Planning

Components of a Planning System

- In any general problem solving systems, elementary techniques to perform following functions are required
  - Choose the best rule (based on heuristics) to be applied
  - Apply the chosen rule to get new problem state
  - Detect when a solution has been found
  - Detect dead ends so that new directions are explored.

Advanced Problem Solving Approaches

- In order to solve nontrivial problems, it is necessary to combine
  - Basic problem solving strategies
  - Knowledge representation mechanisms
  - Partial solutions and at the end combine into complete problem solution (decomposition)
- Planning refers to the process of computing several steps of a problem solving before executing any of them.
- Planning is useful as a problem solving technique for non decomposable problem.

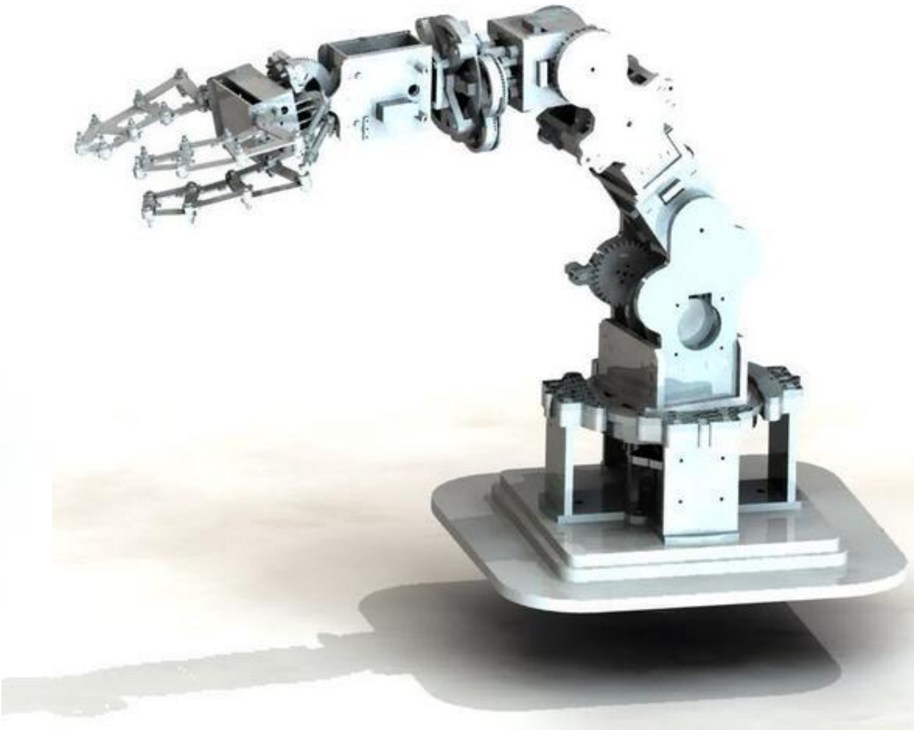
Choose Rules to apply

- Most widely used technique for selecting appropriate rules is to
  - first isolate a set of differences between the desired goal state and current state,
  - identify those rules that are relevant to reducing these difference,
  - if more rules are found then apply heuristic information to choose out of them.

Apply Rules

- In simple problem solving system, applying rules was easy as each rule specifies the problem state that would result from its application.
- In complex problem we deal with rules that specify only a small part of the complete problem state.

Block World Problem



Example: Block World Problem

- Block world problem assumptions
  - Square blocks of same size
  - Blocks can be stacked one upon another.
  - Flat surface (table) on which blocks can be placed.
  - Robot arm that can manipulate the blocks. It can hold only one block at a time.
- In block world problem, the state is described by a set of predicates representing the facts that were true in that state.
- One must describe for every action, each of the changes it makes to the state description.
- In addition, some statements that everything else remains unchanged is also necessary.

Actions (Operations) done by Robo

- UNSTACK (X, Y) : **[US (X, Y)]**
  – Pick up X from its current position on block Y. The arm must be empty and X has no block on top of it.
- STACK (X, Y): **[S (X, Y)]**
  – Place block X on block Y. Arm must holding X and the top of Y is clear.
- PICKUP (X): **[PU (X)]**
  – Pick up X from the table and hold it. Initially the arm must be empty and top of X is clear.
- PUTDOWN (X): **[PD (X)]**
  – Put block X down on the table. The arm must have been holding block X.

