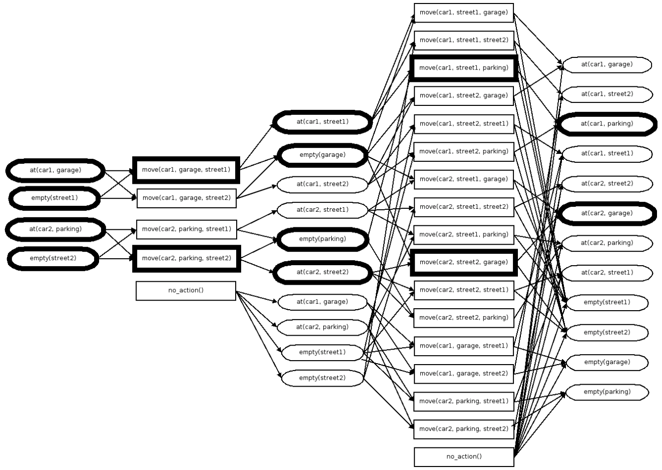
Plan-Space Planning

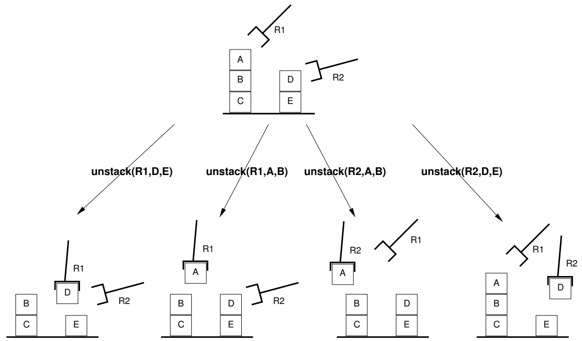
Chapter 10

Different Plans, Search Spaces, and Approaches

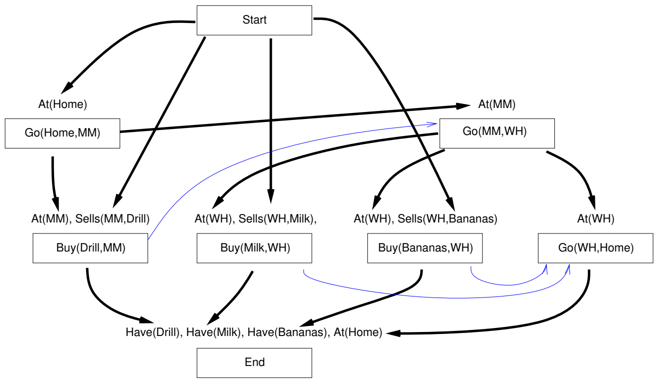
Parallel plans:
graph-based and sat-based approaches



Totally-ordered plans:
state space search approaches



Partially-ordered plans:
plan space approaches



Outline

◊ Motivation

◊ Partial plans

◊ Flaws

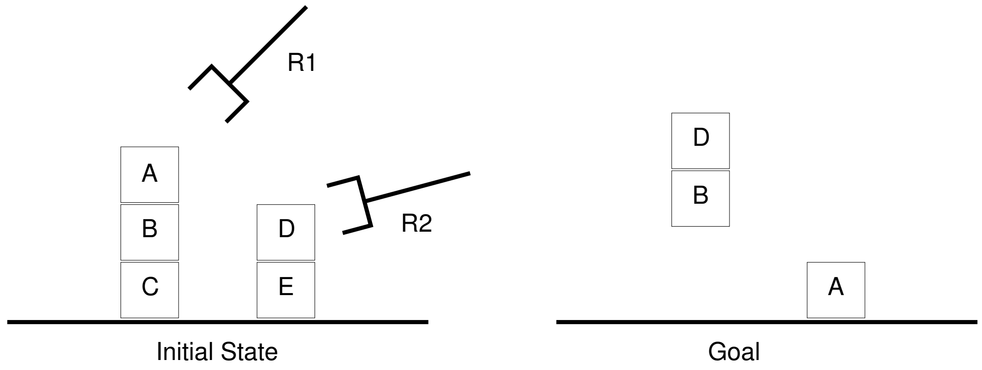
◊ Plan-space planning algorithm

◊ Example

Motivation

State-space search produces inflexible plans.

Part of the ordering in an action sequence is not related to causality:

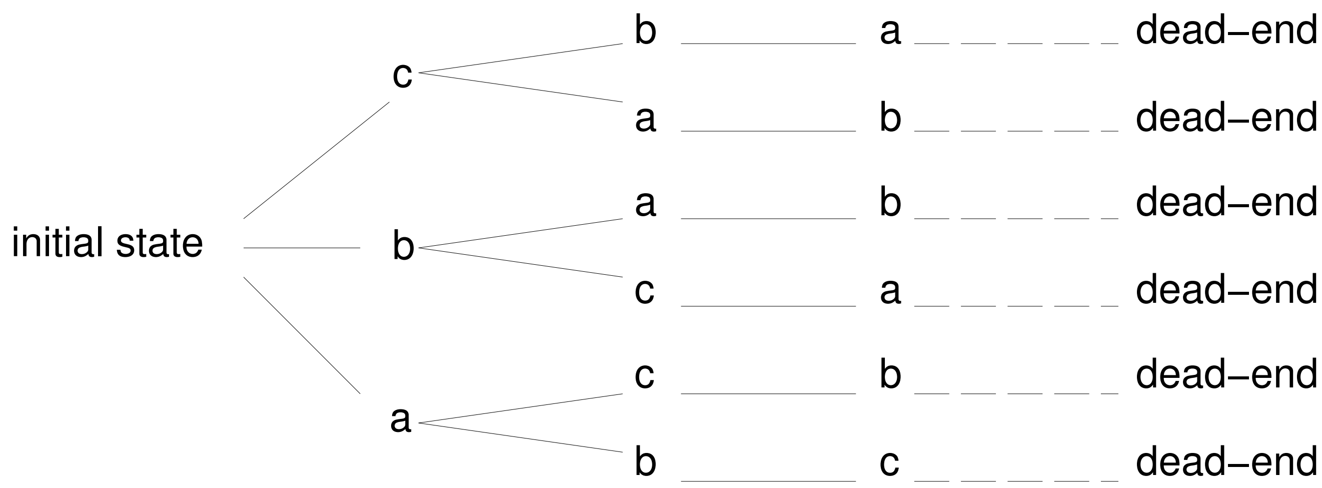


sequence:
$\langle$unstack(R1, A, B), unstack(R2, D, E), putdown(R1, A), stack(R2, D, B)$\rangle$

partially ordered plan only needs: unstack(R1, A, B) $<$ putdown(R1, A), unstack(R2, D, E) $<$ stack(R2, D, B), and unstack(R1, A, B) $<$ stack(R2, D, B)

Motivation

State-space search wastes time examining many different orderings of the same set of actions:



Not ordering actions unecessarily can speed up planning

Motivation

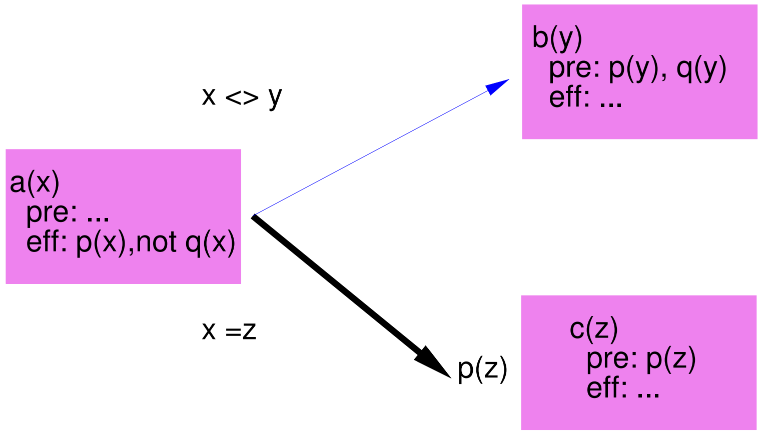
Plan space search:

