Unit 4

Planning

- State of the world is represented as collection of variables-Planning Domain Definition Language:PDDL
- search problem: the initial state, the actions that are available in a state, the result of applying an action, and the goal test.
- Each state is represented as a conjunction of fluents that are ground, functionless atoms
- Example: a state in a package delivery problem might be At(Truck 1, Melbourne) ∧ At(Truck 2, Sydney).
- A set of ground (variable-free) actions can be represented by a single action schema.
- The schema is a lifted representation—it lifts the level of reasoning from propositional logic to a restricted subset of first-order logic.

For example, here is an action schema for flying a plane from one location to another:

- Action(Fly(p, from, to),
- PRECOND: At(p, from) $\land$ Plane(p) $\land$ Airport (from) $\land$ Airport (to)

  (the states in which the action can be executed)
- EFFECT: $\neg$At(p, from) $\land$ At(p, to))

  (defines the result of executing the action)

- The schema consists of the action name, a list of all the variables used in the schema, a precondition and an effect.
- action that results from substituting values for all the variables:

  Action(Fly(P1, SFO, JFK),
- PRECOND:At(P1, SFO) ∧ Plane(P1) ∧ Airport (SFO) ∧ Airport (JFK)
- EFFECT:¬At(P1, SFO) ∧ At(P1, JFK))
- The precondition and effect of an action are each conjunctions of literals (positive or negated atomic sentences).

- An action a can be executed in state s if s entails the precondition of a.
- Entailment can also be expressed with the set semantics:
- s |= q iff every positive literal in q is in s and every negated literal in q is not.
- In formal notation we say

$$(a \in \text{ACTIONS}(s)) \Leftrightarrow s \models \text{PRECOND}(a) \text{ ,}$$

where any variables in a are universally quantified.

- For example,
- ∀ p, from, to (Fly(p, from, to) ∈ ACTIONS(s)) ⇔

  s |= (At(p, from) ∧ Plane(p) ∧ Airport (from) ∧ Airport (to))
- We say that action a is applicable in state s if the preconditions are satisfied by s. When an action schema a contains variables, it may have multiple applicable instance
- Fly action can be instantiated as Fly(P1, SFO, JFK) or as Fly(P2, JFK, SFO), both of which are applicable in the initial state.

- If an action a has v variables, then, in a domain with k unique names of objects, it takes O(v^k) time in the worst case to find the applicable ground actions.
- The result of executing action a in state s is defined as a state s which is represented by the set of fluents formed by starting with s, removing the fluents that appear as negative
- DELETE LIST literals in the action’s effects (what we call the delete list or DEL(a)), and adding the fluents that are positive literals in the action’s effects (what we call the add list or ADD(a)):
- RESULT(s, a) = (s − DEL(a))∪ ADD(a)

- For example, with the action Fly(P1, SFO, JFK), we would remove At(P1, SFO) and add At(P1, JFK).
- It is a requirement of action schemas that any variable in the effect must also appear in the precondition.
- That way, when the precondition is matched against the state s, all the variables will be bound, and RESULT(s, a) will therefore have only ground atoms.

