Foundations of Automated Planning

Roman Barták

Charles University, Czech Republic

Planning problem is given by a triple (O,s_0,g).

- O defines the the **operators** and predicates used

  operator o is a triple (name(o), precond(o), effects(o))

  - **name(o):** name of the operator in the form $n(x_1,\ldots,x_k)$
  - **precond(o):** literals that must hold in the state so the operator is applicable on it
  - **effects(o):** literals that will become true after operator application

  action is a fully instantiated operator (substitute constants to attributes)

- s_0 is an **initial state** (provides particular constants - objects)
- g is a **goal** (a set of instantiated literals)
  - state **s** satisfies the goal condition **g** if and only if $\mathbf{g}^+ \subseteq \mathbf{s} \ \wedge \ \mathbf{g}^- \cap \mathbf{s} = \emptyset$

Plan π is a sequence of actions $\langle a_1,a_2,\ldots,a_k\rangle$.

Plan π is a **solution of** P if and only if $\gamma(s_0,\pi)$ satisfies g.

Action **a** is **applicable** to state **s** if and only if:

$\text{precond}^+(\mathbf{a}) \subseteq \mathbf{s} \ \wedge \ \text{precond}^-(\mathbf{a}) \cap \mathbf{s} = \varnothing$

transition function γ:

$\gamma(\mathbf{s},\mathbf{a}) = (\mathbf{s} - \text{effects}^-(\mathbf{a})) \cup \text{effects}^+(\mathbf{a})$, if **a** is applicable on **s**

Action **a** is **relevant** for a goal **g** if and only if:

action contributes to g: $g \cap \text{effects}(a) \neq \varnothing$

action effects are not in conflict with g:

$g^- \cap \text{effects}^+(a) = \varnothing \ /\backslash \ g^+ \cap \text{effects}^-(a) = \varnothing$

regression set for a goal g for a (relevant) action a:

$\gamma^{-1}(g,a) = (g - \text{effects}(a)) \cup \text{precond}(a)$

Almost all planning algorithms are based on some form of **search**.

The algorithms differ in the search space to be explored and in the way of exploration.

- **State Space Planning**
  - search nodes directly correspond to world states
- **Plan Space Planning**
  - search nodes correspond to partial plans

The search space corresponds to the state space of the planning problem.

- search nodes correspond to world states
- arcs correspond to state transitions by means of actions
- the task is to find a path from the initial state to some goal state

Basic approaches

- forward search
- backward search
  - lifting
  - STRIPS

Note: all algorithms will be presented for the classical representation

Start in the initial state and go towards some goal state.

We need to know:

- whether a given state is a **goal state**
- how to find a set of **applicable actions** for a given state
- how to define a state after **applying a given actio**n

Forward planning: algorithm

Forward-search(O, s_0, g)

$s \leftarrow s_0$

$\pi \leftarrow$ the empty plan

loop

if s satisfies g then return π

$E \leftarrow \{a | a$ is a ground instance of an operator in O,

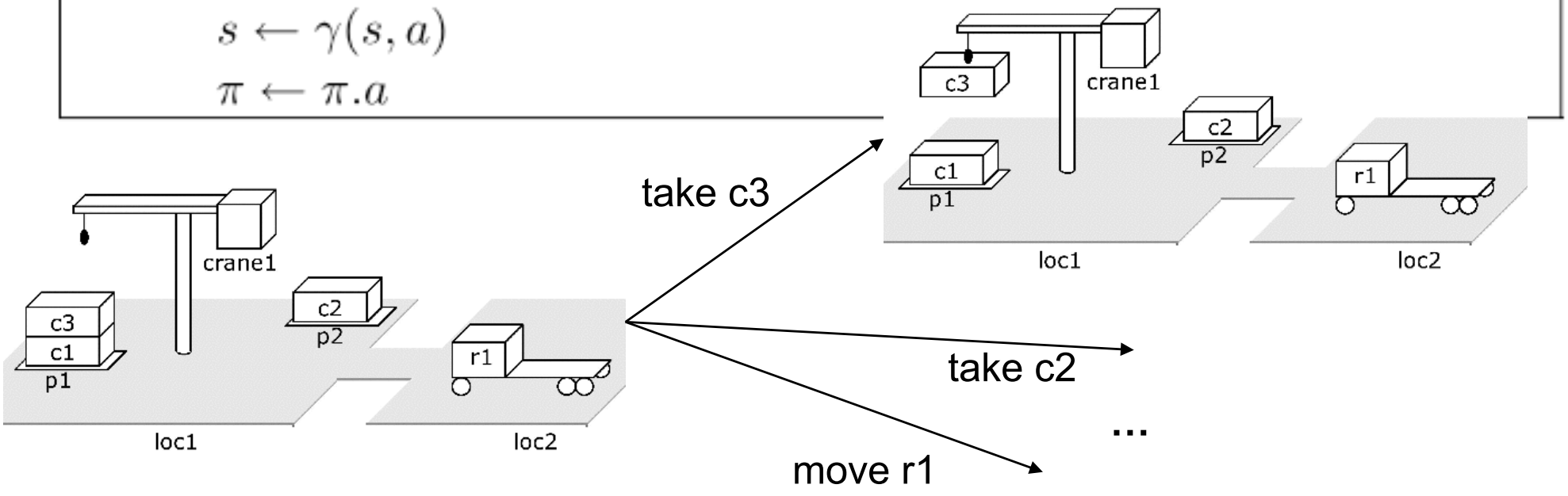
and $\text{precond}(a)$ is true in $s\}$

if $E = \emptyset$ then return failure

nondeterministically choose an action $a \in E$

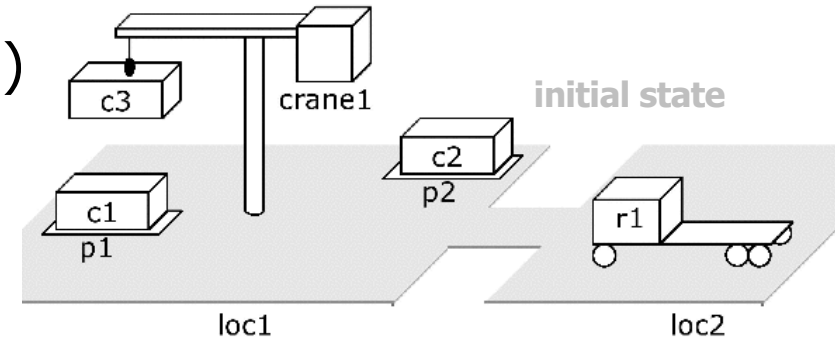
$s \leftarrow \gamma(s, a)$

$\pi \leftarrow \pi.a$



Forward planning: example

{belong(crane1,loc1), adjacent(loc2,loc1)
holding(crane1,c3), unloaded(r1),
at(r1,loc2), ¬occupied(loc1),
occupied(loc2),…}



move(r1,loc2,loc1)

move(r, l, m)
;; robot r moves from location l to location m
precond: adjacent(l, m), at(r, l), ¬ occupied(m)
effects: at(r, m), occupied(m), ¬ occupied(l), ¬ at(r, l)

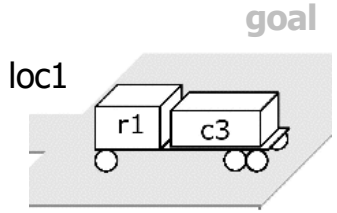
{belong(crane1,loc1),
adjacent(loc2,loc1), holding(crane1,c3), unloaded(r1),
at(r1,loc1), occupied(loc1), …}

load(crane1,loc1,c3,r1)

load(k, l, c, r)
;; crane k at location l loads container c onto robot r
precond: belong(k, l), holding(k, c), at(r, l), unloaded(r)
effects: empty(k), ¬ holding(k, c), loaded(r, c), ¬ unloaded(r)

{belong(crane1,loc1), adjacent(loc2,loc1),
empty(crane1), loaded(r1,c3),
at(r1,loc1), occupied(loc1), …}

Goal = {at(r1,loc1),loaded(r1,c3)}



Forward planning algorithm is sound.

- If some plan is found then it is a solution plan.
- It is easy to verify by using $s = \gamma(s_0,\pi)$.

Forward planning algorithm is complete.

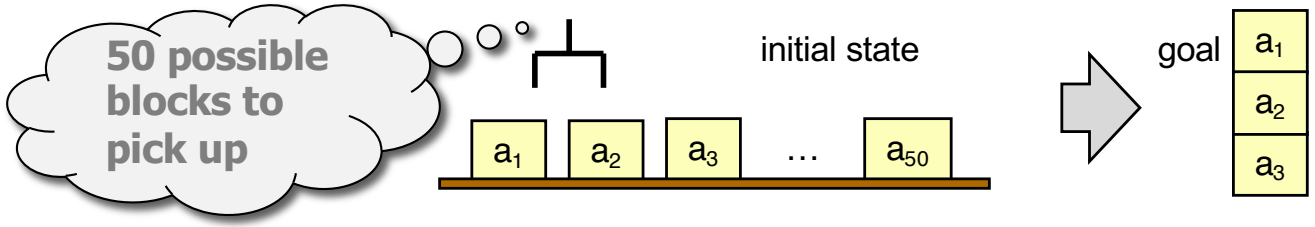
- If there is any solution plan then at least one search branch corresponds to this plan.
- induction by the plan length
- at each step, the algorithm can select the correct action from the solution plan (if correct actions were selected in the previous steps)

We need to implement the presented algorithm in a deterministic way:

- **breadth-first search**
  - sound, complete, but memory consuming
- **depth-first search**
  - sound, completeness can be guaranteed by loop checks (no state repeats at the same branch)
- **A***
  - if we have some admissible heuristic
  - the most widely used approach

What is the major problem of forward planning?

