A Complete Narrowing Calculus for Higher-Order Functional Logic Programming

Koichi Nakahara,¹ Aart Middeldorp,² Tetsuo Ida²

¹ Canon Inc.
Shimomaruko, Ohta-ku, Tokyo 146, Japan
² Institute of Information Sciences and Electronics
and
Center for Tsukuba Advanced Research Alliance
University of Tsukuba, Tsukuba 305, Japan

Abstract. Using higher-order functions is standard practice in functional programming, but most functional logic programming languages that have been described in the literature lack this feature. The natural way to deal with higher-order functions in the framework of (first-order) term rewriting is through so-called applicative term rewriting systems. In this paper we argue that existing calculi for lazy narrowing either do not apply to applicative systems or handle applicative terms very inefficiently. We propose a new lazy narrowing calculus for applicative term rewriting systems and prove its completeness.

1 Introduction

There is a growing interest in combining the functional and logic programming paradigms in a single language, see Hanus [6] for a recent overview of the field. The underlying computational mechanism of most of these integrated languages is (conditional) narrowing. Examples of such languages include BABEL [9] and K-LEAF [4]. Both BABEL and K-LEAF lack higher-order features. Bosco and Giovannetti [2] extended K-LEAF to the higher-order language IDEAL. The semantics of IDEAL is given by means a translation from IDEAL programs into K-LEAF programs. González-Moreno et al. proposed in [5] the language SFL, a higher-order extension of BABEL. The higher-order aspects of these two languages are modeled by means of first-order applicative (conditional) constructor-based term rewriting systems. This means in particular that higher-order unification—like in the higher-order logic programming language λ-PROLOG [11]—is avoided because there are no λ-abstractions around. The following example program is taken from [5]:

\[
\begin{align*}
\text{plus 0} y &= y \\
\text{plus} (Sx) y &= S(\text{plus} x y) \\
\text{double} x &= \text{plus} x x
\end{align*}
\]

\[
\begin{align*}
\text{map} f [] &= [] \\
\text{map} f [x | y] &= [f x | \text{map} f y] \\
\text{compose} f g x &= f(g x)
\end{align*}
\]

* Most of the work reported in this paper was carried out while the first author was at the University of Tsukuba, Doctoral Program of Engineering.
The functions \texttt{map} and \texttt{compose} are higher-order. Solving the goal
\[
\text{map } f \ [S0, 0, S0] = [S(S(S0)), S0, S(S(S0))] \]
means finding a substitution for the higher-order variable \( f \) such that the value of the left-hand side of the goal equals the right-hand side. One easily verifies that
\[
f \rightarrow \text{compose } S \text{ double} \]
is a solution to the goal, but actually computing such solutions is a different matter. The operational semantics of SFL is a particular kind of conditional narrowing and González-Moreno \textit{et al.} [5] prove its soundness and completeness with respect to a declarative semantics that is based on applicative algebras over Scott domains.

In this paper we are concerned with lazy narrowing strategies for applicative term rewriting systems. Most lazy narrowing strategies that have been proposed in the literature are defined for constructor-based term rewriting systems, e.g. [1, 10, 14]. An easy but important observation is that while every applicative term rewriting system is a particular kind of term rewriting system, not every applicative constructor-based term rewriting system is a constructor-based term rewriting system. Nevertheless, an applicative orthogonal (constructor-based) term rewriting system is an orthogonal term rewriting system, so lazy narrowing strategies that are defined and proved complete for the latter class can be used as a computation model for higher-order functional logic programming.

We analyze the behaviour of \texttt{OINC}—a simple calculus proposed in [7] which realizes lazy narrowing—for applicative orthogonal term rewriting systems. It turns out that \texttt{OINC} handles applicative terms very inefficiently. We transform \texttt{OINC} into a calculus \texttt{NCA} that deals with applicative terms in an efficient way and we prove the completeness of \texttt{NCA}. We would like to stress that the ideas developed in this paper do not depend on \texttt{OINC}. The only reason for choosing \texttt{OINC} is the simplicity of its inference rules.

This paper is organized as follows. In the next section we introduce applicative term rewriting. In Sect. 3 we recall the calculus \texttt{OINC}. In Sect. 4 we observe that \texttt{OINC} doesn't manipulate applicative term rewriting systems in a very efficient way. The new calculus \texttt{NCA} is defined to overcome this inefficiency. The completeness of \texttt{NCA} is proved in Sect. 5. Section 6 is concerned with a further optimization of our calculus, namely we extend \texttt{NCA} with special inference rules for dealing with strict equality in an efficient way. In Sect. 7 we compare the relative efficiency of \texttt{NCA} and \texttt{OINC} on a small example. We conclude in Sect. 8 with suggestions for future research.

\section{Preliminaries}

We assume the reader is familiar with the basics of term rewriting. (See [3] and [8] for extensive surveys.) In this preliminary section we recall only some less common definitions and we introduce the notion of applicative term rewriting.