# Classical Planning

CE417: Introduction to Artificial Intelligence
Sharif University of Technology
Spring 2016

Soleymani

AIMA, 3rd Edition, Chapter 10 & more about planning

# What is planning?

- Planning problem: finding a sequence of actions that leads to a goal state starting from any of the initial states
  - Solution (obtained sequence of actions) is **optimal** if it minimizes sum of action costs
  - Search-based problem solving agents as a special case of planning agents
    - $Result(s, a)$ as a black-box function and states are also black-boxes that are either goal or not goal.

# General planning problem

- Environment can be
  - dynamic, nondeterministic, partially observable, continuous, multi-agent

- Actions may
  - take time (have durations)
  - have continuous effects
  - be taken concurrently

- Initial state may be arbitrarily many and goal may be optimizing an objective function

# Classical planning

- We focus on classical planning
  - Environment: deterministic, static, fully observable, discrete, single agent
  - Actions: duration-less, taken only one at a time
  - Initial state: a unique known one
  - Goal state: specified goal states

- Importance
  - Most of the recent progress are based on classical planning
  - Provides also useful idea for more complex problems

# Applications

- Robotics
- Spacecraft and Mars rover mission controls
- Transportation of cargos, peoples, …
- Interactive decision making
  - Military operations
  - Astronomic observations

# Why planning?

- Planning as a form of general problem solving
  - **Idea:** problems described at high-level and solved automatically

- Representation of planning problems
  - Scaling up to larger problems
  - Deriving domain independent heuristics automatically

# Representation has a key role

- A state is represented more clear than atomic (black-box) ones.

- Actions for state-transition are represented in a concise and declarative manner.

- This type of representation can be used to solve problem effectively.

# Representation of states and actions

- Representation of **states** (logic, set theory, …)
  - Conjunction of ground, functionless, and positive literals
    - Closed world assumption: any fluent that are not mentioned are false

- Representation of **actions** (logic, set theory, …)
  - Specifying the result of an action in terms of what changes
    - e.g., described by sets of preconditions and effects (post-conditions)

# Representing actions

- Actions are described in terms of preconditions and effects.
  - Preconditions are predicates that must be true **before** applying the action
  - Effects are predicates that are made true (or false) **after** executing the action

# Representation language

- Concise description

- PDDL (Planning Domain Definition Language)
  - States, actions and goals are described in the language of <u>symbolic logic</u>
    - Predicates denote particular features of the world.
    - Does not allow quantifiers and functions

- Other languages: STRIPS, ADL

# PDDL description of a planning problem

- Initial state
  - Conjunction of ground atoms
- Goal states
  - Conjunction of literals (positive or negative) that may contain variables
    - Variables are treated as existentially quantified
- Actions
  - Action schema (lifted representation)
    - Action name
    - List of variables
    - Precondition
    - Effect

# Example: Air cargo transfer

- Domain
  - Objects:
    - airports $(SFO, JFK, \ldots)$, cargos $(C_1, C_2, \ldots)$, airplanes $(P_1, P_2, \ldots)$
  - Predicates:
    - $At(p, a), In(c, p), Plane(p), Cargo(c), Airport(a)$

- States:
  - Planes and cargos are at specific airports.

- Actions:
  - $Load\ (cargo, plane, airport)$
  - $Fly\ (plane, airport_1, airport_2)$
  - $Unload\ (cargo, plane, airport)$

# Example: Air cargo transfer (actions in PDDL)

$Action(Load(c, p, a),$

PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a),$

EFFECT: $\neg At(c, a) \land In(c, p))$

$Action(Unload(c, p, a),$

PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a),$

EFFECT: $At(c, a) \land \neg In(c, p))$

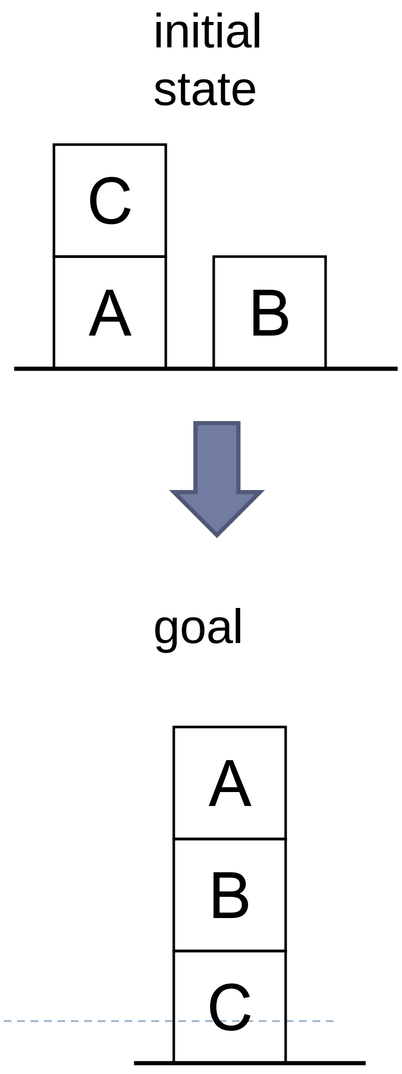
$Action(Fly(p, from, to),$

PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to),$

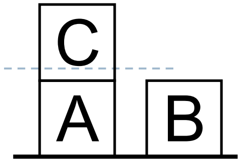
EFFECT: $\neg At(p, from) \land At(p, to))$

# Example: Blocks World

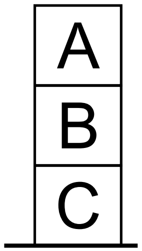
- Domain
  - Objects:
    - A set of blocks $(A, B, C, \ldots)$ and a table $(Table)$.
  - Predicates:
    - $On(b, x), Clear(b), Block(b)$
- States:
  - Blocks are stacked on other blocks and the table.
- Actions (here, two actions):
  - Move from a tower or table to another tower
  - Move to table



# Example: Blocks World




- **Initial state**
  - $On(A, Table) \land On(B, Table) \land \ On(C, A)$
    $\land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C)$

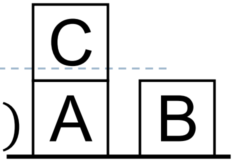



- **Goal state**
  - $On(A, B) \land On(B, C)$

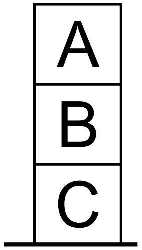
- **Actions**
  - $Move(b, x, y)$
    - PRECOND: $On(b, x) \land Clear(b) \land Clear(y) \land Block(b)$
      $\land Block(y) \ \land (b \neq x) \ \land \ (b \neq y) \ \land \ (x \neq y)$
    - EFFECT: $On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y)$

  - $MoveToTable(b, x)$
    - PRECOND: $On(b, x) \land Clear(b) \land Block(b) \land (b \neq x)$
    - EFFECT: $On(b, Table) \land Clear(x) \land \neg On(b, x)$

# Example: Blocks World in PDDL




$Init(On(A, Table) \land On(B, Table) \land\ On(C, A) \land Block(A)\ \land Block(B)$
$\quad \land Block(C) \land Clear(B) \land Clear(C))$




$Goal\big(On(A, B) \land On(B, C)\big)$

$Action(Move(b, x, y),$

PRECOND: $On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y)$
$\land (b \neq x)\ \land\ (b \neq y)\ \land\ (x \neq y),$

EFFECT: $On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y))$

$Action(MoveToTable(b, x),$

PRECOND: $On(b, x) \land Clear(b) \land Block(b) \land (b \neq x),$

EFFECT: $On(b, Table) \land Clear(x) \land \neg On(b, x))$

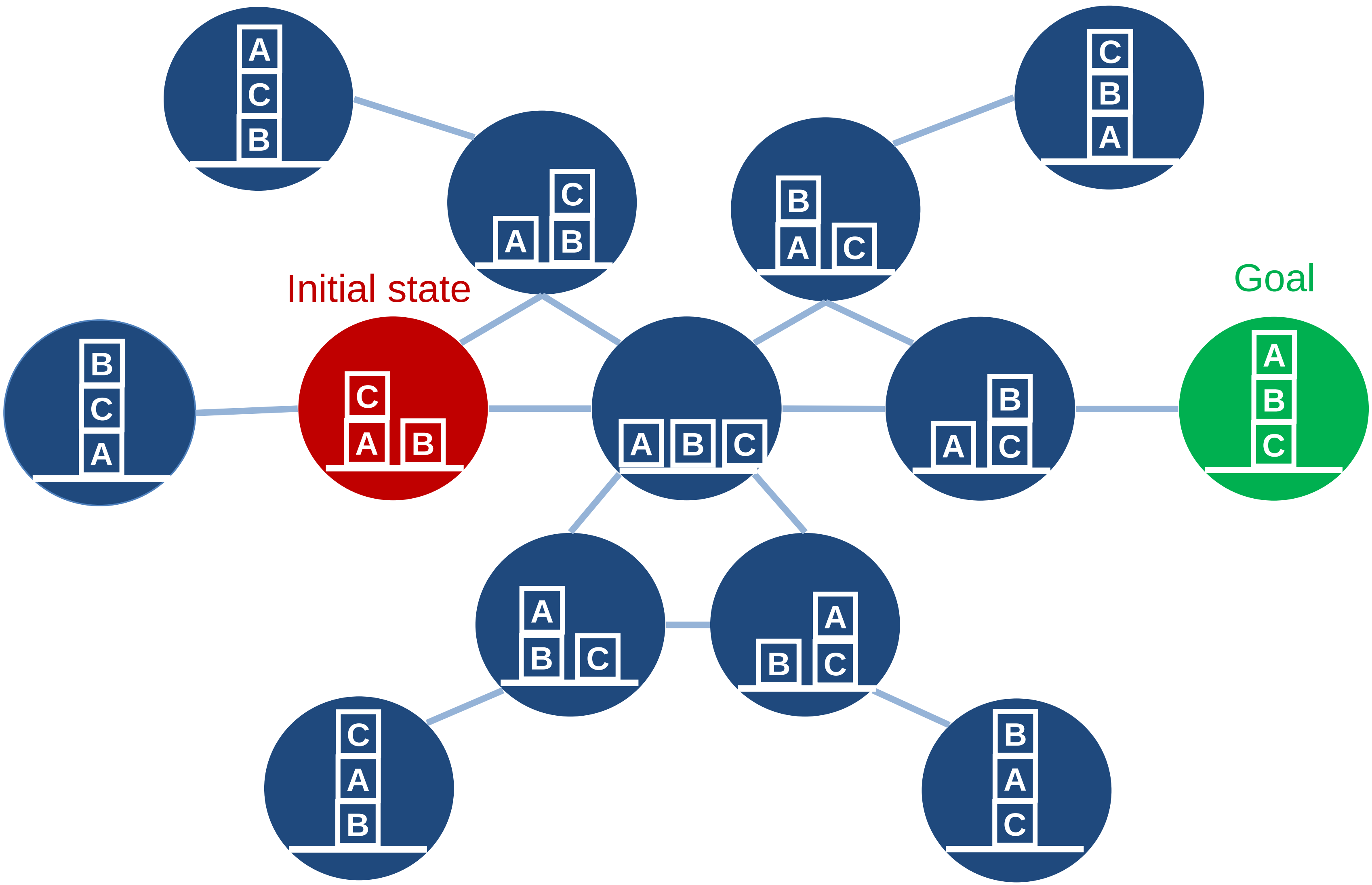
# Planning problem & approaches

- Planning problem:
  - Given: start state, goal conditions, actions
  - Aim: finding a sequence of actions leading from start to goal

- Some of the most popular approaches to solve it:
  - Forward state-space search (+ heuristics)
    - e.g., Fast-Forward (FF)
  - Backward state-space search (+ constraints)
    - e.g., GraphPlan
  - Reduction to propositional satisfiability problem
    - SATPlan
  - Search in the space of plans
    - Partial Order Planning (POP)

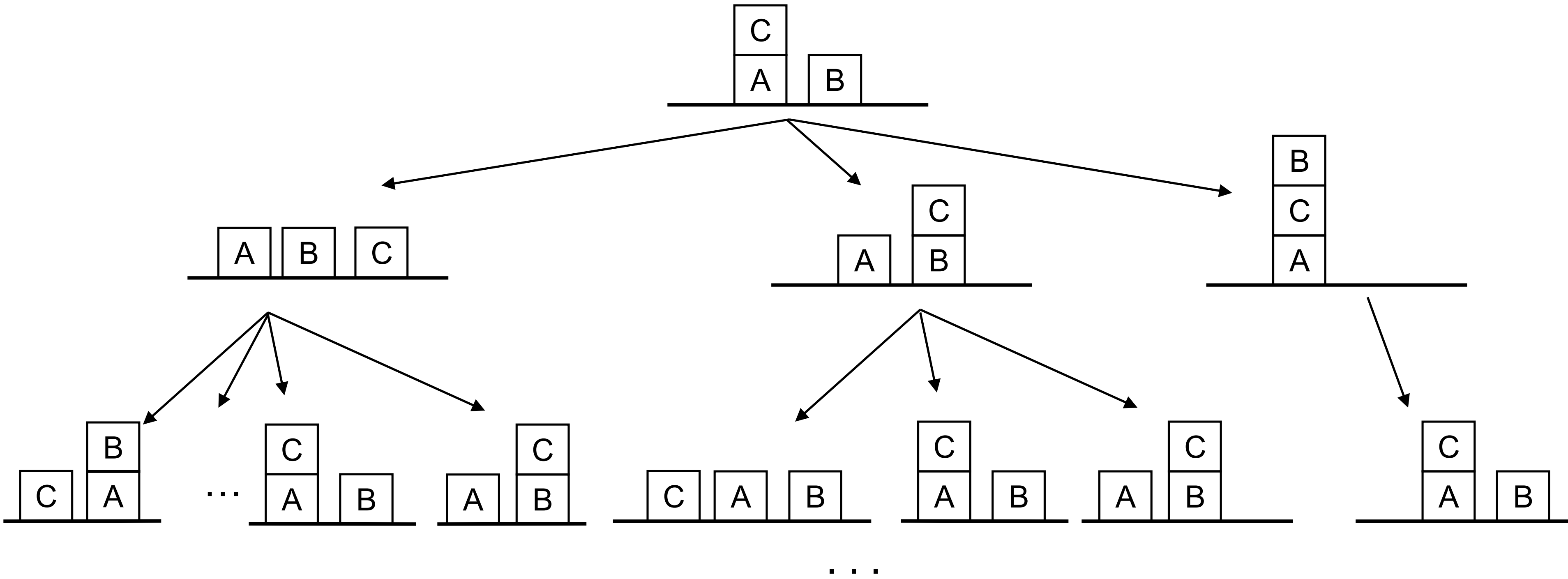
# Planning as a state-space search

- States
- Actions
- Path Cost
- Goal Test


Initial state
Goal

# Forward state-space search

- Progression: Starting from initial state and using actions to reach a goal state.



# Forward state-space search (Cont.)

- An action $a$ is **applicable** in state $s$ if its precondition is satisfied:

$$\left.\begin{aligned} PRE^{+}(a) &\subseteq s \\ PRE^{-}(a) \cap s &= \varnothing \end{aligned}\right\} \Leftrightarrow (a \in \mathrm{ACTIONS}(s))$$

- The state $s'$ resulted when executing $a$ in $s$ is given by (**progressing** $s$ through $a$):

$$s' = (s - \mathrm{DEL}(a)) \cup \mathrm{ADD}(a)$$

$$\mathrm{RESULTS}(s, a) = \left(s - \mathrm{DEL}(a)\right) \cup \mathrm{ADD}(a)$$

# Forward Search

Forward-Search($s$, $g$)

**if** $s$ satisfies $g$ **then return** []

*applicable* = $\{a \mid a \text{ is applicable in } s\}$

**if** *applicable* = $\varnothing$ **then return** failure

**for each** $a \in$ *applicable* **do**

$s' = (s - \mathrm{DEL}(a)) \cup \mathrm{ADD}(a)$

$\pi'$ = Forward-Search($s'$, $g$)

**if** $\pi' \neq$ failure **then**

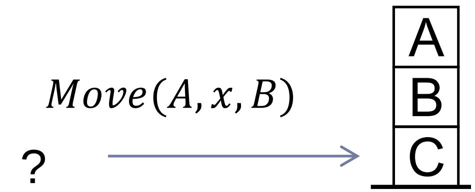
**return** $[a|\pi']$

**return** failure

# Backward state-space search

- An action $a$ is **relevant** for $g$, if $a$ can be the last step in a plan leading to $g$:

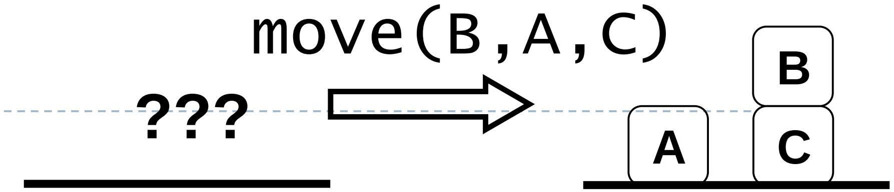
$$g \cap \mathrm{ADD}(a) \neq \emptyset$$
$$g \cap \mathrm{DEL}(a) = \emptyset$$




- Regression: To achieve goal $g$, we regress it through a relevant action $a$ ($a$ as final step of plan to reach $g$):

$$g' = (g - \mathrm{ADD}(a)) \cup \mathrm{PRE}(a)$$

# Regression Example




- $goal = \{On(B, C),\ On(Table, A)\}$

- **Relevant** action: $a = Move(B, A, C)$ $\left\{ \begin{array}{ll} \mathrm{ADD}(a): & On(B, C) \\ \mathrm{DEL}(a): & On(B, A) \\ \mathrm{PREC}(a): & On(B, A), \\ & Clear(B), Clear(C) \end{array} \right.$

$$g \cap \mathrm{ADD(a)} = \{\mathrm{On(B,C)}\} \neq \varnothing$$
$$g \cap \mathrm{DEL(a)} = \varnothing$$

- **Regression** (add preconds. of $a$, remove predicates in add list $a$)
  - $goal = \{\cancel{on(B, C)}, on(Table, A), on(B, A), clear(B), clear(C)\}$

# Backward Search

Backward-Search($s$, $g$)

  **if** $s$ satisfies $g$ **then return** []

  *relevant* = {$a$ | $a$ is relevant to $g$}

  **if** *relevant* = ∅ **then return** failure

  **for each** $a \in$ *relevant* **do**

    $g' = (g - \mathrm{ADD}(a)) \cup \mathrm{PREC}(a)$

    $\pi'$ = Backward-Search($s$, $g'$)

    **if** $\pi' \neq$ failure **then**

      **return** [$\pi'$|a]

  **return** failure

# Backward Search

- Instantiating Schema
  - Goal as a conjunction of literals that may contain variables
  - To be more efficient, instantiate schema variables by **unification**, rather than generating and testing different actions
- For most domains, it has lower branching factor than forward search
- Heuristics are more difficult to use
  - It is based on set of states rather than individual states.

