Knowledge Representation

Introduction to Genetic Algorithms:

Genetic algorithms are optimization methods inspired by natural selection. They start with a population of possible solutions, which evolve over time to find the best answer to a problem. Each solution's effectiveness is measured using a fitness function. The best solutions are selected to create new ones through crossover (mixing traits) and mutation (introducing random changes). Over several generations, these processes lead to better solutions. Genetic algorithms are useful for complex problems but may take time to find the best answer and sometimes get stuck on suboptimal solutions. They are widely used in fields like engineering and AI.

Terminologies related to Genetic Algorithm:

- Population: The group of potential solutions.
- Chromosome: The representation of a solution.
- Gene: A part of the chromosome representing a trait.
- Fitness Function: The function that evaluates how good a solution is.
- Selection: The process of choosing the best solutions for reproduction.
- Crossover (Recombination): The operator that combines two parent solutions to create offspring.
- Mutation: The operator that introduces random changes to a solution.
- Generation: One iteration of the genetic algorithm cycle.

Knowledge Representation:

Knowledge: Knowledge is awareness or familiarity gained by experiences of facts, data, and situations.

Knowledge representation is a key concept in artificial intelligence, focusing on how information about the world can be structured in a way that enables computers to solve complex problems. The goal is to create a computer-tractable form of knowledge that allows machines to reason and make decisions.

The process begins with an informal understanding of the problem, which is then formalized for computer processing. The computer generates an output that can be translated into a solution understandable to humans or verified for consistency. This progression from raw data to wisdom involves organizing and analyzing data to extract meaningful information, which, when interpreted, becomes knowledge.

A knowledge representation language consists of two main aspects:

- Syntax: Defines the valid configurations of symbols or words in the language.
- Semantics: Connects these configurations to real-world facts, dictating the meaning of sentences.

Challenges in knowledge representation include handling commonsense knowledge, balancing computational efficiency with inference accuracy, and managing uncertain information.

A good representation system must have four fundamental components:

- **Lexical**: Determines the vocabulary used.
- **Syntactic**: Describes the rules for arranging symbols.
- **Semantic**: Links the representations to real-world meanings.
- **Procedural**: Defines how the representations can be created, modified, and used to answer questions.

Types of Knowledge:

1. **Declarative Knowledge**: This is the knowledge of facts, concepts, and objects—essentially knowing "about" something. It's also known as descriptive knowledge and is often expressed in declarative sentences. This type of knowledge is simpler and foundational.
2. **Procedural Knowledge**: Also called imperative knowledge, this is the "know-how" type of knowledge. It includes rules, strategies, and procedures, and is directly applicable to tasks. The knowledge is specific to tasks and how they should be performed.
3. **Meta-Knowledge**: This refers to knowledge about other types of knowledge. It's essentially an understanding of the structure and nature of knowledge itself.
4. **Heuristic Knowledge**: This involves practical, experience-based techniques or "rules of thumb" that guide problem-solving. It's often used by experts to make decisions based on past experiences, though it doesn't guarantee correct solutions.
5. **Structural Knowledge**: This is the foundational knowledge used in problem-solving. It defines relationships between different concepts, such as how they are grouped or related, like "part of" or "kind of" relationships.

Issues in Knowledge Representation:

When we talk about Knowledge Representation, we're trying to find ways to organize and express information so that it can be used effectively to make conclusions or decisions. The core aim of Knowledge Representation is to enable logical reasoning from stored information. However, there are several issues that make this task challenging. Let's break down these problems into more manageable parts:

1. **Important Attributes:**

   In Knowledge Representation, two key attributes are "instance" and "IsA". These attributes are significant because they facilitate the inheritance of properties.

   Example: Consider the sentence "Joe is a musician."

   - "Is a" (referred to as IsA) expresses the class-instance relationship between "Joe" and "musician."
   - In this context, "Joe" is an instance of the class "musician." This means that Joe is a specific example of the broader category known as musicians.
   - "Joe" plays the role of an instance, while "musician" represents the class in this relationship.
   - This is specified as:
     [Joe] IsA [Musician]
     i.e., [Instance] IsA [Class]

2. **Relationship among attributes:**
   The attributes we use to describe objects are themselves entities that we represent. The relationship between the attributes of an object, independent of specific knowledge they encode, may hold properties like: Inverses, existence in an isa hierarchy, techniques for reasoning about values and single valued attributes.

