# Planning

## Lecture 1

# Outline

♢ The planning problem, STRIPS operators and planning specifications (RN 11.1)

♢ Planning in the state-space (RN 11.2)

♢ Partial-order planning (RN 11.3)

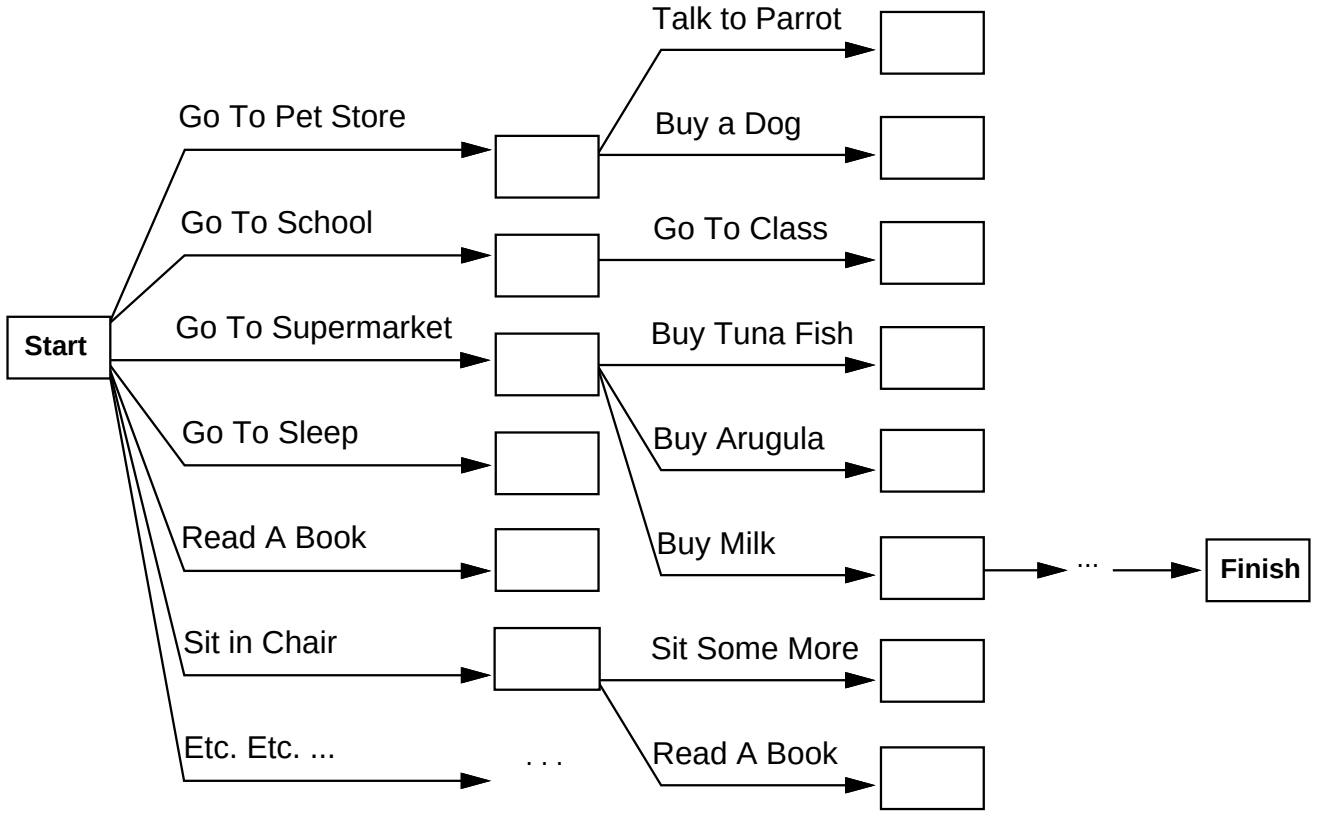
# Classic Planning

Environment:

