Planning & Scheduling

Roman Barták
Department of Theoretical Computer Science and Mathematical Logic

Plan Space Planning

Just to recall

Planning is based on search.

- **State space planning**
  - search nodes correspond to world states
  - we can search either forward or backward

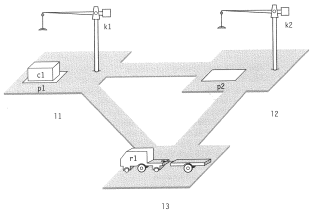
  Problems:
  - **large branching factor**
    - lifting helps to decrease the branching factor by delaying instantiation of variables
  - **alternative action orders not leading to a goal**
    - We do not need to force the order of action until we really need it due to causal relations
  - **least-commitment strategy**
    - "what can be done tomorrow should be done tomorrow"

Plan space planning: core idea

- 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

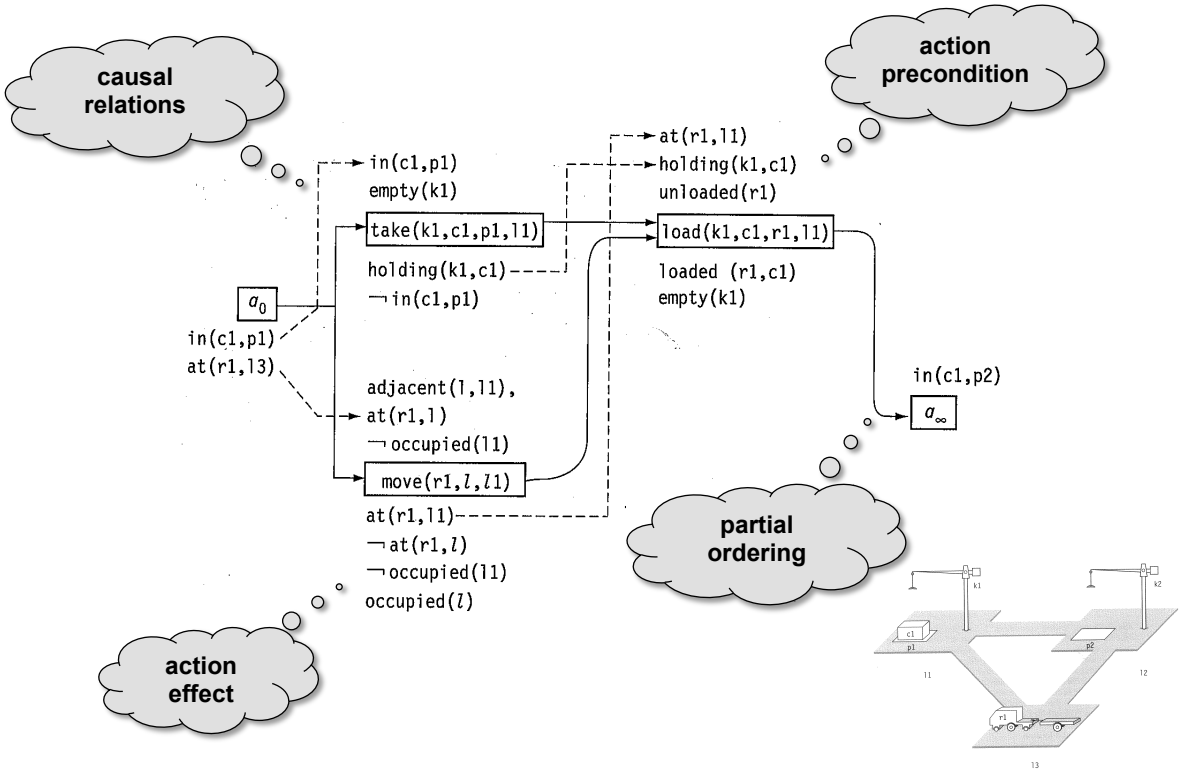
- **The initial state and the goal** are encoded using two **special actions** in the initial partial plan:
  - **Action a_0 represents the initial state** in such a way that atoms from the initial state define effects of the action and there are no preconditions. This action will be before all other 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

