A Generic Framework for Symbolic Execution: Theory and Applications

Andrei Arusoaie

- thesis abstract in English -

Introduction

Symbolic execution is one of the most popular techniques used for analyzing programs. It has been used especially for test case generation, but there exist several other applications (e.g. program verification, program debugging, etc.).

The doctoral thesis titled *A generic framework for symbolic execution: theory and applications* presents a generic framework for symbolic execution, where the genericity is given by the fact that this framework is based on formal definitions of programming languages. This is an advantage because symbolic execution is implemented at the level of the language definition and is not based on the syntax or on the compiler of a particular language. In this thesis, the symbolic execution framework is formally presented, using algebraic specifications. This allows proving some important properties of symbolic execution:

- *Coverage*: for each concrete execution there exists a corresponding symbolic execution on the same program (execution) path.
- *Precision*: for each symbolic execution there exists a concrete execution on the same program (execution) path.

These two properties are important because, in the case in which, after analyzing a program, certain results are obtained about symbolic executions, they ensure that the results can be correctly transferred to the concrete executions. Moreover, because of these properties, the symbolic execution framework that we propose can be used for program verification.

Contributions

The major contributions of this doctoral thesis are:

1. A formal framework for symbolic execution and an implementation thereof, based on the operational semantics of programming languages:
    a. on the theoretical side, we formally define programming languages and symbolic execution and then we prove formally the properties of *Coverage* and *Precision*;
    b. on the practical side, we present a prototype of this symbolic framework that is implemented on top of the K framework for language definitions and which is based on language transformations;
2. Applications of symbolic execution in program verification:
    a. program verification based on Hoare logic, where we show how symbolic execution can be used to verify Hoare triples for a given programming language;
    b. program verification based on Reachability Logic, where we present an alternative proof system for Reachability

Logic and an inference rule application strategy that allows to automate the checking of Reachability Logic formulae; for this proof system and for the proposed strategy, we have implemented a prototype and we have shown soundness (based on the soundness of the Reachability Logic proof system) and a form of weak completeness (for proving formulae to be false).

Contents

The thesis is organized in six chapters, each of which contains its own sections.

Chapter 1

This chapter introduces symbolic execution based on existing approaches that use symbolic execution for various types of applications. In presenting these approaches, the basic principles of symbolic execution are emphasized, but also the fact that the approaches are tied to particular programming languages or depend on the compiler. This also motivates the present doctoral thesis, which is to create a language independent framework for symbolic execution, based on the operational semantics of programming languages.

Chapter 2

In this chapter, the notions and basic definitions used in the thesis are introduced. First-order logic (the multi-sorted version), Matching Logic and Reachability Logic are introduced using this notation. The chapter ends with the presentation of the K framework based on an existing language definition (CinK).

Chapter 3

This chapter contains the main contribution of the thesis, which is the formal framework for symbolic execution. It is here that the formal notions of programming language, unification and symbolic execution relation are presented. At the same time, in this chapter we show that unification can be reduced to matching (under certain conditions) and, using this result, we show the properties of *Coverage* and *Precision*. At the end of the chapter we show that symbolic execution can be obtained by language transformation, which is useful from a practical point of view (because the implementation is based on language transformation).

Chapter 4

The applications of symbolic execution presented in this chapter are the verification of Hoare triples and, respectively, the verification of Reachability Logic formulae.

The first section of the chapter introduces Hoare logic and its proof system for a simple language called IMP. Using a transformation of Hoare triples into Reachability Logic formulae, we build the equivalent proof system using Reachability Logic formulae. We then enrich the IMP language with annotations specific to Hoare logic (preconditions, post-conditions, invariants) and we show that, by using symbolic execution for this language, we find proofs in the equivalence deductive system for Reachability Logic formulae corresponding to Hoare triples.

In the second section of this chapter, we propose an alternative proof system for Reachability Logic and an inference rule application strategy. In the proposed proof system, symbolic execution appears as a separate inference rule. The soundness of the proof system and of the proposed strategy is shown using the soundness of the Reachability Logic proof system. At the same time, we show a form of weak completeness that allows proving Reachability Logic formulae to be false.

Chapter 5

This chapter describes the prototypes implemented for symbolic execution and program verification. In the first section, we present the language transformation within the K compiler, so that symbolic execution in the original language is equivalent to concrete execution in the transformed language. In the second section, a series of examples that emphasize the functionalities of the prototype and the fact that it is language independent are presented. The last section of this chapter presents the prototype developed for the verification of programs using Reachability Logic and the proofs of two nontrivial programs created using it.

Conclusions

The last chapter summarizes the contributions of this doctoral thesis and presents directions for future work that involve extending existing prototypes to be used in program verification.