Unification in Computer Science and Logic

Unification is an algorithmic procedure used in computer science and logic to solve equations involving symbolic expressions. It is a fundamental concept in automated deduction, enabling the process of identifying two symbolic expressions by replacing certain sub-expression variables with other expressions. This process plays a crucial role in various fields, particularly in logic programming, programming language type systems, SMT solvers, Cryptographic protocol analysis and term-rewriting algorithms.

Unification, a concept attributed to John Alan Robinson, is the core operation of his resolution principle. Robinson demonstrated that unifiable terms have at most one general unifier. Various frameworks of unification are distinguished based on the expressions involved in the unification problem. In first-order unification, higher-order variables (variables representing functions) are allowed in the expressions. Free unification, or syntactic unification, requires a solution to make both sides of the equation equal.

The solution to a unification problem is represented by substitution, which involves mapping a symbolic value to each variable in the problem's expressions. Essentially, unification aims to find a substitution that unifies two given terms. The goal of a higher-order unification algorithm is to provide a minimal and complete substitution set (a set containing all relevant solutions without any redundant members) for a given problem. Therefore, unification is not only concerned with the solvability of a given problem but also, if solvable, with computing the most general unifier.

Unification is a fundamental concept in computational logic, enabling pattern matching and symbolic reasoning across multiple domains. It is crucial in **Prolog** for logic programming, **AI expert systems** for inference, and **functional programming** for symbolic computation. It also aids in **parsing techniques**, **deductive databases**, and **natural language processing** by resolving structural patterns. In **theorem proving** and **constraint solving**, unification supports automated deduction and proof verification. Additionally, it ensures consistency in **type inference algorithms** and plays a role in **cryptographic protocol analysis** by verifying security properties. Overall, unification is an essential tool for reasoning and problem-solving in computer science, AI, and formal verification systems.

Satisfiability Modulo Theories (SMT)

In computer science and mathematical logic, satisfiability modulo theories (SMT) addresses the problem of determining whether a mathematical formula is satisfiable. It extends the Boolean satisfiability problem (SAT) to encompass more complex formulas involving real numbers, integers, and various data structures such as lists, arrays, bit vectors, and strings. The term "modulo" refers to these expressions being interpreted within a certain formal theory in first-order logic with equality, often excluding quantifiers.

SMT solvers are tools designed to solve the SMT problem for a practical subset of inputs. While Boolean satisfiability is already NP-complete, the SMT problem is typically NP-hard, and in many theories, it is undecidable. Researchers explore which theories or subsets of theories lead to a decidable SMT problem and investigate the computational complexity of the decidable

cases. These decision procedures are often implemented directly in SMT solvers; for example, the decidability of Presburger arithmetic.

SMT can be regarded as a constraint satisfaction problem, providing a structured approach to constraint-solving.

Let Ψ_1 and Ψ_2 be two atomic sentences and σ be a unifier such that, $\boldsymbol{\Psi_1\sigma = \Psi_2\sigma}$, then it can be expressed as **UNIFY(Ψ_1, Ψ_2)**.

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

Let Ψ_1 = King(x), Ψ_2 = 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. **E.g.** Let's say there are two different expressions, **P(x, y), and P(a, f(z))**. In this example, we need to make both above statements identical to each other. For this, we will perform the substitution.

P(x,y).....(i)
P(a, f(z))..... (ii)

Substitute x with a, and y with f(z) in the first expression, and it will be represented as **a/x** and f(z)/y. With both the substitutions, the first expression will be identical to the second expression and the substitution set will be: **[a/x, f(z)/y]**.

Following are some basic 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.

<u>Unification Algorithm:</u>

Algorithm: Unify(Ψ_1, Ψ_2)

Step. 1: If Ψ_1 or Ψ_2 is a variable or constant, then:

a) If Ψ_1 or Ψ_2 are identical, then return NIL.

b) Else if $\Psi_1$is a variable,

a. then if Ψ_1 occurs in Ψ_2, then return FAILURE

b. Else return { (Ψ_2/ Ψ_1)}.

c) Else if Ψ_2 is a variable,

a. If Ψ_2 occurs in Ψ_1 then return FAILURE,

b. Else return {(Ψ_1/ Ψ_2)}.

d) Else return FAILURE.