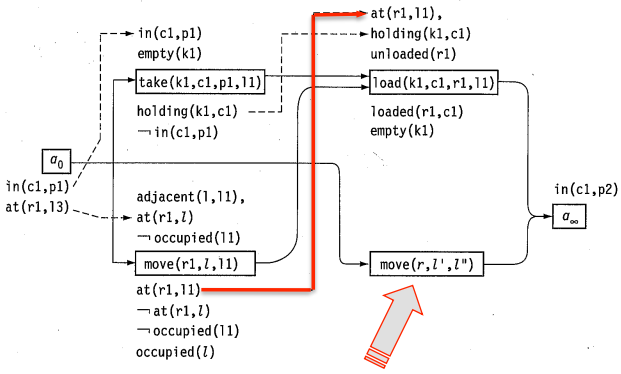


Open goals

- **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 $\mathbf{a} \rightarrow^{\mathbf{p}} \mathbf{b}$

Threats

- **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 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**



Solution plan

- 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).

PSP – some details

- **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.

Properties of PSP

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.

Algorithm PoP

PoP is a popular instance of algorithm PSP.

```
PoP(π, agenda)          ;; where π = (A, ≺, B, L)
    if agenda = ∅ then return(π)
    select any pair (a_j, p) in and remove it from agenda
    relevant ← Providers(p, π)
    if relevant = ∅ then return(failure)
    nondeterministically choose an action a_i ∈ relevant
    L ← L ∪ {⟨a_i —p→ a_j⟩}
    update B with the binding constraints of this causal link
    if a_i is a new action in A then do:
        update A with a_i
        update ≺ with (a_i ≺ a_j), (a_0 ≺ a_i ≺ a_∞)
        update agenda with all preconditions of a_i
    for each threat on ⟨a_i —p→ a_j⟩ or due to a_i do:
        resolvers ← set of resolvers for this threat
        if resolvers = ∅ then return(failure)
        nondeterministically choose a resolver in resolvers
        add that resolver to ≺ or to B
    return(PoP(π, agenda))
end
```

- **Agenda** is a set of pairs (**a**,**p**), where **p** is an open precondition of action **a**.
- **First find an action** a_i to cover some **p** from the agenda.
- **At the second stage resolve all threats** that appeared by adding action a_i or from a causal relation with a_i.

Plan-space planning: a running example

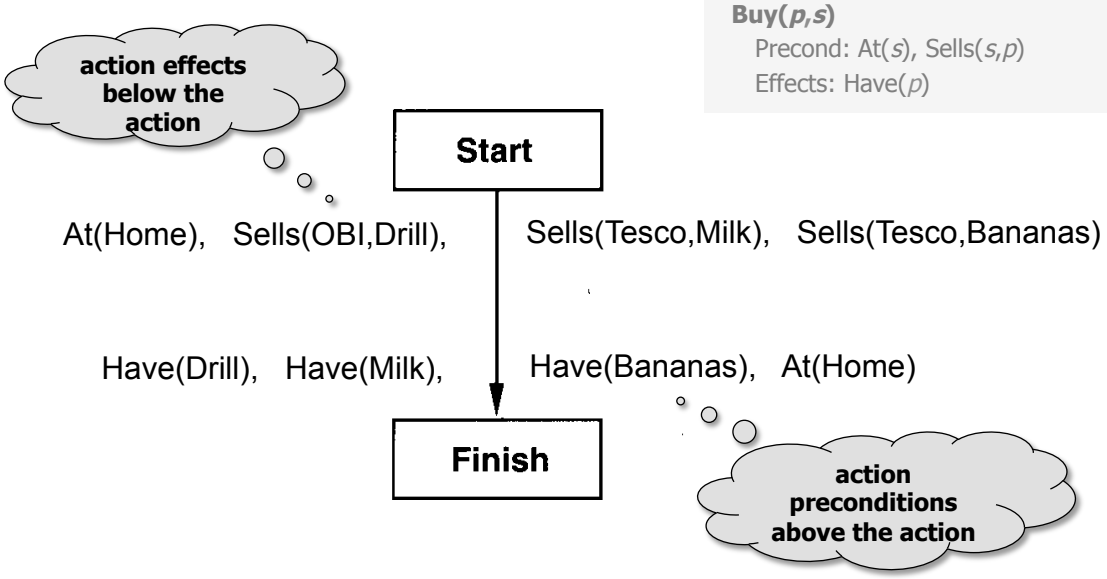
- **Initial state:**
  - At(Home), Sells(OBI,Drill), Sells(Tesco,Milk), Sells(Tesco,Banana)
  - so action **Start** is defined as:
    Precond: none
    Effects: At(Home), Sells(OBI,Drill), Sells(Tesco,Milk), Sells(Tesco,Banana)
- **Goal:**
  - Have(Drill), Have(Milk), Have(Banana), At(Home)
  - so action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