Large branching factor – the number of options



- This is a problem for deterministic algorithm that needs to explore the possible options.

Possible approaches:

- **heuristic** recommends an action to apply
- **pruning** of the search space
  - For example, if plans π_1 and π_2 reached the same state then we know that plans $\pi_1\pi_3$ and $\pi_2\pi_3$ will also reach the same state. Hence the longer of the plans π_1 and π_2 does not need to expanded.
    We need to remember the visited states ☹.

Start with a goal (not a goal state as there might be more goal states) and through sub-goals try to reach the initial state.

We need to know:

- whether the state **satisfies the current goal**
- how to find **relevant actions** for any goal
- how to define the **previous goal** such that the action converts it to a current goal

Action a is relevant for a goal g if and only if:

- action **a** contributes to goal **g**: $\mathbf{g} \cap \text{effects}(\mathbf{a}) \neq \emptyset$
- effects of action **a** are not conflicting goal **g**:
  - $g^- \cap \text{effects}^+(a) = \emptyset$
  - $g^+ \cap \text{effects}^-(a) = \emptyset$

A **regression set** of the goal **g** for (relevant) action **a** is

$\gamma^{-1}(g,a) = (g - \text{effects}(a)) \cup \text{precond}(a)$

Example:

goal: **{on(a,b), on(b,c)}**

action **stack(a,b)** is relevant

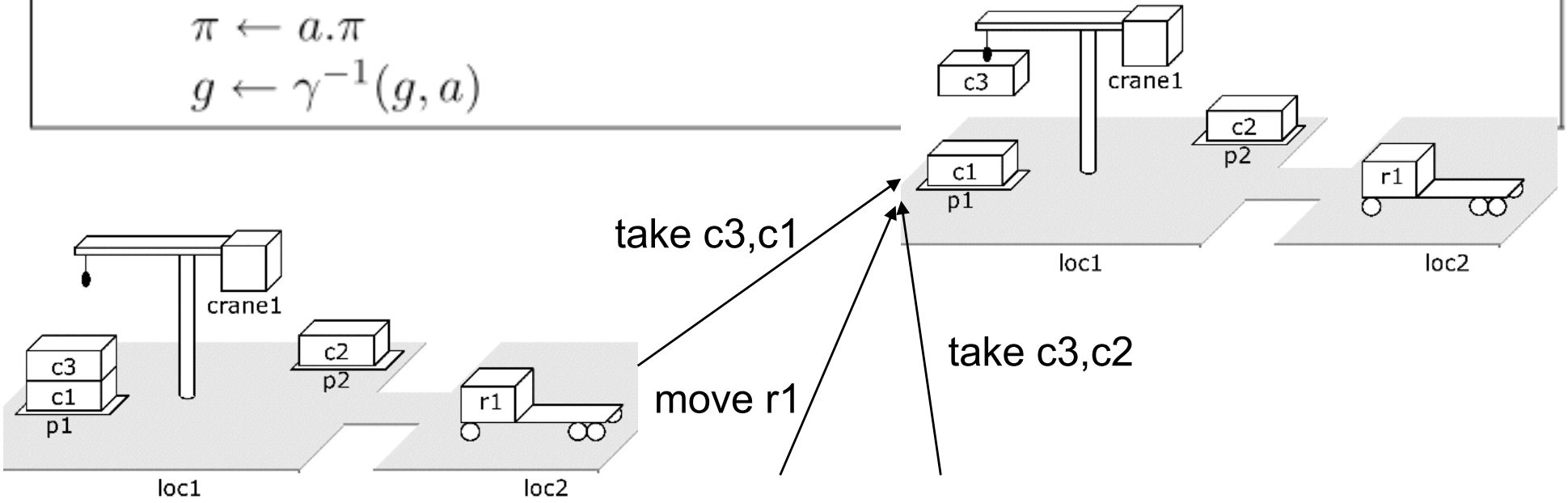
stack(*x*,*y*)
Precond: holding(*x*), clear(*y*)
Effects: ~holding(*x*), ~clear(*y*),
on(*x*,*y*), clear(*x*), handempty

by backward application of the action we get a new goal:

{holding(a), clear(b), on(b,c)}

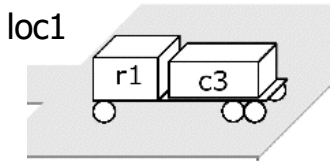
Backward planning: algorithm

Backward-search(O, s_0, g)
  $\pi \leftarrow$ the empty plan
  loop
    if s_0 satisfies g then return π
    $A \leftarrow \{a | a$ is a ground instance of an operator in O
        and $\gamma^{-1}(g, a)$ is defined$\}$
    if $A = \emptyset$ then return failure
    nondeterministically choose an action $a \in A$
    $\pi \leftarrow a.\pi$
    $g \leftarrow \gamma^{-1}(g, a)$



Backward planning: an example

Goal = {at(r1,loc1),loaded(r1,c3)}



load(crane1,loc1,c3,r1)

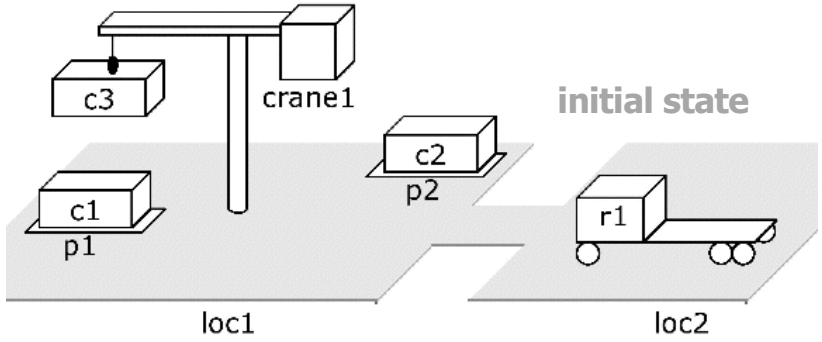
load(k, l, c, r)
;; crane k at location l loads container c onto robot r
precond: belong(k, l), holding(k, c), at(r, l), unloaded(r)
effects: empty(k), $\neg$ holding(k, c), loaded(r, c), $\neg$ unloaded(r)

{at(r1,loc1), belong(crane1,loc1),
holding(crane1,c3), unloaded(r1)}

move(r1,loc2,loc1)

move(r, l, m)
;; robot r moves from location l to location m
precond: adjacent(l, m), at(r, l), $\neg$ occupied(m)
effects: at(r, m), occupied(m), $\neg$ occupied(l), $\neg$ at(r, l)

{belong(crane1,loc1), holding(crane1,c3),
unloaded(r1),
adjacent(loc2,loc1),
at(r1,loc2),
¬ occupied(loc1)}



Backward planning is **sound and complete**.

We can implement a **deterministic** version of the algorithm (via search).

- For completeness we need loop checks.
  - Let $(g_1,\ldots,g_k)$ be a sequence of goals. If $\exists i<k\ g_i \subseteq g_k$ then we can stop search exploring this branch.

Branching

- The number of options can be smaller than for the forward planning (less relevant actions for the goal).
- Still, it could be too large.
  - If we want a robot to be at the position loc51 and there are direct connections from states loc1,…,loc50, then we have 50 relevant actions. However, at this stage, the start location is not important!
  - We can instantiate actions only partially (some variables remain free. This is called **lifting**.

Backward planning: a lifted version

Lifted-backward-search(O, s_0, g)
  $\pi \leftarrow$ the empty plan
  loop
    if s_0 satisfies g then return π
    $A \leftarrow \{(o, \theta) | o$ is a standardization of an operator in O,
      θ is an mgu for an atom of g and an atom of effects (o),
      and $\gamma^{-1}(\theta(g), \theta(o))$ is defined$\}$
    if $A = \emptyset$ then return failure
    nondeterministically choose a pair $(o, \theta) \in A$
    $\pi \leftarrow$ the concatenation of $\theta(o)$ and $\theta(\pi)$
    $g \leftarrow \gamma^{-1}(\theta(g), \theta(o))$

Notes:

- standardization = a copy with fresh variables
- mgu = most general unifier
- by using the variables we can decrease the branching factor but the trade off is more complicated loop check

How can we further reduce the search space?

