University of Malta
BSc(Hons.) IT Year II

Problem Solving Techniques in Artificial Intelligence

Lecture 7

Planning

- Planning v. Problem Solving
- Linear Planning
- Representation of Actions
- STRIPS Planning Algorithm
- Sussman Anomaly

Department of Computer Science and AI

Planning vs. Problem Solving

Problem solving

- Representation of Actions: successor functions
- Representation of States: complete descriptions
- Representation of Goals: goal test
- Representation of Plans: sequence of actions

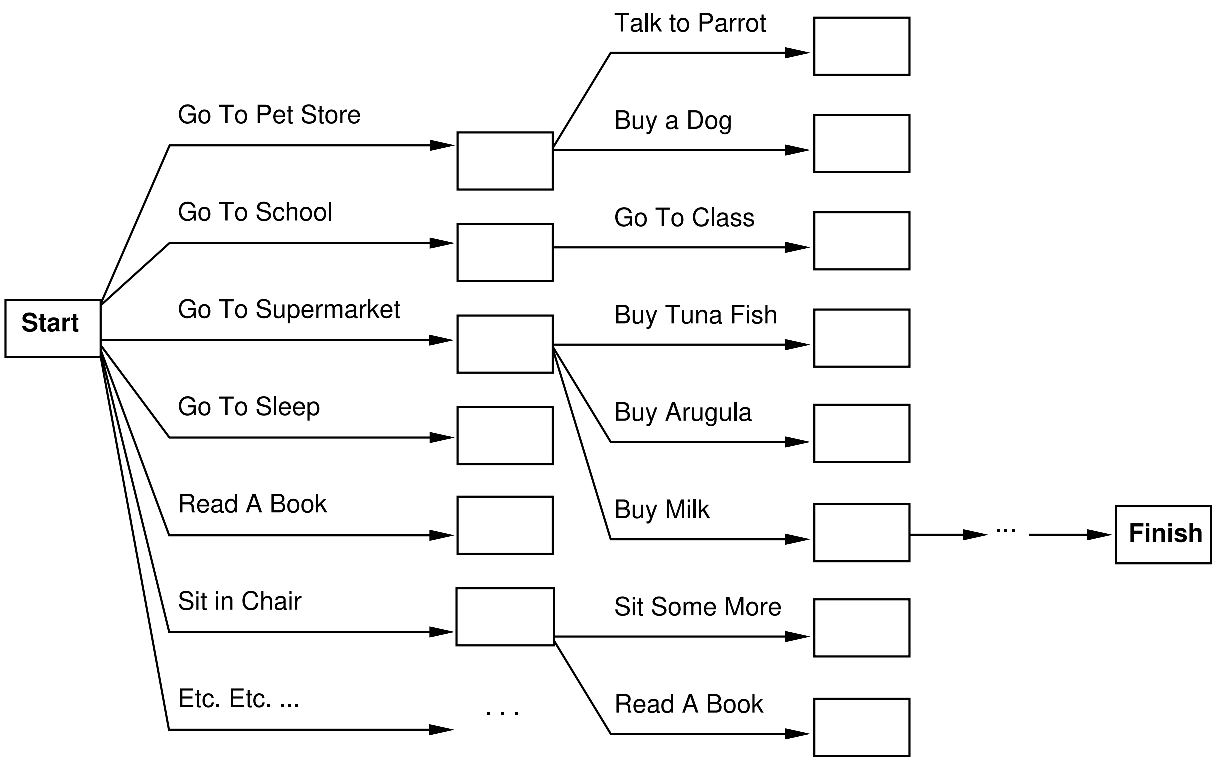
Search space becomes intractable when number of actions gets large.

Considering sequences of action starting from initial state is inefficient. E.g. agent might need to decide how to obtain the items before deciding where to go.

Agent needs flexibility be able to work on parts of the problem which are most solvable at the moment.

Planning

Problem: Get a litre of milk and a bunch of bananas and a 6v .1 amp bulb.



Planning vs. Problem Solving

Planning

- Open up representation of states and goals. Typically states and goals consist of sets of sentences in a first order language. Actions are represented by logical descriptions of preconditions and effects. Planner can reason about connections between states and actions.

- Planner is free to add actions to the plan wherever they are needed rather than in some strict sequence starting at initial state. Planner may decide to buy milk before knowing where to buy it, how to get there or what to do afterwards. Order of planning $\neq$ order of execution.

