SEMANTICS-BASED WCET ANALYSIS

MIHAIL ASĂVOAE

A THESIS SUBMITTED FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE
UNIVERSITY "ALEXANDRU IOAN CUZA" IASI

2012

List of Publications

Revised publications directly related with the dissertation

1. Mihail Asăvoae, and Dorel Lucanu and Grigore Roşu
   Towards Semantics-Based WCET Analysis
   Proceedings of the 11th International Workshop on Worst-Case Execution-Time Analysis (WCET2011)
   editor Christopher Healy
   Ed. Austrian Computer Society (OCG). (to appear)
   in Oasics Series, Schloss Dagstuhl (to appear)

2. Mihail Asăvoae, and Irina Măriuca Asăvoae:
   Using the Executable Semantics for CFG Extraction and Unfolding
   13th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2011, Timisoara, Romania, 26-29 September 2011,
   editors Dongming Wang, Viorel Negru, Tetsuo Ida, Tudor Jebelean, Dana Petcu, Stephen M. Watt and Daniela Zaharie,
   pages 123–127, IEEE Computer Society, 2011, ISBN 978–1–4673–0207–4.

3. Mihail Asăvoae, Irina Măriuca Asăvoae, and Dorel Lucanu:
   On Abstractions for Timing Analysis in the $\mathbb{K}$ *Framework*
   Second International Workshop on Foundational and Practical Aspects of Resource Analysis, FOPARA 2011, Madrid, Spain, May 2011, Revised selected papers,
   editors Ricardo Peña, Marko van Eekelen, and

Olha Shkaravska,
volume 7177 of Lecture Notes in Computer Science pages 90–107, Springer, 2011,
ISBN 978–3–642–32494–9.

4. Mihail Asăvoae:
   $\mathbb{K}$ *Semantics for Assembly Languages: A Case Study*
   Submitted to The K Workshop 2011

Technical reports and extended abstracts directly related with the dissertation

1. Mihail Asăvoae and Dorel Lucanu
   Formal Executable Semantics for Timing Analysis
   Technical Report TR SIC-08/11, Universidad Complutense de Madrid, Departamento de Sistemas Informáticos y Computación, pages 64–

78

2. Mihail Asăvoae:
   A $\mathbb{K}$-Based Methodology for Modular Design of Embedded Systems
   Extended abstract in pre-proceedings of WADT 2012, 21st International Workshop of Algebraic Development Techniques, June 7-10, Salamanca, Spain
   Technical Report TR–08/12, Universidad Complutense de Madrid, Departamento de Sistemas Informáticos y Computación, editors Narciso Martí-Oliet and Miguel Palomino, pages 16–17

Publications co-related with the dissertation

Revised papers:

1. Irina Măriuca Asăvoae, and Mihail Asăvoae:
   Collecting Semantics under Predicate Abstraction in the $\mathbb{K}$ Framework,

Rewriting Logic and Its Applications - 8th International Workshop, WRLA 2010, Held as a Satellite Event of ETAPS 2010, Paphos, Cyprus, March 20-21, 2010, Revised Selected Papers, editor P.C. Olveczky, volume 6381 of Lecture Notes in Computer Science, pages 123–139, Springer, 2010,
ISBN 978–3–642–16309–8.

2. Irina Măriuca Asăvoae, Mihail Asăvoae, and Dorel Lucanu:
Path Directed Symbolic Execution in the $\mathbb{K}$ Framework
12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2010, Timisoara, Romania, 23-26 September 2010, editors Tetsuo Ida, Viorel Negru, Tudor Jebelean, Dana Petcu, Stephen M. Watt and Daniela Zaharie, pages 133–141, IEEE Computer Society, 2010,
ISBN 978–0–7695–4324–6.