- no notion of states, just partial plans
- adopts a least-commitment strategy: don't commit to orderings, instantiations, etc, unless necessary
- produces a partially ordered plan: represents all sequences of actions compatible with the partial ordering
- benefits: speed-ups (in principle), flexible execution, easier replanning

Plan space search: basic idea

Plan-space search builds a **partial plan**:

- multiset O of operators $\{o_1, \ldots, o_n\}$
- set $<$ of ordering constraints $o_i < o_j$ (with transitivity built in)
- set B of binding constraints $x = y$, $x \neq y$, $x \in D$, $x \notin D$, substitutions
- set L of causal links $o_i \xrightarrow{p} o_j$ stating that (effect p) of o_i establishes precondition p of o_j, with $o_i < o_j$ and binding constraints in B for parameters of o_i and o_j appearing in p



Plan-space search: basic idea

Nodes are partial plans

- initial node is $(O : \{\text{start}, \text{end}\}, <: \{\text{start} < \text{end}\}, B : \{\,\}, L : \{\,\})$ with $\text{EFF}(\text{start}) = s_0$ and $\text{PRE}(\text{end}) = g$.

Successors are determined by plan refinment operations

- each operation add elements to O, $<$, B, L to resolve a flaw in the plan

Search through the plan space until a partial plan is found which has no flaw:

- no open precondition: all preconditions of all operators in O are established by causal links in L
- no threat (each linearisation is safe): for every causal link $o_i \xrightarrow{p} o_j$, every o_k with $\text{EFF}^-(o_k)$ unifable with p is such that $o_k < o_i$ or $o_j < o_k$
- $<$ and B are consistent

How to resolve an open precondition flaw?

Flaw: an operator o in the plan has a precondition p which is not established

p

o

holding(r,A)

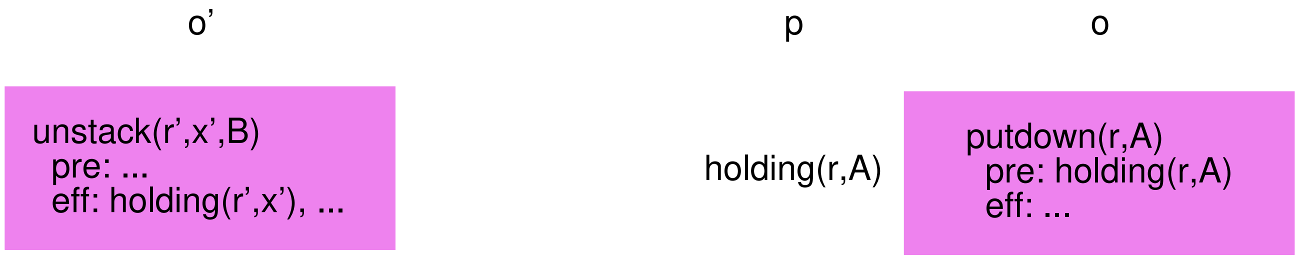
putdown(r,A)
pre: holding(r,A)
eff: ...

How to resolve an open precondition flaw?

Flaw: an operator o in the plan has a precondition p which is not established

Resolving the flaw:

1. find an operator o' (either already in the plan or insert it) which can be used to establish p, i.e. o' can be ordered before o and one of its effects can unify with p

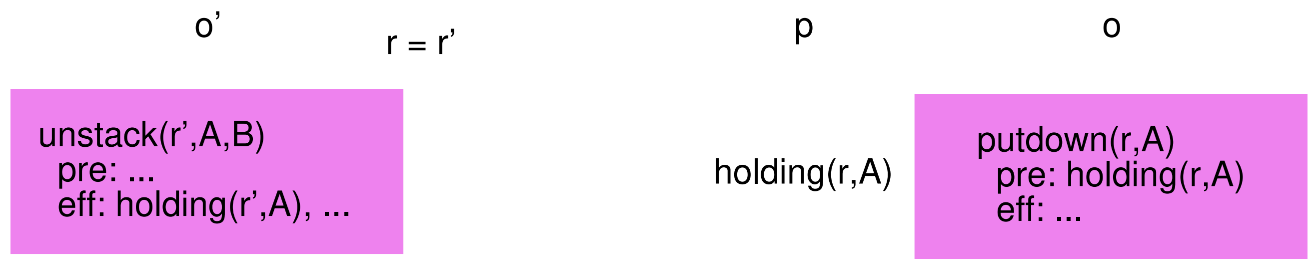


How to resolve an open precondition flaw?

Flaw: an operator o in the plan has a precondition p which is not established

Resolving the flaw:

1. find an operator o' (either already in the plan or insert it) which can be used to establish p, i.e. o' can be ordered before o and one of its effects can unify with p
2. add to B binding constraints to unify the effect of o' with p

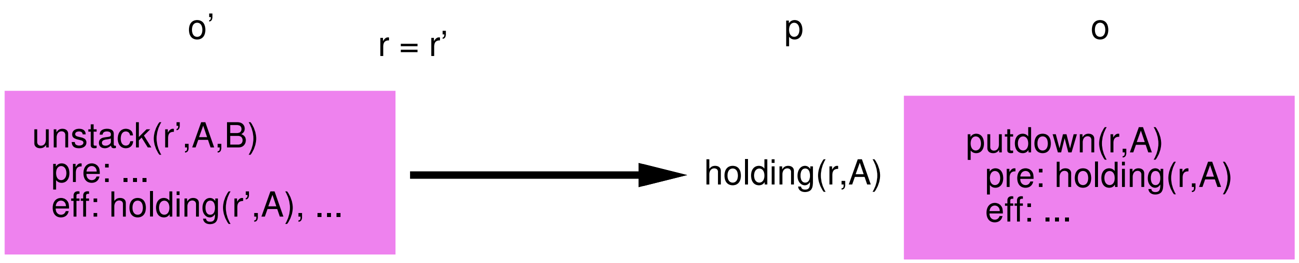


How to resolve an open precondition flaw?

