AI Planning

STRIPS

Simplified (no variables in actions)

What is AI Planning?

- Given goals and logic
  - Find sequence of actions to achieve goals
  - Automated!
- One of the oldest subjects in AI
- Why does planning involve?
  - It involves knowledge representation
  - And causal reasoning
- Useful for robotics, space exploration, etc.

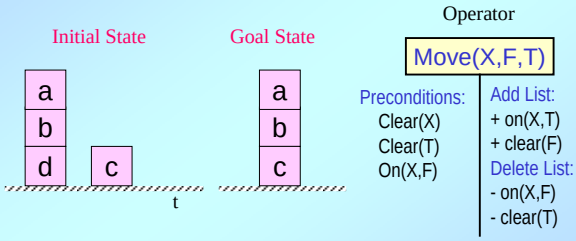
Knowledge Representation Problem

- How to represent this world?
- What are relevant and what are not?
- What actions will cause what results?
- What will my action change and not change?
- The *Closed World Assumption*
  - Anything that are not known to be true are false

Why is Planning Difficult?

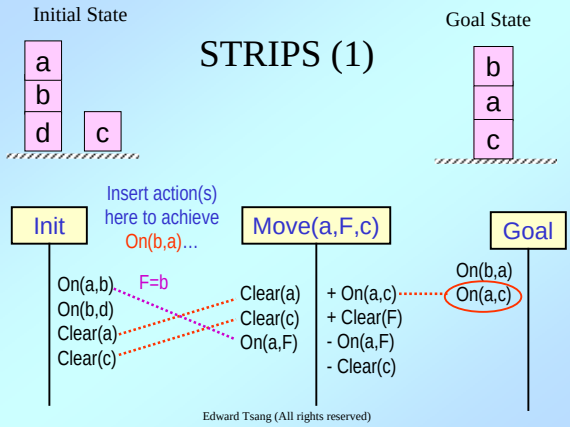
- The *Frame Problem*
  - What are the consequences of my actions?
  - Relatively easy on "opening the door"
  - Difficult for "dropping my glass"
- The *Ramification Problem*
  - I can only open the door if it is not locked
  - … and the knob is there
  - … and my hand is not injured
  - …. plus many, many factors, too many to mention

Blocks World Planning Problem

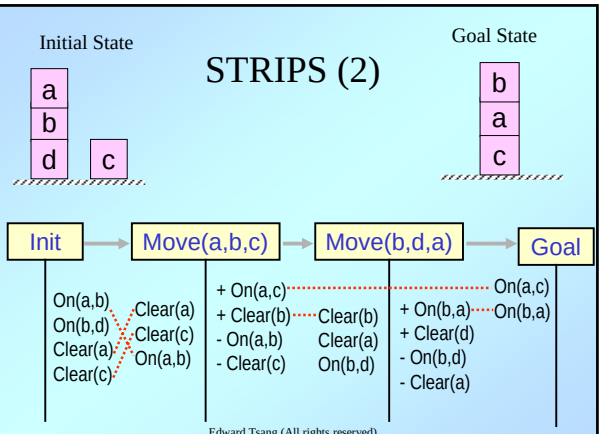


- Task: find a plan to achieve Goal from Initial State

STRIPS (1)



STRIPS (2)

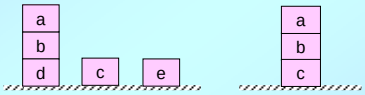


Blocks World Problem in Prolog

```
block(a).
block(b).
block(c).
block(d).
init( [on(block(a), block(b)), clear(block(a)),
   on(block(b), block(d)),
   clear(block(c)) ] ).
goal( [ on(block(b), block(a)), on(block(a), block(c)) ] ).
```

Blocks World Problem in Prolog

```
block(a).
block(b).
block(c).
block(d).
block(e).
init( [on(block(a), block(b)), clear(block(a)),
   on(block(b), block(d)),
   clear(block(c)), clear(block(e)) ] ).
goal( [ on(block(a), block(b)), on(block(b), block(c)) ] ).
```



Blocks World Operator Definition

```
/*
   operator/4 defines the operator name, precondition,
   addlist and deletelist.
*/
   /* moving block X from block Z to top of block Y */
operator(
   move(block(X), block(Z), block(Y)),
   precondition( [on(block(X), block(Z)), clear(block(X)),
   clear(block(Y))] ),
   addlist( [on(block(X), block(Y)), clear(block(Z))] ),
   deletelist( [on(block(X), block(Z)), clear(block(Y))] ) ) :-
        block(X), block(Y), block(Z).
```


Simplified STRIPS Planner

```
/*
   This program will try to find the shortest sequence of
   actions that will achieve the goals using "iterative
   deepening":
*/
plan :-
        init( InitState ),      /* domain specific knowledge */
        goal( Goals ),          /* domain specific knowledge */
        depth( MaxDepth ),
        strips( InitState, Goals, Solution, MaxDepth ),
        report_solution(Solution, InitState, Goals).
```


Definition of depth/1

```
/*
   depth(N) returns N from 0, 1, 2, ...
*/
depth( 0 ).
depth( N ) :-
    depth( Nless1 ),
    N is Nless1 + 1.
```

Specification of STRIPS

```
/*
   strips( InitialState, GoalState, Solution, MaxDepth )
   instantiates Solution to a plan (of no more than MaxDepth
   steps) that brings the world from the InitialState to the
   GoalState.
   A plan is a sequence of operators, to be executed in the
   order specified.
   For simplicity, variables binding has not been taken care
   of rigorously; e.g. if on(b, X) is in the delete list of an
   operator and on(b, c) is in the goal list, then one should
   make sure that X =/= c.
*/
```


Simplified STRIPS Planner

```
strips( InitState, Goals, [ ], _ ) :-
   delete_if_present( InitState, Goals, [ ] ).
strips( InitState, Goals, Solution, MaxDepth ) :-
   MaxDepth > 0, !,
   member( Goal1, Goals ),
   operator( Op, precondition(PC), addlist(AL), deletelist(DL) ),
   member( Goal1, AL ),
   not((member(X, DL), member(Y, Goals), X==Y)),
   delete_if_present( AL, Goals, UnsatisfiedGoals ),
   set_union( UnsatisfiedGoals, PC, NewGoals ),
   strips( InitState, NewGoals, Plan, MaxDepth-1 ),
   append( Plan, [Op], Solution ).

/* delete_if_present( L1, L2, L3 ) instantiate L3 to the list of
elements in L2 that are absent in the list L1. */
```


Sample Output

```
?- plan.
Attempting to use 0 steps...
Attempting to use 1 steps...
Attempting to use 2 steps...
** Initial State: [on(block(a), block(b)), clear(block(a)),
   on(block(b), block(d)), clear(block(c))]
** A plan is found:
      move(block(a), block(b), block(c))
      move(block(b), block(d), block(a))
** Goals: [on(block(b), block(a)), on(block(a), block(c))]
```