence of actions!

A search node [2] can simply be a goal set!

POCL: Comparison with Backward Search

- In POCL planning there is no sequence – and no clear "before" relation!



The goal is a set of literals – simple!

But no set of literals can describe what must be true before e.g. `(putdown C1, B)` ...

... because we could add a new action "in parallel" ...

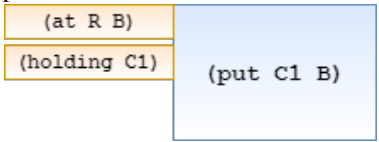
... or even between `(putdown C1 B)` and the goal!

- There are consequences for the POCL plan structure and the node plan structure...

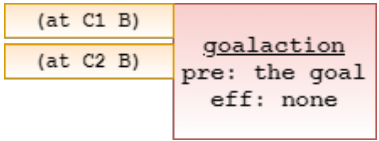
POCL: Conditions, Goal Action

- ... must keep track of individual propositions to be achieved
  - Throughout the plan – not a single state $g_1 = \gamma^{-1}(g, a_1)$
  - May come from preconditions of every action in the plan

Notation chosen:
Preconditions on the left/top side



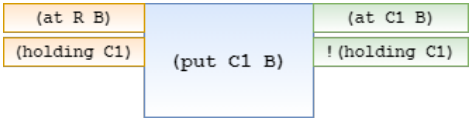
  - May come from problem goal as in backward search
    - Trick: Use a uniform representation
    - Add a "fake" goal action to every plan with the goals as preconditions!



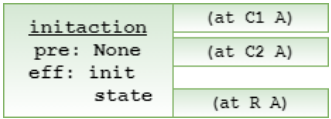
POCL: Effects, Initial Action

- Must keep track of individual propositions that are achieved
  - Throughout the plan - not from a single *relevant* action
  - May come from effects of every action in the plan

Notation chosen:
Effects on the right/bottom side

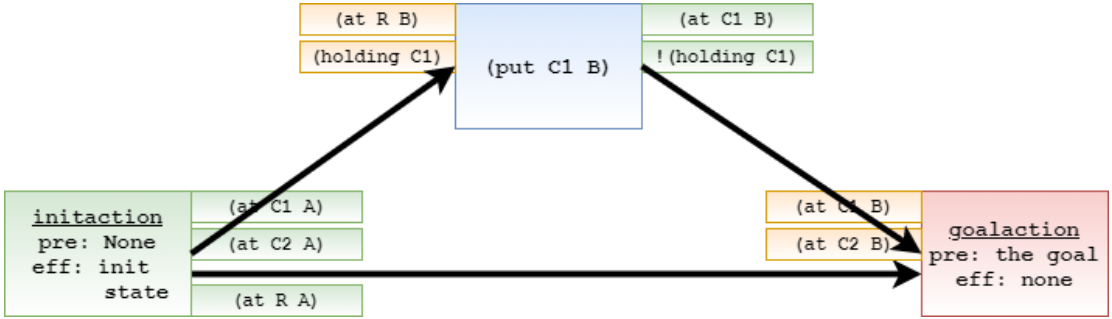


  - May come from initial state
    - Trick: Use a uniform representation
    - Add a "fake" initial action with the initial state as effect



POCL: Precedence Constraints

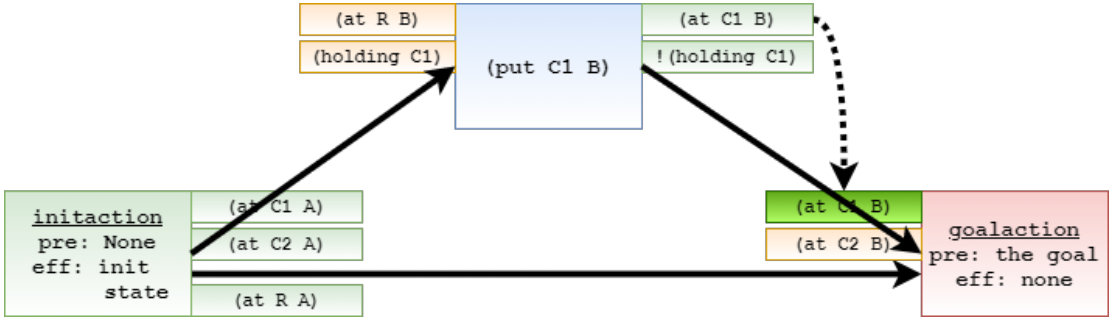
- Plan structure so far



POCL: Causal Links

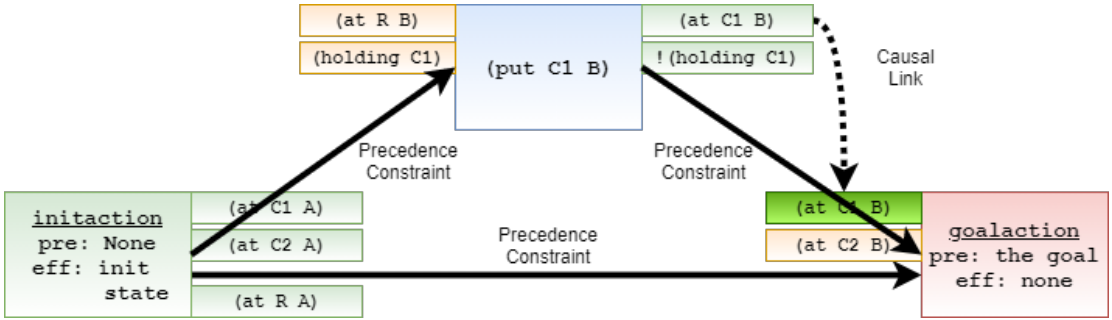
- Let's keep track of which actions achieves which precondition: Causal links

Causal links (dashed):
`(at C1 B)` must remain true between end of `(put C1 B)` and the beginning of `goalaction`.
No one must delete it!
Important for *threat management* (later)



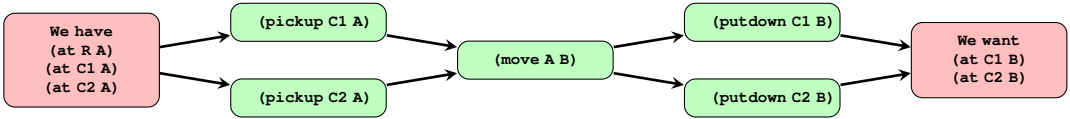
POCL: Partial-Order Plans

- A ground partial-order plan consists of:
  - A set of actions
  - A set of precedence constraints $a \rightarrow b$ (a must precede b)
  - A set of causal links $a \xrightarrow{p} b$ - action a establishes the preconditions p needed by b



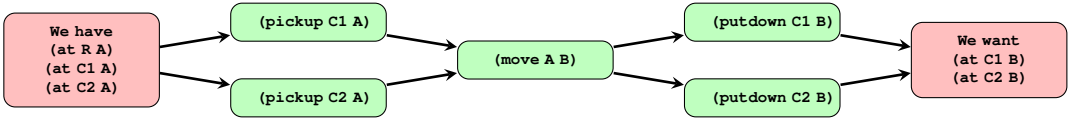
Partial-Order Solutions

- Original motivation: performance
  - A partial-order plan is a solution iff all sequential plans satisfying the ordering are solutions
  - A partial-order plan is a executable iff all sequential plans satisfying the ordering are solutions
    - `(pickup C1 A); (pickup C2 A); (move A B); (putdown C1 B); (putdown C2 B)`
    - `(pickup C2 A); (pickup C1 A); (move A B); (putdown C1 B); (putdown C2 B)`
    - `(pickup C1 A); (pickup C2 A); (move A B); (putdown C2 B); (putdown C1 B)`
    - `(pickup C2 A); (pickup C1 A); (move A B); (putdown C2 B); (putdown C1 B)`



