Unification in Lambda-Calculi with if-then-else

Michael Beeson

Department of Mathematics and Computer Science
San Jose State University
San Jose, California 95192, USA
beeson@mathcs.sjsu.edu

Abstract. A new unification algorithm is introduced, which (unlike previous algorithms for unification in \(\lambda\)-calculus) shares the pleasant properties of first-order unification. Proofs of these properties are given, in particular uniqueness of the answer and the most-general-unifier property. This unification algorithm can be used to generalize first-order proof-search algorithms to second-order logic, making possible for example a straightforward treatment of McCarthy's circumscription schema.\(^1\)

1 Introduction

The origins of this work lie partly in an attempt to build a theorem-prover capable of implementing John McCarthy's work on circumscription. Circumscription is a second-order axiom schema, and cannot, in its full generality, be reduced to first-order. Indeed, it does not seem any easier to build a theorem-prover to handle all cases of circumscription than to handle second-order logic in general.

Second-order logic uses \(\lambda\)-terms to define predicates, and hence any attempt to mechanize second-order logic necessarily will involve unification of \(\lambda\)-terms. Huet [Huet 1975] introduced an algorithm for unifying terms in \(\lambda\)-calculus, and Miller and Nadathur [Miller and Nadathur 1988] introduced an extension of Prolog to a fragment of second-order logic (in fact higher-order logic) based on a similar unification algorithm. However, it turned out that Huet's algorithm isn't enough to handle circumscription. The difficulty is not hard to understand: Suppose we want \(Q(1) = 1\) and \(Q(2) = 0\) (\(Q\) can be thought of as a predicate by treating 1 as true and 0 as false). Huet's unification, given the problem \(Q(1) = 1\), is only going to produce \(\lambda x.1\) and \(\lambda x.x\) as possible values of \(Q\), neither one of which satisfies \(Q(2) = 0\). Moreover, it is unpleasant that Huet's notion of unification does not have "most general unifiers". These difficulties are related; when the right notion of unification is defined, there are "more" unifiers, and there is also a "minimal" or "most general" unifier, of which all the others are "extensions".

This paper contains the definition of this new unification algorithm, and the proof that it always produces a most general unifier.

\(^{1}\) Research partially supported by National Science Foundation grant number CCR-9528913.
2 Related Work

This unification algorithm can be used to extend to second-order logic the "backwards Gentzen" theorem-prover described for first-order logic in [Beeson 1991].

As far as I know, the new unification algorithm presented here makes possible the first direct implementation of circumscription, i.e. using second-order logic directly in proof search. Lifschitz has given a method of reducing some cases of circumscription to first-order logic, where it seems certain that existing theorem-provers could handle the transformed problem, although the experiment hasn't actually been carried out ([Lifschitz 1985]). A theorem-prover described in [Baker-Ginsberg 1989] and [Ginsberg 1989] is related to circumscription via Lifschitz's result, but does not involve circumscription in its mechanism. Experiments with this algorithm in automatic deduction by circumscription will be described elsewhere.

3 $\lambda$-Calculus and Definition by Cases

We are interested in applications of unification to various logical systems, including second-order logic and various type theories. All of these logics can be expressed as subsystems of $\lambda$-calculus, so it is natural to study unification in the general setting of $\lambda$-calculus. However, the definition of unification that we introduce requires an if-then-else operator, which does not exist in pure untyped lambda calculus, although the required operator can be defined in various typed calculi. In order to prove our theorems once instead of many times, we extend the usual $\lambda$-calculus to include the minimally required rules for definition by cases. We call the extended system $\lambda D$. It is formed by adding to the ordinary $\lambda$-calculus a new constant $d$ for definition by cases. The expression $d(x, a, a, b)$ means "if $x = y$ then $a$ else $b$". The system and corresponding reduction relation are specified by the following reduction rules. In these rules, equality means "reduces to".

\[
\begin{align*}
d(x, x, a, b) &= a \\
d(x, y, Zy, Zx) &= Zx \\
d(x, y, y, x) &= x \\
d(x, y, a, a) &= a
\end{align*}
\]

Taking $Z = \text{Ax.x}$ and $Z = \text{Ax.a}$, the last two rules follow as equalities from the second, but as reduction rules instead of just equalities they are not superfluous.

A natural question is whether $\lambda D$ satisfies the Church-Rosser theorem. J. W. Klop showed that it does not (private correspondence). Equality of terms in $\lambda D$ is defined by the transitive closure of the relation "a and b have a common reduct". The following theorem shows that in spite of the failure of Church-Rosser, $\lambda D$ is consistent and indeed is a conservative extension of $\lambda\eta$, for equations between closed terms.