Flaw: an operator o in the plan has a precondition p which is not established

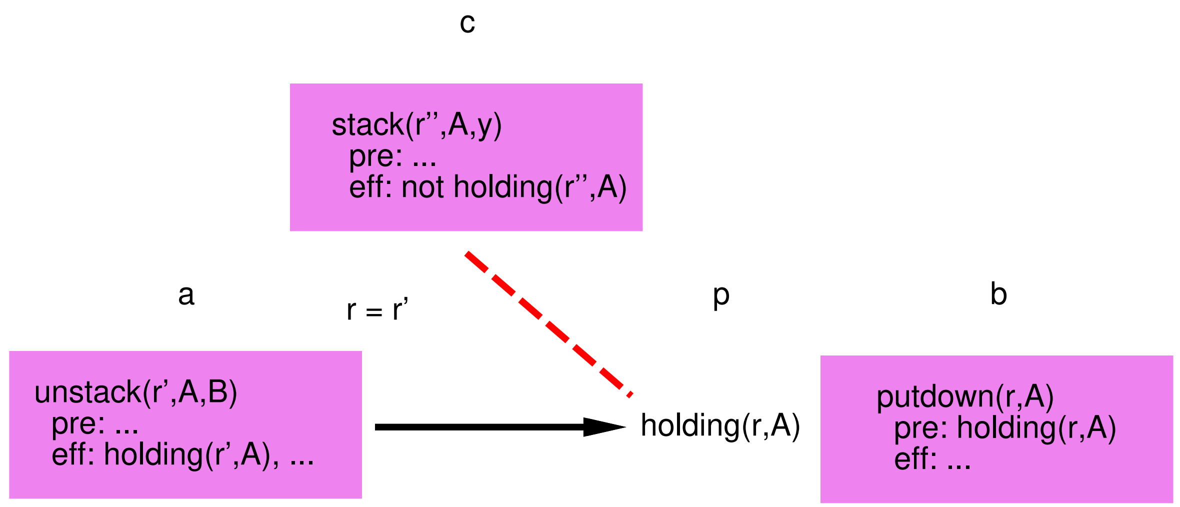
Resolving the flaw:

1. find an operator o' (either already in the plan or insert it) which can be used to establish p, i.e. o' can be ordered before o and one of its effects can unify with p
2. add to B binding constraints to unify the effect of o' with p
3. add to L the causal link $o' \xrightarrow{p} o$ (and the ordering constraint $o' < o$).



How to resolve a threat flaw?

Flaw: An operator a establishes a condition p for operator b, but another operator c is capable of deleting p before b gets to use it

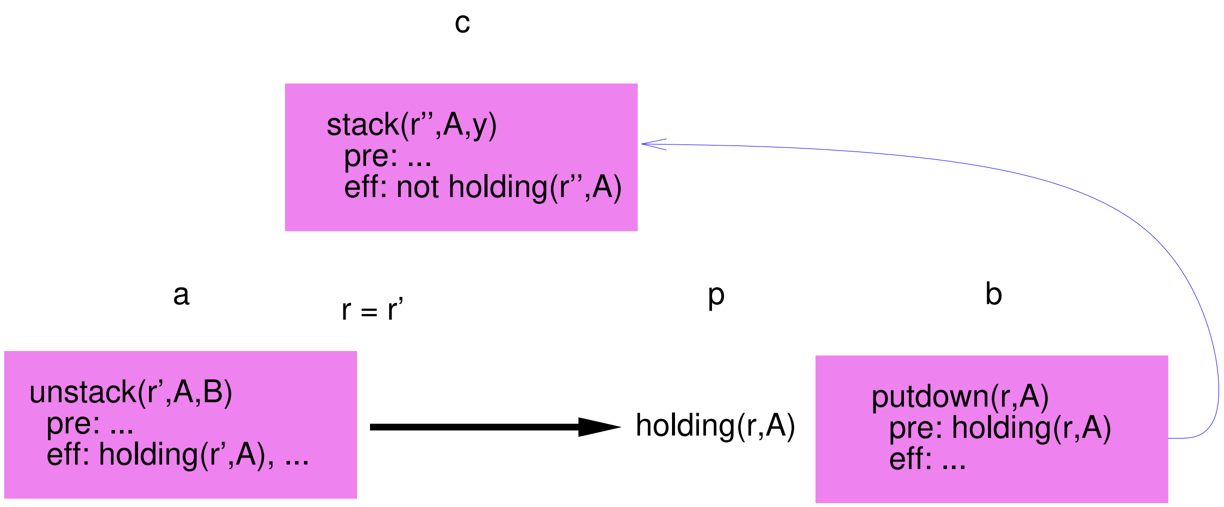


How to resolve a threat flaw?

Flaw: An operator a establishes a condition p for operator b, but another operator c is capable of deleting p before b gets to use it

Resolving the flaw - 3 possibilities:

1. order c after b

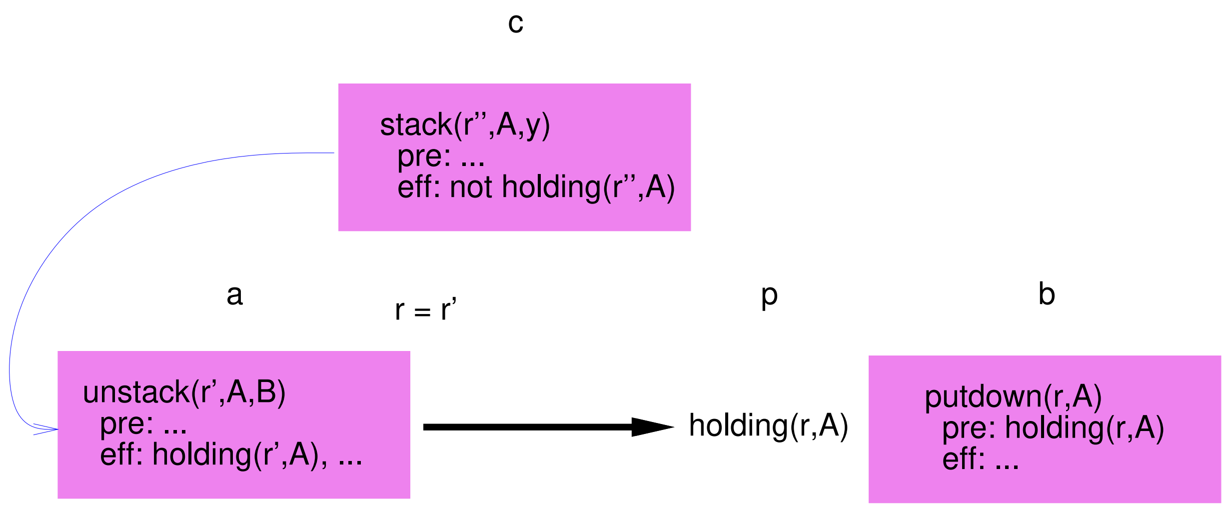


How to resolve a threat flaw?

Flaw: An operator a establishes a condition p for operator b, but another operator c is capable of deleting p before b gets to use it

Resolving the flaw - 3 possibilities:

1. order c after b
2. order c before a

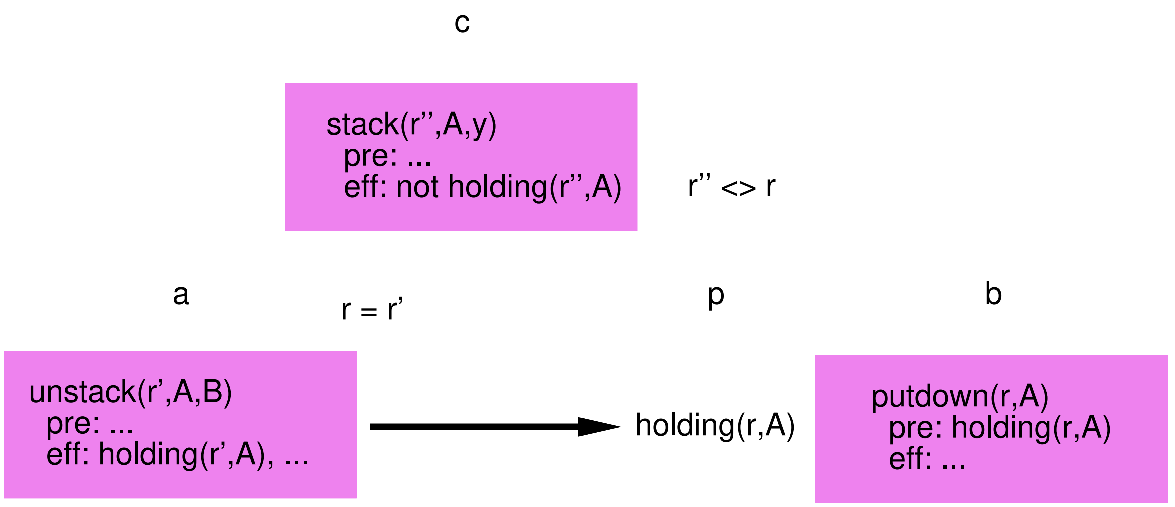


How to resolve a threat flaw?

