ARTIFICIAL INTELLIGENCE
(CS5402)

Name: Lakhan Kumawat

Roll: 1906055

Program: B.Tech CSE
(5th Sem JUL-DEC 2021)

AI Assignments
I & II

Q1. Explain the stochastic and evolutionary search algorithms used for problem-solving through search

Solution:

STOCHASTIC SEARCH ALGORITHM:-

Stochastic search algorithms are designed for problems with inherent random noise or deterministic problems solved by injected randomness.

Stochastic Search is an optional solution to many complex integration issues.

Stochastic optimization (SO) methods are development methods that create and use random variables.

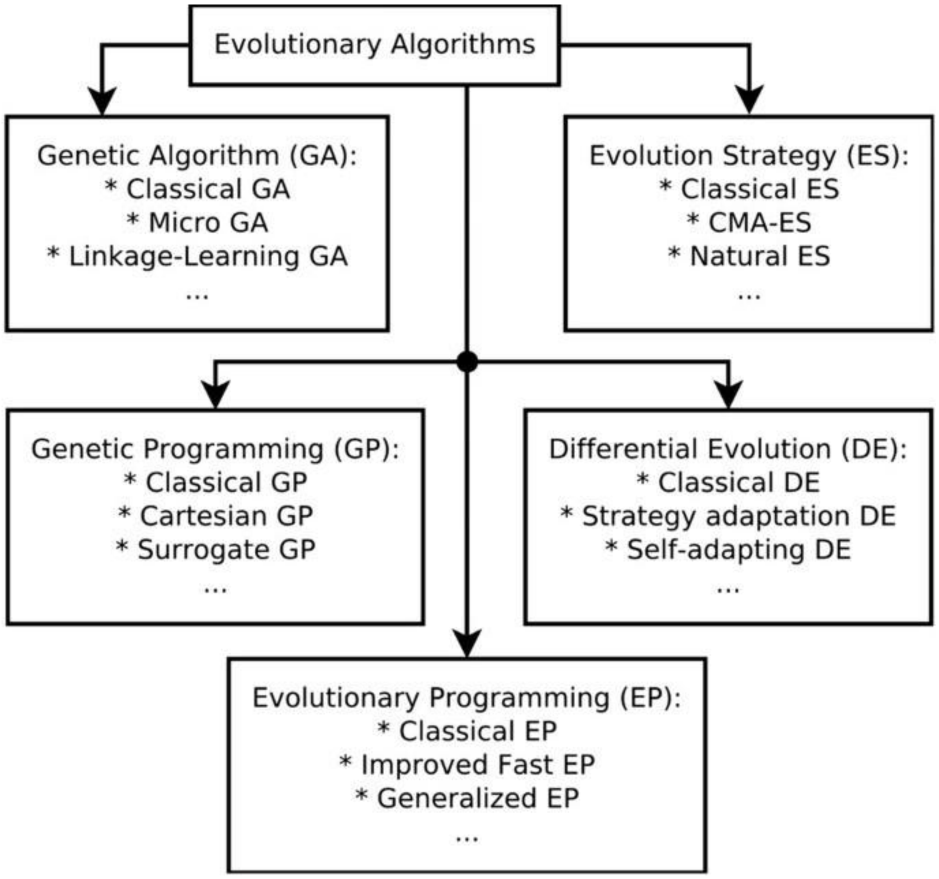
In stochastic problems, random variables arise from the development of the development problem itself, involving random objective tasks or random limits.

Stochastic optimization methods also include methods with random iterates.

Other stochastic efficiency methods use random iterates to solve stochastic problems, combining both definitions of stock efficiency.

Stochastic development strategies typically formulate specific approaches to determining problems.

EVOLUTIONARY ALGORITHMS:



In Computational Intelligence (CI), EA is a subset of Evolutionary computation, as well as a common human-based metaheuristic development algorithm.

Methods: biological evolution such as reproduction, genetic modification, reunification, and selection.

Evolutionary algorithms often make solutions for almost all types of problems because they do not make any assumptions about the area of competency.

TYPES OF EA:-

- Genetic Algorithm (GA)
- Genetic programming
- Evolutionary programming
- Evolution strategy
- Differential evolution
- Neuroevolution
- Learning classifier system

Q.2 Describe the use of local consistency and its types in the inference in CSPs of Constraint Propagation. Also explain how backtracking search is used for CSPs.

SOLUTION:-

Constraint satisfaction problems (CSPs) are mathematical questions defined as a set of objects whose state must satisfy a number of constraints or limitations. CSPs represent the entities in a problem as a homogeneous collection of finite constraints over variables, which is solved by constraint satisfaction methods.

CSP (constraint satisfaction problem): Use a combined representation (a set of variables, each with a value) for each situation, a problem solved

where each variable has a value that satisfies all constraints in the variation called CSP.
CSP consists of 3 sections:

· X is a set of variables, {X1,..., Xn}.

· D set of domains, {D1,..., Dn}, one for each variation.

Each Di domain contains a set of valid values, {v1,..., vk} of variable Xi.

· C set of constraints that define a valid combination of values.

Each Ci field contains a pair <width, rel>, where the width is a tuple of the variables that participate in the block, and the rel is a relationship that defines the values that those words can take.

Relationships can be represented as follows: a. a clear list of all limit tuples; or b. an invisible relationship that supports two functions. (eg if X1 and X2 both have the domain {A, B}, the limit for "two variables must have different values" can be written as. <(X1, X2), [(A, B), (B), A)]> or b. <(X1, X2), X1 ≠ X2>

Work to be done:

Each region in the CSP is defined by a numerical allocation of some of the variables, {Xi = vi, Xj = vj,...};

Work to be done that does not break any barriers is called a fixed or official assignment;

The full assignment is where all the changes are assigned;

CSP solution is a consistent, complete function;

A partial assignment is one that assigns prices to some of the variables only.

Backtracking search, a type of deep search, is often used to resolve CSPs. Touch can be combined with search.

Commutativity: All CSPs are flexible. The problem changes if the order of use of any set of actions does not affect the outcome.

Reverse search: Advanced search that selects single variable values at a time and in volumes when the variable has no legal values remaining to be assigned.

The regression algorithm repeatedly selects an unallocated version, and then tries all the values in the domain of that variable, trying to find a solution. If a discrepancy is found, then BACKTRACK returns the failure, causing the previous call to attempt another value.

There is no need to provide BACKTRACKING-SEARCH for a specified initial domain status, action function, change model, or goal test.

BACKTRACKING-SARCH retains only one state representation and replaces that representation with a new one.

Graph Coloring Problem is an example of CSP:

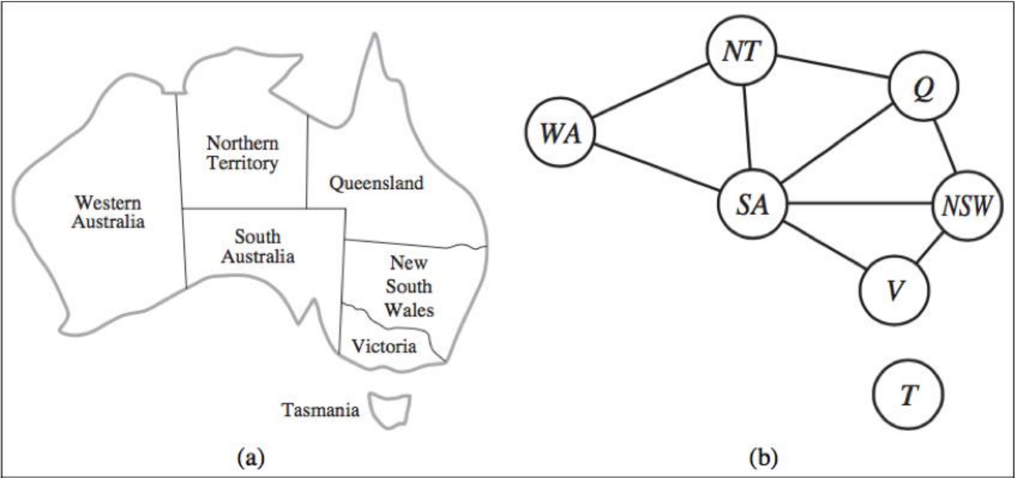


Figure 6.1 (a) The principal states and territories of Australia. Coloring this map can be viewed as a constraint satisfaction problem (CSP). The goal is to assign colors to each region so that no neighboring regions have the same color. (b) The map-coloring problem represented as a constraint graph.

Q.3 Differentiate between propositional vs. first-order logic with inferencerules. Write the forward and backward chaining algorithms and tree structure solutions(diagram)for the following knowledge-based automated reasoning agent and prove that West is a criminal:

"The law says that it is a crime for an American to sell weapons to hostile nations. Thecountry Nono, an enemy of America, has some missiles, and all of its missiles were soldto it by Colonel West, who is American."

SOLUTION:

Propositional Logic (PL)

A propositional logic is a statement of analysis that may be true or false. It is basically a way of representing knowledge in a logical and mathematical way. There are two types of PropositionaL logic: Atomic Proposals and Compounds.

First-Order Logic (FOL)

First-Order Logic is another knowledge representation in AI which is an extended part of PL. FOL articulates the natural language statements briefly. Another name of First-Order Logic is 'Predicate Logic'.

Differences between PL and FOL:

- Propositional Logic converts a complete sentence into a symbol and makes it logical whereas in First-Order Logic relation of a particular sentence will be made that involves relations, constants, functions, and constants.
- The limitation of PL is that it does not represent any individual entities whereas FOL can easily represent the individual establishment that means if you are writing a single sentence then it can be easily represented in FOL.

- PL does not signify or express the generalization, specialization or pattern for example 'QUANTIFIERS' cannot be used in PL but in FOL users can easily use quantifiers as it does express the generalization, specialization, and pattern.

Forward Chaining

Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.

The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

Properties of Forward-Chaining:

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

Example:

"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen."

Prove that "Robert is criminal."

To solve the above problem, first, we will convert all the above facts into first-order definite clauses, and then we will use a forward-chaining algorithm to reach the goal.

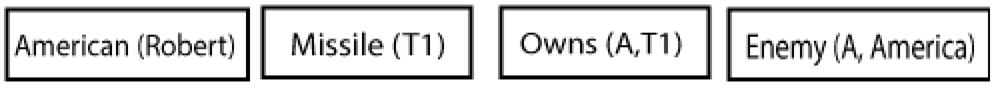
Facts Conversion into FOL:

- It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)
  American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)
- Country A has some missiles. ?p Owns(A, p) ∧ Missile(p). It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.
  Owns(A, T1) ......(2)
  Missile(T1) ....(3)
- All of the missiles were sold to country A by Robert.
  ?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) ......(4)
- Missiles are weapons.
  Missile(p) → Weapons (p) ....(5)
- Enemy of America is known as hostile.
  Enemy(p, America) →Hostile(p) ....(6)
- Country A is an enemy of America.
  Enemy (A, America) ....(7)
- Robert is American
  American(Robert). ....(8)

Forward chaining proof:

Step-1:

In the first step we will start with the known facts and will choose the sentences which do not have implications, such as: American(Robert), Enemy(A, America), Owns(A, T1), and Missile(T1). All these facts will be represented as below.



Step-2:

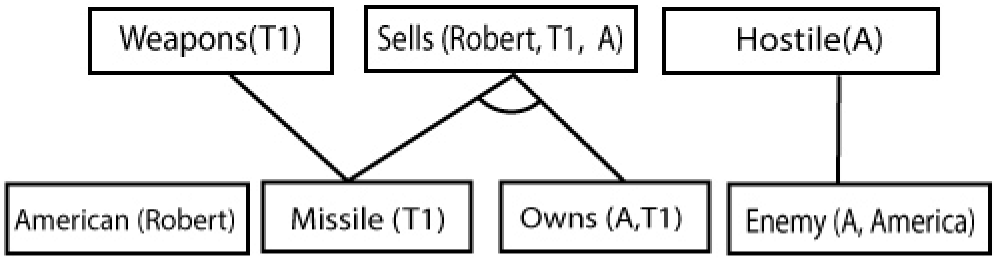
At the second step, we will see those facts which infer from available facts and with satisfied premises.

Rule-(1) does not satisfy premises, so it will not be added in the first iteration.

Rule-(2) and (3) are already added.

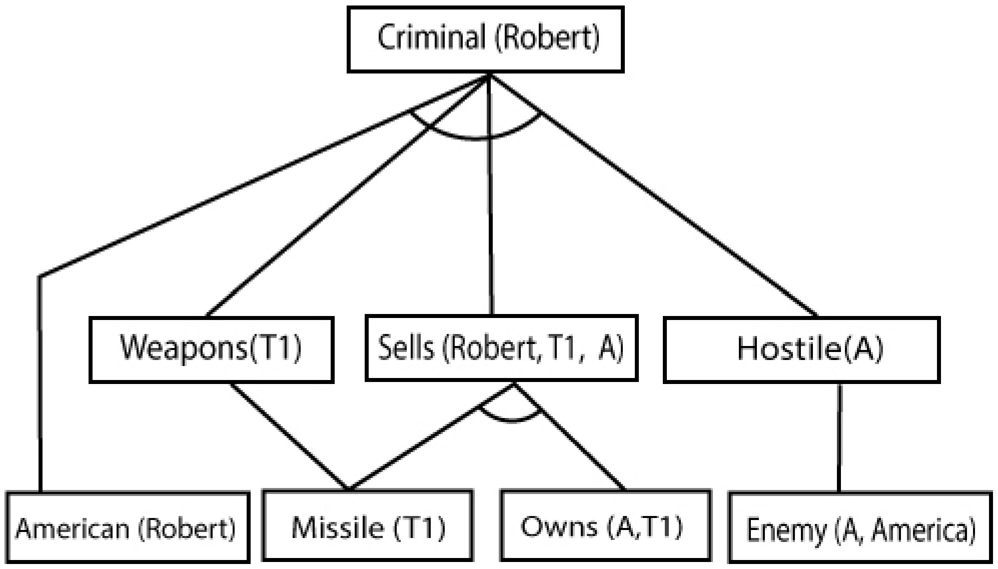
Rule-(4) satisfy with the substitution {p/T1}, so Sells (Robert, T1, A) is added, which infers from the conjunction of Rule (2) and (3).

Rule-(6) is satisfied with the substitution(p/A), so Hostile(A) is added and which infers from Rule-(7).



Step-3:

At step-3, as we can check Rule-(1) is satisfied with the substitution {p/Robert, q/T1, r/A}, so we can add Criminal(Robert) which infers all the available facts. And hence we reached our goal statement.



Hence it is proved that Robert is Criminal using forward chaining approach.

B. Backward Chaining:

Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

Properties of backward chaining:

- It is known as a top-down approach.
- Backward-chaining is based on modus ponens inference rule.
- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
- Backward -chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.
- The backward-chaining method mostly used a depth-first search strategy for proof.

Example:

In backward-chaining, we will use the same above example, and will rewrite all the rules.

- American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)
  Owns(A, T1) ....(2)
- Missile(T1)
- ?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) ......(4)
- Missile(p) → Weapons (p) ....(5)

- Enemy(p, America) →Hostile(p) ....(6)
- Enemy (A, America) ....(7)
- American(Robert). ....(8)

Backward-Chaining proof:

In Backward chaining, we will start with our goal predicate, which is Criminal(Robert), and then infer further rules.

