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 σ such that $\sigma s = \sigma t$.
- If $\sigma s = \sigma t$, then σ is called a unifier of s and t or a solution to the equation $s =^? t$.

- More generally, unification is about deciding $\sigma s \approx_{\mathcal{E}} \sigma t$ (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

Unifiers

$f(x) =^? f(a)$	has exactly one unifier: $\{x \mapsto a\}$
$x =^? f(y)$	has many unifiers: $\{x \mapsto f(y)\}, \{x \mapsto f(a), y \mapsto a\}, \ldots$
$f(x) =^? g(y)$	has no unifier
$x =^? f(x)$	has no unifier

- 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 σ 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 $\vec{S}$ as follows:

$$\vec{S} := \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$$

- It is easy to see that $\vec{S}$ 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	$\{t =^? t\} \uplus S$	$\Longrightarrow S$
DECOMPOSE	$\{f(\overline{t_n}) =^? f(\overline{u_n})\} \uplus S$	$\Longrightarrow \{t_1 =^? u_1, ..., t_n =^? u_n\} \cup S$
ORIENT	$\{t =^? x\} \uplus S$	$\Longrightarrow \{x =^? t\} \cup S$ if $t \notin X$
ELIMINATE	$\{x =^? t\} \uplus S$	$\Longrightarrow \{x =^? t\} \cup \{x \mapsto t\}(S)$ if $x \in Var(S)$ and $x \notin Var(t)$ ("occurs check")

- 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)

Success

$$
\begin{array}{ll}
\{x =^? f(a), g(x,x) =^? g(x,y)\} & \Longrightarrow_{\text{ELIMINATE}} \\
\{x =^? f(a), g(f(a), f(a)) =^? g(f(a), y)\} & \Longrightarrow_{\text{DECOMPOSE}} \\
\{x =^? f(a), f(a) =^? f(a), f(a) =^? y\} & \Longrightarrow_{\text{DELETE}} \\
\{x =^? f(a), f(a) =^? y\} & \Longrightarrow_{\text{ORIENT}} \\
\{x =^? f(a), y =^? f(a)\} &
\end{array}
$$

Example (2)

Failure

$$
\begin{array}{ll}
\{f(x,x) =^? f(y, g(y))\} & \Longrightarrow_{\text{DECOMPOSE}} \\
\{x =^? y, x =^? g(y)\} & \Longrightarrow_{\text{ELIMINATE}} \\
\{x =^? y, y =^? g(y)\} &
\end{array}
$$

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

Unification Algorithm

Definition

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

Properties of Unify

- *Unify* is nondeterministic:
  If more than one transformation rule is applicable, say $S \Longrightarrow T_1$ and $S \Longrightarrow T_2$, then *Unify* may choose arbitrarily between T_1 and T_2.
- *Unify* is sound:
  If *Unify(S)* returns a substitution σ, then σ 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 σ 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 $f(\dots) =^? g(\dots)$ with $f \neq g$ or
  - an equation $x =^? t$ with $x \in Var(t)$ and $x \neq t$

  then failure is immediate.
- Introduce a special unification problem $\perp$ which is not in solved form.
- Add two more transformation rules:

CLASH	$\{f(\overline{t_n}) =^? g(\overline{u_n})\} \uplus S$	$\Longrightarrow \perp$	if $f \neq g$
OCCURS–CHECK	$\{x =^? t\} \uplus S$	$\Longrightarrow \perp$	if $x \in Var(t)$ and $x \neq t$