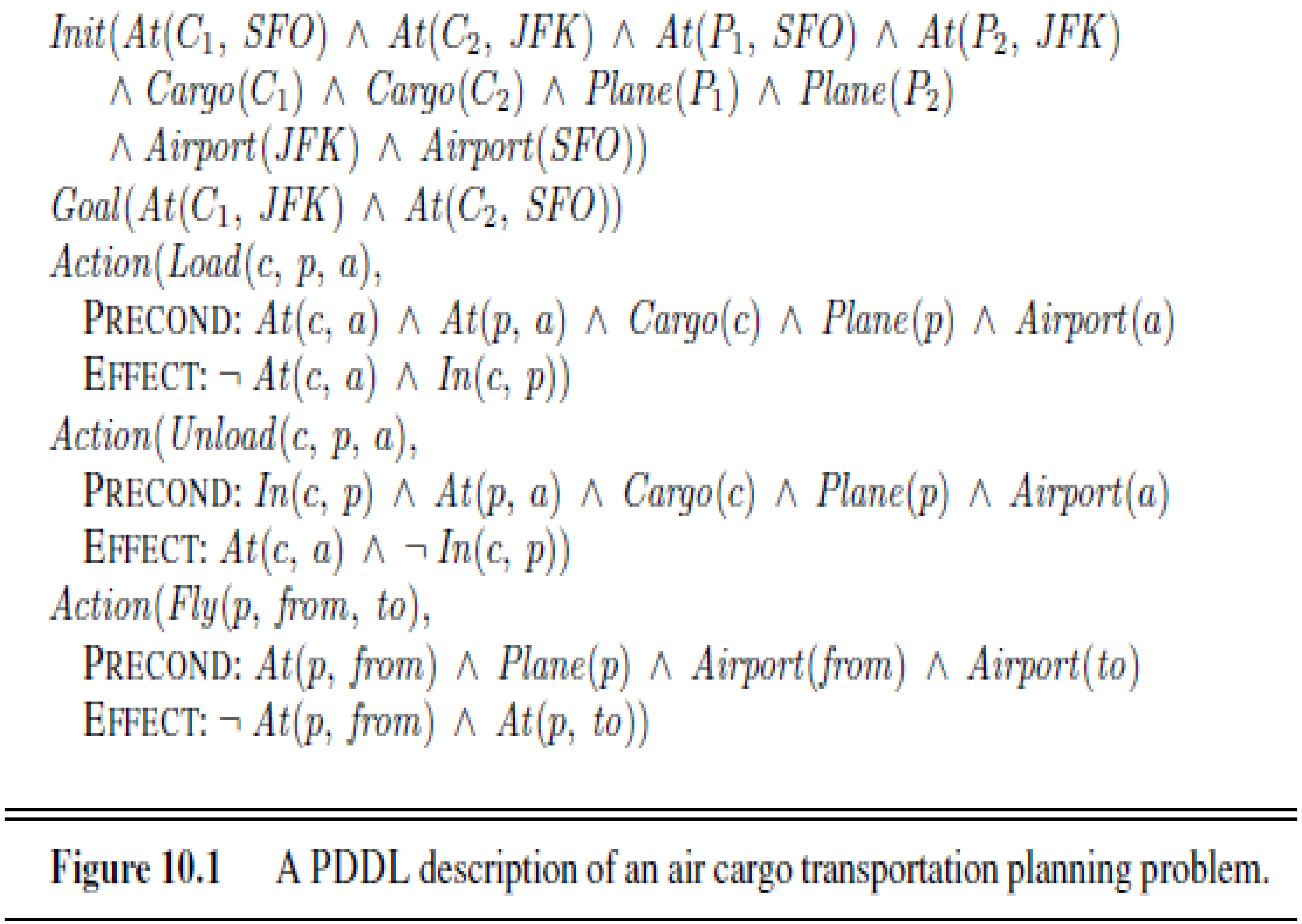
Example: Air cargo transport

- three actions: Load , Unload, and Fly
- The actions affect two predicates: In(c, p) means that cargo c is inside plane p, and At(x, a) means that object x (either plane or cargo) is at airport a.
- Note that some care must be taken to make sure the At predicates are maintained properly.

The following plan is a solution to the problem:

- [Load (C1, P1, SFO), Fly(P1, SFO, JFK),Unload(C1, P1, JFK),
- Load (C2, P2, JFK), Fly(P2, JFK, SFO),Unload(C2, P2, SFO)] .

