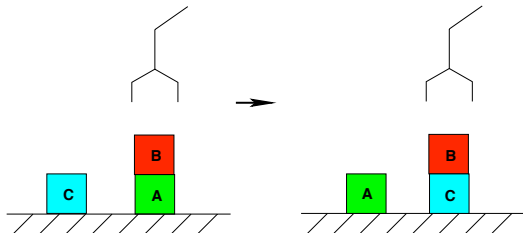# Planning
Introduction to Artificial Intelligence

G. Lakemeyer

Winter Term 2016/17

Planning

Given: A (logical) description of the initial state, a description of the goal state, a description of actions (preconditions and effects).

Problem: Find a plan involving these actions that takes you from the initial state to the goal state.

Why is Planning Different from Search?

- In contrast to states in search problems,
  states in planning need not be completely specified.
- Search treats states as black boxes.
  In planning one wants to look at the parts. E.g.: which block is free?
- Search generates all successor states.
  Planning only generates some.
- Search wants to find a sequence of actions leading to a goal.
  Planning looks for a description of a plan, e.g. actions may only be
  partially ordered.

© G. Lakemeyer

Why is Planning Different from Search?

There are too many actions to choose from. In general, impossible to generate all successor states.
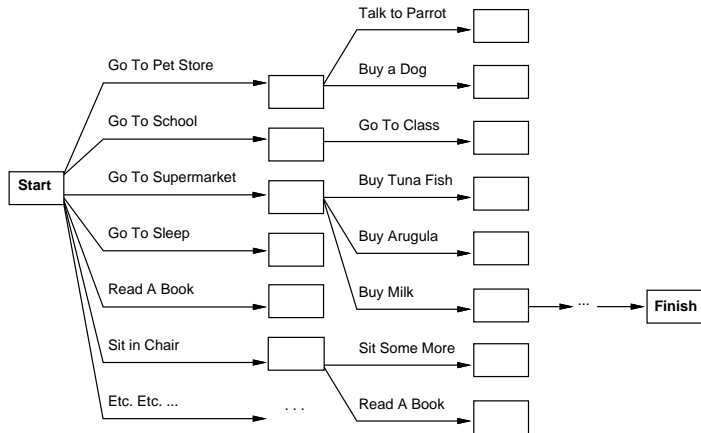
STRIPS Operators

STRIPS: STanford Research Institute Problem Solver
(Planner of the early Seventies. While STRIPS itself is no longer in use, its operator descriptions are.)

Actions are triples of the following form:

Action name:	Function name with parameters
Preconditions:	only positive literals
Effects:	positive und negative literals

In addition:

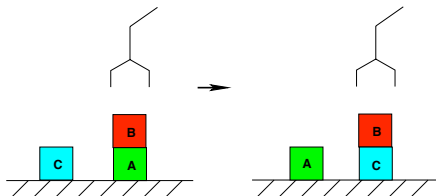|	
Initial State:	set of ground literals, no function symbols other than constants.
Goal State:	set of literals (possibly with free variables, implicitly existentially quantified)

Example Strips Operator

At(here), Path(here, there)

Go(there)

At(there), ¬ At(here)

```
Op(Action: Go(there),
   Precond: At(here) ∧ Path(here,there),
   Effect: At(there) ∧¬ At(here))
```

STRIPS Operators for the Blocks World


```
Op(Action: pickup(x,y),
   Precond: Block(x), On(x,y),
     Clear(x), Empty(hand),
   Effect: Holding(x), ¬Empty(hand), ¬Clear(x),
     ¬On(x,y), Clear(y))

Op(Action: puton(x,y),
   Precond: Block(y), Holding(x), Clear(y)
   Effect: ¬Holding(x), ¬Clear(y),
     On(x,y), Empty(hand), Clear(x))

Op(Action: putonTable(x),
   Precond: Block(x), Holding(x),
   Effect: ¬Holding(x), On(x,table),
     Empty(hand), Clear(x))
```

What is a Plan?

> Plan step $=$ STRIPS-Operator

A Plan consists of

- a set of partially ordererd ($\prec$) plan steps,
  where $S_i \prec S_j$ iff S_i must be executed before S_j.

- a set of variable assignments $x = t$,
  where x is a variable and t is a constant or a variable.

- a set of causal relations,
  where $S_i \xrightarrow{c} S_j$ means "S_i satisfies the precondition c for S_j."

