Termination of Combined
(Rewrite and λ-Calculus) Systems

Carlos Loria-Saenz* and Joachim Steinbach**

Universität Kaiserslautern
Fachbereich Informatik, Postfach 3049
6750 Kaiserslautern (Germany)

Abstract. We consider the termination problem of combinations of term-rewriting systems and the λ-calculus employing standard methods of the theory of term rewriting systems in contrast to (extensions of) Tait-Girard's technique. In particular for some class of higher-order rule systems, we explicitly construct a well-founded ordering over λ-terms whose combination with the β-reduction is terminating. Then, by embedding the higher-order rewriting relation into this ordering, we can prove termination for combinations of such a class of higher-order term rewriting systems and the λ-calculus.

1 Introduction

Term rewriting systems (TRS) and the λ-calculus are important abstract models for computation and logic and their combination can be used to obtain a general and rich representation of functional programming languages. As an illustration, using first-order rule systems, we can specify usual arithmetic operations such as the addition on natural numbers. With the help of higher-order rules, we can additionally specify the generating function iter1 which allows to define many other functions over lists by only instantiating the variables F and x (like length):

\[
\begin{align*}
x + 0 & \rightarrow x \quad (1) \\
s(x) + y & \rightarrow s(x + y) \quad (2) \\
\text{iter}(F, x, [ ]) & \rightarrow x \quad (3) \\
\text{iter}(F, x, y :: l) & \rightarrow \text{iter}(F, (Fx y), l) \quad (4) \\
\text{length}(l) & \rightarrow \text{iter}(\lambda u. \lambda w. s(u), 0, l) \quad (5)
\end{align*}
\]

In order to use such systems we need to combine the corresponding relations. One way to achieve that, is to take the union of the R-reduction (the application of rules to algebraic or higher-order terms) and the standard λ-reduction (the application to λ-terms). Thus, the termination problem in this case is identical

* Supported by DAAD-ITCR, e-mail: loria@informatik.uni-kl.de
** Supported by SFB 314 (D4-Projekt), e-mail: steinba@informatik.uni-kl.de
1 (F is a higher-order variable, x, y and l are first-order ones, [] represents nil and :: the cons operation)
to the termination proof of the union between a terminating R-reduction and the typed \( \lambda \)-reduction. The results of this paper refer to this approach. Note that the termination of the combination has been proved by means of an extension of the Tait-Girard technique based on computability predicates (see [BG91] and [JO91, DO90]).

In the following, we assume familiarity with all the notations and definitions used in connection with TRSs and the \( \lambda \)-calculus. For simplicity, we consider the denominated simply-typed lambda-calculus. The relation \( \Rightarrow_\beta \) denotes the \( \beta \)-reduction. Note that for the typed \( \lambda \)-calculus, a well-known termination result of \( \Rightarrow_\beta \) holds (see [HS86] for a proof using the mentioned Tait-Girard technique). We also assume that the \( \beta \)-normal forms are given in long-\( \beta \)-normal form. The combination of a rule system \( R \) and the (typed) \( \lambda \)-calculus is represented by the union of the corresponding reduction relations and denoted by \( \Rightarrow_{\beta,R} \). \( \text{FV}(M) \) denotes the set of free variables occurring in the well-typed term \( M \). Usually, we employ the following conventions: \( x, y, z \) are first-order variables, \( s, t, l, r \) are first-order terms. \( X, Y, Z \) are higher-order variables. \( M, N, P \) are well-typed terms, \( \sigma \) is a well-typed substitution.

2 An Ordering for Higher-Order Rewriting and the \( \lambda \)-Calculus

The problem of proving termination of higher-order combinations is more difficult than that of the first-order case: arbitrary combinations of terminating higher-order systems are not terminating in general. What we do in the higher-order case is to explicitly construct a well-founded ordering over \( \lambda \)-terms such that the combination of this ordering with the \( \beta \)-reduction is terminating. Then, to achieve termination, the rewriting relation must be included into this ordering.

2.1 Sketch of an Ordering

To construct an ordering over \( \lambda \)-terms, we extend the usual idea for first-order systems using orderings based on precedences (especially the recursive path ordering RPO of [Der82]). We extend these orderings to higher-order terms. We illustrate the basic ideas of the construction of the ordering by an example. Suppose, we want to compare the terms \( \text{map}(X, y :: L) \) and \( X(y) :: \text{map}(X, L) \). To do that, we proceed as in the first-order-case using an RPO, say \( >_1 \). Assume further that we have a quasi-ordering \( >_{\mathcal{F}} \) over the operators of \( \mathcal{F} \). In order to compare higher-order terms, we apply some term transformations (see [Der82]). The main feature of our technique is the following one: An operator is closely connected with the higher-order operators of its subterms. This way, operators are interpreted as special terms\(^2\). This leads to an extended set of operators denoted by \( \mathcal{F}_0 \). Then, we construct a quasi-ordering \( >_0 \) over \( \mathcal{F}_0 \). Thus, \( >_0 \) (the strict part of \( >_0 \)) is intended to operate as a precedence ordering for our RPO \( >_1 \). For instance, by requiring that \( \text{map} >_{\mathcal{F}} \) holds w.r.t. the given precedence

\(^2\) For example, the extended operator for \( \text{map}(X, y :: L) \) is of the form \( \text{map}(X) \).