Planning

Planning finds sequence of actions that achieves a given goal when performed starting in a given state.

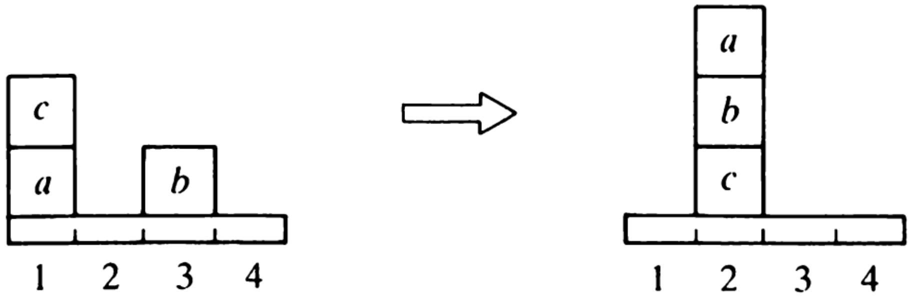
- We studied how to take actions in the world (search)
- We studied how to represent objects, relations, etc. (logic)
- Planning combine the two!

Planning vs. Problem solving

1- In planning, states and goals are represented by sentences: Have(milk). Actions are represented by rules: preconditions and effects. Buy(x)→Have(x).
2- In planning, goals are independent, thus can be solved by "divide-and-conquer" strategy. Have(milk) ^ Have(banana).
3- Planner is free to add action whenever they are needed, rather than in an incremental sequence of search.

Blocks World Problem

In the blocks world, the planner finds a sequence of actions that achieve the goal: a on b, and b on c.



For the above example, we have two relationships:

- On(Block, Object).
- sClear(Object).

STRIPS Language

1- States are list of conjunctive relationships that are currently true.
   - Initial state: [clear (2) ^ clear(4) ^ clear(b) ^ clear(c) ^ on(c,a) ^ on(a,1) ^ on(b,3)].
   - Goals are defined as: [on (a,b) ^ on (b,c)].
2- Any actions that are not mentioned in the states are assumed to be false. Ex: from the initial state, we get ¬clear(3) ^ ¬clear(1) ^ ....
3- Each action is defined by two terms:
   - *Precondition*: the conditions that has to be satisfied for the action to be possible.
   - *Effect*: the effect of the action either adds relationships or deletes some of them.

For example, the action **move(b,3,c)** (move block b from location 3 to block c).

- Precondition: [clear(b) ^ clear(c) ^ on(b,3)].
- Effect: add the relationships on(b,c) and clear(3), and delete on(b,3) and clear(c).

Thus the new state is:

[on(b, c), clear(3), clear(2), clear(4), clear(b), on(a,1), on(c, a)]

The effects of an action can be:

1- Positive: add some relationships.
2- Negative: delete some relationships

Preconditions of action **Action** when condition **Cond** is true will be defined by the predicate: **Can(Action, Cond).**

The effects of action will be defined by two predicates:
adds(Actions, Addrels), where Addrels is a list of added relationships.
delete(Action, Delrels), Dlrels is a list of removed relationships.

The goal of a plane can be a list of relationships: [**on(a,b), on(b,c)**].

For the blocks world actions will be of the form:
Move(Block, From, To), where **Block** is the block to be moved, **From** is position, and **To** is the new position.

```
% Definition of action move( Block, From, To) in blocks world
% can( Action, Condition): Action possible if Condition true
can( move( Block, From, To), [ clear( Block), clear( To), on( Block, From)] ) :-
  block( Block),                % Block to be moved
  object( To),                  % 'To' is a block or a place
  To \== Block,                 % Block cannot be moved to itself
  object( From),                % 'From' is a block or a place
  From \== To,                  % Move to new position
  Block \== From.               % Block not moved from itself
% adds( Action, Relationships): Action establishes Relationships
adds( move(X,From,To), [ on(X,To), clear(From)]).
% deletes( Action, Relationships): Action destroys Relationships
deletes( move(X,From,To), [ on(X,From), clear(To)]).
object( X) :-              % X is an object if
  place( X)                % X is a place
  ;                        % or
  block( X).               % X is a block
% A blocks world
block( a).
block( b).
block( c).
place( 1).
place( 2).
place( 3).
place( 4).
% A state in the blocks world
%
%         c
%         a   b
%         = = = =
% place   1 2 3 4
state1( [ clear(2), clear(4), clear(b), clear(c), on(a,1), on(b,3), on(c,a) ] ).
```

The Planner work

Suppose the goal **on(a,b)**.