- Predicates used to describe the state
  - ON(X, Y) - Block X on block Y.
  - ONT(X) - Block X on the table.
  - CL(X) - Top of X clear.
  - HOLD(X) - Robot-Arm holding X.
  - AE - Robot-arm empty.
- Logical statements true in this block world.
  - Holding X means, arm is not empty

    $(\exists X)\ HOLD\ (X) \rightarrow \sim AE$
  - X is on a table means that X is not on the top of any block

    $(\forall X)\ ONT\ (X) \rightarrow \sim\ (\exists Y)\ ON\ (X, Y)$
  - Any block with no block on has clear top

    $(\forall X)\ (\sim\ (\exists Y)\ ON\ (Y,X)) \rightarrow\ CL\ (X)$

Effect of Unstack operation

- The effect of US(X, Y) is described by the following axiom

  $[CL(X, State) \Lambda ON(X, Y, State)] \rightarrow$

  $[HOLD(X, DO(US (X, Y), State)) \Lambda$

  $CL(Y, DO(US(X, Y), State))]$

  - DO is a function that generates a new state as a result of given action and a state.
- For each operator, set of rules (called frame axioms) are defined where the components of the state are
  - affected by an operator
    - If US(A, B) is executed in state S0, then we can infer that HOLD (A, S1) Λ CLEAR (B, S1) holds true, where S1 is new state after Unstack operation is executed.
  - not affected by an operator
    - If US(A, B) is executed in state S0, B in S1 is still on the table but we can't derive it. So frame rule stating this fact is defined as $ONT(Z, S) \rightarrow ONT(Z, DO(US (A, B), S))$

- Advantage of this approach is that
  - simple mechanism of resolution can perform all the operations that are required on the state descriptions.
- Disadvantage is that
  - number of axioms becomes very large for complex problem such as COLOR of block also does not change.
  - So we have to specify rule for each attribute.

$$\text{COLOR}(X, \text{red}, S) \rightarrow \text{COLOR}(X, \text{red}, \text{DO}(\text{US}(Y, Z), s))$$

- To handle complex problem domain, there is a need of mechanism that does not require large number of explicit frame axioms.

STRIPS Mechanism

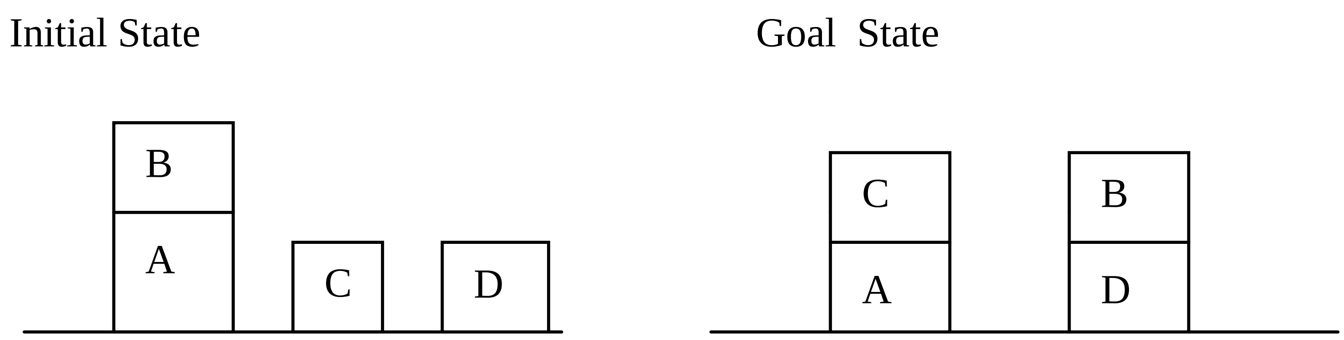
- One such mechanism was used in early robot problem solving system named STRIPS (developed by Fikes, 1971).
- In this approach, each operation is described by three lists.
  - **Pre_Cond list** contains predicates which have to be true before operation.
  - **ADD list** contains those predicates which will be true after operation
  - **DELETE list** contain those predicates which are no longer true after operation
- Predicates not included on either of these lists are assumed to be unaffected by the operation.
- Frame axioms are specified implicitly in STRIPS which greatly reduces amount of information stored.