Others:

1. Adrián Riesco, Irina Măriuca Asăvoae, and Mihail Asăvoae:
   A Generic Program Slicing Technique based on Language Definitions
   Extended abstract in pre-proceedings of WADT 2012, 21st International Workshop of Algebraic Development Techniques, June 7-10, Salamanca, Spain
   Technical Report TR–08/12, Universidad Complutense de Madrid Departamento de Sistemas Informáticos y Computación, editors Narciso Martí-Oliet and Miguel Palomino, pages 16–17, pages 91–92

Chapter 1

Introduction

This dissertation presents the design of a definitional semantics-based WCET analyzer, bridging the gap between the principles of formal executable specification, promoted by the $\mathbb{K}$ framework and the existing methods and techniques, which were successfully applied in analysis and verification of embedded software. The standard view on WCET

analysis considers the set of all possible executions of a given program on a specified, underlying hardware. Therefore, there is a projection of design and implementation methods at both the level of the program and the architecture. We present a novel, unified view on this projection, using the $\mathbb{K}$ framework definitional power (i.e. via configuration representation and manipulation). We define a formal executable semantics of a RISC assembly language and a parametric specification for micro-architecture behavior, namely for instruction and data caches. In this way, a program can be formally executed, in concrete, on a family of architectures (i.e. with respect to several design parameters). This combined system is modular, consists of a number of communicating modules corresponding

to hardware and software components. Moreover, this organization allows us to investigate, at the level of individual or group of modules, how to embed, definitionally, various abstractions for timing analysis. We present embeddings for program-related abstractions such as extraction of a control-flow graph, generation of structural integer linear programming constraints, constant propagation and interval analysis. Also, we present the integration of hardware-related abstractions for cache behavior prediction. Our work fully adheres to the principles of design of programming languages and their afferent analysis tools, as advertised by $\mathbb{K}$ framework — formal definition of a programming language should be used unmodified in program analysis and verification.

1.1 Motivation

Ideally, program analysis tools should be based on rigorous semantics of the employed programming languages. Unfortunately, giving a formal semantics (using conventional approaches) to a real language is a non-trivial matter; moreover, even when a semantics is available, it is often not easy to use it for program analysis. Recent research in rewriting logic semantics and in tool development based on such semantics [3, 1] shows encouraging results with respect to both expressiveness and scalability. Moreover, the application of these techniques in the context of real-world low-level languages such as Verilog [2] gives us hope that the theoretically ideal semantics-based approach to program analysis may

be, after all, also practically feasible.

We decide to explore the expressiveness and the scalability issues when we propose the design and implementation of a WCET analyzer. A particularity of this approach is to use a formal executable semantics as the basis for developing analysis and verification methods. Since we follow the $\mathbb{K}$ framework desiderate of separation of concerns, we investigate to what extent (and for what kind of abstract executions) the concrete semantics could be used, as it is, during the analysis process. Informally, this view of building a system over a reliable, trusted kernel corresponds to the generic approach used in theorem provers, for example in the PVS system [4]. The kernel - in our case the formal executable semantics of the assembly lan-

guage + micro-architecture description, is considered trusted with respect to extensive testing. The system is designed with several clear objectives in mind, as presented in the next subsection, and it is stretched when we integrate one of the most successful methodologies in WCET analysis - the ILP + AI approach.

To summarize our goal in one simple sentence: we propose a definitional, rewrite-based WCET analyzer, at both the level of design and the implementation. Before we provide additional insights into our decisions/objectives, we present the two points of view that lead to our solutions.

Our list of sub-objectives include, along with exploiting the executability and modularity of $\mathbb{K}$, parameterization and reusability. We elaborate a

bit on each of these four desiderates and how they could be identified in our definitional WCET analyzer.

Executability

The $\mathbb{K}$ framework provides executability, which allows to test the specification and to get correctness guarantees about it - a $\mathbb{K}$ run of a specification is correct with respect to that particular specification. The formal semantics of the MIPS-based assembly language and the architecture description could be concretely executed and, using test cases, made it trusted.

Modularity

The $\mathbb{K}$ framework, through the rewriting logic, provides also the modularity of the analyzer. Since we basically reason about the execution of a program

on a specified architecture, it is convenient to keep the models of these two components separately. From a structural point of view, $\mathbb{K}$ uses modules to enclose the necessary information to specify a particular system (or concept). From a functional point of view, these $\mathbb{K}$ modules communicate between among each other using specialized terms, which act like messages.