STRIPS algorithm reduces the search space of backward planning in the following way:

- **only part of the goal is assumed in each step, namely the preconditions of the last selected action**
  - instead of $\gamma^{-1}(\mathbf{s},\mathbf{a})$ we can use precond(**a**) as the new goal
  - the rest of the goal must be covered later
  - This makes the algorithm incomplete!
- **If the current state satisfies the preconditions of the selected action then this action is used and never removed later upon backtracking.**

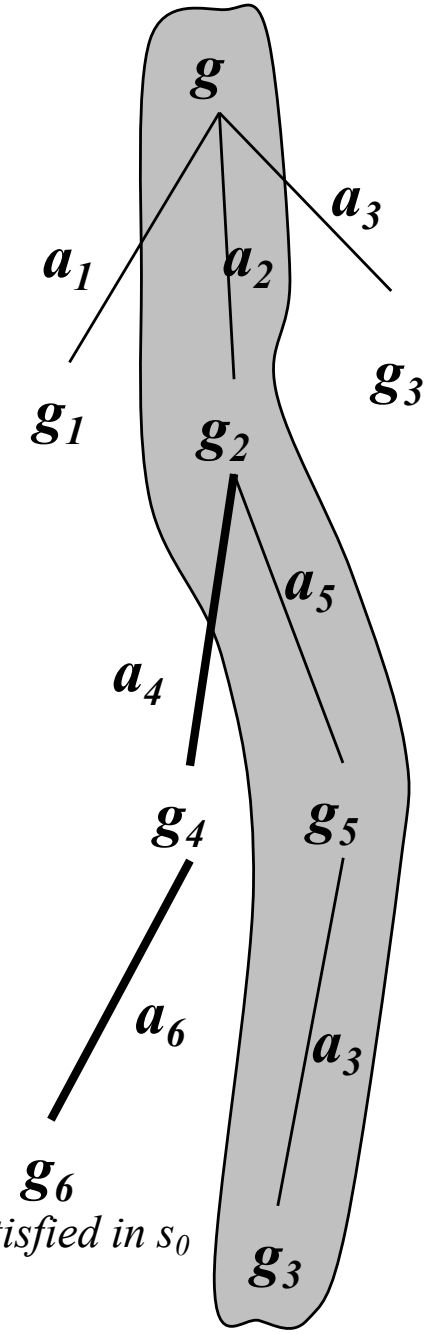
The original STRIPS algorithm is a lifted version of the algorithm below.

```
Ground-STRIPS(O, s, g)
    π ← the empty plan
    loop
        if s satisfies g then return π
        A ← {a | a is a ground instance of an operator in O,
                  and a is relevant for g}
        if A = ∅ then return failure
        nondeterministically choose any action a ∈ A
        π′ ← Ground-STRIPS(O, s, precond(a))
        if π′ = failure then return failure
        ;; if we get here, then π′ achieves precond(a) from s
        s ← γ(s, π′)
        ;; s now satisfies precond(a)
        s ← γ(s, a)
        π ← π . π′ . a
```

$g_2 = (g - \text{effects}(a_2)) \cup \text{precond}(a_2)$

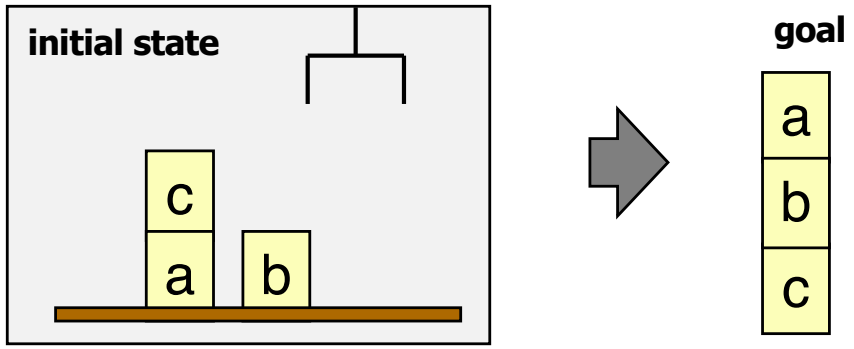
$\pi' = \langle a_6, a_4 \rangle$ is a plan for $\text{precond}(a_2)$

$s = \gamma(\gamma(s_0, a_6), a_4)$ is a state satisfying $\text{precond}(a_2)$



Sussman anomaly is a famous example that causes troubles to the STRIPS algorithm (the algorithm can only find redundant plans).

Block world



A plan found by STRIPS may look like this:

- unstack(c,a),putdown(c),pickup(a),stack(a,b)

  now we satisfied subgoal on(a,b)
- unstack(a,b),putdown(a),pickup(b),stack(b,c)

  now we satisfied subgoal on(b,c), but we need to re-satisfy on(a,b) again
- pickup(a),stack(a,b)

red actions can be deleted

Solving Sussman anomaly

- **interleaving plans**
  - plan-space planning
- **using domain dependent information**
  - When does a solution plan exist for a blocks world?
    - all blocks from the goal are present in the initial state
    - no block in the goal stays on two other blocks (or on itself)
    - …
  - How to find a solution plan?

    Actually, it is easy and very fast!
    - put all blocks on the table (separately)
    - build the requested towers

    We can do it even better with additional knowledge!

The principle of plan space planning is similar to backward planning:

- start from an **"empty" plan** containing just the description of initial state and goal
- **add other actions** to satisfy not yet covered (open) goals
- if necessary **add other relations** between actions in the plan

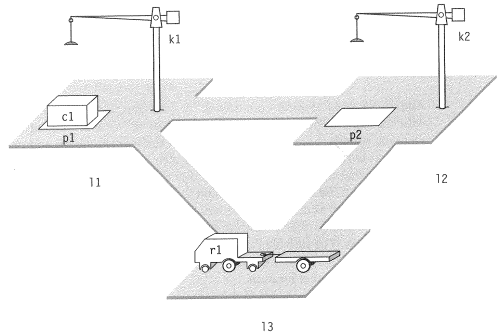
Planning is realised as **repairing flaws in a partial plan**

- go from one partial plan to another partial plan until a complete plan is found

Plan space planning: an example

Assume a partial plan with the following two actions:

- take(k1,c1,p1,l1)
- load(k1,c1,r1,l1)



Possible modifications of the plan:

- **adding a new action**
  - to apply action **load**, robot r1 must be at location l1
  - action **move**(r1,L,l1) moves robot r1 to location l1 from some location L
- **binding the variables**
  - action **move** is used for the right robot and the right location
- **ordering some actions**
  - the robot must **move** to the location before the action **load** can be used
  - the order with respect to action **take** is not relevant
- **adding a causal relation**
  - new action is added to move the robot to a given location that is a precondition of another action
  - the causal relation between **move** and **load** ensures that no other action between them moves the robot to another location

The initial state and the goal are encoded using two **special actions** in the initial partial plan:

- **Action a_0 represents the initial state** in such a way that atoms from the initial state define effects of the action and there are no preconditions. This action will be before all other actions in the partial plan.
- **Action a_∞ represents the goal** in a similar way – atoms from the goal define the precondition of that action and there is no effect. This action will be after all other actions.

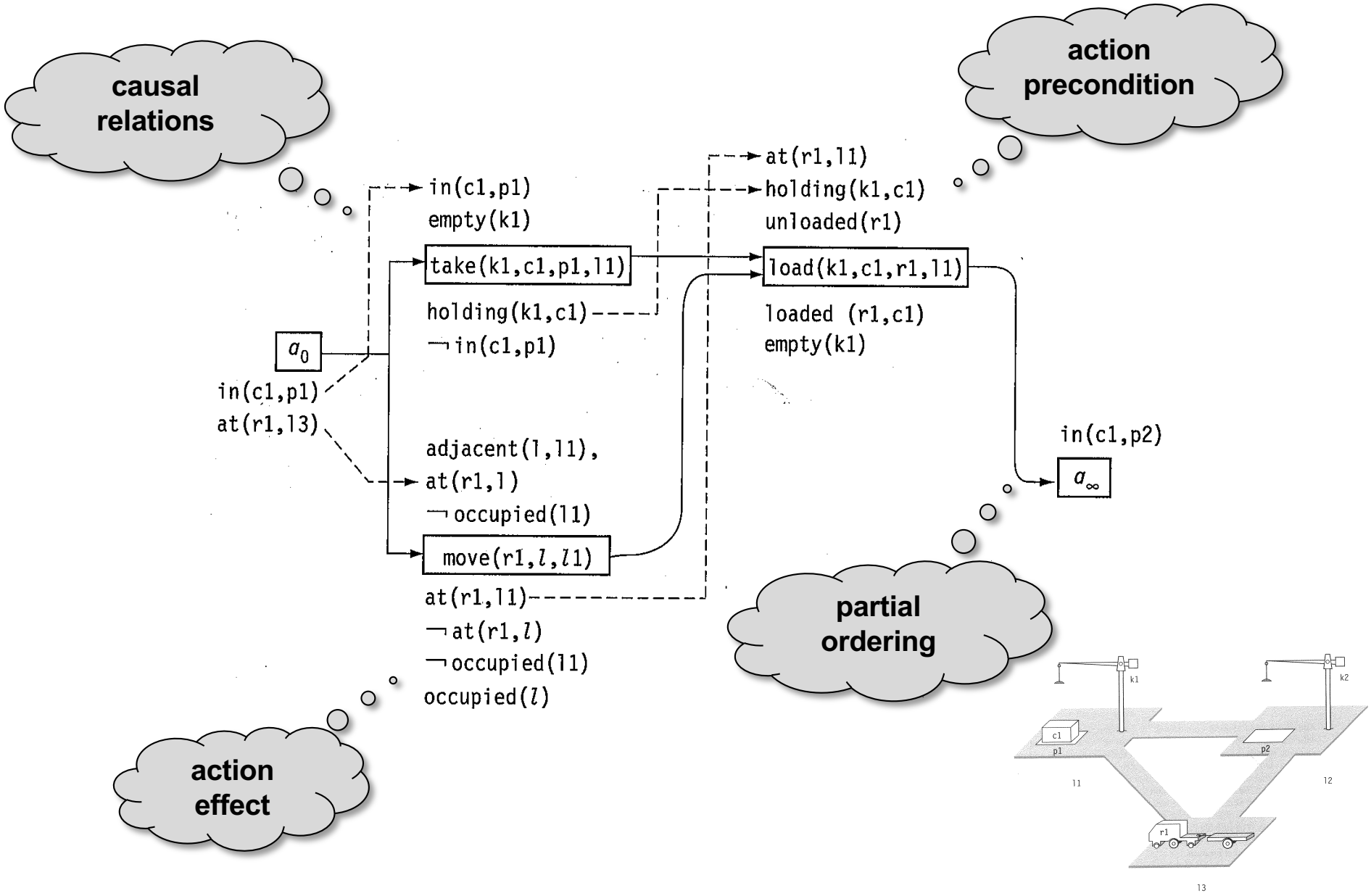
Planning is realised by **repairing flaws** in the partial plan.