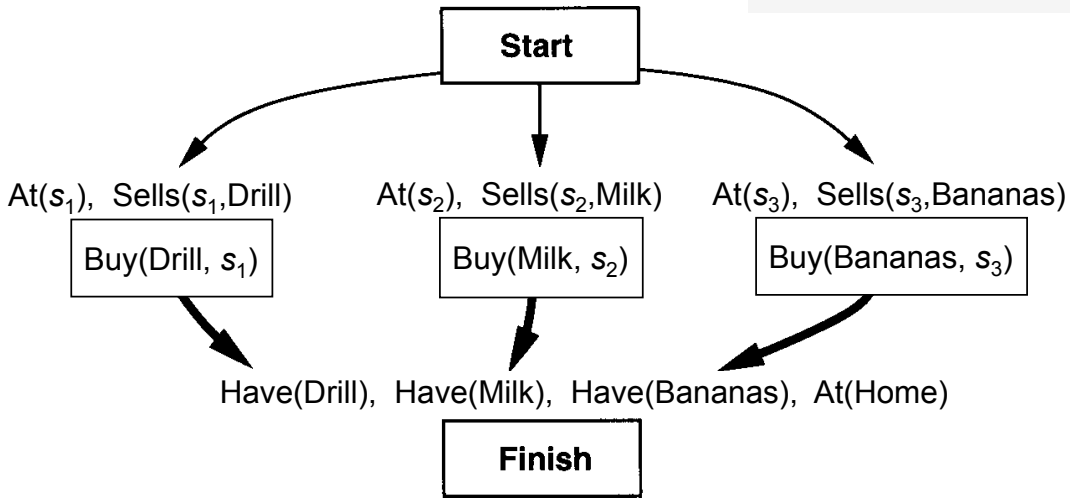
Plan-space planning: a running example

- 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*)



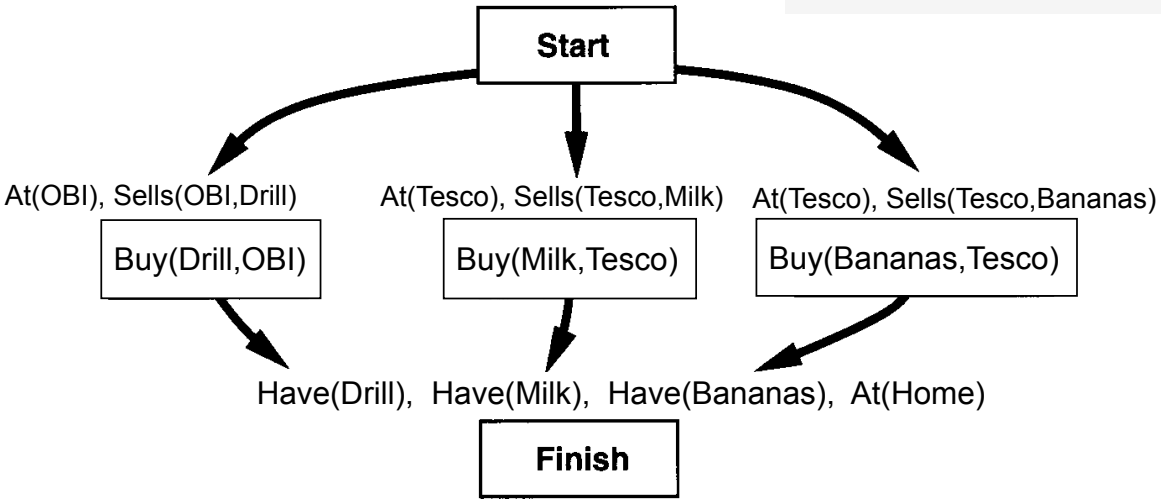
Plan-space planning: a running example

- 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*)