Step-1:

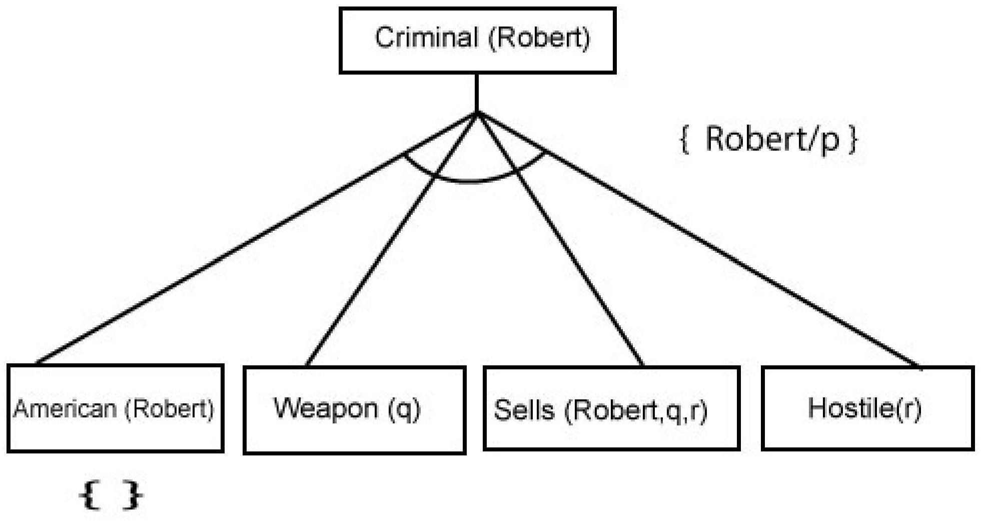
At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.

Criminal (Robert)

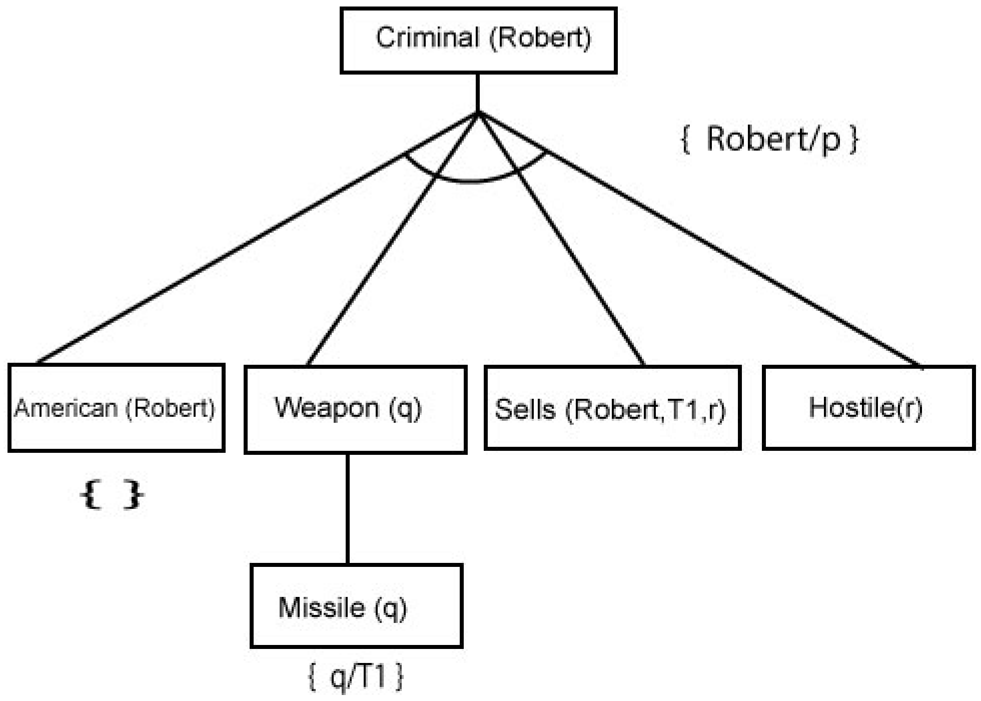
Step-2:

At the second step, we will infer other facts form goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution {Robert/P}. So we will add all the conjunctive facts below the first level and will replace p with Robert.

Here we can see American (Robert) is a fact, so it is proved here.

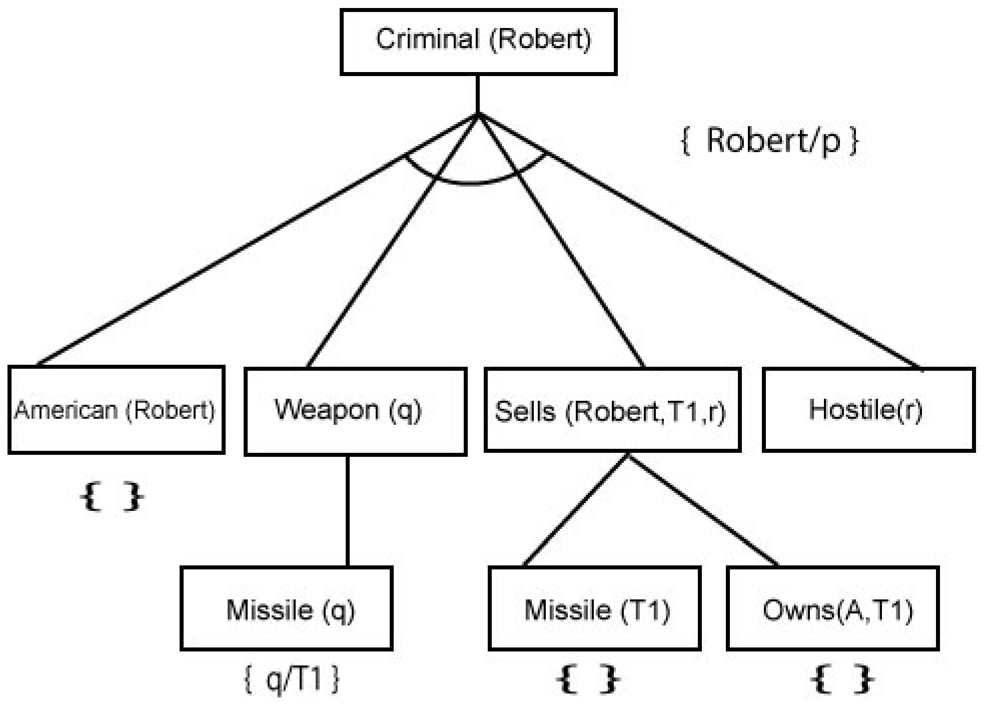


Step-3:t At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.



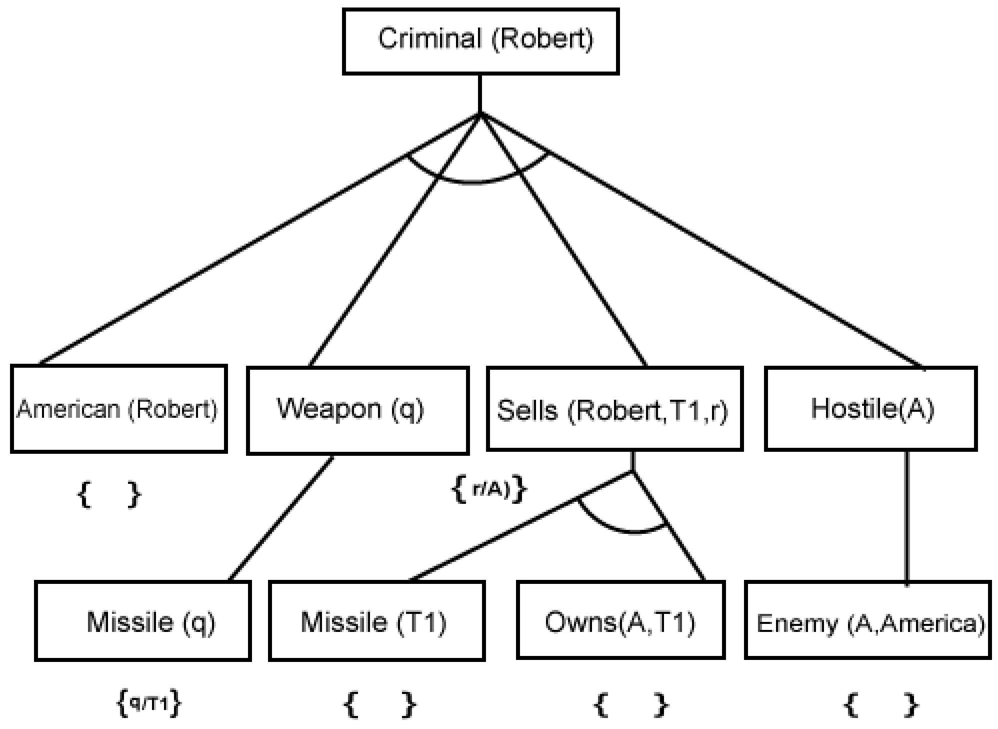
Step-4:

At step-4, we can infer facts Missile(T1) and Owns(A, T1) form Sells(Robert, T1, r) which satisfies the Rule- 4, with the substitution of A in place of r. So these two statements are proved here.



Step-5:

At step-5, we can infer the fact Enemy(A, America) from Hostile(A) which satisfies Rule- 6. And hence all the statements are proved true using backward chaining.



Q.4 Write steps to find out resolution of the conjunctive normal form(CNF)for the first-order logic with the help of suitable example and also discuss the resolution inference rule with its completeness proof of resolution.

SOLUTION:

Conjunctive Normal Form: A sentence represented as a combination of clauses is called a conjunctive normal form or CNF.

The resolution inference rule:

The resolution rule for first-order logic is simply a lifted version of the propositional rule. Resolution can resolve two clauses if they contain complementary literals, which are assumed to be standardized apart so that they share no variables.

$$\frac{l_1 \vee \dots\dots \vee l_k, \qquad m_1 \vee \dots\dots \vee m_n}{\text{SUBST}(\theta, l_1 \vee \dots\dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots\dots \vee m_{j-1} \vee m_{j+1} \vee \dots\dots \vee m_n)}$$

Where l_i and m_j are complementary literals.

This rule is also called the binary resolution rule because it only resolves exactly two literals.

Example:

We can resolve two clauses which are given below:

[Animal (g(x) V Loves (f(x), x)] and [¬ Loves(a, b) V ¬Kills(a, b)]

Where two complimentary literals are: Loves (f(x), x) and ¬ Loves (a, b)

These literals can be unified with unifier θ= [a/f(x), and b/x] , and it will generate a resolvent clause:

[Animal (g(x) V ¬ Kills(f(x), x)].

Steps for Resolution:

1. Conversion of facts into first-order logic.
2. Convert FOL statements into CNF
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

To better understand all the above steps, we will take an example in which we will apply resolution.

Example:

a. John likes all kind of food.

b. Apple and vegetable are food

c. Anything anyone eats and not killed is food.

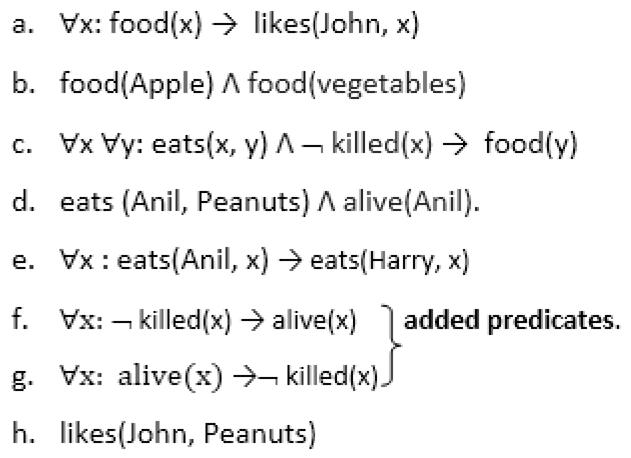
d. Anil eats peanuts and still alive

e. Harry eats everything that Anil eats.
Prove by resolution that:

f. John likes peanuts.

Step-1: Conversion of Facts into FOL

In the first step we will convert all the given statements into its first order logic.



a. ∀x: food(x) → likes(John, x)
b. food(Apple) ∧ food(vegetables)
c. ∀x ∀y: eats(x, y) ∧ ¬ killed(x) → food(y)
d. eats (Anil, Peanuts) ∧ alive(Anil).
e. ∀x : eats(Anil, x) → eats(Harry, x)
f. ∀x: ¬ killed(x) → alive(x) } **added predicates.**
g. ∀x: alive(x) → ¬ killed(x)
h. likes(John, Peanuts)

Step-2: Conversion of FOL into CNF

In First order logic resolution, it is required to convert the FOL into CNF as CNF form makes easier for resolution proofs.

- Eliminate all implication (→) and rewrite

a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)

d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀x ¬ eats(Anil, x) V eats(Harry, x)

f. ∀x¬ [¬ killed(x)] V alive(x)

g. ∀x ¬ alive(x) V ¬ killed(x)

h. likes(John, Peanuts).

Move negation (¬)inwards and rewrite

a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)

d. eats (Anil, Peanuts) ∧ alive(Anil)

e. ∀x ¬ eats(Anil, x) V eats(Harry, x)

f. ∀x ¬killed(x)] V alive(x)

g. ∀x ¬ alive(x) V ¬ killed(x)

h. likes(John, Peanuts).

Rename variables or standardize variables

a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) ∧ food(vegetables)

c. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)

d. eats (Anil, Peanuts) ∧ alive(Anil)

e. ∀w¬ eats(Anil, w) V eats(Harry, w)

f. ∀g ¬killed(g)] V alive(g)

g. ∀k ¬ alive(k) V ¬ killed(k)

h. likes(John, Peanuts).

Eliminate existential instantiation quantifier by elimination.
In this step, we will eliminate existential quantifier ∃, and this process is known as Skolemization. But in this example problem since there is no existential quantifier so all the statements will remain same in this step.

Drop Universal quantifiers.
In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.

a. ¬ food(x) V likes(John, x)

b. food(Apple)

c. food(vegetables)

d. ¬ eats(y, z) V killed(y) V food(z)

e. eats (Anil, Peanuts)

f. alive(Anil)

g. ¬ eats(Anil, w) V eats(Harry, w)

h. killed(g) V alive(g)

i. ¬ alive(k) V ¬ killed(k)

j. likes(John, Peanuts).

Note: Statements "food(Apple) Λ food(vegetables)" and "eats (Anil, Peanuts) Λ alive(Anil)" can be written in two separate statements.

