Assignment 6: Implementation of Unification Algorithm by considering Resolution concept

Name: ______shreyash nale___________________

Roll No.: ______________331002___________

Branch: Computer Engineering

Semester: ___5th______

Subject: Artificial Intelligence

Date of Submission: 22-09-2025

1. Aim

The objective of this assignment is to understand and implement the Unification algorithm, a fundamental component of automated theorem proving in First-Order Logic (FOL). We will explore its role within the Resolution inference rule to determine if two logical expressions can be made identical through substitution.

2. Theory

2.1. Introduction to Automated Reasoning In Artificial Intelligence, a key goal is to create systems that can reason logically. Automated theorem provers are programs that use logical inference to derive new knowledge from existing facts and rules. First-Order Logic (FOL) provides a powerful and expressive language for representing this knowledge.

2.2. Resolution in First-Order Logic Resolution is a powerful inference rule used for automated theorem proving, particularly in systems based on FOL. It is a single, complete rule that can be used to show that a set of logical clauses is unsatisfiable. The process generally involves two main steps:

1. **Conversion to Clause Form (Conjunctive Normal Form - CNF):** All sentences in the knowledge base are converted into a standard form of clauses. A clause is a disjunction of literals (e.g., P(x) ∨ ¬Q(y)).
2. **Resolution Inference:** The resolution rule is applied iteratively. For two clauses containing complementary literals (e.g., L1 and ¬L2), if these literals can be made identical, a new clause (the resolvent) is generated containing all other literals from the parent clauses.

For example, from P(x) ∨ Q(x) and ¬P(A) ∨ R(x), if we can make P(x) and P(A) match, we can infer a new clause. This "matching" process is where unification comes in.

2.3. Unification Unification is the core process by which we find a substitution that makes two logical expressions look identical. In the context of resolution, it is used to make two complementary literals match so the resolution rule can be applied.

- **Expression:** An expression can be a constant, a variable, or a function with arguments (e.g., `A`, `x`, `FatherOf(John)`).
- **Substitution:** A substitution is a mapping of variables to terms. For example, a substitution $\theta=\{x/A, y/FatherOf(B)\}$ means "replace variable `x` with constant `A`, and variable `y` with the term `FatherOf(B)`".
- **Unifier:** A substitution θ is a unifier for two expressions E1 and E2 if applying the substitution makes them identical, i.e., $E1\theta=E2\theta$.
- **Most General Unifier (MGU):** The MGU is a unifier that is the most general possible. It makes the fewest commitments and does not substitute more than is necessary. For example, to unify P(x) and P(y), the MGU is $\{x/y\}$ (or $\{y/x\}$). A less general unifier would be $\{x/A, y/A\}$, which also works but is overly specific. Automated provers always seek the MGU.

For resolution to work on literals like $P(x,B)$ and $\neg P(A,y)$, the arguments must be unified. Here, we need to find a substitution that makes (x,B) and (A,y) identical. The MGU for this is $\{x/A, y/B\}$.

3. Unification Algorithm

The standard unification algorithm operates recursively. It compares the structure of two expressions and builds a substitution.

Input: Two logical expressions, E1 and E2. **Output:** The Most General Unifier (MGU) if it exists, otherwise failure.

1. **Initialization:** Start with an empty substitution $\theta=\{\}$.
2. **Comparison:**
    - If $E1=E2$, return the empty substitution (they already match).
    - If E1 is a variable, and E1 does not occur in E2 (occurs check), then substitute $E1/E2$. Return $\{E1/E2\}$.
    - If E2 is a variable, and E2 does not occur in E1 (occurs check), then substitute $E2/E1$. Return $\{E2/E1\}$.
    - If both E1 and E2 are functions (or predicates), their names must match, and they must have the same number of arguments. If not, fail.
        - Unify the first arguments of E1 and E2 to get a substitution θ1. If it fails, unification fails.
        - Apply θ1 to the remaining arguments of E1 and E2.
        - Recursively unify the modified remaining arguments to get θ2. If it fails, unification fails.
        - Return the composition of θ1 and θ2.
    - In all other cases, fail (e.g., trying to unify two different constants like `A` and `B`, or a function with a constant).

The "Occurs Check" is crucial to prevent infinite loops, such as when trying to unify x with F(x). Substituting x with F(x) would lead to F(x) and F(F(x)), and so on.

4. Example Walkthrough

Let's unify the following two expressions, which could be arguments of literals in a resolution step:

- $E1=P(x,F(y),B)$
- $E2=P(A,F(G(z)),w)$

1. **Compare Predicates:** Both are `P`. They match. Move to arguments.
2. **Unify Arguments 1:** Unify `x` and `A`.
    - `x` is a variable. The MGU is $\theta1=\{x/A\}$.
3. **Unify Arguments 2:** Apply $\theta1$ to the remaining parts and unify them. The expressions for the second arguments are `F(y)` and `F(G(z))`.
    - Compare function names: Both are `F`. Match.
    - Unify their arguments: `y` and `G(z)`.
    - `y` is a variable. The MGU is $\theta2=\{y/G(z)\}$.
4. **Unify Arguments 3:** Apply the combined substitution $\{x/A,y/G(z)\}$ to the remaining parts and unify them. The expressions for the third arguments are `B` and `w`.
    - Unify `B` and `w`.
    - `w` is a variable. The MGU is $\theta3=\{w/B\}$.
5. **Compose Substitutions:** The final MGU is the composition of all substitutions found:
    - MGU = $\theta1 \cup \theta2 \cup \theta3=\{x/A,y/G(z),w/B\}$

Applying this MGU to both original expressions:

- $E1\theta=P(A,F(G(z)),B)$
- $E2\theta=P(A,F(G(z)),B)$ They are now identical.

5. Conclusion

The Unification algorithm is an indispensable tool in logic programming and automated reasoning. It provides the mechanism for the Resolution rule to operate on expressions in First-Order Logic by finding the most general substitution to make terms identical. Its implementation requires careful handling of recursion, data structures to represent logical terms, and the critical "occurs check" to prevent logical inconsistencies and infinite loops. Through this assignment, we have successfully analyzed the theory and procedure of this fundamental AI algorithm.