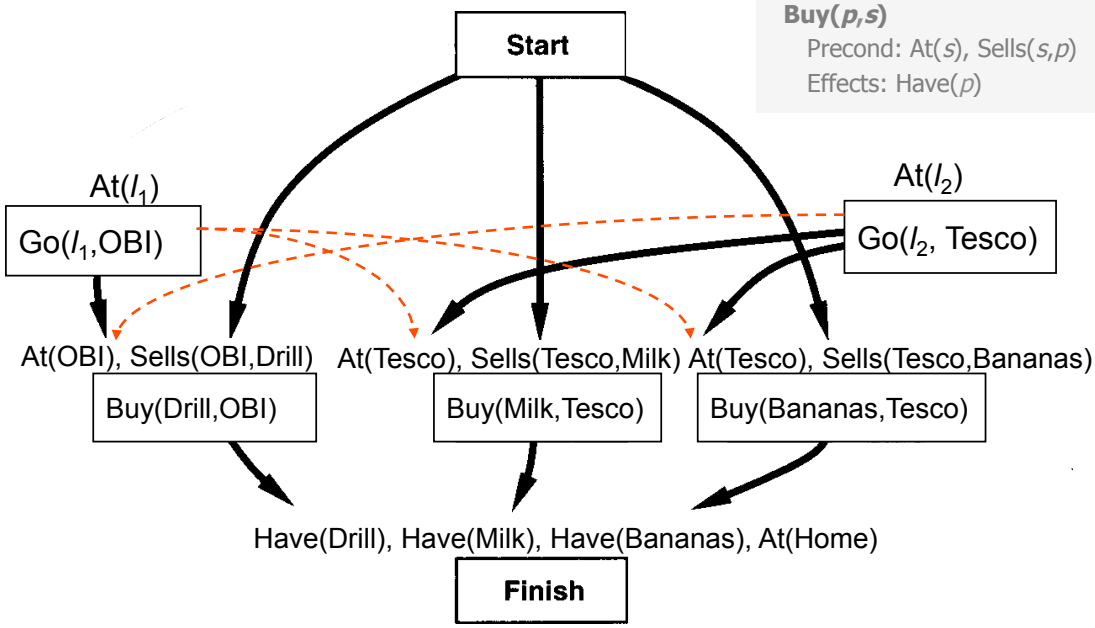
Plan-space planning: a running example

- 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*)



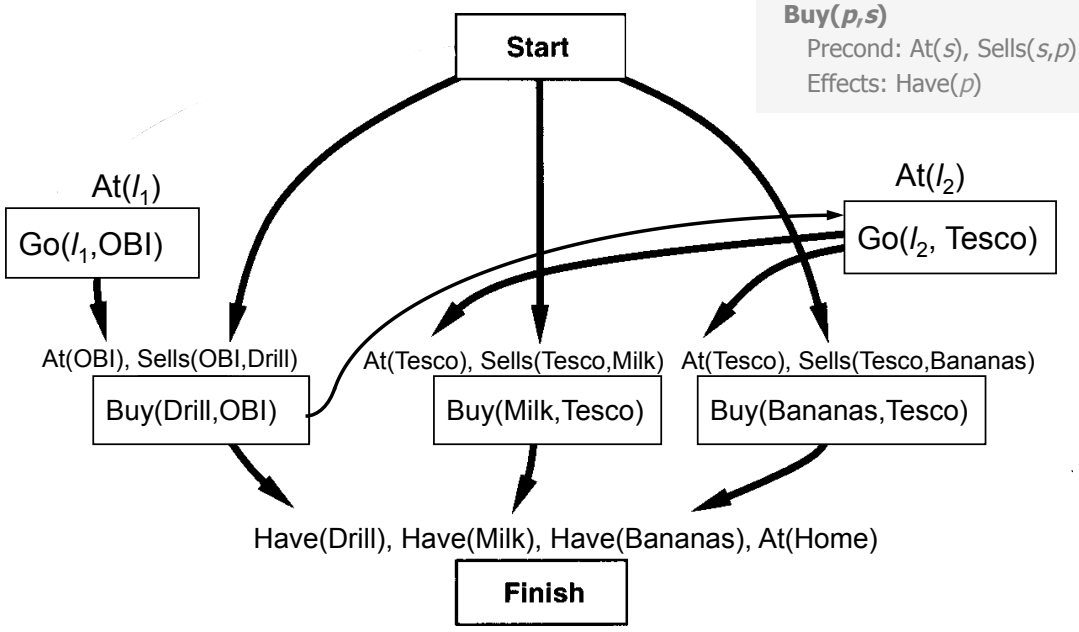
Plan-space planning: a running example

- 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*)



