JULIENNE: a Trace Slicer for Conditional Rewrite Theories*

M. Alpuente[1], D. Ballis[2], F. Frechina[1], and D. 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 [8]. Rewriting logic-based tools, like the Maude-NPA protocol analyzer, Maude LTLR model checker, and the inductive theorem prover ITP (just to mention a few [9]), 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 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 target state. For instance, consider the following rules 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

* 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.

execution trace `<while false do X := X + 1, {}>` $\rightarrow$ `<skip, {}>` $\rightarrow$ {}, 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 way we guarantee the correctness of the simplified trace. 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 [2,12], in [3], the relevant data are traced back by means of a less expensive, incremental technique of matching refinement. 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.

Julienne has been written in Maude and is publicly available at `http://users.dsic.upv.es/grupos/elp/soft.html`. Unlike a previous trace slicing facility, which was based on the unconditional slicing technique of [2] and customizedly implemented within the automated verifier Web-TLR [4,1], Julienne 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.

Section 2 provides an overview of the system architecture of Julienne. Section 3 summarizes some experiments that demonstrate the effectiveness of the tool. Section 4 concludes, summarizing and discussing related work. A typical slicing session is given in Appendix A.

2 Julienne system architecture

The slicing engine of Julienne is written in Maude and consists of about 170 Maude function definitions (approximately 1K lines of source code). Julienne also comes with an intuitive Web user interface that is based on the AJAX technology, which allows the slicing engine to be used through a Java Web service. The architecture of Julienne, which is depicted in Figure 1, consists of three system modules named **IT-Builder**, **Slicer**, and **Pretty-Printer**.

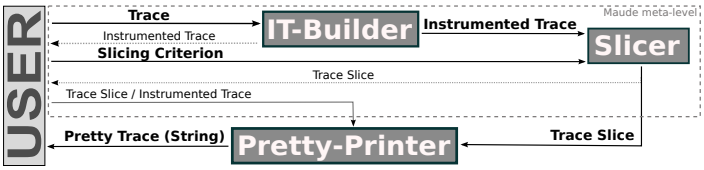

Figure 1. Julienne architecture

IT-Builder. The Instrumented Trace Builder (**IT-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 that are not explicitly recorded in execution traces. Showing all rewrites is not only required to successfully apply our trace slicing methodology in [3], but it can also be extremely useful for debugging purposes because it permits 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 $t \rightarrow^* s$ (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 $\bullet$-variables and (ii) a compatibility condition that ensures the correction 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. Roughly speaking, the slicing criterion is specified by a pattern p that consists of wild cards `?` and `_` that are used to identify (resp. discard) the data of interest (resp. unneeded) inside s. Target symbols in s are then automatically retrieved by pattern matching s within p. For instance, the slicing criterion `<while ? do _, _>` identifies as being relevant only the guard parameter of the `while` statement while the remaining parameters in the configuration are disregarded.

Pretty-Printer. This module implements the meta-level command `prettyPrint`, which provides a human-readable, nicely structured version of the generated trace slice where the carried compatibility condition can be displayed or hidden depending on the interest of the user.

3 Experimental evaluation

We have experimentally evaluated our tool in several case studies that are available at the Julienne Web site [11] and within the distribution package, which also contains a user guide, the source files of the slicer, and related literature.

To properly evaluate the performance and scalability of our slicing tool, we have tested Julienne on rather large execution traces, such as the counterexample traces delivered by the Maude LTLR model-checker [9]. As an illustrative case, 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 in 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).

4 Conclusions

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. JULIENNE generalizes and supersedes a previous unconditional slicer mentioned in [2].

We are not aware of any *trace slicer* that is comparable to JULIENNE for either imperative or declarative languages. To the best of our knowledge, there are just a couple of tools that only slightly relate to ours. The Haskell interactive debugger Hat [7] allows execution traces to be explored backwards, starting from the program output or an error message (computation abort). This is carried out by navigating a graph-like data structure (redex trail) that records dependencies among function calls, whereas our tool does not resort to supplementary data structures. Moreover, even if Hat is able to highlight the top-level "parent" function of any expression in the final trace state, it cannot be used to compose a full trace slice. Actually, at any point of the recreated trail, the function arguments are shown in fully evaluated form (the way they would appear at the end of the computation) even though, at the point at which they are shown, they would not yet have necessarily reached that form. To help debug Maude programs, Maude has a tracing facility that allows the execution sequence to be traced from front to back and also allows the user to select the statements to be applied at each step. Hence, it allows the trace to be shortened by manually focusing on statements, while JULIENNE runs automatically and drastically cuts down the original trace states. It can also be tuned to reveal each single rewrite step that is obtained by applying a conditional equation, equational axiom, or rule, which greatly improves the standard view of execution traces in Maude. A totally different, bytecode trace *compression* was proposed in [10] to help perform Java program analysis (e.g. dynamic *program slicing* [13]) over the compact representation.

References

1. Alpuente, M., Ballis, D., Espert, J., Romero, D.: Model-checking Web Applications with Web-TLR. In: Proc. ATVA 2010.Springer LNCS 6252: 341–346 (2010)
2. Alpuente, M., Ballis, D., Espert, J., Romero, D.: Backward Trace Slicing for Rewriting Logic Theories. In: Proc. CADE 2011. Springer LNCS 6803: 34–48 (2011)
3. Alpuente, M., Ballis, D., Frechina, F., Romero, D.: Backward Trace Slicing for Conditional Rewrite Theories. In: Proc. LPAR-18. Springer LNCS 7180 (2012), to appear
4. Alpuente, M., Ballis, D., Romero, D.: Specification and Verification of Web Applications in Rewriting Logic. In: Proc. FM 2009. Springer LNCS 5850: 790–805 (2009)
5. Baggi, M., Ballis, D., Falaschi, M.: Quantitative Pathway Logic for Computational Biology. In: Proc. CMSB '09. Springer LNCS 5688: 68–82 (2009)
6. Chen, F., Rosu, G.: Parametric trace slicing and monitoring. In: Proc. TACAS 2009. Springer LNCS 5505: 246–261(2009)
7. Chitil, O., Runciman, C., Wallace, M.: Freja, Hat and Hood - a Comparative Evaluation of three Systems for Tracing and Debugging Lazy Functional Programs. In: Proc. IFL 2000. Springer LNCS 2011: 176–193 (2000)
8. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talco, C.: Maude Manual (Version 2.6). Tech. rep., SRI , available at: `http://maude.cs.uiuc.edu/maude2-manual/`
9. Clavel, M., Durán, F., Hendrix, J., Lucas, S., Meseguer, J., Ölveczky, P.C.: The Maude Formal Tool Environment. In: Proc. CALCO. Springer LNCS 4624: 173–178 (2007)
10. JSLICE: a Dynamic Slicing Tool for Java (2008), available at: `http://jslice.sourceforge.net/`
11. The JULIENNE web site (2012), available at: `http://safe-tools.dsic.upv.es/julienne/`
12. TeReSe (ed.): Term Rewriting Systems. Cambridge University Press, Cambridge, UK (2003)
13. Tip, F.: A Survey of Program Slicing Techniques. J. Prog. Lang. 3(3): 121–189 (1995)