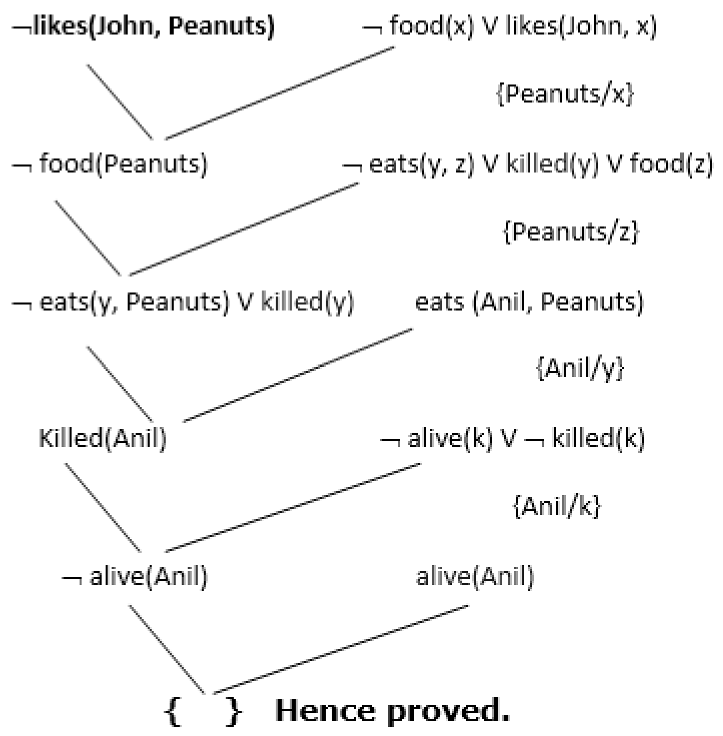
- Distribute conjunction ∧ over disjunction ¬.
  This step will not make any change in this problem.

Step-3: Negate the statement to be proved

In this statement, we will apply negation to the conclusion statements, which will be written as ¬likes(John, Peanuts)

Step-4: Draw Resolution graph:

Now in this step, we will solve the problem by resolution tree using substitution. For the above problem, it will be given as follows:



Hence the negation of the conclusion has been proved as a complete contradiction with the given set of statements.

Q.5 Define the blocks world example domain with suitable example and also describe the components of a planning system.Also describe the goal stack planning method with the help of suitable example.

SOLUTION:

The world contains:

Flat area as a table area

Adequate set of matching blocks identified by letters.

The blocks can be stacked against each other to form towers of seemingly infinite length.

The packaging is accessed using a robotic arm that has important functions and that can be tested intelligently and assembled using logical operation.

A robot can capture one block at a time, and only one block can be removed at a time.

We shall use the four actions:

UNSTACK(A,B)

-- pick up clear block A from block B;

STACK(A,B)

-- place block A using the arm onto clear block B;

PICKUP(A)

-- lift clear block A with the empty arm;

PUTDOWN(A)

-- place the held block A onto a free space on the table.

and the five predicates:

ON(A,B)

-- block A is on block B.

ONTABLE(A)

-- block A is on the table.

CLEAR(A)

-- block A has nothing on it.

HOLDING(A)

-- the arm holds block A.

ARMEMPTY

-- the arm holds nothing.

We use a logical block but not a logical comment we can say If the arm is holding the block is empty If block A is on the table it is not in any other block.

Why Should You Use the Blocks World as an Example?

The block world is selected because:

simple enough and well behaved.

easily understood

however it still provides a good sample of learning planning space:

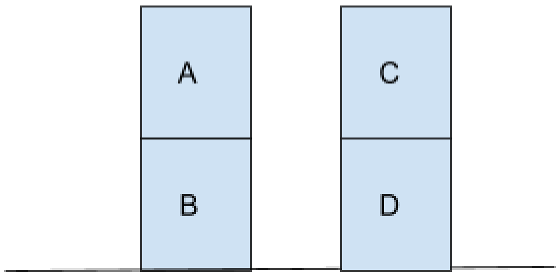
problems can be broken down into minor problems that are almost completely different

we can show how partial solutions should be put together to form a complete, practical solution.

Q.6: Examine the nonlinear planning using constraint posting with suitable example and differentiate it with goal-stack method.

SOLUTION:

Non Linear Planning with Constraint Posting



Initial State

A plan that consists of sub-problems, which are solved simultaneously is said to be a non-linear plan.

In case of the goal stack planning, as discussed previously, it poses some problems. Achieving a goal could possibly undo any of the already achieved goals and its called as Sussman`s anomaly.In linear planning, just one goal is taken at a time and solved completely before the next one is taken.

Let us take an example. You want to take the car for servicing and have to make an important phone call. In case of Linear planning, First you will achieve the goal of making a phone call and then will take the car for servicing. Rather than completing both the tasks in a linear way, after completion of the task 1, as partial step, i.e., start the car and put on the Bluetooth, then complete the task 2 of phone call and then finally, complete the task 1 by leaving the car at the service station. This can be an example of non-linear planning.

There is a technique called constraint posting that often comes with Non-Linear Planning.The idea of constraint posting is to build up a plan by incrementally,

1. Adding or Suggesting Operators

2. Ordering Operators
3. Binding the variable to the Operators

At any given time in problem solving process, we may have a set of useful operators but perhaps no clear idea of how those operators should be ordered with respect to each others.

Here a solution is partially ordered to generate the actual plan, we convert the partial order into any number of total order.

Let us now examine several operations for nonlinear planning in a constraint posting environment by incrementally generating a nonlinear plan to solve the Sussman anomaly problem.

We begin with a null plan. Next we look at the goal state and put forward steps for achieving that goal. Also we use heuristic in this plan and it is given below,

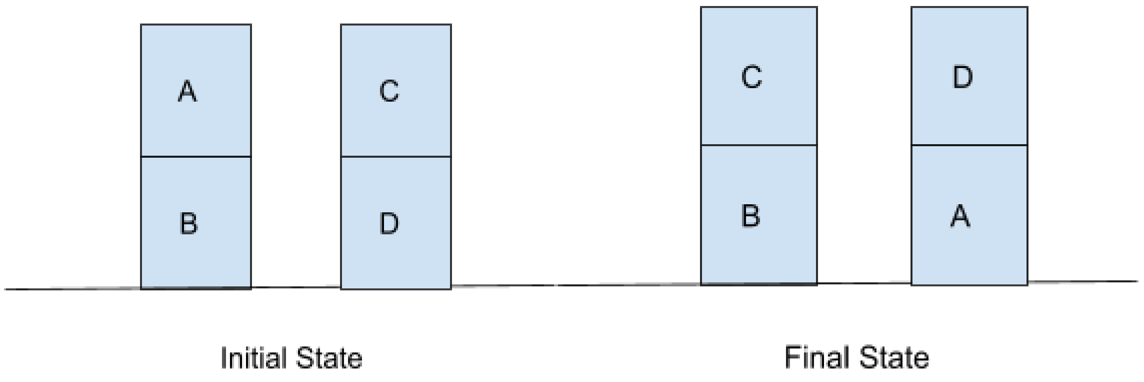
Step Addition : Creating new steps for a plan

Promotion : Restrict one step to come before another in a final plan.

DeClobbering : Placing one step s2 between two old steps s1 and s3 such that s2 makes it possible for s3 to be apply which was previously prevented by s1

Simple Establishment : Assigning a value to a variable.

Separation : Preventing the assignment of certain values to a variable.



ON(A,B) /\

ON (A,B) /\ ON (C,D) /\ ONTABLE(B)
/\ ONTABLE(D)

ON(C,D)

ON (C,B) /\ ON (D,A) /\ ONTABLE(B) /\
CLEAR(C) /\ CLEAR(D) /\ ONTABLE(A)

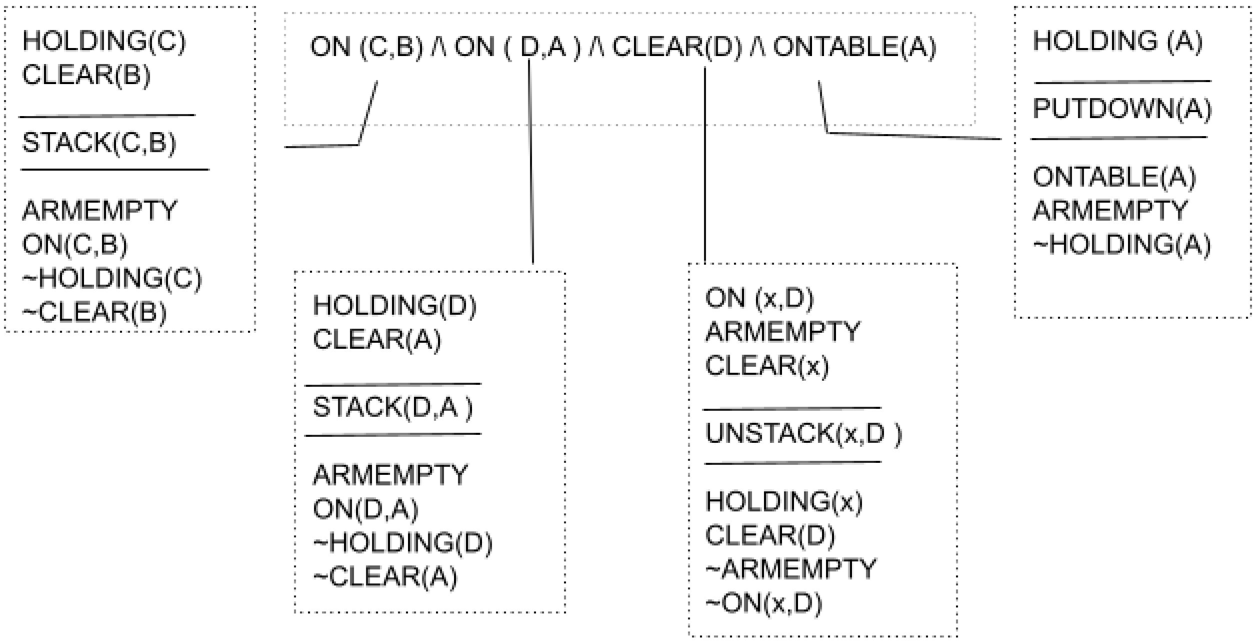
Let's take the initial state as our current state and start the plan creation with the goal state.

ON (C,B) /\ ON (D,A) /\ ONTABLE(B) /\
CLEAR(C) /\ CLEAR(D) /\ ONTABLE(A)

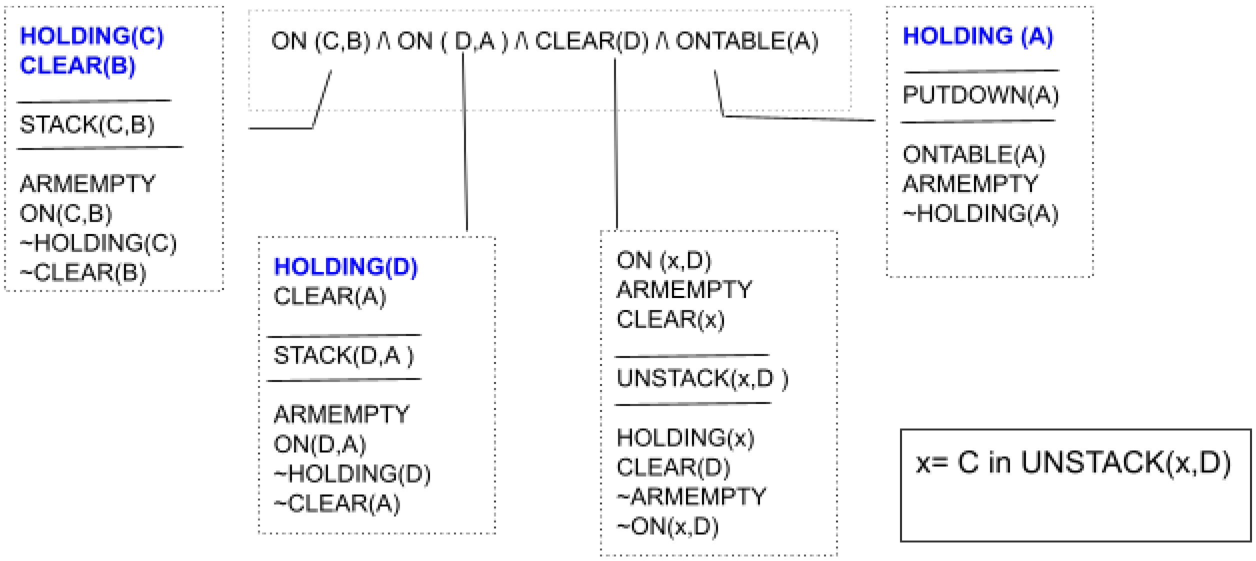
If we observe above we could understand that certain subgoals are already true and it is highlighted in Red. Therefore we don't have to consider that part in our process. So let's consider the unachieved goals,

ON (C,B) /\ ON (D,A) /\ CLEAR(D) /\ ONTABLE(A)

These three subgoals, as indicated in the above diagram, are not true in our current situation so use step addition to achieve these subgoals.



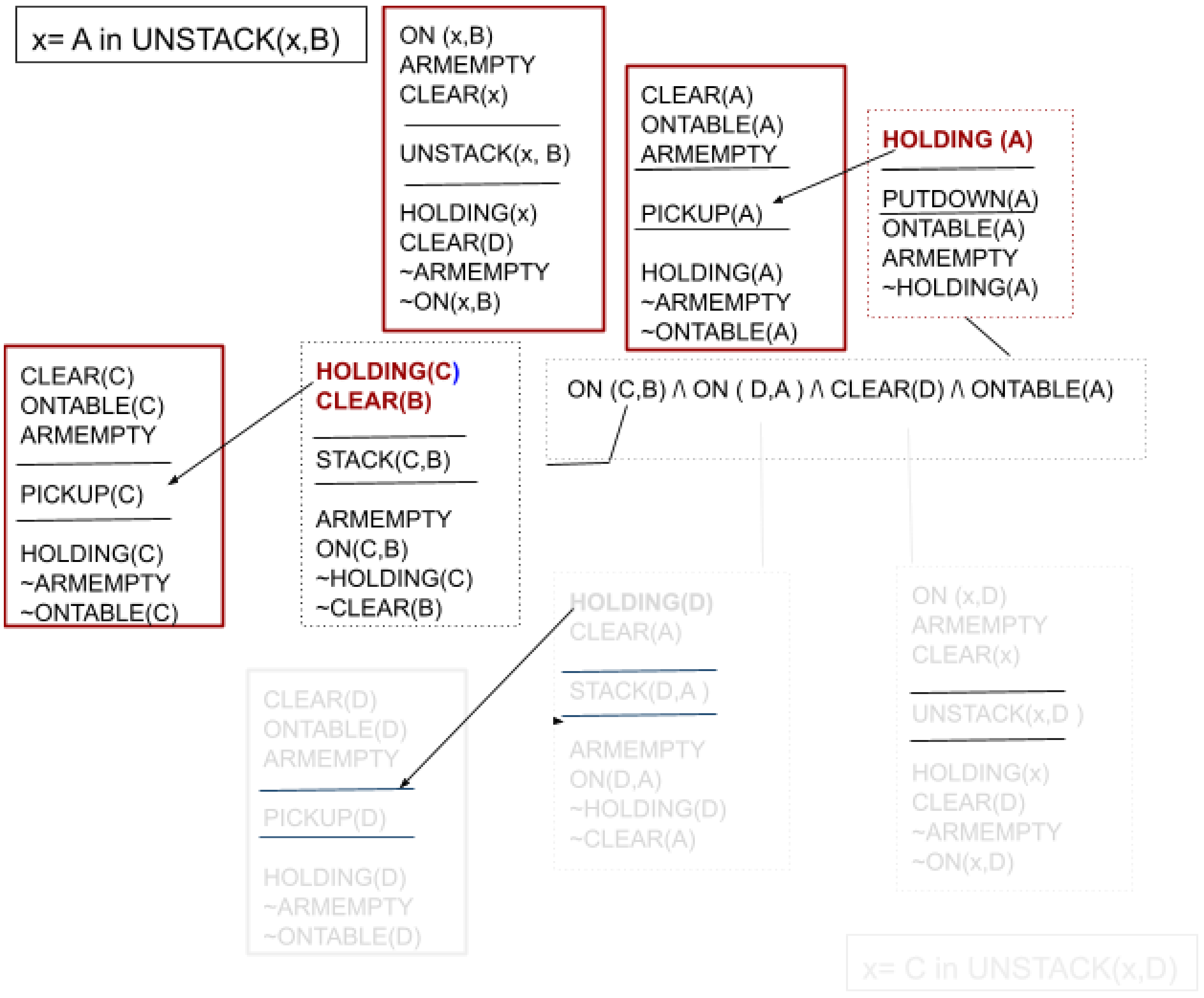
Now check the precondition of all the actions are satisfied or not. The unsatisfied preconditions are highlighted in below diagram.



