CS 565, Jens Palsberg Oct 21, 1998

Lecture Note 5: First-Order Unification

Contents

1 Terminology

Let us consider equations in which the variables range over finite trees. The problem of first-order unification is to decide whether a set of such equations has a solution. Moreover, if there is a solution, then we want the most general solution. One can solve this unification problem in $O(n)$ time, where n is the size of the equation set (M. S. Paterson and M. N. Wegman, "Linear Unification", Journal of Computer and System Sciences, 16:158–167, 1978.) Here, we will not cover this linear-time algorithm. Rather we will present a simpler algorithm with worse worst-case complexity.

For simplicity of notation, we will focus on equations where each side is a simple-type expression generated from the grammar

$$t ::= t \to t \mid \mathsf{Int} \mid \alpha$$

where α ranges over a set of type variables. A substitution is of the form

$$[\alpha_1 := s_1, \ldots, \alpha_m := s_m]$$

where $\alpha_1, \ldots, \alpha_m$ are type variables, and $s_1, \ldots, s_m$ are type expressions. When a substitution is applied to a type expression, $\alpha_1, \ldots, \alpha_m$ are replaced simultaneously. For example

$$\begin{aligned}
((\mathsf{Int} \to \alpha) \to \beta)[\alpha := \mathsf{Int}, \beta := (\mathsf{Int} \to \mathsf{Int})] &= (\mathsf{Int} \to \mathsf{Int}) \to (\mathsf{Int} \to \mathsf{Int}) \\
((\mathsf{Int} \to \alpha) \to \beta)[\alpha := \mathsf{Int}, \beta := (\mathsf{Int} \to \alpha)] &= (\mathsf{Int} \to \mathsf{Int}) \to (\mathsf{Int} \to \alpha) \\
((\mathsf{Int} \to \alpha) \to \gamma)[\alpha := \mathsf{Int}, \beta := (\mathsf{Int} \to \alpha)] &= (\mathsf{Int} \to \mathsf{Int}) \to \gamma.
\end{aligned}$$

Let G be the set of equations

$$\begin{aligned}
s_1 &= t_1 \\
&\ldots \\
s_n &= t_n.
\end{aligned}$$

If σ is a substitution, then we denote by $G\sigma$ the set of equations

$$\begin{aligned}
s_1\sigma &= t_1\sigma \\
&\ldots \\
s_n\sigma &= t_n\sigma.
\end{aligned}$$

If for every $i \in 1..n$ we have that $s_i\sigma$ is identical to $t_i\sigma$, then we say that G has solution σ. For example, the two equations

$$\begin{array}{rcl}\alpha & = & \beta \rightarrow \mathsf{Int} \\ \beta & = & \mathsf{Int} \rightarrow \mathsf{Int}\ ,\end{array}$$

have the solution σ where

$$\sigma \quad = \quad [\alpha := ((\mathsf{Int} \rightarrow \mathsf{Int}) \rightarrow \mathsf{Int}),\ \beta := (\mathsf{Int} \rightarrow \mathsf{Int})].$$

The equation

$$\alpha \quad = \quad \alpha \rightarrow \mathsf{Int}\ ,$$

has no solution because of the cycle. In general, the forms of equations we consider can be unsolvable for two reasons:

1. we encounter a *cycle*, that is, an equation of one of the forms

   $$\begin{array}{rcl}\alpha & = & (\ldots\ \alpha\ \ldots) \rightarrow \ldots \qquad \text{or} \\ \alpha & = & \ldots\ \rightarrow (\ldots\ \alpha\ \ldots)\ ,\end{array}$$

   or

2. we encounter an equation where neither of the two sides are variables, and where the two sides have *different root symbols*, that is, the equation is of the form $\mathsf{Int} = s \rightarrow t$.

Note that although equations with cycles and/or different root symbols may not be immediately present in an equation set, we may encounter such equations after replacing equals by equals.

If we have an equation $s = t$, and a substitution σ such that $s\sigma = t\sigma$, then σ is called a *unifier* of s and t, and we say that σ *unifies* s and t, that σ *solves* $s = t$, and that σ *satisfies* $s = t$. If for type expressions s, t, there exists a unifier σ of s and t, then we say that s and t are *unifiable*.

We will denote the composition of two substitutions σ, θ by $\sigma \circ \theta$, and this notation means

$$t(\sigma \circ \theta) \quad = \quad (t\sigma)\theta.$$

A substitution σ which solves an equation system is called a *most general solution* of that equation system provided that for any other solution θ, there exists a substitution τ such that $\theta = \sigma \circ \tau$.

