Static Program Analysis using Symbolic Computation

Mădălina Eraşcu
merascu@risc.jku.at

Abstract. **The task of ensuring correctness of computer programs is especially important in the case of programs that run on safety-critical systems, because the lack of it might lead to huge life and financial losses. We approach the program correctness problem using static program analysis combined with computational logic, computer algebra, and algorithmic combinatorics methods.**

Sample Verification Conditions

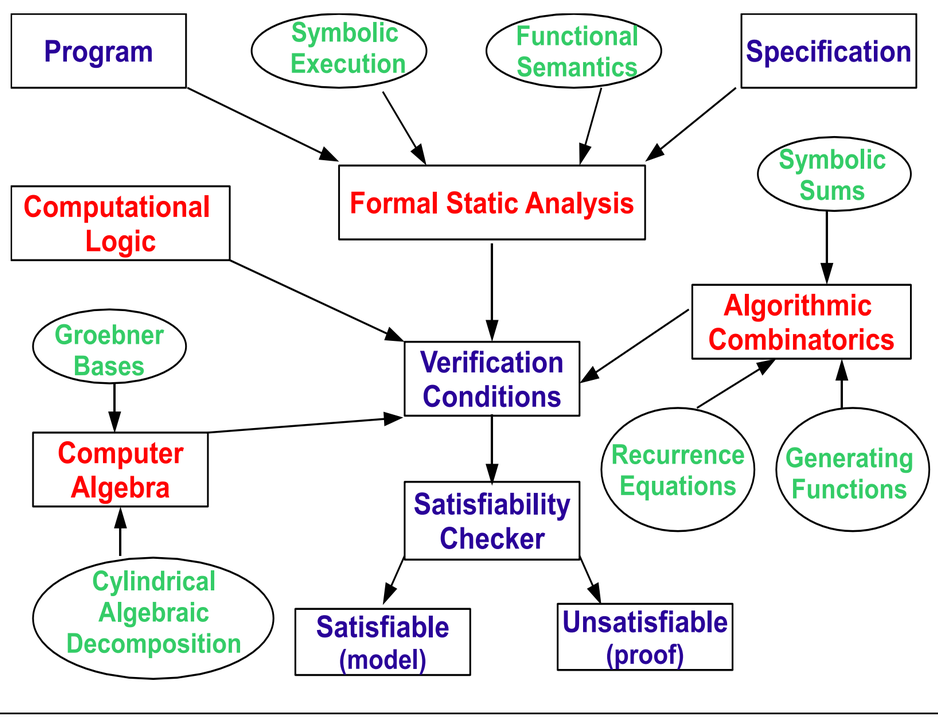
1 Partial Correctness

$$Pre[a,b] \wedge a \neq 0 \wedge b \neq 0 \wedge a > b \wedge a \geq b \wedge Pre[a-b,b] \wedge Post[a-b,b,\beta] \Rightarrow Post[a,b,\beta]$$

2 Termination

$$\Big(\underset{\substack{a,b \\ Pre[a,b]}}{\forall} \bigwedge \begin{cases} a = 0 \Rightarrow \pi[a,b] \\ b = 0 \Rightarrow \pi[a,b] \\ (a \neq 0 \wedge b \neq 0 \wedge a > b \wedge \pi[a-b,b]) \Rightarrow \pi[a,b] \\ (a \neq 0 \wedge b \neq 0 \wedge a \not> b \wedge \pi[a,b-a]) \Rightarrow \pi[a,b] \end{cases} \Big) \Longrightarrow \Big(\underset{\substack{a,b \\ Pre[a,b]}}{\forall} \pi[a,b] \Big)$$

Problem Specification



Challenges

1. **Complexity of the algebraic algorithms.** The algebraic algorithms can be applied to large classes of verification conditions, but their drawback are the high complexity bounds. They have to be adapted to our specific problems and used in combination with logical and other ,,cheap methods" methods.
2. **Quantifiers.** Finding models of satisfiability for formulae containing quantifiers ($\exists$, $\forall$) is hard even for decidable theories (e.g. linear arithmetic over integers – the partial correctness verification condition in our example belongs to this theory, reals, datatypes).
3. **Undecidability of the program termination.** Due to this result there will always be a non-terminating program whose non-termination can not be proven. Therefore we will restrict our analysis to specific classes of programs where logical, algebraic, and combinatorial methods can be used in order to confirm/infirm termination.

Example

$\underline{\texttt{in}}\ a, b$: $\underline{\texttt{integers where}}\ \overbrace{a \geq 0, b \geq 0}^{Pre[a,b]}$

$\underline{\texttt{out}}\ \beta$: $\underline{\texttt{integer where}}$

$$\left.\begin{array}{l} \underset{k_1,k_2}{\exists} (a = k_1 * \beta) \wedge (b = k_2 * \beta) \wedge \\ \underset{g}{\forall}\ \underset{l_1,l_2}{\exists} (a = l_1 * g) \wedge (b = l_2 * g) \Rightarrow \beta \geq g \end{array}\right\} Post[a,b,\beta]$$

$\underline{\texttt{if}}\ (a = 0)$

  $\underline{\texttt{return}}[b]$

$\underline{\texttt{if}}\ (b \neq 0)$

  $\underline{\texttt{if}}\ (a > b)$

    $a := GCD[a-b, b]$,

    $a := GCD[a, b-a]$

$\underline{\texttt{return}}[a]$

Illustrating Example

We consider the example computing the greatest common divisor of two positive integers a and b. This program is correct with respect to its specification, composed by the precondition $Pre[a,b]$ and postcondition $Post[a,b,\beta]$, if the output of the algorithm is correct (partial correctness) and it terminates. Using a program calculus based on forward symbolic execution and functional semantics, we transform the program in first order logic formulae called verification conditions. In this way, the problem of checking the correctness of the program reduces to a satisfiability proof of the verification conditions. In a first phase, computational logic methods are used to approach the proof of the verification conditions, and if the proof does not succeed then advanced computer algebra algorithms (for partial correctness) and algorithmic combinatoric techniques (for termination) will be further applied.

Particularities of our Approach

We aim at the identification of the minimal logical apparatus necessary for formulating and proving (in a computer assisted manner) a correct collection of methods for proving a program calculus. Moreover the interplay between computational logic, computer algebra, and algorithmic combinatorics for proving program correctness was not so much exploited in the program verification community.

The task of performing program proofs is a very difficult one; one must rely on powerful provers and existing knowledge.

Our method for proving the correctness of the program calculus is based only on the underlying theory on which the program operates, on the natural numbers theory, and on first order logic inferences. This small number of ingredients makes the automatic proofs simpler and natural, and it is mainly due to the termination condition, an induction principle developed from the structure of the program with respect to iterative structures (recursive calls and `while` loops). This termination condition ensures the logical existence of the function implemented by the iterative structure. The existence is not automatic, because an iterative structure corresponds from the logical point of view to an implicit definition. Moreover, the termination condition can be also used for proving the uniqueness of the function as well as the total correctness of the iterative structure.

Current Achievements

- Automated proof of correctness of the program calculus in case of single recursion programs and arbitrarily nested, possibly abrupt terminating `while` loops, in the TH∃OREM∀ mathematical assistant.
- Prototype implementation of a verification environment consisting in a verification conditions generator and a satisfiability checker for them.

Recipient of a DOC-fFORTE-fellowship of the Austrian Academy of Sciences at
Research Institute for Symbolic Computation
Johannes Kepler Universität Linz · Altenberger Straße 69 · A-4040 Linz · Austria