Unfolding of Equational Logic Programs *

M. Alpuente† M. Falaschi ‡ M.J. Ramis† G. Vidal†

Abstract

Equational Logic Programming is a programming paradigm which integrates both Equational and Logic Programming (see [10, 11, 14, 15] for surveys on this area). In this paradigm, an equational logic program can be seen as a Conditional Term Rewriting System (CTRS for short), i.e. a set of conditional equations which are implicitely oriented from left to right, and the operational semantics is usually based on some variant of narrowing [17].

The aim of this work is to develop a transformation technique for equational logic programs based on unfolding. In general, program transformation is a method for deriving correct and efficient programs. Unfolding is a well-known program transformation strategy which was first formulated in the case of equational programs by Burstall and Darlington [4] and later introduced in logic programming by Komorowski [13]. The combined effect of unification with simplification is also achieved in [5, 7, 16] by means of some superposition procedure for program synthesis.

We want to define an unfolding transformation on equational logic programs which preserves the computed answers substitutions (obtained by narrowing) along the synthesising process. To the best of our knowledge this is the first approach to this problem. At first we might think that a generic unfolding, possibly valid for any narrowing strategy, would simply consist of applying narrowing steps to the right-hand side and the condition of the rules to obtain the unfolded program. However, with this simple method we can lose completeness when the program is not completely defined (even if it is a constructor based canonical TRS).

Example 1 *Let $\mathcal{R}$ be the following program,*

$$\mathcal{R} = \{\ f(c(X)) \rightarrow c(f(X))\ \}.$$

The term $c(f(X))$ can only be narrowed to $c(c(f(Y)))$ with substitution $\{X/s(Y)\}$. Then we obtain the following unfolded program:

$$Unf(\mathcal{R}) = \{\ f(c(c(Y))) \rightarrow c(c(f(Y)))\ \}.$$

Therefore, an equational goal of the form $\Leftarrow f(c(a)) = c(f(a))$ is only true w.r.t. the original program.

Moreover, the computed answers substitutions (c.a.s.) are not preserved in the unfolding transformation when we use certain narrowing strategies (even if the program is completely defined). Consider the following example which shows how the c.a.s. obtained by *basic narrowing* [12] are not preserved in the unfolded program.

*This work has been partially supported by CICYT under grant TIC 92-0793-C02-02.

†DSIC, Universidad Politécnica de Valencia, Camino de Vera s/n, Apdo. 22012, 46020 Valencia, Spain.

‡Dipartimento di Elettronica e Informatica, Università di Padova, Via Gradenigo 6/A, 35131 Padova, Italy.

Example 2 *Let $\mathcal{R}$ be the following program*

$$\mathcal{R} = \{ \begin{array}{rcl} X + 0 & \rightarrow & X \\ X + s(Y) & \rightarrow & s(X + Y) \end{array} \}.$$

Then there are two possible basic narrowing steps from the term $s(X + Y)$

$$\begin{array}{ll} s(X + Y) \leadsto_{\{Y/0\}} s(X) & \textit{(using the first rule),} \\ s(X + Y) \leadsto_{\{Y/s(Z)\}} s(s(X + Z)) & \textit{(using the second rule).} \end{array}$$

Therefore we obtain the unfolded program:

$$Unf(\mathcal{R}) = \{ \begin{array}{rcl} X + 0 & \rightarrow & X \\ X + s(0) & \rightarrow & s(X) \\ X + s(s(Z)) & \rightarrow & s(s(X + Z)) \end{array} \}.$$

If we want to solve the goal $\Leftarrow X + s(Y) = Z$ w.r.t. the program $\mathcal{R}$, then there is the successful basic narrowing derivation

$$\Leftarrow X + s(Y) = Z \leadsto_{\{\,\}} \Leftarrow s(X + Y) = Z \leadsto_{\{Z/s(X+Y)\}} \Leftarrow true$$

by applying the second rule to the term $X + s(Y)$ and then unifying syntactically the equation $s(X + Y) = Z$. Therefore $\{Z/s(X + Y)\}$ is a c.a.s. of the initial goal. However, this derivation is not possible w.r.t. the unfolded program $Unf(\mathcal{R})$.

