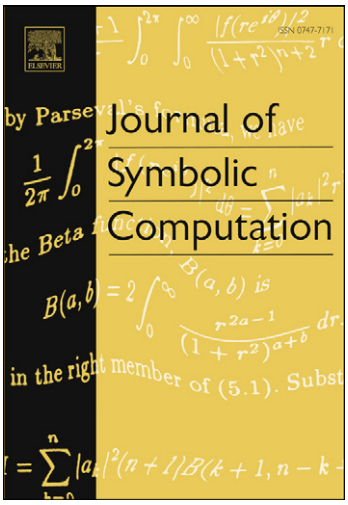

Exploring conditional rewriting logic computations

M. Alpuente, D. Ballis, F. Frechina, J. Sapiña

PII: S0747-7171(14)00096-0
DOI: 10.1016/j.jsc.2014.09.028
Reference: YJSCO 1552

To appear in: *Journal of Symbolic Computation*

Received date: 8 February 2014
Accepted date: 25 June 2014

Please cite this article in press as: Alpuente, M., et al. Exploring conditional rewriting logic computations. *J. Symb. Comput.* (2015), http://dx.doi.org/10.1016/j.jsc.2014.09.028

Exploring Conditional Rewriting Logic Computations ⋆

M. Alpuente [a] D. Ballis [b] F. Frechina [a] J. Sapiña [a]

[a] *DSIC-ELP, Universitat Politècnica de València, Camino de Vera s/n, Apdo 22012, 46071 Valencia, Spain,*

[b] *CLIP-Lab, Technical University of Madrid, E-28660, Boadilla del Monte, Madrid, Spain, and DIMI, Università degli Studi di Udine, Via delle Scienze 206, 33100, Udine, Italy.*

Key Words: rewriting logic, trace exploration, maude, conditional rewrite theories

Abstract

Trace exploration is concerned with techniques that allow computation traces to be dynamically searched for specific contents. Depending on whether the exploration is carried backward or forward, trace exploration techniques allow *provenance tracking* or *impact tracking* to be done. The aim of *provenance tracking* is to show how (parts of) a program output depends on (parts of) its input and to help estimate which input data need to be modified to accomplish a change in the outcome. The aim of *impact tracking* is to identify the scope and potential consequences of changing the program input. Rewriting Logic (RWL) is a logic of change that supplements (an extension of) the equational logic by adding rewrite rules that are used to describe (nondeterministic) transitions between states. In this paper, we present a rich and highly dynamic, parameterized technique for the *forward* inspection of RWL computations that allows the nondeterministic execution of a given *conditional* rewrite theory to be followed up in different ways. With this technique, an analyst can browse, slice, filter, or search the traces as they come to life during the program execution. The navigation of the trace is driven by a user-defined, inspection criterion that specifies the required exploration mode. By selecting different inspection criteria, one can automatically derive a family of practical algorithms such as program steppers and more sophisticated dynamic trace slicers that compute summaries of the computation tree, thereby facilitating the dynamic detection of control and data dependencies across the tree. Our methodology, which is implemented in the Anima graphical tool, allows users to evaluate the effects of a given statement or instruction in isolation, track input change impact, and gain insight into program behavior (or misbehavior).