# State-space search problems

- Both of forward and backward algorithms may have repeated states problem
  - **visited** states must be recorded

- What's wrong with search?
  - **Branching factor** is usually too high.
    - Combinatorial explosion if state given by set of possible worlds/logical interpretations/variable assignments

# Heuristic for planning

- Solving problems by searching atomic states (Chapter 3)
  - Human intelligence is usually used to define **domain-specific heuristics**
- Assumption: "path cost = number of plan steps"
  - We want to estimate # of steps needed to reach $g$ from $s$
- In the planning problems, we use factored representation of states
  - This allows us to find **domain-independent heuristics**

# Heuristic for planning

- Heuristics:
  - Relaxed problems:
    - Ignore delete lists
    - Ignore preconditions
  - Problem decomposition
    - Sub-goal independence assumption

# Heuristics: relaxed problems

1) Ignore delete lists:

   Delete negative effects from actions, solve relaxed problem and use the length of plan as heuristic

   - **Admissible**?
   - Can we solve this problem in polynomial time?

2) Ignore preconditions:

   Delete all preconditions from actions, solve relaxed problem and use the length of plan as heuristic

   - **Admissible**?
   - Can we solve this problem in polynomial time?

# Heuristics: problem decomposition

$f(p, s)$: minimum # of steps needed to reach proposition $p$ from $s$

- <u>Sum of the cost of reaching each sub-goal from $s$</u>

$$h_{sum}(s) = \sum_{g \in G} f(g, s)$$

  - Not necessarily admissible
    - independence assumption can be pessimistic

- <u>Max of the cost of reaching each sub-goal from $s$</u>

$$h_{max}(s) = \max_{g \in G} f(g, s)$$

## Heuristics: problem decomposition (sum or max)

- Max or sum?
  - Admissibility vs. accuracy
  - **Sum** works well in practice for problems that are largely decomposable.
- How to compute $f(p,s)$?

# Ignore delete lists & problem decomposition

- When both ignoring delete lists & decomposing the problem
  - we can compute $f(p, s)$ in polynomial time using the Planning Graph (we will see it in the next slides).
  - Examples of such heuristics used in these planners:
    - HSP
    - Fast-Forward (FF)
      - Competed in fully automated track of AIPS'2000
        - Granted ``Group A distinguished performance Planning System''
      - Estimate the heuristic with the help of a planning graph

J. Hoffman, B. Nebel, "The FF planning system: Fast plan generation through heuristic search", Journal of Artificial Intelligence Research 14 (2001), 253-302

# Planning graph

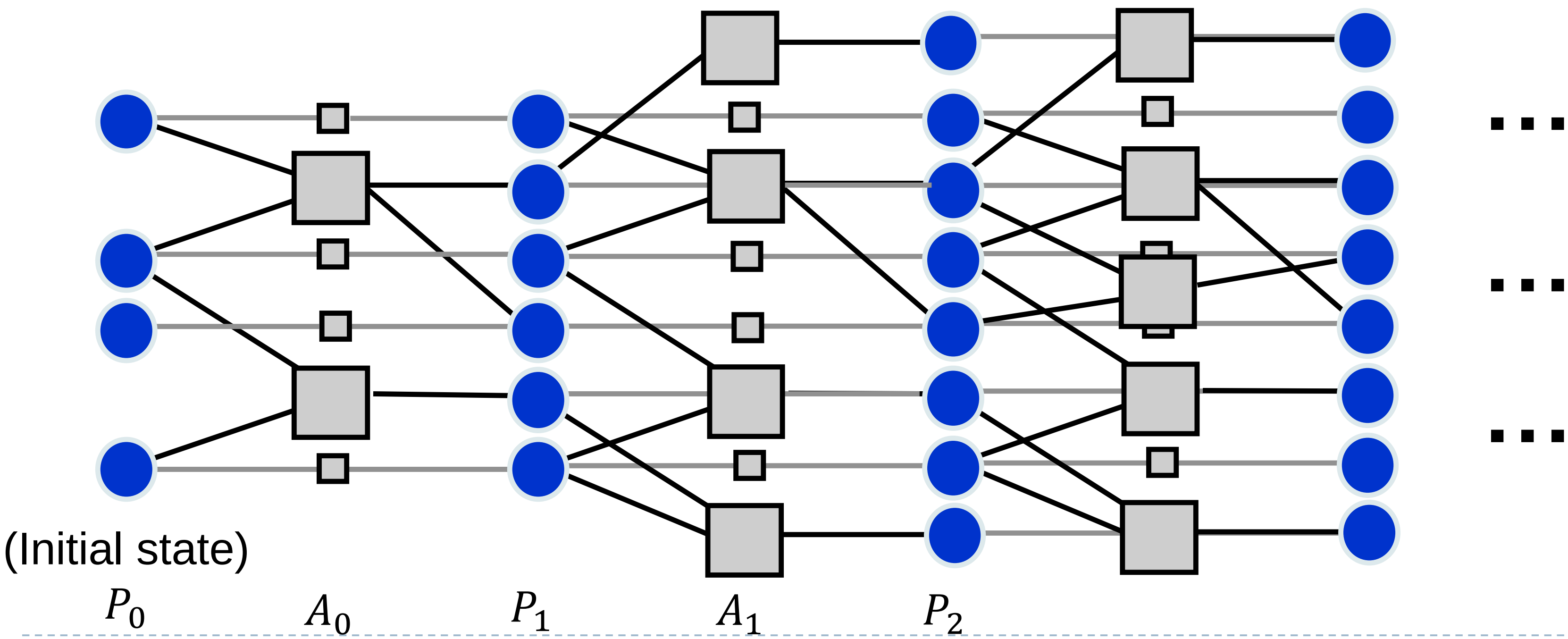
- A way to find accurate heuristics
- (Under)estimating no. of steps required to reach $g$
  - Admissible
- A layered graph that keeps track of literal pairs and action pairs that cannot be reached simultaneously (mutexes)

# Planning graph: structure

- Directed, leveled graph
  - Two types of levels:
    - $P$: proposition levels
    - $A$: action levels
    - Proposition and action levels alternate
  - Edges (between levels)
    - Precondition: each action at $A_i$ is connected to its preconditions at $P_i$
    - Effect: each action at $A_i$ is connected to its effects at $P_{i+1}$

# Planning graph: layers

- $P_i$ contains all the literals that could hold at time $i$
- $A_i$ contains all actions whose preconditions are satisfied in $P_i$ plus no-op actions (to solve frame problem).

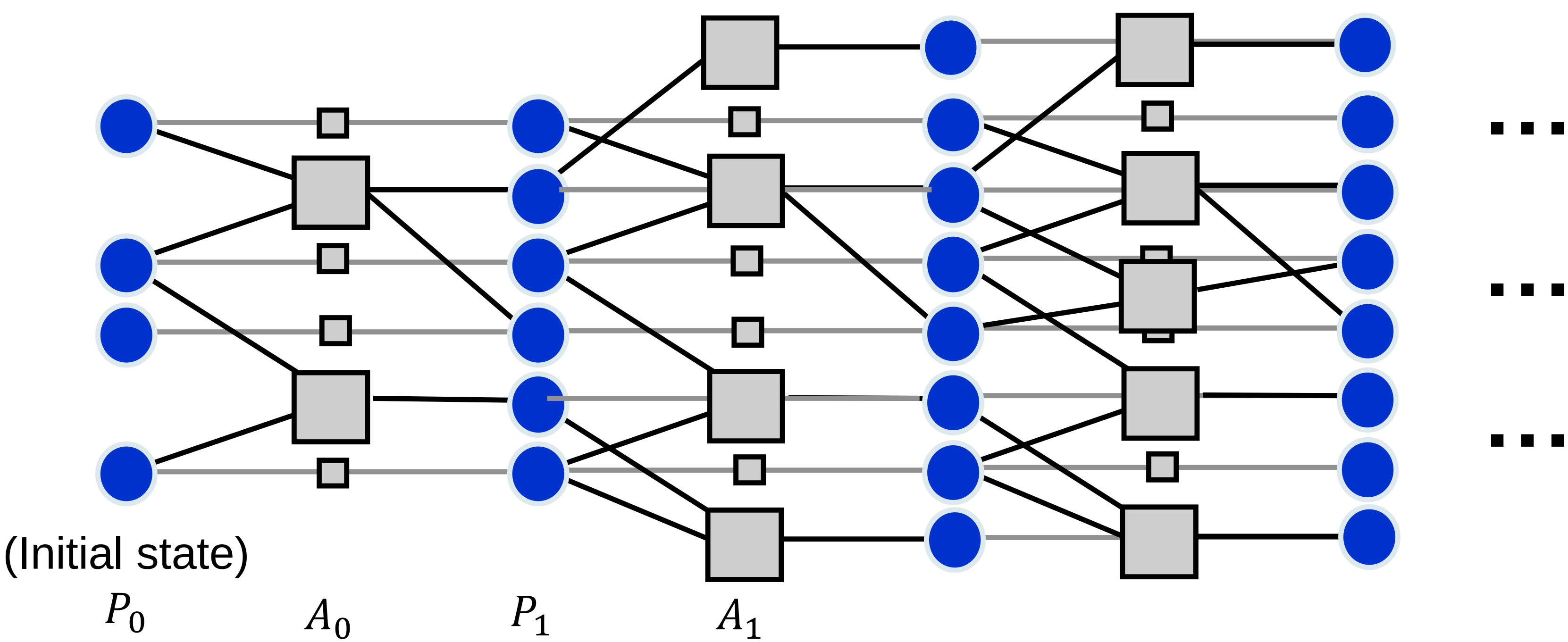


# Planning graph: layers

$$P_0 \;=\; \{p \in Init\}$$

$$A_i \;=\; \{a \text{ is an action} |\ \mathrm{PRECONDS}(a) \subseteq P_i\}$$

$$P_{i+1} = \{p \in \mathrm{EFFECT}(a) |\ a\ \in A_i\}$$



# Planning graph: Cake example

$Init(Have(Cake))$

$Goal(Have(Cake) \wedge Eaten(Cake))$

$Action(Eat(Cake)$

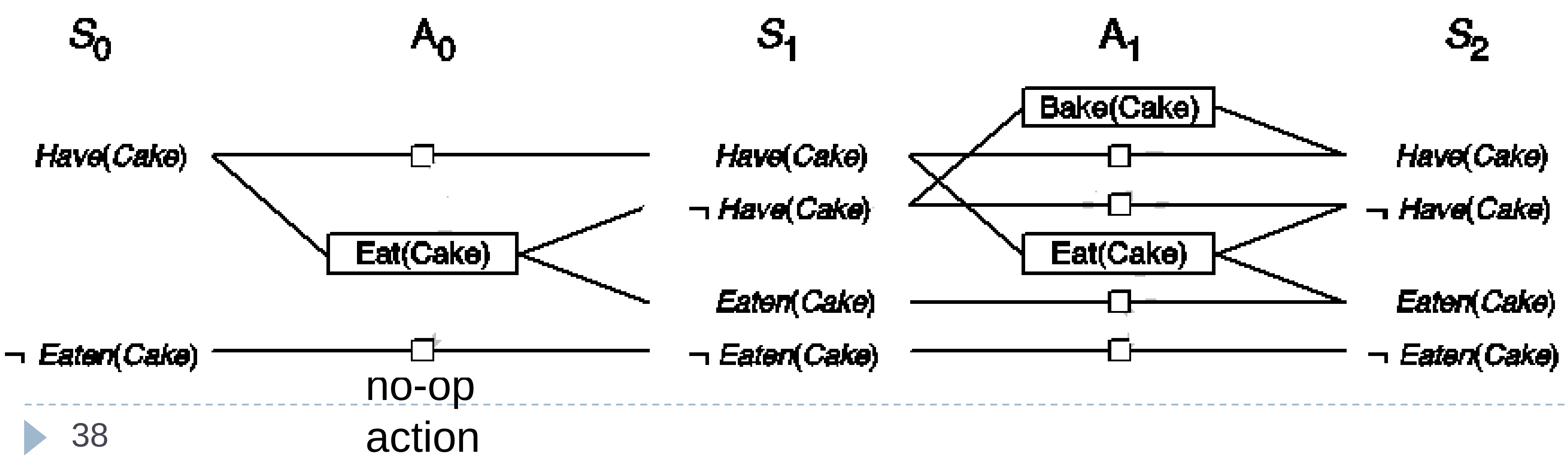
PRECOND: $Have(Cake)$

EFFECT: $\neg Have(Cake) \wedge Eaten(Cake))$

$Action(Bake(Cake)$

PRECOND: $\neg Have(Cake)$

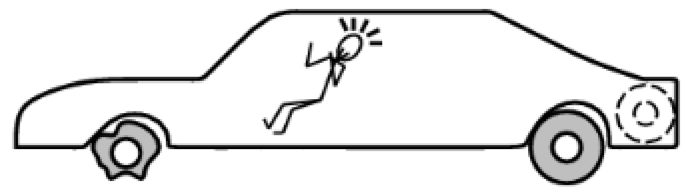
EFFECT: $Have(Cake))$



# Planning graph: Spare tire example

$Init(Tire(Flat) \land Tire(Spare) \land At(Flat, Axle) \land At(Spare, Trunk))$

$Goal(At(Spare, Axle))$

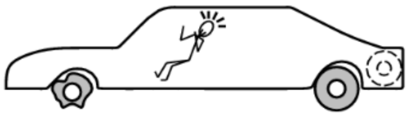


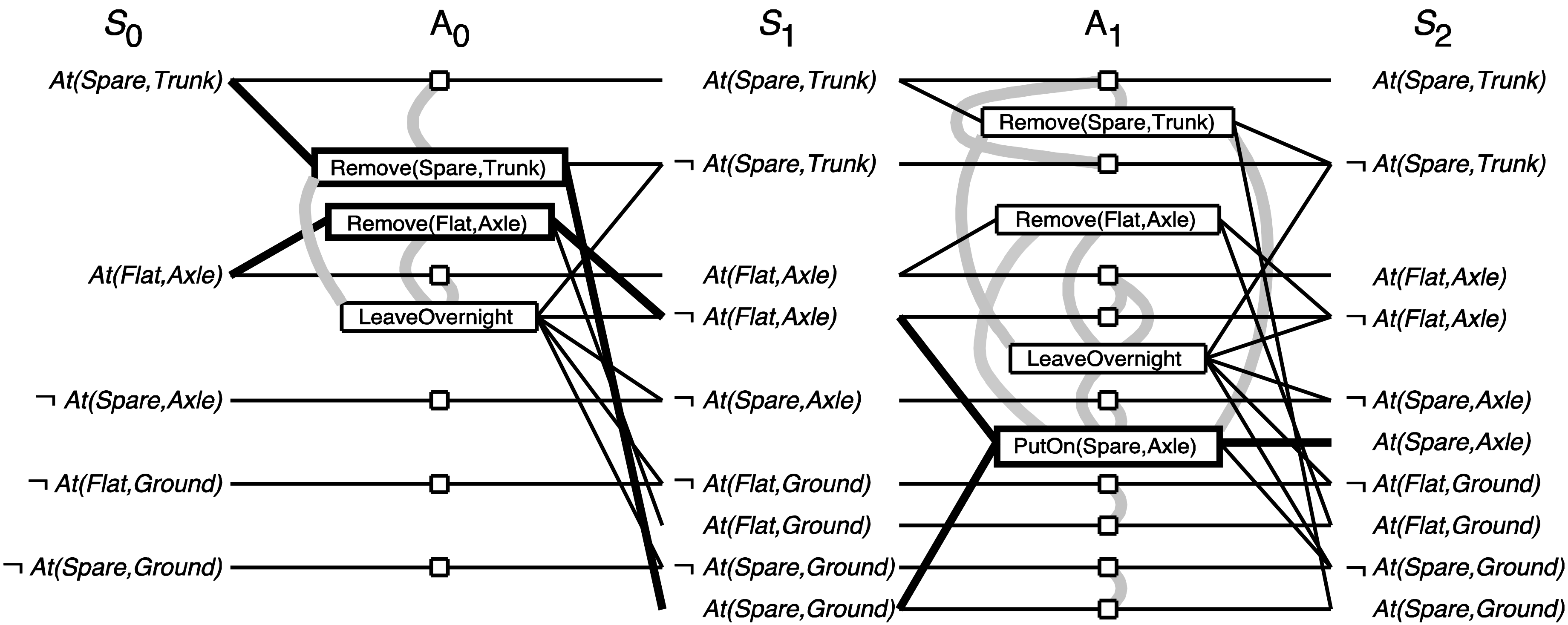
$Action(Remove(obj, loc),$
$\quad PRECOND: At(obj, loc),$
$\quad EFFECT: \neg At(obj, loc) \land At(obj, Ground))$

$Action(PutOn(t, Axle),$
$\quad PRECOND: Tire(t) \land At(t, Ground) \land \neg At(Flat, Axle)$
$\quad EFFECT: \neg At(t, Ground) \land At(Flat, Axle))$

$Action(LeaveOvernight,$
$\quad PRECOND:$
$\quad EFFECT: \neg At(Spare, Ground) \land \neg At(Spare, Trunk) \land \neg At(Spare, Axle)$
$\quad\quad \land \neg At(Flat, Ground) \land \neg At(Flat, Trunk) \land \neg At(Flat, Axle))$

# Planning graph: Spare tire example





# Planning graphs: properties

- In level $P_i$, both $P$ and $\neg P$ may exist.

- A literal may appear at level $P_i$ while actually it could not be true until a later level (if any)

- A literal will never appear late in the planning graph.

# Planning graphs: cost of each goal literal

- How difficult is it to achieve a goal literal $g_i$ from $s$?
  - Level-cost of $g_i$ $(lc(g_i, s))$ : It shows the first level of PG at which $g_i$ appears.

- Relation to previously introduced heuristics?
  - Is it accurate?

# Planning graphs: heuristics

$$h_{\max\_level}(s) = \max_{g_i \in goal} lc(g_i, s)$$

$$h_{level\_sum}(s) = \sum_{g_i \in goal} lc(g_i, s)$$

# Planning graphs: constraints

- Mutual exclusion (mutex) links

- <u>Two actions</u> at a given action level are mutually exclusive if no valid plan could possibly contain both.

- Two propositions at a given proposition level are mutually exclusive if no valid plan could possibly make both true.

- This structure helps in reducing the search for a sub-graph of a Planning Graph that might correspond to a valid plan.