Flaw: An operator a establishes a condition p for operator b, but another operator c is capable of deleting p before b gets to use it

Resolving the flaw - 3 possibilities:

1. order c after b
2. order c before a
3. add a binding constraint preventing c to delete p



Plan-space planning algorithm

```
function PLAN-SPACE-PLANNING(π) returns a plan, or failure
    F ← OPEN-PRECONDITIONS(π) ∪ THREATS(π)
    if F = { } then return π
    select a flaw f ∈ F
    R ← RESOLVE(f, π)
    if R = { } then return failure
    choose a resolver r ∈ R
    π′ ← REFINE(r, π)
    return PLAN-SPACE-PLANNING(π′)
```

PLAN-SPACE-PLANNING is sound and complete

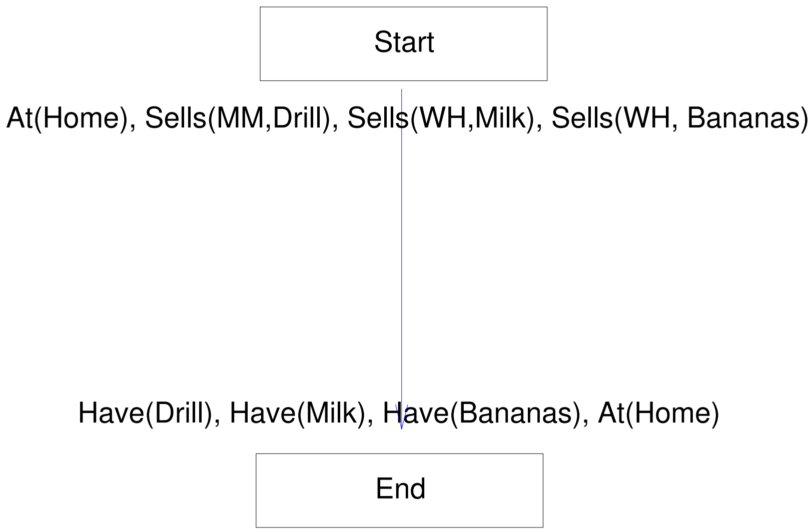
Grounded variant: no binding constraints needed

Example

- operator Start
  - precondition: $\{\,\}$
  - effect: $\{$At$($Home$)$, Sells$($MM,Drill$)$, Sells$($WH,Milk$)$, Sells$($WH,Bananas$)\}$
- operator End
  - precondition: $\{$At$($Home$)$, Have$($Drill$)$, Have$($Milk$)$, Have$($Bananas$)\}$
  - effect: $\{\,\}$
- operator Go(l, l')
  - precondition: $\{$At$(l)\}$
  - effect: $\{$At(l'), $\neg$At$(l)\}$
- operator Buy(i, s)
  - precondition: $\{$At(s), Sells$(s, i)\}$
  - effect: $\{$Have$(i)\}$

