Assignment 6: Implementation of Unification Algorithm by considering Resolution concept

Name: Rahul Sharma

Roll No.: 331015

Branch: Computer Engineering

Semester: 5th

Subject: Artificial Intelligence

Date of Submission: 22-09-2025

1. Aim

The goal of this assignment is to comprehend and implement the Unification algorithm, a crucial component of automated theorem proving in First-Order Logic (FOL). We will investigate its function within the Resolution inference rule to ascertain if two logical expressions can be rendered identical through substitution.

2. Theory

2.1. Introduction to Automated Reasoning In Artificial Intelligence, a primary objective is to build systems that can reason logically. Automated theorem provers are applications that utilize logical inference to deduce new information from existing facts and principles. First-Order Logic (FOL) offers a potent and expressive language for representing this information.

2.2. Resolution in First-Order Logic Resolution is a potent inference rule employed for automated theorem proving, especially in systems based on FOL. It is a sole, complete rule that can be utilized to demonstrate that a set of logical clauses is unsatisfiable. The procedure commonly involves two primary steps:

1. **Conversion to Clause Form (Conjunctive Normal Form - CNF):** All statements in the knowledge base are transformed into a standard form of clauses. A clause is a disjunction of literals (e.g., $P(x) \vee \neg Q(y)$).
2. **Resolution Inference:** The resolution rule is applied repeatedly. For two clauses holding complementary literals (e.g., L1 and ¬L2), if these literals can be rendered identical, a new clause (the resolvent) is produced containing all other literals from the original clauses.

For instance, from $P(x) \vee Q(x)$ and $\neg P(A) \vee R(x)$, if we can make P(x) and P(A) correspond, we can deduce a new clause. This "correspondence" process is where unification is applied.

2.3. Unification Unification is the fundamental procedure by which we find a substitution that causes two logical expressions to appear identical. In the framework of resolution, it is employed to

make two complementary literals match so the resolution rule can be executed.

- **Expression:** An expression can be a constant, a variable, or a function with parameters (e.g., `A`, `x`, `FatherOf(John)`).
- **Substitution:** A substitution is a mapping of variables to terms. For instance, a substitution θ={x/A,y/FatherOf(B)} signifies "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 renders them identical, i.e., E1θ=E2θ.
- **Most General Unifier (MGU):** The MGU is a unifier that is the most general possible. It makes the minimum commitments and does not substitute more than is required. For instance, to unify P(x) and P(y), the MGU is {x/y} (or {y/x}). A less general unifier might be {x/A,y/A}, which also functions but is overly specific. Automated provers consistently seek the MGU.

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

3. Unification Algorithm

The conventional unification algorithm functions recursively. It compares the structure of two expressions and constructs 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 θ={}.
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 vital to prevent infinite recursions, for example when attempting to unify x with F(x). Replacing x with F(x) would result in F(x) and F(F(x)), and so forth.

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 θ1={x/A}.
3. **Unify Arguments 2:** Apply θ1 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 θ2={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 θ3={w/B}.
5. **Compose Substitutions:** The final MGU is the composition of all substitutions found:
    - MGU = θ1 ∪ θ2 ∪ θ3={x/A,y/G(z),w/B}

Applying this MGU to both original expressions:

- E1θ=P(A,F(G(z)),B)
- E2θ=P(A,F(G(z)),B) They are now identical.

5. Conclusion

The Unification algorithm is an essential utility in logic programming and automated reasoning. It supplies the mechanism for the Resolution rule to function on expressions in First-Order Logic by discovering the most general substitution to make terms identical. Its execution demands careful management of recursion, data structures to represent logical terms, and the critical "occurs check" to avert logical inconsistencies and infinite loops. Through this assignment, we have effectively examined the theory and procedure of this foundational AI algorithm.