Rewriting Logic Techniques for Program Analysis and Optimization *

J. Sapiña

DSIC-ELP, Universitat Politècnica de València, Camino de Vera s/n, Apdo 22012, 46071 Valencia, Spain, jsapina@dsic.upv.es

Abstract. Debugging is the process of locating and fixing errors in computer programs. Debugging is essential in software development and almost every programming language has its own specialized tools for the task, with high variability regarding their debugging power. This paper briefly describes ongoing research towards a Ph.D. thesis on universal debugging, a proposal to develop a program debugging framework that is potentially compatible with any programming language, under the supervision of Professor of Computer Science María Alpuente.

1 Introduction

Debugging is one of the most expensive yet crucial tasks in software development. According to recent research [10], developers spend half their time finding and correcting errors. This means that by 2013, the global cost of debugging is estimated to be $312 billion. Since effective and comprehensive automated debugging is still far away [12], software debugging will still be carried out manually by developers for quite some time with the help of dedicated debugging tools. In order to lessen the debugging costs, in this thesis we focus on improving the debugging task as much as possible by advancing these tools.

The quality of debugging tools is mostly directly related to the popularity of the language to which they serve. Widespread programming languages have at their disposal a large amount of debugging tools, resulting in a fierce tool competition that ultimately results in better debugging facilities. Other not-so-popular languages barely have (quality) tools available. The availability of good debugging tools becomes then one of the determining criteria in the selection of a programming language, to the possible detriment of other languages that best fit the project. In order to unify efforts and raise the quality of debugging tools, we propose the development of a universal debugger based on the tandem matching logic/\(K\) [11], which is a rewriting-based, executable semantic framework in which programming languages, type systems and formal analysis tools can be defined. The main advantage of our approach is to make available almost

* This work has been partially supported by the EU (FEDER) and the Spanish MEC project ref. TIN2010-21062-C02-02, the Spanish MICINN complementary action ref. TIN2009-07495-E, and by Generalitat Valenciana ref. PROMETEO2011/052. J. Sapiña was supported by FPI-UPV grant SP2013-0083.
for free, or at a very low cost, quality debugging tools for any language whose semantics has been defined in the K-Framework.

Plan of the paper. Section 2 gives an overview of the related literature regarding this research. Section 3 briefly describes the proposed universal debugging framework including the hypothesis, objectives, methodology, and some technical aspects. Finally, Section 4 concludes.

2 Related Work

In the past, there have been numerous attempts to develop semantics-based program analysis and debugging techniques. Field et al. proposed in [8] a parametric program slicing technique that relies on semantic specifications together with dynamic dependence tracking and an equational logic designed to serve as a transformation toolkit for the analysis and manipulation of imperative languages [7]. However, the huge gap between theory and practice has been a handicap for the adoption of this technology. In the last decade, Roșu and his team have greatly advanced in the research of semantics-based analysis techniques, thus narrowing the aforementioned gap. This has been achieved with the development of K [14], a rewriting-based executable semantic framework in which programming languages syntax and semantics can be defined. K consists of (i) K rewriting [13], a concurrent rewrite abstract machine that enhances the potential for concurrency of a rewrite theory, (ii) the K technique [13], which transforms K rewriting into an effective framework for the definition of programming languages, and (iii) a specialized notation. A prototype implementation of the K-Framework can be found in the K-Maude tool [16], which successfully translates K semantics definitions into Maude [6], a high-performance reflective specification language and system supporting both rewriting logic (RWL) and purely equational logic. This combination of features, together with the reflective ability of K to manipulate the source code of a program as a piece of data, allows one to execute and transform both the source code and any output results, including execution traces, by standard rewriting. Many languages that are widely distributed throughout the industry such as C, Java, Python, Haskell, and Verilog have a semantics definition written in K. To the best of our knowledge, no extensible RWL-based universal debugging framework that applies to all these languages has been formally developed to date.

3 Universal Debugging

In this section we present the hypothesis, objectives, and methodology to be followed during the thesis development, together with some challenges that need to be addressed in order to achieve our objectives.

3.1 Hypothesis

In the K-Framework, it is possible to virtually execute a program through the real execution of a suitable rewrite system that defines the semantics of
the language in which the program is written. Hence, it is also possible to
create a parametric framework for the analysis and inspection of both, the
virtual and real executions, resulting in a universal debugging framework
potentially compatible with any programming language whose semantics
has been previously formalized in the given scheme.