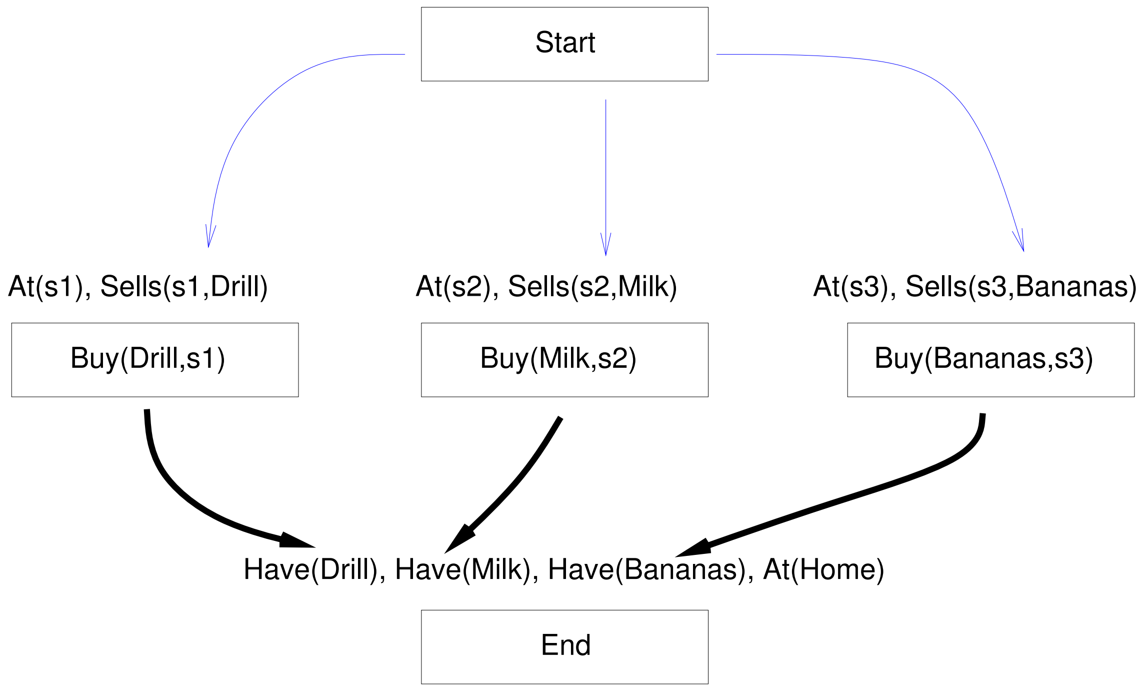
Example

Initial Plan



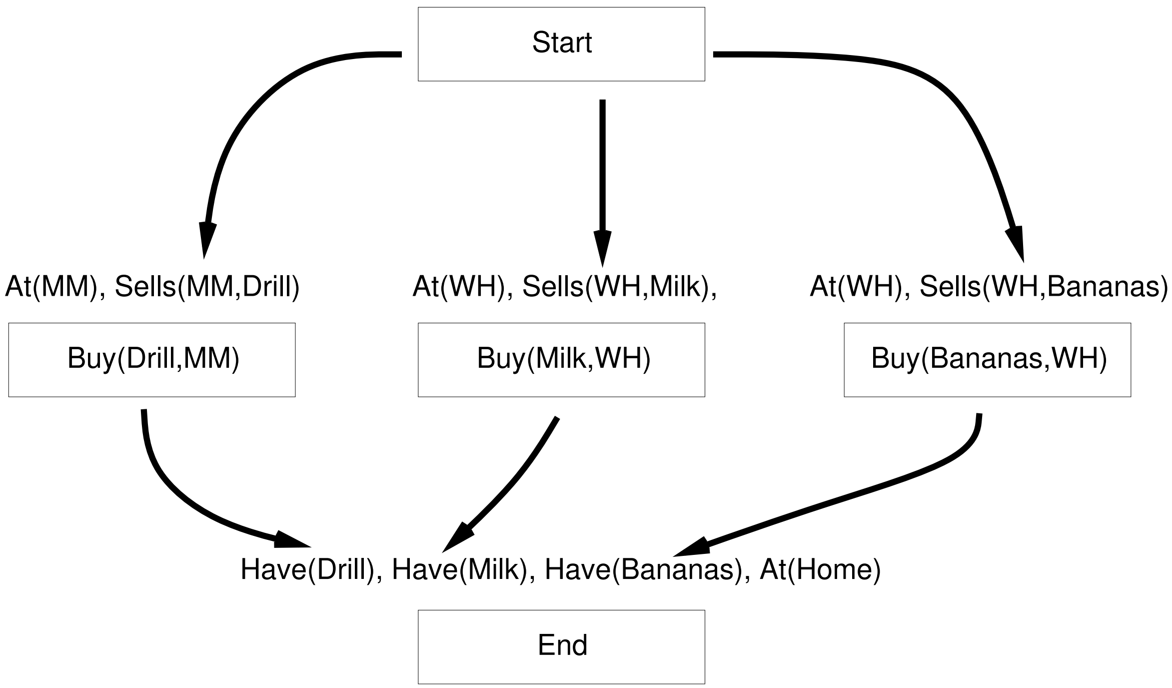
Example

The only possible ways to establish the "Have" preconditions



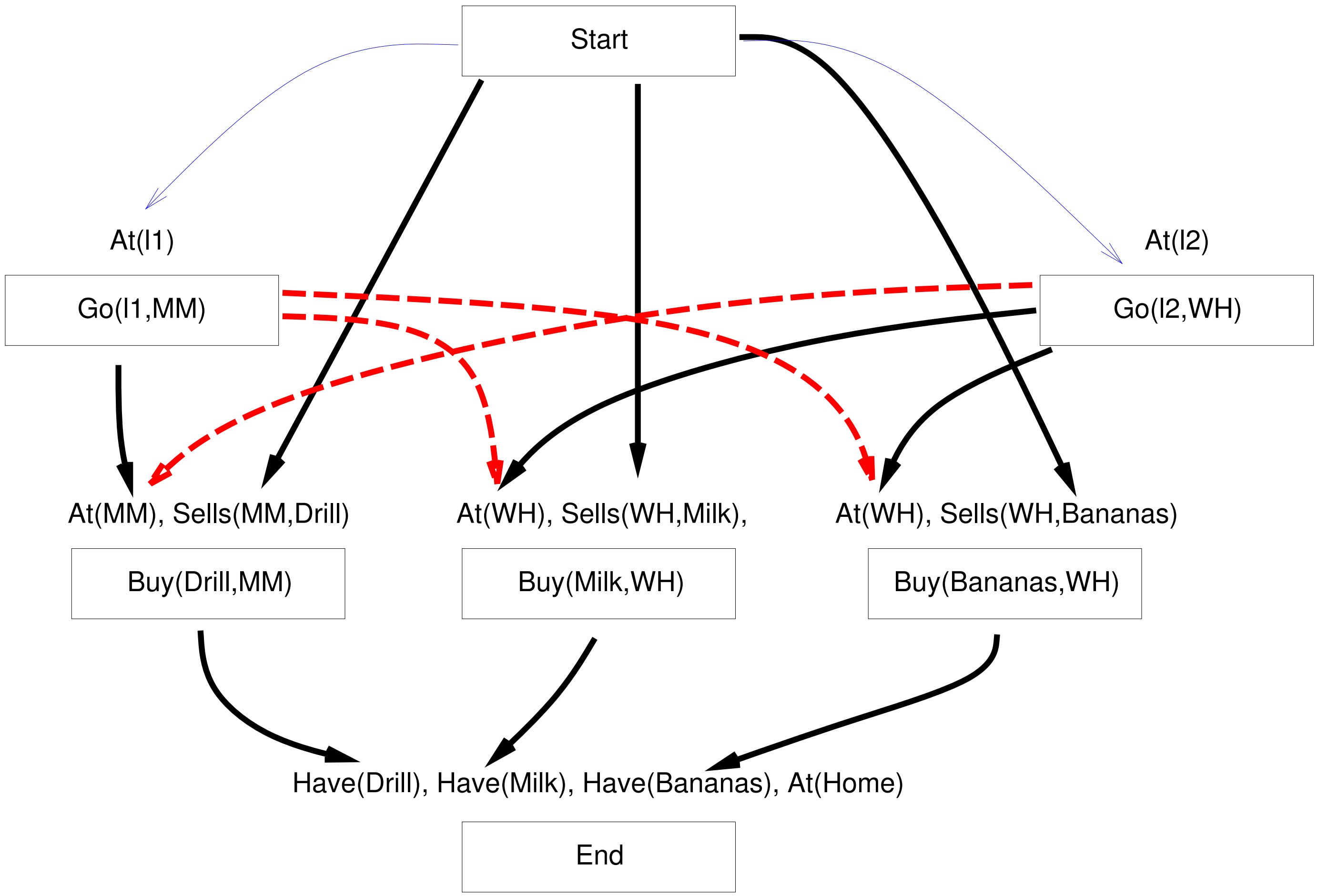
Example

The only possible ways to establish the "Sells" preconditions