2 A Unification Algorithm

We now present a first-order unification algorithm which either decides that an equation system has no solution, or produces the most general solution. The algorithm uses three variables G, $Failure$, σ:

- G ranges over sets of equations, and it is initially the input set of equations,
- $Failure$ is a boolean variable which is initially false, and
- σ is a substitution which is initially empty.

After initialization, the algorithm proceeds as follows:

while $G \neq \emptyset$ and (not Failure) do

> Choose and remove an equation e from G, and let $(s = t)$ be $(e\sigma)$. There are now six cases depending on the form of $(s = t)$.
>
> 1. If s, t are identical variables, or s, t are both Int, then do nothing.
> 2. If $s = s_1 \to s_2$ and $t = t_1 \to t_2$, then $G = G \cup \{s_1 = t_1, s_2 = t_2\}$.
> 3. If ($s = \mathsf{Int}$ and $t = t_1 \to t_2$), or ($s = s_1 \to s_2$ and $t = \mathsf{Int}$), then $Failure = true$.
> 4. If s is a variable that does not occur in t, then $\sigma = \sigma \circ [s := t]$.
> 5. If t is a variable that does not occur in s, then $\sigma = \sigma \circ [t := s]$.
> 6. If $s \neq t$, and either s is a variable that occurs in t, or t is a variable that occurs in s, then $Failure = true$.

end while

If $Failure$, then output "no solution", else output σ.

END of algorithm

The loop invariant is:

> If θ is the most general solution of G, then $\sigma \circ \theta$ is the most general solution of the input set of equations.

In case (4) and (5), the phrase "does not occur in" guards against cycles. This check is expensive computationally; it is widely known as the "occurs-check."

3 Examples

The first example has no solution because of a cycle, and the second has a solution.

3.1 Example 1

Equations:

$$\begin{aligned} \alpha &= \mathsf{Int} \to \beta \\ \beta &= \alpha \to \mathsf{Int}. \end{aligned}$$

Initially we have

$$\begin{aligned} G &= \{(\alpha = \mathsf{Int} \to \beta), (\beta = \alpha \to \mathsf{Int})\} \\ Failure &= false \\ \sigma &= \emptyset. \end{aligned}$$

Choose and remove, say, $(\alpha = \mathsf{Int} \to \beta)$ from G. We have $(\alpha = \mathsf{Int} \to \beta)\sigma \equiv (\alpha = \mathsf{Int} \to \beta)$, so we are in case (4). Change σ to $[\alpha := (\mathsf{Int} \to \beta)]$. We now have $G = \{(\beta = \alpha \to \mathsf{Int})\}$. Choose(!) and remove $(\beta = \alpha \to \mathsf{Int})$ from G. We have $(\beta = \alpha \to \mathsf{Int})\sigma \equiv (\beta = (\mathsf{Int} \to \beta) \to \mathsf{Int})$, so there is a cycle and we are in case (6), and the algorithm outputs that there is no solution.

3.2 Example 2

Equations:

$$\begin{aligned} \alpha &= \beta \to \mathsf{Int} \\ \beta &= \mathsf{Int} \to \mathsf{Int}. \end{aligned}$$

Initially we have

$$\begin{aligned} G &= \{(\alpha = \beta \to \mathsf{Int}), (\beta = \mathsf{Int} \to \mathsf{Int})\} \\ Failure &= false \\ \sigma &= \emptyset. \end{aligned}$$

Choose and remove, say, $(\alpha = \beta \to \mathsf{Int})$ from G. We have $(\alpha = \beta \to \mathsf{Int})\sigma \equiv (\alpha = \beta \to \mathsf{Int})$, so we are in case (4). Change σ to $[\alpha := (\beta \to \mathsf{Int})]$. We now have $G = \{(\beta = \mathsf{Int} \to \mathsf{Int})\}$. Choose(!) and remove $(\beta = \mathsf{Int} \to \mathsf{Int})$ from G. We have $(\beta = \mathsf{Int} \to \mathsf{Int})\sigma \equiv (\beta = \mathsf{Int} \to \mathsf{Int})$, so we are in case (4). Change σ to

$$\begin{aligned} &[\alpha := (\beta \to \mathsf{Int})] \circ [\beta := (\mathsf{Int} \to \mathsf{Int})] = \\ &[\alpha := (\mathsf{Int} \to \mathsf{Int}) \to \mathsf{Int},\ \ \beta := (\mathsf{Int} \to \mathsf{Int})]. \end{aligned}$$

We now have $G = \emptyset$, so the algorithm outputs σ as a most general solution of the initial equations.