UNIT-III UNIFICATION AND STATE SPACE SEARCH

UNIFICATION

Unification is a process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process.

It takes two literals as input and makes them identical using substitution.

Key Concepts of Unification

1. **Logical Terms**:
    - A term can be:
        - A **constant** (e.g., apple, 5).
        - A **variable** (e.g., X, Y).
        - A **function** (e.g., f(a), where f is a function, and a is an argument).
2. **Substitution**:
    - A mapping of variables to terms.
    - Example: If X = apple, substituting X with apple unifies expressions involving X.
3. **Unifier**:
    - A substitution that makes two terms or expressions identical.
    - Example: To unify f(X) and f(apple), the unifier is X = apple.
4. **Most General Unifier (MGU)**:
    - The simplest unifier that can unify two terms, leaving no unnecessary constraints.
    - Example: Unifying f(X, Y) and f(a, b) gives X = a and Y = b.

Let Ψ1 and Ψ2 be two atomic sentences and ? be a unifier such that, Ψ1? = Ψ2?, then it can be expressed as UNIFY(Ψ1, Ψ2).

Example: Find the MGU for Unify{King(x), King(John)}

Let Ψ1 = King(x), Ψ2 = King(John),

Substitution θ = {John/x} is a unifier for these atoms and applying this substitution, and both expressions will be identical.

The UNIFY algorithm is used for unification, which takes two atomic sentences and returns a unifier for those sentences (If any exist).

Unification is a key component of all first-order inference algorithms.

It returns fail if the expressions do not match with each other.

The substitution variables are called Most General Unifier or MGU.

CONDITIONS FOR UNIFICATION

- Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.
- Number of Arguments in both expressions must be identical.
- Unification will fail if there are two similar variables present in the same expression.

UNIFICATION ALGORITHM

The unification algorithm attempts to find the MGU for two terms. The algorithm involves recursively applying substitutions until the terms become identical or a conflict is found that prevents unification.

Steps in the Unification Algorithm:

1. Initialization: Start with the two terms you want to unify.
2. Decompose Compound Terms: If the terms are compound (i.e., functions with arguments), break them down into their constituent parts.

3. Check for Conflicts: If a variable is being unified with a term that contains that variable (occurs check), unification fails.
4. Apply Substitutions: Continuously apply the substitutions and simplify the terms until they are identical or no further simplification is possible.

Example 1:
Suppose we have the following expressions:
Expression 1: f(a, X, g(Y))
Expression 2: f(Z, b, g(h))

To unify these expressions, we need to find a substitution that makes them equal. The unification process involves matching corresponding parts and finding a set of variable assignments that satisfy both expressions:

Match f(a, X, g(Y)) with f(Z, b, g(h)):

X unifies with b (X/b)
Y unifies with h (Y/h)
Z unifies with a (Z/a)

The resulting substitution is: {X/b, Y/h, Z/a}. Applying this substitution to both expressions gives us:

f(a, b, g(h)) = f(a, b, g(h))
The expressions are now unified.

Example 2:
Let's consider a more complex example involving predicates and quantifiers in first-order logic:
Expression 1: ∀x P(x, f(Y))
Expression 2: P(Z, f(a))

To unify these expressions, we need to find a substitution that makes them equal. The unification process involves matching the quantifiers and predicates and finding a set of variable assignments:

Match ∀x P(x, f(Y)) with P(Z, f(a)):
x unifies with Z (x/Z)
Y unifies with a (Y/a)

The resulting substitution is: {x/Z, Y/a}. Applying this substitution to both expressions gives us:

∀Z P(Z, f(a)) = P(Z, f(a))
The expressions are now unified.

CHALLENGES IN UNIFICATION

1. **Occurs Check**:
    - Prevents cyclic substitutions. For example, unifying X with f(X) is invalid.
2. **Complexity**:
    - In cases with nested terms, finding the MGU can be computationally expensive.
3. **Handling Inconsistencies**:
    - If no unifier exists (e.g., trying to unify f(a) with g(a)), systems must detect and handle the failure.

PLANNING: DESIGNING PROGRAMS TO SEARCH FOR DATA OR SOLUTIONS TO PROBLEM

Artificial intelligence is an important technology in the future. Whether it is intelligent robots, self-driving cars, or smart cities, they will all use different aspects of artificial intelligence!!! But Planning is very important to make any such AI project.

Even Planning is an important part of Artificial Intelligence which deals with the tasks and domains of a particular problem. Planning is considered the logical side of acting.

Everything we humans do is with a definite goal in mind, and all our actions are oriented towards achieving our goal. Similarly, Planning is also done for Artificial Intelligence.

