Julienne: A Trace Slicer for Conditional Rewrite Theories*

María Alpuente[1], Demis Ballis[2], Francisco Frechina[1], and Daniel Romero[1]

[1] DSIC-ELP, Universitat Politècnica de València,
Camino de Vera s/n, Apdo 22012, 46071 Valencia, Spain
{alpuente,ffrechina,dromero}@dsic.upv.es
[2] DIMI, Università degli Studi di Udine,
Via delle Scienze 206, 33100 Udine, Italy
demis.ballis@uniud.it

Abstract. Trace slicing is a transformation technique that reduces the size of execution traces for the purpose of program analysis and debugging. Based on the appropriate use of antecedents, trace slicing tracks back reverse dependences and causality along execution traces and then cuts off irrelevant information that does not influence the data observed from the trace. In this paper, we describe the first slicing tool for conditional rewrite theories that can be used to drastically reduce complex, textually-large system computations w.r.t. a user-defined slicing criterion that selects those data that we want to track back from a given point.

1 Introduction

Software systems commonly generate large and complex execution traces, whose analysis (or even simple inspection) is extremely time-consuming and, in some cases, is not feasible to perform by hand. Trace slicing is a technique that simplifies execution traces by focusing on selected execution aspects, which makes it well suited to program analysis, debugging, and monitoring [6].

Rewriting Logic (RWL) is a very general *logical* and *semantic framework* that is particularly suitable for formalizing highly concurrent, complex systems (e.g., biological systems [5] and Web systems [1,4]). RWL is efficiently implemented in the high-performance system Maude [7]. Rewriting logic-based tools, like the Maude-NPA protocol analyzer, Maude LTLR model checker, and the Java PathExplorer runtime verification tool (just to mention a few [11]), are used in the analysis and verification of programs and protocols wherein the states are represented as algebraic entities that use equational logic and the transitions are represented using conditional rewrite rules. These transitions are performed *modulo* conditional equational theories that may also contain algebraic axioms

* This work has been partially supported by the EU (FEDER) and the Spanish MEC TIN2010-21062-C02-02 project, by Generalitat Valenciana, ref. PROMETEO2011/052. Also, D. Romero is supported by FPI-MEC grant BES-2008-004860 and F. Frechina is supported by FPU-ME grant AP2010-5681.

such as commutativity and associativity. The execution traces produced by such tools are usually very complex and are therefore not amenable to manual inspection. However, not all the information that is in the trace is needed for analyzing a given piece of information in a given state of the trace. For instance, consider the following rules[1] that define (a part of) the standard semantics of a simple imperative language: 1) `crl <while B do I, St> => <skip, St> if <B, St> => false /\ isCommand(I)`, 2) `rl <skip, St> => St`, and 3) `rl <false, St> => false`. Then, in the execution trace `<while false do X := X + 1, {}>` $\to$ `<skip, {}>` $\to$ `{}`, we can observe that the statement `X := X + 1` is not relevant to compute the output `{}`. Therefore, the trace could be simplified by replacing `X := X + 1` with a special variable • and by enforcing the compatibility condition `isCommand(•)`. This condition guarantees the correctness of the simplified trace [3]. In other words, any concretization of the simplified trace (which instantiates the variable • and meets the compatibility condition) is a valid trace that still generates the target data that we are observing (in this case, the output `{}`).

The JULIENNE slicing tool is based on the conditional slicing technique described in [3] that slices an input execution trace with regard to a set of *target symbols* (which occur in a selected state of the trace), by propagating them backwards through the trace so that all pieces of information that are not an antecedent of the target symbols are simply discarded. Unlike standard backward tracing approaches, which are based on a costly, dynamic labeling procedure [2,10], in [3], the relevant data are traced back by means of a less expensive, incremental technique of matching refinement. JULIENNE generalizes and supersedes a previous unconditional slicer mentioned in [2]. The system copes with the extremely rich variety of conditions that occur in Maude theories (i.e., equational conditions $s = t$, matching conditions $p := t$, and rewrite expressions $t \Rightarrow p$) by taking into account the precise way in which Maude mechanizes the conditional rewriting process so that all those rewrite steps are revisited backwards in an instrumented, fine-grained way. In order to formally guarantee the strong correctness of the generated trace slice, the instantiated conditions of the equations and rules are recursively processed, which may imply slicing a number of (originally internal) execution traces, and a Boolean compatibility condition is carried, which ensures the executability of the sliced rewrite steps.

2 The Slicing Tool JULIENNE

The slicing tool JULIENNE is written in Maude and consists of about 170 Maude function definitions (approximately 1K lines of source code). It is a stand-alone application (which can be invoked as a Full Maude trace slicing command or used online through a Java Web service) that correctly handles general rewrite theories that may contain (conditional) rules and equations, built-in operators, and algebraic axioms. JULIENNE also comes with an intuitive Web user interface

[1] We use Maude notation (c)rl to introduce (conditional) rewrite rules.

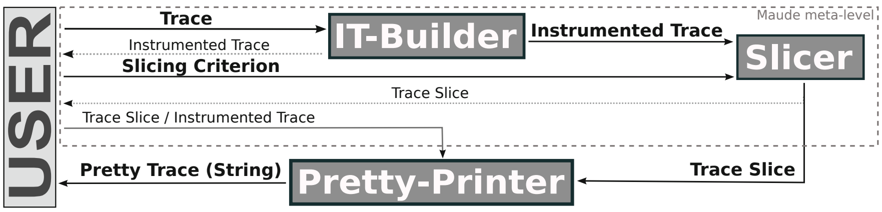

Fig. 1. Julienne architecture

that is based on the AJAX technology, which allows the slicing engine to be used through the WWW. It is publicly available at [9].