These examples indicate that we have to choose more carefully both the conditions on the program and the narrowing strategy. There exists many variants of narrowing, namely *basic narrowing* [12], *innermost narrowing* [9], *selection narrowing* [2], *lazy narrowing* [14], etc. In order to define an unfolding transformation preserving computed answers substitutions we have to use a narrowing strategy in which we could obtain the narrowing positions for any goal (global information) from the narrowing positions of the right-hand side and the condition of the rules (local information). This equivalence between global and local information is achieved when using an innermost strategy. Therefore we use the innermost conditional narrowing strategy of Fribourg [9] as a basis for the definition of the unfolding transformation. This narrowing strategy is a sound and complete algorithm for completely defined constructor based CTRSs.

There are several applications of unfolding. First it provides a general theoretical basis for program transformation techniques. For example it is possible to define an OR compositional semantics, i.e. a semantics which is compositional with respect to the union of theories, in a way analogous to [3]. This leads to techniques for modular transformation of programs [8]. Another application is in improving the accuracy of program analyses [1]. Iterating unfolding of a program k times induces a kind of depth-k analysis.

References

[1] M. Alpuente, M. Falaschi, N. Manzo. Analyses of Unsatisfiability for Equational Logic Programming To appear in *Journal of Logic Programming*, Elsevier, 1994.

[2] P. Bosco, E. Giovannetti, and C. Moiso. Narrowing vs. SLD-resolution. *Theoretical Computer Science*, 59:3–23, 1988.

[3] A. Bossi, M. Gabbrielli, G.Levi and M.C. Meo. A Compositional Semantics for Logic Programs. *Theoretical Computer Science*, 122(1-2):3–47, 1994.

[4] R.M. Burstall and J. Darlington. A Transformation System for Developing Recursive Programs. *Journal of the ACM*, 24(1):44–67, 1977.

[5] N. Dershowitz. Computing with Rewrite Systems. *Information and Control*, 64(2–3):122–157, 1985.

[6] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 243–320. Elsevier, Amsterdam and The MIT Press, Cambridge, 1990.

[7] N. Dershowitz and U. Reddy. Deductive and Inductive Synthesis of Equational Programs. *Journal of Symbolic Computation*, 15:467–494, 1993.

[8] S. Etalle and M. Gabbrielli. Modular Transformations of CLP Programs. Technical Report CWI, Amsterdam, 1994.

[9] L. Fribourg. Slog: a logic programming language interpreter based on clausal superposition and rewriting. In *Proc. Second IEEE Int'l Symp. on Logic Programming*, pages 172–185. IEEE, 1985.

[10] Michael Hanus. The Integration of Functions into Logic Programming: From Theory to Practice. *Journal of Logic Programming*, 1994.

[11] S. Hölldobler. *Foundations of Equational Logic Programming*, volume 353 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 1989.

[12] J.M. Hullot. Canonical Forms and Unification. In *5th Int'l Conf. on Automated Deduction*, volume 87 of *Lecture Notes in Computer Science*, pages 318–334. Springer-Verlag, Berlin, 1980.

[13] H.J. Komorowski. Partial Evaluation as a Means for Inferencing Data Structures in an Applicative Language: A Theory and Implementation in the Case of Prolog. In *9th ACM Symposium on Principles of Programming Languages (Albuquerque, NM)*, pages 255–267, 1982.

[14] J.J. Moreno and M. Rodriguez-Artalejo. Logic Programming with Functions and Predicates: The language Babel. *Journal of Logic Programming*, 12(3):191–224, 1992.

[15] U.S. Reddy. Narrowing as the Operational Semantics of Functional Languages. In *Proc. Second IEEE Int'l Symp. on Logic Programming*, pages 138–151. IEEE, 1985.

[16] U.S. Reddy. Rewriting Techniques for Program Synthesis. In *Proc. of RTA'89*, volume 355 of *Lecture Notes in Computer Science*, pages 388–403. Springer-Verlag, Berlin, 1989.

[17] J.R. Slagle. Automated Theorem-Proving for Theories with Simplifiers, Commutativity and Associativity. *journal of the ACM*, 21(4):622–642, 1974.