- ♢ fully observable
- ♢ deterministic
- ♢ static
- ♢ discrete (and finite)

# Search vs. planning

Consider the task *get milk, bananas, and a cordless drill*
Standard search algorithms seem to fail miserably:



# Search vs. planning contd.

Planning systems do the following:

1) open up action and goal representation to allow selection
2) divide-and-conquer by subgoaling
3) relax requirement for sequential construction of solutions

| | **Search** | **Planning** |
|---|---|---|
| **States** | (Lisp) data structures | Logical sentences |
| **Actions** | (Lisp) code | Preconditions/outcomes |
| **Goal** | (Lisp) code | Logical sentence (conjunction) |
| **Plan** | Sequence from $S_0$ | Constraints on actions |

# STRIPS

STRIPS: STanford Research Institute Problem Solver, "restricted" language for action description:

States are represented by **sets of literals**:

- ♢ with **closed-world assumption**
- ♢ function free (finite domain)

Init state:

$$At(Home) \wedge \neg Has(Milk) \wedge \neg Has(Bananas) \wedge \neg Has(Drill) \wedge \cdots$$

Goal:

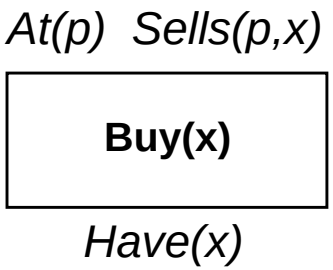
$$At(Home) \wedge Has(Milk) \wedge Has(Bananas) \wedge Has(Drill)$$

"restricted" language aims at ad hoc efficient algorithms

# Action Representation

Precondition: conjunction of positive literals
Effect: conjunction of literals

ACTION: $Buy(x)$




PRECONDITION: $At(p), Sells(p, x)$
EFFECT: $Have(x)$

Note 1: variables can be instantiated to a finite number of individuals.
Note 2: effects specify literals to be ADDed or REMOVEd from state.

# Persistence and plans

STRIPS solves the **frame** problem by assuming that:

Everything not explicitely changed by effects persists.

STRIPS solves the **qualification** problem as Sitcalc.

A complete set of STRIPS operators can be translated into a set of **successor-state** axioms.

Planning:

A **plan** is a sequence of actions that allows us to reach a state where the goal condition holds, starting from the initial state.

# Semantics of STRIPS

- Sitcal
- Operational

Note: even though states are represented as logical formulae, STRIPS does not provide a logical account of how the state changes!

# Other planning languages

ADL: Action Description Language:

- Open world assumption in the state representation
- Quantified sentences as goals
- Richer language for expressing conditions (sorts, inequalities, ...)

PDDL: Planning Domain Description Language

- de facto standard plan specification language
- generalizes both STRIPS and ADL
- several extensions to express non deterministic actions, sensing, ...

# PDDL: example

```
(define (domain cleaning_domain)
   (:action goto_R
       :precondition location_C
       :effect location_R
       :not-inertial (location_C))
   (:action pickup_O
       :precondition (and (not has_O)
                          (or (and location_R in_O_R)
                              (and location_X in_O_X)) )
       :effect has_O
       :not-inertial (in_O_R))
   (:action scan_R
       :precondition location_R
       :sense in_O_R))
```

# Planning techniques

- Search in the state space
- Search in the plan space
- GRAPHPLAN for heuristics and plan generation
- Planning as propositional satisfiability

# Search in the state space

## Representation:

- states are characterized by sets of propositions (by a formula)
- operators are determined by action specifications and persistence
- final state must satisfy the goal
- step cost = 1 (usually)

## Search (for example using A*):

- progression
- regression
- heuristics

# Search

In progression **irrelevant** actions cause the search space to blow-up.

Backward search (regression):

♢ STRIPS operators are easily **invertible**

♢ focusses on the **relevant** actions (making true some of the goal conjuncts)

♢ discards **inconsistent** actions (making false some of the goal conjuncts)

Naive regression performs better than naive progression but they are both unsatisfactory.