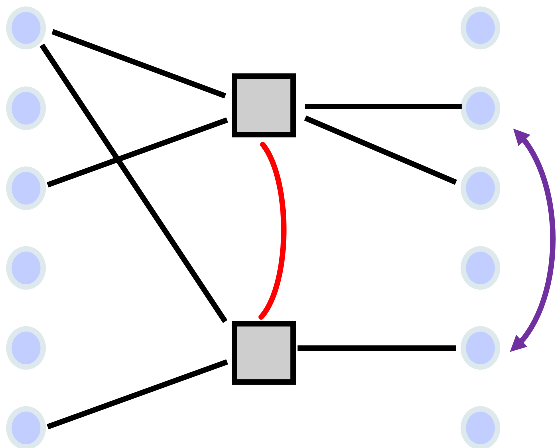
# Planning graphs: constraints

- Mutexes between actions
  - Inconsistent effects: one action negates an effect of the other
  - Interference: one of the effects of one action is the negation of a precondition of the other
  - Competing needs: mutually exclusive preconditions

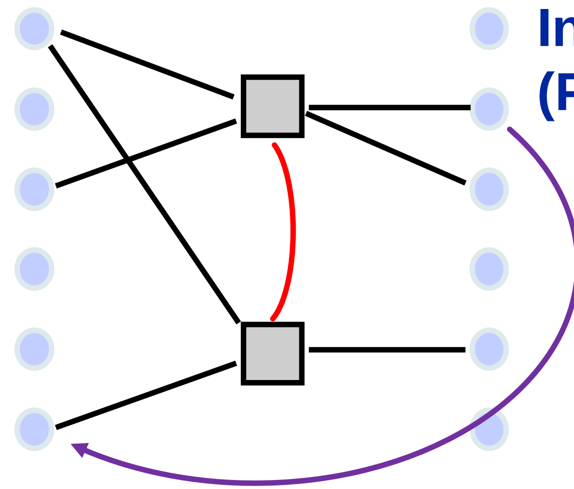
- Mutexes between literals
  - One of the literals is the negation of the other
  - Inconsistent support: Each possible pair of actions that could achieve them (in this level) is mutually exclusive.

# Planning graphs: constraints
# Types of mutexes

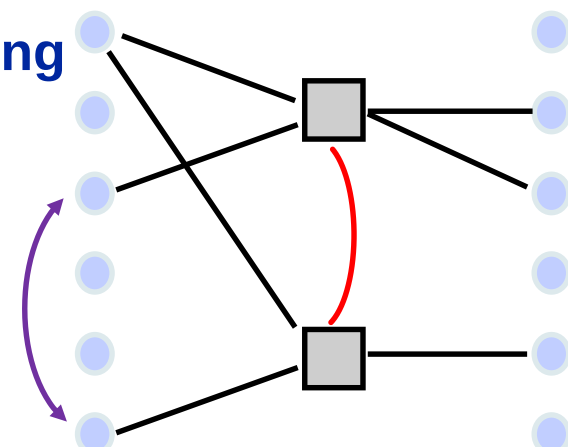
**Inconsistent Effects**



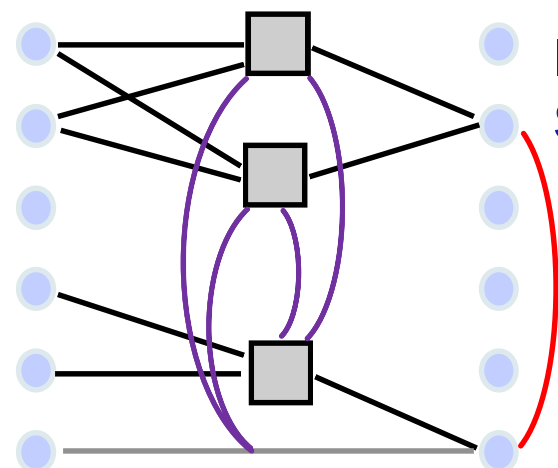
**Interference (Prec-Effect)**



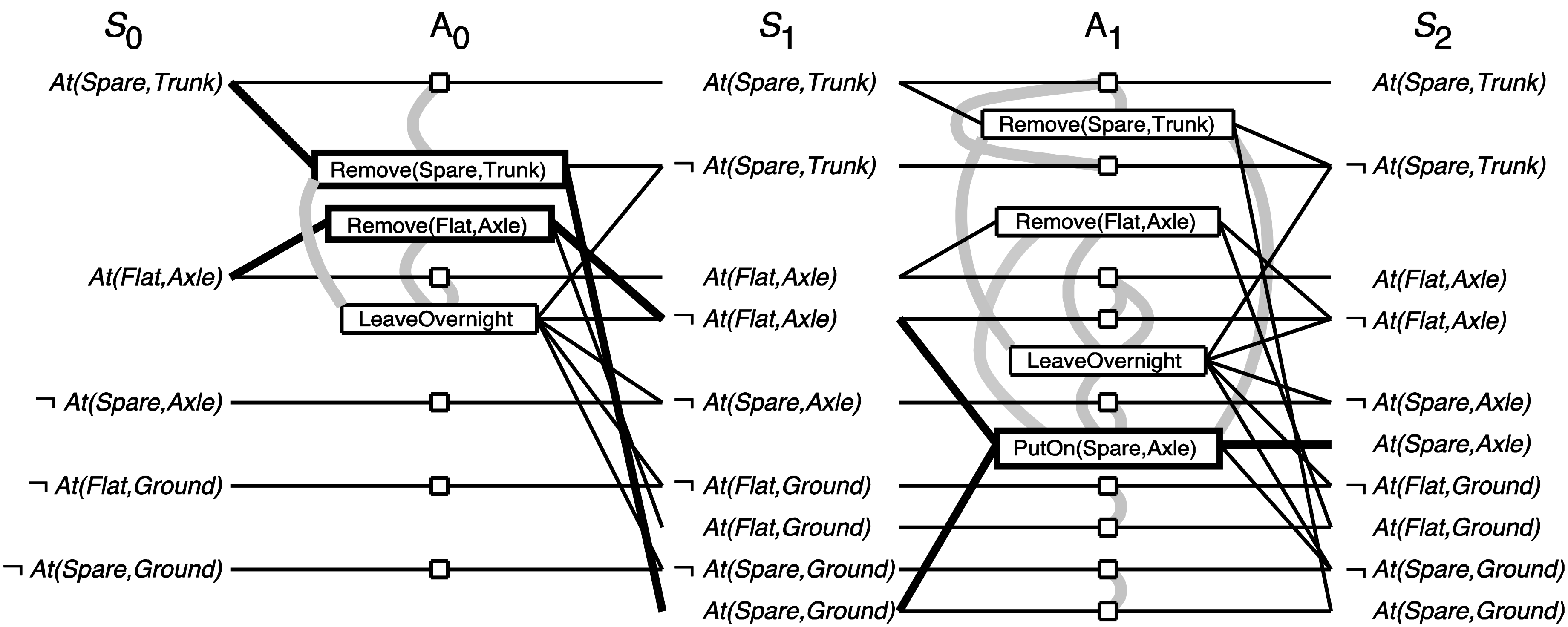
**Competing Needs**



**Inconsistent Support**



# Planning graph: Spare tire example



# Planning graph: more accurate heuristic

- We want to define a more accurate heuristic using the mutexes:

$h_2$ : the level at which all the goal literals appear without any pair of them being mutually exclusive.

- $h_1$ (max-level heuristic) is extended to $h_2$ considering mutexes between all pairs of propositions.
  - $h_2$ is more useful than $h_1$ $(0 \leq h_1 \leq h_2 \leq h^*)$

# Planning graph: more accurate heuristics

- $h_2$ can be extended to $h_3$ by defining and considering inconsistencies of <u>triplets</u> of propositions

- In general
  - $h_k$ are admissible
  - $h_{k+1} \geq h_k$
  - Computing $h_k$ is $O(n^k)$ with $n$ propositions

- $k = 2$ is commonly used

# GraphPlan: basic idea

- Construct a graph that encodes constraints on plans

- Use this graph to constrain search for a valid plan:
  - If a valid plan exists it is a sub-graph of the Planning Graph.
    - Actions at the same level don't interfere
    - Each action's preconditions are made true by the plan
    - Goals are satisfied

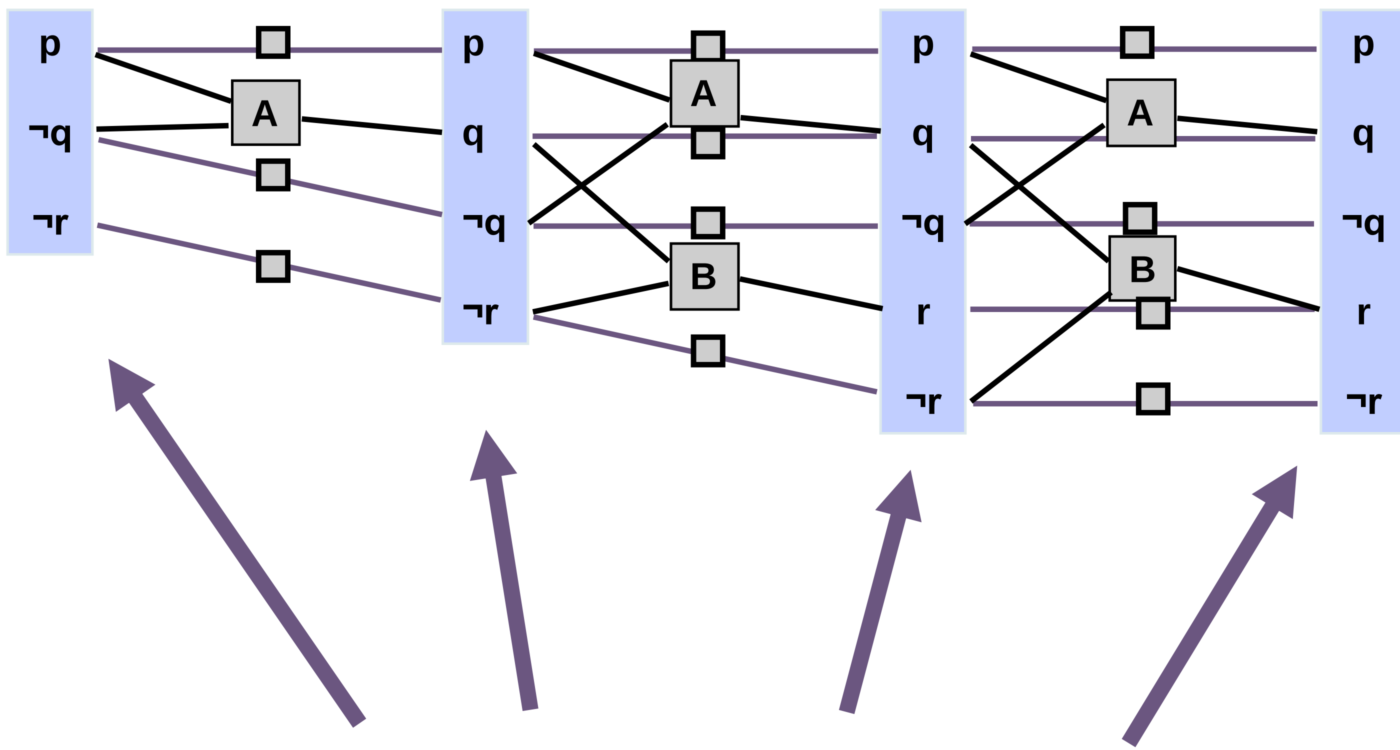
- Planning graph can be built for each problem in polynomial time.

# GraphPlan: level off

- Definition: Planning Graph levels off if two consecutive proposition levels are identical (both literals and mutexes).

- We will show that the set of literals never decreases in the proposition levels and mutexes don't reappear.

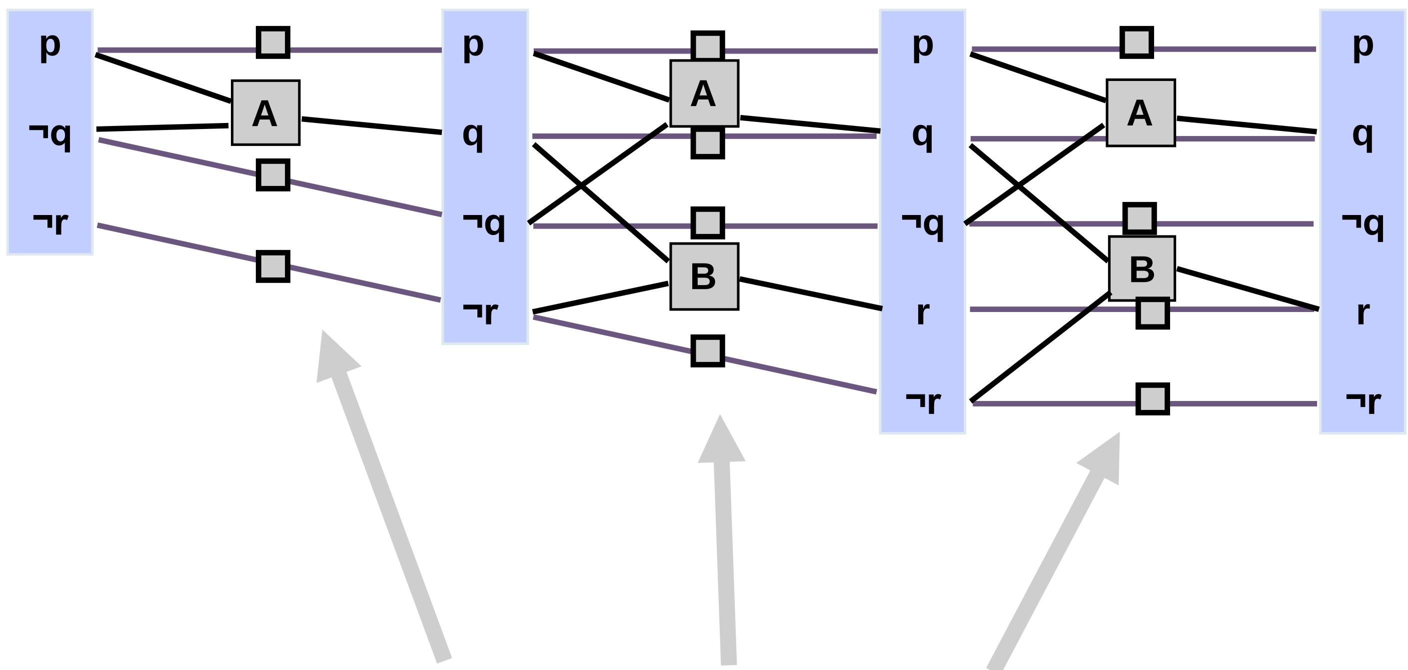
# GraphPlan: level off (Observation 1)




**Literals monotonically increase**

Propositions are always carried forward by no-ops.

# GraphPlan: level off (Observation 2)

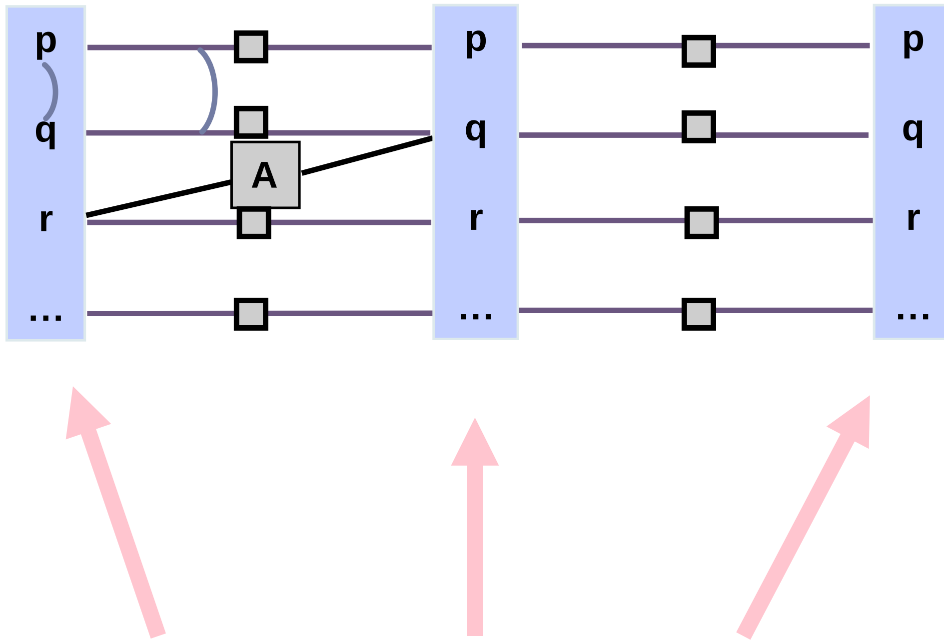



**Actions monotonically increase**

**(Once an action appears at a level, it will appear at all subsequent levels)**

If preconds. of an action appear at one level, they will appear at subsequent levels and thus the action will appear so.

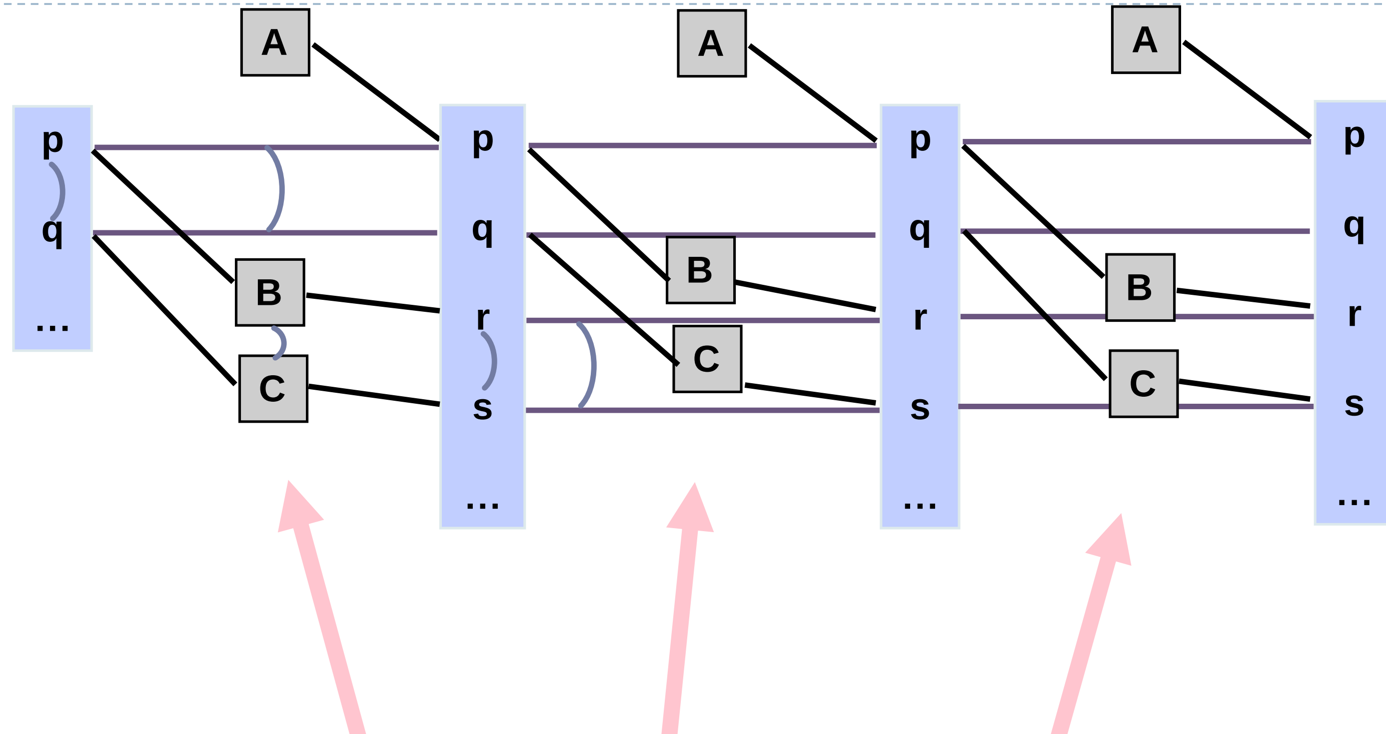
# GraphPlan: level off (Observation 3)




**Proposition mutex relationships monotonically decrease**

Available actions are monotonically increasing. Thus mutex relations between literals are decreasing.
(When mutexes between literals are due to mutex relations between actions, they may be removed in the next levels)

# GraphPlan: level off (Observation 4)




**Action mutex relationships monotonically decrease**

Mutex relations between actions due to competing needs (when preconditions are not negations of each other) must be decreasing.

