# Unification Notation

Mattox Beckman

April 1, 2019

## 1 Definitions

You have seen how to solve unification problems informally, using the four operations in a manner similar to how you would solve a set of simultaneous linear equations. Now that you (we hope!) have a good intuition about how these are solved, we would like to introduce a more formal notation.

Let us define a *unification problem* as a set $S$ of *constraint pairs* of the form $\{s_1 = t_1, s_2 = t_2, ..., s_n = t_n\}$. The goal is to use this set $S$ to generate a *substitution function* $\sigma$. We will do this by way of a function unify.

A member of a pair can be a variable (usually denoted by Greek letters), a functor (a pattern that "looks like a function call"), or a symbol (usually denoted by roman letters).

The other notation you need to know is the notation for substitution. We will use the notation $[\alpha \to \beta]$ to indicate the function that replaces all free[1] occurrences of $\alpha$ by $\beta$ in its argument. To have more than one substitution we can compose[2] them. So if we have the substitution $[\alpha \to x]$ and $[\beta \to y]$ we can write $[\alpha \to x] \circ [\beta \to y]$. We can also write this composition as $[\alpha \to x; \beta \to y]$.

You will see two notations for using substitution in the literature. The most common is to put the substitution *after* the term. For example, we might say $(f(\alpha))[\alpha \to x] = f(x)$. Because substitution can be thought of as a function, we prefer to put it *before* the term. Thus, $[\alpha \to x](f(\alpha)) = f(x)$.

To solve a unification problem unify($S$), we will pick a pair $s_1 = s_2$ and apply one of the four rules to it, if any apply. If no rules apply then unification fails. unify($\{\}$) is the identity function.

## 2 The four rules

Here are how the four rules appear using this notation.

### 2.1 Decompose

Supposing that $f$ is an arbitrary functor, then we have:

$$\text{unify}(\{f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)\} \cup S) = \text{unify}(\{x_1 = y_1, \ldots, x_n = y_n\} \cup S)$$

### 2.2 Delete

Supposing $X$ is an arbitrary pattern, then we have:

$$\text{unify}(\{X = X\} \cup S) = \text{unify}(S)$$

### 2.3 Orient

If $\alpha$ is an arbitrary variable and $X$ is an arbitrary pattern but **not** a variable, then we have:

$$\text{unify}(\{X = \alpha\} \cup S) = \text{unify}(\{\alpha = X\} \cup S)$$

[1] We say "free" for completeness, but in the problems we are looking at, we will not have bound variables. When we use unification to do more advanced things like type inferencing, we will use *gensym* to ensure that all variables are distinct.

[2] The operator $\circ$ is the function composition operator, in case you forgot. Thus, $(f \circ g)(x) \equiv f(g(x))$.

### 2.4 Eliminate

If $\alpha$ is an arbitrary variable and $X$ is an arbitrary pattern that does not contain $\alpha$, then
$\text{unify}(\{\alpha = X\} \cup S) = \text{unify}([\alpha \to X](S)) \circ [\alpha \to X]$

So, given the substitution $[\alpha \to X]$, we apply it to the remaining pairs of the unification problem and "emit" it as part of our solution.

We must check first that $\alpha \notin X$. This is called the *occurs check*. If that fails, the result will be an infinite substitution. You sometimes will see this in certain [Haskell]{style="font-variant:small-caps;"} type errors when it complains that it cannot generate an "infinite type". This usually happens if you mix up a function that uses lists with a line that outputs the element of that list.

Here is an example that causes this error.

```
Prelude> let foo [] = []
Prelude|     foo (x:xs) = x : foo [xs]
Prelude|

<interactive>:4:27:
    Occurs check: cannot construct the infinite type: t ~ [t]
    Relevant bindings include
      xs :: [t] (bound at <interactive>:4:12)
      x :: t (bound at <interactive>:4:10)
      foo :: [t] -> [t] (bound at <interactive>:3:5)
    In the expression: xs
    In the first argument of `foo', namely `[xs]'
```

# 3 Sample Unification

Here is the sample unification problem in the slides. This example appears in [BN99].

$$
\begin{array}{ll}
 & \text{unify}(\{\alpha = f(x),\ g(\alpha, \alpha) = g(\alpha, \beta)\}) \\
\textit{eliminate} & \text{unify}(\{g(f(x), f(x)) = g(f(x), \beta)\}) \circ [\alpha \to f(x)] \\
\textit{decompose} & \text{unify}(\{f(x) = f(x), f(x) = \beta\}) \circ [\alpha \to f(x)] \\
\textit{delete} & \text{unify}(\{f(x) = \beta\}) \circ [\alpha \to f(x)] \\
\textit{orient} & \text{unify}(\{\beta = f(x)\}) \circ [\alpha \to f(x)] \\
\textit{eliminate} & \text{unify}(\{\}) \circ [\beta \to f(x)] \circ [\alpha \to f(x)] \\
= & [\beta \to f(x)] \circ [\alpha \to f(x)] \\
= & [\beta \to f(x); \alpha \to f(x)]
\end{array}
$$

# References


[BN99] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, Aug. 1999, p. 316.