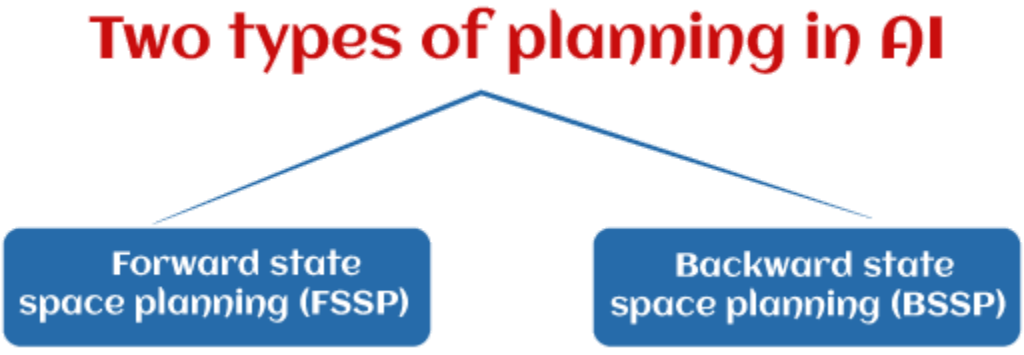
For example, Planning is required to reach a particular destination. It is necessary to find the best route in Planning, but the tasks to be done at a particular time and why they are done are also very important.

That is why Planning is considered the logical side of acting. In other words, Planning is about deciding the tasks to be performed by the artificial intelligence system and the system's functioning under domain-independent conditions.

What is a Plan?

We require domain description, task specification, and goal description for any planning system. A plan is considered a sequence of actions, and each action has its preconditions that must be satisfied before it can act and some effects that can be positive or negative.

So, we have **Forward State Space Planning (FSSP)** and **Backward State Space Planning (BSSP)** at the basic level.



1. Forward State Space Planning (FSSP)

FSSP behaves in the same way as forwarding state-space search. It says that given an initial state S in any domain, we perform some necessary actions and obtain a new state S' (which also

contains some new terms), called a progression. It continues until we reach the target position. Action should be taken in this matter.

- **Disadvantage**: Large branching factor
- **Advantage**: The algorithm is Sound

---

2. Backward State Space Planning (BSSP)

BSSP behaves similarly to backward state-space search. In this, we move from the target state g to the sub-goal g, tracing the previous action to achieve that goal. This process is called regression (going back to the previous goal or sub-goal). These sub-goals should also be checked for consistency. The action should be relevant in this case.

- **Disadvantages**: not sound algorithm (sometimes inconsistency can be found)
- **Advantage**: Small branching factor (much smaller than FSSP)

So for an efficient planning system, we need to combine the features of FSSP and BSSP, which gives rise to target stack planning which will be discussed in the next article.

What is planning in AI?

Planning in artificial intelligence is about decision-making actions performed by robots or computer programs to achieve a specific goal.

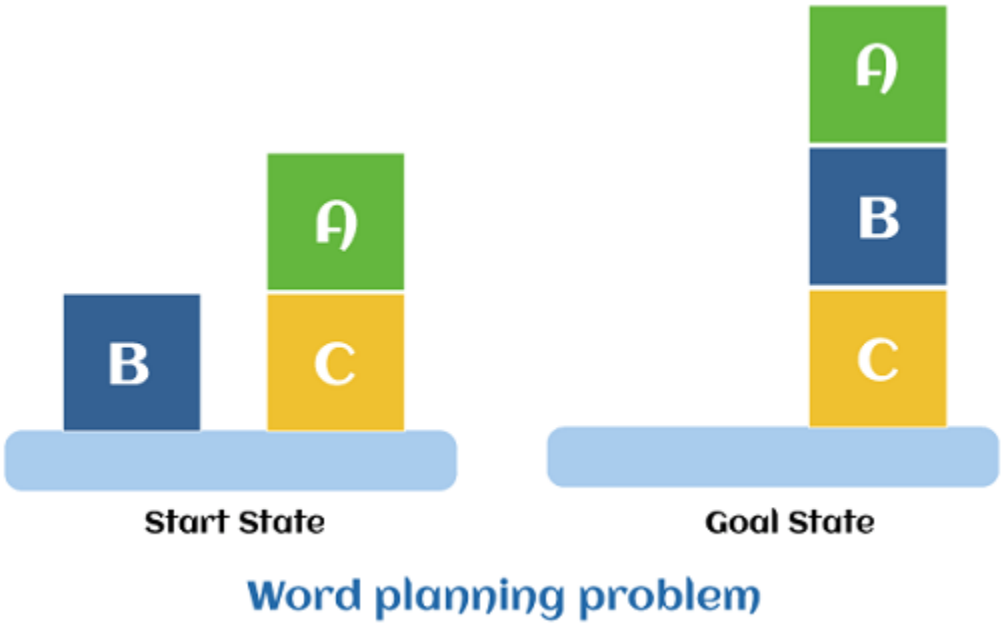
Execution of the plan is about choosing a sequence of tasks with a high probability of accomplishing a specific task.

Block-world planning problem

- The block-world problem is known as the Sussmann anomaly.
- The non-interlaced planners of the early 1970s were unable to solve this problem. Therefore it is considered odd.
- When two sub-goals, G1 and G2, are given, a non-interleaved planner either produces a plan for G1 that is combined with a plan for **G2** or vice versa.

- In the block-world problem, three blocks labeled 'A', 'B', and 'C' are allowed to rest on a flat surface. The given condition is that only one block can be moved at a time to achieve the target.

The start position and target position are shown in the following diagram.