- The following plan is a solution to the problem:
- [Load (C1, P1, SFO), Fly(P1, SFO, JFK),Unload(C1, P1, JFK),
- Load (C2, P2, JFK), Fly(P2, JFK, SFO),Unload(C2, P2, SFO)]



$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK)$
$\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)$
$\quad \land Airport(JFK) \land Airport(SFO))$
$Goal(At(C_1, JFK) \land At(C_2, SFO))$
$Action(Load(c, p, a),$
$\quad$ PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
$\quad$ EFFECT: $\neg At(c, a) \land In(c, p))$
$Action(Unload(c, p, a),$
$\quad$ PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
$\quad$ EFFECT: $At(c, a) \land \neg In(c, p))$
$Action(Fly(p, from, to),$
$\quad$ PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
$\quad$ EFFECT: $\neg At(p, from) \land At(p, to))$

Figure 10.1 A PDDL description of an air cargo transportation planning problem.

- Finally, there is the problem of spurious actions such as Fly(P1, JFK, JFK), which should be a no-op, but which has contradictory effects (according to the definition, the effect would include At(P1, JFK) ∧ ¬At(P1, JFK)).

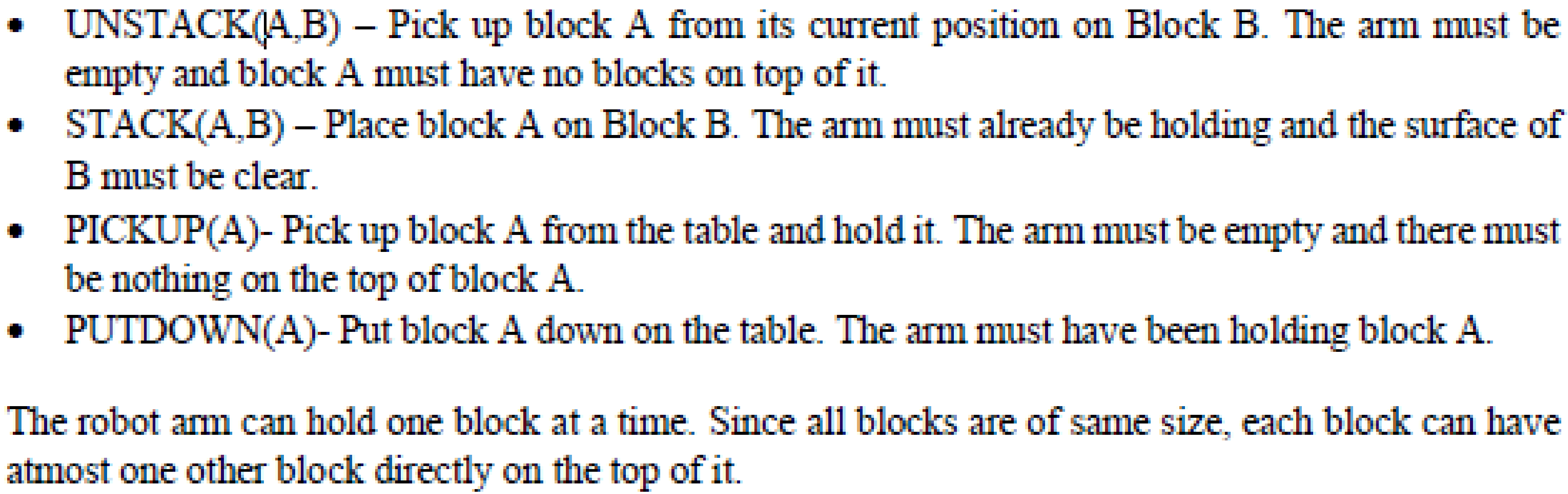
- It is common to ignore such problems, because they seldom cause incorrect plans to be produced.

