

# Higher Order Unification

Chris Kauffman

Fall 2004

In the higher-order logic setting several things change which cause the first order unification scheme to become inadequate. First, a typed lambda calculus is used to construct terms. In this way, variables may appear in the place of function symbols which is not dealt with in the first order case. Secondly, the notion of equality of terms is expanded to either  $\beta$ -equivalence ( $\equiv_\beta$ ) or  $\beta\eta$ -equivalence ( $\equiv_{\beta\eta}$ ).

In the first order framework a pair of terms like

$$t_1 = (F A)$$

$$t_2 = (f x)$$

could not be unified as their function symbols differ. In the higher-order setting, if  $f$  and  $x$  are treated as variables, substituting  $F$  for  $f$  and  $A$  for  $x$  unifies the two terms. However, it is not the only solution to the unification problem: substituting  $\lambda y.FA$  for  $f$  and anything for  $x$  will also unify the two terms. Note also that these two unifiers, call them  $\sigma_1 = \{\langle f, F \rangle, \langle x, A \rangle\}$  and  $\sigma_2 = \{\langle f, \lambda y.FA \rangle, \langle x, z \rangle\}$  are independent of one another: there is no  $\eta$  such that  $\sigma_1\eta = \sigma_2$  or  $\sigma_1 = \sigma_2\eta$ . Examining the two substitutions reveals that they have a very different structure. In fact, this problem, along with many others in the higher-order setting, does not have a most general unifier. Instead, the notion of *complete sets of unifiers* is used.

We will examine methods for unifying higher-order terms discussed in [1]. The method grows a *matching tree* for a pair of terms. The leaves in this tree will indicate either failure or success. The edges are substitutions such that the composition of substitutions on a path from the root node to a 'success' leaf comprises a unifier for the two terms being analyzed. The method hinges on the use of two routines, *SIMPL* and *MATCH* and has the nice property of finding a complete set of unifiers while avoiding redundancy whenever possible.

## References

- [1] G.P. Huet. *A Unification Algorithm for Typed  $\lambda$ -Calculus*. Theoretical Computer Science, pages 27-57. North-Holland Publishing Company, 1975.