# GraphPlan Algorithm

necessary, but usually insufficient condition for plan existence

1) Graph levels are constructed until <u>all goals are reached and not mutex</u>.
   - If PG levels off before reaching this level, GraphPlan returns failure.

2) **ExtractSolution** phase: search the PG for a valid plan

3) If non found, add a level to the PG and go to step 2.

GraphPlan builds graph **forward** and extracts plan **backwards**

# GraphPlan: "Extract Solution" phase

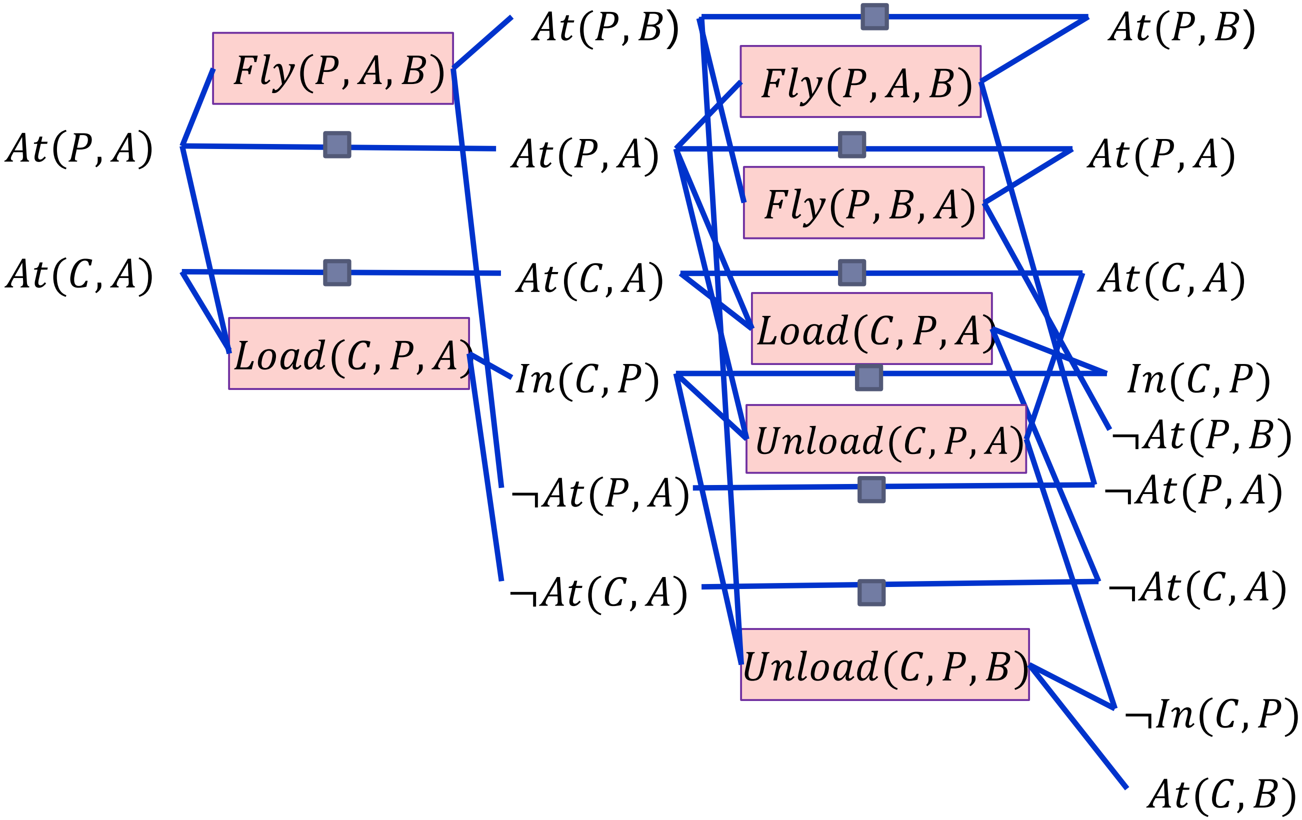
- Some ways
  - As a **backward search**
    - looks for actions that produce goals while pruning as many of them as possible via incompatibility information.
  - As a **heuristic search** computes an **admissible heuristic** for each state and then uses it during search.
  - As a **SAT** problem (related to SATPlan algorithm)
    - Variables: a variable for an action at each level
    - Domain={0,1}
    - Constraints: mutexes

# Extract Solution: backward search

Start from the last level & agenda=goals

- Termination: $k = 0$
- Action Selection: At each level $k$, select any conflict-free subset of actions in $A_{k-1}$ whose effects cover current goals.
  - If no such subset is found return failure
- Preconditions of selected actions become new goals for recursive call at level $k - 1$.