- The correct approach is to add inequality preconditions saying that the from and to airports must be different;

Example: The blocks world

- This domain consists of a set of cube-shaped blocks sitting on a table.
- The blocks can be stacked, but only one block can fit directly on top of another.
- A robot arm can pick up a block and move it to another position, either on the table or on top of another block.
- The arm can pick up only one block at a time, so it cannot pick up a block that has another one on it.
- The goal will always be to build one or more stacks of blocks, specified in terms of what blocks are on top of what other blocks

Actions



- UNSTACK(A,B) – Pick up block A from its current position on Block B. The arm must be empty and block A must have no blocks on top of it.
- STACK(A,B) – Place block A on Block B. The arm must already be holding and the surface of B must be clear.
- PICKUP(A)- Pick up block A from the table and hold it. The arm must be empty and there must be nothing on the top of block A.
- PUTDOWN(A)- Put block A down on the table. The arm must have been holding block A.

The robot arm can hold one block at a time. Since all blocks are of same size, each block can have atmost one other block directly on the top of it.

For the operations we need the following predicates.

- ON(A,B) – Bock A is on Block B
- ONTABLE(A)- block A is on table
- CLEAR(A)- there is nothing on the top of A
- HOLDING(A)- The arm is holding block A
- ARMEMPTY- The arm is holding nothing

Some logical statements:

$$[\exists x{:}\mathrm{HOLDING}(x)] \rightarrow \neg\, \mathrm{ARMEMPTY}$$

If the arm is holding anything then it is not empty.

$$\forall x{:}\ \mathrm{ONTABLE}(x) \rightarrow \neg\, \exists y{:}\mathrm{ON}(x,y)$$

If a block is on table, then it is not also on another block

$$\forall x{:}\ [\neg\, \exists y{:}\ \mathrm{ON}(y,x)\ ] \rightarrow \mathrm{CLEAR}(x)$$

Any block with no block on it is clear

Components of Planning Systems

1. **Choosing Rules to apply**

The most widely used technique for selecting rules to apply is first to isolate a set of differences between the desired goal state and the current state and then to identify those rules that are relevant to reducing those differences.

If several rules are found, a variety of other heuristic information can be exploited to choose among them.

Means end analysis- Fencing example , if our goal is to have a white fence around out yard and we currently have a brown fence, we would select operators whose result involves a change of color of an object. If, on the other hand, we currently have no fencer, we must first consider operators that involve constructing wooden objects.

2. **Applying Rules**

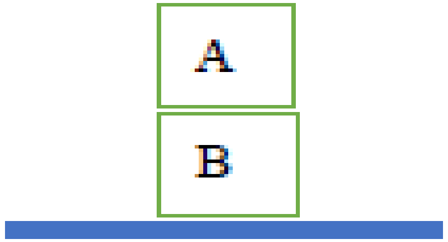
In simple systems applying rules was easy. Each rule simply specified the problem state that would result from its application.

We must be able to do with rules that specify only a small part of the complete problem state.

- For each action, describe the each of changes it makes to the state description. In addition, some statements, that everything else remains unchanged is also necessary.
- In a system, a given state was described by set of predicates representing the facts that were true in each state.
- Each distinct state was represented explicitly as part of the predicate.

Example the state S0 of sample blocks world problem could be represented.

On(A,B,S0) Λ ONTABLE(B,S0) Λ CLEAR(A,S0)



The manipulation of these state descriptions was done using a resolution theorem prover. So, for example, the effect of the operator UNSTACK(x,y) could be described by the following axiom.