The search nodes correspond to partial plans.

A partial plan Π is a tuple (A,<,B,L), where

- A is a set of partially instantiated planning operators $\{a_1,\ldots,a_k\}$
- $<$ is a partial order on A ($a_i<a_j$)
- B is set of constraints in the form x=y, $x\neq y$ or $x\in D_i$
- L is a set of causal relations ($a_i \rightarrow^p a_j$)
  - a_i,a_j are ordered actions $a_i<a_j$
  - p is a literal that is effect of a_i and precondition of a_j
  - B contains relations that bind the corresponding variables in p

Partial plan: an example



Open goal is an example of a **flaw**.

This is a precondition **p** of some operator **b** in the partial plan such that no action was decided to satisfy this precondition (there is no causal relation $a_i \rightarrow^p b$).

The open goal p of action b can be resolved by:

- finding an operator **a** (either present in the partial plan or a new one) that can give **p** (**p** is among the effects of **a** and **a** can be before **b**)
- binding the variables from **p**
- adding a causal relation $\mathbf{a} \rightarrow^{\mathbf{p}} \mathbf{b}$

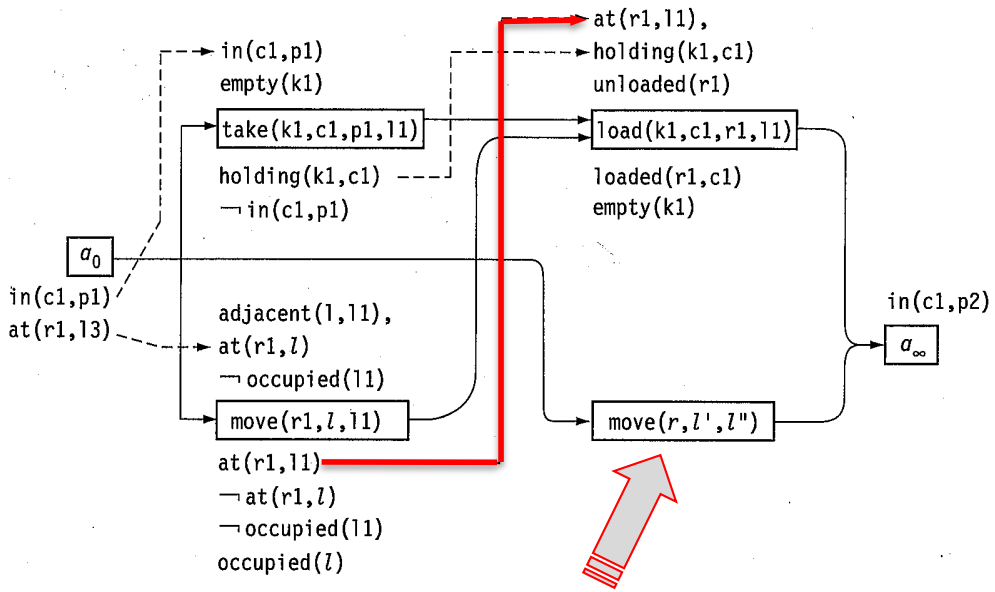
Threat is another example of **flaw**.

This is action that can influence existing causal relation.

- Let $a_i \rightarrow^p a_j$ be a causal relation and action **b** has among its effects a literal unifiable with the negation of **p** and action **b** can be between actions a_i and a_j. Then **b** is threat for that causal relation.

We can **remove the threat** by one of the following ways**:**

- ordering **b** before $\mathbf{a_i}$
- ordering **b** after $\mathbf{a_j}$
- binding variables in **b** in such a way that **p** does not bind with the negation of **p**



Partial plan Π = (A,<,B,L) is a **solution plan** for the problem P = (Σ,s_0,g) if:

- partial ordering < and constraints B are globally consistent
  - there are no cycles in the partial ordering
  - we can assign variables in such a way that constraints from B hold
- Any linearly ordered sequence of fully instantiated actions from A satisfying < and B goes from s_0 to a state satisfying g.

Hmm, but this definition **does not say how** to verify that a partial plan is a solution plan!

How to efficiently verify that a partial plan is a solution plan?

Claim:

Partial plan Π = (A,<,B,L) is a solution plan if:

- there are no flaws (no open goals and no threats)
- partial ordering < and constraints B are globally consistent

Proof by induction using the plan length

- $\{a_0, a_1, a_\infty\}$ is a solution plan
- for more actions take one of the possible first actions and join it with action a_0