3. **Choosing Granularity:**
   Choosing granularity in Knowledge Representation is about deciding how detailed your representation of knowledge should be. This choice affects how effectively the information can be used for reasoning and how efficiently it can be stored.
   For example, if you're representing the fact that "John spotted Sue," you could choose different levels of granularity:

   - **High-Level Fact:** Simply stating "John spotted Sue" provides a broad overview. This is easy to manage but might not give enough detail for deeper analysis. It tells us who was involved but doesn’t explain the context or specifics of the interaction.

   - **Low-Level Primitives:** A more detailed representation might be "Spotted (agent(John), object(Sue))." This includes precise information about the roles of John and Sue in the event, specifying that John is the one who spotted and Sue is the object of the spotting. This level of detail is useful for complex reasoning but requires more storage and processing.

- **Balance:** You need to find a middle ground between having enough detail for accurate analysis and keeping things manageable and efficient.

4. Set of Objects

Certain properties are true for entire groups of objects rather than for individual ones. For instance, statements like "There are more sheep than people in Australia" and "English speakers can be found all over the world" are true for entire sets of objects (sheep, people, English speakers), not just for individual members.

To represent these facts efficiently, it's better to attach the property to the entire set rather than each individual. This approach is practical because:

- **Efficiency**: If a property applies to most or all elements of a set, associating it with the set as a whole is more efficient than repeating it for each member.
- **Logical and Hierarchical Representation**: This can be done using logical representations like universal quantifiers or through hierarchical structures, which simplify the management and updating of information.

By representing properties at the set level, we streamline the process of handling and reasoning about large groups of objects.

5. Finding Right Structure:

Finding the right structure in Knowledge Representation involves selecting the best way to organize and describe information for a given situation. Here are the key steps:

- **Initial Selection:** Choose an initial structure that seems best suited to represent the information effectively.
- **Detailing**: Add the necessary details to the chosen structure based on the specific situation.
- **Revising**: If the initial structure isn't effective, find and switch to a more suitable structure.

- **Exploring Alternatives:** If none of the existing structures work, consider creating a new structure that better fits the needs.
- **Creation and Memory**: Develop and remember new structures if necessary, for future use.

Approaches to knowledge representation:

Approaches to Knowledge Representation are methods used to organize and structure information so that a system can understand and use it for tasks like reasoning and decision-making. Each approach is tailored to different types of problems, helping systems effectively process and apply knowledge.

There are mainly four approaches to knowledge representation, which are givenbelow:

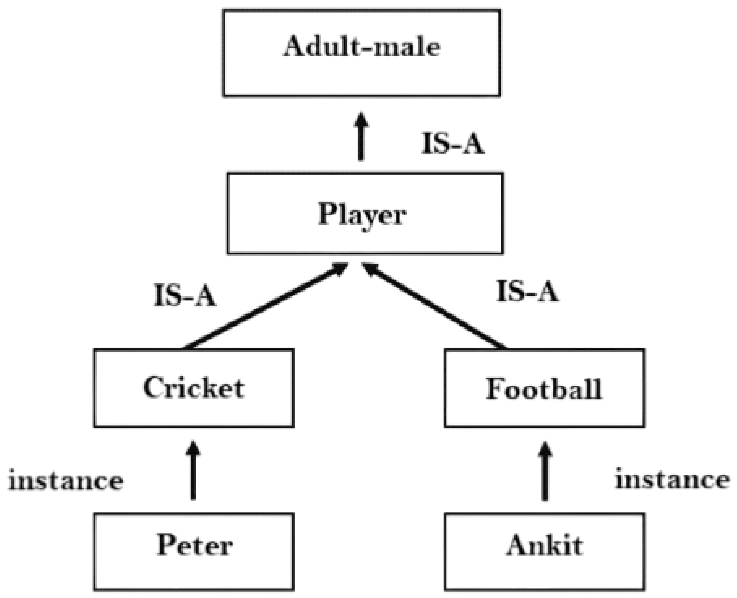
1. Simple relational knowledge:

- It is the simplest way of storing facts which uses the relational method, and each fact about a set of the object is set out systematically in columns.
- This approach of knowledge representation is famous in database systems where the relationship between different entities is represented.
- This approach has little opportunity for inference.
- Example: The following is the simple relational knowledge representation:

