VIAP-Automated System for Verifying Integer Assignment Programs with Loops
(Competition Contribution)

Pritom Rajkhowa and Fangzhen Lin

Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong,
{prajkhowa,flin}@cse.ust.hk

Abstract. VIAP(Verifier for Integer Assignment Programs) is an automated system for verification of safety properties of a procedural program with integer assignments and loops. It translates a given program to a set of first-order axioms with quantification over natural numbers, and passes the resulting axioms, together with the given precondition and postcondition, to the SMT solver Z3. The system is fully automatic and points to a new way of proving properties of programs.

1 Introduction

VIAP is a fully automatic program verifier designed for imperative programs proving the correctness of these programs with respect to a specification given as program assertions and assumptions. VIAP first translates the program to a set of first-order axioms with natural number quantifications using algorithm proposed by Lin [1]. It also tries to compute closed-form solutions of inductive definitions in Lin's translation using SymPy [2]. The theorem proving part uses Z3[3] directly when the translated axioms have no inductive definitions. Otherwise, it tries a simple inductive proof using Z3[3] as the base theorem prover. VIAP can prove the (partial) correctness of the programs without needing loop invariants.

To illustrate how our system works, consider the simple program below:

```
int x = 1;
int y = 0;
while (y < 1000) {
       x = x + y;
       y = y + 1;
 }
```

With some simple simplification, the translation outlined in [1] would generate the following axioms:

$$
\begin{aligned}
&x_1 = x_3(N),\\
&y_1 = y_3(N), \quad x_3(0) = 1, \quad x_3(n+1) = x_3(n) + y_3(n),\\
&y_3(0) = 0, \quad y_3(n+1) = y_3(n) + 1,\\
&\neg(y_3(N) \leq 1000),\\
&(n < N) \rightarrow (y_3(n) \leq 1000),
\end{aligned}
$$

where x_1 denotes the output value of the program variable `x` and $x_3(n)$ a temporary function denoting the value of `x` after the nth iteration of the while loop. Similar conventions apply to y and its subscripted versions. The constant N is introduced to denote the number of iterations of the while loop before it exits. The variable n ranges over natural numbers and is universally quantified.

By using sympy our translator computes closed-form solutions for $x_3()$ and $y_3()$, eliminates them, and produces the following axioms:

$$
\begin{aligned}
&x_1 = (N * N - N + 2)/2, \quad y_1 = N,\\
&\neg(N \leq 1000),\\
&(n < N) \rightarrow (n \leq 1000).
\end{aligned}
$$

With these set of axioms, Z3 can then be made to prove the assertion $((N * N - N + 2)/2) >= N$ at the end of the loop by constructing following Z3 query.

```
(1)   x1=Int('x1')
(2)   N=Const('N',IntSort())
(3)   n=Int('n')
(4)   y1=Int('y1')
(5)   s.add(y1 == N)
(6)   s.add(x1 == ((-N + N*N + 2)/(2)))
(7)   s.add(main == 0)
(8)   s.add(N >= 1000)
(9)   s.add(ForAll([n],Implies(And(n < N,n>=0),n < 1000))))
(10)  s.add(Or(N==0,(N-1) < 1000))
(11)  s.add(N>=0)
(12)  s.add(Not((((-N + N*N + 2)/(2))>=N)))
```

Z3 returns with "unsat", thus proves that the axioms entails $((N * N - N + 2)/2) >= N$.

2 VIAP Architecture

VIAP is implemented in python 2. VIAP has been developed in a modular fashion and its architecture is layered in to two parts:

– **Front-End** : The system accepts a program written in C (C99 language) as input and translates it to first order axioms. SymPy(Version 1.1.1) solves the recurrences generated during the translation if closed form solutions are available.
– **Back-End** : Takes the set of translated first order axioms and translates all the axioms to Z3(Version 4.5) compatible equation by pre-processing them using SymPy(Version 1.1.1). Then the proof engine applies different strategies and tries to prove post-conditions in Z3[4]).