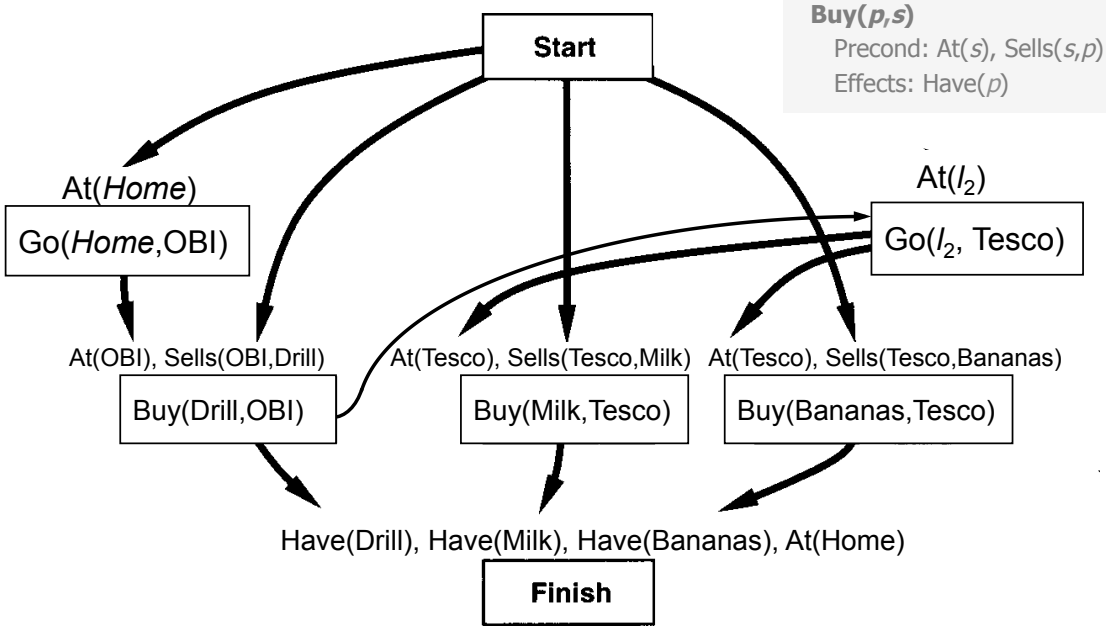
Plan-space planning: a running example

- **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*)