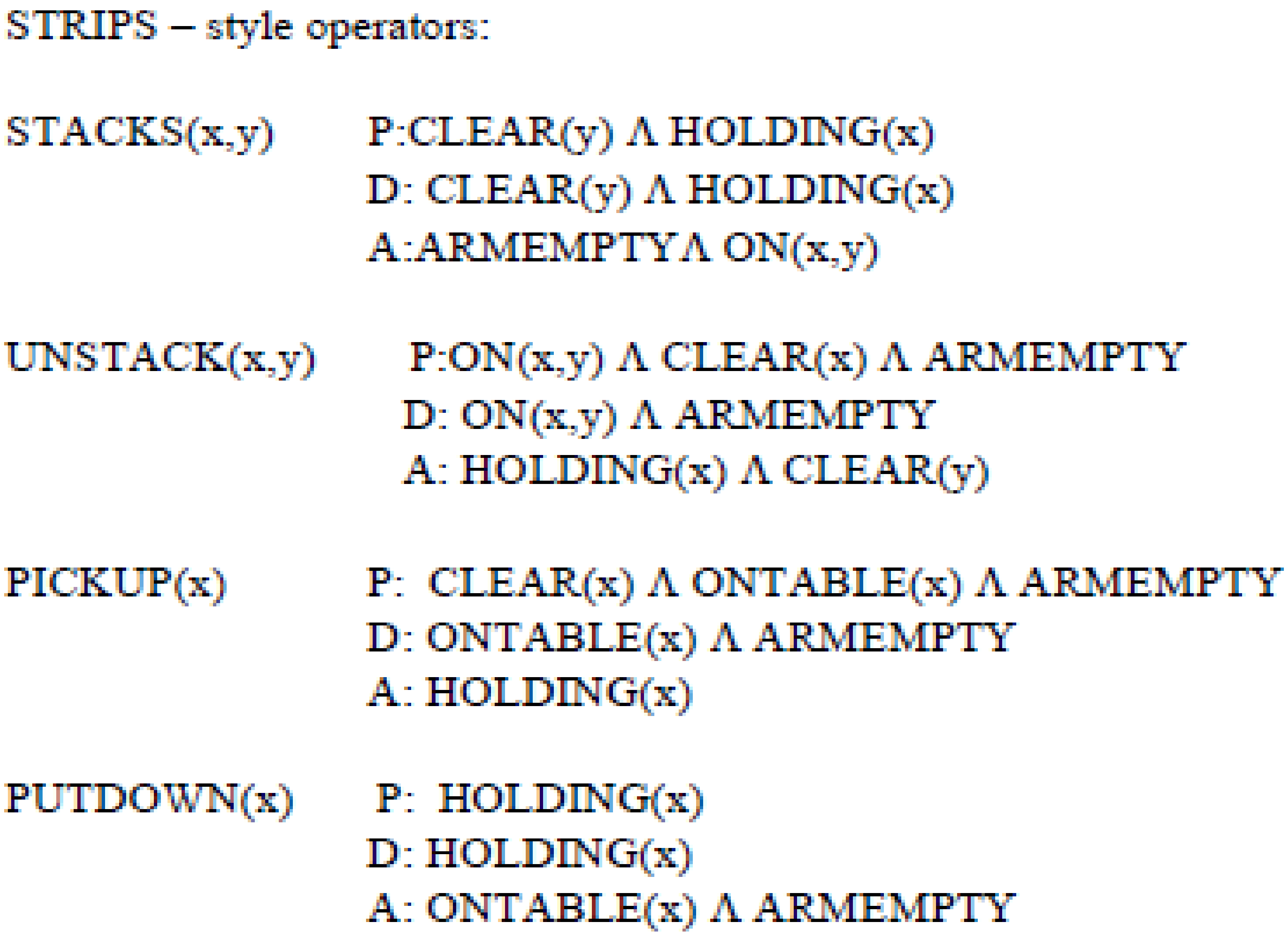
[CLEAR(x,s) Λ ON(x,y,s)] → [HOLDING(x,DO(UNSTACK(x,y),s)) Λ
CLEAR(y,DO(UNSTACK(x,y,s))]

Here DO is a function that specifies , for a given state and a given action, the new state that results from the execution of the action. The axiom state that if CLEAR(x) and ON(x,y) both hold in state s, then HOLDING(x) and CLEAR(y) will hold in the state that results from Doing an UNSTACK(x,y), starting in state s.

- If we execute UNSTACK(A,B) in state S0 as defined above, then we can prove , using our assertions about S0 and our axiom about UNSTACK, that in the state results from unstacking operarion (we call this state S1)
- HOLDING(A,S1) CLEAR(B,S1)
- We know that B still on the table. But what we have here so far, we cannot derive it. We need set of rules to prove it, called frame axioms. , that describe components of the state that are not affected by each operator.