Player	Weight	Age
Player1	65	23
Player2	58	18
Player3	75	24

2. Inheritable knowledge:

- In the inheritable knowledge approach, all data must be stored into a hierarchy of classes.
- All classes should be arranged in a generalized form or a hierarchal manner.
- In this approach, we apply inheritance property.
- Elements inherit values from other members of a class.
- This approach contains inheritable knowledge which shows a relation between instance and class, and it is called instance relation.
- Every individual frame can represent the collection of attributes and its value.
- In this approach, objects and values are represented in Boxed nodes.
- We use Arrows which point from objects to their values.
- **Example:**



3. Inferential knowledge:

- Inferential knowledge approach represents knowledge in the form of formal logics.
- This approach can be used to derive more facts.
- It guaranteed correctness.
- **Example:** Let's suppose there are two statements:
    - Marcus is a man
    - All men are mortal

      Then it can represent as:

      man(Marcus)

      ∀x = man (x)----------> mortal (x)s

4. Procedural knowledge:

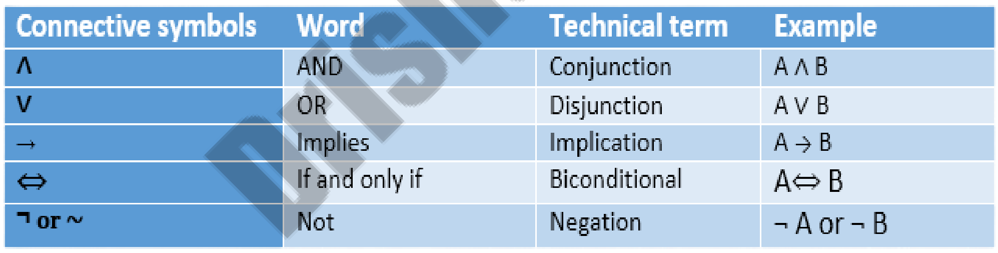
- Procedural knowledge approach uses small programs and codes which describes how to do specific things, and how to proceed.
- In this approach, one important rule is used which is **If-Then rule**.
- In this knowledge, we can use various coding languages such as **LISP language** and **Prolog language**.
- We can easily represent heuristic or domain-specific knowledge using this approach.
- But it is not necessary that we can represent all cases in this approach.

Representing Knowledge using Propositional calculus and predicate logic:

1. Propositional Logic (Propositional Calculus):

Propositional logic is a branch of logic where sentences (propositions) are declared to be either true or false. It is the most basic form of logic representation in AI.

- **Propositions:** These are simple, declarative sentences like "The sky is blue" or "It is raining." Each of these can be true or false.
- **Logical Connectives:** Propositional logic uses several operators to combine propositions:
    - AND (∧)
    - OR (∨)
    - NOT (¬)
    - IMPLICATION (→)
    - BICONDITIONAL (↔)
- **Syntax and Semantics:**
    - **Syntax:** Rules that determine how propositions can be combined using logical connectives.
    - **Semantics:** The truth values (True/False) assigned to the propositions and the resulting truth values of the combined propositions.



Connective symbols	Word	Technical term	Example
∧	AND	Conjunction	A ∧ B
∨	OR	Disjunction	A ∨ B
→	Implies	Implication	A → B
⇔	If and only if	Biconditional	A⇔ B
¬ or ~	Not	Negation	¬ A or ¬ B

- **Example:**
    - Let p = "It is raining"
    - Let q = "I will take an umbrella"
    - A knowledge representation could be: p→qp → qp→q (If it is raining, then I will take an umbrella).
- **Truth Table:**

  Propositional logic often uses truth tables to represent how the truth value of a complex expression depends on the truth values of its components. Following are the truth table for all logical connectives:

For Negation:

P	¬ P
True	**False**
False	**True**

For Conjunction:

P	Q	P∧Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**False**

For disjunction:

P	Q	P ∨ Q.
True	True	**True**
False	True	**True**
True	False	**True**
False	False	**False**

For Implication:

P	Q	P→ Q
True	True	**True**
True	False	**False**
False	True	**True**
False	False	**True**