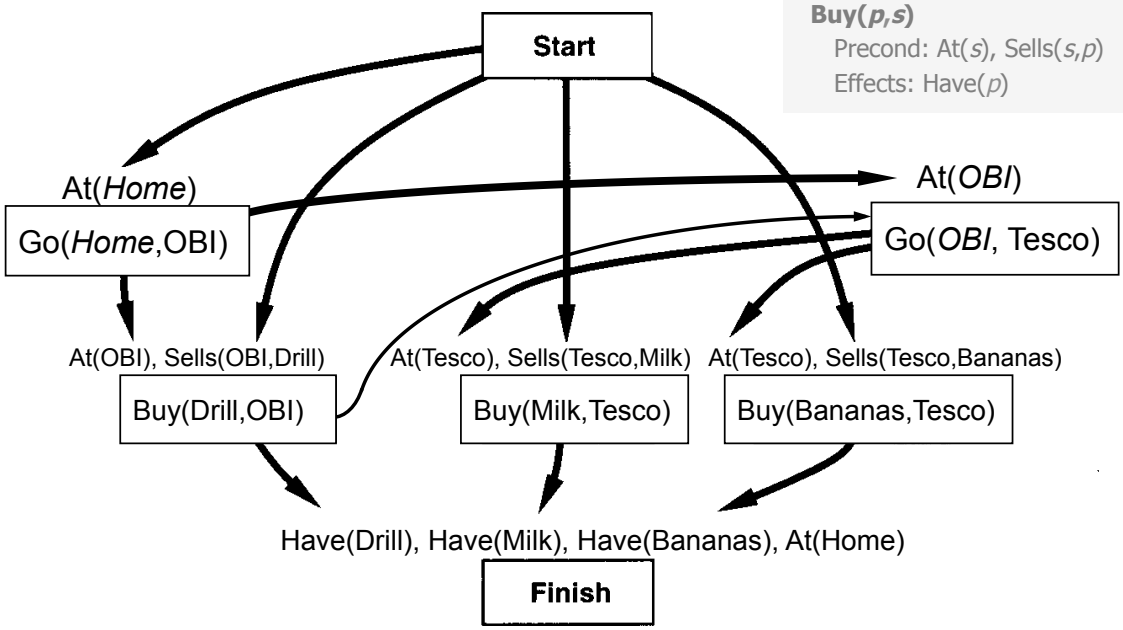
Plan-space planning: a running example

- **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*)



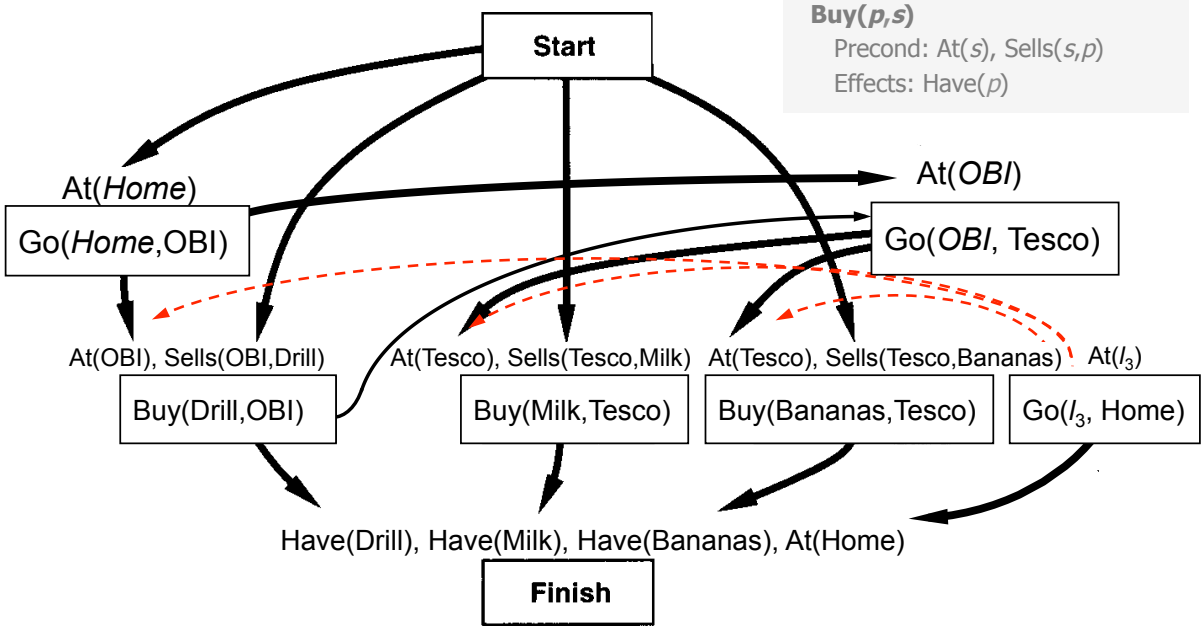
Plan-space planning: a running example

- **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*)