Now let's try to achieve the preconditions that are highlighted above in the diagram so that various actions can be used to achieve the goal.

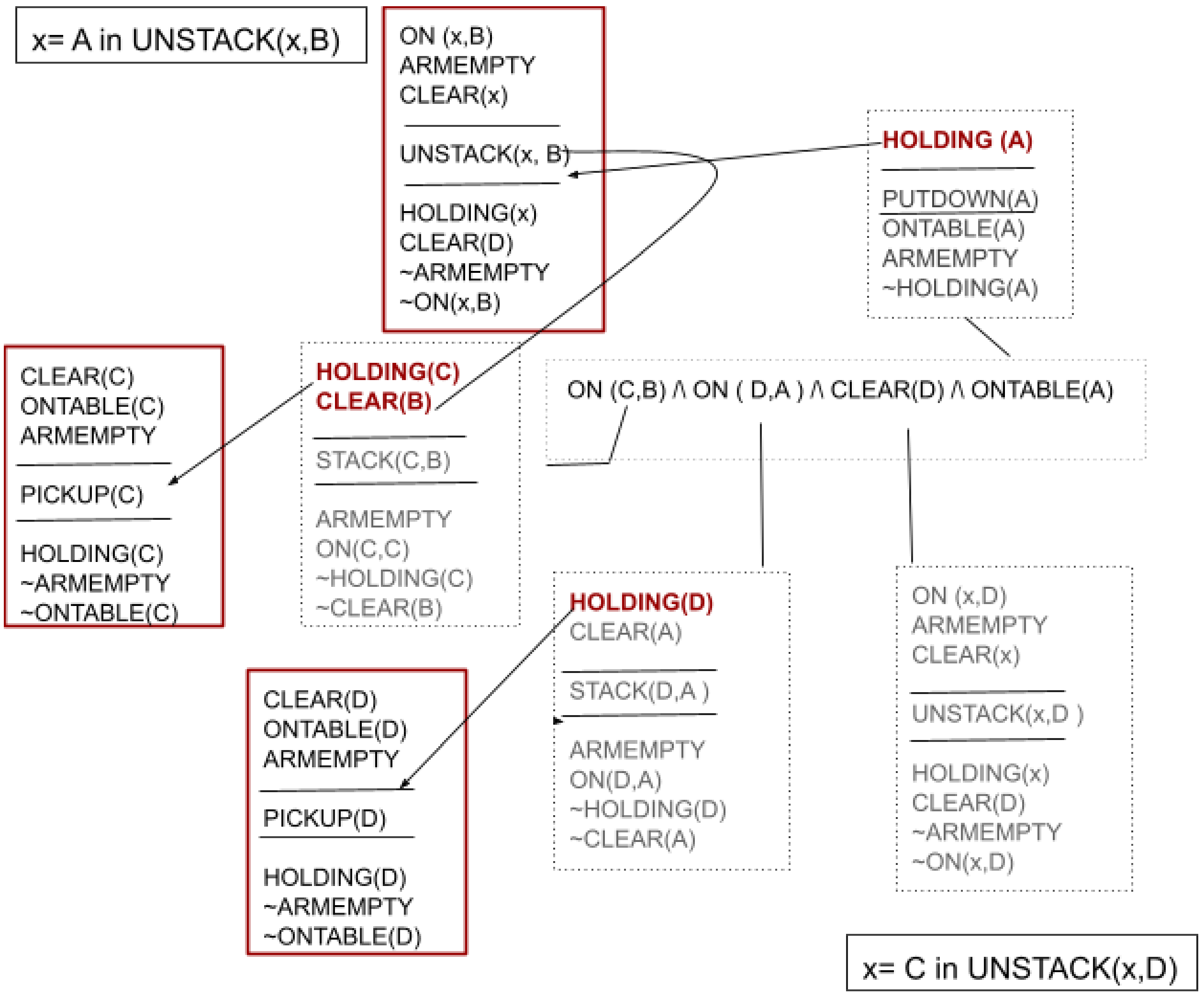
First look in to the action STACK(C,B) where HOLDING(C) and CLEAR(B) is not true. This is determined by checking it with our current situation, where the robotic arm is empty. So use Step Addition to satisfy HOLDING(C) and it can be done by using PICKUP(C) whose effect is HOLDING(C) also using Step Addition include UNSTACK(x,B) to satisfy

CLEAR(B). Similarly the preconditions of other action STACK(D,A) and PUTDOWN(A) are achieved by including relevant actions through step addition. Wait a moment, there is no need to use step addition to satisfy a goal or precondition all the time.We can offcourse use promotion if that too can help satisfying a goal. If both promotion and step addition could make the precondition/goal to be achieved then prefer promotion comparing to step addition. This could be observed in the below figure.


x= A in UNSTACK(x,B)
ON (x,B) ARMEMPTY CLEAR(x)
UNSTACK(x, B)
HOLDING(x) CLEAR(D) ~ARMEMPTY ~ON(x,B)
CLEAR(A) ONTABLE(A) ARMEMPTY
PICKUP(A)
HOLDING(A) ~ARMEMPTY ~ONTABLE(A)
HOLDING (A)
PUTDOWN(A)
ONTABLE(A) ARMEMPTY ~HOLDING(A)
CLEAR(C) ONTABLE(C) ARMEMPTY
PICKUP(C)
HOLDING(C) ~ARMEMPTY ~ONTABLE(C)
HOLDING(C) CLEAR(B)
STACK(C,B)
ARMEMPTY ON(C,B) ~HOLDING(C) ~CLEAR(B)
ON (C,B) Λ ON (D,A) Λ CLEAR(D) Λ ONTABLE(A)
HOLDING(D) CLEAR(A)
STACK(D,A)
ARMEMPTY ON(D,A) ~HOLDING(D) ~CLEAR(A)
ON (x,D) ARMEMPTY CLEAR(x)
UNSTACK(x,D)
HOLDING(x) CLEAR(D) ~ARMEMPTY ~ON(x,D)
CLEAR(D) ONTABLE(D) ARMEMPTY
PICKUP(D)
HOLDING(D) ~ARMEMPTY ~ONTABLE(D)
x= C in UNSTACK(x,D)

HOLDING(A) in action PUTDOWN(A) is not true so we could achieve it by including PICKUP(A) but do we really need step addition here. If we could see that the UNSTACK(A,y) also will make the HOLDING(A) to be true right. So we have included UNSTACK(A,B) already in our a partial plan. So just use promotion here by suggesting that UNSTACK(A,B) comes before

PUTDOWN(A). By this we can possibly eliminate the additional action. So the modified partial plan will be like the one given below.

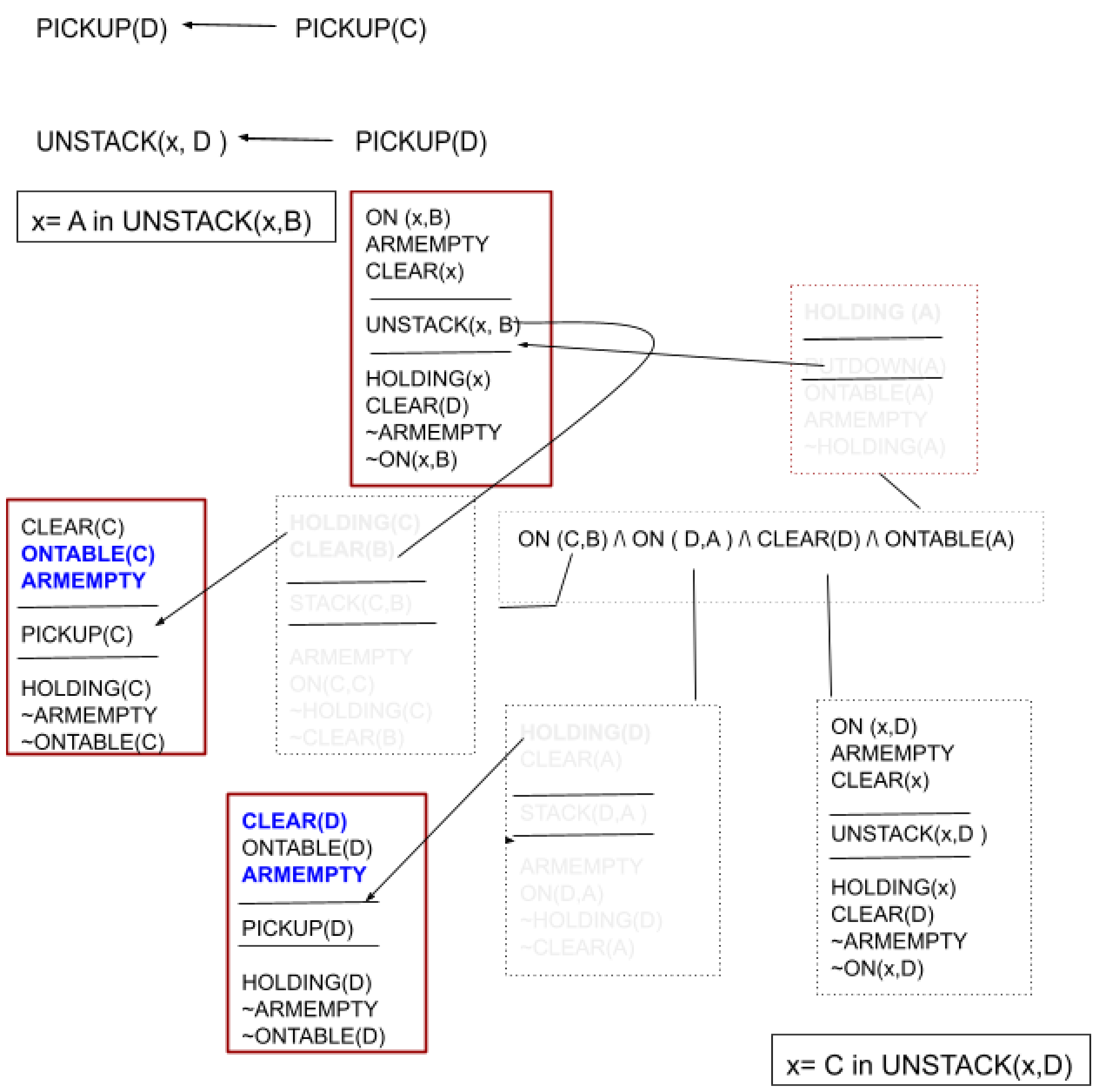


The satisfied preconditions are marked again with maroon and we also need to apply promotion as below,

$$\begin{tikzcd}
\text{UNSTACK(x,B)} & \text{PUTDOWN(A)} \arrow[l] \\
\text{PICKUP(C)} & \text{STACK(C,B)} \arrow[l] \\
\text{PICKUP(D)} & \text{STACK(D,A)} \arrow[l]
\end{tikzcd}$$

Now let's proceed on finding whether the preconditions of the actions PICKUP(C), PICKUP(D) AND UNSTACK(x, B) are satisfied.

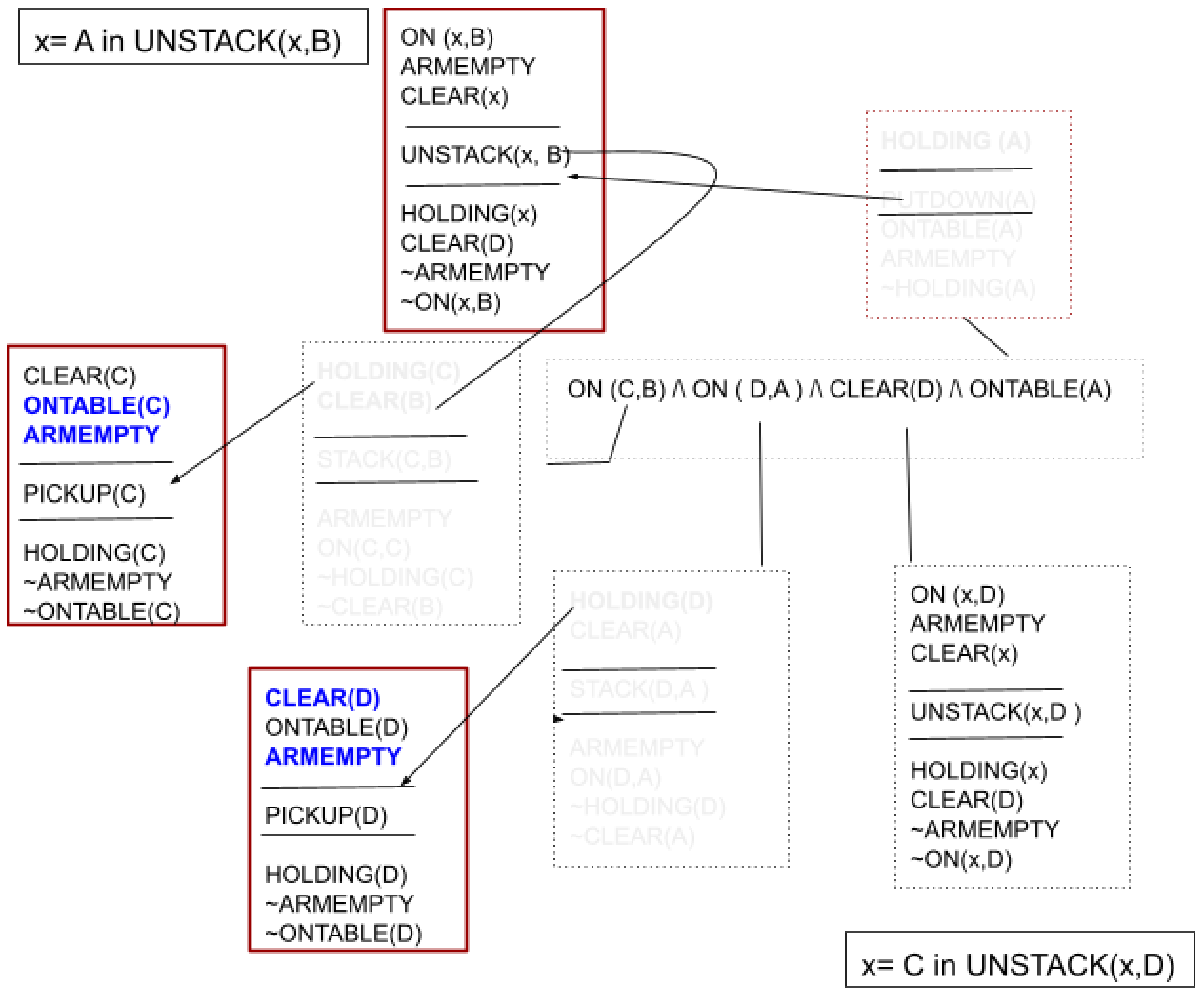
In the below diagram the ARMEMPTY precondition of PICKUP(C) and PICKUP(D) are true when comparing the initial situation but anyhow only one out of these two action can happen first and that can cause the ARMEMPTY to be false in second. Any how either of these has to happen. So let's consider PICKUP(D) happens first. Also the precondition CLEAR(D) in PICKUP(D) can be satisfied by using promotion, so introduce UNSTACK(x,D) before PICKUP(D)



While considering the UNSTACK(x,B) we can use simple establishment and assign A for x and then we can see that all the preconditions are satisfied except ARMEMPTY. So use PUTDOWN(C) before UNSTACK(x,B) and this achieves ARMEMPTY. Also use promotion and achieve PICKUP(C) or PICKUP(D) after UNSTACK(x,B) otherwise the ARMEMPTY will be made false. So apply promotion here,where UNSTACK(x,B) comes before PICKUP(C) and PICKUP(D).