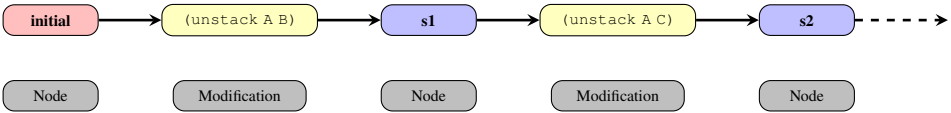
Partial-Order Solutions

- Can be extended to allow concurrent action execution
  - Requires a new formal model!
    - The so far considered transition model *does not define* what happens if `C1` and `C2` are picked up simultaneously

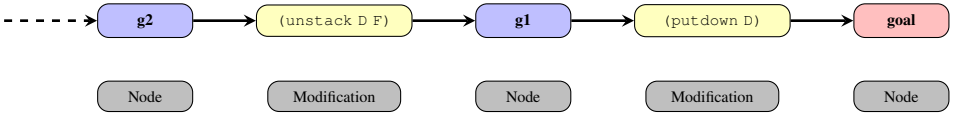


Context: Forward, Backward

- Forward Search: a search node is a "current state"

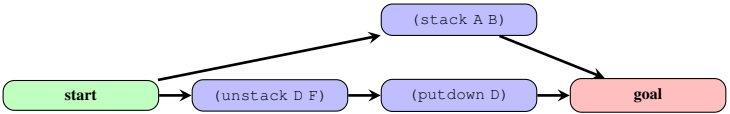


- Backward Search: a search node is a "current goal"



No current state during search!

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

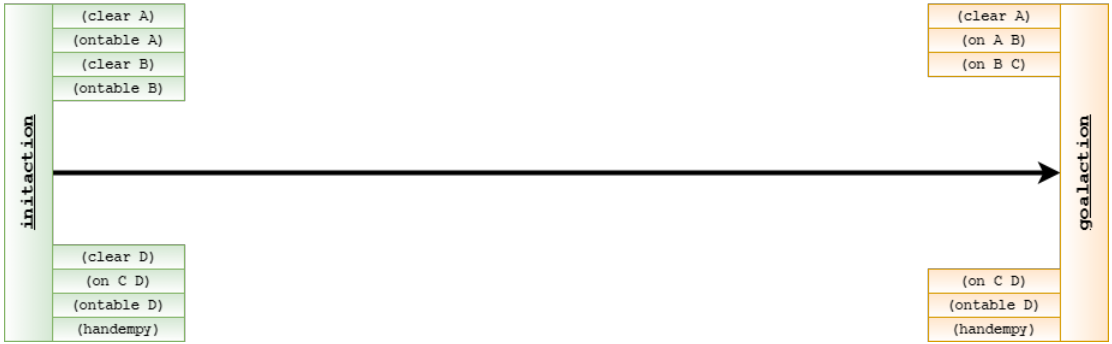


A search node cannot correspond to a state or to a goal!

Search Nodes are Partial Order Plans

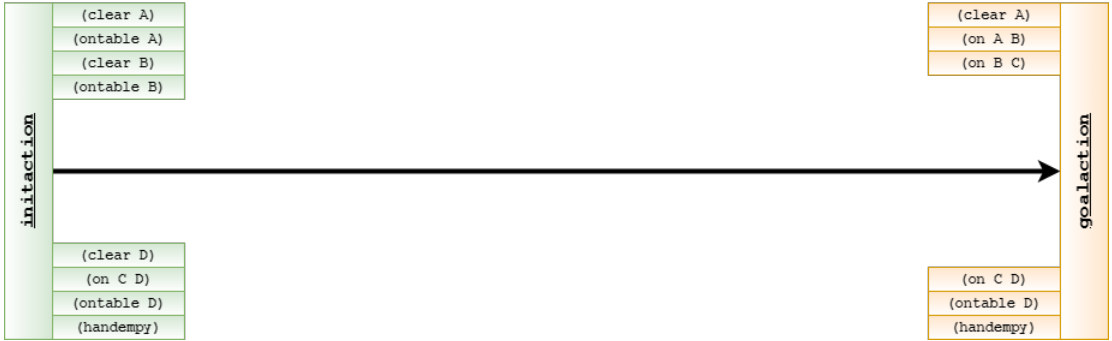
- [1] Each node must contain *more information*: the entire plan!
  - [2] The initial search node contains the initial plan
    - The special initial action and goal action
    - A single precedence constraint

This is one form of "plan-space" planning!



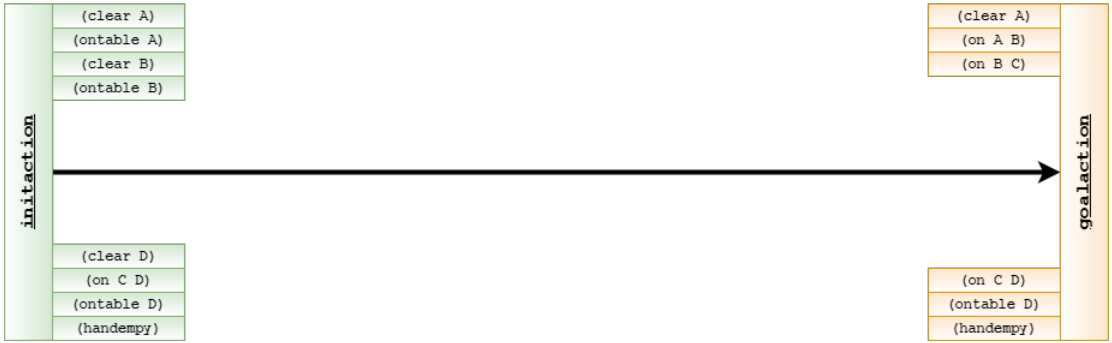
Branching rule

- [3] We need a branching rule!
  - Forward planning: One successor per action applicable is s
  - Backward planning: One successor per action relevant to g
  - POCL: ???



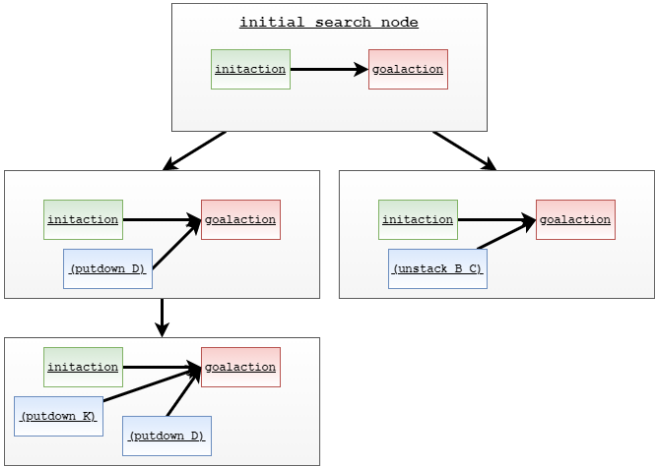
[3] Branching rule

- Identify specific reasons for modifying the plan, called flaws (i.e. todos)
  - 1) Open goal: we have not decided *how* to achieve a precondition (E.g. `(clear A)`)
  - 2) Threat: An action may *interfere* with another
- One successor for each different way of repairing a flaw



Search Space

- [6] Use search strategies, backtracking, heuristics, ... to search this space!