2. Predicate Logic (Predicate Calculus):

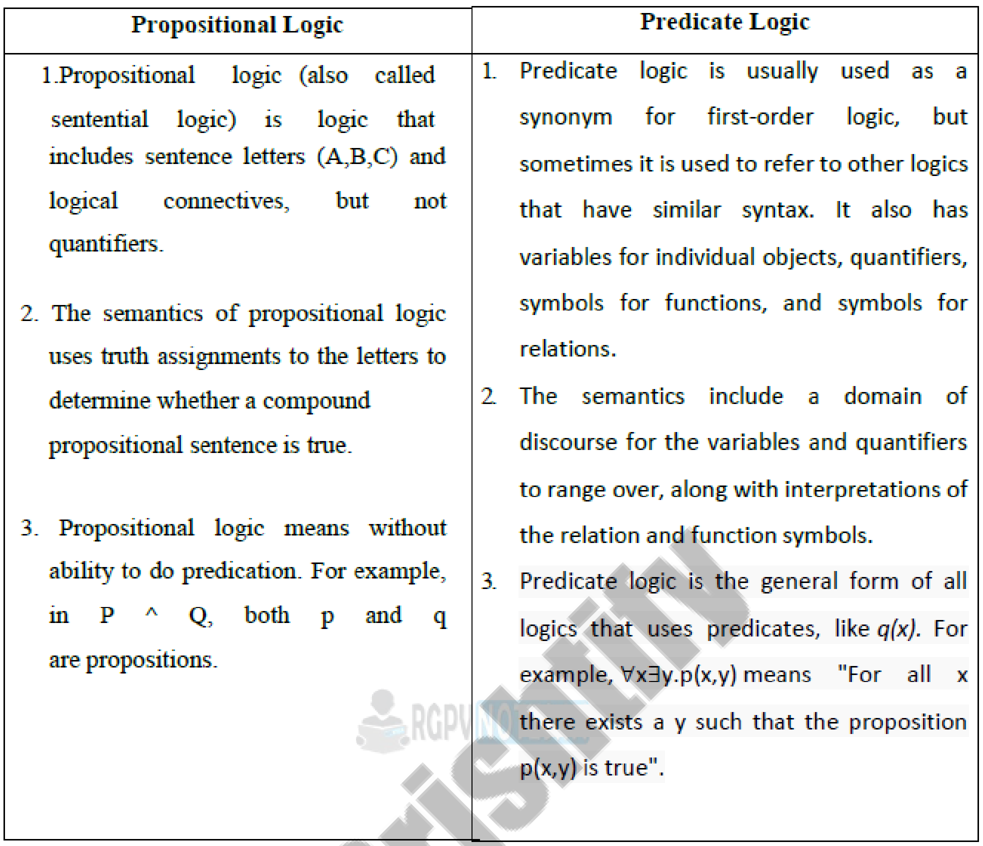
Predicate logic extends propositional logic by dealing with predicates and quantifiers, making it more expressive and powerful for knowledge representation.

- **Predicates**: Functions that return True or False based on the input values. For example, "IsTall(X)" might represent the predicate "X is tall."
- **Quantifiers**:
    - **Universal Quantifier (∀):** Indicates that a statement applies to all elements of a domain. Example: ∀X (IsTall(X)) means "All X are tall."
    - **Existential Quantifier (∃):** Indicates that a statement applies to at least one element of a domain. Example: ∃X (IsTall(X)) means "There exists an X that is tall."
- **Example:**
    - Consider the proposition: "All humans are mortal."
    - In predicate logic, we represent this as: ∀X(Human(X)→Mortal(X))

- Here, Human(X) is a predicate indicating "X is a human," and Mortal(X) indicates "X is mortal."

Comparison between propositional logic and predicate logic:

Comparison of Propositional and Predicate logic



Propositional Logic	Predicate Logic
1.Propositional logic (also called sentential logic) is logic that includes sentence letters (A,B,C) and logical connectives, but not quantifiers.	1. Predicate logic is usually used as a synonym for first-order logic, but sometimes it is used to refer to other logics that have similar syntax. It also has variables for individual objects, quantifiers, symbols for functions, and symbols for relations.
2. The semantics of propositional logic uses truth assignments to the letters to determine whether a compound propositional sentence is true.	2. The semantics include a domain of discourse for the variables and quantifiers to range over, along with interpretations of the relation and function symbols.
3. Propositional logic means without ability to do predication. For example, in P ^ Q, both p and q are propositions.	3. Predicate logic is the general form of all logics that uses predicates, like *q(x)*. For example, ∀x∃y.p(x,y) means "For all x there exists a y such that the proposition p(x,y) is true".

Logical Consequences:

Logical consequences refer to the relationship between statements in a logical system, where one statement (the consequence) logically follows from one or more other statements (the premises). If the premises are true, then the logical consequence must also be true. This concept is fundamental in logic and reasoning, forming the basis for deduction and proof methods.