2.1 Translation

Our system support the full C99 language (according to the standard ISO/IEC 9899). Our translation is based on the one described in [1]. It translates a program to a set of first-order axioms by introducing for each loop a natural number term that denotes the number of iterations the loop ran before it exits. For a while loop like `while C { B }`, it generates the following two formulas:

$$\neg C(N), \tag{1}$$
$$\forall n.n < N \rightarrow C(n) \tag{2}$$

where N is a new natural number constant denoting the number of iterations that the loop ran before it exits, and $C(n)$ a formula denoting the true value of the condition C at the end of the nth iteration of the loop.

2.2 Proof Strategies

– **Strategy 1(Direct proof):** In this Strategy, system attempts to prove assertions(s) from translated axioms, and assumption(s) by directly SMT solver Z3 [3].
– **Strategy 2(Induction):** In this induction scheme, VIAP attempts to prove any assertion of the form $\beta(N)$ by proving $\forall n.\beta(n)$ holds where n is the loop variable and N is the corresponding loop constant introduce by system. System adopt nested induction scheme to prove assertion(s) which involves two and more loop variables.

3 Strength and Weaknesses

VIAP supports user assertions, including reachability of labels in the C-code. In SV-COMP 2018, these checks are only enabled for the ReachSafety category more specifically ReachSafety-Arrays, ReachSafety-Loops, ReachSafety-Recursive sub-categories. VIAP participated only in Arrays,Loops and Recursive subcategories of ReachSafety. However, VIAP provides little or no support for reasoning about dynamic linked data structures, bit-level precision, or programs with floating points.

Most formal verification tools for non-parallel imperative computer programs rely on having a suitable loop invariant to reason about the correctness of a given

loop. VIAP does not require loop-invariants to verify the correctness of programs. It does not produce a witness for a successful proof that the program satisfies the stated assertions. This is because our system relies almost entirely on Z3 to prove the assertions. It does not attempt to compute or discover loop invariants; instead, it translates programs to a set of inductive equations and first-order constraints about loops. Besides Z3, it only makes use of the algebraic system *SymPy* for simplifying algebraic expressions and solving some simple recurrences. It also sometimes uses a simple mathematical induction on natural numbers. As far as we know, there is no simple solution to the problem of computing a "witness" when an automated theorem prover like Z3 claims that a proof is found.

4 Tool Setup and Configuration

VIAP is participating for the first time. The version of VIAP(version 1.0) submitted to SV-COMP 2018 can be downloaded at:

`https://bitbucket.org/pritom_rajkhowa/verifierintegerassignment/raw/6fbe7baf9be7630eaea05737f121017146361bc3/viap.zip`

VIAP is provided as a set of binaries and libraries for Linux x86-64 architecture. The options for running the tool are:

```
./viap_tool.py  --spec=SPEC INPUT
```

SPEC is the property file, and INPUT is a C file. The output of VIAP is VIAP_OUTPUT_True when the program is safe. When a counterexample is found, it output VIAP_OUTPUT_False and a file named $errorWitness.graphml$ that contains the witness of error-path is generated in the VIAP root folder. If VIAP is unable find any result it output "UNKNOWN"

5 Software Project and Contributors

VIAP is an open-source project, mainly developed by Pritom Rajkhowa and Professor Fangzhen Lin of the Hong Kong University of Science and Technology. We are grateful to the developers of Z3 and SymPy for making their systems available for open use.

References

1. F. Lin, "A formalization of programs in first-order logic with a discrete linear order," *Artificial Intelligence*, vol. 235, pp. 1 – 25, 2016.
2. SymPy Development Team, *SymPy: Python library for symbolic mathematics*, 2016.
3. L.de.Moura and N.Bjorner, "The z3 smt solver." http://research.microsoft.com/en-us/um/redmond/projects/z3/, 2012.
4. P. Rajkhowa and F. Lin, "Viap - automated system for verifying integer assignment programs with loops," in *19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2016, Timisoara, Romania, September 21-24, 2017*, 2017.