Higher-Order Rewriting: Framework, Confluence and Termination

Jean-Pierre Jouannaud*

LIX/CNRS UMR 7161 & École Polytechnique,
F-91400 Palaiseau
http://www.lix.polytechnique.fr/Labo/Jean-Pierre.Jouannaud/

1 Introduction

Equations are ubiquitous in mathematics and in computer science as well. This first sentence of a survey on first-order rewriting borrowed again and again characterizes best the fundamental reason why rewriting, as a technology for processing equations, is so important in our discipline [10]. Here, we consider *higher-order rewriting*, that is, rewriting higher-order functional expressions at higher-types. Higher-order rewriting is a useful generalization of first-order rewriting: by rewriting higher-order functional expressions, one can process abstract syntax as done for example in program verification with the prover Isabelle [27]; by rewriting expressions at higher-types, one can implement complex recursion schemas in proof assistants like Coq [12].

In our view, the role of higher-order rewriting is to design a type-theoretic framework in which computation and deduction are integrated by means of higher-order rewrite rules, while preserving decidability of typing and coherence of the underlying logic. The latter itself reduces to type preservation, confluence and strong normalization.

It is important to understand why there have been very different proposals for higher-order rewriting, starting with Klop's *combinatory reduction systems* in 1980, Nipkow's *higher-order rewriting* in 1991 and Jouannaud and Okada's *executable higher-order algebraic specifications* in 1991 as well: these three approaches tackle the same problem, in different contexts, with different goals requiring different assumptions.

Jan Willem Klop was mostly interested in generalizing the theory of lambda calculus, and more precisely the confluence and finite developments theorems. Klop does not assume any type structure. As a consequence, the most primitive operation of rewriting, searching for a redex, is already a problem. Because he wanted to encode pure lambda calculus and other calculi as combinatory reduction systems, he could not stick to a pure syntactic search based on first-order pattern matching. He therefore chose to search via finite developments, the only way to base a finite search on beta-reduction in the absence of typing

* Project LogiCal, Pôle Commun de Recherche en Informatique du Plateau de Saclay, CNRS, École Polytechnique, INRIA, Université Paris-Sud.

A. Middeldorp et al. (Eds.): Processes... (Klop Festschrift), LNCS 3838, pp. 224–250, 2005.

assumptions. And because of his interest in simulating pure lambda calculi, he had no real need for termination, hence concentrated on confluence results. Therefore, his theory in its various incarnations is strongly influenced by the theory of residuals initially developed for the pure lambda calculus.

Nipkow was mainly interested in investigating the meta-theory of Isabelle and in proving properties of functional programs by rewriting, goals which are actually rather close to the previous one. Functional programs are typed lambda terms, hence he needed a typed structure. He chose searching via higher-order pattern matching, because plain pattern matching is too poor for expressing interesting transformations over programs with finitely many rules. This choice of higher-order pattern matching instead of finite developments is of course very natural in a typed framework. He assumes termination for which proof methods were still lacking at that time. His (local) confluence results rely on the computation of higher-order critical pairs -because higher-order pattern matching is used for searching redexes- via higher-order unification. Nipkow restricted lefthand sides of rewrite rules to be patterns in the sense of Miller [25]. The main reason for this restriction is that higher-order pattern matching and unification are tractable in this case which, by chance, fits well with most intended applications.

Jouannaud and Okada were aiming at developping a theory of typed rewrite rules that would generalize the notion of recursor in the calculus of inductive constructions, itself generalizing Gödel's system T. This explains their use of plain pattern matching for searching a redex: there is no reason for a more sophisticated search with recursors. In this context, the need for strongly terminating calculi has two origins: ensuring consistency of the underlying logic, that is, the absence of a proof for falsity on the one hand, and decidability of the type system in the presence of dependent types on the other hand. Confluence is needed as well, of course, as is type preservation. The latter is easy to ensure while the former is based on the computation of first-order critical pairs -because first-order pattern matching is used for searching redexes. This explains the emphasis on termination criteria in this work and its subsequent developments.

Our goal in this paper is to present a unified framework borrowed from Jouannaud, Rubio and van Raamsdonk [22] for the most part, in which redexes can be searched for by using either plain or higher-order rewriting, confluence can be proved by computing plain or higher-order critical pairs, and termination can be proved by using the higher-order recursive path ordering of Jouannaud and Rubio [19].

We first present examples showing the need for both search mechanisms based on plain and higher-order pattern matching on the one hand, and for a rich type structure on the other hand. These examples show the need for rules of higher type, therefore contradicting a common belief that application makes rules of higher type unnecessary. They also recall that Klop's idea of variables with arities is very handy. Then, we present our framework in more detail, before we address confluence issues, and finally termination criteria. Missing notations and terminology used in rewriting or type theory can be found in [10, 2].