Step.2: If the initial Predicate symbol in Ψ_1 and Ψ_2 are not same, then return FAILURE.

Step. 3: IF Ψ_1 and Ψ_2 have a different number of arguments, then return FAILURE.

Step. 4: Set Substitution set(SUBST) to NIL.

Step. 5: For i=1 to the number of elements in Ψ_1.

a) Call Unify function with the ith element of Ψ_1 and ith element of Ψ_2, and put the result into S.

b) If S = failure then returns Failure

c) If S ≠ NIL then do,

a. Apply S to the remainder of both Ψ_1 and Ψ_2.

b. SUBST= APPEND(S, SUBST).

Step.6: Return SUBST.

Implementation of the Algorithm

Step.1: Initialize the substitution set to be empty.

Step.2: Recursively unify atomic sentences:

Check for Identical expression match. If one expression is a variable v_i, and the other is a term t_i which does not contain variable v_i then:

Substitute t_i / v_i in the existing substitutions Add t_i /v_i to the substitution setlist (SUBST). If both the expressions are functions, then function name must be similar, and the number of arguments must be the same in both the expression.

For each pair of the following atomic sentences find the most general unifier (If exist).

Problem-1. Find the MGU of {p(f(a), g(Y)) and p(X, X)}

Soln: S_0 => Here, Ψ_1 = p(f(a), g(Y)), and Ψ_2 = p(X, X)
SUBST θ= {f(a) / X}
S1 => Ψ_1 = p(f(a), g(Y)), and Ψ_2 = p(f(a), f(a))
SUBST θ= {f(a) / g(Y)}, **Unification failed**.

Problem-2. Find the MGU of {p(b, X, f(g(Z))) and p(Z, f(Y), f(Y))}

Here, Ψ_1 = p(b, X, f(g(Z))) , and Ψ_2 = p(Z, f(Y), f(Y))
S_0 => { p(b, X, f(g(Z))); p(Z, f(Y), f(Y))}
SUBST θ={b/Z}

S_1 => { p(b, X, f(g(b))); p(b, f(Y), f(Y))}
SUBST θ={f(Y) /X}

S_2 => { p(b, f(Y), f(g(b))); p(b, f(Y), f(Y))}
SUBST θ= {g(b) /Y}

S_2 => { p(b, f(g(b)), f(g(b)); p(b, f(g(b)), f(g(b))}

Unified Successfully.
And Unifier = { b/Z, f(Y) /X , g(b) /Y}.

Problem-3. Find the MGU of {p (X, X), and p (Z, f(Z))}

Here, Ψ_1 = {p (X, X), and Ψ_2 = p (Z, f(Z))
S_0 => {p (X, X), p (Z, f(Z))}
SUBST θ= {X/Z}
S1 => {p (Z, Z), p (Z, f(Z))}
SUBST θ= {f(Z) / Z}, **Unification Failed**.

4. Find the MGU of UNIFY(prime (11), prime(y))

Here, Ψ_1 = {prime(11) , and Ψ_2 = prime(y)}
S_0 => {prime(11) , prime(y)}

SUBST θ= {11/y}

S_1 => {prime(11) , prime(11)} , Successfully unified.
Unifier: {11/y}.

unification is not possible for these expressions.

Problem-5. Find the MGU of Q(a, g(x, a), f(y)), Q(a, g(f(b), a), x)}

Here, Ψ_1 = Q(a, g(x, a), f(y)), and Ψ_2 = Q(a, g(f(b), a), x)
S_0 => {Q(a, g(x, a), f(y)); Q(a, g(f(b), a), x)}
SUBST θ= {f(b)/x}
S_1 => {Q(a, g(f(b), a), f(y)); Q(a, g(f(b), a), f(b))}

SUBST θ= {b/y}
S_1 => {Q(a, g(f(b), a), f(b)); Q(a, g(f(b), a), f(b))},

Successfully Unified. Unifier: [a/a, f(b)/x, b/y].