[4] Solution iff there are no flaws
(We will see later how to do it)

[5] Plan extraction: pick any sequential order consistent with the precedence constraints

```
(putdown D); (putdown K)
(putdown K); (putdown D)
```

Flaws

- Flaw, *noun*:
  - a feature that mars the perfection of something; defect; fault: beauty without flaw; the flaws in our plan.
  - a defect impairing legal soundness or validity.
  - a crack, break, breach, or rent.
- Flaw, in POCL planning:
  - Something we need to take care of to complete the plan
  - Technical definition: An *open goal* or a *threat*

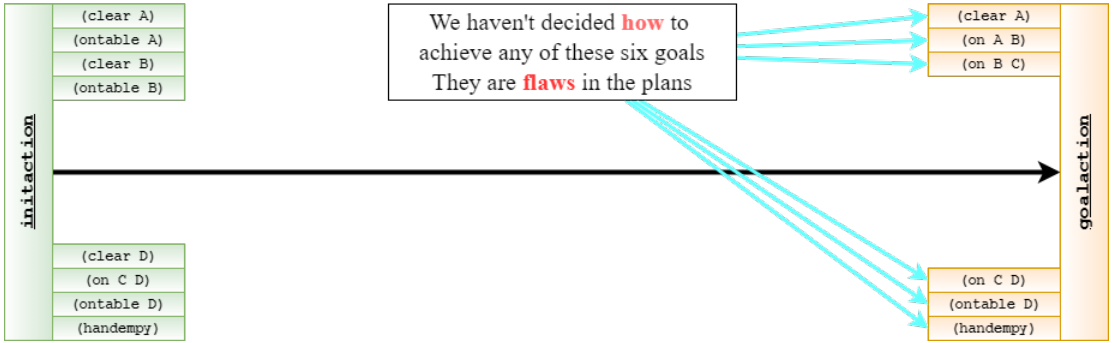
- Not:
  - Something that has "gone wrong"!
  - A problem during planning
  - A mistake in the final solution
  - ...

Flaw types

- Open Goals
- Threats

Flaw Type 1: Open Goals

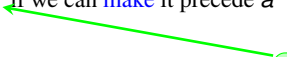
- An action *a* has precondition *p* with no incoming causal link



`(clear A)` is already true in s0, but there is no causal link...
Adding one causal link from s0 means `(clear A)` must never be deleted!
We need other alternatives: delete `(clear A)`, then re-achieve it for `goalaction`..

Flaw Type 1: Open Goals

- To resolve an open goal
  - Find an action b that cause 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

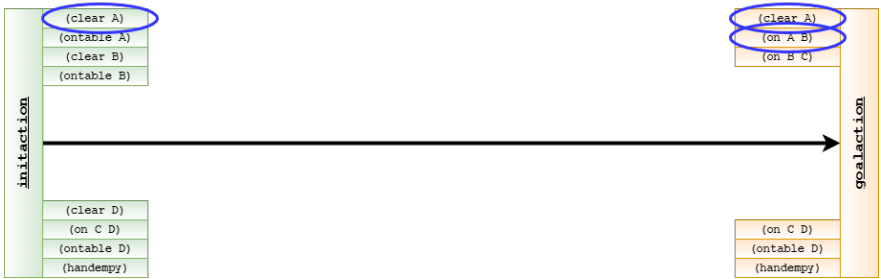


Partial order! This was not possible in backward search...

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

- We can chose to find support for `(clear A)` — 8 successors!
  - 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)`
- We can chose to find support for `(on A B)` — +1 successor
  - Only from a new instance of `(stack A B)`
- ...

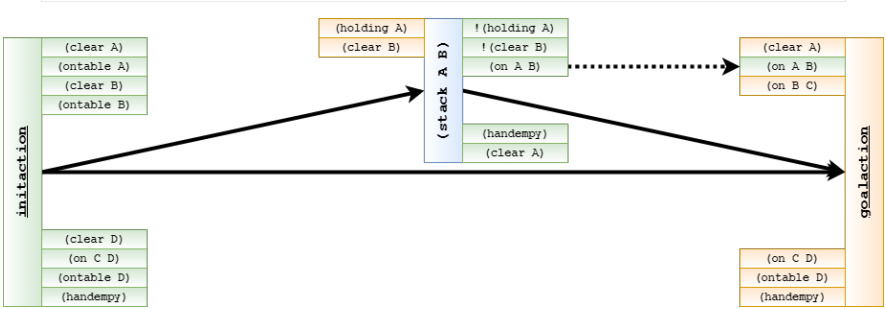


Resolving Open Goals (cont.)

- Suppose we add `(stack A B)` to achieve `(on A B)`
  - Must add a causal link for `(on A B)` (dashed lines)

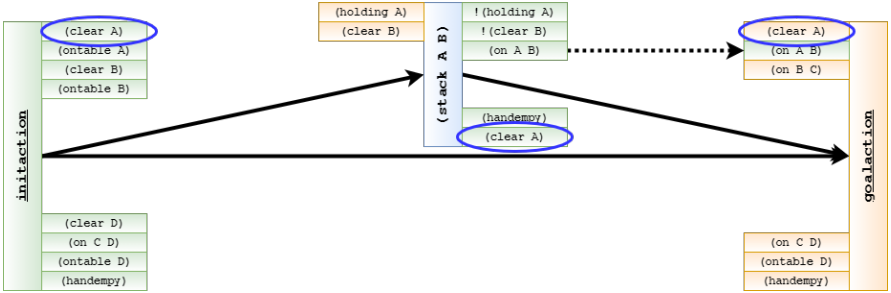
This instance of `(stack A B)` is responsible for achieving `(on A B)` for the `goalaction`

  - Must also add precedence constraints
  - Looks totally ordered: we actually have only one "real" action!



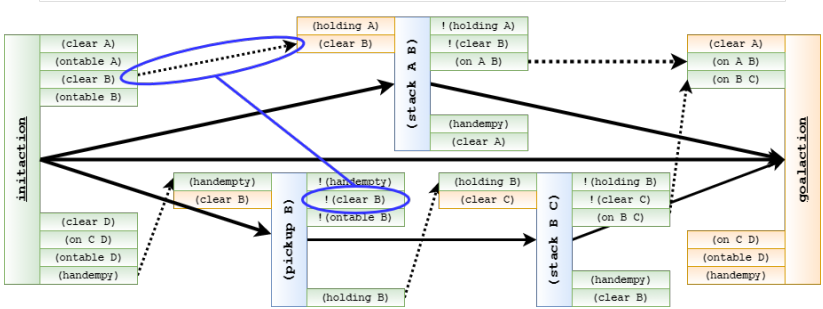
Resolving Open Goals (cont.)

- Now we have 7 open goals
  - We can choose to find support for `(clear A)`:
    - From `initaction`; from the instance of `(stack A B)` 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)`, `(unstack D A)`
  - ...



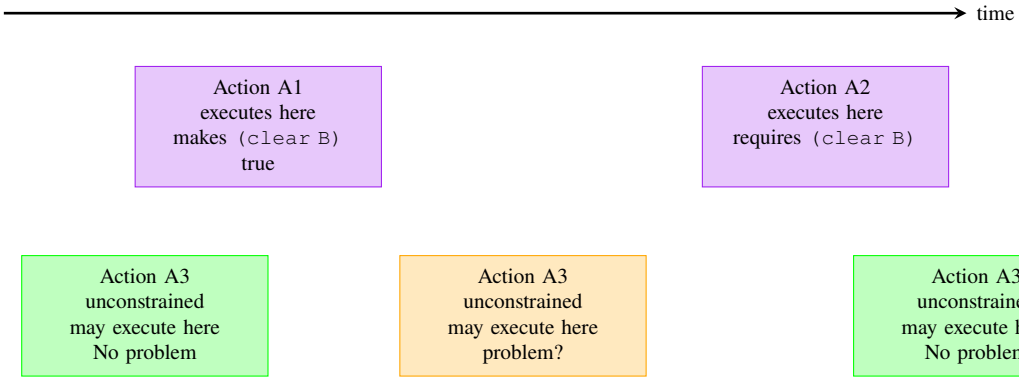
Flaw Type 2: Threats

- A threat against a causal link
  - `initaction` should achieve `(clear B)` for `(stack A B)` - there is a causal link
  - `(pickup B)` deletes `(clear B)`, and may occur between `initaction` and `(stack A B)`
  - We cannot be certain that `(clear B)` still holds when `(stack A B)` starts!



