(Syntactic) Unification

**Definition**

- (Syntactic) unification is the following problem: Given \( s \) and \( t \), find a substitution \( \sigma \) such that \( \sigma s = \sigma t \).
- If \( \sigma s = \sigma t \), then \( \sigma \) is called a unifier of \( s \) and \( t \) or a solution to the equation \( s =? t \).
- More generally, unification is about deciding \( \sigma s \approx_E \sigma t \) (non-ground word problem). But here, by unification we mean syntactic unification.
- Unification is decidable.
- Unification is theoretically and practically interesting:
  - Symbolic computation algorithms
  - Prolog
  - Type inference

**Example**

<table>
<thead>
<tr>
<th>Unifiers</th>
<th>( f(x) =?= f(a) )</th>
<th>has exactly one unifier: ( { x \mapsto a } )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( x =?= f(y) )</td>
<td>has many unifiers: ( { x \mapsto f(y) }, { x \mapsto f(a), y \mapsto a }, \ldots )</td>
</tr>
<tr>
<td></td>
<td>( f(x) =?= g(y) )</td>
<td>has no unifier</td>
</tr>
<tr>
<td></td>
<td>( x =?= f(x) )</td>
<td>has no unifier</td>
</tr>
</tbody>
</table>

- An equation \( s =? t \) may have zero, one, or more solutions.
- Some solutions are more general than others: \( \{ x \mapsto f(y) \} \) is more general than \( \{ x \mapsto f(a), y \mapsto a \} \).

**Unification Problems**

**Definition**

- A unification problem is a finite set of equations
  \[ S = \{ s_1 =? t_1, \ldots, s_n =? t_n \} \].
- A unifier or solution of \( S \) is a substitution \( \sigma \) such that \( \sigma s_i = \sigma t_i \) for all \( i = 1, \ldots, n \).
- \( \mathcal{U}(S) \) denotes the set of all unifiers of \( S \).
- \( S \) is unifiable if \( \mathcal{U}(S) \neq \emptyset \).
Solving the Unification Problem

Definition

A unification problem \( S = \{ x_1 = ? t_1, \ldots, x_n = ? t_n \} \) is in solved form iff

- the \( x_i \) are pairwise distinct variables,
- none of the \( x_i \) occurs in any of the \( t_j \).

In this case, we define the substitution \( \tilde{S} \) as follows:

\[
\tilde{S} := \{ x_1 \mapsto t_1, \ldots, x_n \mapsto t_n \}
\]

It is easy to see that \( \tilde{S} \) is a unifier of \( S \).

Next we show how to transform a unification problem into solved form, provided the unification problem has a solution.

Transformation into Solved Form

Transformation Rules

- **DELETE** \( \{ t = ? t \} \cup S \implies S \)
- **DECOMPOSE** \( \{ f(t_1) = f(t_2) \} \cup S \implies \{ t_1 = ? u_1, \ldots, t_n = ? u_n \} \cup S \)
- **ORIENT** \( \{ t = ? x \} \cup S \implies \{ x = ? t \} \cup S \) if \( t \notin X \)
- **ELIMINATE** \( \{ x = ? t \} \cup S \implies \{ x = ? t \} \cup \{ x \mapsto t \}(S) \)
  - if \( x \in \text{Var}(S) \)
  - and \( x \notin \text{Var}(t) \) (“occurs check”)

The symbol \( \cup \) denotes disjoint union: \( M_1 \cup M_2 := M_1 \cup M_2 \) provided \( M_1 \cap M_2 = \emptyset \).

Applying a substitution to a set of equations \( S \) means applying it to both sides of all equations in \( S \).

Example (1)

Success

\[
\begin{align*}
\{ x = ? f(a), g(x, x) = ? g(x, y) \} & \implies \text{ELIMINATE} \\
\{ x = ? f(a), g(f(a), f(a)) = ? g(f(a), y) \} & \implies \text{DECOMPOSE} \\
\{ x = ? f(a), f(a) = ? f(a), f(a) = ? y \} & \implies \text{DELETE} \\
\{ x = ? f(a), f(a) = ? y \} & \implies \text{ORIENT} \\
\{ x = ? f(a), y = ? f(a) \} & \implies \text{ORIENT}
\end{align*}
\]

Example (2)

Failure

\[
\begin{align*}
\{ f(x, x) = ? f(y, g(y)) \} & \implies \text{DECOMPOSE} \\
\{ x = ? y, x = ? g(y) \} & \implies \text{ELIMINATE} \\
\{ x = ? y, y = ? g(y) \} & \implies \text{ELIMINATE}
\end{align*}
\]

- No transformation rule is applicable to \( \{ x = ? y, y = ? g(y) \} \).
- \text{ELIMINATE} is not applicable to \( y = ? g(y) \) because the occurs check fails.
**Unification Algorithm**

**Definition**

\[ \text{Unify}(S) = \text{while there is some } T \text{ such that } S \Rightarrow T \text{ do} \]
\[ S := T; \]
\[ \text{end while} \]
\[ \text{if } S \text{ is in solved form then return } \vec{S} \text{ else fail} \]

**Properties of Unify**

- \textit{Unify} is nondeterministic:
  - If more than one transformation rule is applicable, say \( S \Rightarrow T_1 \) and \( S \Rightarrow T_2 \), then \textit{Unify} may choose arbitrarily between \( T_1 \) and \( T_2 \).
- \textit{Unify} is sound:
  - If \textit{Unify}(\( S \)) returns a substitution \( \sigma \), then \( \sigma \) is a unifier of \( S \).
- \textit{Unify} is complete:
  - If a unification problem \( S \) is solvable then \textit{Unify}(\( S \)) does not fail.
- \textit{Unify} terminates for all inputs.

**Further Properties of \textit{Unify}**

\textit{Unify} has some further properties that we can only state loosely because we did not formally introduce the necessary concepts:

- \textit{Unify} computes a most general unifier: e.g., for \( x \neq f(y) \) it will compute \( \{ x \mapsto f(y) \} \), not \( \{ x \mapsto f(a), y \mapsto a \} \).
- \textit{Unify} computes an idempotent unifier, i.e., a unifier \( \sigma \) such that \( \sigma \sigma = \sigma \). This rules out strange solutions such as \( \{ x \mapsto f(y), z_1 \mapsto z_2, z_2 \mapsto z_1 \} \) for the problem \( x \neq f(y) \).

**Earlier Failure Detection**

- Detecting unsolvability can be expensive because \textit{Unify} first computes a normal form.
- But if the unification problem contains
  - an equation \( f(\ldots) \neq g(\ldots) \) with \( f \neq g \) or
  - an equation \( x \neq t \) with \( x \in \text{Var}(t) \) and \( x \neq t \)
  then failure is immediate.
- Introduce a special unification problem \( \bot \) which is not in solved form.
- Add two more transformation rules:
  - \textsc{Clash} \quad \{ f(\overline{t}_n) \neq g(\overline{t}_n) \} \uplus S \Rightarrow \bot \quad \text{if } f \neq g
  - \textsc{Occurs-Check} \quad \{ x \neq t \} \uplus S \Rightarrow \bot \quad \text{if } x \in \text{Var}(t) \quad \text{and } x \neq t