STRIPS – Style Operators

- S (X, Y)
  - Pre: CL (Y) Λ HOLD (X)
  - Del: CL (Y) Λ HOLD (X)
  - Add: AE Λ ON (X, Y)
- US (X, Y)
  - Pre: ON (X, Y) Λ CL (X) Λ AE
  - Del: ON (X, Y) Λ AE
  - Add: HOLD (X) Λ CL (Y)
- PU (X)
  - Pre: ONT (X) Λ CL (X) Λ AE
  - Del: ONT (X) Λ AE
  - Add: HOLD (X)
- PD (X)
  - Pre: HOLD (X)
  - Del: HOLD (X)
  - Add: ONT (X) Λ AE

Goal stack method - Example

- Logical representation of Initial and Goal states:
  - Initial State: ON(B, A) Λ ONT(C) Λ ONT(A) Λ ONT(D) Λ CL(B) ΛCL(C) Λ CL(D) Λ AE
  - Goal State: ON(C, A) Λ ON(B, D) Λ ONT(A) Λ ONT(D) Λ CL(C) Λ CL(B) Λ AE



- We notice that following sub-goals in goal state are also true in initial state.

  ONT(A) Λ ONT(D) Λ CL(C) Λ CL(B) Λ AE

- Represent for the sake of simplicity - **TSUBG**.
- Only sub-goals ON(C, A) & ON(B, D) are to be satisfied and finally make sure that TSUBG remains true.
- Either start solving first ON(C, A) or ON(B, D). Let us solve first ON(C, A).

Goal Stack:

ON(C, A)

ON(B, D)

ON(C, A) Λ ON(B, D) Λ TSUBG

- To solve ON(C, A), operation S(C, A) could only be applied.
- So replace ON(C, A) with S(C, A) in goal stack.

Goal Stack:

S (C, A)

ON(B, D)

ON(C, A) Λ ON(B, D) Λ TSUBG

- S(C, A) can be applied if its preconditions are true. So add its preconditions on the stack.

Goal Stack:

CL(A)	
HOLD(C)	Preconditions of STACK
CL(A) Λ HOLD(C)	
S (C, A)	Operator
ON(B, D)	
ON(C, A) Λ ON(B, D) Λ TSUBG	

- Next check if CL(A) is true in State_0.
- Since it is not true in State_0, only operator that could make it true is US(B, A).
- So replace CL(A) with US(B, A) and add its preconditions.

Goal Stack: ON(B, A)

CL(B)	Preconditions of UNSTACK
AE	
ON(B, A) Λ CL(B) Λ AE	
US(B, A) Operator	
HOLD(C)	
CL(A)) Λ HOLD(C)	
S (C, A)	Operator
ON(B, D)	
ON(C, A) Λ ON(B, D) Λ TSUBG	

- ON(B, A), CL(B) and AE are all true in initial state, so pop these al[illegible]h its compound goal.
- Next pop top operator US(B, A) and produce new state by using its ADD and DELETE lists.
- Add US(B, A) in a queue of sequence ofoperators.

SQUEUE = US (B, A)

State_1:

ONT(A) Λ ONT(C) Λ ONT(D) Λ HOLD(B) Λ CL(A) Λ CL(C) Λ CL(D)

Goal Stack:

HOLD(C)

CL(A)) Λ HOLD(C)

S (C, A) Operator

ON(B, D)

ON(C, A) Λ ON(B, D) Λ TSUBG

- To satisfy the goal HOLD(C), two operators can be used e.g., PU(C) or US(C, X), where X could be any block. Let us choose PU(C) and proceed further.
- Repeat the process. Change in states is shown below.