Basic Concept

Logical consequence is about determining whether a conclusion necessarily follows from a set of premises. For example, if we know "All humans are mortal" and "Socrates is a human," the logical consequence is "Socrates is mortal."

Formal Definition

In formal logic, a statement B is a logical consequence of a set of statements $\{A_1,A_2,\ldots,A_n\}$if, whenever all the statements $A_1,A_2,\ldots,A_n$ are true, B must also be true. This relationship is denoted as:

$\{A_1,A_2,\ldots,A_n\}\vDash B$

This notation means that BBB is true in all cases where the statements $A_1, A_2, \ldots, A_n$ are true.

Types of Logical Consequences

- **Syntactic Consequence:** Involves deriving the consequence through a formal system of rules (like in a proof). The focus here is on the structure of the statements.
- **Semantic Consequence:** Focuses on the meaning or interpretation of the statements. A statement is a semantic consequence if it is true in every model where the premises are true.

Example of Logical Consequence:

Premise 1: All birds can fly.

Premise 2: A sparrow is a bird.

Logical Consequence: A sparrow can fly.

Here, the statement "A sparrow can fly" logically follows from the premises. If both premises are true (all birds can fly, and a sparrow is a bird), then the conclusion must also be true.

Syntax and Semantics of an Expression:

Syntax and **semantics** are two crucial aspects of understanding and using a formal language, such as those used in logic, programming, and mathematics

1. **Syntax:**

   Syntax refers to the structure or the rules that govern how symbols and words can be arranged to form valid expressions or statements in a language. It is about the formal arrangement of symbols without considering their meaning. Syntax dictates the correct format or pattern that an expression must follow to be considered valid in the language.

   Example:

   In a mathematical expression, the syntax dictates that operators like "+" and "-" must be placed between numbers or variables. For example, "3 + 5" is syntactically correct, whereas "+ 3 5" is not.

   Similarly, in programming, a syntax error might occur if you forget a semicolon at the end of a statement in languages like C++ or Java.

2. **Semantics:**

   Semantics deals with the meaning behind the symbols and expressions that follow the syntactical rules. It involves interpreting the symbols in a way that makes sense according to the rules and context of the language.

Example: Continuing with the mathematical expression "3 + 5", the semantics would be the interpretation that this expression represents the sum of 3 and 5, which equals 8.

In programming, the semantics of a statement like int x = 3 + 5; means that x will hold the value of 8 after this line is executed.

Putting it together with an example:

Consider the expression in propositional logic: **¬(P∧Q)**

- **Syntax:** The syntax rules dictate that the negation symbol (¬) must be followed by a well-formed formula, and the conjunction (∧) connects two propositions P and Q.
- **Semantics:** The semantics of this expression is that it represents a statement that is true if it is not the case that both PPP and QQQ are true simultaneously.

Semantic Tableau:

The **semantic tableau** is a logical proof method used to determine the satisfiability of a formula or the consistency of a set of logical statements. It organizes the evaluation process into a tree-like structure, where each branch represents different logical possibilities. The process involves systematically breaking down complex formulas into simpler components using logical rules.

- **Decomposition**: Formulas are split into their logical components. For example, a conjunction P∧Q would break into P and Q, while a disjunction P∨Q creates two branches: one for P and one for Q.
- **Closure**: A branch closes if it contains a contradiction, such as both P and ¬P on the same branch. If all branches close, the original formula is unsatisfiable.
- **Outcome**: If any branch remains open (no contradiction), the formula is satisfiable. If all branches close, the formula is valid (unsatisfiable when negated).

<u>**Rules for Semantic Tableau**</u>:

1. Negation (¬):
    - Negation of a conjunction ¬(A∧B) Split into ¬A and ¬B on different branches.
    - Negation of a disjunction ¬(A∨B) Both ¬A and ¬B on the same branch.

2. **Conjunction (∧)**:

   A∧B goes on the same branch.

3. **Disjunction (∨)**:

   A∨B creates two branches: one for A and another for B.

4. **Implication (→)**:

   A→B becomes ¬A∨B, split into two branches.

5. **Biconditional (↔):**

   A↔B splits into (A∧B) and (¬A ∧ ¬B), each on separate branches.

6. **Closure:**

   A branch closes if it contains both a formula and its negation (e.g., A and ¬A).

**** example to be checked from college…**

Reasoning Methods:

Forward Reasoning:

This approach starts with known facts and applies inference rules to derive new facts until a conclusion is reached or no more information can be generated. For example, if you know "All humans are mortal" and "Socrates is a human," forward reasoning uses these facts to conclude "Socrates is mortal." The process is data-driven and involves systematically applying rules to initial facts.