# Heuristics for state space search

The basic idea for heuristics is to find a good estimate of the distance to the goal.

Heuristics can be found by:

1. **relaxing the problem**: admissible heuristics as simplified problems (e.g. removing preconditions, removing negated effects (empty-delete-list) ...);
2. **subgoal independence assumption**: the cost of solving a conjunction is the sum of the costs of solving each of the conjuncts (when pessimistic is not admissible).

State space planners won the AIPS 2000 competition, thus renewing the interest for their practical applicability.

# Partially ordered plans

*Partially ordered* collection of steps with

- $Start$ step has the initial state description as its effect
- $Finish$ step has the goal description as its precondition
- causal links from outcome of one step to precondition of another
- temporal ordering between pairs of steps

Principle of **least committment**:

- partial ordering (instead of total)
- not fully instantiated plan (in the first-order case)

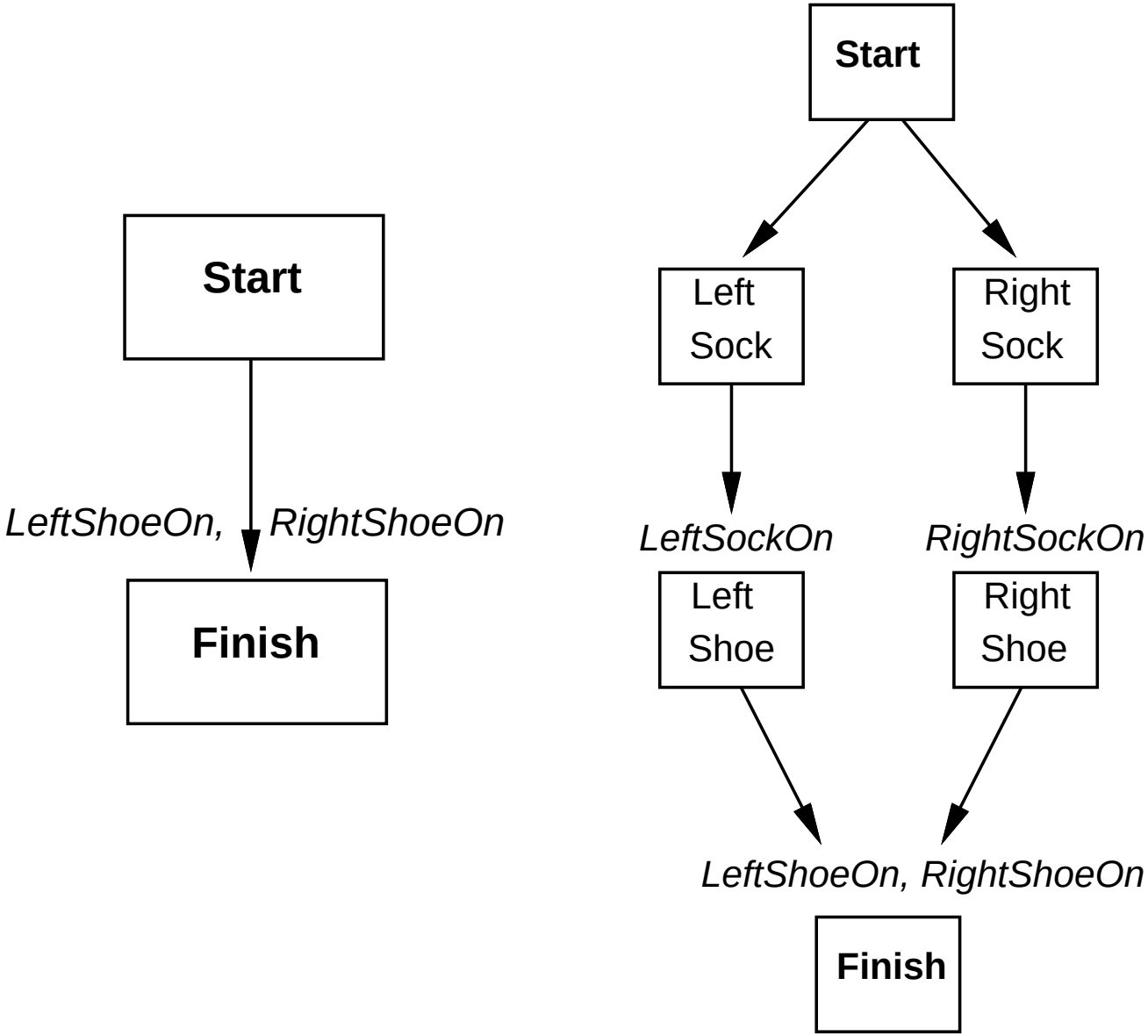
# State space vs plan space

state space: node = state in the world
plan space: node = partial plan

Open condition = precondition of a step not yet causally linked

A plan is complete iff every precondition is achieved

A precondition is achieved iff:
it is the effect of an earlier step and no possibly intervening step undoes it


Start
LeftShoeOn, RightShoeOn
Finish
Start
Left Sock
Right Sock
LeftSockOn
RightSockOn
Left Shoe
Right Shoe
LeftShoeOn, RightShoeOn
Finish

# Plan Representation

- set of steps
- set of ordering constraints $A \prec B$
- set of causal links $A \xrightarrow{p} B$

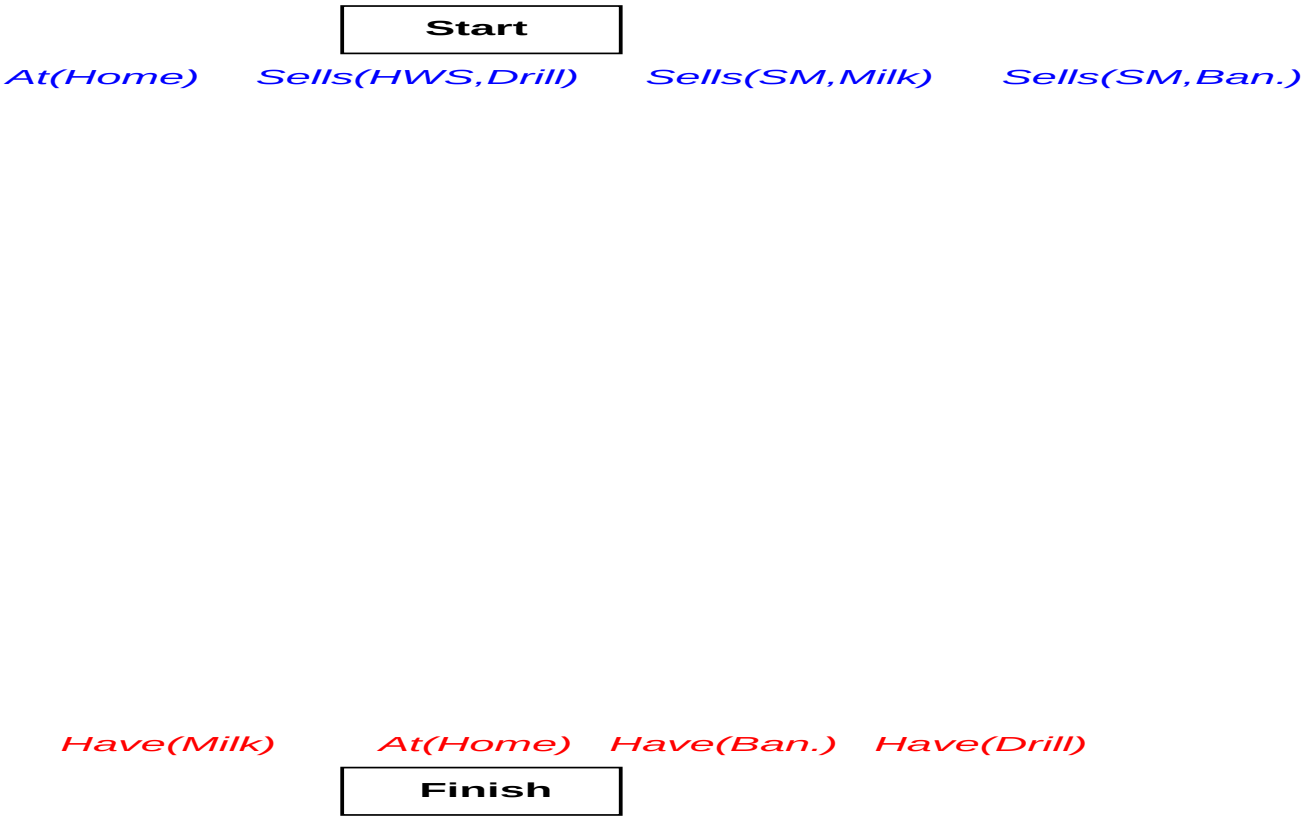
  $A$ achieves $p$ for $B$
- set of open preconditions

