Foundations of Programming Languages and Software Engineering

Jan-Georg Smaus (Peter Thiemann)

Universität Freiburg

July 2011

- Unification
(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 $s = ? t$. 
(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.
(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 $s \equiv t$.

More generally, unification is about deciding $\sigma s \equiv \varepsilon \sigma t$ (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
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 $\{x \mapsto f(a), y \mapsto a\}$. 
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
\[ \sigma s_i = \sigma t_i \text{ for all } i = 1, \ldots, n. \]

\( \mathcal{U}(S) \) 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** \[ \{ t =? t \} \uplus S \implies S \]

**DECOMPOSE** \[ \{ f(t_n) =? f(u_n) \} \uplus S \implies \{ t_1 =? u_1, \ldots, t_n =? u_n \} \cup S \]

**ORIENT** \[ \{ t =? x \} \uplus S \implies \{ x =? t \} \cup S \text{ if } t \notin X \]

**ELIMINATE** \[ \{ x =? t \} \uplus S \implies \{ x =? t \} \cup \{ x \mapsto t \}(S) \]

- The symbol \( \uplus \) denotes disjoint union: \( M_1 \uplus M_2 := M_1 \cup M_2 \) 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)

<table>
<thead>
<tr>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x = \text{?} \ f(a), g(x, x) = \text{?} \ g(x, y)}</td>
</tr>
<tr>
<td>{x = \text{?} \ f(a), g(f(a), f(a)) = \text{?} \ g(f(a), y)}</td>
</tr>
<tr>
<td>{x = \text{?} \ f(a), f(a) = \text{?} \ f(a), f(a) = \text{?} \ y}</td>
</tr>
<tr>
<td>{x = \text{?} \ f(a), f(a) = \text{?} \ y}</td>
</tr>
<tr>
<td>{x = \text{?} \ f(a), y = \text{?} \ f(a)}</td>
</tr>
</tbody>
</table>

Foundations of Programming Languages and Software Engineering
Example (2)

Failure

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

- 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.
**Definition**

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

- **Unify** is nondeterministic:
  If more than one transformation rule is applicable, say $S \rightarrow T_1$ and $S \rightarrow T_2$, then **Unify** may choose arbitrarily between $T_1$ and $T_2$.

- **Unify** is sound:
  If **Unify**($S$) returns a substitution $\sigma$, then $\sigma$ is a unifier of $S$.

- **Unify** is complete:
  If a unification problem $S$ is solvable then **Unify**($S$) does not fail.

- **Unify** terminates for all inputs.
Further Properties of *Unify*

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

- *Unify* computes a **most general** unifier: e.g., for \( x = ? f(y) \) it will compute \( \{ x \mapsto f(y) \} \), not \( \{ x \mapsto f(a), y \mapsto a \} \).

- *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 \( x = ? f(y) \).
Earlier Failure Detection

- Detecting unsolvability can be expensive because Unify first computes a normal form.
- But if the unification problem contains
  - an equation \( f(\ldots) = ? g(\ldots) \) with \( f \neq g \) or
  - an equation \( x = ? t \) 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:
  \[
  \text{C}LASH \quad \{ f(t_n) = ? g(u_n) \} \uplus S \implies \bot \quad \text{if} \quad f \neq g
  \]
  \[
  \text{O}ccurs-C\text{heck} \quad \{ x = ? t \} \uplus S \implies \bot \quad \text{if} \quad x \in \text{Var}(t) \quad \text{and} \quad x \neq t
  \]