PSP = 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
```

Notes:

- The selection of flaw is deterministic (all flaws must be resolved).
- The resolvent is selected non-deterministically (search in case of failure).

Open goals can be maintained in an **agenda** of action preconditions without causal relations. Adding a causal relation for **p** removes **p** from the agenda.

All threats can be found in time $O(n^3)$ by verifying triples of actions or threats can be maintained incrementally: after adding a new action, check causal relations influenced ($O(n^2)$), after adding a causal relation find its threats ($O(n)$).

Open goals and threats are resolved only by **consistent refinements** of the partial plan.

- consistent ordering can be detected by finding cycles or by maintaining a transitive closure of <
- consistency of constraints in B
  - If there is no negation then we can use arc consistency.
  - In case of negation, the problem of checking global consistency is NP-complete.

Algorithm PSP is **complete and sound**.

- **soundness**
  - If the algorithm finishes, it returns a consistent plan with no flaws so it is a solution plan.
- **completeness**
  - If there is a solution plan then the algorithm has the option to select the right actions to the partial plan.

Be careful about the **deterministic implementation**!

- **The search space is not finite!**
- A complete deterministic procedure must guarantee that it eventually finds a solution plan of any length – **iterative deepening** can be applied.

Initial state:

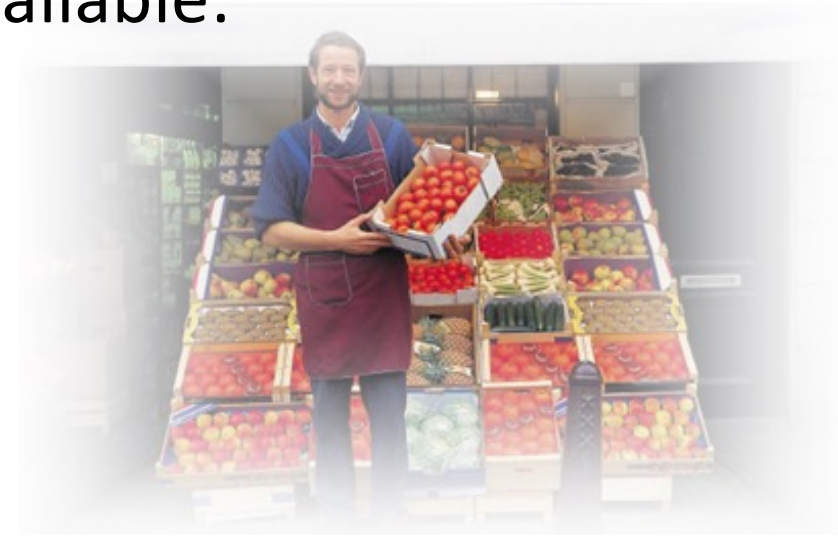
- At(Home), Sells(OBI,Drill), Sells(Tesco,Milk), Sells(Tesco,Banana)
- so action **Start** is defined as:
  Precond: none
  Effects: At(Home), Sells(OBI,Drill), Sells(Tesco,Milk), Sells(Tesco,Banana)

Goal:

- Have(Drill), Have(Milk), Have(Banana), At(Home)
- so action **Finish** is defined as:
  Precond: Have(Drill), Have(Milk), Have(Banana), At(Home)
  Effects: none

The following two **operators** are available:

- **Go(*l*,*m*)** ;; go from location *l* to *m*
  Precond: At(*l*)
  Effects: At(*m*), ¬At(*l*)
- **Buy(*p*,*s*)** ;; buy *p* at location *s*
  Precond: At(*s*), Sells(*s*,*p*)
  Effects: Have(*p*)

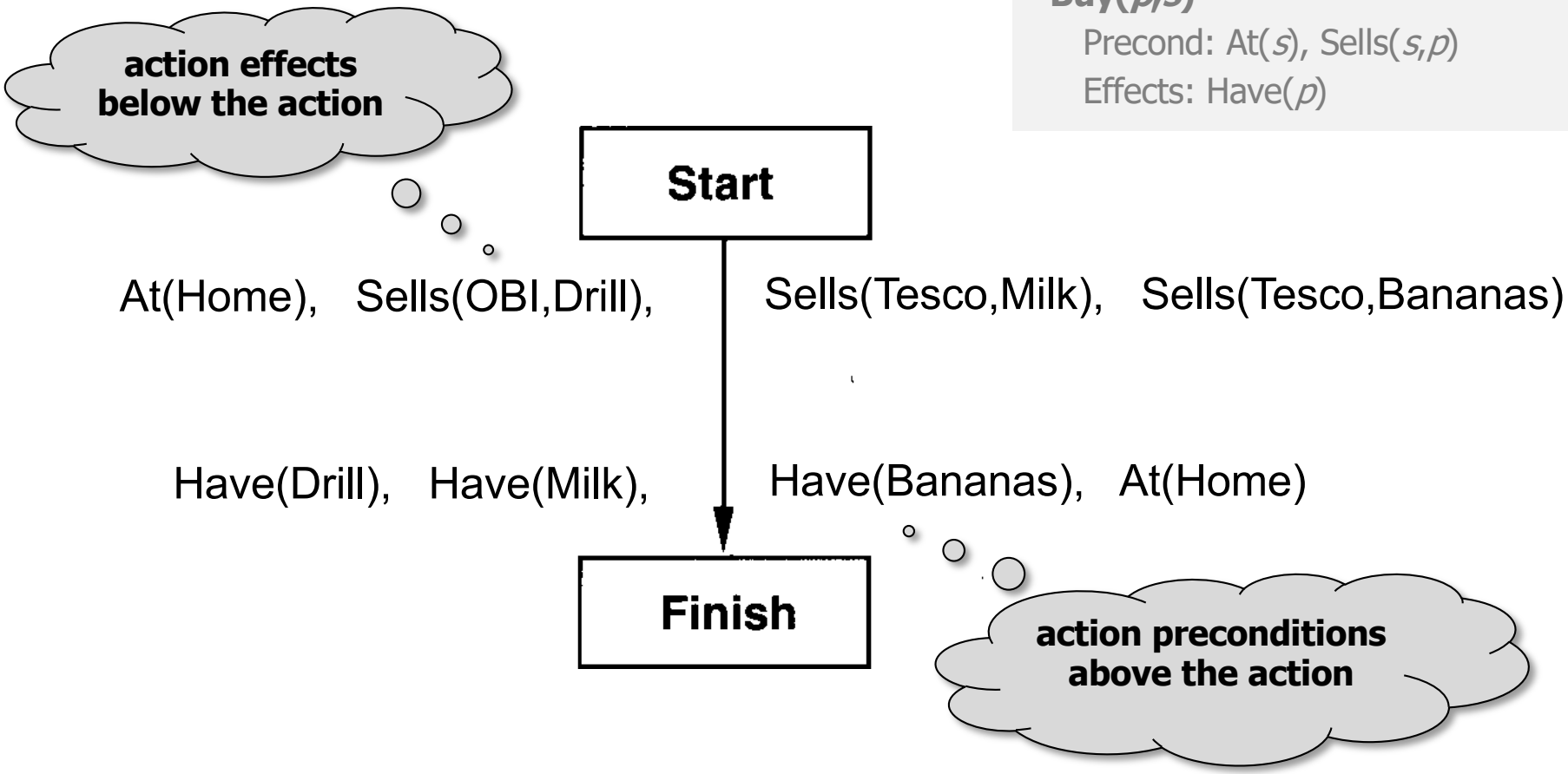


The initial (empty) plan

Operators

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)



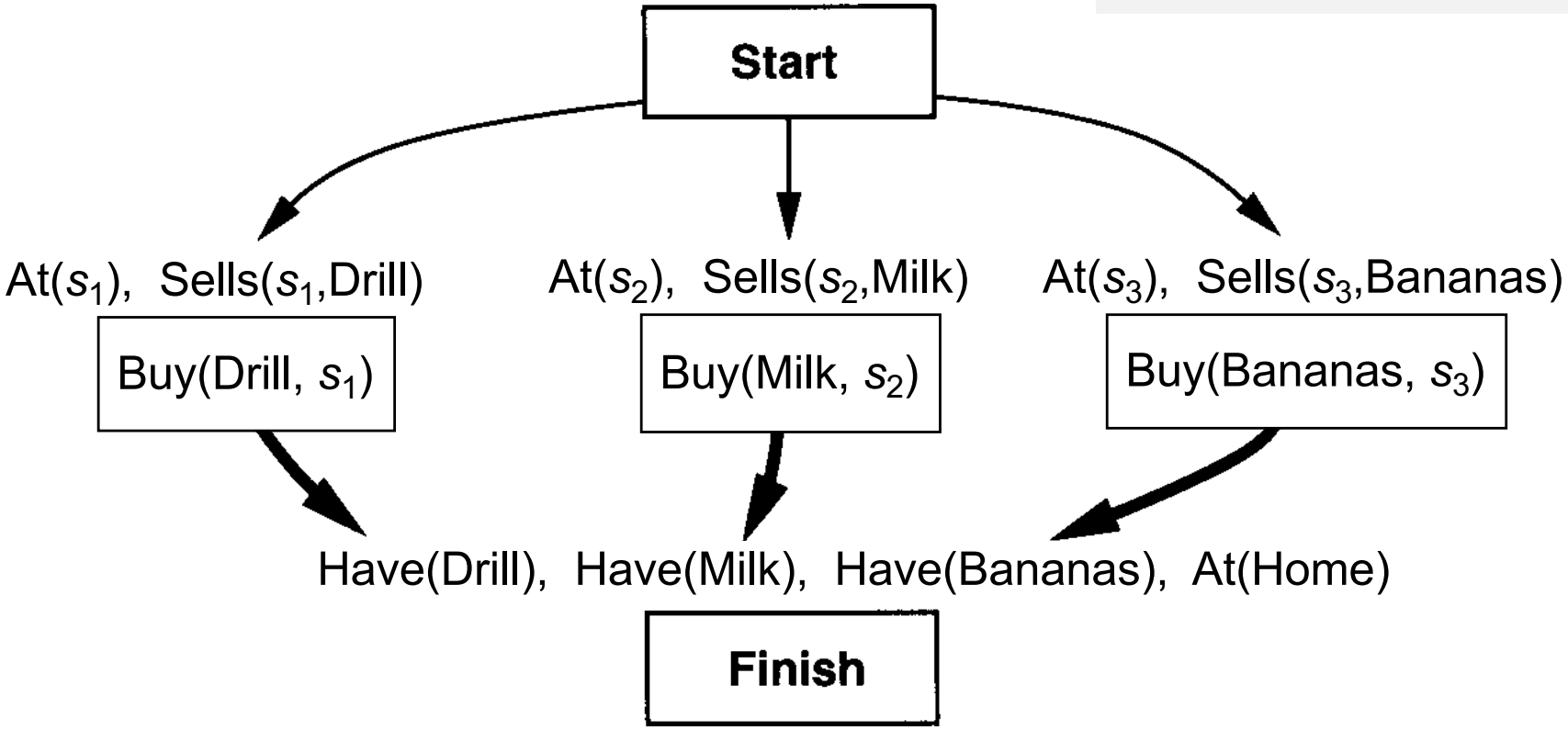
Plan-space planning: a running example

There is only one way to satisfy the **open goals Have**, and this is via **actions Buy** (no threats added).