Flaw Type 2: Threats

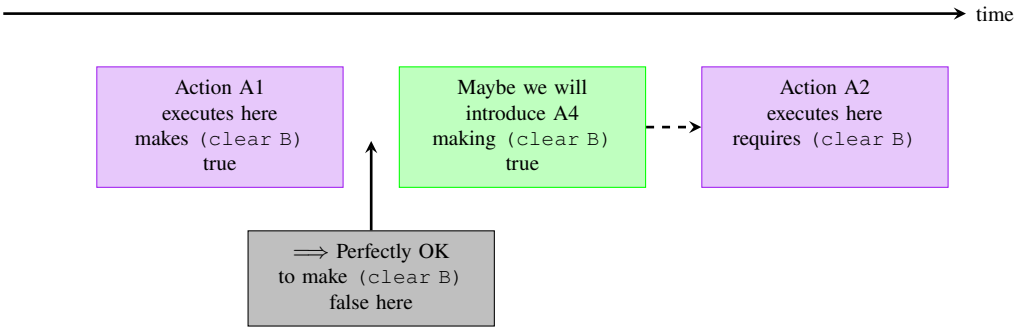
- A threat against a causal link



No! there is no causal link, so no reason to assume `(clear B)` must be preserved from `A1` to `A2`

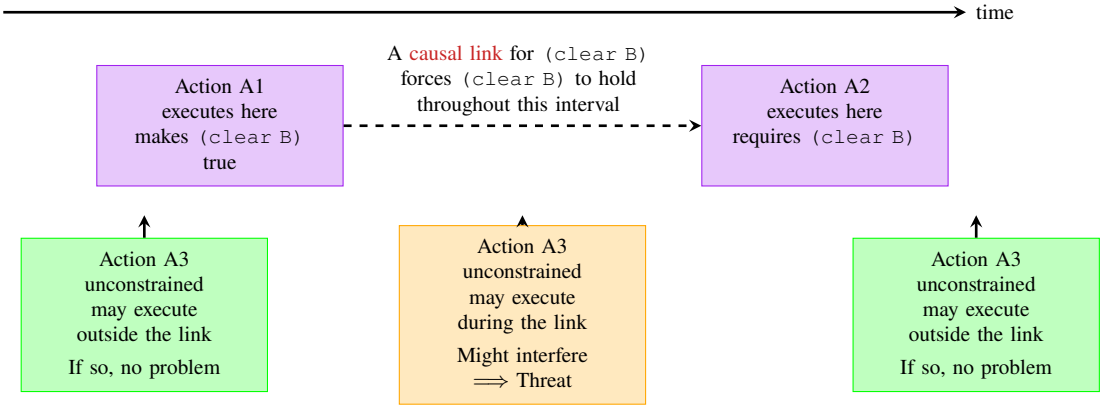
Flaw Type 2: Threats

- Why no threats without causal links?



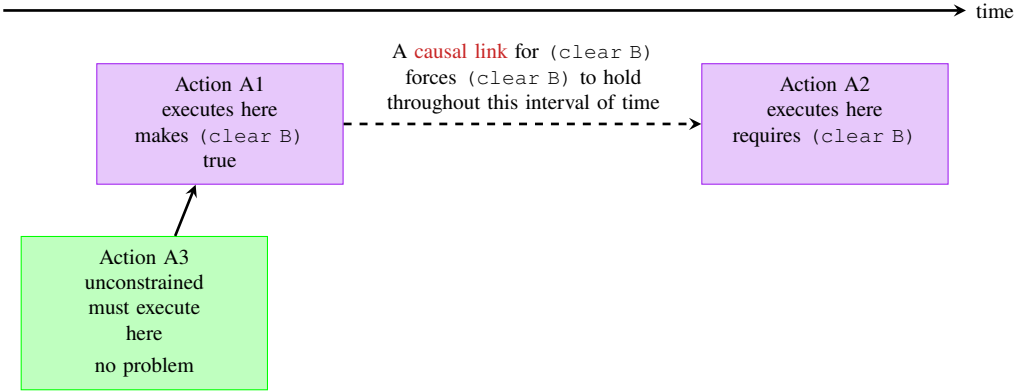
Flaw Type 2: Threats

- But when we have a causal link:

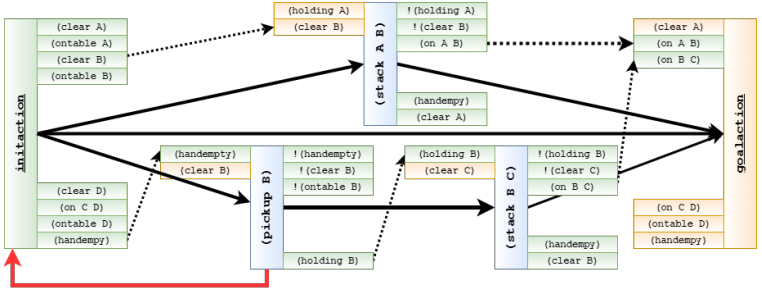


Resolving Threats: Rule 1

- The action that disturbs the causal link is placed before the action that support/achieves the precondition
  - Only possible if the resulting partial order is consistent (acyclic)!



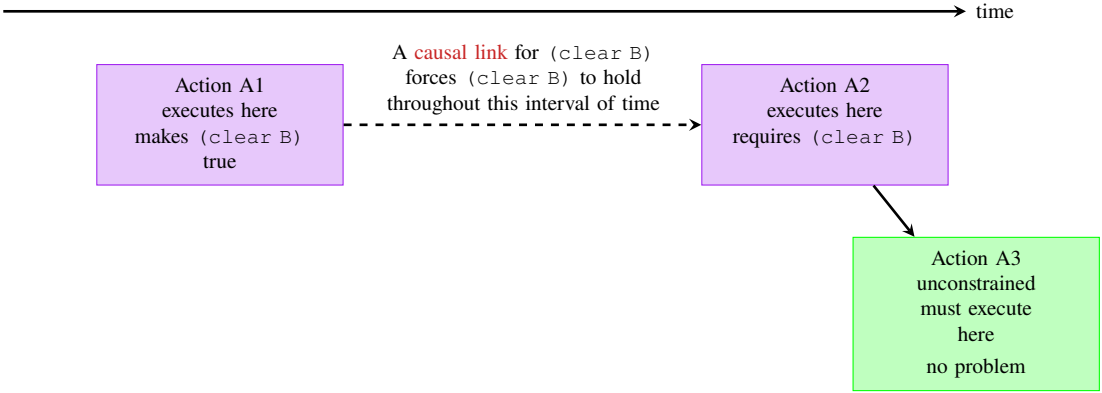
Resolving Threats



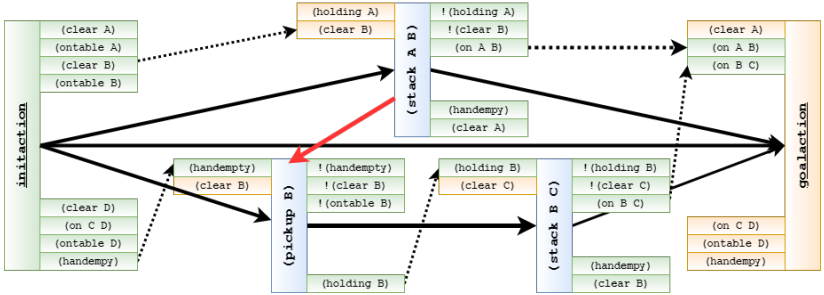
In this case not consistent! (a cycle is created)

Resolving Threats: Rule 2

- The action that disturbs the causal link is placed after the action that requires the precondition
  - Only possible if the the resulting partial order is consistent (acyclic)!