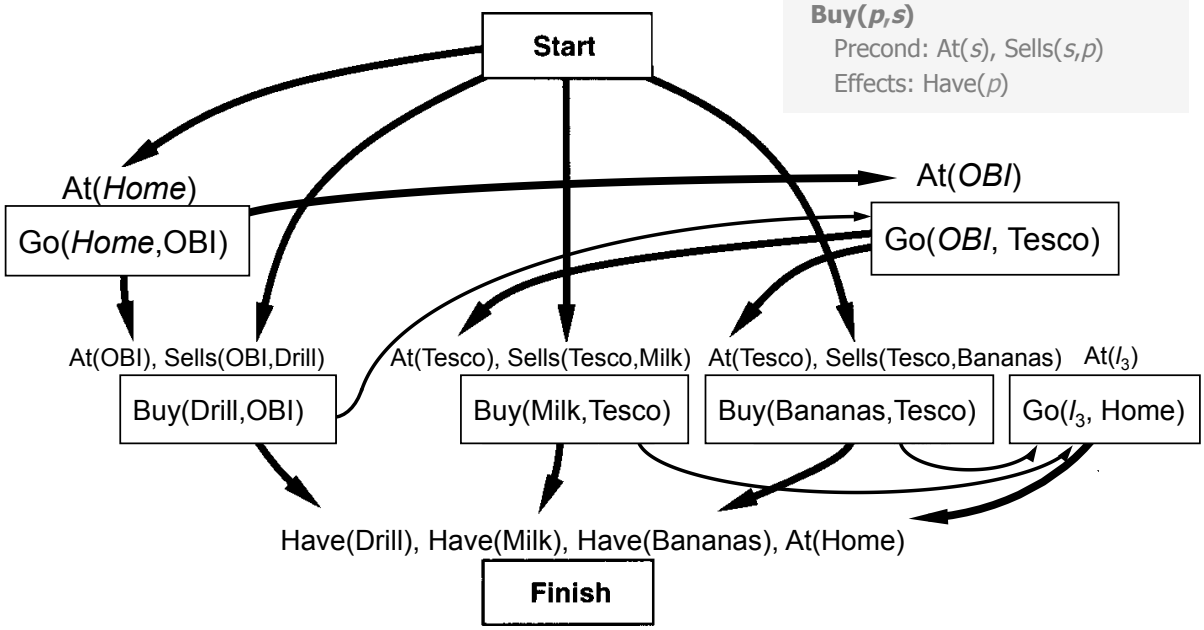
Plan-space planning: a running example

- **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*)



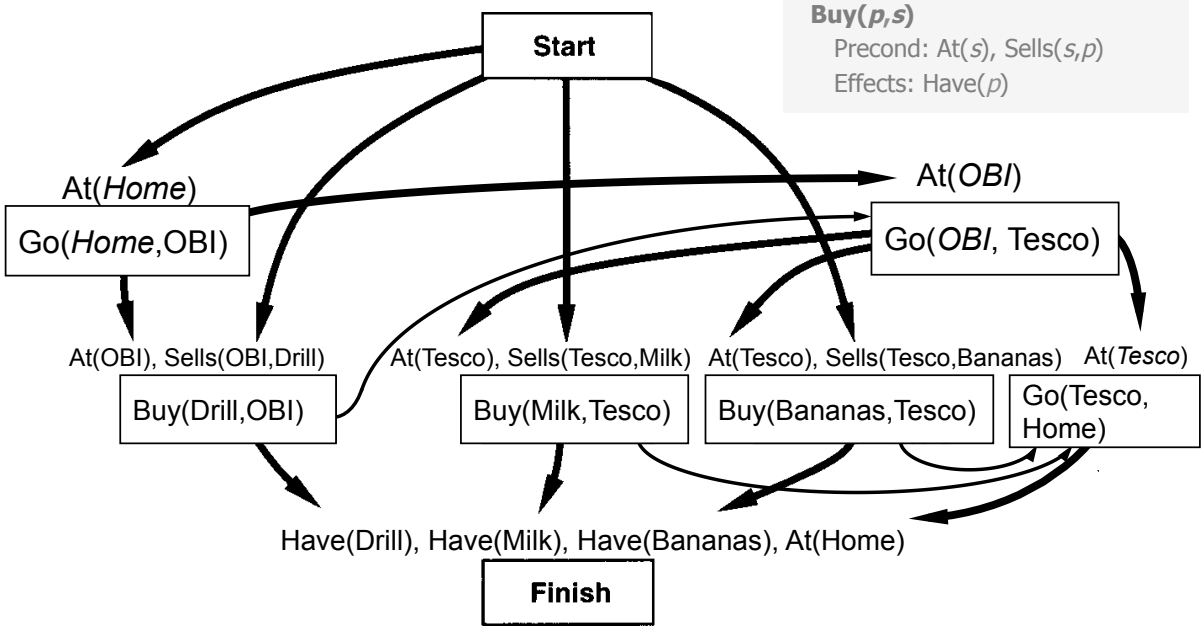
Plan-space planning: a running example

- **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

Department of Theoretical Computer Science and Mathematical Logic
bartak@ktiml.mff.cuni.cz