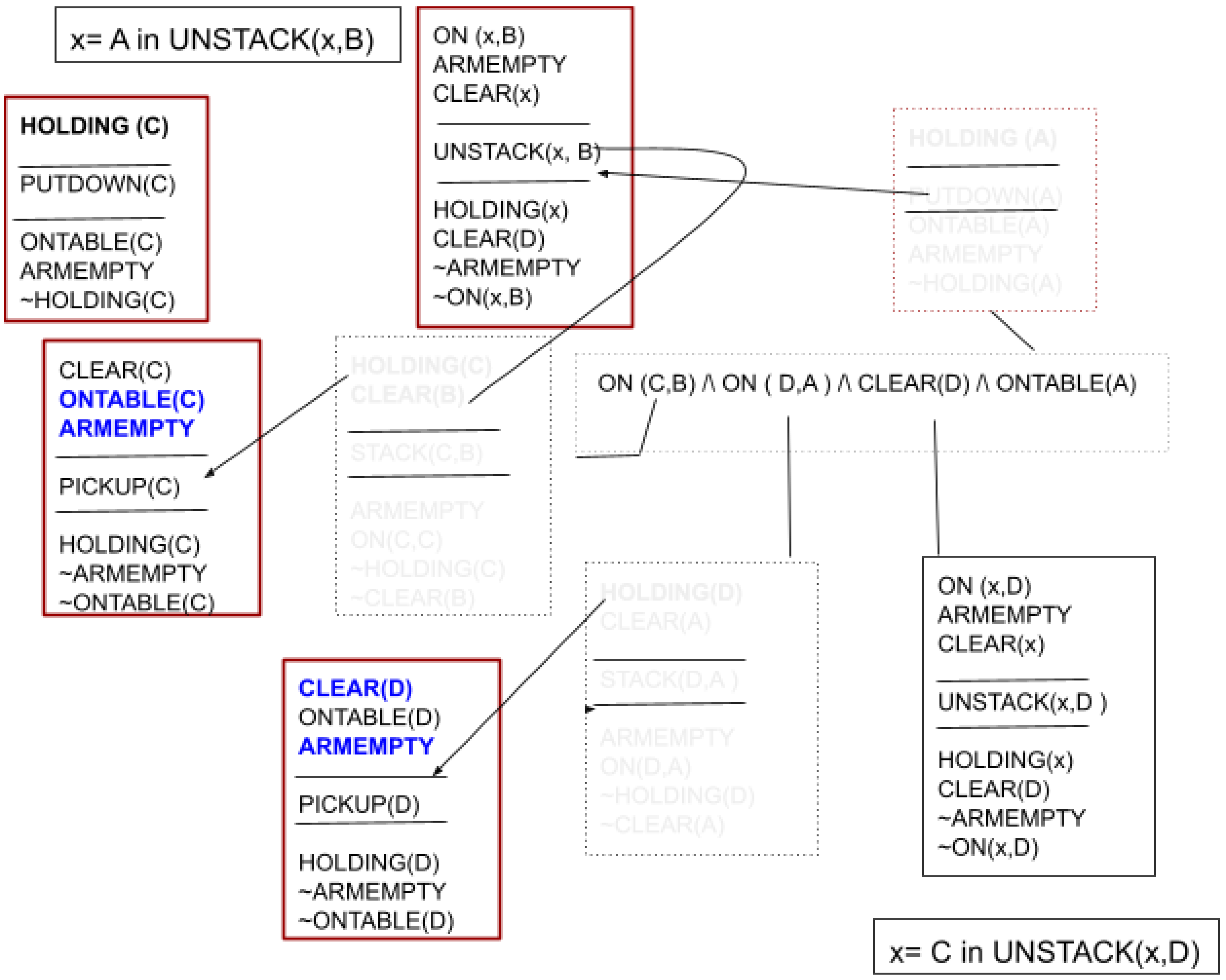
$$\begin{tikzcd}
\text{UNSTACK(x,B)} & \text{PICKUP(C)} \arrow[l] \\
\text{UNSTACK(x,B)} & \text{PICKUP(D)} \arrow[l]
\end{tikzcd}$$

$$\begin{tikzcd}
\text{PUTDOWN(C)} & \text{UNSTACK(x,B)} \arrow[l]
\end{tikzcd}$$



To achieve ONTABLE(C) in the action PICKUP(C) use step addition to include PUTDOWN(C). And the action PUTDOWN(C) comes before PICKUP(C) in the partial plan as shown below.

$$\begin{tikzcd}\text{PUTDOWN(C)} & \text{PICKUP(C)} \arrow[l]\end{tikzcd}$$



In the above diagram note that in PUTDOWN(C) the precondition HOLDING(C) is not true so use promotion to achieve it. Use UNSTACK(C,D) before PUTDOWN(C).

$$\begin{tikzcd}\text{UNSTACK(C,D)} & \text{PUTDOWN(C)} \arrow[l]\end{tikzcd}$$

Upon this we can see every actions precondition got satisfied and now lets construct the total plan from the partial plan.To do that we need consider the dependency among various action that we have seen so far.

I have furnished below all the actions with dependencies from the beginning ,with that a complete plan could be generated.

$$\begin{tikzcd}
\text{UNSTACK(x,B)} & \text{PUTDOWN(A)} \arrow[l] \\
\text{PICKUP(C)} & \text{STACK(C,B)} \arrow[l] \\
\text{PICKUP(D)} & \text{STACK(D,A)} \arrow[l]
\end{tikzcd}$$

$$\begin{tikzcd}
\text{UNSTACK(x,B)} & \text{PICKUP(C)} \arrow[l] \\
\text{UNSTACK(x,B)} & \text{PICKUP(D)} \arrow[l]
\end{tikzcd}$$

$$\begin{tikzcd}
\text{PICKUP(D)} & \text{PICKUP(C)} \arrow[l]
\end{tikzcd}$$

$$\begin{tikzcd}
\text{PUTDOWN(C)} & \text{PICKUP(C)} \arrow[l]
\end{tikzcd}$$

$$\begin{tikzcd}
\text{PUTDOWN(C)} & \text{UNSTACK(x,B)} \arrow[l]
\end{tikzcd}$$

$$\begin{tikzcd}
\text{UNSTACK(C,D)} & \text{PUTDOWN(C)} \arrow[l]
\end{tikzcd}$$

First Step:

$$\begin{tikzcd}
\text{UNSTACK(C,D)} & \text{PUTDOWN(C)} \arrow[l]
\end{tikzcd}$$

Second Step:

$$\begin{tikzcd}
\text{UNSTACK(C,D)} & \text{PUTDOWN(C)} \arrow[l] & \text{UNSTACK(A,B)} \arrow[l]
\end{tikzcd}$$

Third Step :

$$\begin{tikzcd}
\text{UNSTACK(C,D)} & \text{PUTDOWN(C)} \arrow[l] & \text{UNSTACK(A,B)} \arrow[l] \\
{} & \text{PUTDOWN(A)} \arrow[l] &
\end{tikzcd}$$

Fourth Step

$$\begin{tikzcd}
\text{UNSTACK(C,D)} & \text{PUTDOWN(C)} \arrow[l] & \text{UNSTACK(A,B)} \arrow[l] \\
{} & \text{PUTDOWN(A)} \arrow[l] &
\end{tikzcd}$$

Fifth Step :

$$\begin{tikzcd}
\text{UNSTACK(C,D)} & \text{PUTDOWN(C)} \arrow[l] & \text{UNSTACK(A,B)} \arrow[l] \\
{} & \text{PUTDOWN(A)} \arrow[l] & \text{PICKUP(D)} \arrow[l]
\end{tikzcd}$$

Sixth Step:

$$\begin{tikzcd}
\text{UNSTACK(C,D)} & \text{PUTDOWN(C)} \arrow[l] & \text{UNSTACK(A,B)} \arrow[l] & \\
{} & \text{PUTDOWN(A)} \arrow[l] & \text{PICKUP(D)} \arrow[l] & \text{STACK(D,A)} \arrow[l]
\end{tikzcd}$$

Seventh Step:

$$\begin{tikzcd}
& \text{UNSTACK(C,D)} & \text{PUTDOWN(C)} \arrow[l] & \text{UNSTACK(A,B)} \arrow[l] & \\
& {} & \text{PUTDOWN(A)} \arrow[l] & \text{PICKUP(D)} \arrow[l] & \text{STACK(D,A)} \arrow[l] \\
{} & \text{PICKUP(C)} \arrow[l] & & &
\end{tikzcd}$$

Eighth Step:

UNSTACK(C,D) ← PUTDOWN(C) ← UNSTACK(A,B)

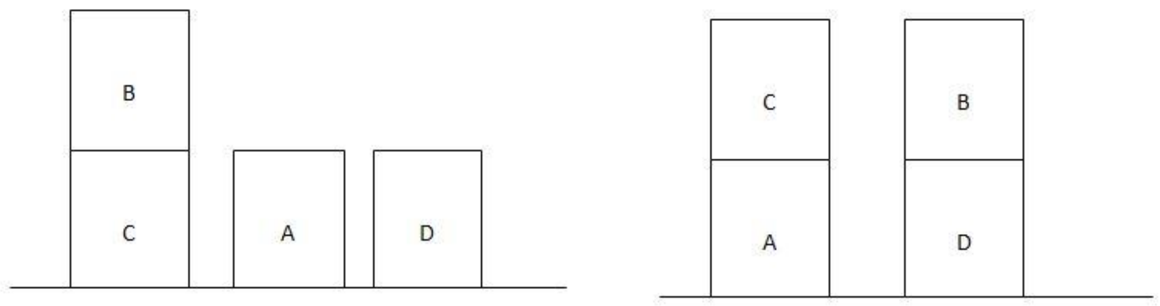
← PUTDOWN(A) ← PICKUP(D) ← STACK(D,A)

← PICKUP(C) ← STACK(C,B)

The Complete Plan

UNSTACK(C,D) ← PUTDOWN(C) ← UNSTACK(A,B) ← PUTDOWN(A) ←
PICKUP(D) ← STACK(D,A) ← PICKUP(C) ← STACK(C,B)

Goal Stack Planning



Initial State

ON(B,C) ⋀ ONTABLE(C) ⋀ ONTABLE(A) ⋀ ONTABLE(D)
CLEAR(B) ⋀ CLEAR(A) ⋀ CLEAR(D)

Goal State

ON(C,A) ⋀ ON(B,D) ⋀ ONTABLE(A) ⋀ ONTABLE(D)
⋀ CLEAR(C) ⋀ CLEAR(B)

Introduction :

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 for specifying and manipulating the world with which it is working.

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.

Algorithm:

```
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
```

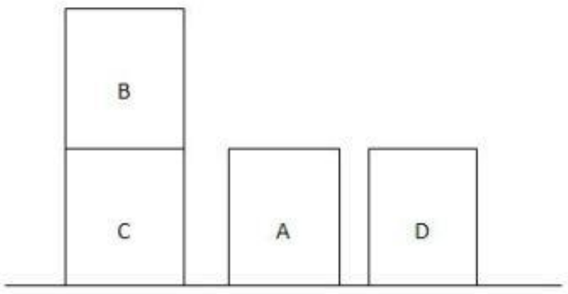
Explanation:

The Goal Stack Planning Algorithms works will 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.

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.

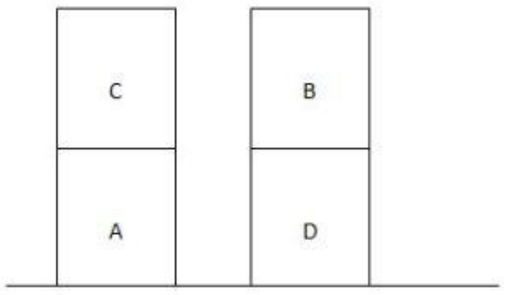
Example :

The below example is from block world domain.



Initial State

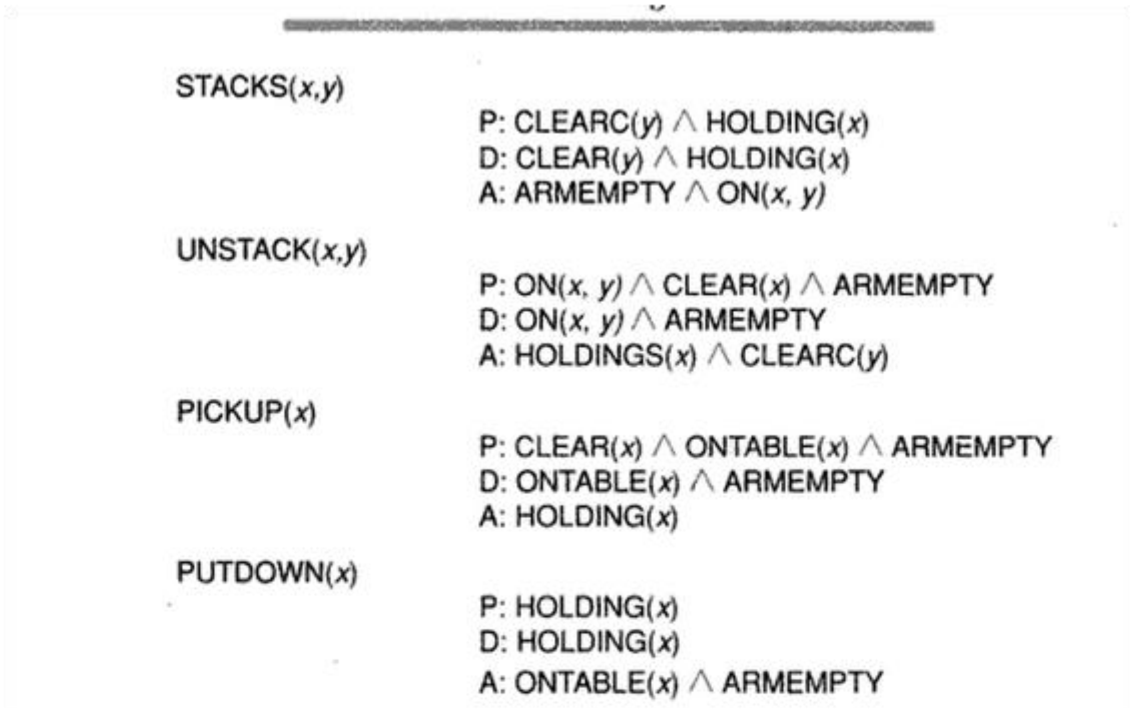
ON(B,C) ⋀ ONTABLE(C) ⋀ ONTABLE(A) ⋀ ONTABLE(D)
CLEAR(B) ⋀ CLEAR(A) ⋀ CLEAR(D)



Goal State

ON(C,A) ⋀ ON(B,D) ⋀ ONTABLE(A) ⋀ ONTABLE(D)
⋀ CLEAR(C) ⋀ CLEAR(B)

Lets start here with the example above, the initial state is our current description of our world. The Goal state is what we have to achieve. The following list of actions can be applied to the various situation in our problem.