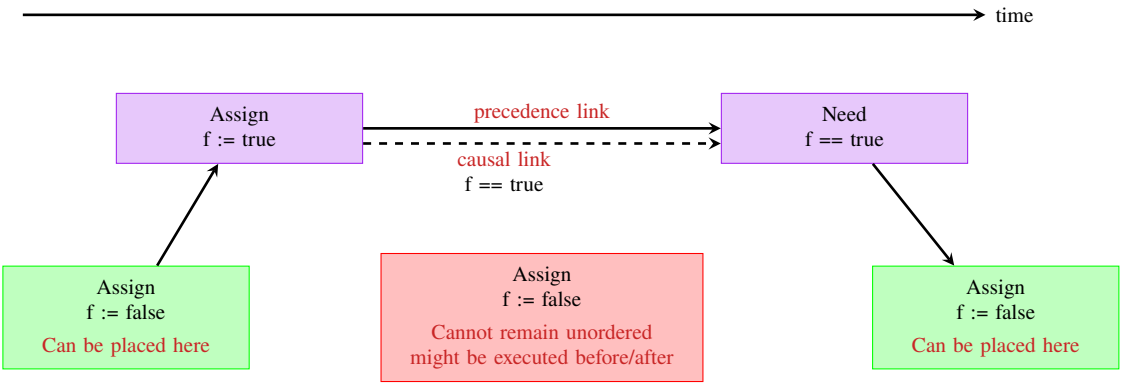


Resolving Threats

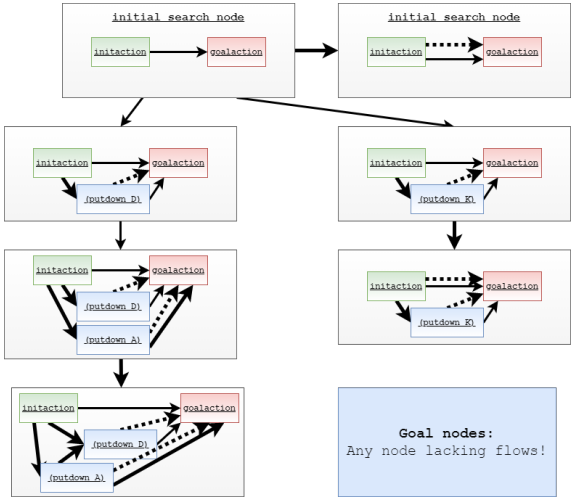


In this case it works! (There are no cycles)

Resolving Threats: Summary



POCL Search Space



Planning as Search

```
function SEARCH(problem)                                          → [2]
    initial-node ← MAKE-INITIAL-NODE(problem)
    open ← {initial-node}
    while (open ≠ ∅) do
        node ← SEARCH-STRATEGY-REMOVE-FROM(open)                  → [6]
        if IS-SOLUTION(node) then                                 → [4]
            return EXTRACT-PLAN-FROM(node)                        → [5]
        end if
        for each newnode ∈ SUCCESSORS(node) do                    → [3] All ways of resolving some flaw
            open ← open ∪ {newnode}
        end for
    end while
    return Failure                                                → Expanded the entire search space without finding a solution
end function
```

POCL Planning: possible formulation (sound/complete)

```
function SEARCH(problem)
    initial-node ← MAKE-INITIAL-NODE(problem.init,problem.goal)        → [2]
    open ← {initial-node}
    while (open ≠ ∅) do
        π ← SEARCH-STRATEGY-REMOVE-FROM(open)                          → [6]
        flaws ← OPENGOALS(π) ∪ THREATS(π)
        if flaws = ∅ then                                              → Can prove: π is a solution if there are no remaining flaws
            return π                                                   → [5] Returns a partially ordered solution plan. Any total ordering is a plan achieving the goal!
        end if
        select any flaw φ ∈ flaws                                      → One flaw chosen!
        resolvers ← FINDRSOLVERS(φ,π)                                  → May be the empty set
        for each r ∈ resolvers do
            π' ← REFINE(r, π)                                          → Actually apply the resolver
            open ← open ∪ {π'}                                         → But all resolvers must be tested...
        end for
    end while
    return Failure                                                     → Expanded the entire search space without finding a solution
end function
```

POCL: Successors

We said: "Every flaw leads to successors"

- It is sufficient to try one (any) flaw to resolve!
- Testing other flaw will be redundant!
  - Every flaw has to be resolved
  - Choosing the flaw *later* cannot help us resolve it: all possibilities already exists
  - Chosing the flaw *later* cannot help us resolve some other flaw

We must "test" different resolvers

- Choosing one resolver can prevent other problems resolutions
- Open goal: Use action A or action B?
- Threat: Which order to choose?

select any flaw $\varphi \in$ flaws
resolvers $\leftarrow$ FindRsolvers(φ,π)

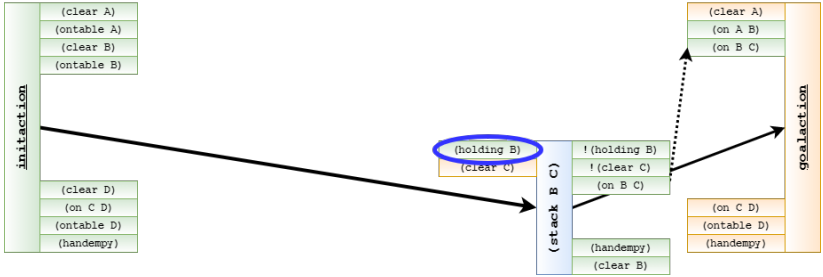
Enables the use of heuristics to select flaws and to prioritize open nodes

for each r $\in$ resolvers **do**
  $\pi' \leftarrow$ Refine(r, π)
  open $\leftarrow$ open $\cup$ $\{\pi'\}$
end for

Partial Instantiation

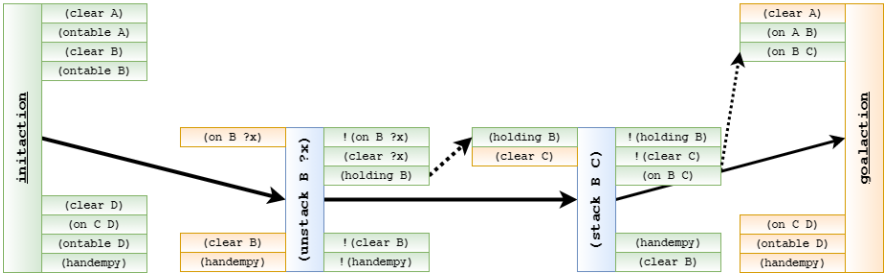
Suppose in the example we want to achieve `(holding B)`

- Ground search generates many alternatives
  - Add `(unstack B A)`, `(unstack B F)`, `(unstack B G)`, ...
  - Add `(pickup B)`
- Lifted search generates two partially instantiated alternatives
  - Add `(unstack B ?x)`
  - Add `(pickup B)`



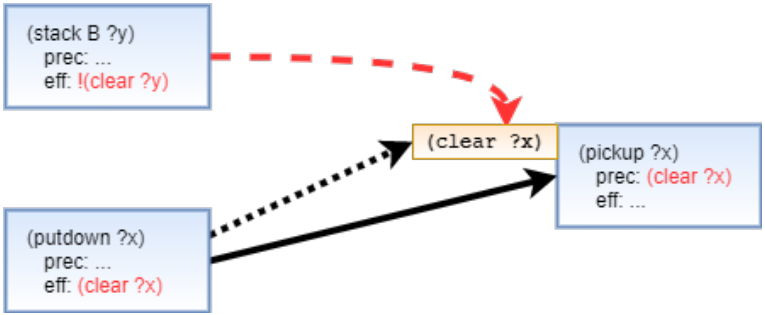
Lifted Partial-Order Plans

- A set of possibly unground actions
- A set of precedence constraints: a must precede b
- A set of causal links: a establishes precondition p needed by b
- A set of binding constraints
  - equality constraints: $v_1 = v_2$ or $v_1 = c$
  - inequality constraints: $v_1 \neq v_2$ or $v_1 \neq c$