State_1:

ONT(A) ΛONT(C) Λ ONT(D) Λ HOLD(B) ΛCL(A) Λ CL(C) ΛCL(D)

SQUEUE = US (B, A)

- Next operator to be popped of is S(B, D). So

State_2:

ONT(A) ΛONT(C) Λ ONT(D) Λ ON(B, D) ΛCL(A) Λ CL(C) ΛCL(B)ΛAE

SQUEUE = US (B, A), S(B, D)

State_3:

ONT(A) ΛHOLD(C) Λ ONT(D) Λ ON(B, D) ΛCL(A) ΛCL(B)

SQUEUE = US (B, A), S(B, D), PU(C)

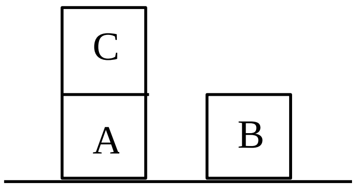
State_4:

ONT(A) ΛON(C, A) Λ ONT(D) Λ ON(B, D) ΛCL(C) ΛCL(B) ΛAE

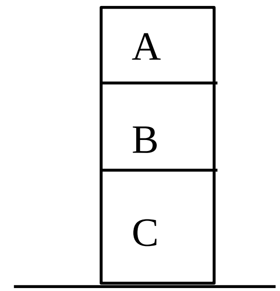
SQUEUE = US (B, A), S(B, D), PU(C), S(C, A)

Example 2

Initial State (State0)

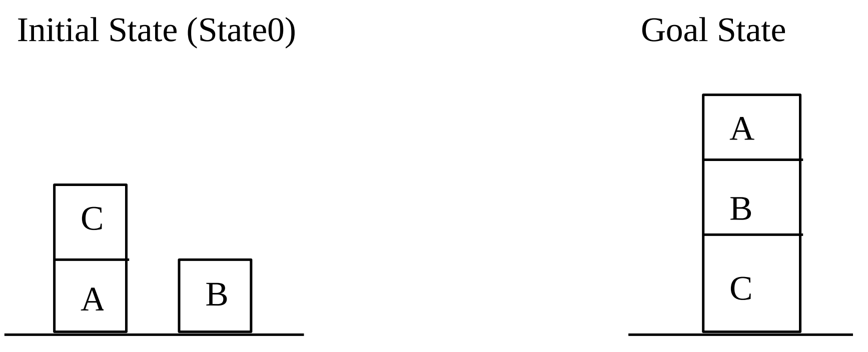


Goal State



Example 2

- The Goal stack method is not efficient for difficult problems such as Sussman anomaly problem.
- It fails to find good solution.
- Let us consider the Sussman anomaly problem

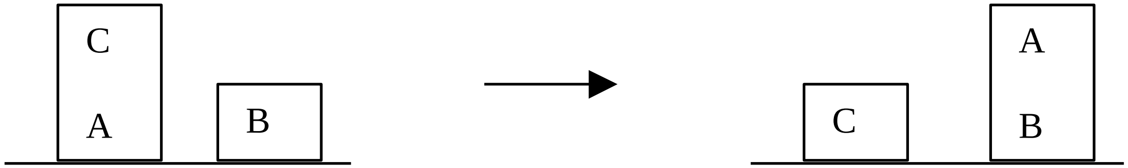


Initial State: ON(C, A) Λ ONT(A) Λ ONT(B)

Goal State: ON(A, B) Λ ON(B, C)

- Remove CL and AE predicates for the sake of simplicity.
- To satisfy ON(A, B), following operators are applied

  US(C, A) , PD(C), PU(A) and S(A, B))

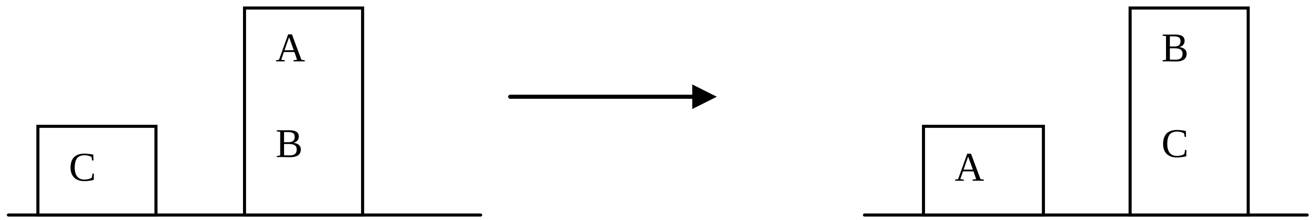


State_1: ON(B, A) Λ ONT(C)