Complete and Consistent Plans

Complete Plan:

Every precondition of every plan step is satisfied, that is:

$\forall S_j$ with $c \in Precond(S_j)$ $\exists S_i$ with $S_i \prec S_j$ and $c \in Effects(S_i)$

and for every linearization of the plan we have:

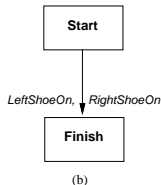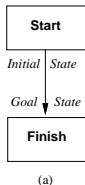$\forall S_k$ with $S_i \prec S_k \prec S_j, \neg c \notin Effects(S_k)$.

Consistent Plan:

If $S_i \prec S_j$ then $S_j \nprec S_i$ and if $x = A$ then $x \neq B$ for distinct A and B.
(Unique Names Assumption!)

A complete and consistent plan is called a solution.

Problem Description

Problem description = initial plan

(a)   (b)

```
Plan(Steps:
    S₁:Op(Action: Start),
    S₂:Op(Action: Finish)
        Precond: RightShoeOn ∧ LeftShoeOn)
    Orderings: {S₁ ≺ S₂}
    Bindings: {}
    Links: {})
```

Features of the Problem Description

- Initial state and goal state are encoded as STRIPS-operators.
- Plan step: take a plan step with ≥ 1 unsatisfied preconditions; insert a new plan step which satisfies one or more of these conditions. (Helps focus the search.)
- Decisions about order, variable assignments, etc. are delayed as long as possible.
- Leads to partially ordered plans.

Partially Ordered Plans

Partial Order Plan:
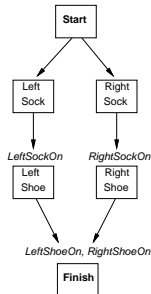
Total Order Plans:
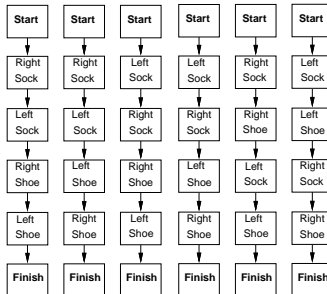

```
Op(Action: RightShoe,
   Precond: RightSockOn,
   Effect: RightShoeOn)

Op(Action: RightSock,
   Effect: RightSockOn)
```
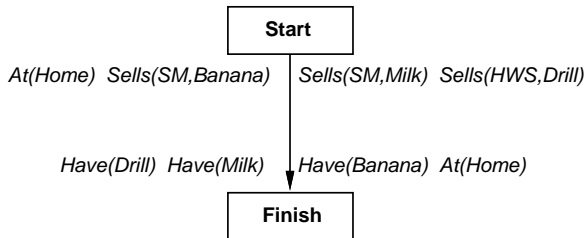
```
Op(Action: LeftShoe,
   Precond: LeftSockOn,
   Effect: LeftShoeOn)

Op(Action: LeftSock,
   Effect: LeftSockOn)
```
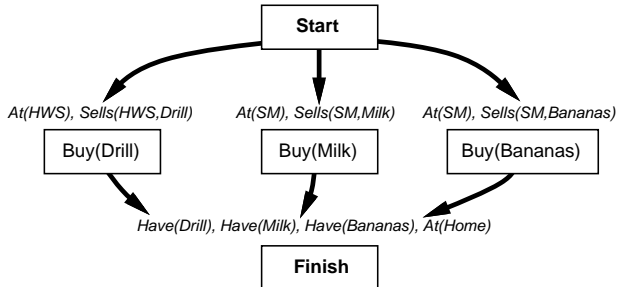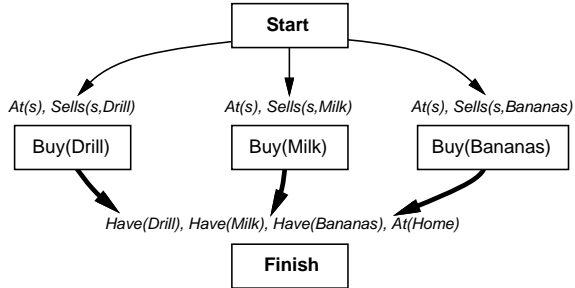
Shopping Example

```
                              ┌──────────┐
                              │  Start   │
                              └──────────┘
  At(Home)  Sells(SM,Banana)   Sells(SM,Milk)  Sells(HWS,Drill)


       Have(Drill)  Have(Milk) │ Have(Banana)  At(Home)
                              ┌──────────┐
                              │  Finish  │
                              └──────────┘
```

Start state: Op(**Action:** Start,
                  Effect: At(Home)∧Sells(HWS,Drill)∧
                      Sells(SM,Milk)∧Sells(SM,Bananas))

Goal state: Op(**Action:** Finish,
                  Precond: Have(Drill)∧Have(Milk)∧
                      Have(Bananas)∧At(Home))