Resolving Threats: alternative approach

- For partially uninstantiated actions, we may find potential threats
  - `(stack B ?y)` may threaten the causal link, but only if $?x = ?y$
  - Can be resolved adding a constraint: $?x \neq ?y$

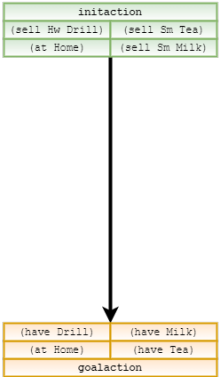


Example

- Taken and adapted from Russell and Norvig [5]
  - **Operator**: `(go ?from ?to)`
    pre: `(at ?from)`
    eff: `(at ?to)`, ¬ `(at ?from)`
  - **Operator**: `(buy ?product ?store)`
    pre: `(at ?store)`, `(sell ?store ?product)`
    eff: `(have ?product)`
  - **Initial state**
    `(at Home), (sell Hws Drill), (sell Sm Milk), (sell Sm Tea)`
  - **Goal**
    `(at Home), (have Drill), (have Milk), (have Tea)`

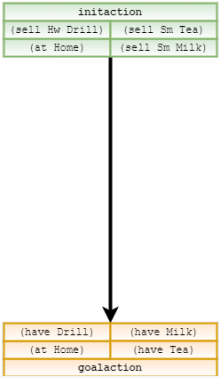
Example (1)

- Initial plan: `initaction`, `goalaction`, and a precedence constraint



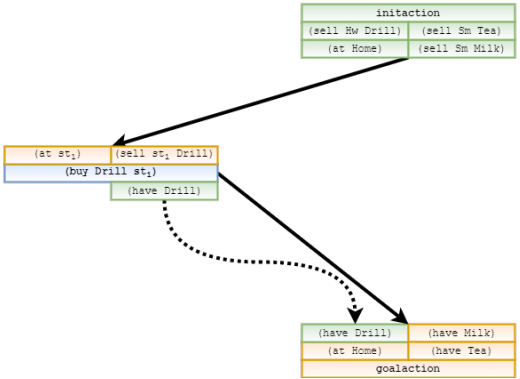
Example (2)

- Four flaws exists: open goals
  - The heuristics suggests to resolve `(have Drill)` first



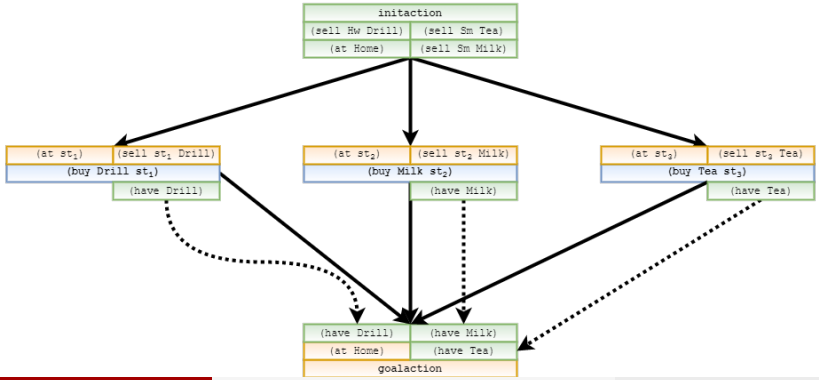
Example (3)

- No action in plan achieves `(have Drill)`, but `(buy ?p ?s)` achieves `(have ?p)`
  - Partially instantiate `(buy Drill ?s)` (ignoring where to buy)



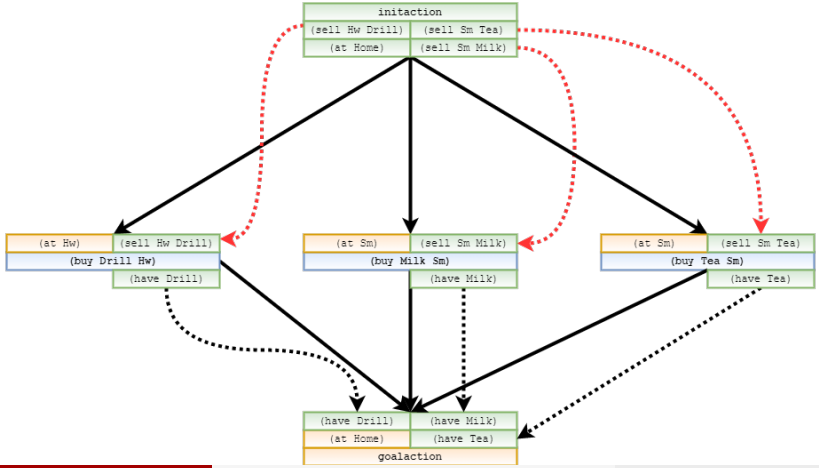
Example (4)

- First three refinements: the possible ways to achieve `(have ?p)` preconditions
  - We do not care the order in which to buy things!



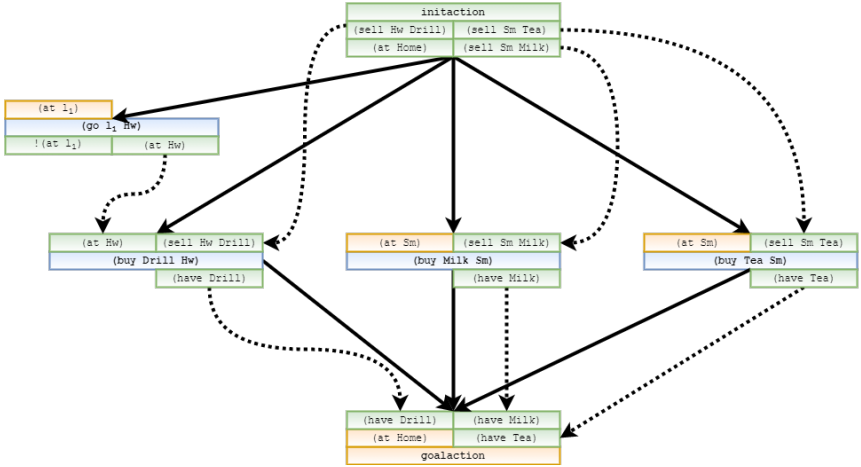
Example (5)

- Three more refinements: no action causes `(sell ?p)` - except `initaction`
  - $\Longrightarrow$ use it for support



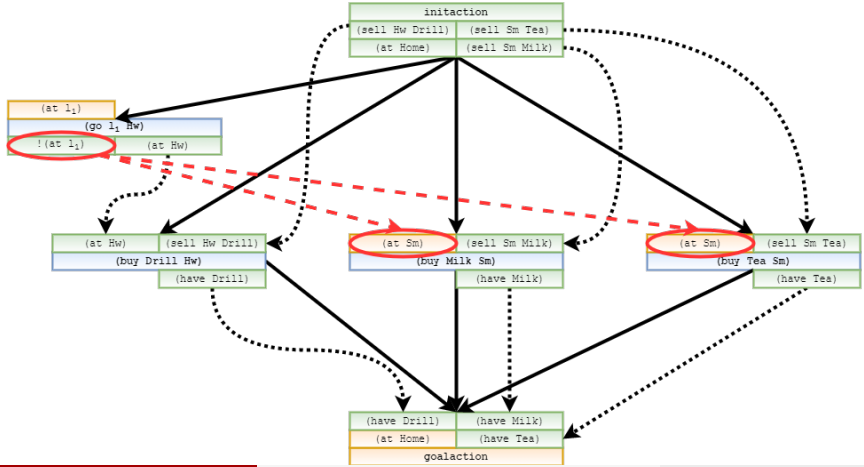
Example (6)