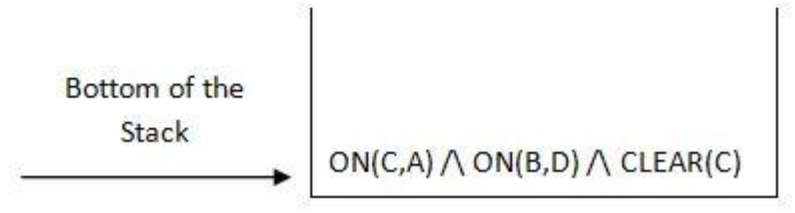
STACKS(*x*,*y*)
P: CLEARC(*y*) ∧ HOLDING(*x*)
D: CLEAR(*y*) ∧ HOLDING(*x*)
A: ARMEMPTY ∧ ON(*x*, *y*)

UNSTACK(*x*,*y*)
P: ON(*x*, *y*) ∧ CLEAR(*x*) ∧ ARMEMPTY
D: ON(*x*, *y*) ∧ ARMEMPTY
A: HOLDINGS(*x*) ∧ CLEARC(*y*)

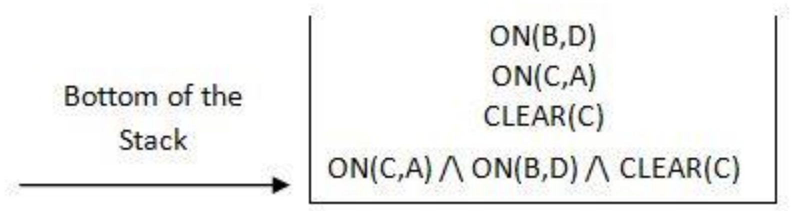
PICKUP(*x*)
P: CLEAR(*x*) ∧ ONTABLE(*x*) ∧ ARMEMPTY
D: ONTABLE(*x*) ∧ ARMEMPTY
A: HOLDING(*x*)

PUTDOWN(*x*)
P: HOLDING(*x*)
D: HOLDING(*x*)
A: ONTABLE(*x*) ∧ ARMEMPTY

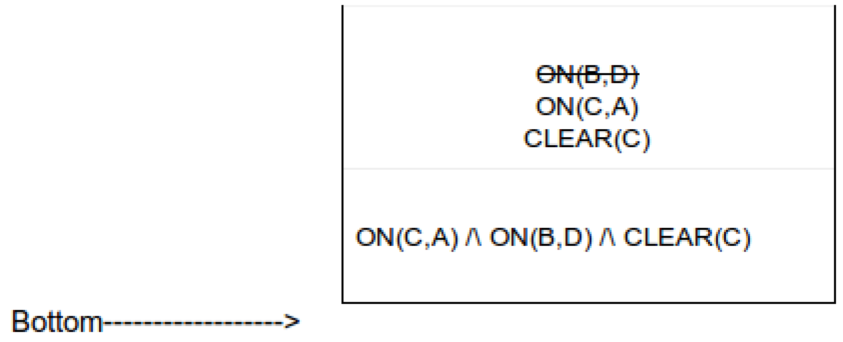
First step is to push the goal into the stack.



Next push the individual predicates of the goal into the stack.

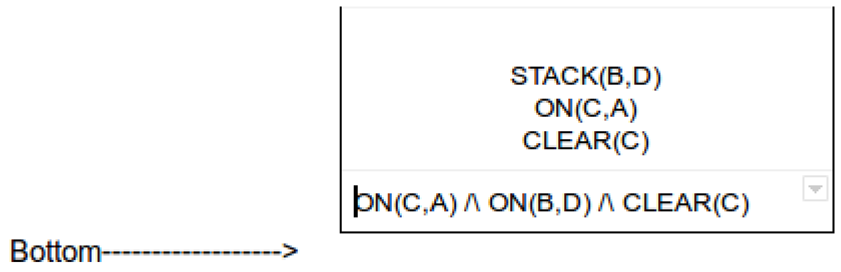


Now pop an element out from the stack

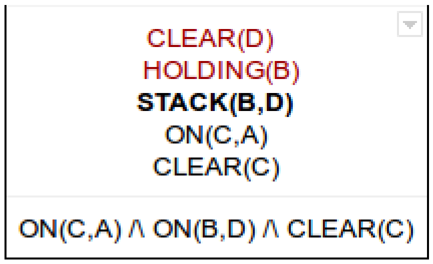


.

The popped element is indicated with a strike-through in the above diagram. The element is ON(B,D) which is a predicate and it is not true in our current world. So the next step is to push the relevant action which could achieve the subgoal ON(B,D) in to the stack.



Now again push the precondition of the action Stack(B,D) into the stack.



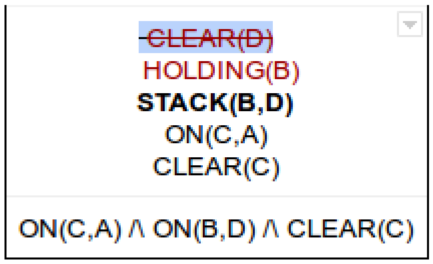
CLEAR(D)
HOLDING(B)
STACK(B,D)
ON(C,A)
CLEAR(C)
ON(C,A) Λ ON(B,D) Λ CLEAR(C)

Bottom------------------->

The preconditions are highlighted with red and the actions are marked with Bold. Now coming to the point we have to make the precondition to become true in order to apply the action. Here we need to note an interesting thing about the preconditions that, it can be pushed into the stack in any order but in certain situations we have to make some exception and that is clearly seen in the above diagram. The HOLDING(B) is pushed first and CLEAR(D) is pushed next indicating that the HOLDING subgoal has to be done second comparing with the CLEAR. Because we are considering the block world with single arm robot and everything that we usually do here is depending on the robotic arm if we first achieve HOLDING(D) then we have to undo the subgoal in order the achieve some other subgoal. So if our compound goal has HOLDING (x) subgoal, achieve it at the end.

Lets go back to our example here,

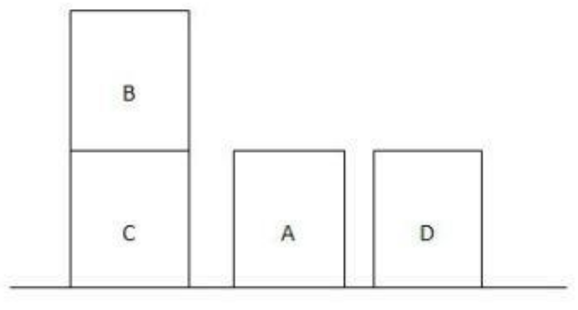
POP an element out from the stack.



~~CLEAR(D)~~
HOLDING(B)
STACK(B,D)
ON(C,A)
CLEAR(C)
ON(C,A) Λ ON(B,D) Λ CLEAR(C)

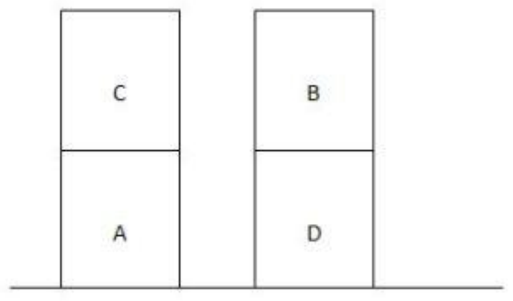
Bottom------------------->

After popping we see that CLEAR(D) is true in the current world model so we don't have to do anything.



Initial State

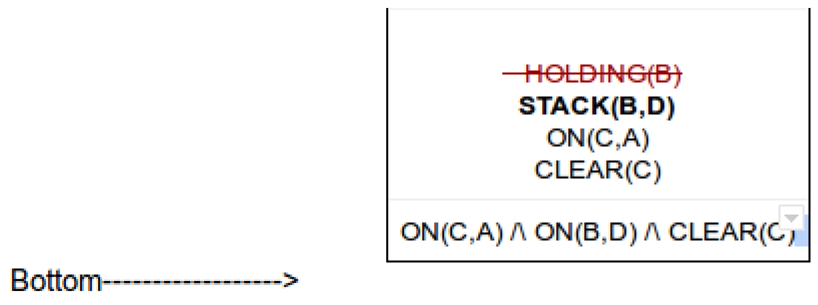
ON(B,C) ⋀ ONTABLE(C) ⋀ ONTABLE(A) ⋀ ONTABLE(D) CLEAR(B) ⋀ CLEAR(A) ⋀ CLEAR(D)



Goal State

ON(C,A) ⋀ ON(B,D) ⋀ ONTABLE(A) ⋀ ONTABLE(D) ⋀ CLEAR(C) ⋀ CLEAR(B)

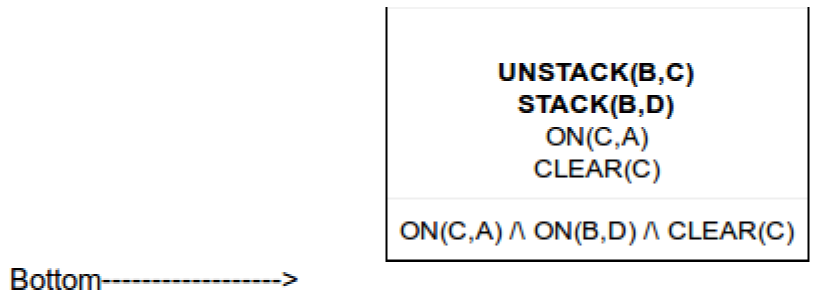
So again pop the stack,



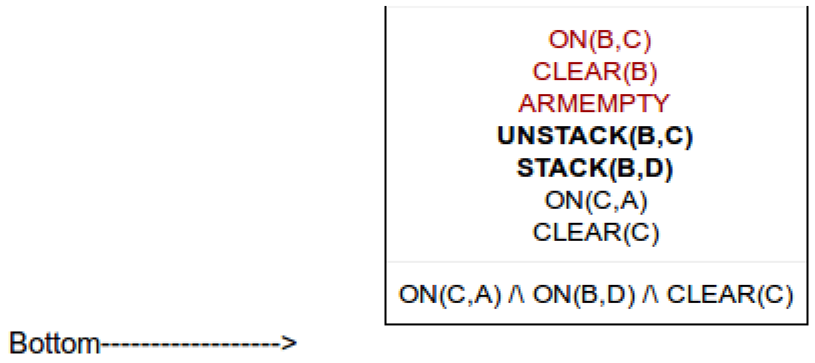
~~HOLDING(B)~~
STACK(B,D)
ON(C,A)
CLEAR(C)
ON(C,A) ∧ ON(B,D) ∧ CLEAR(C)

The popped element is HOLDING(B) which is a predicate and note that it is not true in our current world. So we have to push the relevant action into the stack. Inorder to make the HOLDING(D) to be true there are possibly two action that can achieve it.One is PICKUP(D) and the other is UNSTACK(D,y). But now in-order to choose the best among the two actions available we have to think ahead and utilize the heuristics possibly. For instance if we choose PICKUP(B) then first of all BLOCK D should be available on the table. For that we have to UNSTACK(B,D) and it will achieve HOLDING(B) which is what we want but if we use PICKUP then we need to PUTDOWN(B) making HOLDING(B) false and then use PICKUP(B) action to achieve HOLDING(B) again which can be easily achieved by using UNSTACK. So the best action is UNSTACK(B,y) and it also makes the current situation more close to the goal state. The variable y indicates any block below D.

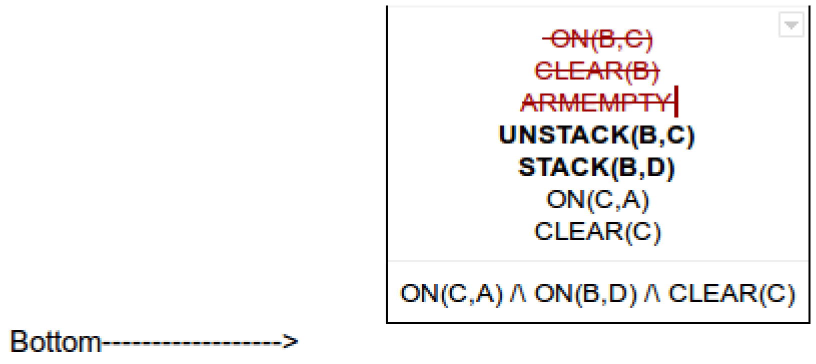
Lets push the action UNSTACK(B,C) into the stack.



Now push the individual precondition of UNSTACK(B,C) into the stack.



POP the stack. Note here that on popping we could see that ON(B,C) ,CLEAR(B) AND ARMEMPTY are true in our current world. So dont do anything.

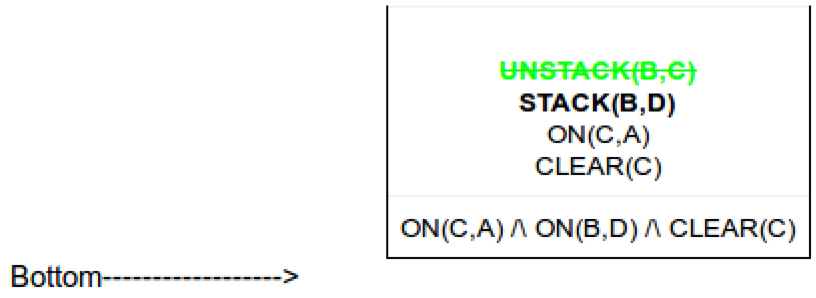


~~ON(B,C)~~
~~CLEAR(B)~~
~~ARMEMPTY~~
UNSTACK(B,C)
STACK(B,D)
ON(C,A)
CLEAR(C)

ON(C,A) /\ ON(B,D) /\ CLEAR(C)

Bottom------------------->

Now again pop the stack .