$$
\begin{aligned}
Plan(&\textsc{Steps}:\{\, S_1\!:Op(Start),\\
&\qquad S_2\!:Op(End,\\
&\qquad\quad \textsc{Precond}:RightShoeOn \wedge LeftShoeOn)\},\\
&\textsc{Orderings}:\{S_1 \prec S_2\},\\
&\textsc{Links}:\{\},\\
&\textsc{Open Preconditions}:\{RightShoeOn, LeftShoeOn\})
\end{aligned}
$$

# Procedure

1. The initial plan includes the plan constraints for $Start$ and $Finish$, with ordering $Start \prec Finish$;
2. The successor function
   (a) pick one open precondition $p$ on action $B$
   (b) pick one action $A$ that achieves $p$
   (c) add the causal link $A \xrightarrow{p} B$ and the ordering constraint $A \prec B$; if $A$ is new add also $Start \prec A$ and $B \prec Finish$
   (d) resolve conflicts, if possible, otherwise backtrack
3. The goal test succeeds when there are no more open preconditions

# Example

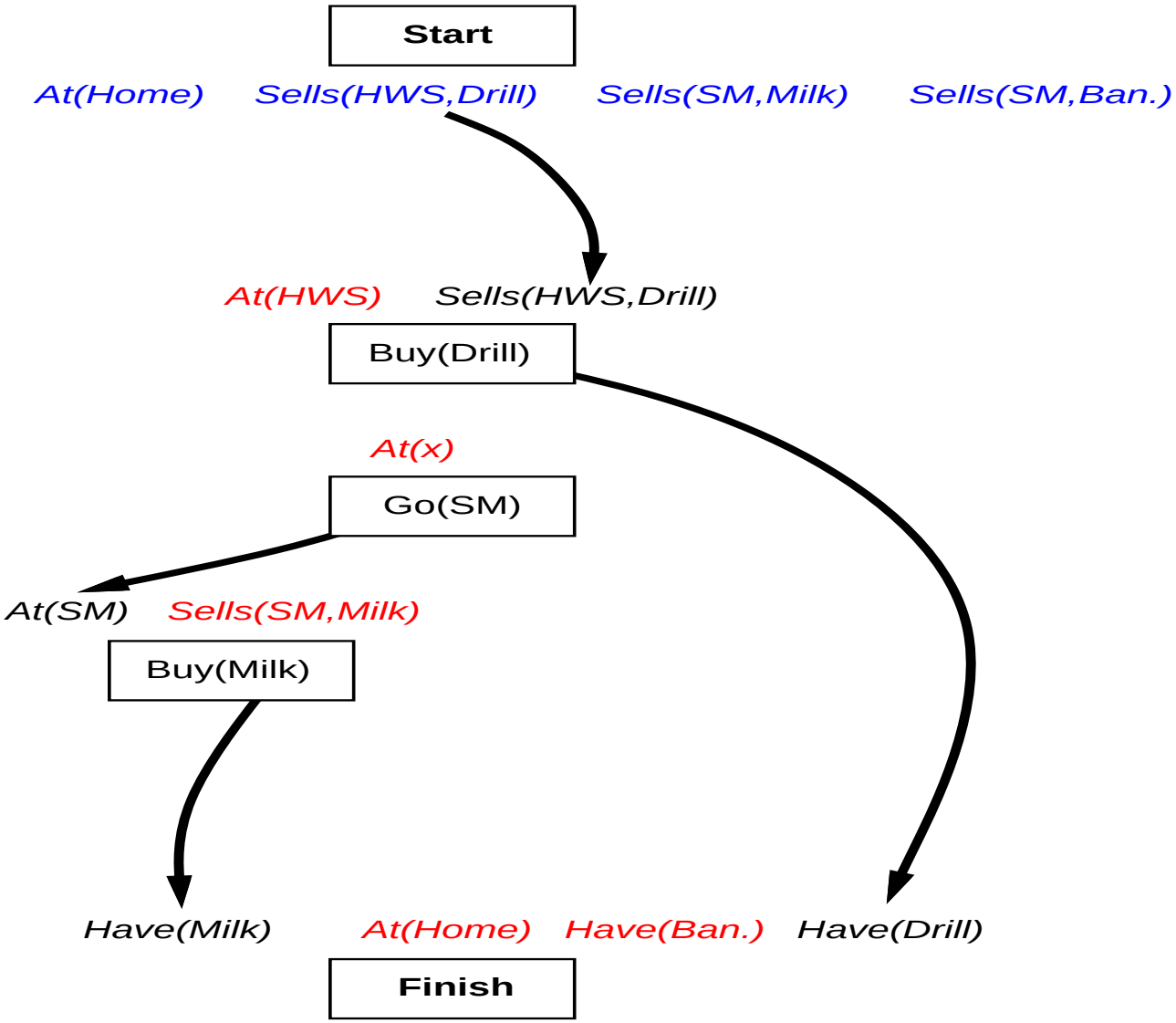

Start
At(Home)
Sells(HWS,Drill)
Sells(SM,Milk)
Sells(SM,Ban.)
Have(Milk)
At(Home)
Have(Ban.)
Have(Drill)
Finish