Parameterization

The part of the system that deals with the micro-architecture modeling is designed and implemented to cater for a number of hardware-specific arguments. In our particular case, we refer to a parametric implementation for cache memories behavior (i.e. cache sizes, cache organization, replacement policies etc). This parameterization is present in the

generic hardware simulators, useful mostly to experiment with the execution of programs in various environments. In our case, this parameterization is also useful when we encode cache-specific behavior analyses, and to obtain, in this way, a family of abstract executions.

Reusability

One desiderate of our design is to consider a core specification, deemed as being trusted and to extend it with additional functionality, mostly oriented towards analysis and verification. Then, from this point of view, of the relationship concrete-abstract executions, we argue that the implementation is reusable. A direct consequence is to simplify the process of implementing abstractions, directly over a concrete or another abstract specification of a

system.

1.2 Contributions

Towards achieving our objectives, the key contributions of this thesis are:

1. **Formal Definition of a RISC Assembly language**

   We design and implement, in $\mathbb{K}$ a complete formal definition for a MIPS-like assembly language — the PISA instruction set from the Simplescalar toolset. This is novel with respect to the existing language definitions in rewriting logic and the $\mathbb{K}$ framework. From this definition, we could extract several useful subsets, such as the integer-based subset of the language or the representation for the disassembled exe-

cutables, the latter being the focal point in our further developments.

2. $\mathbb{K}$ **Specifications for Split-Caches**

   We design and implement a modular and parametric $\mathbb{K}$ specification for instruction and data cache memories behavior. The parametric aspects refer to both general caching (i.e. cache size, associativity, replacement policies on read, look-up) as well as particular to family of caches (writing policies in data caches). This is new with respect to the existing architecture specifications in rewrite-based programming environments. The architecture part is designed to accommodate further extensions for timing analysis.

3. **General Methodology for Integration of Ab-**

stractions

We propose a general methodology for integrating abstractions, directly over the concrete specification of the system, consisting of the programming language definition and the micro-architecture modeling. This methodology is presented as a meta-algorithm that is instantiated for each abstraction of interest, by following specific design steps. The consequence is that, during its execution, the abstract steps are interleaved with the concrete steps. In this way, we could define and measure a reusability factor. Instances of this approach - Contribution 4 and Contribution 5 - have been tested on a set of standard benchmarks for WCET estimation. This methodology is new and $\mathbb{K}$ specific, as it

takes advantage of a configuration abstraction mechanism provided by the $\mathbb{K}$ framework.

4. **Control Flow Abstractions in $\mathbb{K}$**

   We design and implement, in $\mathbb{K}$, two standard flow-oriented abstractions for timing analysis: an abstraction for control-flow graph (CFG) extraction and an abstraction for generation structural integer linear programming (ILP) constraints. The first analysis enjoys two variants: one which considers minor modifications of the original semantics and relies on reachability analysis to extract the CFG edges and another which keeps the core language semantics unmodified and interleaves concrete and abstract execution steps. The second analysis is part of the general methodology of ILP + AI for WCET analysis

and extracts a particular kind of structural information from the program. This is new with respect to a general way of defining abstractions, using the general configuration abstraction of the $\mathbb{K}$ framework.

5. **Cache Behavior Abstractions in $\mathbb{K}$**

   We design and implement, in $\mathbb{K}$, a family of standard analyses for instruction cache behavior. These analyses - may, must and persistence, are used to classify the program's instructions with respect to their interaction with the caching system. Their definition is an instance of the previously mentioned meta-algorithm - Contribution 3.

   All these set the base for the first definitional WCET analyzer.

References

[1] T. F. Şerbanuţă and G. Roşu. K-Maude: A rewriting based tool for semantics of programming languages. *WRLA 2010*, volume 6381 of *LNCS*, pages 104–122, 2010.

[2] P. O. Meredith, M. Katelman, J. Meseguer, and G. Roşu. A formal executable semantics of Verilog. *MEMOCODE'10*, pages 179–188. IEEE, 2010.

[3] G. Roşu and T. F. Şerbănuţă. An overview of the K semantic framework. *Journal of Logic and Algebraic Programming*, 79(6):397–434, 2010.

[4] N. Shankar. Rewriting, inference, and proof. *Workshop on Rewriting Logic and Applications*, pages 1–14, 2010.