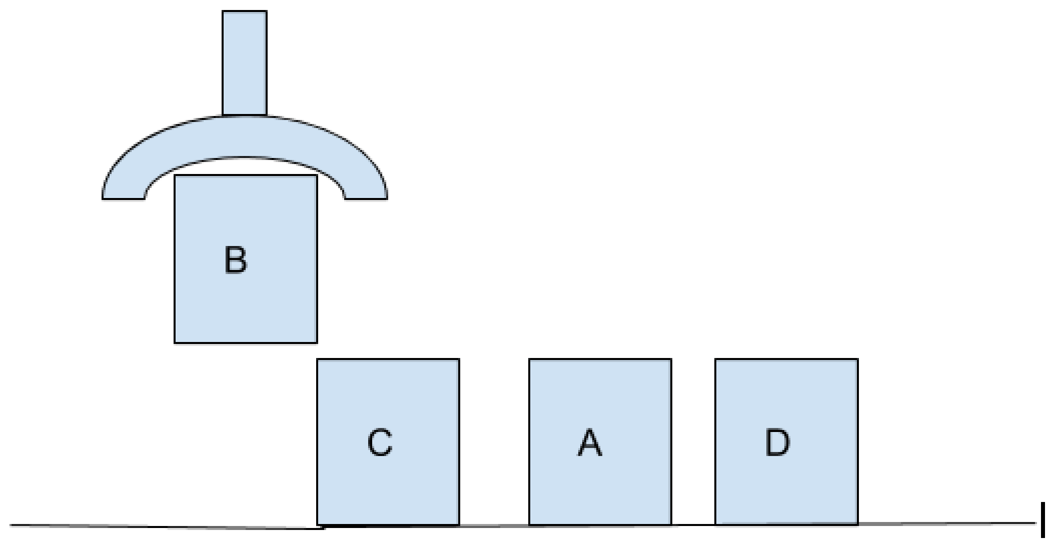
~~**UNSTACK(B,C)**~~
STACK(B,D)
ON(C,A)
CLEAR(C)

ON(C,A) /\ ON(B,D) /\ CLEAR(C)

Bottom------------------->

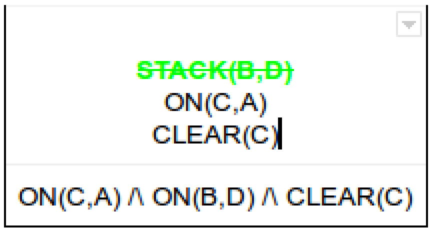
When we do that we will get an action, so just apply the action to the current world and add that action to plan list.

Plan= { UNSTACK(B,C) }

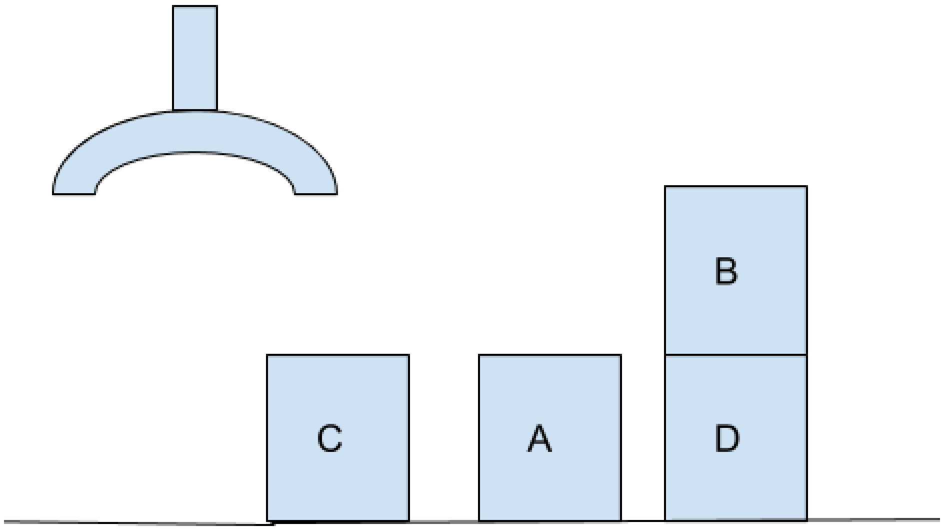


Again pop an element. Now its STACK(B,D) which is an action so apply that to the current state and add it to the PLAN.

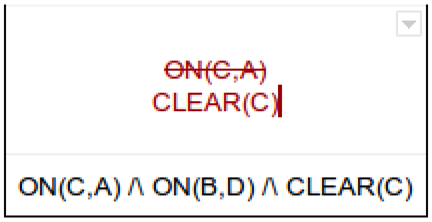
PLAN= { UNSTACK(B,C), STACK(B,D) }


~~STACK(B,D)~~
ON(C,A)
CLEAR(C)
ON(C,A) Λ ON(B,D) Λ CLEAR(C)

Bottom------------------->

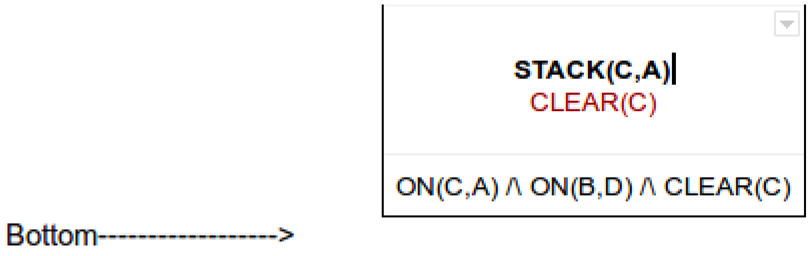


Now the stack will look like the one given below and our current world is like the one above.


~~ON(C,A)~~
CLEAR(C)
ON(C,A) Λ ON(B,D) Λ CLEAR(C)

Bottom------------------->

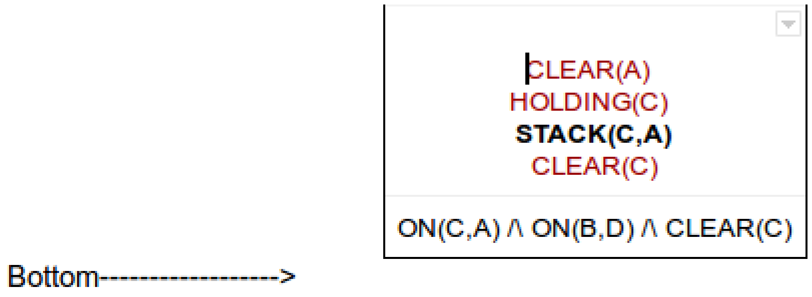
Again pop the stack. The popped element is a predicate and it is not true in our current world so push the relevant action into the stack.



STACK(C,A)
CLEAR(C)
ON(C,A) Λ ON(B,D) Λ CLEAR(C)

Bottom------------------->

STACK(C,A) is pushed now into the stack and now push the individual preconditions of the action into the stack.

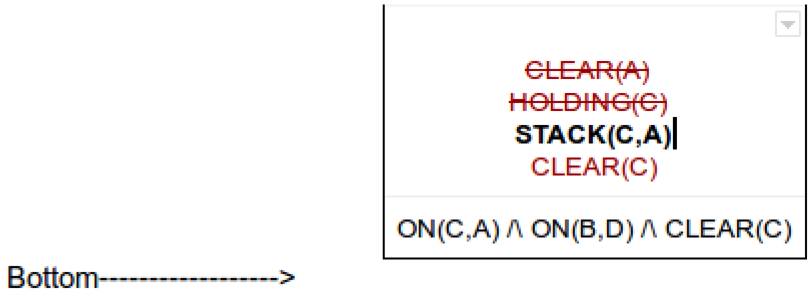


CLEAR(A)
HOLDING(C)
STACK(C,A)
CLEAR(C)
ON(C,A) Λ ON(B,D) Λ CLEAR(C)

Bottom------------------->

Now pop the stack. We will get CLEAR(A) and it is true in our current world so do nothing.

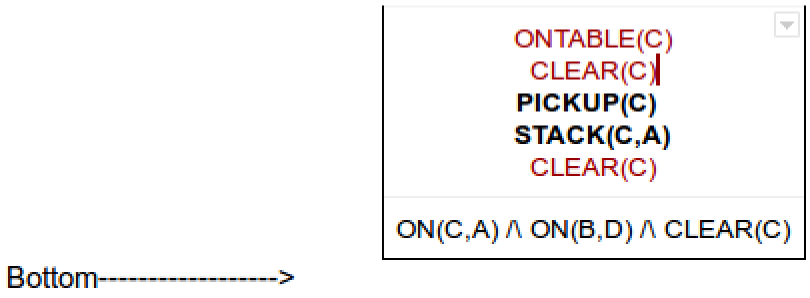
Next element that is popped is HOLDING(C) which is not true so push the relevant action into the stack.



~~CLEAR(A)~~
~~HOLDING(C)~~
STACK(C,A)
CLEAR(C)
ON(C,A) Λ ON(B,D) Λ CLEAR(C)

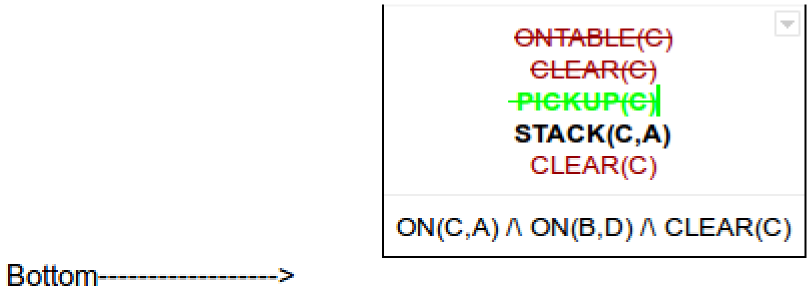
Bottom------------------->

In order to achieve HOLDING(C) we have to push the action PICKUP(C) and its individual preconditions into the stack.

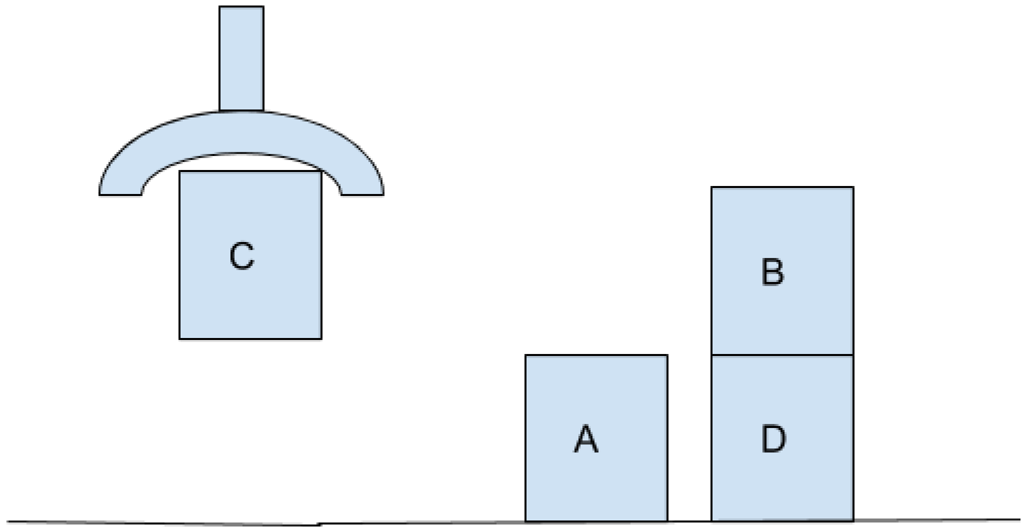

ONTABLE(C)
CLEAR(C)
PICKUP(C)
STACK(C,A)
CLEAR(C)
ON(C,A) ∧ ON(B,D) ∧ CLEAR(C)

Bottom------------------->

Now doing pop we will get ONTABLE(C) which is true in our current world.Next CLEAR(C) is popped and that also is achieved.Then PICKUP(C) is popped which is an action so apply it to the current world and add it to the PLAN. The world model and stack will look like below,

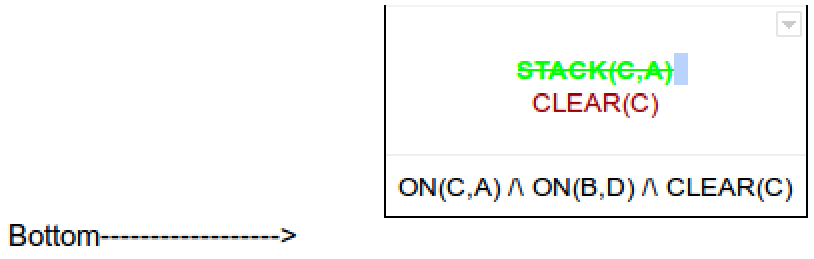

~~ONTABLE(C)~~
~~CLEAR(C)~~
~~**PICKUP(C)**~~
STACK(C,A)
CLEAR(C)
ON(C,A) ∧ ON(B,D) ∧ CLEAR(C)

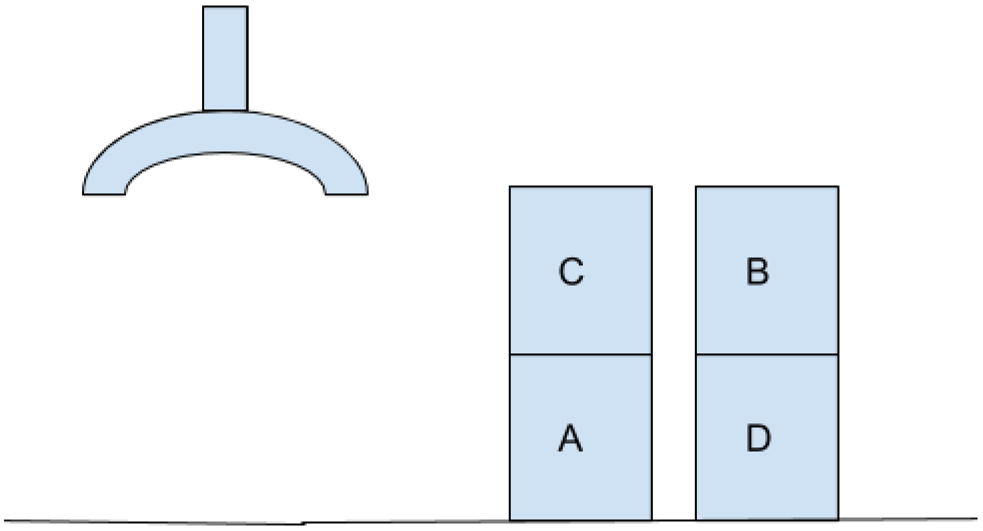
Bottom------------------->



PLAN= { UNSTACK(B,C), STACK(B,D) ,PICKUP(C) }

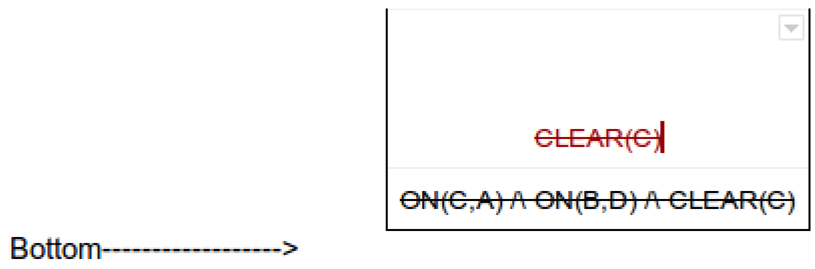
Again POP the stack, we will get STACK(C,A) which is an action apply it to the world and insert it to the PLAN.





PLAN= { UNSTACK(B,D), STACK(B,D) ,PICKUP(C) ,STACK(C,A) }

Now pop the stack we will get CLEAR(C) which is already achieved in our current situation. So we don't need to do anything. At last when we pop the element we will get all the three subgoal which is true and our PLAN will contain all the necessary actions to achieve the goal.



PLAN= { UNSTACK(B,D), STACK(B,D) ,PICKUP(C) ,STACK(C,A) }

Q.7 Define the Bayes theorem and apply the Bayes'rule for examining the simple case and combining evidence with suitable examples.

SOLUTION:

Bayes' rule

Bayes' theorem (alternatively Bayes' law or Bayes' rule) has been called the most powerful rule of probability and statistics. It describes the probability of an event, based on prior knowledge of conditions that might be related to the event.