Actions: Op(**Action:** Go(there),   Op(**Action:** Buy(x),
              Precond: At(here),   **Precond:** At(store)∧
              Effect: At(there)∧   Sells(store,x)
                  ¬At(here))   **Effect:** Have(x))

Example (2)

Start

At(s), Sells(s,Drill)   *At(s), Sells(s,Milk)*   *At(s), Sells(s,Bananas)*

Buy(Drill)   Buy(Milk)   Buy(Bananas)

Have(Drill), Have(Milk), Have(Bananas), At(Home)

Finish

Start

At(HWS), Sells(HWS,Drill)   *At(SM), Sells(SM,Milk)*   *At(SM), Sells(SM,Bananas)*

Buy(Drill)   Buy(Milk)   Buy(Bananas)

Have(Drill), Have(Milk), Have(Bananas), At(Home)

Finish

© G. Lakemeyer

Example (3)

Example (4)

Start

At(Home) — Go(HWS)

At(Home) — Go(SM)

At(HWS), Sells(HWS,Drill) — Buy(Drill)

At(SM), Sells(SM,Milk) — Buy(Milk)

At(SM), Sells(SM,Bananas) — Buy(Bananas)

Have(Drill) , Have(Milk) , Have(Bananas) , At(Home)

Finish

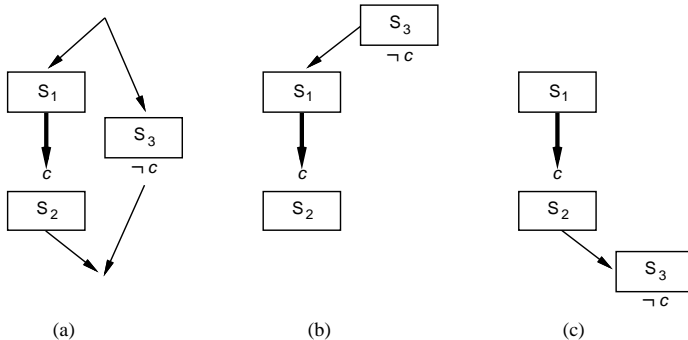### Dead end!

Go(HWS) and Go(SM) block each other because one destroys the precondition of the other.
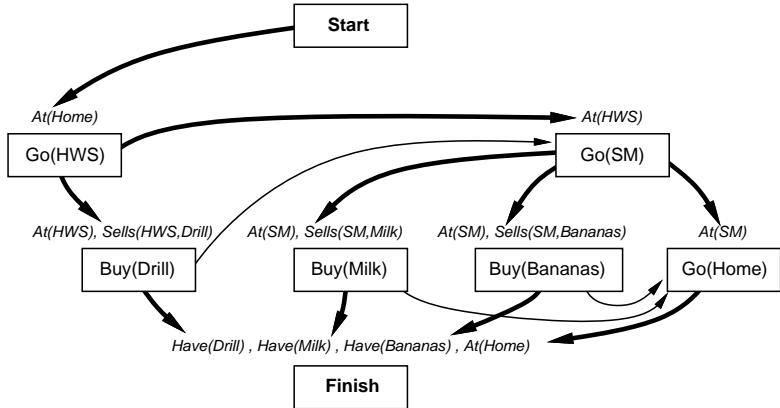
© G. Lakemeyer

Protection of Causal Relations

(a) (b) (c)

a) Conflict

   Conflict resolutions:
   - b) Demotion
   - c) Promotion

Example (5)

End of Example

© G. Lakemeyer
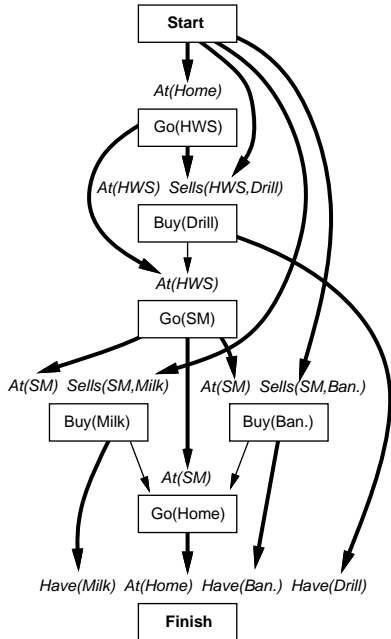
The POP Algorithm

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*

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*)

procedure RESOLVE-THREATS(*plan*)

    for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**
        choose either
            Promotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)
            Demotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)
        if not CONSISTENT(*plan*) **then fail**
    end