- Assumption that most parts of the world are independent of most other parts. Divide and conquer approach can be used to achieve complex goals.

Plans and Actions

All planning systems assemble plans for achieving goals out of building blocks. Typical systems assume such building blocks are actions that transform one world state into another. Actions have the following structure

- **Name:** what the action is called, e.g. “paint object a new colour.
- **Preconditions:** what has to hold for the action to be applicable.

  Example. To paint an object it should not be oily, and have an old colour.
- **Add-list:** a list of assertions that hold after the action has taken place. The colour of the object is the new colour.
- **Delete-list:** The colour of the object is the original colour.

Prolog Representation of an Action

```
strips_rule(
 paint(Obj, NewColour),
 [neg(oily(Obj)),colour_of(Obj,OldColour)],
 [colour_of(Obj,NewColour)],
 [colour_of(Obj,OldColour)]).
```

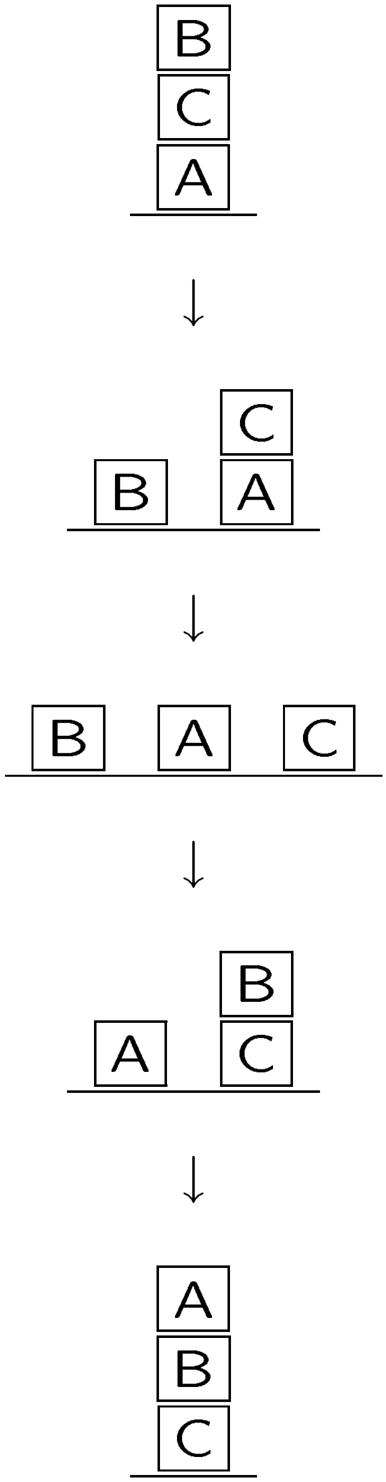
```
preclist(Act,PL) :- strips_rule(Act,PL,_,_).
addlist(Act,AL)  :- strips_rule(Act,_,AL,_).
dellist(Act,AL)  :- strips_rule(Act,_,_,DL).

prec(Act,Cond)   :- preclist(Act,PL),
                    member(Cond,PL).
adds(Act,Cond)   :- addlist(Act,AL),
                    member(Cond,AL).
dels(Act,Cond)   :- dellist(Act,DL),
                    member(Cond,DL).
```

The Blocks World

```
/* stack(X,Y) */
strips_rule(stack(X,Y),
            [holding(X),clear(Y)],
            [holding(X),clear(Y)],
            [handempty,on(X,Y),clear(X)]).

/* unstack(X,Y) */

strips_rule(unstack(X,Y),
            [on(X,Y), clear(X), handempty],
            [handempty,on(X,Y)],
            [holding(X),clear(Y)]).

/* pickup(X) */
strips_rule(pickup(X),
            [ontable(X), clear(X), handempty],
            [ontable(X),clear(X),handempty],
            [holding(X)]).

/* putdown(X) */
strips_rule(putdown(X),
```

```
                [holding(X)],
                [holding(X)],
                [ontable(X),handempty,clear(X)]).

preclist(Act,PL) :- strips_rule(Act,PL,_,_).
dellist(Act,DL)  :- strips_rule(Act,_,DL,_).
addlist(Act,AL)  :- strips_rule(Act,_,_,AL).


prec(Act,Cond)   :- preclist(Act,PL),
                    member(Cond,PL).
adds(Act,Cond)   :- addlist(Act,AL),
                    member(Cond,AL).
dels(Act,Cond)   :- dellist(Act,DL),
                    member(Cond,DL).
```