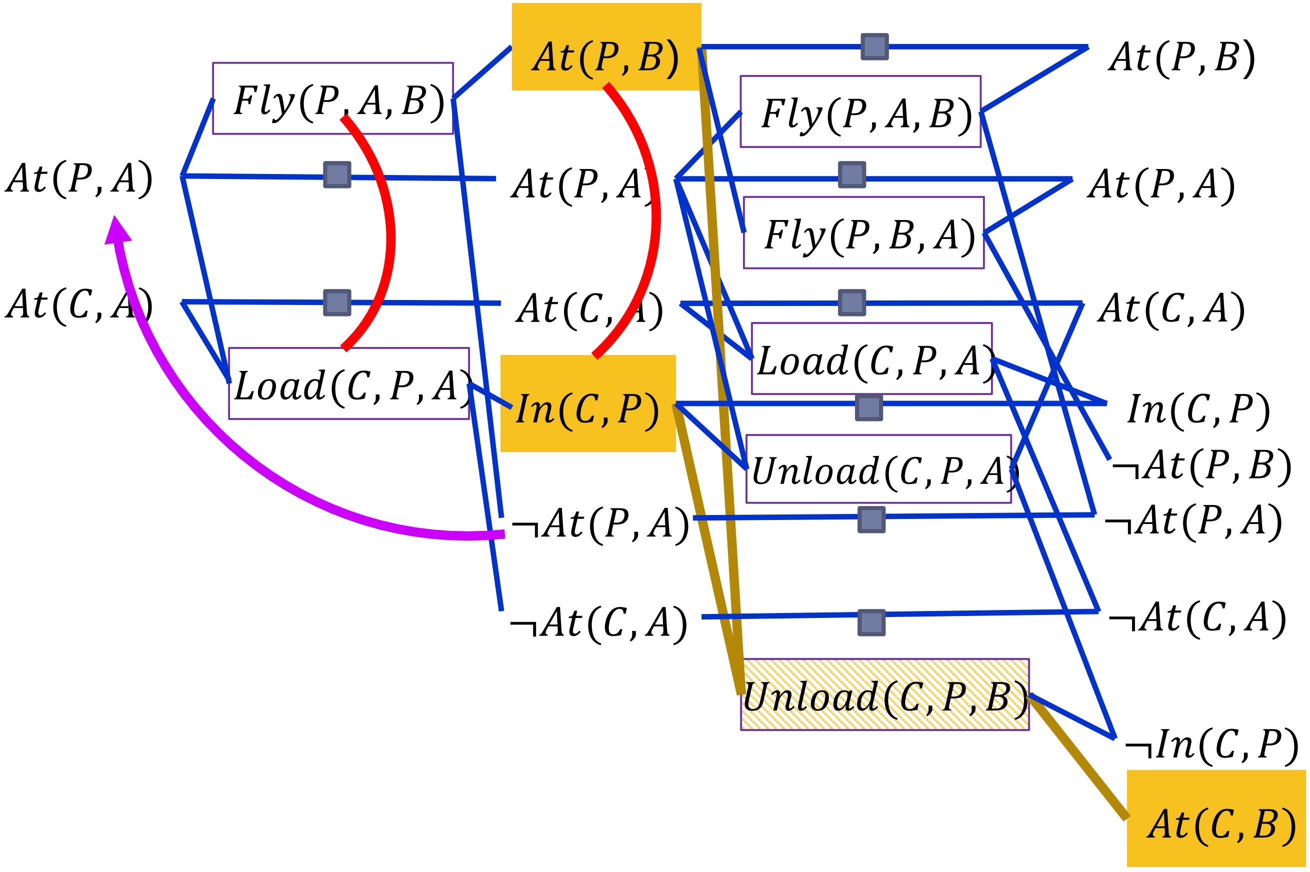
# GraphPlan: Example




A: Airport
P: Plane
C: Cargo

Goal
$At(C,B)$

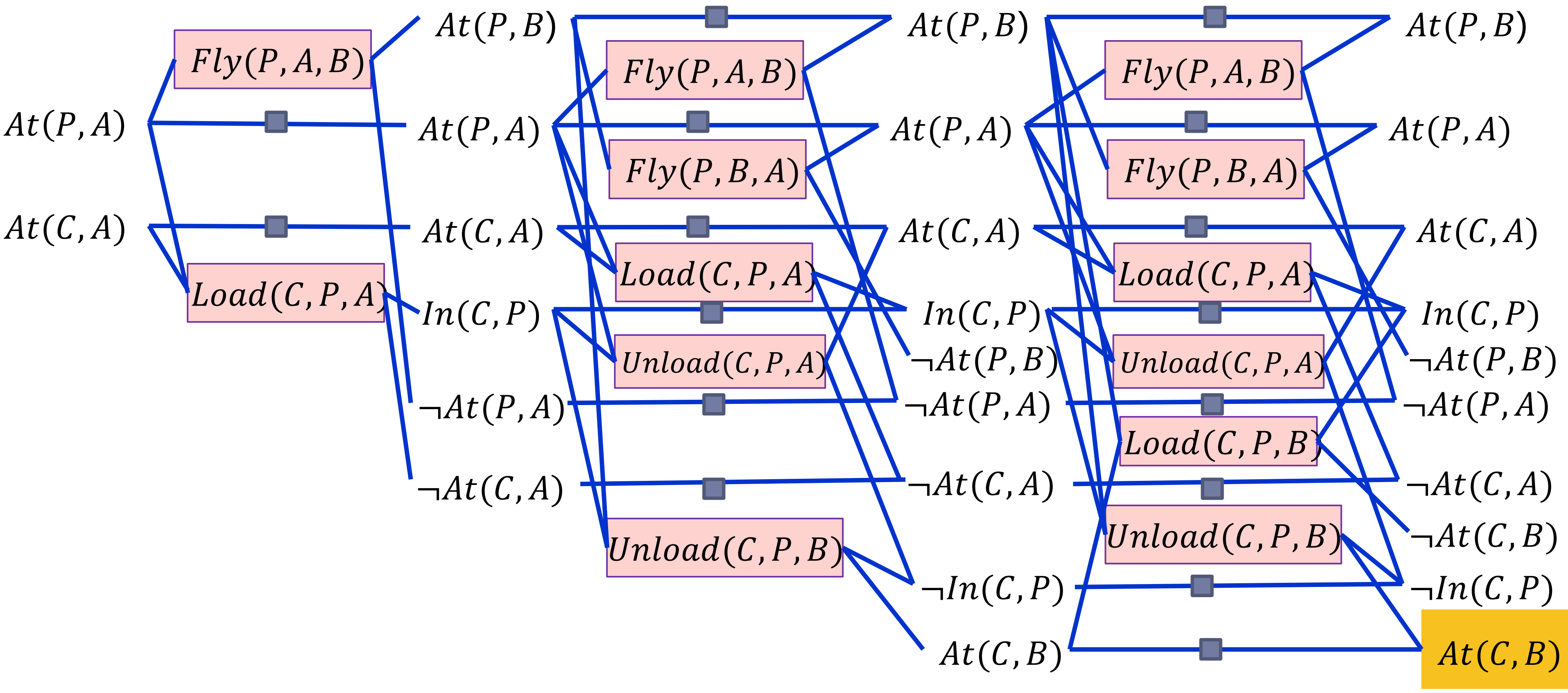
# GraphPlan: Example




A: Airport
P: Plane
C: Cargo

Goal
$At(C,B)$

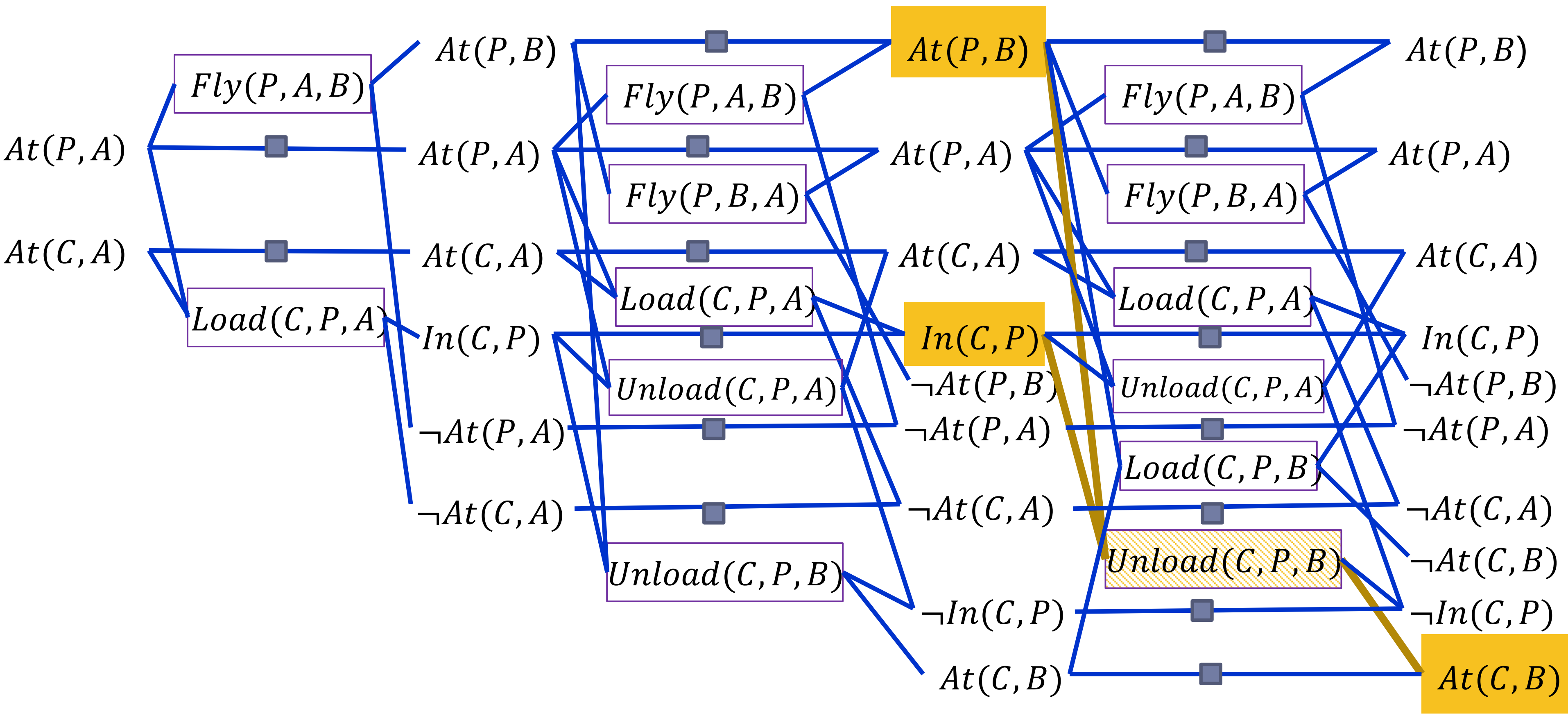
# GraphPlan: Example




A: Airport
P: Plane
C: Cargo

Goal
$At(C,B)$

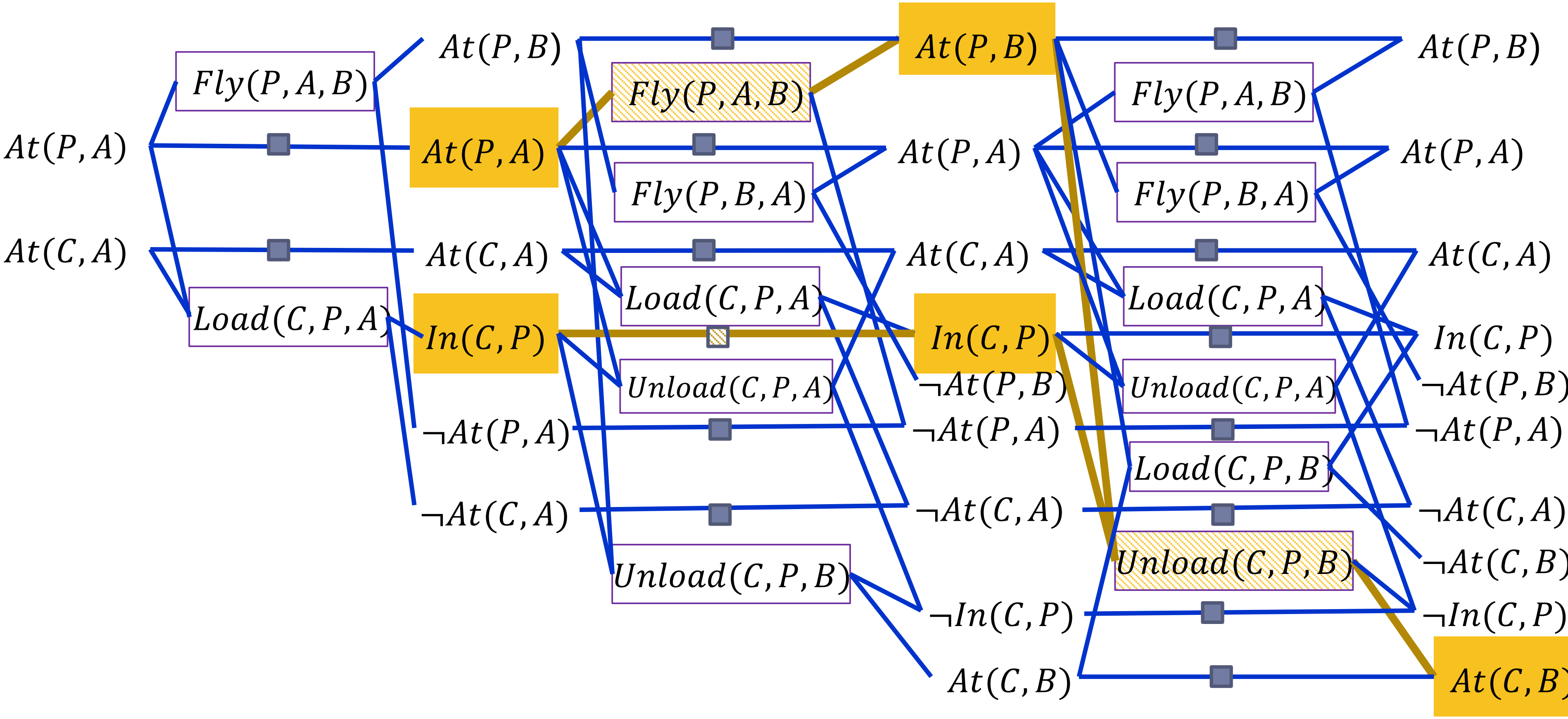
# GraphPlan: Example




A: Airport
P: Plane
C: Cargo

Goal
$At(C,B)$

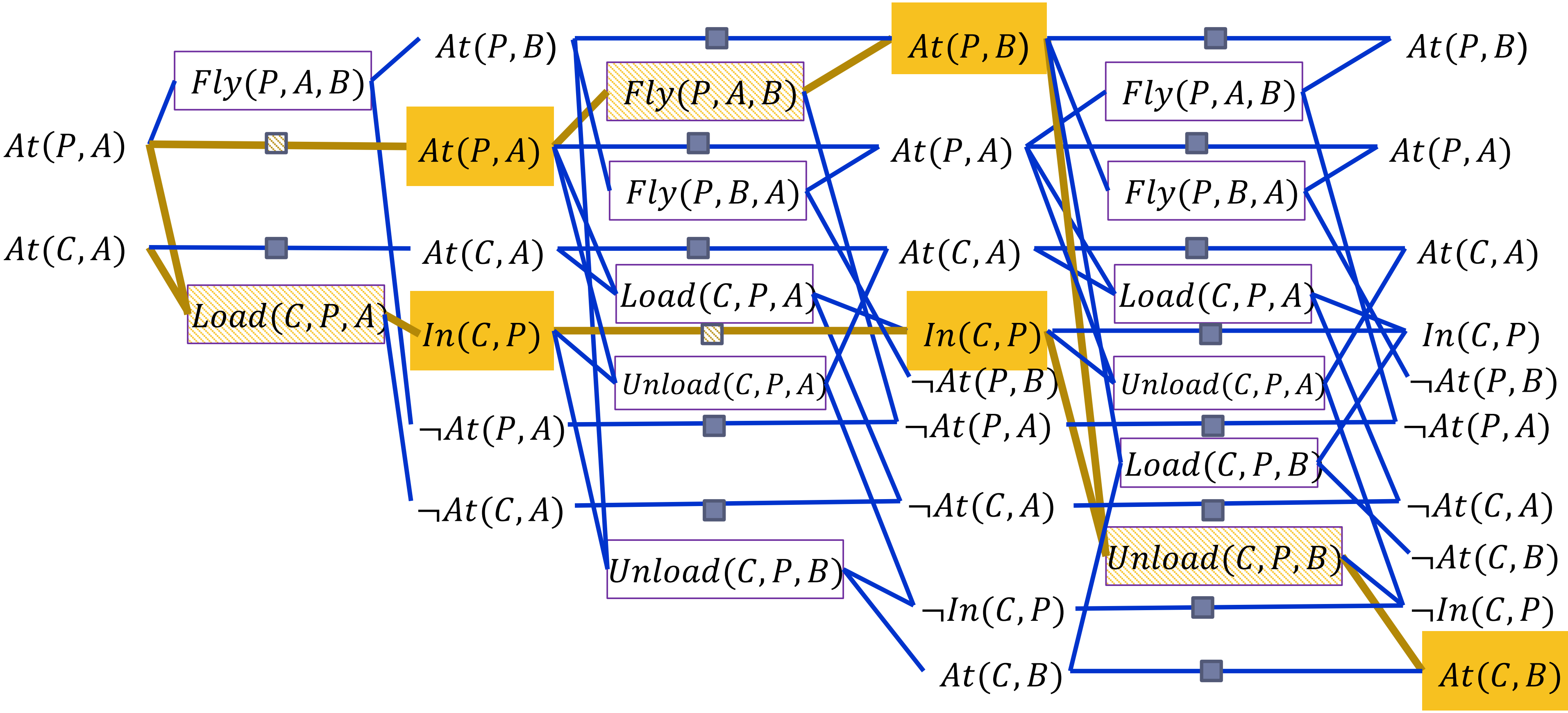
# GraphPlan: Example




A: Airport
P: Plane
C: Cargo

Goal
$At(C,B)$

# GraphPlan: Example




A: Airport
P: Plane
C: Cargo

Goal

$At(C,B)$

# GraphPlan: heuristics for backward search

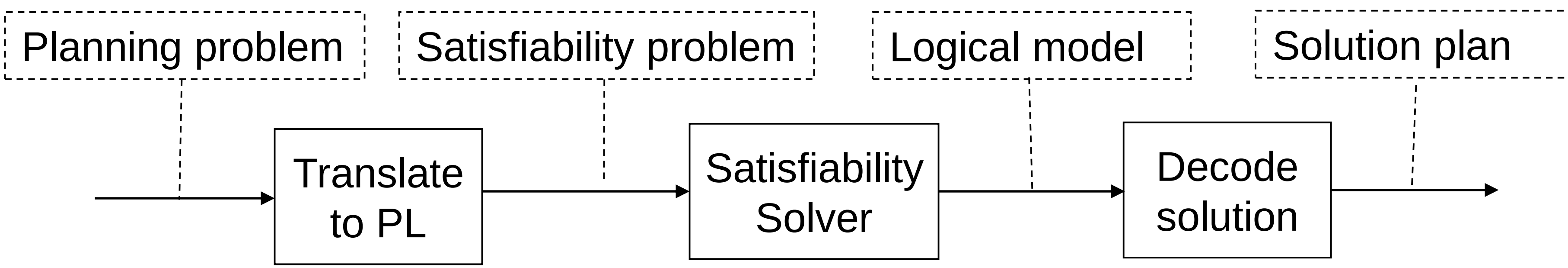
- Pick first the goal literal with the highest level cost

- To achieve a literal prefer actions with easier preconds.
  - Sum (or max) of the level costs of its preconds. is smallest.

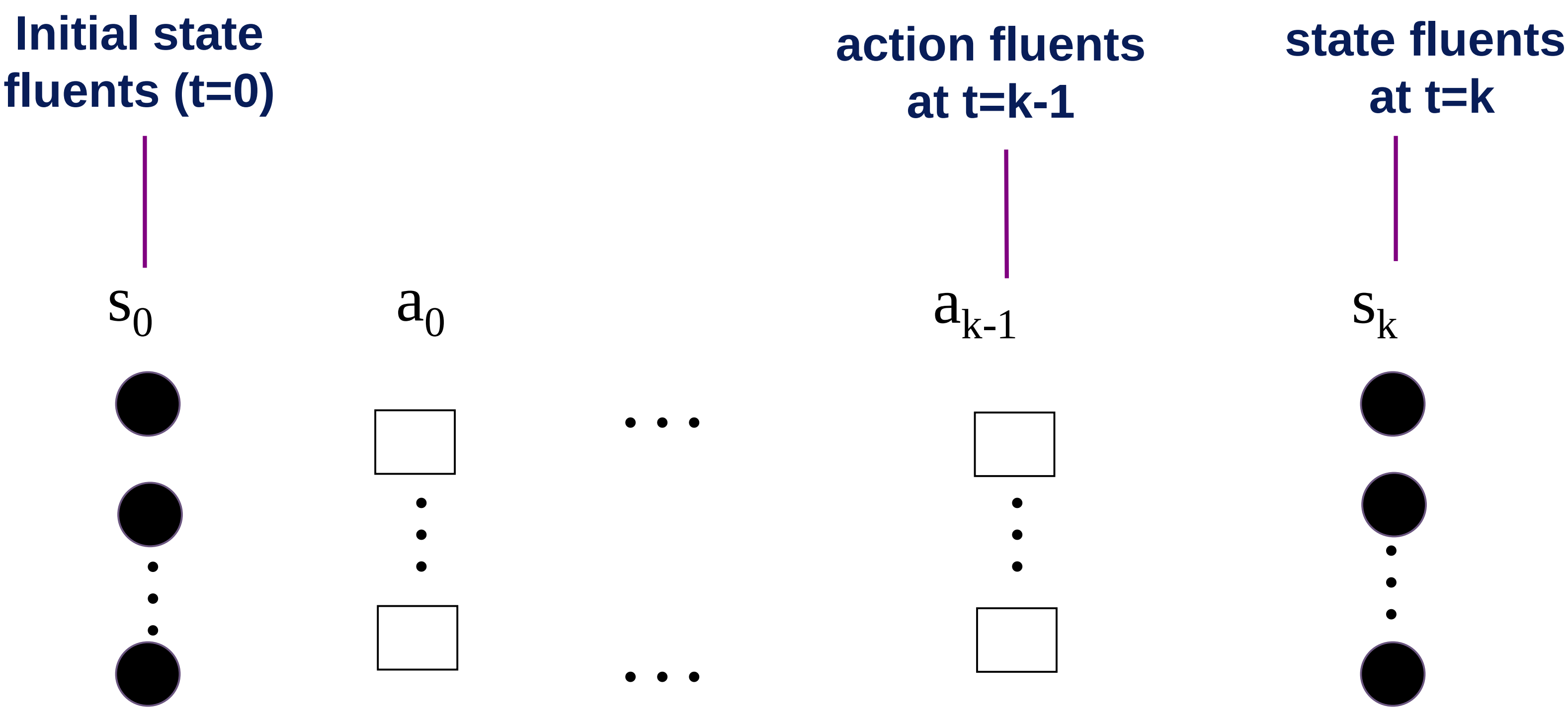
# Planning as a satisfiability problem

- Bounded planning problem $(P, k)$:
  - $P$ is a planning problem
  - Find a solution for $P$ of length $k$

1) Translate $(P, k)$ into a SAT problem.
2) Solve SAT problem.
3) Convert the solution to a plan



# Pictorial view of fluent for (P,k)




- Truth assignment selects a subset of these nodes to be true
- Propositional formulas correspond to valid plans

# Translating PDDL to propositional logic

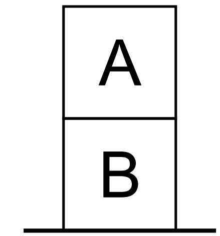
- **Initial state:** Conjunction of all true literals at time 0 (and negation of not mentioned literals)
- **Goal state:** Conjunction of all goal literals at time $k$
  - Instantiate literals containing variable (replace with ∨ over constants).
- **Actions**
  - successor-state axioms at each time up to $t$
    - $F^{t+1} \Leftrightarrow ActionCausesF^t \vee (F^t \wedge \neg ActionCausesNotF^t)$
  - precondition axioms:
    - $A^t \Rightarrow \mathrm{PRECOND}(A)^t$
  - action exclusion axioms:
    - $\neg A_i^t \vee \neg A_j^t$

# Translating PDDL to propositional logic: Example

- **Initial state:** Conjunction of all true literals at time 0 (and negation of not mentioned literals)

$Init(On(A,B) \wedge On(B,Table))$

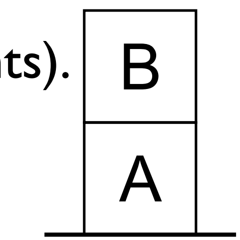
$On(A,B)^0 \wedge On(B,Table)^0 \wedge \neg On(B,A)^0 \wedge \neg On(A,Table)^0$




- **Goal state:** Conjunction of all goal literals at time $k$
  - Instantiate literals containing variable (replace with ∨ over constants).

$Goal(On(B,A))$

$On(B,A)^1$ (for $k = 1$)



# Translating PDDL to propositional logic: Example

- Add successor-state axioms at each time up to $t$
  - $F^{t+1} \Leftrightarrow ActionCausesF^t \vee (F^t \wedge \neg ActionCausesNotF^t)$
    - Example: $On(B,A)^{t+1} \Leftrightarrow Move(B,Table,A)^t \vee [\mathrm{On(B,A)}^t \wedge \neg Move(B,A,Table)^t]$
- Add precondition axioms:
  - $A^t \Rightarrow \mathrm{PRECOND}(A)^t$
    - Example: $Move(B,Table,A)^t \Rightarrow On(B,Table)^t \wedge Clear(B)^t \wedge Clear(A)^t$
  - Is it necessary to include effects: $A^t \Rightarrow \mathrm{EFFECT}(A)^{t+1}$ ?
- Add action exclusion axioms:
  - $\neg A_i^t \vee \neg A_j^t$
    - Example: $\neg Move(B,Table,A)^0 \vee \neg MoveToTable(A,B)^0$

# Propositional logic solver and decoding

- Apply a SAT solver to the whole sentence $\Phi$
  - $\Phi$: conjunction of encoding initial state, goals, successor-state axioms, precondition axioms, action exclusion axioms

- If an assignment of truth values that satisfies $\Phi$ is found, extract action sequence.
  - This means $P$ has a solution of length $k$

- Extract solution: For $i = 0, \ldots, k-1$, there is exactly one action that has been assigned "True"
  - This is the $i$'th action of the plan.

# SATPlan

**function** SATPLAN($init, transition, goal, T\_max$) **returns** solution or failure
  **inputs**: $init, transition, goal$, constitute a description of the problem
    $T\_max$, an upper limit for plan length

  **for** $t = 0$ **to** $T\_\text{max}$ **do**
    $cnf \leftarrow \text{TRANSLATE\_TO\_SAT}(init, transition, goal, t)$
    $model \leftarrow \text{SAT\_SOLVER}(cnf)$
    **if** $model \neq \{\}$ **then**
      **return** $\text{EXTRACT\_SOLUTION}(model)$
  **return** $failure$

It is guaranteed to find the shortest plan if one exist.

# SATPlan example

- Domain:
  - Robot $R$
  - Two locations $L_1, L_2$
  - One operator "move" the robot


- Initial state: $At(R, L_1)$

- Goal: $At(R, L_2)$

- Action schema:
  - $Move(r, l, l')$

    $PRECOND: At(r, l)$
    $EFFECT: At(r, l') \land \neg At(r, l)$

# SATPlan example (translation to SAT)

- Encode $(P, 1)$
  - Initial state:
    - $At(R, L_1, 0) \wedge \neg At(R, L_2, 0)$
  - Goal:
    - $At(R, L_2, 1)$
  - Actions preconditions:
    - $Move(R, L_1, L_2, 0) \Rightarrow At(R, L_1, 0)$
    - $Move(R, L_2, L_1, 0) \Rightarrow At(R, L_2, 0)$
  - Action exclusion axiom:
    - $\neg Move(R, L_2, L_1, 0) \vee \neg Move(R, L_1, L_2, 0)$

# SATPlan example (translation to SAT)

- Fluents (Success-state axioms):
  - $\neg At(R, L_1, 0) \wedge At(R, L_1, 1) \Rightarrow Move(R, L_2, L_1, 0)$
  - $\neg At(R, L_2, 0) \wedge At(R, L_2, 1) \Rightarrow Move(R, L_1, L_2, 0)$
  - $At(R, L_1, 0) \wedge \neg At(R, L_1, 1) \Rightarrow Move(R, L_1, L_2, 0)$
  - $At(R, L_2, 0) \wedge \neg At(R, L_2, 1) \Rightarrow Move(R, L_2, L_1, 0)$

# SATPlan example (translation to SAT)

SAT formula for ($P$,1)

$$At(R, L_1, 0) \land \neg At(R, L_2, 0) \land$$
$$At(R, L_2, 1) \land$$
$$[Move(R, L_1, L_2, 0) \Rightarrow At(R, L_1, 0)] \land$$
$$[Move(R, L_1, L_2, 0) \Rightarrow At(R, L_2, 1)] \land$$
$$[\neg Move(R, L_2, L_1, 0) \lor \neg Move(R, L_1, L_2, 0)] \land$$
$$[\neg At(R, L_1, 0) \land At(R, L_1, 1) \Rightarrow Move(R, L_2, L_1, 0)] \land$$
$$[\neg At(R, L_2, 0) \land At(R, L_2, 1) \Rightarrow Move(R, L_1, L_2, 0)] \land$$
$$[At(R, L_1, 0) \land \neg At(R, L_1, 1) \Rightarrow Move(R, L_1, L_2, 0)] \land$$
$$[At(R, L_2, 0) \land \neg At(R, L_2, 1) \Rightarrow Move(R, L_2, L_1, 0)]$$

Above formula is converted to CNF and solved by a SAT solver.

