Editorial – Term Rewriting

A term rewriting system consists of a set of rules which defines a *rewriting* relation between terms. Usually, the rules represent directed equality: we rewrite when we can replace equals by equals. This paradigm is very similar to functional programming; however, in general, rewriting is non-deterministic in the sense that no restrictions are placed on the selection of rules to be applied or on the selection of the redex. Moreover, there is no restriction on overlapping rules.

This generality makes rewriting a very powerful computational paradigm: current uses of rewriting include solving word problems in universal algebra, automatic theorem proving of equational and inductive theorems, generation of solutions to equations (narrowing), prototyping of equational specifications, synthesis of rewrite rules (these may be regarded as programs, or implementations of more abstract specifications), proving properties of equational specifications such as sufficient completeness and consistency (together these two properties are called persistency), and a framework for logic and functional programming.

The papers presented fall into three categories: the crucial issues in term rewriting, implementations of term rewriting systems, and term rewriting based theorem proving.

Confluence, termination, and typing are three crucial issues in term rewriting. The confluence property ensures that the order of rule application is irrelevant, whereas the termination property ensures that reduction sequences are well-founded. A set of rules which is confluent and terminating is called complete, and makes equality decidable, since repeated application of the rules reduces any expression to a unique normal form; two terms are equal if and only if they have the same normal form. The first paper is by the author of the ERIL term rewriting laboratory, Jeremy Dick. He gives a gentle and well-motivated introduction to term rewriting and the Knuth–Bendix completion algorithm, which for a given termination ordering tests for the confluence property. The algorithm not only tests for confluence, but may also be regarded as a semi-algorithm which generates a confluent set of rules. The algorithm is called a 'completion' algorithm because if it terminates it generates a confluent and terminating set of rewrite rules. The Knuth–Bendix algorithm is only one of a family of completion algorithms, and the paper concludes with an overview of some of the other completion algorithms.

The second paper, by Phil Watson and Jeremy Dick, investigates how to incorporate type systems in equational reasoning. The paper presents us with a series of problems and solutions. They begin with the problems of one-sorted rewriting and then consider many-sorted rewriting (first proposed over a decade ago) as an improvement. After pointing out the inadequacies of the many-sorted approach, order-sorted rewriting is proposed. For example, the ERIL system is order-sorted. This approach too has its restrictions: the typing system is too syntactic and static, and we are finally presented with the recent approaches to order-sorted rewriting which use the ideas of *dynamic* or *semantic* sorts.

The third paper is a survey of rewriting techniques and implementations. The authors, Miki Hermann, Claude Kirchner and Helene Kirchner, have extensive experience of both the theoretical and practical issues involved in term rewriting and they have designed the REVEUR rewriting laboratory. They introduce the capabilities and efficiency of term rewriting systems in general, and they conclude with a catalogue of term rewriting laboratories available for general distribution.

The final two papers are concerned with theorem proving. In the first of these, Tobias Nipkow discusses rewriting from a general theorem-proving perspective. He points out that most of the equational logic-based systems surveyed in the previous paper cannot be extended to include new proof procedures. Nipkow adopts the approach taken by the LCF theorem prover (and later Isabelle and HOL), where proof procedures are tactics, and shows how term-rewriting techniques, which can be derived from equational logic, can be implemented by tactics for first-order logic. He concludes with an example taken from the area of hardware design: the verification of a ripple-carry adder.

The final paper by Peter Padawitz extends rewriting, theorem proving and equation solving to the context of Horn Clauses. The results are based on a recent book devoted to the topic by the author. This area will be of great interest to the computer scientist, who often finds Horn Clauses more appropriate than equational logic. Thus, signatures are extended to include predicates (as well as functions) and equations may be regarded as either *conditional* or *unconditional*. Within this context, Padawitz develops conditional reduction, narrowing, lazy narrowing, and inductive proof methods based on reduction and narrowing.

My thanks to all the authors for their contributions.

MUFFY THOMAS