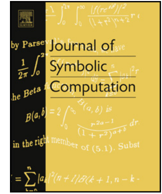
Journal of Symbolic Computation ••• (••••) •••–•••

Contents lists available at ScienceDirect

Journal of Symbolic Computation

www.elsevier.com/locate/jsc



A generic framework for symbolic execution: A coinductive approach

Dorel Lucanu [a], Vlad Rusu [b], Andrei Arusoaie [a,b]

[a] *Faculty of Computer Science, "Alexandru Ioan Cuza" University of Iaşi, Romania*
[b] *Inria Lille Nord Europe, France*

A R T I C L E I N F O

Article history:
Received 27 August 2015
Accepted 6 December 2015
Available online xxxx

Keywords:
Symbolic execution
Programming language
Formal operational semantics
Reachability logic
Circular coinduction
Program verification

A B S T R A C T

We propose a language-independent symbolic execution framework. The approach is parameterised by a language definition, which consists of a signature for the syntax and execution infrastructure of the language, a model interpreting the signature, and rewrite rules for the language's operational semantics. Then, symbolic execution amounts to computing symbolic paths using a *derivative* operation. We prove that the symbolic execution thus defined has the properties naturally expected from it, meaning that the feasible symbolic executions of a program and the concrete executions of the same program mutually simulate each other. We also show how a coinduction-based extension of symbolic execution can be used for the deductive verification of programs. We show how the proposed symbolic-execution approach, and the coinductive verification technique based on it, can be seamlessly implemented in language definition frameworks based on rewriting such as the $\mathbb{K}$ framework. A prototype implementation of our approach has been developed in $\mathbb{K}$. We illustrate it on the symbolic analysis and deductive verification of nontrivial programs.

© 2016 Published by Elsevier Ltd.

1. Introduction

Symbolic execution is a well-known program analysis technique introduced in 1976 by James C. King (1976). Since then, it has proved its usefulness for testing, verifying, and debugging programs.

E-mail addresses: dlucanu@info.uaic.ro (D. Lucanu), vlad.rusu@inria.fr (V. Rusu), andrei.arusoaie@info.uaic.ro (A. Arusoaie).

http://dx.doi.org/10.1016/j.jsc.2016.07.012
0747-7171/© 2016 Published by Elsevier Ltd.

Symbolic execution consists in executing programs with symbolic inputs, instead of concrete ones, and it involves the processing of expressions containing symbolic values (Păsăreanu and Visser, 2009). The main advantage of symbolic execution is that it allows to reason about multiple concrete executions of a program, and its main disadvantage is the state-space explosion determined by decision statements and loops. Recently, the technique has found renewed interest in the formal-methods community due to new algorithmic developments and progress in decision procedures.

In this paper we address two foundational issues regarding the symbolic execution, namely its relationships with the formal definition of the language, for soundness, and with the program logics, for applications to program analysis and verification.

Description of the contribution The main contribution of the paper is a formal, language-independent theory and tool for symbolic execution, based on a language's operational semantics defined by term rewriting.[1] On the theoretical side, we define symbolic execution as the application of rewrite rules in the semantics by *derivation*, a logical description of symbolic successors of a given set of states also symbolically represented as a logical formula in Matching Logic (ML) (Roşu, 2015). We prove that the symbolic execution thus defined has properties ensuring that it is related to concrete program execution in a natural way:

Coverage: to every concrete execution there corresponds a feasible symbolic one;
Precision: to every feasible symbolic execution there corresponds a concrete one;

where two executions are said to be corresponding if they take the same path, and a symbolic execution is feasible if the path conditions along it are satisfiable. Or, stated in terms of simulations: the feasible symbolic executions and the concrete executions of any given program mutually simulate each other.

We also show how a simple extension of our symbolic-execution approach results in a deductive system for proving programs with respect to Reachability Logic (RL) (Ştefănescu et al., 2014) properties; RL is a language-independent program logic also used for defining language semantics, which has been shown to subsume existing language-dependent logics such as Hoare and Separation logics (Roşu and Ştefănescu, 2012a, 2012b). The proposed deductive system is proved to be sound by using a coinductive proof technique. It is shown to be a strict generalization of an approach we presented in Lucanu et al. (2015), in the sense that the procedure for RL proposed there is a strategy of the proof system proposed here. Our 3-rule proof system is also substantially simpler that the original 8-rule proof system given in Ştefănescu et al. (2014); the price to pay is the theoretical relative completeness property, which the original proof system has, whereas ours is not known to have. The proof system we propose is inspired from the circular coinduction proof technique (Roşu and Lucanu, 2009), applied in this paper to programming language definitions (whereas in Roşu and Lucanu, 2009 it is applied to proving observational equalities between possibly infinite data structures, e.g., streams). This was possible by defining an appropriate notion of derivative in the new context and by exploiting the common framework of induction and coinduction based on ground rules (Sangiorgi, 2012). We thus obtain an uniform and rigorous approach for both finite and infinite symbolic executions.

On the practical side, we present an implementation of the theory in a prototype implementation based on in $\mathbb{K}$ (Roşu and Şerbănuţă, 2010), a framework dedicated to defining formal operational semantics of languages. Our current prototype is built on version 3.4 of $\mathbb{K}$ (https://github.com/kframework/k/releases/tag/v3.4) and enhances the previous one based on the language transformation (Arusoaie et al., 2015). $\mathbb{K}$ is based on rewriting, hence, we formally prove that the derivation operation can be correctly implemented by applying certain modified rewrite rules (obtained by automatically transforming the original ones) over ML formulas. This additional intermediary step between abstract theory and implementation is important for ensuring that the resulting prototype tool adequately implements the theory, since the two extreme sides of our approach lie at quite distant

[1] Most existing operational semantics styles (small-step, big-step, reduction with evaluation contexts, ...) have been shown to be representable in this way in Şerbănuţă et al. (2009).

Please cite this article in press as: Lucanu, D., et al. A generic framework for symbolic execution: A coinductive approach. J. Symb. Comput. (2016), http://dx.doi.org/10.1016/j.jsc.2016.07.012

Download English Version:

https://daneshyari.com/en/article/4945993

Download Persian Version:

https://daneshyari.com/article/4945993

Daneshyari.com