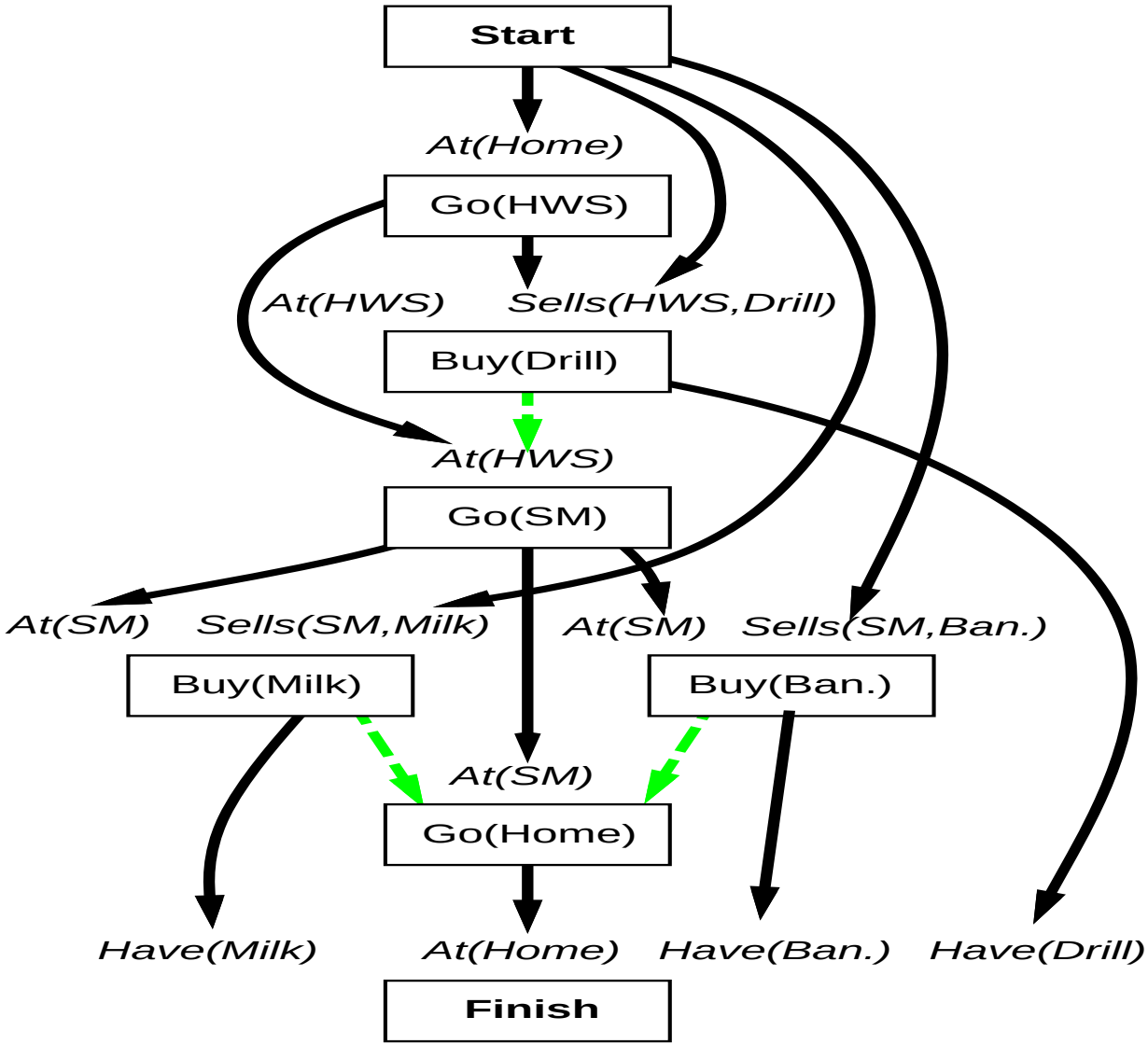
# Actions for the example

ACTION: $Buy(x)$
PRECONDITION: $At(p), Sells(p, x)$
EFFECT: $Have(x)$

ACTION: $Go(x)$
PRECONDITION: $At(y)$
EFFECT: $At(x) \wedge \neg At(y)$

Objects: $Milk, Bananas, Drill, \ldots$
Places: $Home, SM, HWS, \ldots$


Start
At(Home)
Sells(HWS,Drill)
Sells(SM,Milk)
Sells(SM,Ban.)
At(HWS)
Sells(HWS,Drill)
Buy(Drill)
At(x)
Go(SM)
At(SM)
Sells(SM,Milk)
Buy(Milk)
Have(Milk)
At(Home)
Have(Ban.)
Have(Drill)
Finish


Start
At(Home)
Go(HWS)
At(HWS)
Sells(HWS,Drill)
Buy(Drill)
At(HWS)
Go(SM)
At(SM)
Sells(SM,Milk)
At(SM)
Sells(SM,Ban.)
Buy(Milk)
Buy(Ban.)
At(SM)
Go(Home)
Have(Milk)
At(Home)
Have(Ban.)
Have(Drill)
Finish

