Infinitary Rewriting: From Syntax to Semantics*

Richard Kennaway[1], Paula Severi[2], Ronan Sleep[1], and Fer-Jan de Vries[2]

[1] School of Computing Sciences, University of East Anglia, U.K.
[2] Department of Computer Science, University of Leicester, U.K.

1 Introduction

Rewriting is the repeated transformation of a structured object according to a set of rules. This simple concept has turned out to have a rich variety of elaborations, giving rise to many different theoretical frameworks for reasoning about computation. Aside from its theoretical importance, rewriting has also been a significant influence on the design and implementation of real programming languages, most notably the functional and logic programming families of languages. For a theoretical perspective on the place of rewriting in Computer Science, see for example [14]. For a programming language perspective, see for example [16].

Much of the interest in rewriting paradigms for programming arises from the possibility of a dual reading of a rewrite rule. On the one hand, a rule can be read as a syntactic transformation on a structure. On the other hand, a rule can be read as an equation. For example, the rule:

$$\mathit{fibs} = f(0,1) \quad \text{where} \quad f(m,n) = \mathit{Cons}(m, f(n, m+n))$$

can be read either as an equational definition of a structure which is the infinite list of Fibonacci numbers, or alternatively as instructions for a rewriting machine to construct increasingly better approximations to this infinite list. No real machine can compute the whole of an infinite structure, but by defining suitable finite selectors, we can write programs for rewriting machines which define finite structures in terms of infinite ones. Thus giving the command:

$$\mathit{print}(\mathit{nth}(15, \mathit{fibs}))$$

to a suitable rewriting machine will result in the printing of the 15th Fibonacci number. The rewriting machine has to be careful about how it uses the definitions if it is to achieve a result. From a purely rewriting perspective, the problem is to find a sequence of reductions which is normalising, for example the famous normal order reduction for the lambda calculus [4]. Solutions to this problem are the basis of lazy functional languages. Using implementations of such languages, it is possible to program by devising a suitable set of equations over infinite data structures which can be read as syntactic rewrite rules which deliver an effective means of computing the solution.

* Dedicated in friendship to Jan Willem Klop on the occasion of his 60th birthday.

A. Middeldorp et al. (Eds.): Processes... (Klop Festschrift), LNCS 3838, pp. 148–172, 2005.

However, it is rather easy to write down things that look as if they have both equational and rewrite interpretations, but which do not do what one might expect. Here is an example:

$$primesthenfibs = append(primes, fibs)$$

where *append* appends one list to another, and *primes* and *fibs* are the infinite lists of prime and Fibonacci numbers. One can write this program in a lazy functional language such as Haskell, but the result is just the infinite list of primes — the Fibonacci numbers disappear. The problem here is that the first list does not have an end to attach the second list to, so the *append* function seeks forever.

It is clear that some styles of building infinite terms can be computationally useful, whilst others are not. This raises an interesting question for the underlying theory of term rewriting, which is: what happens to various standard results for term rewriting if we allow infinite terms and infinite rewriting sequences, and what should those infinitary concepts be? Do the standard confluence and related results still hold for orthogonal infinitary systems?

We give an account of a theory of infinitary rewriting, beginning with the initial work done with and inspired by Jan Willem Klop, and ending with some recent work on lambda calculus which derives model theoretic notions from the kind of infinite terms which obstruct some traditional theorems of finitary rewriting.

2 Infinite Term Rewriting Systems

In this section we will introduce the basic concepts of infinite term and reduction sequence of transfinite length. We introduce the notion of a strongly convergent reduction sequences for the more general setting of abstract reduction systems. Then we will describe some of the basic theorems that hold for infinite extensions of term rewriting systems. Detailed proofs can be found in [7, 9, 10].

2.1 Infinite Terms

By interpreting finite terms as trees, infinite terms can be defined as trees having infinite branches as in Figure 1. There is a decision to be made about whether an infinite path in such a tree may be allowed to have a symbol at its end, which could then have further descendants, allowing paths from the root of a term to its leaves to have any ordinal length. We have taken the view that such terms have no computational meaning. Although we might imagine the limit of a reduction sequence $A \to B(A) \to B(B(A)) \to \ldots$ to be the term $B(B(B(...(A))))$ with infinitely many occurrences of B, there is no corresponding infinite process by which the symbol at the end of such a branch might be brought back up to the root.

We shall also require trees to be finitely branching, that is, that every operator symbol have finite arity. It is not clear whether allowing infinite arities