3.2 Objectives

The objectives of this thesis are as follows:

- Formal development of program transformation and analysis tech-
niques for the inspection and analysis of RWL computations in Maude,
  with particular emphasis on efficiency.
- Application of these techniques to real languages whose semantics are
  formalized in the \( K \)-Framework.
- Development of the first generic, extensible, modular, and language-
  independent, RWL-based debugging environment, which will allow
  the inspection of a program written in any language supported in the
  \( K \)-Framework.
- Enrichment of the debugging environment by integrating different de-
  bugging paradigms (e.g., algorithmic debugging, reverse debugging,
  automated debugging, etc.)
- Empirical evaluation and analysis of the results.

3.3 Methodology and Workplan

This thesis will be carried out according to the terms dictated by the new
doctoral program governed by the RD 99/2011, which points a three-year
limit for this task. Currently, the author is at the end of the first year.
The proposed planning for the whole period is as follows.

Stage One In this first stage, we focus on developing suitable techniques
to identify and extract the information that is relevant to the source pro-
gram being debugged from the execution in \( K \) of the language semantics.
The milestones of this stage are as follows:

- Development of RWL-based trace slicing, dependence analysis, and
  program animation techniques that help locate those pieces of code
  that are relevant to the analysis.
- Implementation in Maude of prototype tools for the developed tech-
niques, with special focus on efficiency.
- Optimization of the developed techniques and tools.

This stage has been recently completed, resulting in the techniques of \[1,3,4\]
and prototype tools iJulienne \[2\] and Anima \[5\].

Stage Two In this stage, which is currently under early development,
we aim to formalize and implement the core of the universal debugger by
pursuing the following specific objectives:

- Theoretical and practical study of the \( K \)-Framework.
- Formal development of a universal debugger in \( K \).
- Instantiation of the debugging framework for different languages.
- Implementation of a prototype universal debugger and experimental
evaluation, focusing on both efficiency and usability.
Stage Three Finally, we focus on the application of the debugging framework and the practical evaluation of the results. The milestones of this third phase are:

– Applications and extensions of universal debugging and transformation (e.g., algorithmic debugging, reverse debugging, automated debugging, dependence analysis, and program and trace slicing).
– Empirical evaluation and analysis of the results.

An international research internship is planned for this final stage. Moreover, scientific dissemination activities such as attendance to workshops and conferences and publication of journal articles will be conducted at every stage.

3.4 Technical Challenges

We have identified several challenges that must be overcome in order to achieve our goal. First, it is essential to perform detailed analysis of generic Maude execution traces, especially with regard to the application of built-in operators and equations, which are silently performed in Maude in order to optimize the performance. In fact, the details of these applications are only available in plain text format and cannot be manipulated as a meta-level expression by Maude. Our efforts so far have led us to develop a technique and corresponding implementation that faithfully reproduces instrumented versions of Maude execution traces without jeopardizing efficiency (for more details, see Section 6 of [4]). We consider this a great advance since we need not transform a given Maude specification in any way to fully capture the application of either built-in operators or equations.

Another challenge is the accurate detection and extraction of the information related to the program being debugged, getting rid of any structural information required to properly execute the generated rewriting system that models the language semantics. We plan to refine our existing trace slicing techniques to focus only on the information related to the program that is being debugged by suitably overlooking this structural information introduced by the K-Framework.

Finally, one major difficulty is the ability to trace back to the original source code any errors that might be found during the analysis of rewriting executions. We plan to formally define the precise relation between the source code of the program being debugged and the Maude terms representing them to be able to automatically translate back and forth the original source code expressions to its Maude term counterpart.

4 Conclusions

This paper summarizes a proposal to establish the basis of universal program debugging by creating the first RWL-based, language-independent, universal debugging framework.

We consider that our proposal is very suitable for the integration of a variety of debugging techniques, regardless of how different they are, such as reverse debugging and algorithmic debugging. Concerning reverse debugging, there exists a number of schemes, each one based on a different
encoding of the machine states. By using RWL, one can provide a uniform representation and easily record the sequence of rules applied, together with their associated information. As for algorithmic debugging, which is based on the idea of an oracle (typically the user) that guides the execution, we think that the program animation techniques we formalized in [4] can support the algorithmic style.

References

10. Cambridge Venture Project. Judge Business School at the University of Cambridge, Available at: http://www.jbs.cam.ac.uk/media/tag/cambridge-venture-project-cvp/