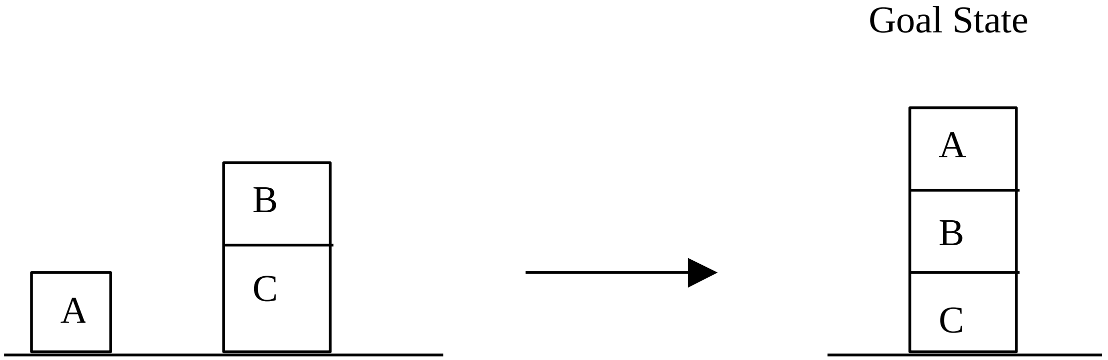
- To satisfy ON(B, C), following operators are applied

US(A, B) , PD(A), PU(B) and S(B, C)

State_2: ON(B, C) Λ ONT(A)



- Finally satisfy combined goal ON(A, B) Λ ON(B, C).
- Combined goal fails as while satisfying ON(B, C), we have undone ON(A, B).
- Difference in goal and current state is ON(A, B).
- Operations required are PU(A) and S(A, B)



Solution

- The complete plan for solution is as follows:

  1. US(C, A)
  2. PD (C)
  3. **PU(A)**
  4. **S(A, B)**
  5. **US(A, B)**
  6. **PD(A)**
  7. PU(B)
  8. S(B, C)
  9. PU(A)
  10. S(A, B)

- Although this plan will achieve the desired goal, but it is not efficient.

- In order to get efficient plan, either repair this plan or use some other method.
- Repairing is done by looking at places where operations are done and undone immediately, such as S(A, B) and US(A, B).
- By removing them, we get

1. US(C, A)
2. PD (C)
3. PU(B)
4. S(B, C)
5. PU(A)
6. S(A, B)

Planning vs. Problem Solving

- Planning and problem solving (Search) are considered **as different approaches** even though they can often be applied to the same problem.
- Basic problem solving searches a state-space of possible actions, starting from an initial state and following **any path that it believes will lead it the goal state.**
- Planning is distinct from this in three key ways:
  1. **Planning "opens up" the representation of states, goals and actions** so that the planner can deduce direct connections between states and actions.
  2. The **planner does not have to solve the problem** in order (from initial to goal state) it can suggest actions to solve any sub-goals at anytime.
  3. Planners assume that most parts of the world **are independent** so they can be stripped apart and solved individually.

Finding a solution

1. Look at the state of the world:
   - Is it the goal state? If so, the list of operators so far is the plan to be applied.
   - If not, go to Step 2.
2. Pick an operator:
   - Check that it has not already been applied (to stop looping).
   - Check that the preconditions are satisfied.

   If either of these checks fails, backtrack to get another operator.
3. Apply the operator:
   1. Make changes to the world: delete from and add to the world state.
   2. Add operator to the list of operators already applied.
   3. Go to Step 1.

STRIPS Representation

- Planning can be considered as a logical inference problem:
  - a plan is inferred from facts and logical relationships.
- STRIPS represented planning problems as a series of state descriptions and operators expressed in first-order predicate logic.

represent the state of the world at three points during the plan:

- Initial state, the state of the world at the start of the problem;
- Current state, and
- Goal state, the state of the world we want to get to.

Operators are actions that can be applied to change the state of the world.

- Each operator has outcomes i.e. how it affects the world.
- Each operator can only be applied in certain circumstances. These are the preconditions of the operator.

Representing Operators

- STRIPS operators are defined as:
  - **NAME**: How we refer to the operator e.g. go(Agent, From, To).
  - **PRECONDITIONS**: What states need to hold for the operator to be applied. e.g. [at(Agent, From)].
  - **ADD LIST**: What new states are added to the world as a result of applying the operator e.g. [at(Agent, To)].
  - **DELETE LIST**: What old states are removed from the world as a result of applying the operator. e.g. [at(Agent, From)].