We gather data and update our initial beliefs. If the data support the hypothesis then the probability goes up, if it does not match, then probability goes down.

Bayesian inference

Bayesian statistics and modeling have had a recent resurgence with the global rise of AI and data-driven machine learning systems in all aspects of business, science, and technology.

Bayesian inference is being applied to genetics, linguistics, image processing, brain imaging, cosmology, machine learning, epidemiology, psychology, forensic science, human object recognition, evolution, visual perception, ecology, and countless other fields where knowledge discovery and predictive analytics are playing a significant role.

EXAMPLE: Drug screening

We will apply the Bayes' rule to a problem of drug screening (e.g. mandatory testing for federal or many other jobs which promise a drug-free work environment).

Suppose that a test for using a particular drug is 97% sensitive and 95% specific. That is, the test will produce 97% true positive results for drug users and 95% true negative results for non-drug users. These are the pieces of data that any screening test will have from their history of tests. Bayes' rule allows us to use this kind of data-driven knowledge to calculate the final probability.

Suppose, we also know that 0.5% of the general population are users of the drug. What is the probability that a randomly selected individual with a positive test is a drug user?

Note, this is the crucial piece of 'Prior' which is a piece of generalized knowledge about the common prevalence rate. This is our prior belief about the probability of a random test subject being a drug user. That means if we choose a random person from the general population, without any testing, we can only say that there is a 0.5% chance of that person being a drug-user.

How to use Bayes' rule then, in this situation?

We will write a custom function that accepts the test capabilities and the prior knowledge of drug user percentage as input and produces the output probability of a test-taker being a user based on a positive result.

PYTHON CODE IMPLEMENTATION:

```
def
drug_user(

                    prob_th=0.5,
    sensitivity=0.99,
    specificity=0.99,
    prevelance=0.01,
    verbose=True):
"""
Computes the posterior using Bayes' rule
"""
p_user = prevelance
p_non_user = 1-prevelance
p_pos_user = sensitivity
p_neg_user = specificity
p_pos_non_user = 1-specificity

num = p_pos_user*p_user
den = p_pos_user*p_user+p_pos_non_user*p_non_user

prob = num/den

if verbose:
    if prob > prob_th:
        print("The test-taker could be an user")
    else:
        print("The test-taker may not be an user")
        return prob
```

If we run the function with the given data, we get the following result,

```
p = drug_user(prob_th=0.5,sensitivity=0.97,specificity=0.95,prevelance=0.005)
print("Probability of the test-taker being a drug user is:",round(p,3))
```

```
The test-taker may not be an user
Probability of the test-taker being a drug user is: 0.089
```

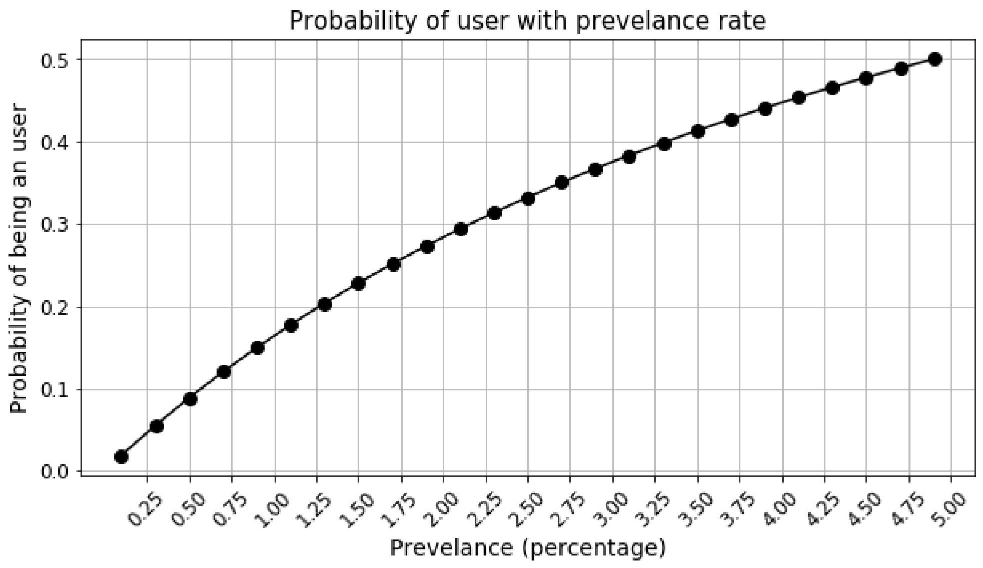
What is fascinating here?

Even with a test that is 97% correct for catching positive cases, and 95% correct for rejecting negative cases, the true probability of being a drug-user with a positive result is only 8.9%!

If you look at the computations, this is because of the extremely low prevalence rate. The number of false positives outweighs the number of true positives.

For example, if 1000 individuals are tested, there are expected to be 995 non-users and 5 users. From the 995 non-users, $0.05 \times 995 \simeq 50$ false positives are expected. From the 5 users, $0.95 \times 5 \approx 5$ true positives are expected. Out of 55 positive results, only 5 are genuine!

Let's see how the probability changes with the prevalence rate:



Q.8 How the hierarchical planning method is used for solving complex operator hard problems with the help of suitable example?

SOLUTION:

Hierarchies are the most common structure used to understand the world better. In galaxies, for instance, multiple-star systems are organised in a hierarchical system. Then, governmental and company organisations are structured using a hierarchy, while the Internet, which is used on a daily basis, has a space of domain names arranged hierarchically. Since Artificial Intelligence (AI) planning portrays information about the world and reasons to solve some of world's problems, Hierarchical Task Network (HTN) planning has been introduced almost 40 years ago to represent and deal with hierarchies. Its requirement for rich domain knowledge to characterise the world enables HTN planning to be very useful, but also to perform well. However, the history of almost 40 years obfuscates the current understanding of HTN planning in terms of accomplishments, planning models, similarities and differences among hierarchical planners, and its current and objective image. On top of these issues, attention attracts the ability of hierarchical planning to truly cope with the requirements of applications from the real world. We propose a framework-based approach to remedy this situation. First, we provide a basis for defining different formal models of hierarchical planning, and define two models that comprise a large portion of HTN planners. Second, we provide a set of concepts that helps to interpret HTN planners from the aspect of their search space. Then, we analyse and compare the planners based on a variety of properties organised in five segments, namely domain authoring, expressiveness, competence, performance and applicability. Furthermore, we select Web service composition as a real-world and current application, and classify and compare the approaches that employ HTN planning to solve the problem of service composition. Finally, we conclude with our findings and present directions for future work.

EXAMPLE:

- 180 overs : 15 spells (12 overs each)
- 5 bowlers : 3 categories (2 pacer/2 spinner/1 pacer&1 spinner)
- Top level possibilities : 3^{15}
- Total possibilities < $3*3^{15}$ (much less than 5^{180})

SOLUTION:

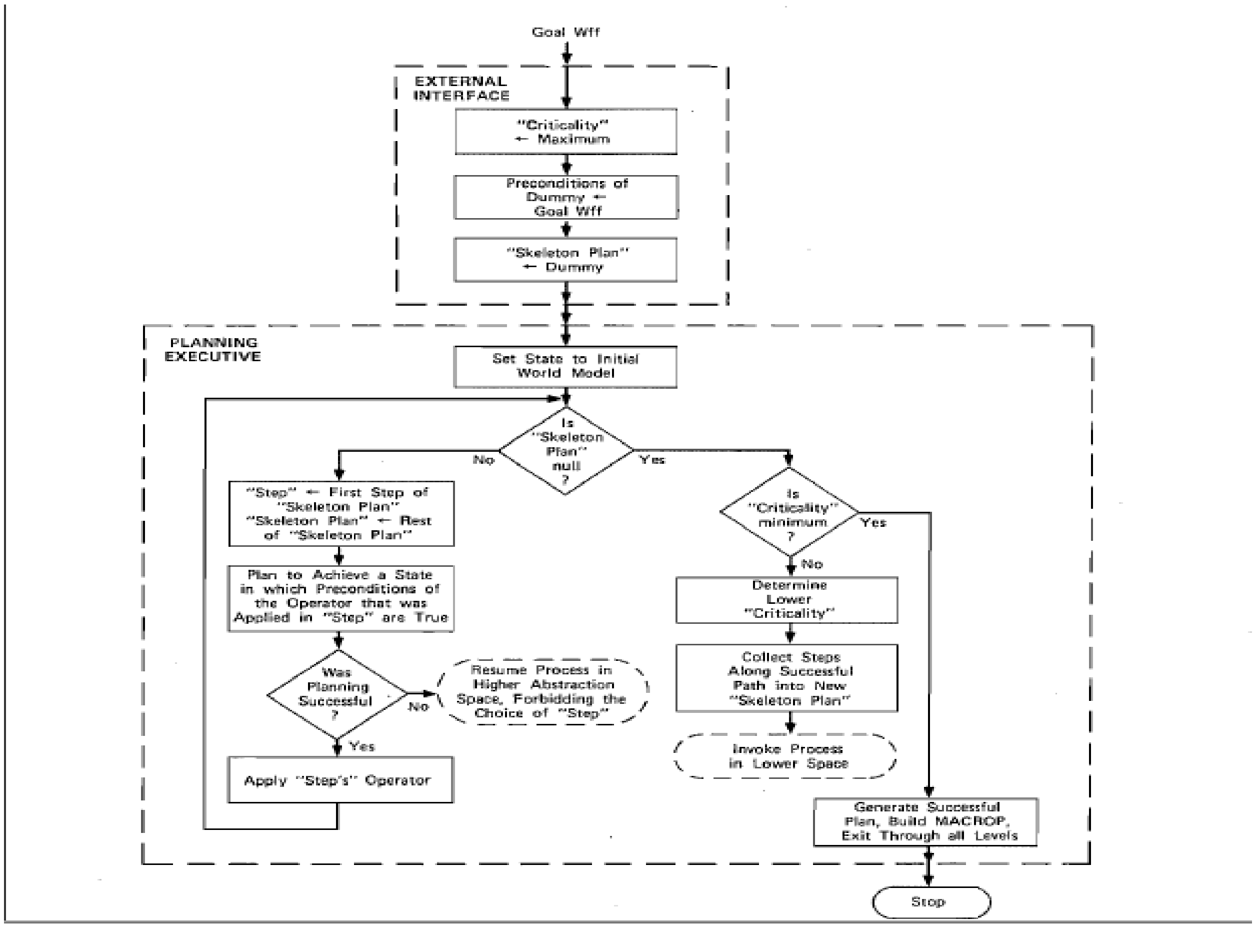
- If entire plan has to be synthesized at the level of most detailed actions, it would be *impossibly long.*
- Natural to 'intelligent' agent
- *Postpone* attempts to solve mere details, *until* major steps are in place.
- Higher level plan may run into difficulties at a lower level, causing the need to return to higher level again to produce appropriately ordered sequence.
- Identify a hierarchy of conditions
- Construct a plan in levels, postponing details to the next level

- Patch higher levels as details become visible

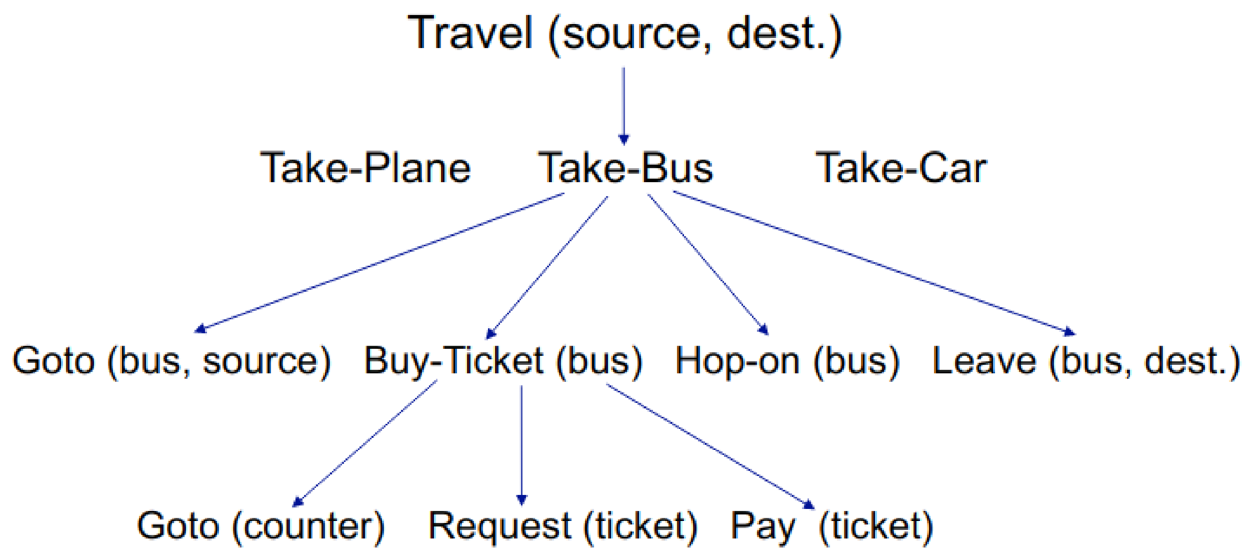
Demonstrated using ABSTRIPS

- Abstraction-Based STRIPS
- Modified version of STRIPS that incorporates hierarchical planning
- Hierarchy of conditions reflect the intrinsic difficulty of achieving various conditions.
- Indicated by criticality value.
- A operation having *minimum criticality* can be trivially achievable, i.e., the operations having very less or no precondition.
    - Example : Opening makemytrip.com

Similarly operation having many preconditions to satisfy will have higher criticality


Goal Wff
EXTERNAL INTERFACE
"Criticality" ← Maximum
Preconditions of Dummy ← Goal Wff
"Skeleton Plan" ← Dummy
PLANNING EXECUTIVE
Set State to Initial World Model
Is "Skeleton Plan" null ?
No
Yes
"Step" ← First Step of "Skeleton Plan" "Skeleton Plan" ← Rest of "Skeleton Plan"
Plan to Achieve a State in which Preconditions of the Operator that was Applied in "Step" are True
Was Planning Successful ?
No
Resume Process in Higher Abstraction Space, Forbidding the Choice of "Step"
Yes
Apply "Step's" Operator
Is "Criticality" minimum ?
Yes
No
Determine Lower "Criticality"
Collect Steps Along Successful Path into New "Skeleton Plan"
Invoke Process in Lower Space
Generate Successful Plan, Build MACROP, Exit Through all Levels
Stop

Example :



- Actions required for “Travelling to Goa”:
    - Opening makemytrip.com (1)
    - Finding flight (2)
    - Buy Ticket (3)
    - Get taxi(2)
    - Reach airport(3)
    - Pay-driver(1)
    - Check in(1)
    - Boarding plane(2)
    - Reach Goa(3)
- 1st level Plan :
    - Buy Ticket (3), Reach airport(3), Reach Goa(3)
- 2nd level Plan :
    - Finding flight (2), Buy Ticket (3), Get taxi(2), Reach airport(3), Boarding plane(2), Reach Goa(3)
- 3rd level Plan (final) :

- Opening makemytrip.com (1), Finding flight (2), Buy Ticket (3), Get taxi(2), Reach airport(3), Pay-driver(1), Check in(1), Boarding plane(2), Reach Goa(3)