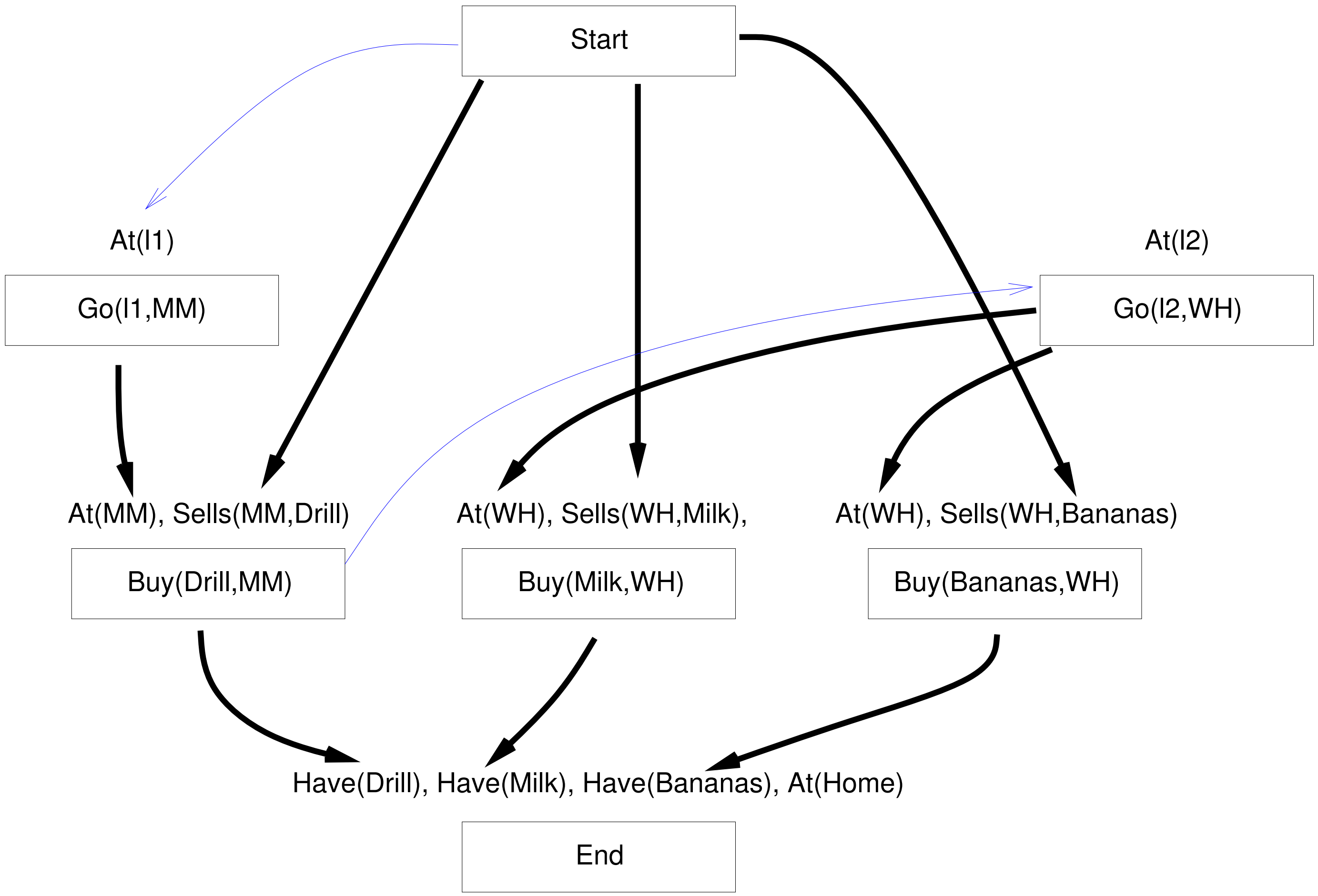
Example

The only ways to establish At(MM) and At(WH).
Note the threats



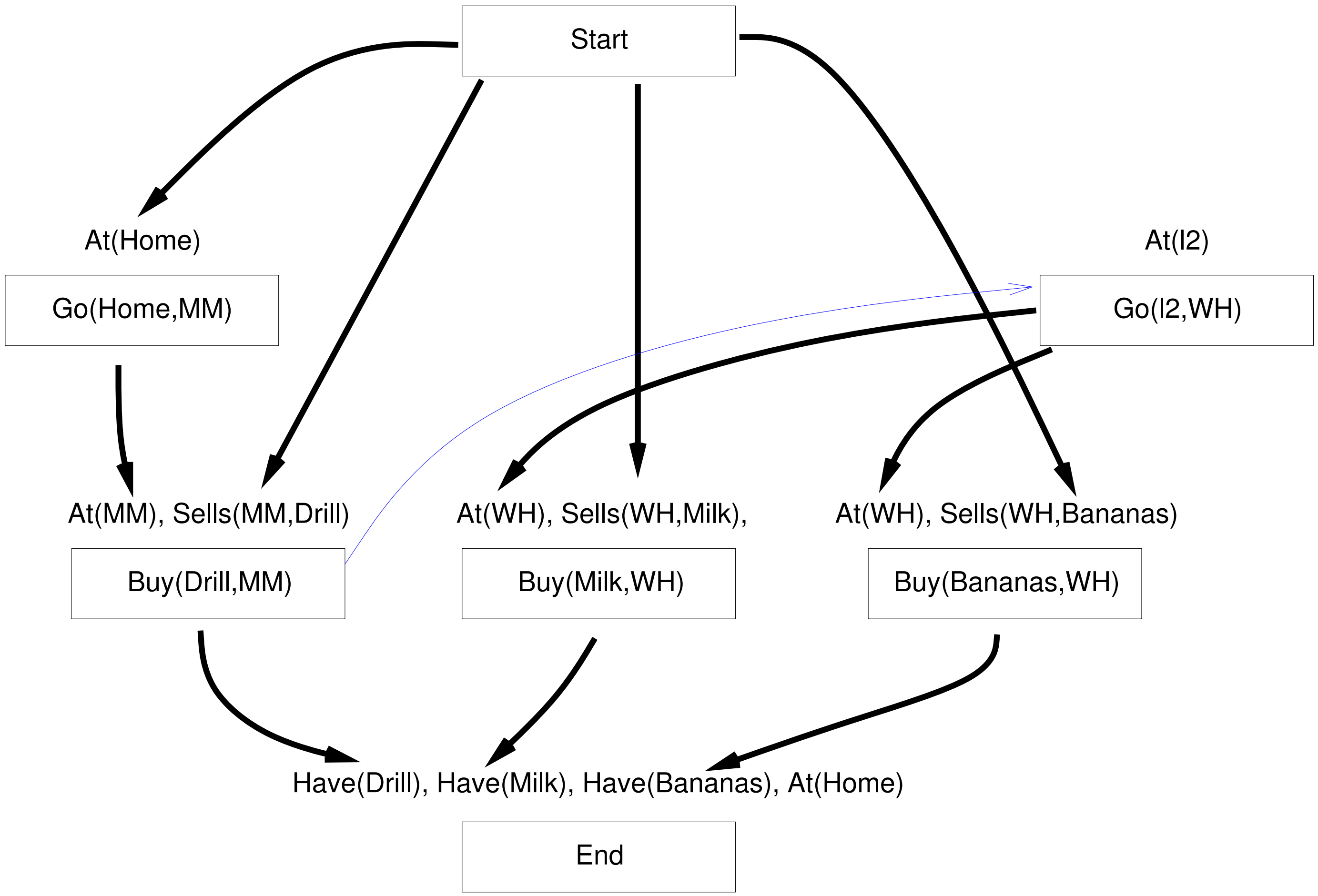
Example

To resolve the 3rd threat, order Go$(l2, \mathsf{WH})$ after Buy(Drill).
This resolves all three threats.