# SATPlan example (Extracting a plan)

- $\Phi$ can be satisfied with $move(R, L_1, L_2, 0) \ = \ true$

  $\Rightarrow$ $move(R, L_1, L_2, 0)$ is a solution (and the only one) for panning problem with 1 step plan

# Layered Plans in SATPlan

- Complete exclusion axiom (only one action at a time):
  - For **<u>all</u>** pairs of actions at each time step i:

$$\neg a_i \vee \neg bi$$

- Partial exclusion axiom (more than one action could be taken at a time step):
  - For <u>any pair of incompatible actions</u> (recall from Graphplan):

$$\neg a_i \vee \neg bi$$

  - Fewer time steps may be required (i.e. shorter formulas)

# Solving SAT problem

- Systematic search
  - DPLL (Davis Putnam Logemann Loveland)
- Local search
  - WalkSAT

# Partial order planning
# Sock-shoe example: PDDL

$Init()$

$Goal(RightShoeOn \wedge LeftShoeOn)$

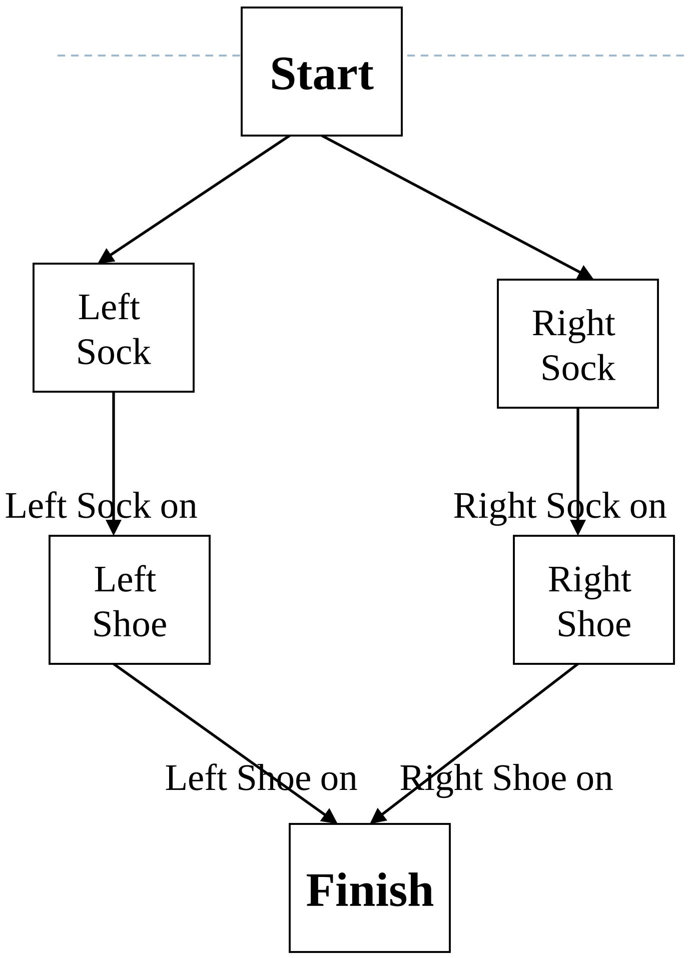
$Action(RightShoe,$
 $\text{PRECOND: } RightSockOn,$
 $\text{EFFECT: } RightShoeOn))$

$Action(RightSock,$
 $\text{EFFECT: } RightSockOn))$

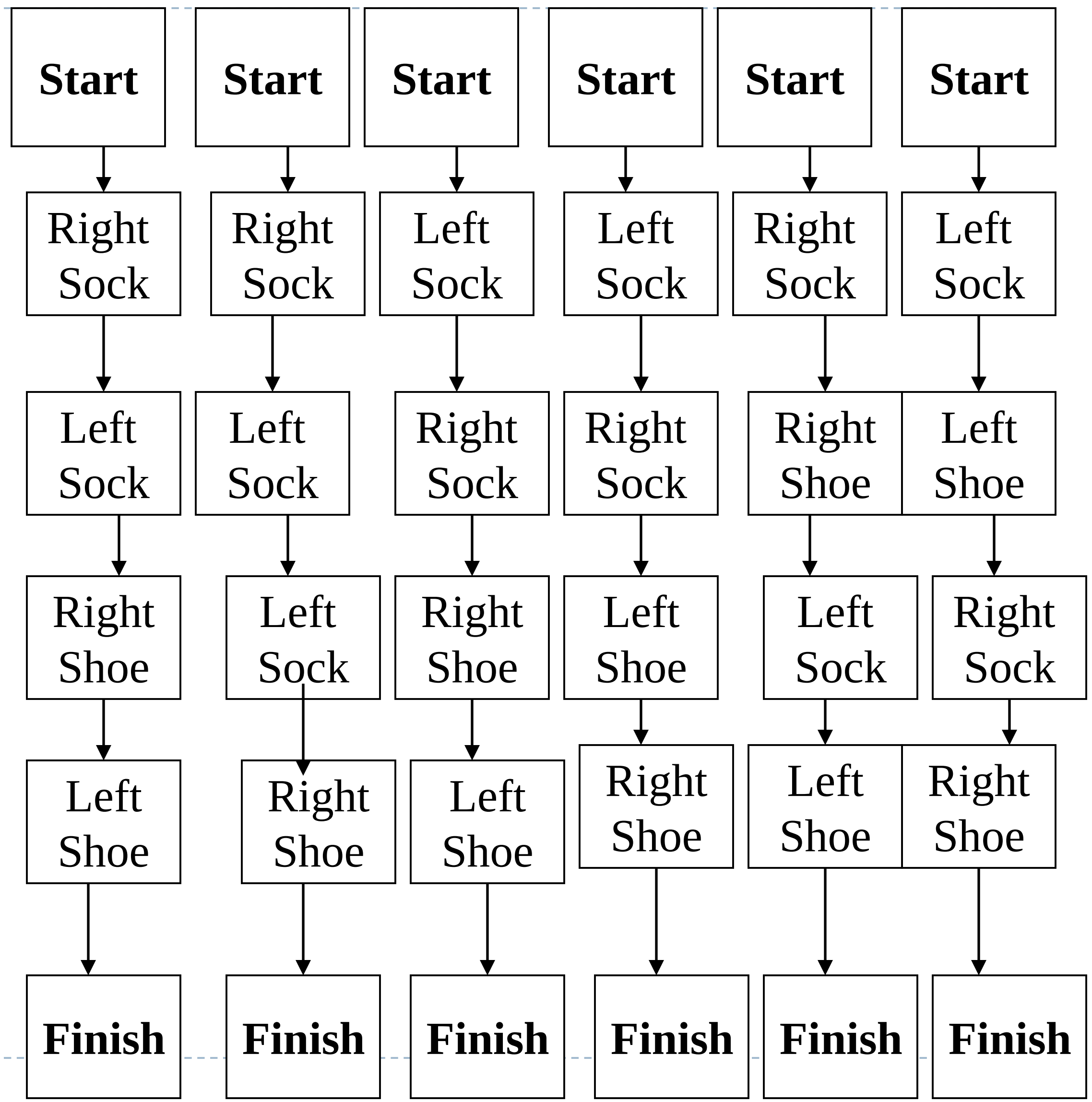
$Action(LeftShoe,$
 $\text{PRECOND: } LeftSockOn,$
 $\text{EFFECT: } LeftShoeOn)$

$Action(LeftSock,$
 $\text{EFFECT: } LeftSockOn)$

Partial Order Plans:

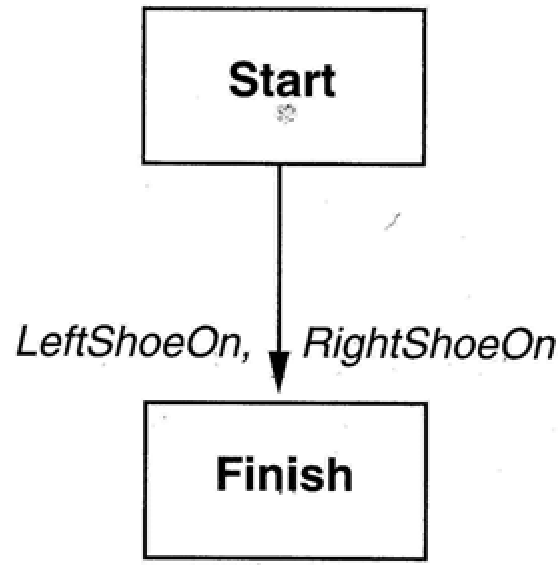



Total Order Plans:



# Partial Order Planning

- Two initial actions
  - Start
    - No precondition
    - All ‘Initial State’ as its effects
  - Finish
    - All ‘Goal State’ as its precondition
    - No Effect



# Partial plan definition

- Partial plan is a $< A, O, L >$ where:
  - $A$: set of actions in the plan (plan steps)
    - Initially {Start, Finish}
  - $O$: set of orderings between actions
    - Initially {Start<Finish}
  - $L$: set of causal links
    - Initially {}

# Causal links and threats

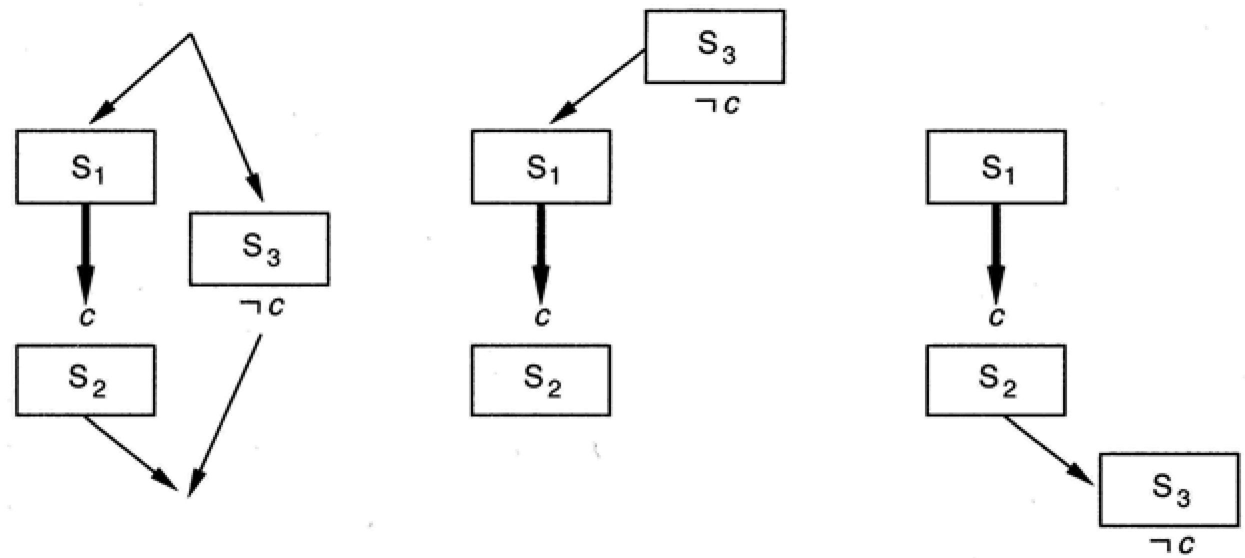
- Causal Link: serve to record the purpose of steps in the plan
  - Purpose of $A_i$ is to achieve the precondition $c$ of $A_j$

$$A_i \xrightarrow{\quad c \quad} A_j$$

- Threat: causal links are used to detect when a newly introduced action interferes with past decisions.

- $A_k$ <u>threatens</u> $A_i \overset{c}{\rightarrow} A_j$ when:
  - $A_k$ can become between $A_i$ and $A_j$ ($O \cup \{A_i < A_k < A_j\}$ is consistent)
  - $A_k$ has $\neg c$ as an effect.

# Resolving Threats




- Resolve Threat: ensuring that threats are ordered to come before or after the protected link
  - **Demotion** (placed before): add $S_3 < S_1$ to $O$
  - **Promotion** (placed after): add $S_2 < S_3$ to $O$

# Spare tire example

$$Init\big(Tire(Flat) \wedge Tire(Spare) \wedge At(Flat, Axle) \wedge At(Spare, Trunk)\big)$$

$$Goal(At(Spare, Axle))$$

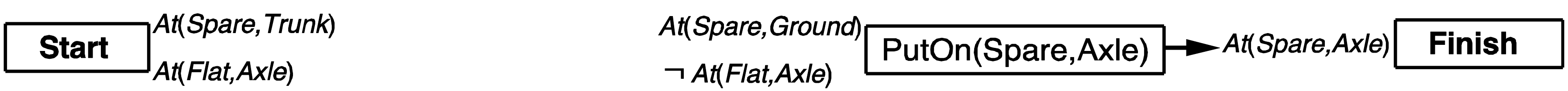
$$\begin{aligned}&Action(Remove(obj, loc),\\ &\qquad PRECOND{:}\, At(obj, loc),\\ &\qquad EFFECT{:}\ \neg At(obj, loc) \wedge At(obj, ground))\end{aligned}$$

$$\begin{aligned}&Action(PutOn(t, axle),\\ &\qquad PRECOND{:}\, Tire(t) \wedge At(t, Ground) \wedge \neg At(Flat, Axle)\\ &\qquad EFFECT{:}\ \neg\, At(t, Ground) \wedge At(Flat, Axle))\end{aligned}$$

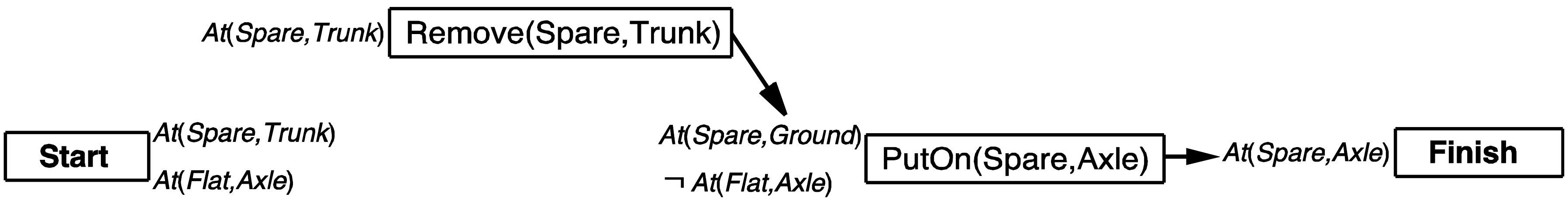
$$\begin{aligned}&Action(LeaveOvernight,\\ &\qquad PRECOND{:}\\ &\qquad EFFECT{:}\ \neg At(Spare, Ground) \wedge \neg At(Spare, Trunk) \wedge \neg At(Spare, Axle)\\ &\qquad\qquad\quad \wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Trunk) \wedge \neg At(Flat, Axle))\end{aligned}$$

# Spare tire example

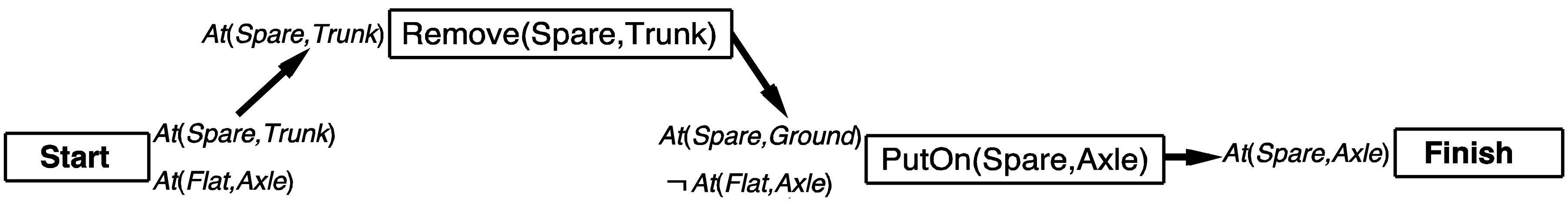
# Spare tire example



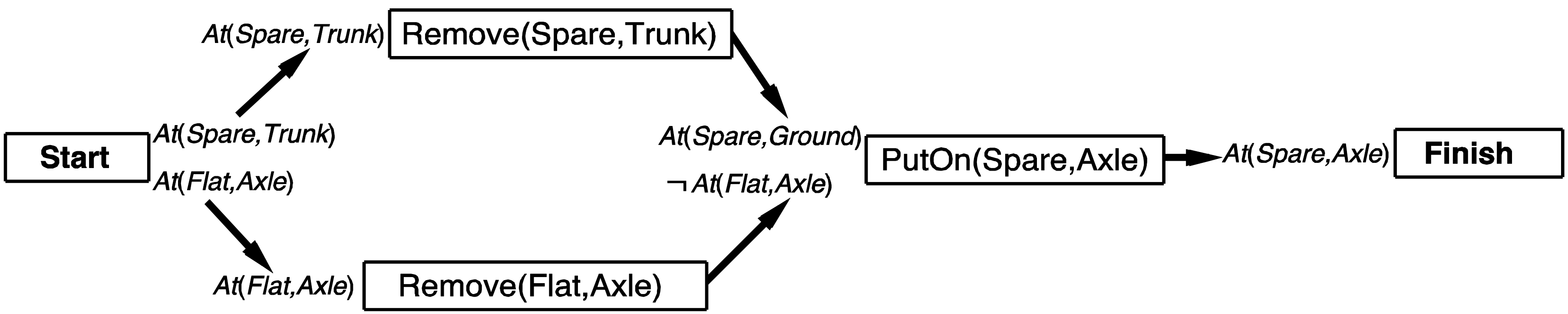
# Spare tire example



# Spare tire example



# Spare tire example



# POP

- Agenda: **open preconditions** (along with actions requiring them)
  - Initially all preconditions of End

**function** POP($< A, O, L >, agenda$)

**if** $agenda = \{\}$ **then return** ($< A, O, L >$)

$(q, A_{need}) \leftarrow$ Select a goal from agenda

$a \leftarrow$ Choose an action that adds $q$

**if** no such action **then return** failure

Update $< A, O, L >$ and $agenda$

Add consistent ordering constraints for causal link protection

**if** no constraint is consistent **then return** failure

POP($< A, O, L >, agenda$)

# POP algorithm (more details)

**POP**($<A, O, L>$, *agenda*)

**1. Termination:** If *agenda* is empty **return** $<A, O, L>$

**2. Goal selection:** Let $<Q, A_{need}>$ be a pair on the agenda

**3. Action selection:** Let $A_{add}$ = choose an action that adds $Q$
**if** no such action exists, **then return failure**
**Let** $L' = L \cup \{A_{add} \xrightarrow{Q} A_{need}\}$, and **let** $O' = O \cup \{A_{add} < A_{need}\}$.
**If** $A_{add}$ is newly instantiated, **then** $A' = A \cup \{A_{add}\}$ and
$O' = O \cup \{A_0 < A_{add} < A_\infty\}$ (otherwise, let $A' = A$)

**4. Updating of goal set:** **Let** $agenda' = agenda - \{<Q, A_{need}>\}$.
**If** $A_{add}$ is newly instantiated, **then for each** conjunction, $Q_i$, of its precondition, add $<Q_i, A_{add}>$ to *agenda'*

**5. Causal link protection:** For every action $A_t$ that might threaten a causal link $A_p \xrightarrow{p} A_c$, add a consistent ordering constraint, either

(a) Demotion: Add $A_t < A_p$ to $O'$

(b) Promotion: Add $A_c < A_t$ to $O'$

**If** neither constraint is consistent, **then return failure**

**6. Recursive invocation:** POP(($<A', O', L'>$, *agenda'*)

# Shopping example

$Init(At(Home) \land Sells(HWS, Drill) \land Sells(SM, Milk), Sells(SM, Banana))$