- To handle complex problem domains, we need a mechanism that does not require a large number of explicit frame axioms. One such mechanism is that used by the early robot problem solving system STRIPS and its descendants.
- In these approach, each operation is described by a list of new predicates that the operator causes to become true and a list of old predicates that it causes to become false.
- These two lists are called ADD and DELETE lists, respectively. Any predicate not included on either the ADD or DELETE list of an operator is assumed to be unaffected by it.

P: PRE-CONDITION, A:ADD, D:DELETE



STRIPS – style operators:

STACKS(x,y) P:CLEAR(y) Λ HOLDING(x)
D: CLEAR(y) Λ HOLDING(x)
A:ARMEMPTYΛ ON(x,y)

UNSTACK(x,y) P:ON(x,y) Λ CLEAR(x) Λ ARMEMPTY
D: ON(x,y) Λ ARMEMPTY
A: HOLDING(x) Λ CLEAR(y)

PICKUP(x) P: CLEAR(x) Λ ONTABLE(x) Λ ARMEMPTY
D: ONTABLE(x) Λ ARMEMPTY
A: HOLDING(x)

PUTDOWN(x) P: HOLDING(x)
D: HOLDING(x)
A: ONTABLE(x) Λ ARMEMPTY

3. Detecting a solution

- Planning system is succeeded in finding a solution to the problem, when the sequence of operation that transforms the initial state to goal state is found.
- In simple problem we can check the completion of task by goal test. But for complex problem, it depends on the way that state descriptions are represented.
- The representation scheme used must have capability to reason with representations to discover whether one matches another.
- Predicate logic has that capacity, with **detective mechanism**.
    - Assume a predicate P(x).
    - To see whether P(x) is satisfied in some state, we ask whether we can prove P(x) given the assertions that describe the state and the axioms that define the world model (such arm holding something or not empty) If we construct such a proof then the problem-solving process terminates.
    - If we cannot, then a sequence of operators that might solve the problem must be proposed.
    - These sequence can then be tested in the same way as the initial state was by asking whether P(x) can be proved from the axioms and the state description that was derived by applying the operators.

4.Detecting the dead ends

- Detection of dead end is important. The same reasoning mechanism can be used to find dead end.
- If the search process is reasoning forward from the initial state, it can prune any path that leads to a state from which the goal state cannot be reached.
- **Painting example**, suppose we have a fixed supply of paint: some white some pink and some red.
- We want to paint a room so that it has light red walls and a white ceiling.
- We could produce light red paint by adding some white paint to red. But then we cannot paint the ceiling with white.
- This approach (red+white) should be abandoned in favour of mixing the pink and red paints together. Prune the path and take the path closer to solution.

5.Repairing an almost correct solution

- One good way of solving decomposable problems is to assume that they are completely decomposable and find the subproblems to be solved.
- Still the solution is not possible then look for another one. Slightly different approach, do sequence of some operations which leads to closer to goal state and now try to solve by subproblem.
- Even better approach, is having specific knowledge about what went wrong and then apply to direct patch. ,
- such as inadequate operator can be invoked to satisfy precondition. The last one is least-commitment strategy.
- Instead of following order of for preconditions, look the effect of each sub solutions to determine the dependencies that exist among them . at that point , an ordering can be chosen.