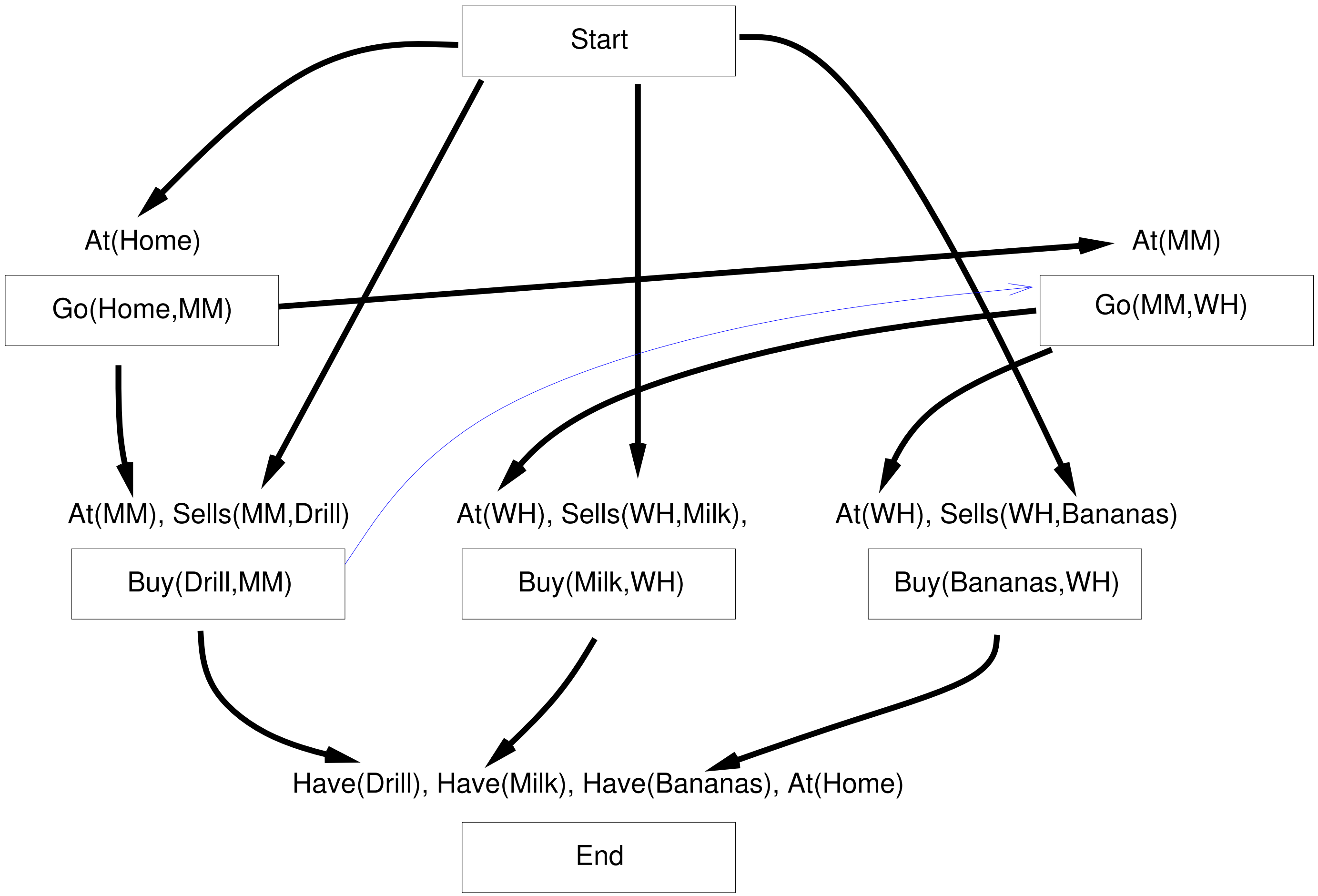
Example

Add binding constraint $l1 =$ Home and causal link to establish At($l1$)



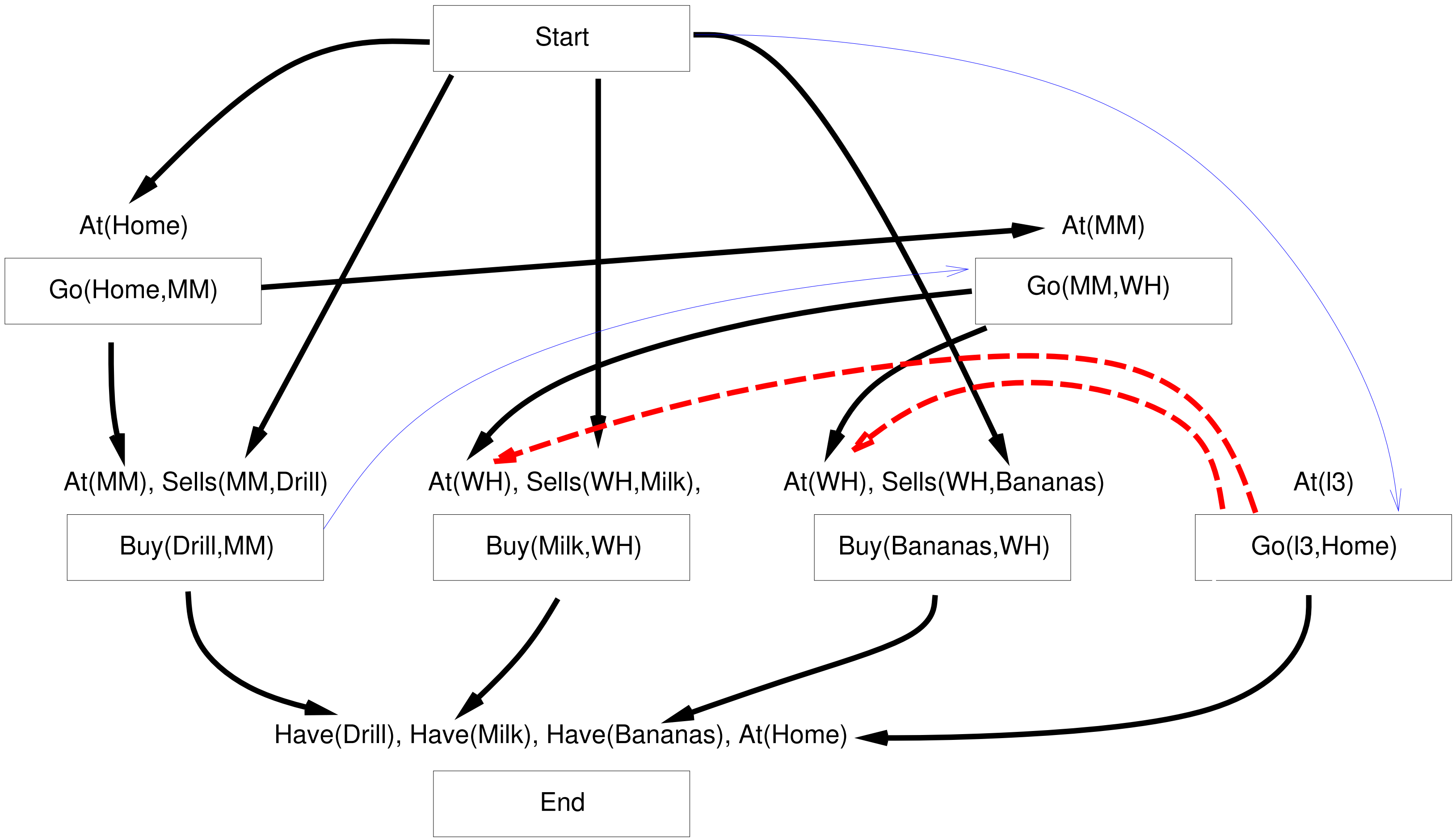
Example

Add binding constraint $l2 =$ MM and causal link to establish At($l2$)