A Planning Problem



Linear versus Non Linear Planning

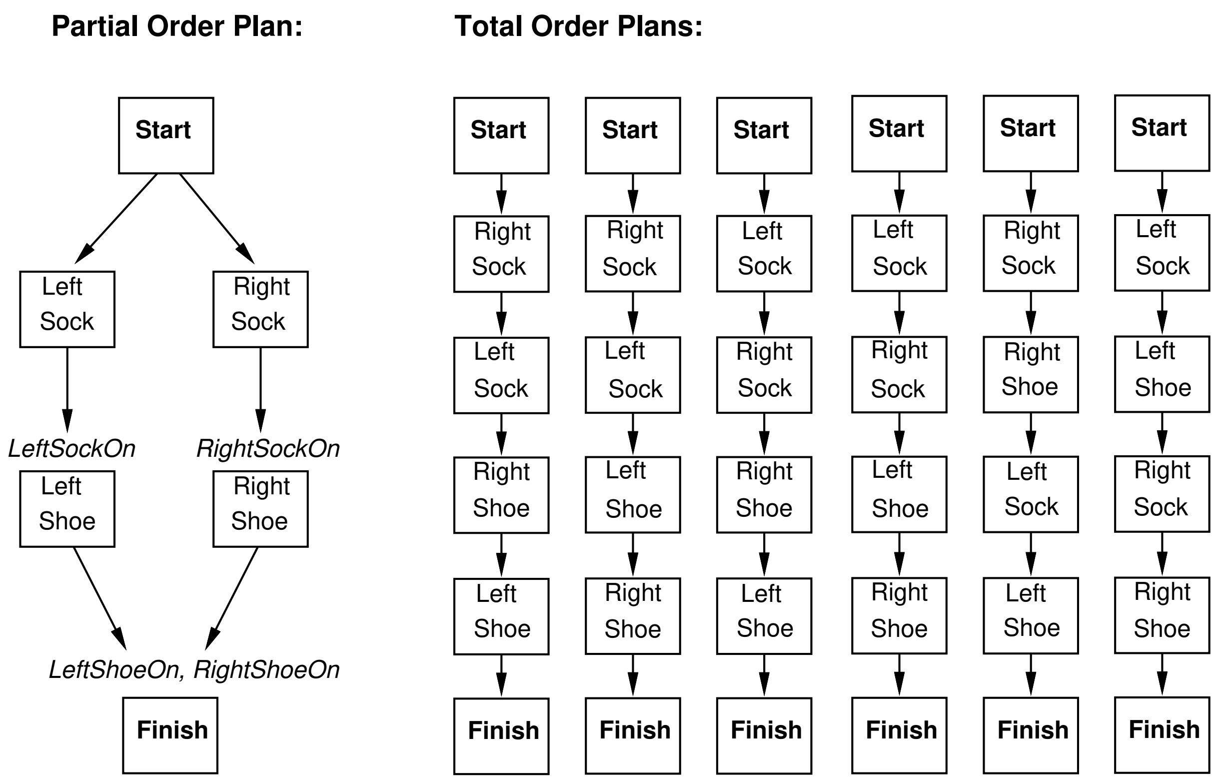
All planners develop plans in stages, starting with the empty plan and finishing with a final plan which is usually a linear sequence of actions.

Between the initial and final plans, planners maintain *partial plans* which are modified and extended during the planning process.

Linear planners are those whose partial plans in those intermediate stages are *linear sequences of actions*, as opposed to *non linear planners* which employ more general graph structures.

STRIPS (Stanford Research Institute Problem Solver – Fikes & Nilsson) is a well-known example of a linear planner.

Linear versus Non-Linear Plans



STRIPS -Informal Description

- Maintains a "current state" (initially the initial state) a goal stack (initially the given goal), and a partial plan (initially empty).
- Terminates when the goal stack is empty (i.e. satisfied by the current state).
- A goal unsatisfied in the current state is chosen from the goal stack.
- An action is found whose addlist contains the goal.
- STRIPS then forms a plan to achieve that action's preconditions by calling itself recursively.
- When that plan is found it applies the action to the current state.
- The plan as well as the action is appended to the current partial plan.
- Finally, it calls itself recursively with the original goal list, the updated current state and partial plan.