Goal stack planning

- One of the earlier planning algorithms called goal stack planning. It was used by STRIPS.
- We work backwards from the goal, looking for an operator which has one or more of the goal literals as one of its effects and then trying to satisfy the preconditions of the operator.
- The preconditions of the operator become subgoals that must be satisfied. We keep doing this until we reach the initial state.
- Planning is process of determining various actions that often lead to a solution.
- Planning is useful for non-decomposable problems where subgoals often interact.
- Goal Stack Planning (in short GSP) is the one of the simplest planning algorithm that is designed to handle problems having compound goals. And it utilizes STRIP as a formal language

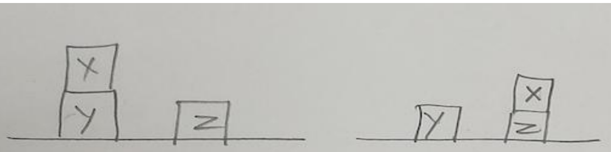
- This approach uses a Stack for plan generation. The stack can contain Sub-goal and actions described using predicates. The Sub-goals can be solved one by one in any order.
    - Push the Goal state in to the Stack
    - Push the individual Predicates of the Goal State into the Stack
    - Loop till the Stack is empty
        - Pop an element E from the stack
        - IF E is a Predicate
            - IF E is True then
                - Do Nothing
            - ELSE
                - Push the relevant action into the Stack
                - Push the individual predicates of the Precondition of the action into the Stack
        - Else IF E is an Action
            - Apply the action to the current State.
            - Add the action ‘a’ to the plan

Goal Stack-**Explanation:**

- The Goal Stack Planning Algorithms works with the stack.
- It starts by pushing the unsatisfied goals into the stack.
- Then it pushes the individual subgoals into the stack and its pops an element out of the stack.
- When popping an element out of the stack the element could be either a predicate describing a situation about our world or it could be an action that can be applied to our world under consideration.
- So based on the kind of element we are popping out from the stack a decision has to be made.
- If it is a Predicate. Then compares it with the description of the current world, if it is satisfied or is already present in our current situation then there is nothing to do because already its true.
- On the contrary if the Predicate is not true then we have to select and push relevant action satisfying the predicate to the Stack.

Goal Stack-**Explanation:**

- So after pushing the relevant action into the stack its precondition should also has to be pushed into the stack.
- In order to apply an operation its precondition has to be satisfied.
- In other words the present situation of the world should be suitable enough to apply an operation.
- For that, the preconditions are pushed into the stack once after an action is pushed.



Initial Goal.

Ontable(Y) ∧ Ontable(z)
∧ On (X, Y) ∧ clear(X)
∧ Clear(z)

Ontable(Y) ∧ Ontable(z)
∧ On(X, Z) ∧ clear(Y)
∧ clear(X)

For Goal stack planning start from goal state. put the predicates into stack.

Clear(X)
Clear(Y)
On(X, z)
On(X, Z) ∧ Clear(Y) ∧ clear(X)

Goal state

~~clear(X)~~
~~clear(Y)~~
On(X, Z)

Remove the predicates which are true

Now On(X, Z) is not true according to ~~initial~~ Current world. So push the actions to achieve the goal of On(X, Z)

Stack(X, Z)
On(X, Z)

→ Now push the pre conditions of Stack(X, Z)

Clear(~~z~~) – P
Holding(X) – P
Stack(X, Z)
On(X, Z)

P: preconditions added, should become true. Clear(z) is true, So remove it.

~~Clear(z)~~ – P
Holding(x) – P
Stack(X, Z)
On(X, Z)

Clear(z) – true pop it

Holding(x) is not true. So add the ~~predicate~~ actions to achieve it.

~~Holding(x)~~
Stack(X, Z)
On(X, Z)

unstack(X, Y)
Stack(X, Z)
On(X, Z)

add the actions of unstack(X, Y)

~~On(X, Y)~~ – P
~~Clear(X)~~ – P
~~ARMEMPTY~~ – P
unstack(X, Y)
Stack(X, Z)
On(X, Z)

→ All 3 pre conditions are true, So pop ~~it~~ them

~~unstack(X, Y)~~
Stack(X, Z)
On(X, Z)

plan = {(unstack (X, Y)} – apply the action at current world.

plan = {(unstack X, Y), stack (X, Z)}

~~Stack(X, Z)~~
On(X, Z)

~~On(X, Z)~~

On(X, Z) is true, pop it

Thus reached goal state.