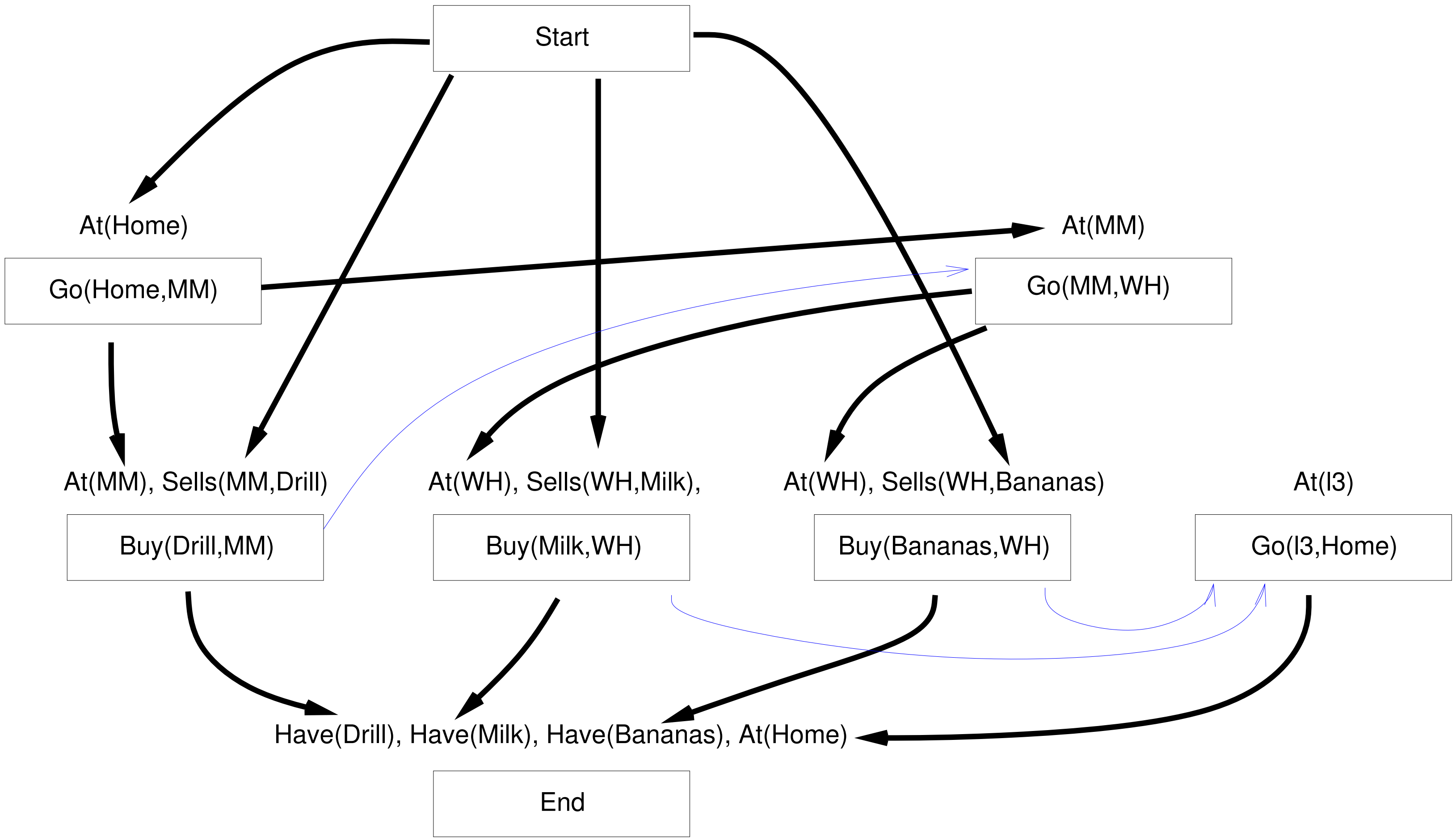
Example

Establish At(Home) for end.
Note the threats.



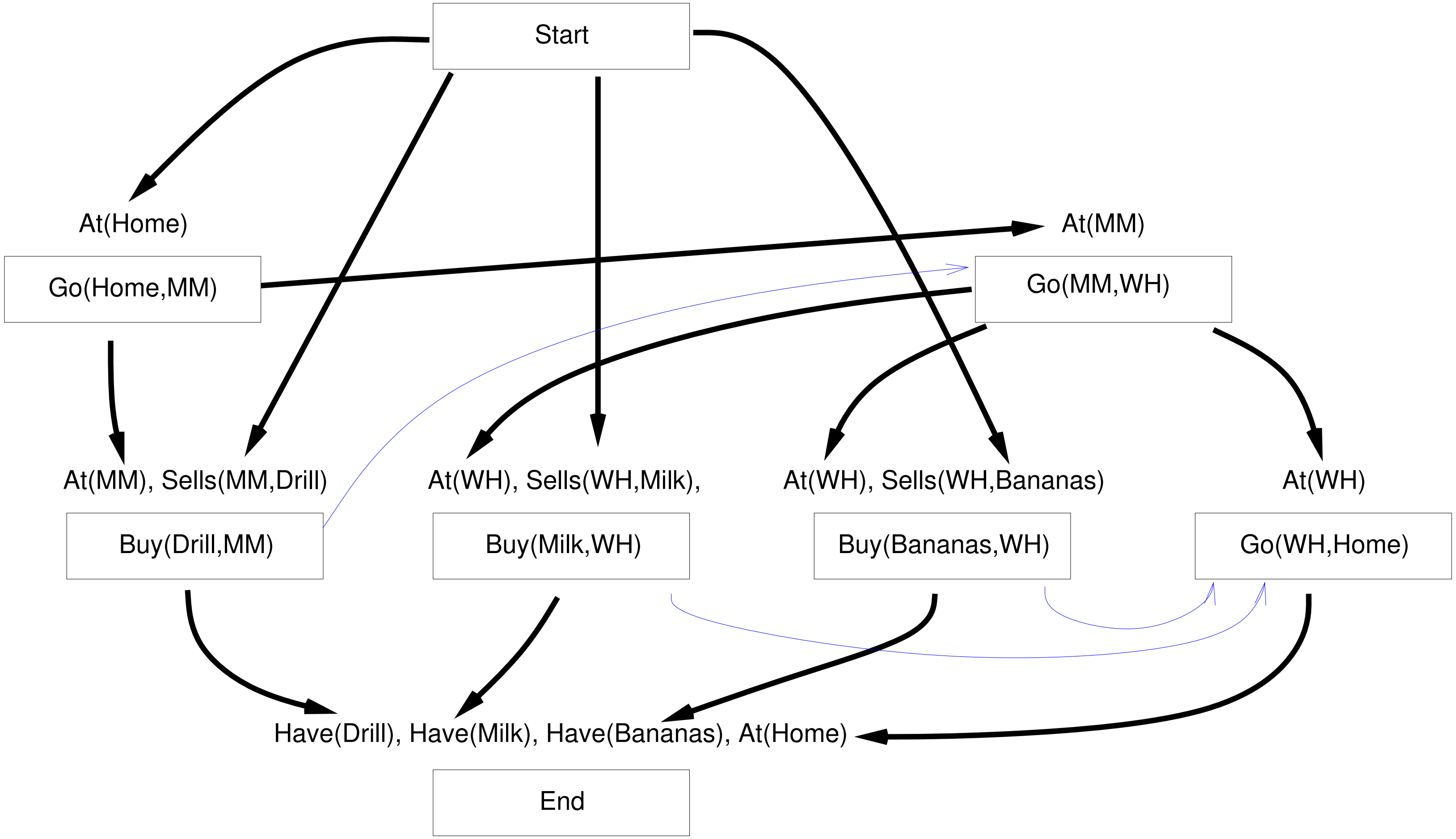
Example

Order Go(Home) after Buy(Milk) and Buy(Banana) to remove the threats.



Example

Add binding constraint $l3 =$ WH and causal link to establish At$(l3)$.
The plan is flawless.



Summary

Graph-based planning produces a polynomial-size graph that gives us a necessary condition for the existence of a parallel plan of a given length. If one really exist, it can be extracted by backward search through the graph.

SAT planning uses a SAT solver to solve the bounded (parallel) plan generation problem. This is efficient when optimal parallel plans are short. Logical planning formalisms must deal with the frame problem.

State-space planning produces totally-ordered plans by a forward or backward search in the state space. This requires domain-independent heuristics or domain-specific control rules to be efficient

Plan-space planning produces partially-ordered plans. This approach does not commit to orderings or bindings unless necessary. It searches the space of partial plans, refining the plan at each step to remove flaws.

Current planning research extends these methods to handle time, uncertainty, multiple agents, discrete/continuous systems, and real world applications.