Electronic Communications of the EASST
Volume 70 (2014)

Proceedings of the
14th International Workshop on
Automated Verification of Critical Systems (AVoCS 2014)

Symbol Elimination for Automated Generation of Program Properties

Laura Kovács

2 pages

Guest Editors: Marieke Huisman, Jaco van de Pol
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Symbol Elimination for Automated Generation of Program Properties

Laura Kovács[*]

Chalmers University of Technology

Abstract: Automatic understanding of the intended meaning of computer programs is a very hard problem, requiring intelligence and reasoning. In this talk we describe applications of our symbol elimination methods in automated proram analysis. Symbol elimination uses first-order theorem proving techniques in conjunction with symbolic computation methods, and derives nontrivial program properties, such as loop invariants and loop bounds, in a fully automatic way. Moreover, symbol elimination can be used as an alternative to interpolation for software verification.

Keywords: program analysis, symbolic computation, automated reasoning, interpolation, invariant generation

Extended Abstract

Individuals, organizations, industries, and nations are increasingly depending on software and systems using software. This software is large and complex and integrated in a continuously changing complex environment. New languages, libraries and tools increase productivity of programmers, creating even more software, but the reliability, safety and security of the software that they produce is still low. We are getting used to the fact that computer systems are error-prone and insecure. Software errors cost world economies billions of euros. They may even result in loss of human lives, for example by causing airplane or car crashes, or malfunctioning medical equipment. To improve software and methods of software development one can use a variety of approaches, including automated software verification and static analysis of programs.

The results summarised in this abstract describe how the combination of automated reasoning and symbolic computation methods can be used for automatic program analysis. Program analysis aims to discover program properties preventing programmers from introducing errors while making software changes and can drastically cut the time needed for program development, making thus a crucial step to automated verification.

The common method of all results presented here is the so-called *symbol elimination* method. Although the symbol elimination terminology has been introduced only recently by us, we argue that symbol elimination can be viewed as a general framework for software verification. That is, various techniques used in software verification, such as Gröbner basis computation or quantifier elimination, can be seen as application of symbol elimination to safety verification of programs.

In a nutshell, symbol elimination is based on the following ideas. Suppose we have a program P with a set of variables V. The set V defines the language of P. We extend the language P to

[*] This work was partially supported by Swedish VR grant D0497701 and the Austrian reserach projects FWF S11410-N23 and WWTF ICT C-050.

a richer language P_0 by adding functions and predicates, such as loop counters. After that, we automatically generate a set Π of first-order properties of the program in the extended language P_0, by using techniques from symbolic computation and theorem proving. These properties are valid properties of the program, however they use the extended language P_0. At a last step of symbol elimination we derive from Π program properties in the original language P, thus "eliminating" the symbols in $P_0 \setminus P$.

The work summarized in this abstract is concerned with the algorithmic treatment of symbol elimination for generating computer program properties such as loop invariants, loop iteration bounds, interpolants, and postconditions.

We start by first presenting how symbol elimination is used in symbolic computation for analysing program loops and inferring loop invariants and postconditions. Our work uses algorithmic combinatorics and algebraic techniques, namely solving linear recurrences with constant coefficients, computing algebraic relations among exponential sequences, and eliminating variables from a system of polynomial equations using Gröbner basis computation and quantifier elimination techniques. We also describe applications of symbol elimination in the timing analysis of programs, or, more generally, for analysing the worse-case execution times of programs.

We further extend our work and present how symbol elimination is applied in first-order theorem proving for generating quantified loop invariants and interpolants. Unlike all previously known techniques, our method allows one to generate first-order invariants containing alternations of quantifiers. The method is based on automatic analysis of the so-called update predicates of loops. We observe that many properties of update predicates can be extracted automatically from the loop description and loop properties obtained by other methods such as a simple analysis of counters occurring in the loop, recurrence solving and quantifier elimination over loop variables. The key ingredient of symbol elimination for generating quantified program properties is then first-order saturation theorem proving. After observing that symbol-eliminating inferences extracted from first-order proofs of program properties can be used for automatic invariant generation and that interpolants obtained from proofs seem to be better for predicate abstraction and invariant generation than those obtained by quantifier elimination, we conclude that symbol elimination can be a key concept for applications of program analysis and verification.

Acknowledgements: The results described in this abstract are based on a joint work with Ioan Dragan (TU Vienna), Jens Knoop (TU Vienna), Andrei Voronkov (U. Manchester), and Jakob Zwirchmayr (IRIT).