Operators

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)



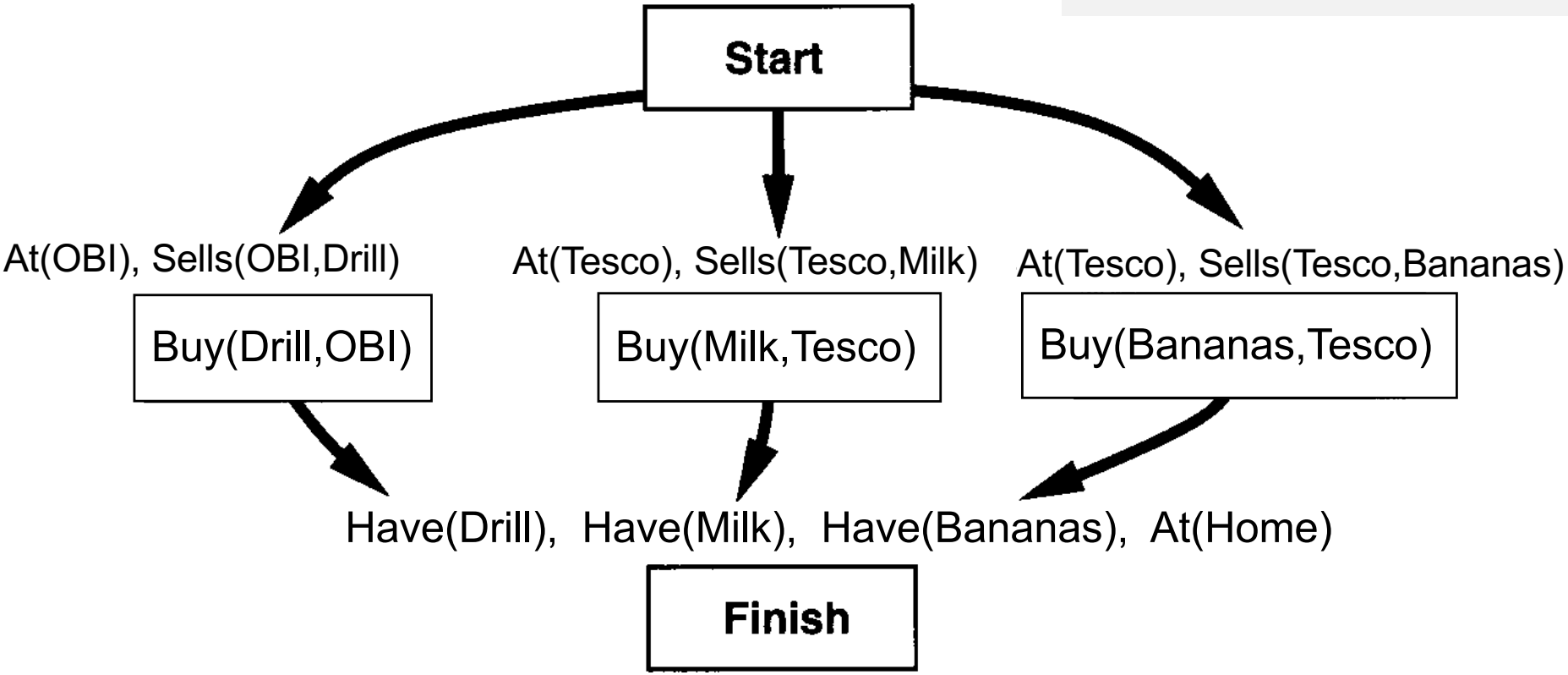
Plan-space planning: a running example

There is again a single way to satisfy preconditions **Sells** and this is substituting the right **constants.**

Operators

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)



Plan-space planning: a running example

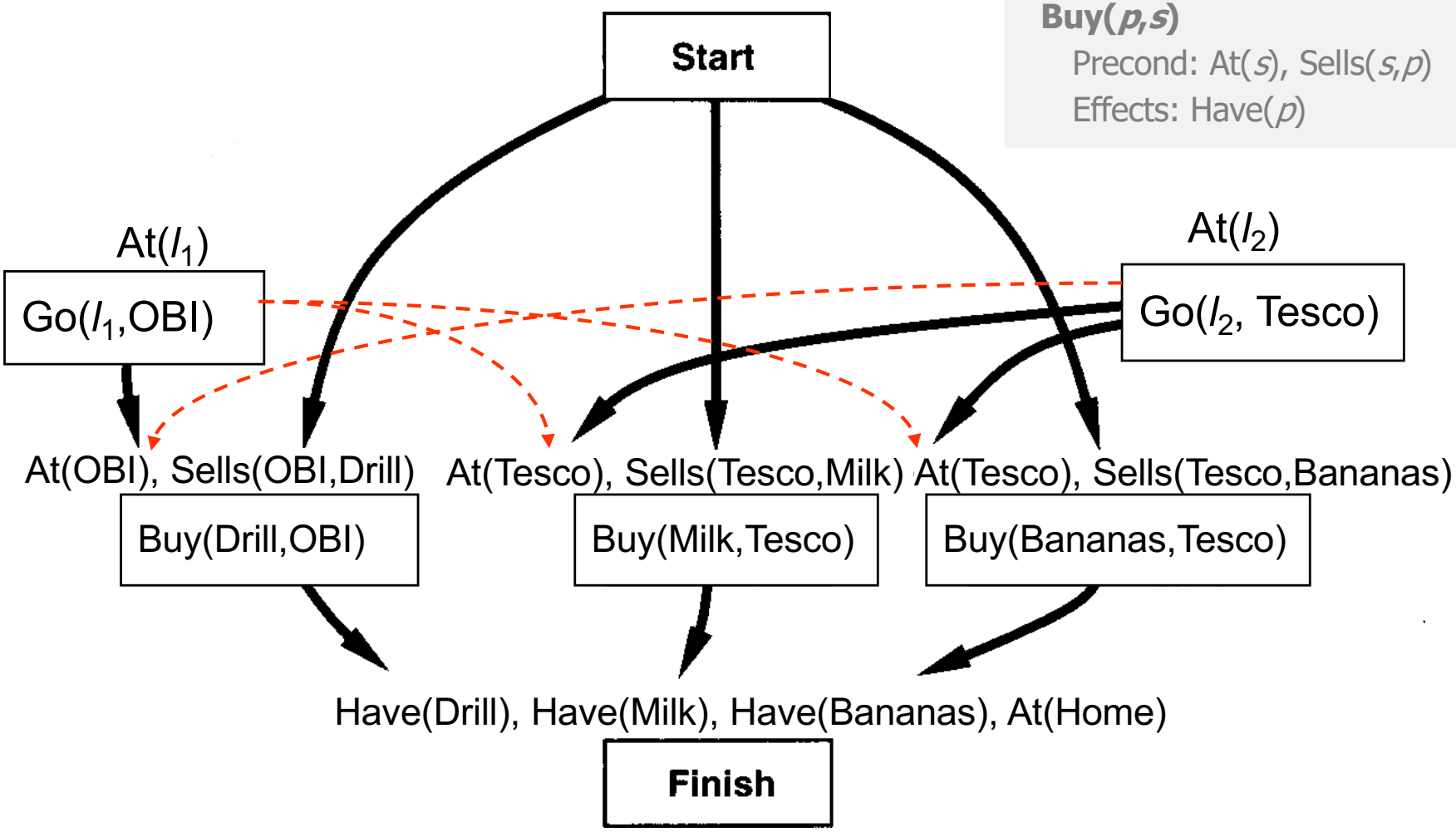
The only way to **satisfy open goals** is by adding actions **Go.**

- There are new threats!

Operators

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)



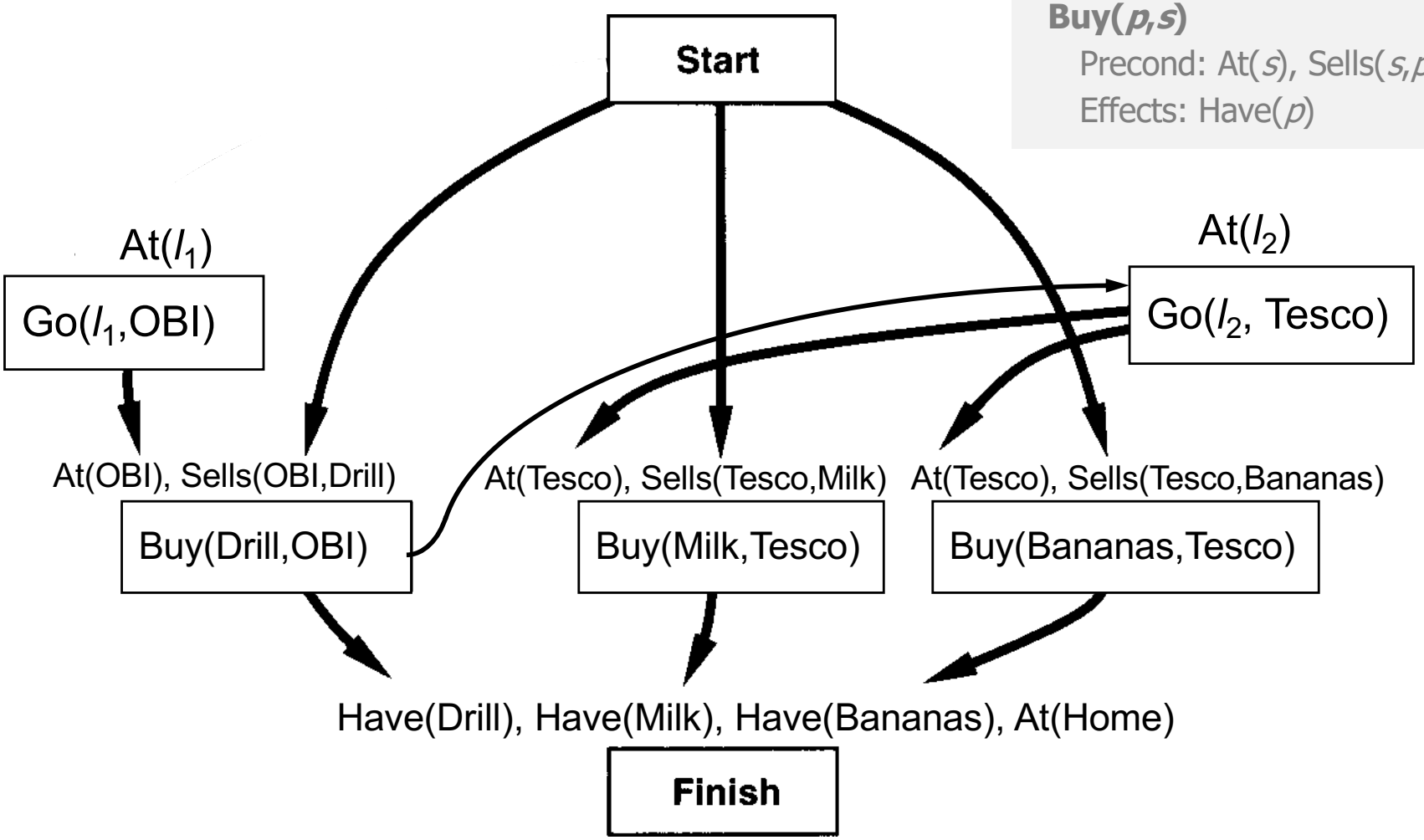
One **threat** can be solved by ordering Buy(Drill) before Go(Tesco)

- This solves all the threats!

Operators

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)

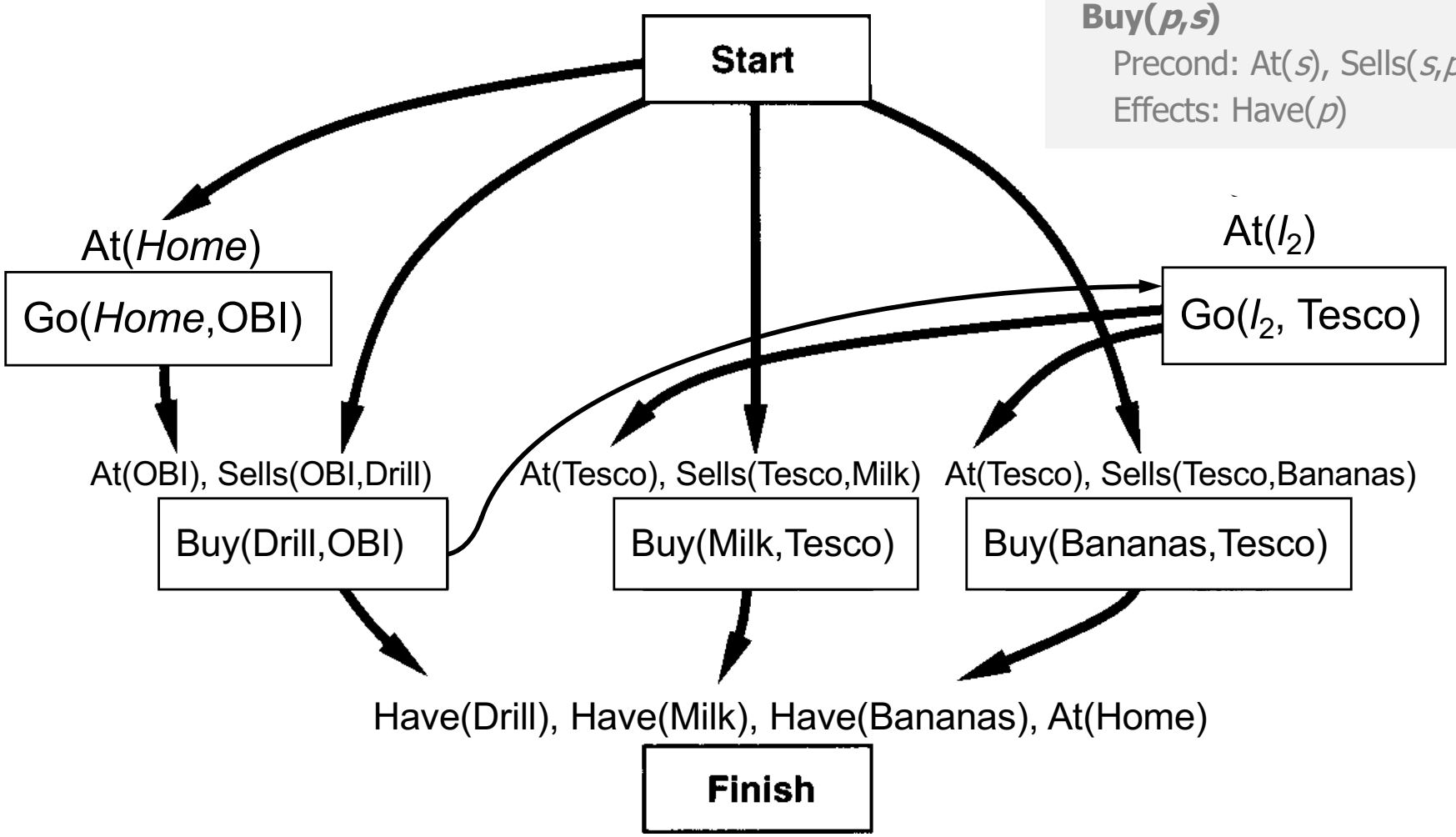


Open goal At(l_1) can be satisfied by **assignment** l_1=Home taken from the action Start.

Operators

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)

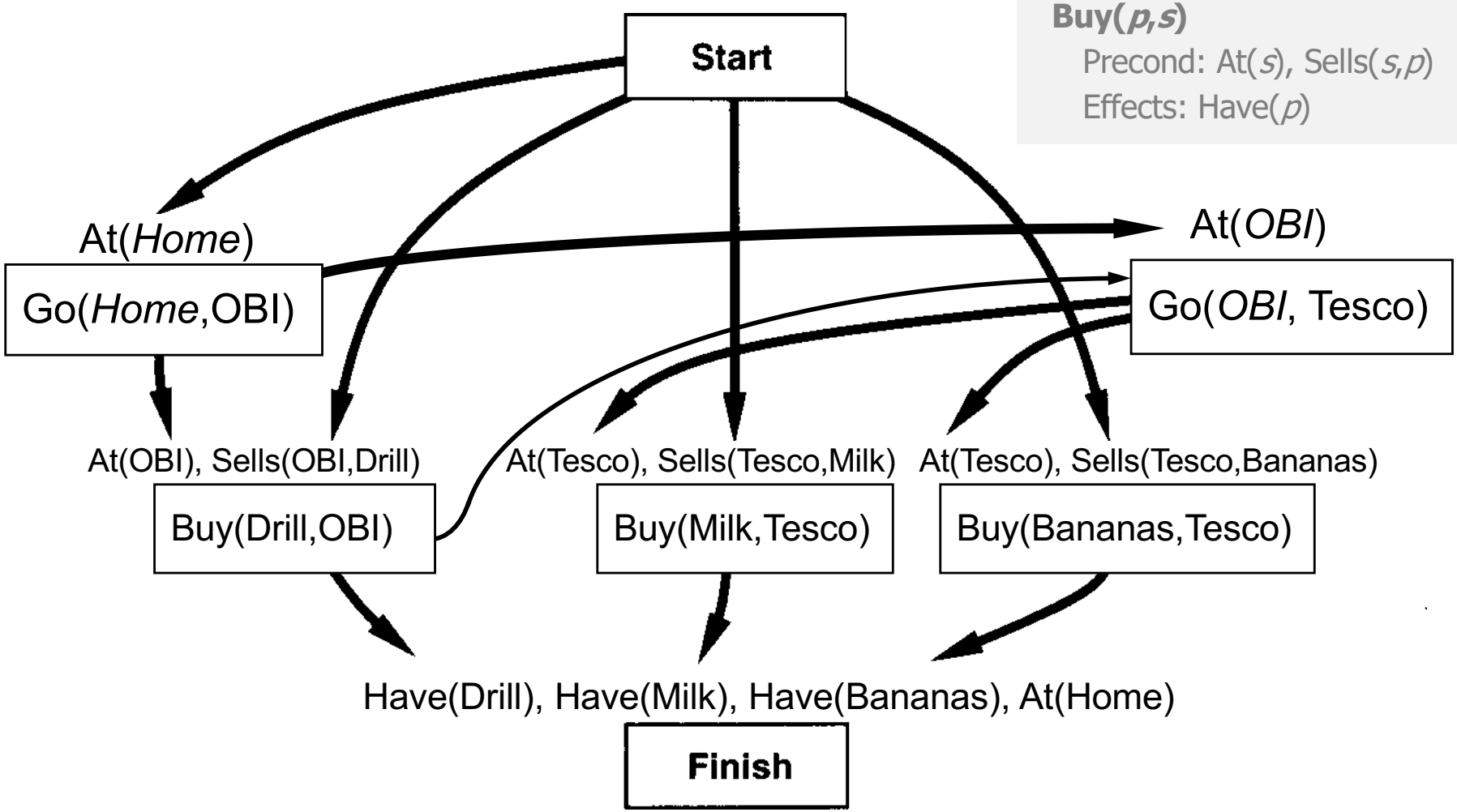


Open goal At(l_2) can be satisfied by **assignment** l_2=OBI from action Go(Home, OBI)

Operators

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)