# Clobbering and conflicts

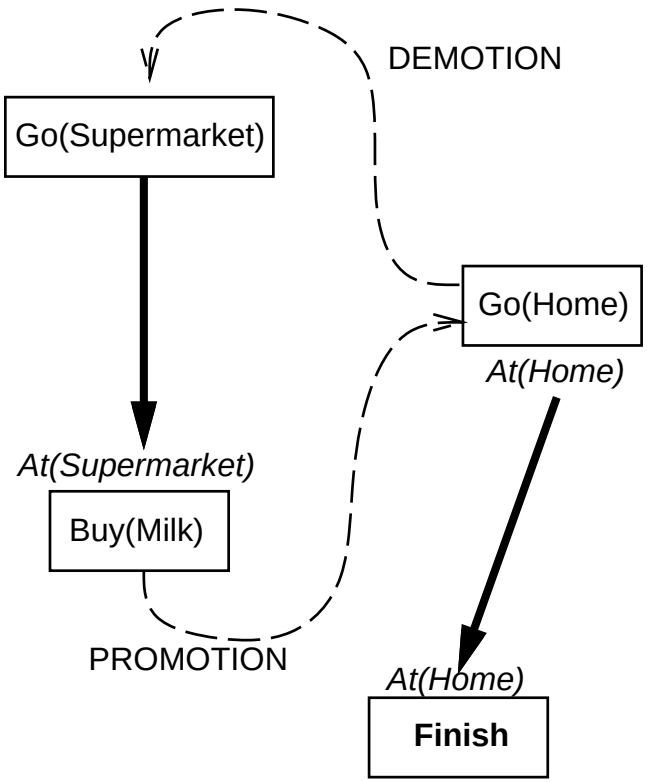
A clobberer is a potentially intervening step that destroys the condition achieved by a causal link. E.g., $Go(Home)$ clobbers $At(Supermarket)$:

More specifically, a **conflict** between the causal link $A \xrightarrow{p} B$ and the action $C$ holds when $C$ has effect $\neg p$.

A conflict can be solved by adding:

- $C \prec A$ (**demotion**) or
- $B \prec C$ (**promotion**)

# Promotion/demotion




Demotion: put before $Go(Supermarket)$

Promotion: put after $Buy(Milk)$

# POP algorithm sketch

**function** POP(*initial, goal, operators*) **returns** *plan*

*plan* ← MAKE-MINIMAL-PLAN(*initial, goal*)
**loop do**
  **if** SOLUTION?( *plan*) **then return** *plan*
  $S_{need}$, $c$ ← SELECT-SUBGOAL( *plan*)
  CHOOSE-OPERATOR( *plan, operators*, $S_{need}$, $c$)
  RESOLVE-THREATS( *plan*)
**end**

---

**function** SELECT-SUBGOAL( *plan*) **returns** $S_{need}$, $c$

pick a plan step $S_{need}$ from STEPS( *plan*)
  with a precondition $c$ that has not been achieved
**return** $S_{need}$, $c$

# POP algorithm contd.

**procedure** CHOOSE-OPERATOR($plan$, $operators$, $S_{need}$, $c$)

**choose** a step $S_{add}$ from $operators$ or STEPS( $plan$) that has $c$ as an effect
**if** there is no such step **then fail**
add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS( $plan$)
add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS( $plan$)
**if** $S_{add}$ is a newly added step from $operators$ **then**
    add $S_{add}$ to STEPS( $plan$)
    add $Start \prec S_{add} \prec Finish$ to ORDERINGS( $plan$)

# Properties of POP

Nondeterministic algorithm: backtracks at choice points on failure:

- choice of action ($S_{add}$) to achieve open precondition ($S_{need}$)
- choice of demotion or promotion for clobberer

Selection of open precondition ($S_{need}$) is irrevocable: the existence of a plan does not depend on the choice of the open preconditions.

POP is sound, and complete,

POP admits extensions for disjunction, universals, negation, conditionals

# Heuristics for POP

General:

- ♢ number of open preconditions
- ♢ most constrained variable

  open precond that are satisfied in fewest ways
- ♢ a special data structure: the **planning graph**

Problem Specific:

Good heuristics can be derived from problem description (by the human operator)

POP is particularly effective on problems with many loosely related subgoals