$Goal(Have(Drill) \land Have(Milk) \land Have(Banana) \land At(Home))$

$Action(Go(there)$

PRECOND: $At(here)$,

EFFECT: $At(there) \land \neg At(here))$

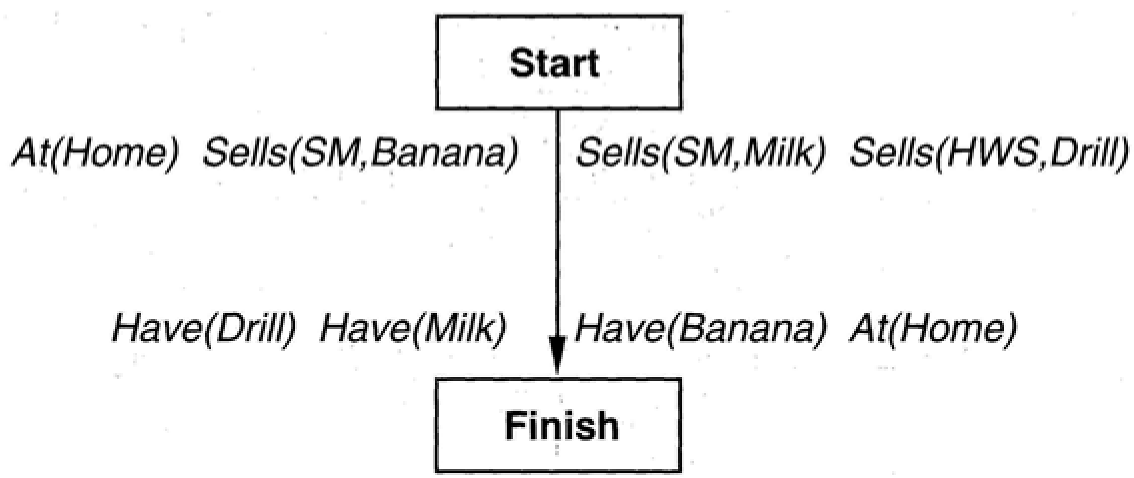
$Action(Buy(x)$,

PRECOND: $At(store) \land Sells(store, x)$,

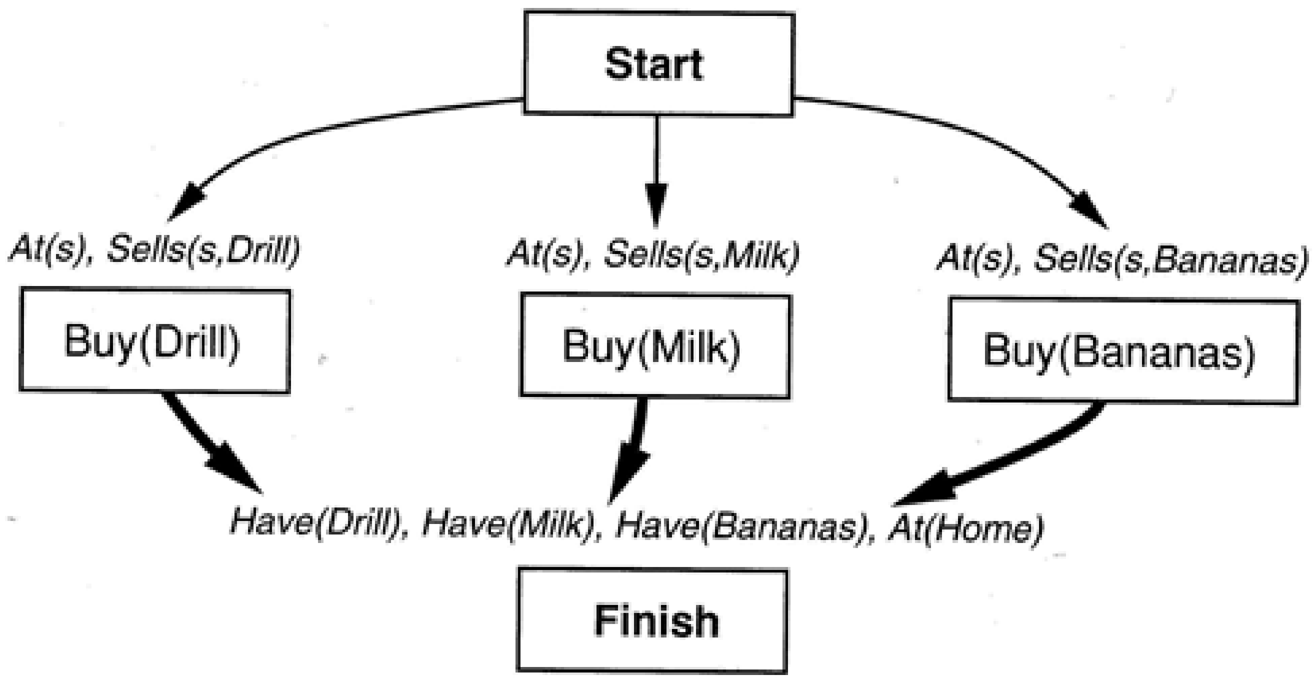
EFFECT: $Have(x))$

# Shopping example

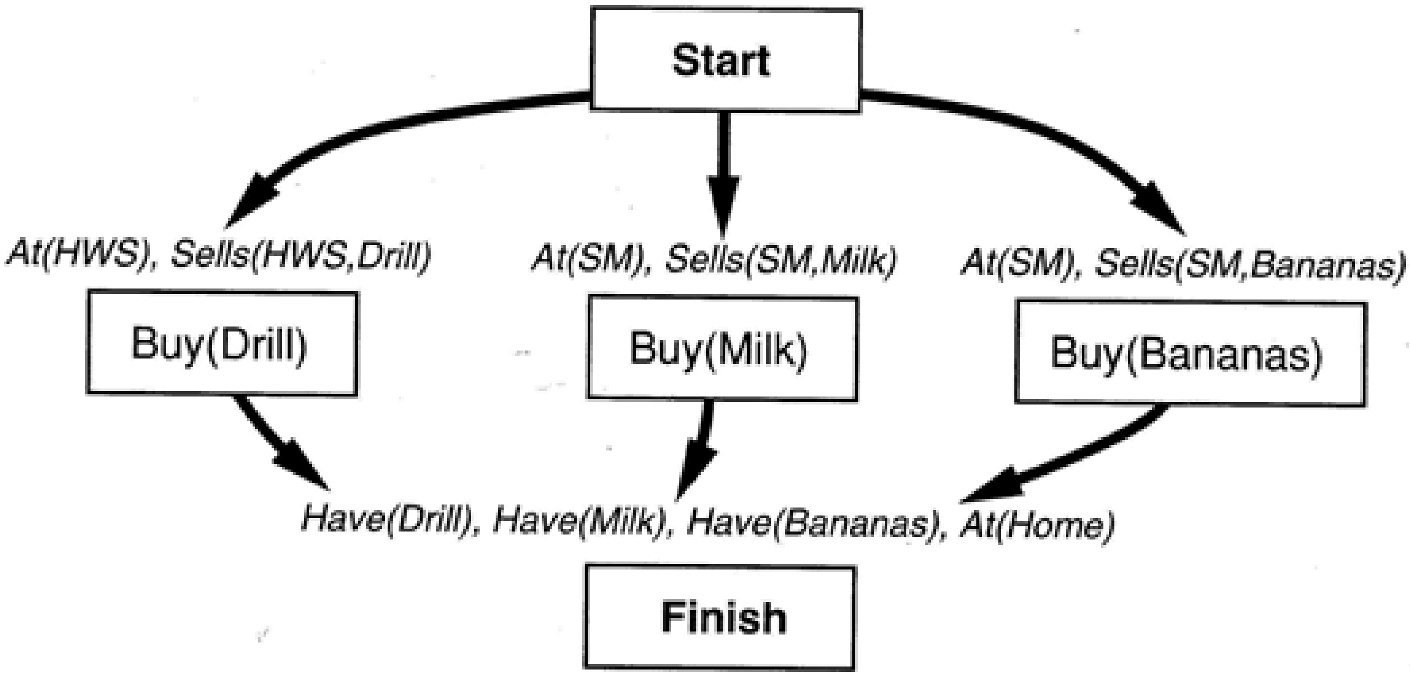
- Many possible ways to elaborate the initial plan
  - Three $Buy$ actions for three preconditions of Finish action
  - $Sells$ precondition of Buy
    - **Bold arrows**: causal links, protection of precondition
    - **Light arrows**: ordering constraints