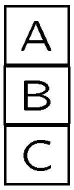
STRIPS Algorithm

```
PROCEDURE STRIPS( GoalList,
                   InitialState,
                     PartialPlan)
Plan         := PartialPlan
CurrentState := InitialState
WHILE exists goal G in GoalList
      unsatisfied in CurrentState DO
  { find an action A with G in its addlist;
    else backtrack
    PreConditions := preconditions of A
    APlan         := STRIPS(PreConditions,
                            CurrentState,
                             PartialPlan)
    CurrentState  := result of executing Aplan
                     in CurrentState
                     + A's addlist
                     - A's deletelist
    Plan          := Plan + APlan + A
  }
return Plan
```

Remarks on the Algorithm

Algorithm contains two non-deterministic choice points:

- Selection order of goals in a compound goal. For instance, to build the following stack

  two goals must be achieved: `on(a,b)` and `on(b,c)`. Clearly, the order in which they are achieved is crucial.

- Selection of an action to achieve an atomic goal. In the simple blocks world we have been considering, an action is deemed to achieve such a goal is that goal is present in its addlist. For example, `clear(X)` appears in the addlist of `stack(X,Y)`, `unstack(Y,X)`, and `putdown(X)`.

Remarks on the Algorithm

- One may bring to bear any search technique at this point.

- If a simple depth first strategy is adopted, there is a risk of infinite loops.

- One may avoid loops by forbidding repeated actions but this renders the planner incomplete.

STRIPS in Prolog

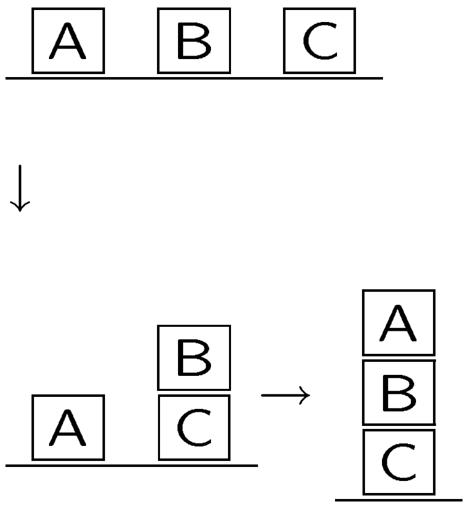
```
strips(Goal, InitState, Plan):-
strips1(Goal, InitState, [], [], _, RevPlan),
        reverse( RevPlan, Plan).

% strips(+GoalList,
          +State,
           +Plan,
            +ForbiddenActions,
             -NewState,
              -NewPlan)


% If goal list is a subset of current state, succeed
% with the partial plan you have.

strips1(GoalList,
         State,
          Plan,
           _,
           State,
            Plan) :-
subset(GoalList,State).
```

STRIPS in Prolog

```
strips1(GoalList, State,
        Plan, ForbiddenActions,
        NewState, NewPlan)        :-

  member(Goal, GoalList),
  not(member(Goal, State)),
  adds(Ac, Goal),
  % No repeat action during the planning
  not(member(Ac, ForbiddenActions)),
  preclist(Ac, PrecList),
  strips1(PrecList,
            State,
              Plan,
               [Ac| ForbiddenActions],
                 TmpState1, TmpPlan1),
  apply_rule(Ac, TmpState1, TmpState2),
  strips1(GoalList,
            TmpState2,
              [Ac|TmpPlan1],
                ForbiddenActions,
                  NewState,
                    NewPlan).
```

STRIPS in Action

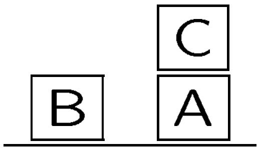
```
?- strips([clear(a),
           clear(b),
           clear(c),
           ontable(a),
           ontable(b),
           ontable(c),
           handempty],
           [on(b,c),on(a,b)],
           Plan).

Plan = [pickup(b), stack(b,c),
        pickup(a), stack(a,b)]
```



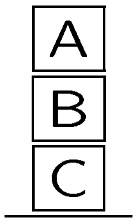
Sussman Anomaly

```
?- strips([clear(c),
           clear(b),
           ontable(b),
           ontable(a),
           on(c,a),
           handempty],
           [on(a,b),on(b,c)],
           Plan).
```

Initial State



Goal State



Exercise: find out whether strips planner available on web site can solve this problem and if not why not.