Forward Reasoning Algorithm:

- Initialize with known facts.
- Apply inference rules to generate new facts.
- Continue until no new facts can be derived or the goal is achieved.

Forward Reasoning Example:

Facts:

- "All birds can fly."
- "A parrot is a bird."

Goal: Determine if a parrot can fly.

Process:

- Start with known facts: "A parrot is a bird" and "All birds can fly."
- Apply the rule: Since a parrot is a bird, and all birds can fly, we can conclude that "A parrot can fly."

Backward Reasoning:

This approach begins with a goal or hypothesis and works backward to determine if it can be supported by the available facts and rules. For instance, if the goal is to determine if "Socrates is mortal," backward reasoning checks if the facts "Socrates is a human" and the rule "All humans are mortal" can be used to support this conclusion. This method is goal-driven and involves verifying the necessary facts to support the desired conclusion.

Backward Reasoning Algorithm:

- Start with the goal or hypothesis.
- Identify the necessary facts and rules needed to support the goal.
- Verify or derive these facts from available information or by further reasoning.

Backward Reasoning Example:

Goal: Determine if a parrot can fly.

Facts:

- "All birds can fly."
- "A parrot is a bird."

Process:

- Start with the goal: "Determine if a parrot can fly."
- Check if the necessary fact is available: "A parrot is a bird."
- Verify the rule: "If something is a bird, it can fly."
- Since both the facts and rule support that a parrot is a bird and all birds can fly, conclude that "A parrot can fly."

Proof Methods:

Proof Methods are techniques used to establish the validity of statements or theorems based on logical reasoning. These methods are crucial in fields such as mathematics and artificial intelligence for verifying the correctness of algorithms, systems, or logical statements.

Theorem Proving:

Theorem proving is about showing that a specific statement (theorem) can be logically derived from a set of basic rules or previously proven statements. It uses precise logical steps to make sure the proof is correct and complete.

Substitution method:

In AI, particularly in logic programming and automated theorem proving, the substitution method is used to replace variables in logical expressions or rules with specific terms. This helps in matching patterns, resolving queries, and proving logical statements.

1. **Steps:**
   - Find the variables in the expression.
   - Decide what terms will replace these variables.
   - Replace the variables with the chosen terms.
   - Use the new expression to solve queries or simplify further.
2. **Example:**

   Rule: ancestor(X, Y) → (parent(X, Z) ∧ ancestor(Z, Y))

   Facts:
   - parent(John, Alice)
   - parent(Alice, Mary)

   Task: Prove ancestor(John, Mary).

   Process:

   1. Substitute X with John and Y with Mary:
      - ancestor(John, Mary) → (parent(John, Z) ∧ ancestor(Z, Mary))
      - Substitute Z with Alice:
        - Check if parent(John, Alice) and ancestor(Alice, Mary) are true.
        - Both facts are true.

   Conclusion:

   Substitution shows that ancestor(John, Mary) is true.

Unification Algorithm:

Unification is a process in AI and logic programming used to make two logical expressions identical by finding a substitution for their variables. It is essential for tasks such as pattern matching and query resolution.

Unification Algorithm:

Steps:

- Input: Two terms or expressions.
- Check if Identical: If they are the same, no changes needed.
- Handle Variables:
    - Replace a variable with the other term.
- Handle Functions:
    - Ensure function names and structures match.
    - Recursively unify their parts.
- Check for Conflicts: If unification leads to contradictions, it fails.
- Output: Return the substitution list or indicate failure.

Unification Example:

Problem: Unify f(X, a) and f(b, Y).

1. **Terms**: f(X, a) and f(b, Y).
2. **Compare Functions**: Both are functions f with two arguments.
3. **Unify Arguments:**
    - For the first arguments: X and b → Replace X with b.
    - For the second arguments: a and Y → Replace Y with a.

Result: The substitution list is {X/b, Y/a}. The terms are unified with these substitutions.

Conversion to the Clausal Form:

Conversion to clausal form is the process of transforming a logical expression into a standardized format called clausal form. This format consists of a conjunction of clauses, where each clause is a disjunction of literals. This standardization is useful in automated theorem proving and logic programming.

Steps to Convert to Clausal Form:

1. Start with a Logical Expression: Begin with a logical formula in propositional

or first-order logic.