The architecture of Julienne, which is depicted in Figure 1, consists of three system modules named **IT-Builder**, **Slicer**, and **Pretty-Printer**.

IT-Builder. The **I**nstrumented **T**race **Builder** module is a pre-processor that provides an expanded instrumented version of the original trace in which all reduction steps are explicitly represented, including equational simplification steps and applications of the *matching modulo* algorithm. Showing all rewrites is not only required to successfully apply our methodology, but it can also be extremely useful for debugging purposes because it allows the user to inspect the equational simplification subcomputations that occur in a given trace.

Slicer. This module implements the trace slicing method of [3] by using Maude reflection and meta-level functionality. Specifically, it defines a new meta-level command called `back-sl` (*backward-slicing*) that takes as input an instrumented trace $t \to^* s$ (given as a Maude term of sort `Trace`) and a slicing criterion that represents the target symbols of the state s to be observed. It then delivers (i) a trace slice in which the data that are not relevant w.r.t. the chosen criterion are replaced by special •-variables and (ii) a compatibility condition that ensures the correctness of the generated trace slice. This module is also endowed with a simple pattern-matching filtering language that helps to select the target symbols in s without the encumbrance of having to refer to them by their addressing positions.

Pretty-Printer. This module implements the command `prettyPrint`, which provides a human-readable, nicely structured view of the generated trace slice where the carried compatibility condition can be displayed or hidden, depending on the interest of the user. Specifically, it delivers a pretty representation of the trace as a term of sort String that is aimed to favor better inspection and debugging activities within the Maude environment.

3 Experimental Evaluation and Conclusion

Julienne is the first slicing tool that can be used to analyze execution traces of RWL-based programs and tools. Julienne greatly reduces the size of the execution traces thus making their analysis feasible even in the case of complex, real-size problems. We have experimentally evaluated our tool in several case studies that are available at the JULIENNE Web site [9] and within the distribution package, which also contains a user guide, the source files of the slicer, and related literature.

We have tested JULIENNE on rather large execution traces, such as the counterexample traces delivered by the Maude LTLR model-checker [8]. We have used JULIENNE to slice execution traces of a real-size Webmail application in order to isolate critical data such as the navigation of a malicious user and the messages exchanged by a specific Web browser with the Webmail server. Typical traces for this application consist of sequences of 100 -1000 states, each of which contains more than 5K characters. In all the experiments, the trace slices that we obtained show impressive reduction rates (up to $\sim 98\%$). Other benchmark programs we have considered include the specification of a fault-tolerant communication protocol, a banking system, and the automated verifier WEB-TLR developed on top of Maude's model-checker itself. In most cases, the delivered trace slices were cleansed enough to be easily inspected by hand. It is very important to note that the slicer does not remove any information that is relevant, independently of the skills of the user.

With regard to the time required to perform the analyses, our implementation is extremely time efficient; the elapsed times are small even for very complex traces and scale linearly. For example, running the slicer for a 20Kb trace w.r.t. a Maude specification with about 150 rules and equations –with AC rewrites– took less than 1 second (480.000 rewrites per second on standard hardware, 2.26GHz Intel Core 2 Duo with 4Gb of RAM memory).

References

1. Alpuente, M., Ballis, D., Espert, J., Romero, D.: Model-Checking Web Applications with WEB-TLR. In: Bouajjani, A., Chin, W.-N. (eds.) ATVA 2010. LNCS, vol. 6252, pp. 341–346. Springer, Heidelberg (2010)
2. Alpuente, M., Ballis, D., Espert, J., Romero, D.: Backward Trace Slicing for Rewriting Logic Theories. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS, vol. 6803, pp. 34–48. Springer, Heidelberg (2011)
3. Alpuente, M., Ballis, D., Frechina, F., Romero, D.: Backward Trace Slicing for Conditional Rewrite Theories. In: Bjørner, N., Voronkov, A. (eds.) LPAR-18 2012. LNCS, vol. 7180, pp. 62–76. Springer, Heidelberg (2012)
4. Alpuente, M., Ballis, D., Romero, D.: Specification and Verification of Web Applications in Rewriting Logic. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 790–805. Springer, Heidelberg (2009)
5. Baggi, M., Ballis, D., Falaschi, M.: Quantitative Pathway Logic for Computational Biology. In: Degano, P., Gorrieri, R. (eds.) CMSB 2009. LNCS, vol. 5688, pp. 68–82. Springer, Heidelberg (2009)
6. Chen, F., Roşu, G.: Parametric Trace Slicing and Monitoring. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 246–261. Springer, Heidelberg (2009)
7. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: Maude Manual (Version 2.6): Tech. rep., SRI, `http://maude.cs.uiuc.edu/maude2-manual/`

8. Clavel, M., Durán, F., Hendrix, J., Lucas, S., Meseguer, J., Ölveczky, P.C.: The Maude Formal Tool Environment. In: Mossakowski, T., Montanari, U., Haveraaen, M. (eds.) CALCO 2007. LNCS, vol. 4624, pp. 173–178. Springer, Heidelberg (2007)
9. The JULIENNE Web site (2012), `http://users.dsic.upv.es/grupos/elp/soft.html`
10. TeReSe (ed.): Term Rewriting Systems. Cambridge University Press, Cambridge, UK (2003)
11. Martí-Oliet, N., Palomino, M., Verdejo, A.: Rewriting logic bibliography by topic: 1990-2011. Journal of Logic and Algebraic Programming (to appear, 2012)