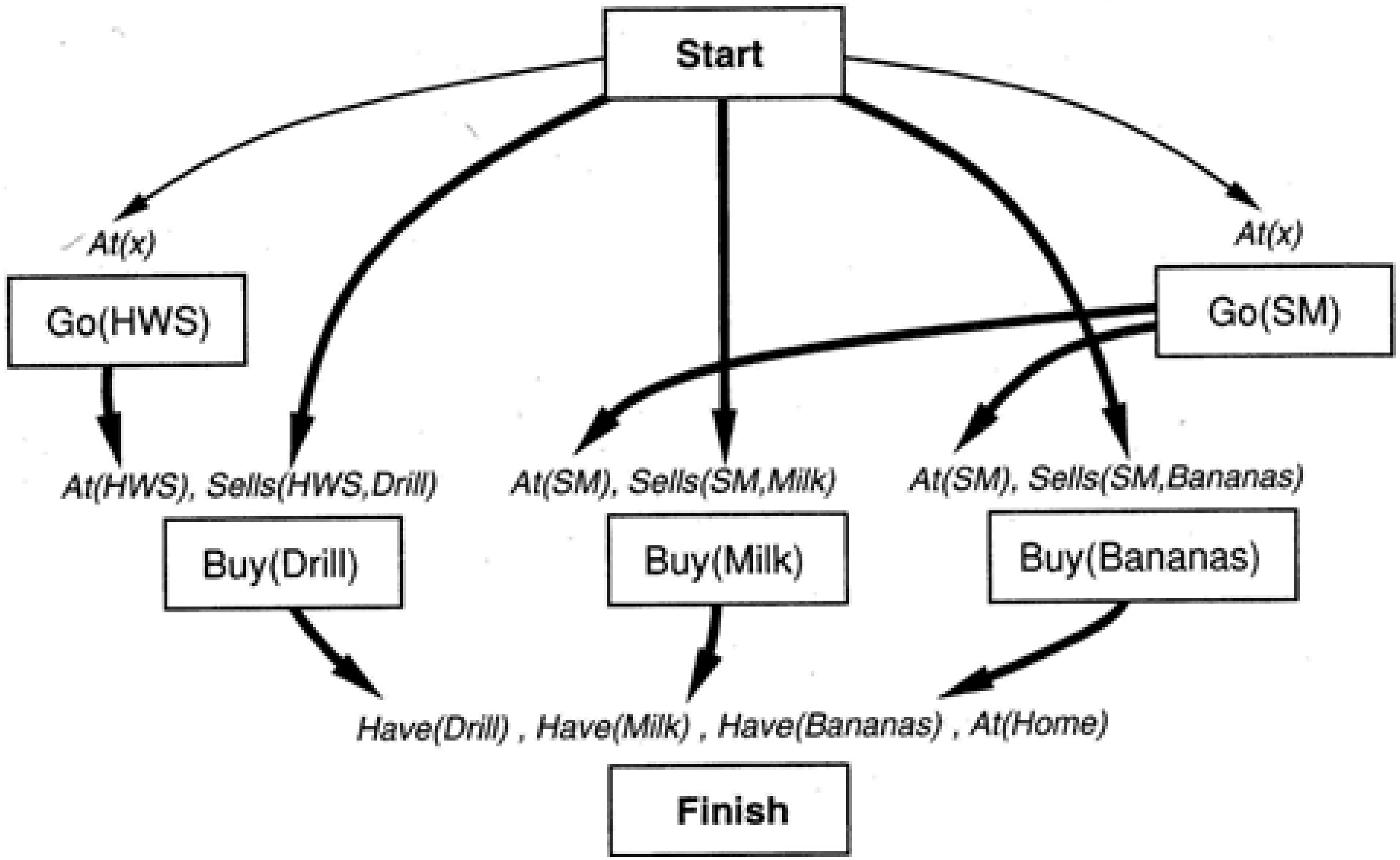
# Shopping example



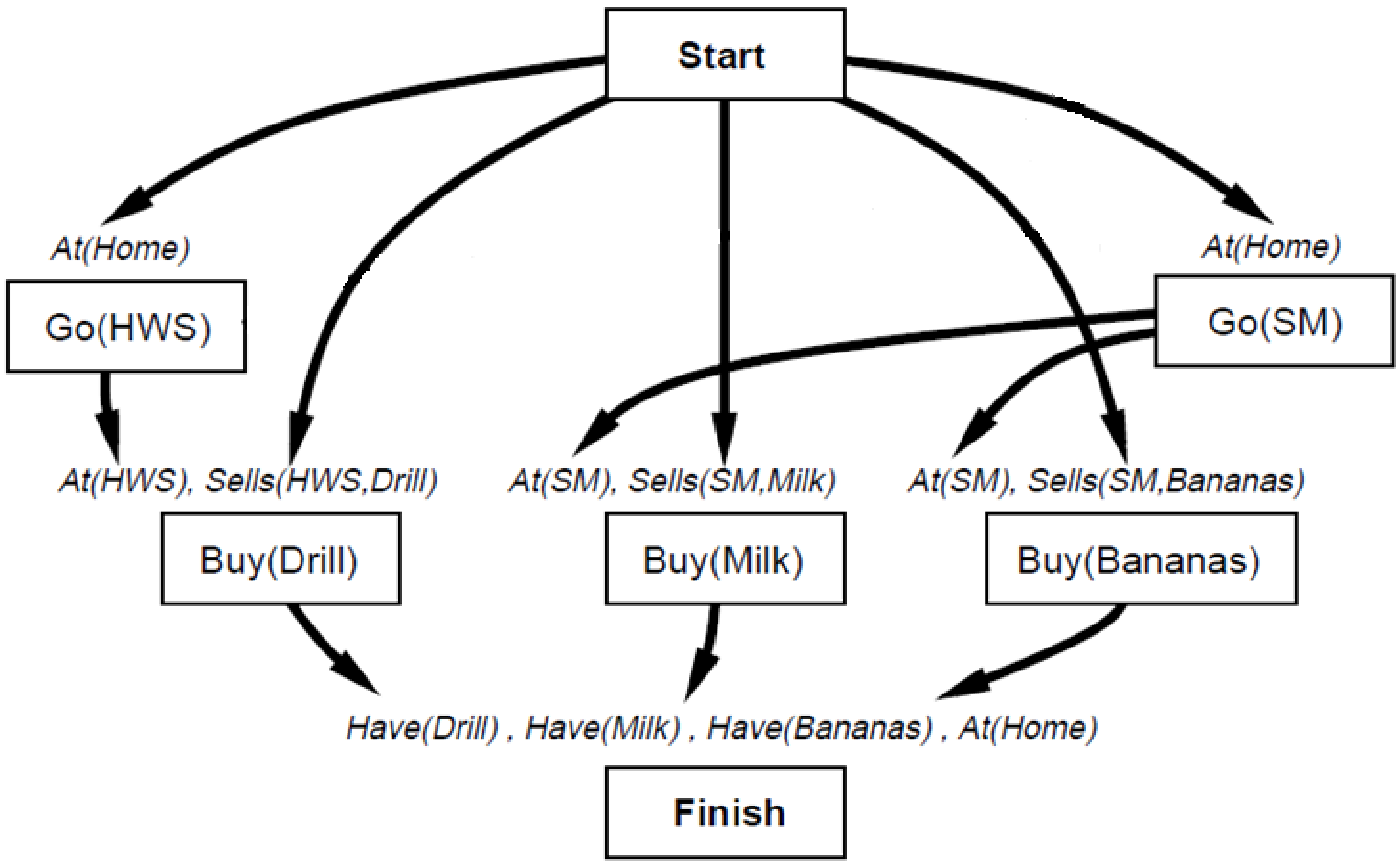
# Shopping example



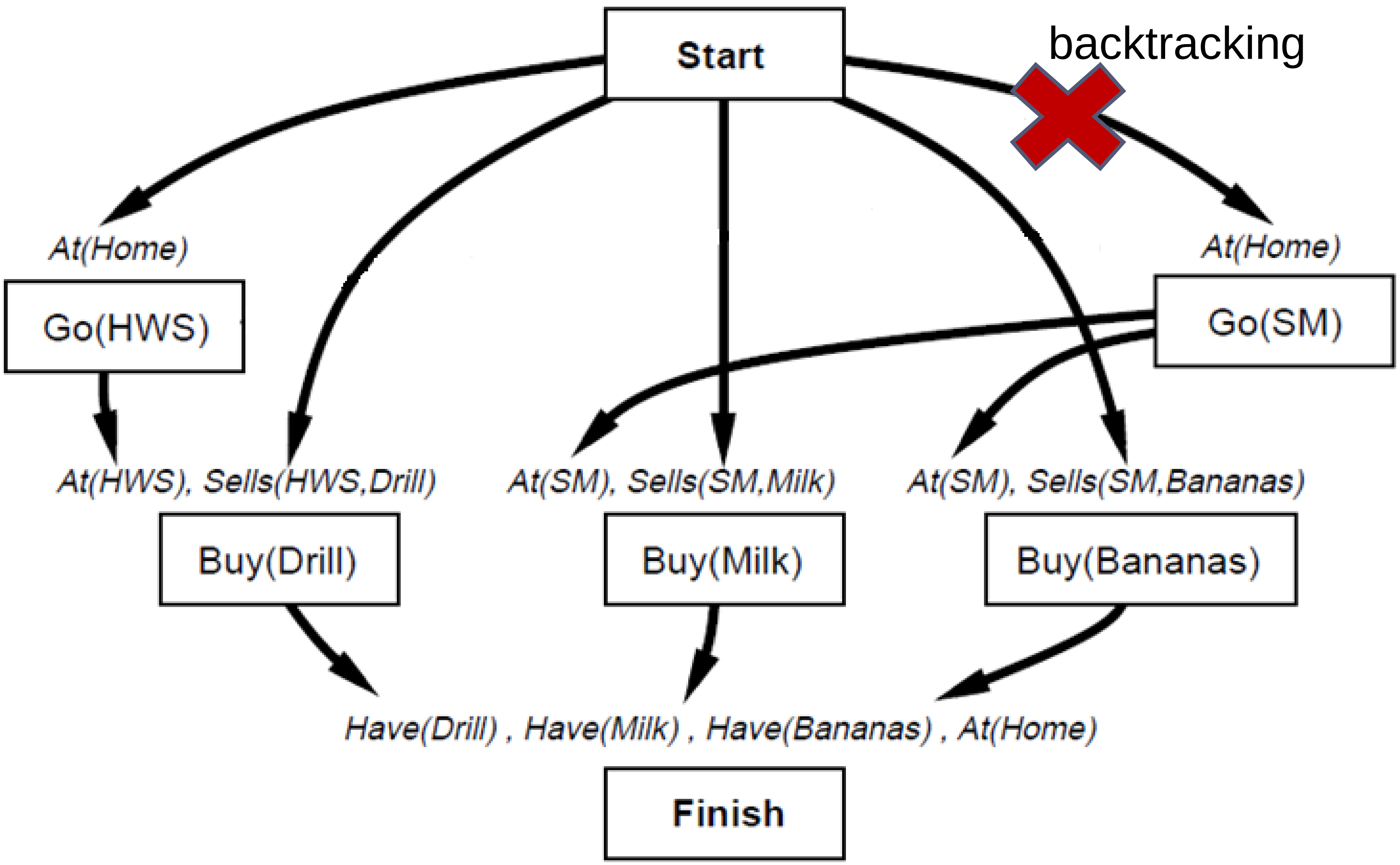
# Shopping example



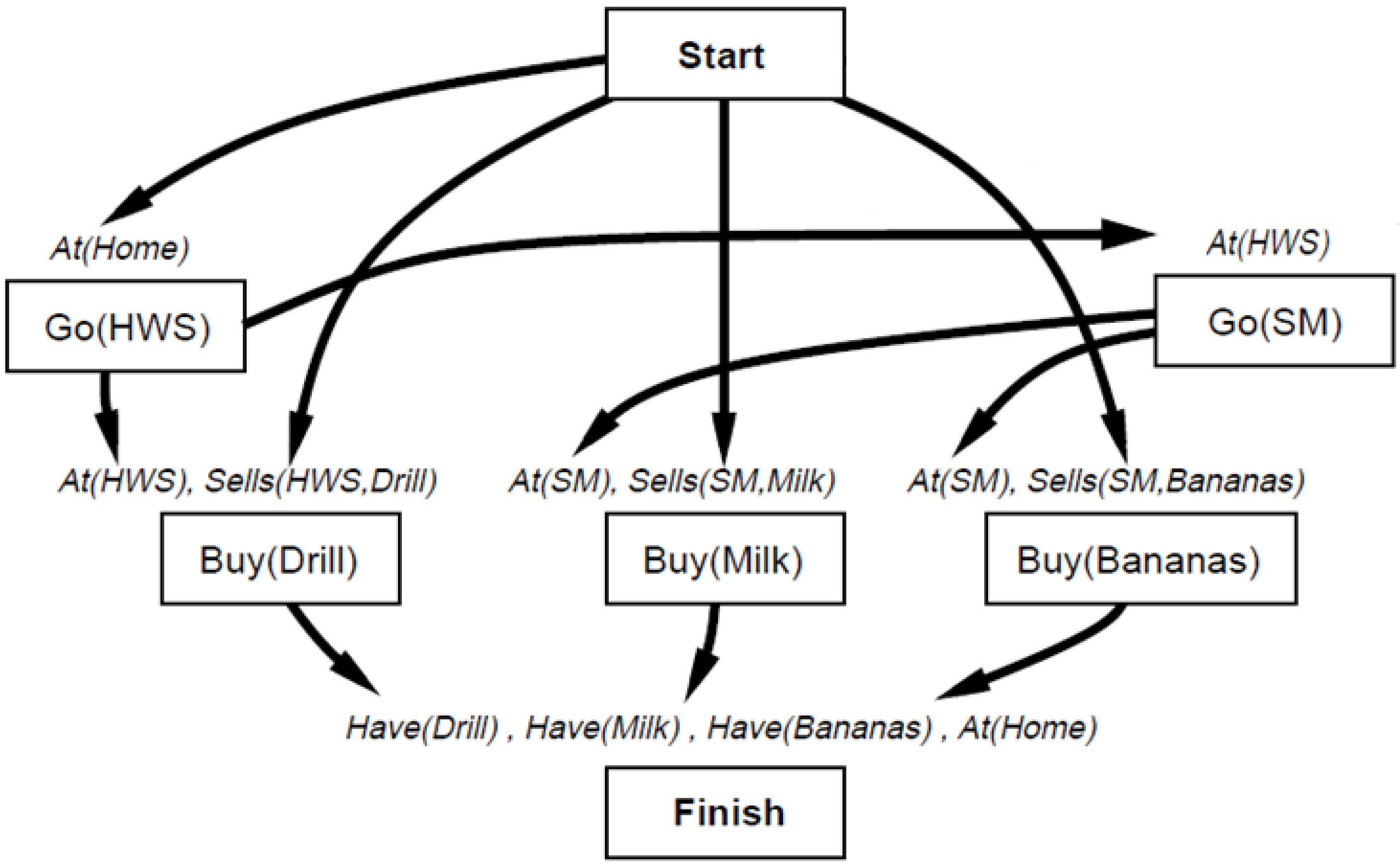
# Shopping example



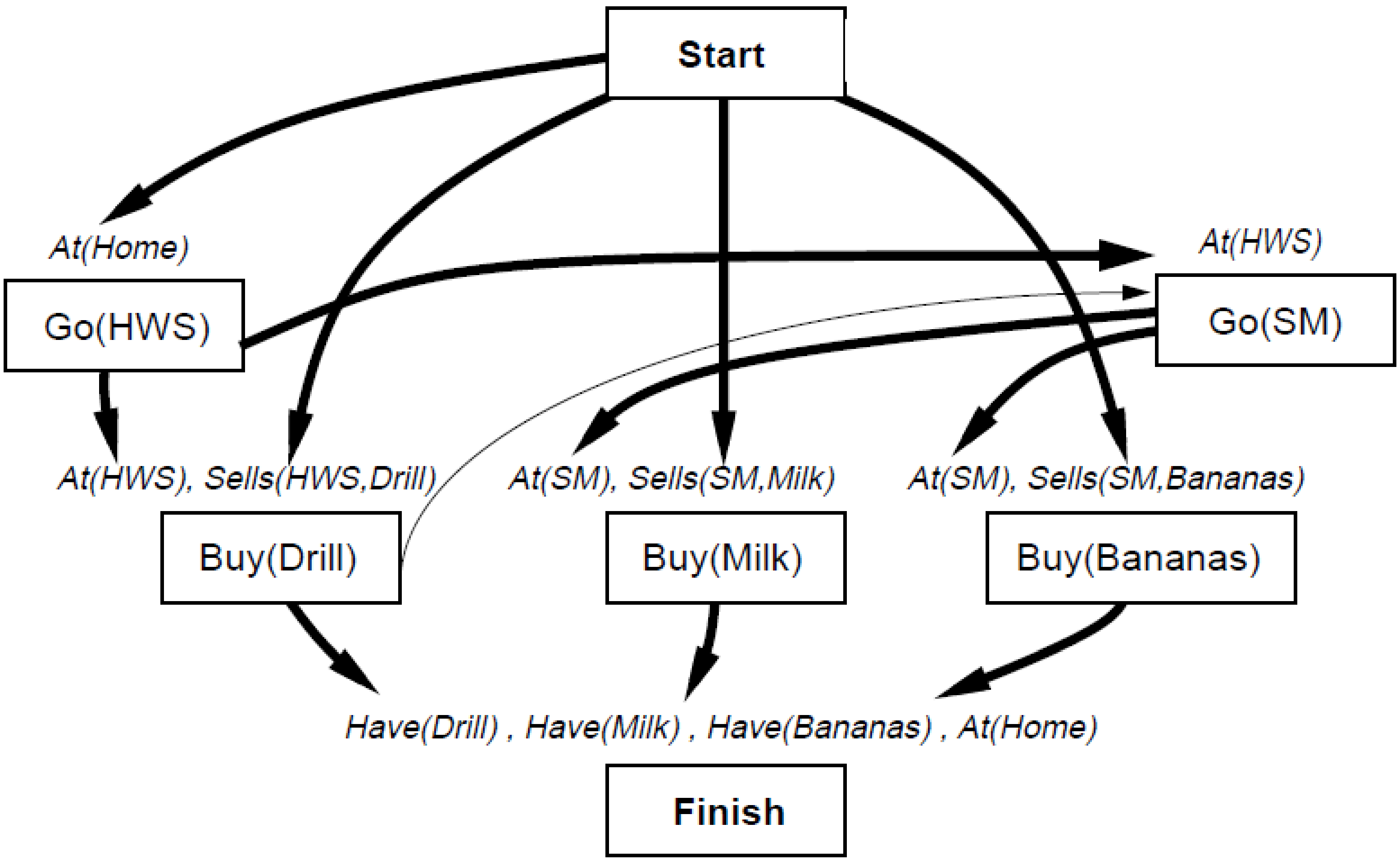
# Shopping example



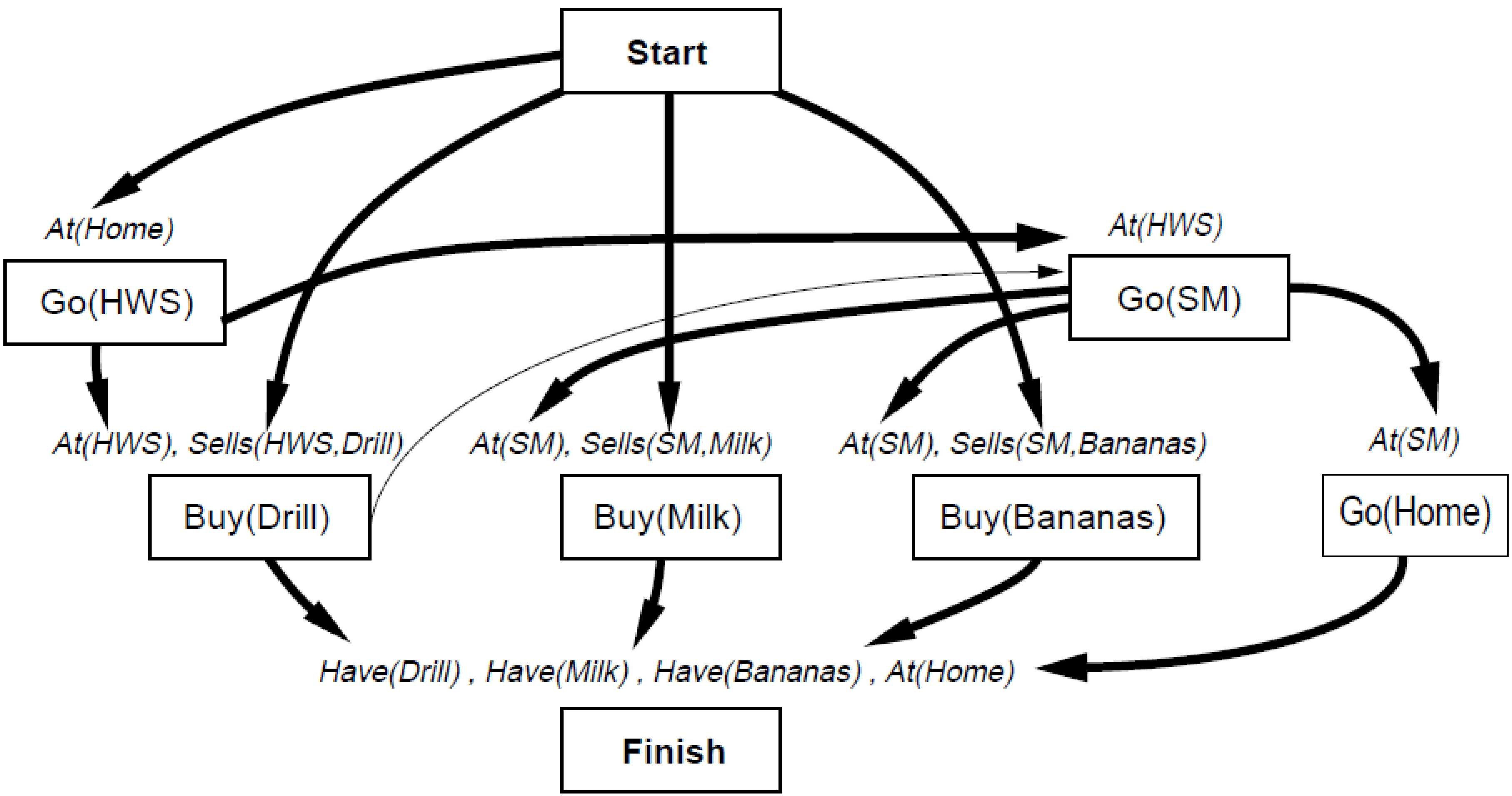
# Shopping example



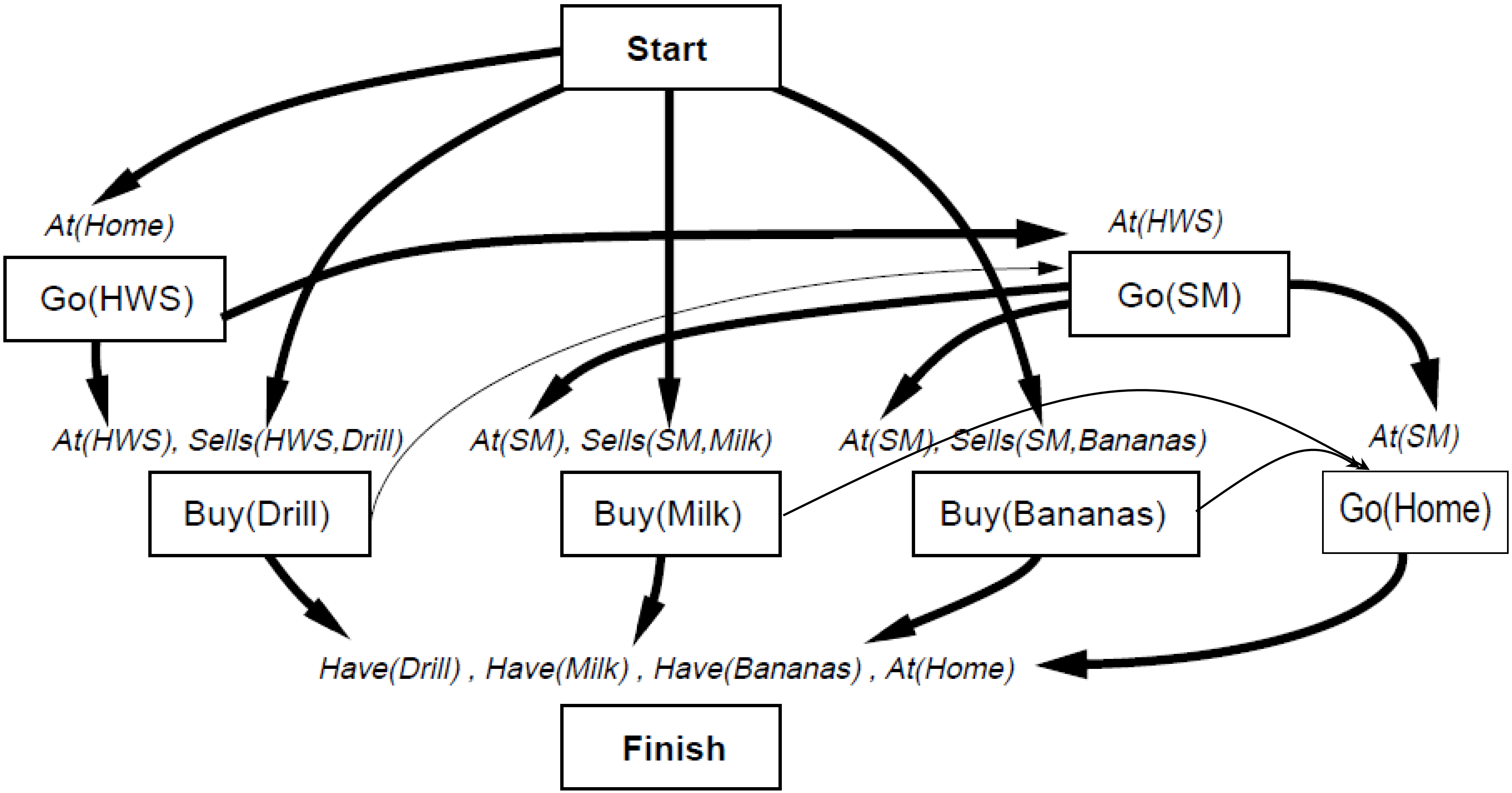
# Shopping example



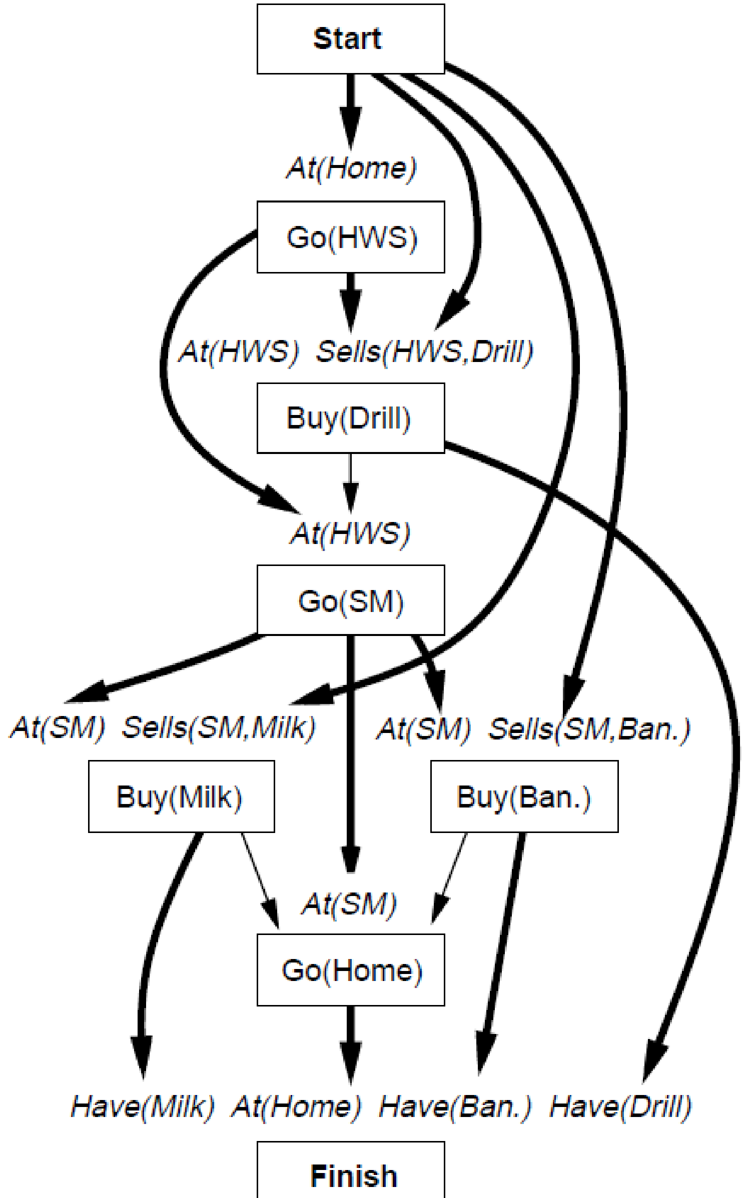
# Shopping example



# Shopping example



# Shopping example



# POP: advantages and disadvantages

- ☑ Least commitment may lead to smaller branching factor
- ☑ Postpone instantiating actions (postpone binding values to variables until necessary): e.g., Move(x,B,y) when needing Clear(B).
- ☑ More human-like plan
- ☒ More complex algorithm than recent planning algorithms
  - higher computation per-node
- ☒ Harder to find proper heuristics for all types of choices
  - Action selection, goal selection, order refinement
- ☒ How to prune infinite long paths?

# Planning advantages

- Planning models are more efficient:

1) Clear action and goal representation to allow selection
2) Problem decomposition by sub-goaling

   some goals are independent of most other parts, thus we can use divide-and-conquer strategy

3) Requirement relaxation for sequential construction of solutions

# Summary

- GraphPlan: winner of 1998 contest
- SATPlan: winner of 2004, 2006 contest
- POP (introduced in mid 1970's): not competitive to GraphPlan and SATPlan
  - Partially ordered plans
  - Can generate more human-like plans that can be checked by human operators