2. Eliminate Implications: Convert implications into disjunctions using the rule: A→B is equivalent to ¬A∨B.
3. Move Negations Inward: Apply De Morgan's laws and double negation to ensure negations only appear directly in front of literals.
4. Standardize Variables: Ensure all variables are uniquely named (important in first-order logic).
5. Convert to Prenex Normal Form: Move all quantifiers to the front of the expression. This step is specific to first-order logic.
6. Convert to Conjunctive Normal Form (CNF):
    - Distribute disjunctions over conjunctions.
    - Apply distributive laws to obtain a conjunction of disjunctions.
7. Extract Clauses: Each disjunction in the CNF is a clause, and the formula is a conjunction of these clauses.

Example:

Problem: Convert the formula (p → q) ∧ (¬q ∨ r) to clausal form.

1. Start with the Formula: (p → q) ∧ (¬q ∨ r).
2. Eliminate Implications:
    - p → q becomes ¬p ∨ q.
    - The formula becomes (¬p ∨ q) ∧ (¬q ∨ r).
3. Move Negations Inward:

    The formula (¬p ∨ q) ∧ (¬q ∨ r) already has negations directly in front of literals.
4. Standardize Variables:

    No additional variables need renaming in this example.
5. Convert to Prenex Normal Form:

    The formula is already in prenex normal form since it has no quantifiers.
6. Convert to CNF:

    The formula (¬p ∨ q) ∧ (¬q ∨ r) is already in CNF (a conjunction of disjunctions).
7. Extract Clauses:

Clauses are ¬p ∨ q and ¬q ∨ r.

Conclusion: The clausal form of the formula (p → q) ∧ (¬q ∨ r) is a conjunction of the clauses ¬p ∨ q and ¬q ∨ r.

Normal Forms:

Normal forms are specific ways to write logical expressions that help make them easier to work with. They follow certain patterns, making it simpler to analyze, simplify, or process logical statements.

Conjunctive Normal Form (CNF):

A formula in CNF is a conjunction (AND) of clauses, where each clause is a disjunction (OR) of literals. It's commonly used in automated reasoning for simplifying logical expressions.

Disjunctive Normal Form (DNF):

A formula in DNF is a disjunction (OR) of conjunctions (ANDs) of literals. This format simplifies analysis by breaking expressions into easier-to-evaluate conditions.

Prenex Normal Form:

In Prenex Normal Form, all quantifiers (∀, ∃) are at the start, followed by a quantifier-free part in CNF or DNF. It helps in separating quantifiers from the rest of the formula, aiding manipulation and reasoning.

Skolem Normal Form:

To achieve Skolem Normal Form, convert a formula to Prenex Normal Form and then replace all existential quantifiers with Skolem functions or constants. This simplifies the formula by eliminating existential quantifiers while keeping logical equivalence.

Resolution:

Resolution is a method used to determine if a set of logical statements is contradictory, meaning they can't all be true at the same time. It helps prove whether something is logically unsatisfiable by trying to find a contradiction.

How It Works:

1. Convert to Clausal Form:

   First, change the logical statements into a standardized format called clausal form (CNF). In CNF, the statements are written as a series of clauses, which are groups of literals connected by "OR" and joined together with "AND".

2. Use the Resolution Rule:

- The resolution rule combines two clauses that have opposite literals (one clause has a literal and the other has its negation) to create a new clause.
- Example: For clauses (A ∨ B) and (¬A ∨ C), resolving on A gives us the new clause (B ∨ C).

3. Generate New Clauses:

   Apply the resolution rule to pairs of clauses to produce new ones. Repeat this process with the new clauses.

4. Look for a Contradiction:

   If this process eventually produces an empty clause (a clause that is always false), it means the original set of clauses is contradictory and cannot all be true at the same time.

5. Check the Result:

   If you get an empty clause, it shows that the original set of statements is unsatisfiable. If you can't derive an empty clause and run out of ways to combine clauses, then the statements might be satisfiable.

Example of Resolution:

Statements:

1. Anyone passing history exams and winning the lottery is happy.
2. Anyone who studies or is lucky can pass all exams.
3. John did not study but he is lucky.
4. Anyone who is lucky wins the lottery.

Sol^{n}. The variables and their meanings used in the resolution example:

- P(x): "x passes history exams"
- W(x): "x wins the lottery"
- H(x): "x is happy"
- S(x): "x studies"
- L(x): "x is lucky"
- ¬: Represents negation (not)
- ∨: Represents logical OR
- ∧: Represents logical AND
- →: Represents logical implication (if... then...)

