Programming Research Group

AN IMPLEMENTATION OF HIGHER-ORDER REWRITING (EXTENDED ABSTRACT)

D.A. Wolfram

PRG-TR-8-93

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

1 Higher-Order Rewriting

Combinatory Reduction Systems of Aczel [1], and Klop [8] generalize first-order rewriting by allowing bound variables to occur in rewrite rules. When restricted to the simply-typed λ-calculus [2, 4], the definitions of rewriting by Nipkow [10], and Wolfram [15, 16] are of increasing generality.

Combinatory Reduction Systems have rewrite rules formed from closed meta-terms where a meta variable Z occurring in the left side of rewrite rule must be the head of a subterm of the form $Z(x_1, \ldots, x_k)$ where the x_i are pairwise distinct variables.

A similar restriction is made in the higher-order rewriting systems of Nipkow [10] but for a different reason. Here, the left sides of rewrite rules are *patterns*, and the right sides are simply typed terms. Patterns are simply-typed λ-terms in $\beta\overline{\eta}$-normal form [16] except that each occurrence of a free variable is the head of a subterm of the form $f(x_1, \ldots, x_k)$ where the x_i are pairwise distinct bound variables in η-normal form. This restriction ensures that the applicability of such a rewrite rule is decidable because higher-order unification for such patterns is decidable [9]. Moreover it is unitary [9] and has linear space and time complexity [13]. A unification algorithm for higher-order patterns which is based on that by Miller [9] has been implemented in Standard ML by Nipkow [11].

Higher-order rewriting defined by Wolfram [15, 16] is more general again. All terms are in $\beta\overline{\eta}$-normal form: they have the form $\underbrace{\lambda x_1 \cdots x_n.}_{binder}\overbrace{\underbrace{@}_{head}\underbrace{(t_1, \ldots, t_m)}_{arguments}}^{matrix}$ where the binder or arguments can be absent, and @ is either a variable or a constant. The type of a term t is written $\tau(t)$.

The higher-order rewrite relation $\longrightarrow_{[i,s=t,\rho]}$ between two such terms is defined as follows.

Definition 1.1 Let p and s be closed terms of the same type in $\beta\overline{\eta}$-normal form. The *higher-order rewrite relation* $p \longrightarrow_{[i,q=r,\pi]} s$ holds if and only if

- $q = r$ is an equation of the form

$$\lambda x_1 \cdots x_n.t_1 = \lambda x_1 \cdots x_n.t_2$$

- i is an occurrence in $\mathcal{O}(p)$ and $\tau(matrix(p/i))$ is $\tau(t_1)$.
- π is a substitution such that $t_1\pi =_{\beta\overline{\eta}} matrix(p/i)$.
- A $\beta\overline{\eta}$-normal form of $p[i \leftarrow binder(p/i).(t_2\pi)]$ is s.

When the equation is oriented and its left side is a higher-order pattern, and $(p/i) =_{\beta\overline{\eta}} (\lambda x_1 \cdots x_n.t_1)\pi$, the definition above reduces to that discussed above [10]. It is unknown whether this relation is decidable when the order of the left side of an equation is greater than three [5].

However, when the equations in E are directed to form a higher- -order rewriting system, and when this is confluent, it is equivalent to a higher-order equational sequent calculus consisting of ten rules in the following sense [16].

Theorem 1.2 *Let $\mathcal{R} = \{q_i \leftarrow r_i \mid i \in I\}$ be a confluent higher-order rewriting system, $E = \{q_i = r_i \,|\, i \in I\}$, and s and t be any terms which are in $\beta\overline{\eta}$-normal form. Then $E \vdash s = t$ if and only if $s \downarrow_{\mathcal{R}} t$.*

2 The Implementation

Our definition of term rewriting system is introduced as a basis for generalizing the theory of first-order systems to the higher-order case. Among many potential applications [15], using higher-order rewriting systems as a generalization of first-order ones [6, 7, 12] would simplify their treatment of bound variables. Special encodings, and supplementary code for α-conversions and substitutions become unnecessary.

As a first step towards this application, we have implemented a form of higher-order rewriting using ACE, the Abstract Clause Engine [14]. This is a framework for searching trees which are built from *abstract clauses.* These are similar to Horn clauses, except that their syntax is not restricted to first-order terms, and the consistency test is defined by the user. ACE has also been used to implement first-order rewriting, first-order equational unification and equational logic programming, combinatorial searches, and problems involving context-free grammars. There are ten built-in search methods each of which can readily be selected by the user [14].

The form of higher-order rewriting which has been implemented uses Definition 1.1 with its special treatment of binders. The left sides of equations are patterns for simplicity. A modified form of Nipkow's code for unifying higher-order patterns has been used in the consistency test [11].

The main example in ACE of this form of higher-order rewriting is an automation of definitions of the Bird-Meertens Formalism [3]. This facilitates equational proofs about properties of functions, functional program transformations, and functional computations.

Acknowledgements

This work is supported by Christ Church, Oxford, and Oxford University Computing Laboratory.

References

[1] P. Aczel, A general Church-Rosser theorem, Technical Report, University of Manchester, 1978.

[2] P.B. Andrews, *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*, Academic, Orlando, 1986.

[3] R.S. Bird, Lectures on constructive functional programming, in: *Constructive Methods in Computing Science*, NATO Advanced Study Institutes, Series F, **55**, M. Broy, (Ed.), Springer, Berlin, 1989, 151–216.

[4] A. Church, A Formulation of the Simple Theory of Types, *Journal of Symbolic Logic* **5** (1940) 56–68. D.C., 1988, 82–90.

[5] G. Dowek, Third order matching is decidable, *Proceedings of the Seventh Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society, Washington, D.C., 1992, 2–10.

[6] S.J. Garland and J.V. Guttag, An Overview of LP: The Larch Prover, *Proceedings of the Third International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science **355**, Springer, Berlin, 1989, 137–151.

[7] J.A. Goguen and T. Winkler, Introducing OBJ3, Report SRI-CSL-88-9, SRI International, Menlo Park, 1988.

[8] J.W. Klop, Term rewriting systems, Report CS-R9073, Centrum voor Wiskunde en Informatica, Amsterdam, 1990.

[9] D. Miller, A logic programming language with lambda-abstraction, function variables, and simple unification, *Journal of Logic and Computation* **1** (1991) 497–536.

[10] T. Nipkow, Higher-order critical pairs, *Sixth Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society, Washington, D.C., July 1991, 342–349.

[11] T. Nipkow, Functional unification of higher-order patterns, *Seventh Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society, Washington, D.C., 1993. (To appear).

[12] M. O'Donnell, *Equational Logic as a Programming Language*, MIT Press, 1985.

[13] Z. Qian, Linear unification of higher-order patterns, *Proceedings of TAPSOFT '93: Theory and Practice of Software Development*, Lecture Notes in Computer Science **668**, Springer, Berlin, 1993, 391–405.

[14] D.A. Wolfram, ACE: the abstract clause engine, System Summary, *Proceedings of the Tenth International Conference on Automated Deduction*, Kaiserslautern, Federal Republic of Germany, 23–27 July 1990, Lecture Notes in Artificial Intelligence **449**, Springer, Berlin, 1990, 679–680.

[15] D.A. Wolfram, Rewriting and equational unification: the higher--order cases, *Proceedings of the Fourth International Conference on Rewriting Techniques and Applications, RTA-91*, Lecture Notes in Computer Science **488**, Springer, Berlin, April 1991, 25–36.

[16] D.A. Wolfram, *The Clausal Theory of Types*, Cambridge Tracts in Theoretical Computer Science **21**, Cambridge University Press, 1993.