The planner would reason as follows:

1- find the action **move(a, From, b)**.
2- loock at the predicate **can** to find the action's preconditions:[clear(a), clear(b), on(a, From)]., **clear(a)** is not true, so the planner consider **clear(a)** as new goal to be achieved.
3- Look at the adds relation again to find action that achieves clear(a). This can any action of the form: **move(Block, a, To)**.
4- The precondition for this action is [clear(Block), clear(To), on(Block,a)]

This is satisfied in our initial situation if: Block=c and To=2.

5- the action move(c,a,2) will generate the state

   [clear(a), clear(b), clear(c), clear(4), on(a,1), on(b,3), on(c,2)]
6- now the action move(a,1,b) can be executed to find the final goal on(a, b).
7- the plan is [move(c,a,2), move(a,1,b)].

To solve a list of goals **Goals** in the state **State,** leading to the state **Finalstate**, do:

If all Goals are true in the state **State** then **Finalstate** =**State**. Otherwise do:

1- select unsolved goal in **Goals**.
2- Find an action **Action** that adds **Goal** to the current state.
3- Enable **Action** by solving the precondition **Condition** of **Action**, giving **Midstate1**.
4- Apply **Action** to **Midstate1**, giving **Midstate2**.
5- Solve **Goals** in **Midstate2**, leading to **Finalstate**.

This programmed in prolog as the procedure:

Plan(State, Goals, Plan, Finalstate)

Where state: the initial state, Finalstate: the final state, Goals: the list of goals, Plan: list of actions that achieves the goals.

If we asked the above program the query:

?- state1(Start), plan(Start, [on(a,b), on(b,c)], Plan,_).

The program may answer:

Plan= [move(c,a,2), move(b,3,a), move(b, a, c), move(a,1,b)] !!!! (use four moves and the second one does not make sense).

The reason for this bad planning is that goals are achieved one by one in a linear order (*linear planning*). So, key to ensure optimal plans is to enable interaction between different goals. This is done through the mechanism of *goal regression*.

```
% plan( State, Goals, Plan, FinalState)

plan( State, Goals, [ ], State) :-                   % Plan empty
   satisfied( State, Goals).                         % Goals true in State

plan( State, Goals, Plan, FinalState) :-
   conc( Plan, _, _),                                % Try plans of increasing length
   conc( PrePlan, [Action | PostPlan], Plan),        % Divide Plan to PrePlan, Action and PostPlan
   select( State, Goals, Goal),                      % Select a goal
   achieves( Action, Goal),                          % Relevant action
   can( Action, Condition),
   plan( State, Condition, PrePlan, MidState1),      % Enable Action
   apply( MidState1, Action, MidState2),             % Apply Action
   plan( MidState2, Goals, PostPlan, FinalState).    % Achieve remaining goals

% satisfied( State, Goals): Goals are true in State

satisfied( State, [ ]).

satisfied( State, [Goal | Goals]) :-
   member( Goal, State),
   satisfied( State, Goals).

select( State, Goals, Goal) :-
   member( Goal, Goals),
   \+ member( Goal, State).                          % Goal not satisfied already

% achieves( Action, Goal): Goal is in add-list of Action

achieves( Action, Goal) :-
   adds( Action, Goals),
   member( Goal, Goals).

% apply( State, Action, NewState): Action executed in State produces NewState

apply( State, Action, NewState) :-
   deletes( Action, DelList),
   delete_all( State, DelList, State1), !,
   adds( Action, AddList),
   conc( AddList, State1, NewState).

% delete_all( L1, L2, Diff) if Diff is set-difference of L1 and L2

delete_all( [ ], _, [ ]).

delete_all( [X | L1], L2, Diff) :-
   member( X, L2), !,
   delete_all( L1, L2, Diff).

delete_all( [X | L1], L2, [X | Diff]) :-
   delete_all( L1, L2, Diff).
```