Components of the planning system

The plan includes the following important steps:

- Choose the best rule to apply the next rule based on the best available guess.
- Apply the chosen rule to calculate the new problem condition.
- Find out when a solution has been found.
- Detect dead ends so they can be discarded and direct system effort in more useful directions.
- Find out when a near-perfect solution is found.

STATE SPACE SEARCH

Building a system to solve a problem requires the following steps.

- Define the problem precisely including detailed specifications and what constitutes an acceptable solution.
- Analyze the problem thoroughly for some features may have a dominant affect on the chosen method of solution.
- Isolate and represent the background knowledge needed in the solution of the problem.
- Choose the best problem solving techniques in the solution.

Defining the Problem as state Search

- Problems dealt with in artificial intelligence generally use a common term called 'state'.
- A state represents a status of the solution at a given step of the problem solving procedure. The solution of a problem, thus, is a collection of the problem states.
- The problem solving procedure applies an operator to a state to get the next state.
- Then it applies another operator to the resulting state to derive a new state. The process of applying an operator to a state and its subsequent transition to the next state, thus, is continued until the goal (desired) state is derived.
- Such a method of solving a problem is generally referred to as state space approach For example, in order to solve the problem play a game, which is restricted to two person table or board games, we require the rules of the game and the targets for winning as well as a means of representing positions in the game.
- The opening position can be defined as the initial state and a winning position as a goal state, there can be more than one. legal moves allow for transfer from initial state to other states leading to the goal state.

- The storage also presents another problem but searching can be achieved by hashing. The number of rules that are used must be minimized and the set can be produced by expressing each rule in as general a form as possible.
- The representation of games in this way leads to a state space representation and it is natural for well organised games with some structure.
- This representation allows for the formal definition of a problem which necessitates the movement from a set of initial positions to one of a set of target positions.
- It means that the solution involves using known techniques and a systematic search.
- This is quite a common method in AI.

Formal description of a problem

- Define a state space that contains all possible configurations of the relevant objects, without enumerating all the states in it. A state space represents a problem in terms of states and operators that change state.
- Define some of these states as possible initial states;
- Specify one or more as acceptable solutions, these are goal states.
- Specify a set of rules as the possible actions allowed. This involves thinking about the generality of the rules, the assumptions made in the informal presentation and how much work can be anticipated by inclusion in the rules.

WATER JUG PROBLEM

We are given two jugs, a four-gallon one and a three-gallon one. Neither has any measuring markers on it. There is a pump which can be used to fill the jugs with water. How can we get exactly two gallons of water into the four-gallon jug?

Start State : (0,0)

Goal State : (2,n) for any value of n.

1. (x,y)→(4,y) // Fill the 4 gallon jug

if $x<4$

2. (x,y)→(x,3) // Fill the 3 gallon jug

if $y<3$

3. (x,y)→(x-d,y) // Pour some water out of the 4-gallon jug

if $x>0$

4. (x,y)→(x,y-d) // Pour some water out of the 3-gallon jug

if $x>0$

5. (x,y)→(0,y) // Empty the 4-gallon jug on the ground

if $x>0$

6. (x,y)→(x,0) // Empty the 3-gallon jug on the ground

if $y>0$

7. (x,y)→(4,y-(4-x)) // Pour water from the 3-gallon jug into the 4 gallons until the 4-gallon jug is full

if $x+y\geq4$ and $y>0$

8. (x,y)→(x-(3-y),3) // Pour water from the 4-gallon jug into the 3 gallons until the 3-gallon jug is full

if $x+y\geq3$ and $x>0$

9. (x,y)→(x+y,0) // Pour water all the water from the 3-gallon jug into the 4-gallon jug

if $x+y\leq4$ and $y>0$

10. (x,y)→(0,x+y) // Pour water all the water from the 4-gallon jug into the 3-gallon jug

if $x+y\leq3$ and $x>0$

The water jug problem – Solution

Gallons in the 4-Gallon jug	Gallons in the 3-Gallon jug	Rule Applied
0	0	2
0	3	9
3	0	2
3	3	7
4	2	5
0	2	9
2	0	-

STATE REPRESENTATION

In order to express the problem using state space, the following elements must be defined:

- **States:** Different arrangements of the issue, frequently shown as graph nodes.
- **Initial State:** The initial setting that the search starts with.
- **Goal State(s):** The ideal configuration(s) denoting a resolution.
- **Actions:** The processes via which a system changes states.
- **Transition Model:** Explains what happens when states are subjected to actions.
- **Path cost:** The expense of moving from an initial state to a certain state, expressed as a numerical value linked to each path.

EXAMPLE - 8 PUZZLE PROBLEM

The state of the 8-puzzle is represented using a 3x3 grid, where each cell can hold one of the numbered tiles or remain empty (occupied by the blank tile). This grid serves as a compact and systematic way to capture the configuration of the puzzle.

In a 3x3 grid, each cell can contain one of the following elements:

Numbered tiles, typically from 1 to 8.

A blank tile, represented as an empty cell.

The arrangement of these elements in the grid defines the state of the puzzle. The state represents the current position of the tiles within the grid, which can vary as the puzzle is manipulated.