- To establish `(at Hws)`: must go there from somewhere



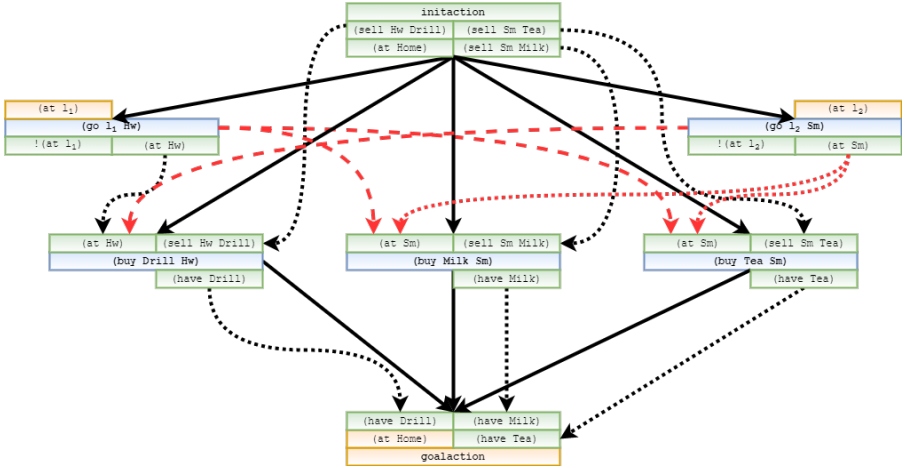
Example (7)

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



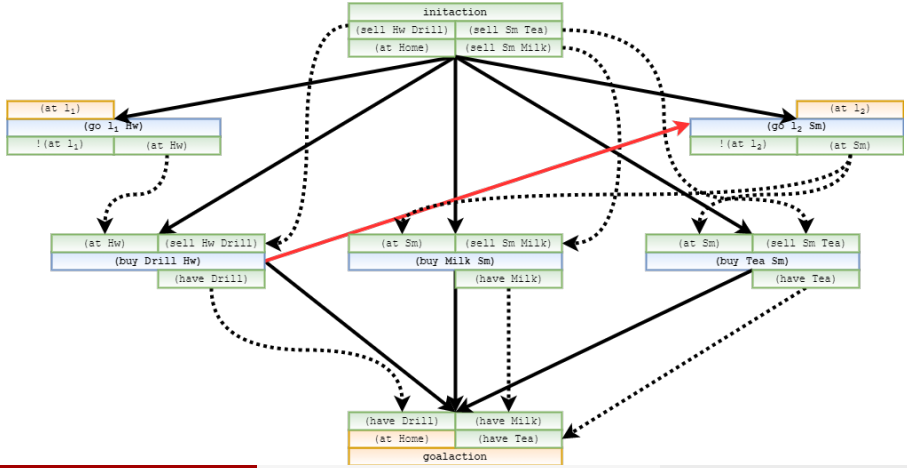
Example (8)

- To establish `(at Sm)`: must go there from somewhere $\Longrightarrow$ mutual threats...
- Let's use same action for both `(at Sm)` $\Longrightarrow$ even more threats – deal with them now or wait!



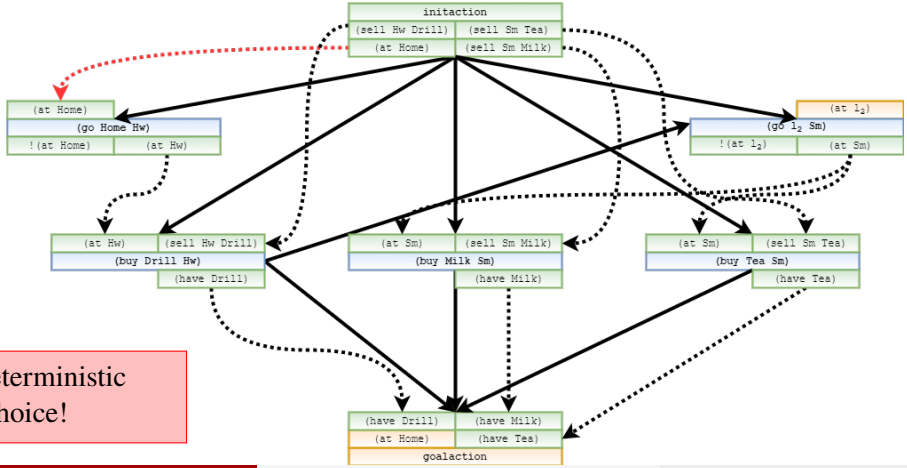
Example (9)

- How to resolve the threat to `(at Hw)`?: Make `(buy Drill)` precede the `(go` l_2 `Sm)`
  - Also happens to resolve the other two threats!



Example (10)

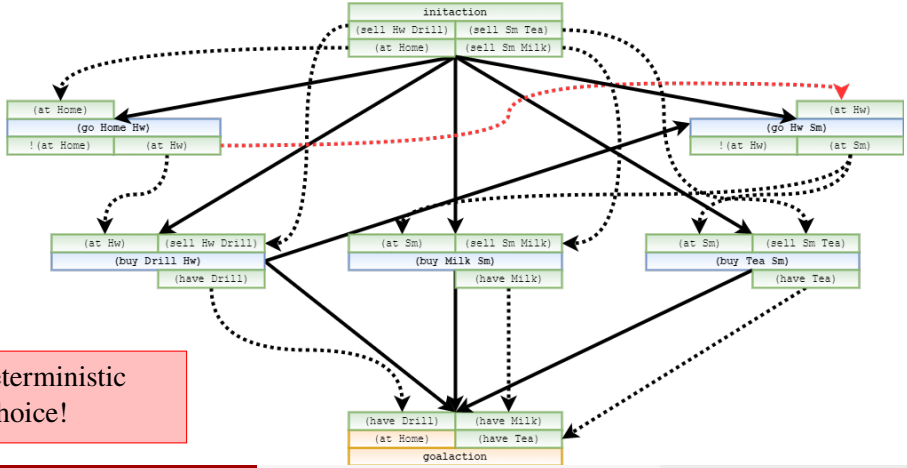
- How to establish `(at` l_1`)`?
  - We do it from `initaction` forcing l_1 = `Home`



Nondeterministic choice!

Example (11)

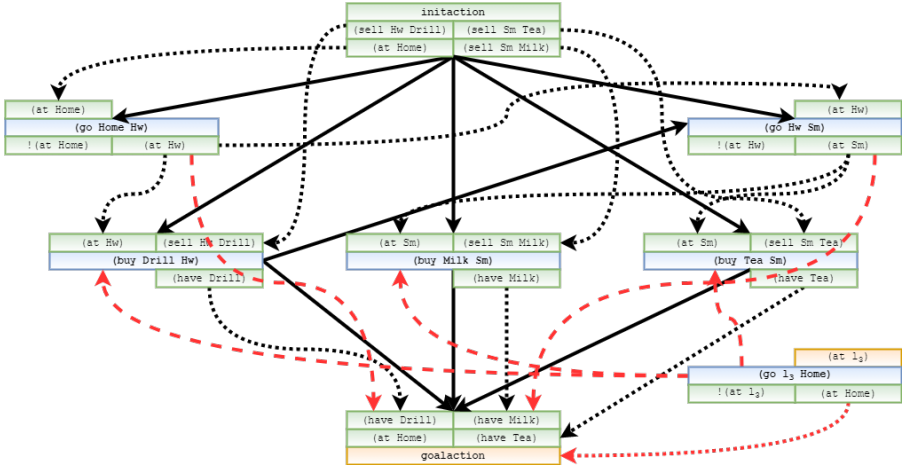
- How to establish `(at` l_2`)`?
  - We do it from `(go Home Hw)` forcing $l_2 =$ `Hw`



Nondeterministic choice!