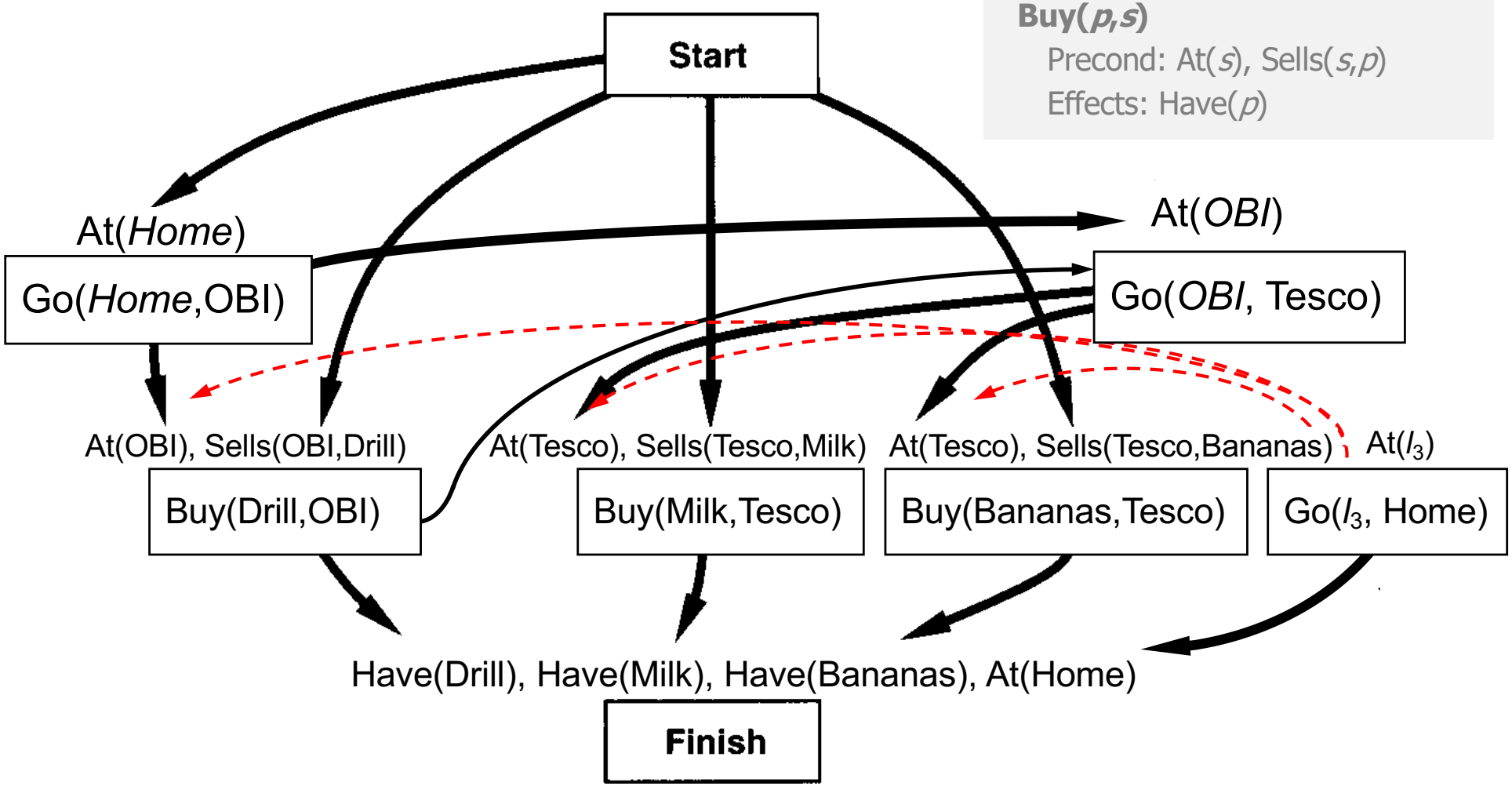
Open goal At(Home) from Finish is satisfied by **action** Go

– new threats appear

Operators

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)

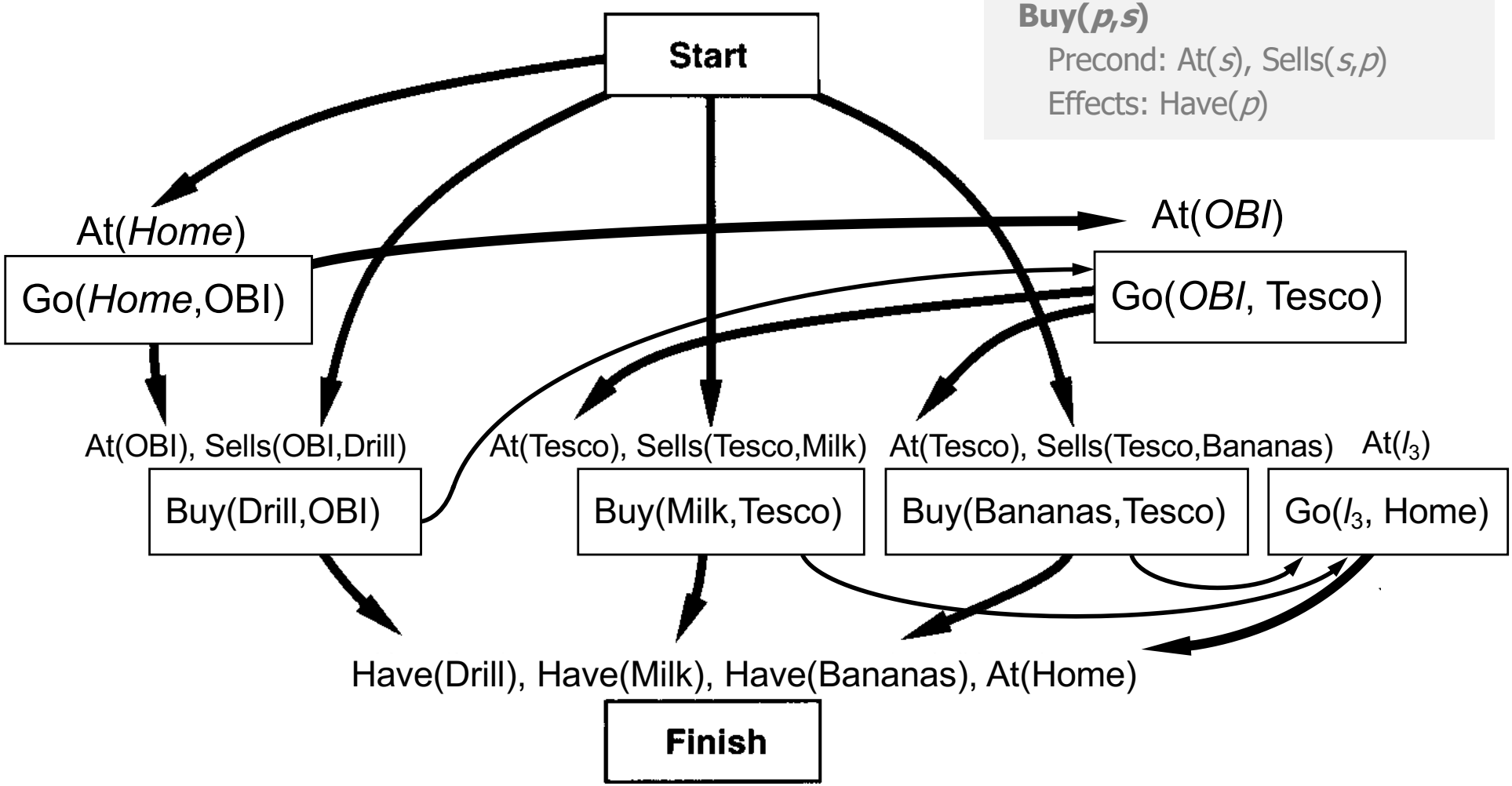


Threats for At(Tesco) are removed by **ordering** Go(Home) after both actions Buy

Operators

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)

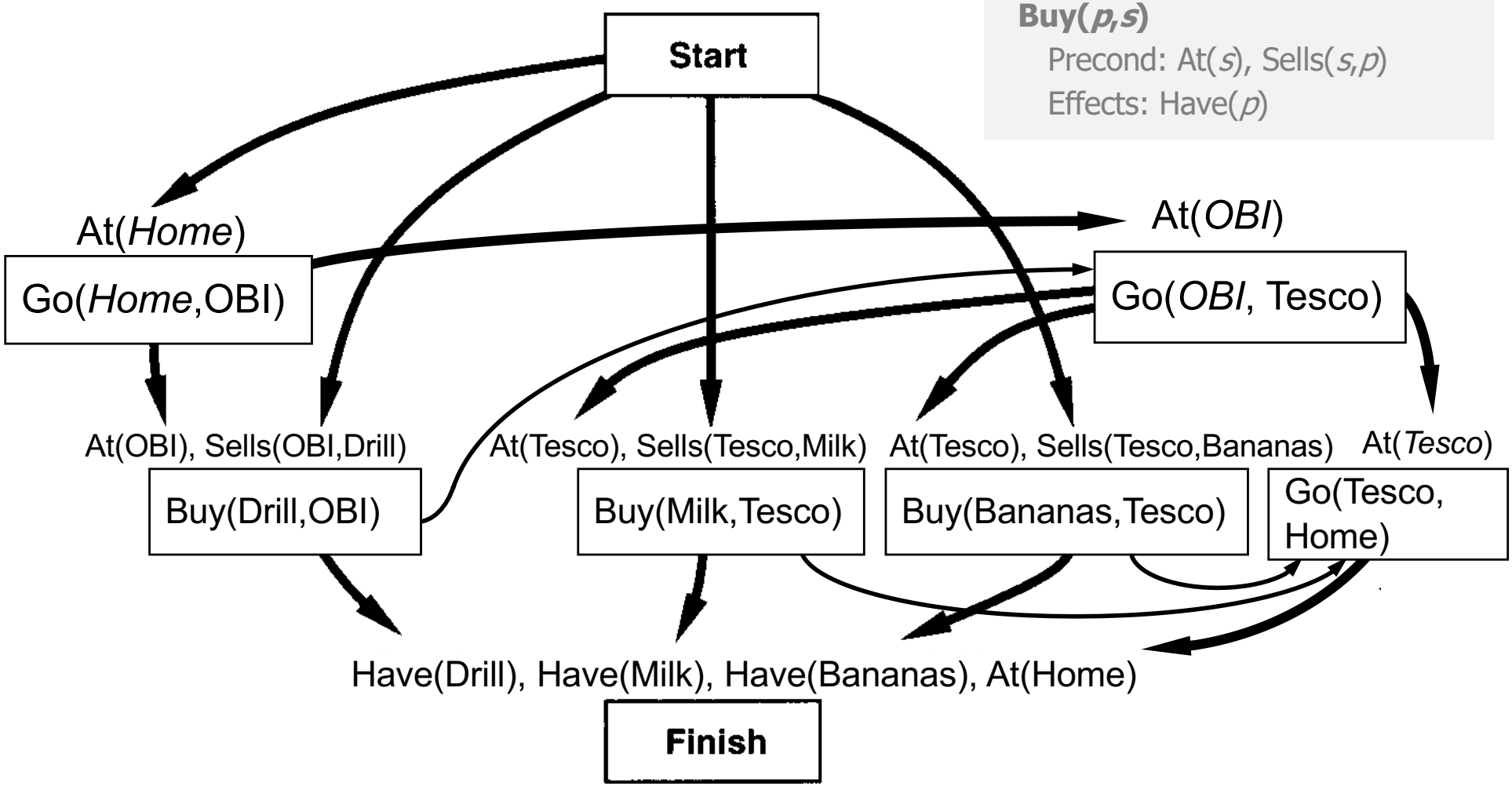


Open goal At(l_3) is satisfied by **asignment** l_3=Tesco from action Go(OBI,Tesco).

Operators

Go(*l*,*m*)
Precond: At(*l*)
Effects: At(*m*), ¬At(*l*)

Buy(*p*,*s*)
Precond: At(*s*), Sells(*s*,*p*)
Effects: Have(*p*)



	State space planning	Plan space planning
search space	finite	infinite
search nodes	simple (world states)	complex (partial plans)
world states	explicit	not used
partial plan	action selection and ordering done together	action selection and ordering separated
plan structure	linear	causal relations

State space planning is much faster today thanks to heuristics based on state evaluation.

However, **plan space planning**:

- makes more **flexible plans** thanks to partial order
- supports **further extensions** such as adding explicit time and resources

Charles University, Prague, Czech Republic
bartak@ktiml.mff.cuni.cz