1. **Statements and Clauses:**

   - **Statement 1:** (P(x) ∧ W(x)) → H(x)
     Clausal Form: ¬P(x) ∨ ¬W(x) ∨ H(x)

   - **Statement 2:** (S(x) ∨ L(x)) → P(x)
     Clausal Form: ¬S(x) ∨ ¬L(x) ∨ P(x)

   - **Statement 3:** ¬S(John) and L(John)

   - **Statement 4:** L(x) → W(x)
     Clausal Form: ¬L(x) ∨ W(x)

2. **Apply Resolution:**

   - **Resolve John's luck with the lottery:**
     Clauses: L(John) and ¬L(John) ∨ W(John)
     Result: W(John) (John wins the lottery)

   - **Resolve John's exam passing:**
     Clauses: ¬S(John) ∨ ¬L(John) ∨ P(John), ¬S(John), and L(John)
     Result: P(John) (John passes history exams)

   - **Resolve John's happiness:**
     Clauses: ¬P(John) ∨ ¬W(John) ∨ H(John), P(John), and W(John)
     Result: H(John) (John is happy)

   Resolution shows that H(John) (John is happy), confirming that John is indeed happy based on the given statements.

Refutation

Refutation is a method to show that a logical statement is false by deriving a contradiction. If you can prove that assuming the statement leads to a logical contradiction, then the statement must be false.

Example:
Assume we want to refute the statement "All cats are black."

- Assumption: "All cats are black."
- Find a counterexample: "There is a white cat."
- Derive a contradiction: If a cat is white, then not all cats are black.

Conclusion: The statement "All cats are black" is refuted because it leads to a contradiction.

Deduction

Deduction is the process of deriving specific conclusions from general premises using logical rules. It involves starting with known facts or premises and applying logical reasoning to reach a conclusion.

Example:

- Premise 1: "All humans are mortal."
- Premise 2: "Socrates is a human."
- Deduce: "Socrates is mortal."

Conclusion: By applying deduction, we conclude that Socrates is mortal based on the given premises.

Inferencing

Inferencing is the process of deriving new information from existing information using logical reasoning. It involves making conclusions based on known facts and relationships.

Example:

- Fact 1: "If it rains, the ground will be wet."
- Fact 2: "It is raining."
- Infer: "The ground is wet."

Conclusion: By inferring from the given facts, we conclude that the ground is wet because it is raining.

Monotonic Reasoning:

Monotonic reasoning is a type of logical process where once a conclusion is reached based on certain facts or premises, that conclusion remains valid even if new information is added. In other words, the set of conclusions only grows as more information is added, and nothing is ever retracted. This type of reasoning is straightforward and works well in situations where the information is complete and unchanging. However, it doesn't handle real-world situations effectively because in reality, new information often contradicts or changes our understanding of things.

Features of Monotonic Reasoning:

- Once a conclusion is drawn, it remains valid regardless of new information.
- The reasoning process is straightforward, with no surprises or retractions.
- Easier to implement and understand due to its fixed nature.
- Cannot adapt to new or changing information that contradicts previous conclusions.

Non- Monotonic Reasoning:

Non-monotonic reasoning, on the other hand, allows for the revision of conclusions as new information becomes available. This type of reasoning is closer to how humans think and reason in everyday life. We often make assumptions based on the best information we have at the time, but we are ready to change our conclusions if new evidence suggests otherwise. This flexibility makes non-monotonic reasoning more practical for dealing with real-world scenarios where the information is incomplete or subject to change.

Features of Non-Monotonic Reasoning:

- Allows for the revision of conclusions when new information is introduced.
- Suitable for real-world situations where information is incomplete or may change.
- More complex as it mimics human reasoning and decision-making.
- Conclusions are not final and can evolve with new evidence.

Difference between Monotonic & Non-Monotonic Reasoning:

Monotonic Reasoning	Non-Monotonic Reasoning
Always adds new facts without retracting old conclusions.	Allows for revision of conclusions when new information is introduced.
Assumes that all facts are consistent and unchanging.	Assumes that information can change or be incomplete.
Less practical, as it doesn't adapt well to changing situations.	More practical, as it adapts to new evidence and changing situations.
Simpler and more straightforward.	More complex, mimicking human reasoning.
Mathematical proofs where conclusions are final.	Legal reasoning where new evidence can change the outcome.