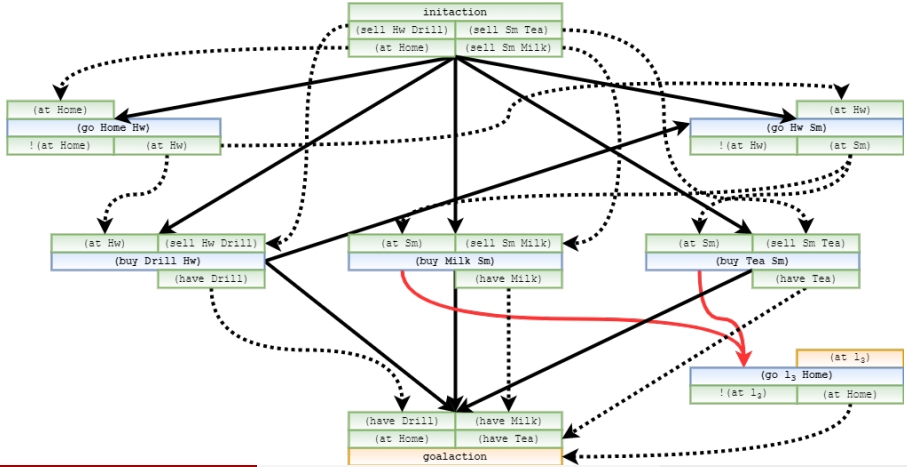
Example (12)

- The only possible way to establish `(at Home)` for `goalaction`
  - Creates several threats



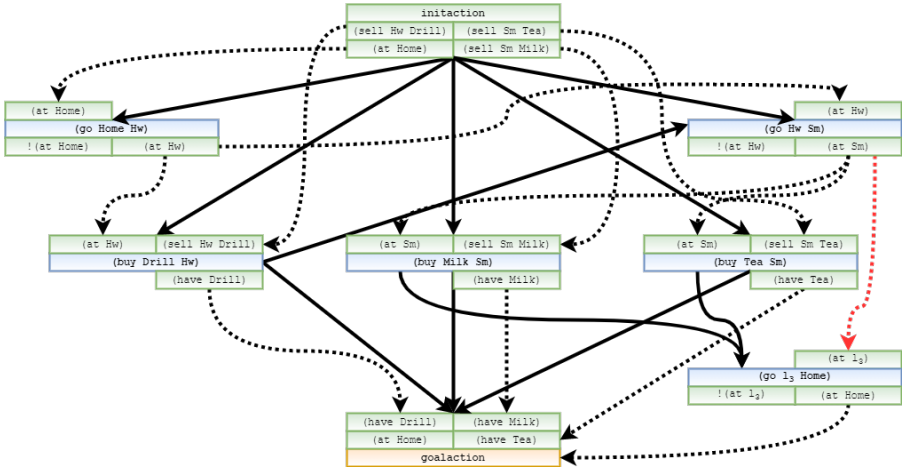
Example (13)

- To remove threats to `(at Sm)` and `(at Hw)`
  - Make `(go Hw Sm)` and `(go Home Hw)` precede `(go` l_3 `Home)`: removes other threats

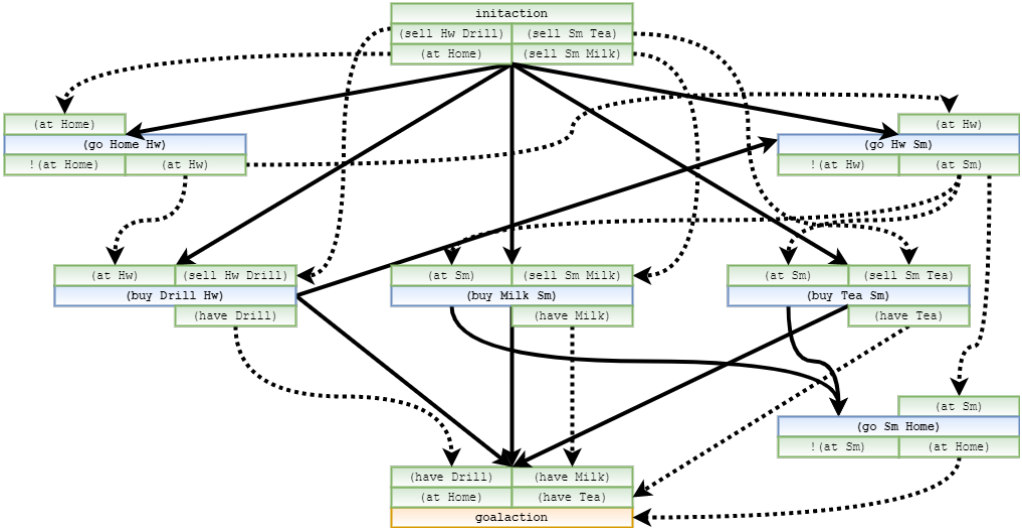


Example (14)

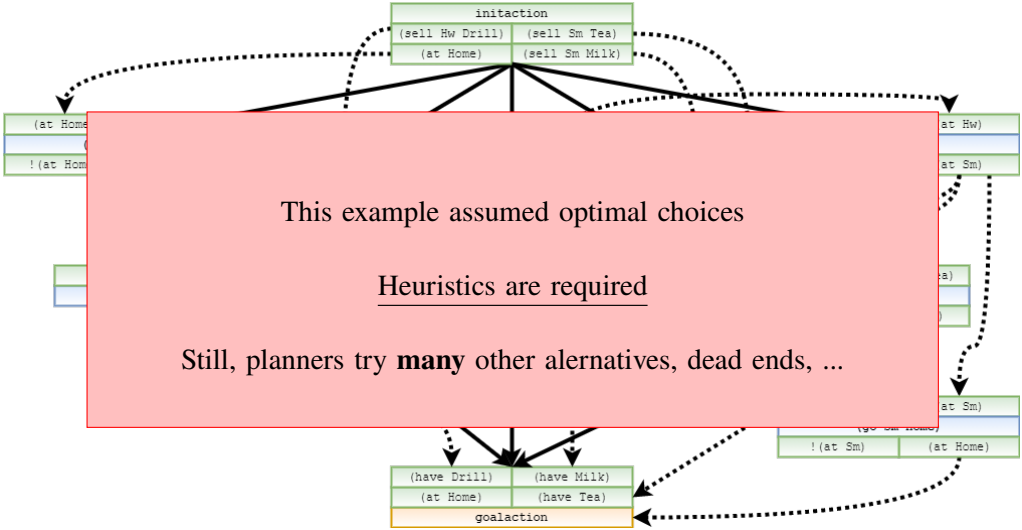
- Establish (at l_3) with $l_3 = \texttt{Sm}$



Example (15): Final plan



Example (16): Final plan



This example assumed optimal choices

Heuristics are required

Still, planners try **many** other alernatives, dead ends, ...

References I

[1] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013. ISBN 9781608459698. doi: 10.2200/S00513ED1V01Y201306AIM022. URL https://doi.org/10.2200/S00513ED1V01Y201306AIM022.

[2] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004. ISBN 978-1-55860-856-6.

[3] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016. ISBN 978-1-107-03727-4. URL http://www.cambridge.org/de/academic/subjects/computer-science/artificial-intelligence-and-natural-language-processing/automated-planning-and-acting?format=HB.

[4] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2019. doi: 10.2200/S00900ED2V01Y201902AIM042. URL https://doi.org/10.2200/S00900ED2V01Y201902AIM042.

[5] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020. ISBN 9780134610993. URL http://aima.cs.berkeley.edu/. 48