FrankenBit: Bit-Precise Verification with Many Bits (Competition Contribution)*

Arie Gurfinkel[1] and Anton Belov[2]

[1] Carnegie Mellon Software Engineering Institute
[2] University College Dublin

Abstract. Bit-precise software verification is an important and difficult problem. While there has been an amazing progress in SAT solving, Satisfiability Modulo Theory of Bit Vectors, and bit-precise Bounded Model Checking, proving bit-precise safety, i.e. synthesizing a safe inductive invariant, remains a challenge. In this paper, we present FrankenBit — a tool that combines bit-precise invariant synthesis with BMC counterexample search. As the name suggests, FrankenBit combines a large variety of existing verification tools and techniques, including LLBMC, UFO, Z3, Boolector, MiniSAT and STP.

1 Verification Approach

FrankenBit combines two orthogonal techniques: one searches for bit-precise counterexamples, and the other synthesizes bit-precise inductive invariants. The counterexample search is done using Bounded Model Checking, and is delegated completely to LLBMC [11]. Invariant synthesis is implemented by first unsoundly approximating programs using Linear Arithmetic (LA), then computing inductive invariants for the approximation, and using those to guide the search for bit-precise invariants. The details of this approach are described in [7].

2 Software Architecture

The architecture of FrankenBit is shown in Fig. 1. First, the input C source is processed and compiled into LLVM [10] bitcode using the UFO front-end (UFO-FE) [1]. This involves normalizing with a custom CIL [12] pass, compiling with `llvm-gcc`, and simplifying using customized optimizations from LLVM version 2.6. The front-end is often sufficient to decide simple verification tasks. Second, two threads are started, one used to synthesize an inductive invariant (left part of Fig. 1), and the other to search for a counterexample (right part of Fig. 1).

* This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution. DM-0000870. The second author is financially supported by SFI PI grant BEACON (09/IN.1/I2618).

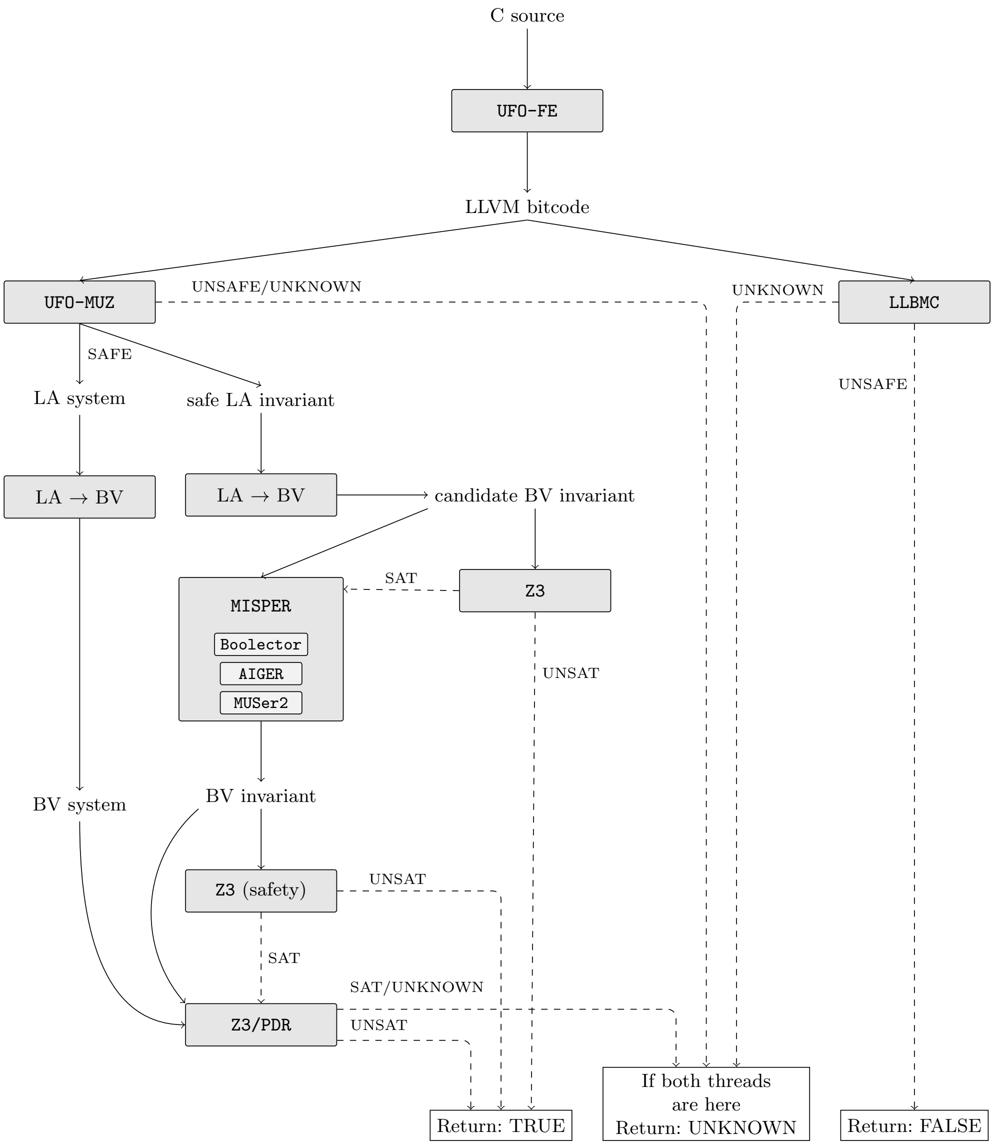

Fig. 1. FrankenBit: Software architecture.

Invariants. Invariants are synthesized using our new algorithm Misper [7]. First, Z3/PDR engine [8] of UFO (UFO-MUZ) abstracts the input over Linear Arithmetic (LA) and synthesizes LA invariant. If this fails, synthesis is aborted. Second, the LA invariant and abstraction are converted to bit-vectors (LA → BV). Third, the candidate bit-vector (BV) invariant is checked using Z3 [4]. If the candidate is not inductive, it is weakened until it becomes inductive using Misper that, in turn, uses Boolector [3] for bit-blasting, aiger for CNF conversion, and MUSer2 [2] for extraction of Minimal Unsatisfiable Subformulas (MUSes). Finally, the safety of the weakened invariant is checked again with Z3 (Z3 safety), and, if necessary, strengthened using the bit-precise version of Z3/PDR.

Counterexamples. The search for counterexamples is delegated to LLBMC [11], that itself uses STP [6], and MiniSAT [5]. In order to run LLBMC on bitcode files produced by UFO-FE, they are first dis-assembled using `llvm-dis` from LLVM v2.9 and then re-assembled using `llmv-as` from LLVM v3.2.

FrankenBit is written in Python and borrows code from Spacer [9].

3 Tool Setup and Configuration

FrankenBit is available for download from `bitbucket.org/arieg/fbit/wiki/svcomp14.wiki`. The options for running the tool are:

```
./bin/fbit.py [-m64] --cex=TRACE --spec=SPEC input
```

where `-m64` turns on 64-bit model, `--cex` and `--spec` are the locations of the counter-example and the specification files, respectively, and `input` is a C file. The result is printed on the output terminal: `TRUE`, `FALSE`, `UNKNOWN`, if the property evaluates, respectively, to true, false, or unknown on the `input`.

FrankenBit is participating in the following categories: `Simple`, `Control Flow and Integer Variables`, and `Device Drivers Linux 64-bit`.

References

1. A. Albarghouthi, A. Gurfinkel, Y. Li, S. Chaki, and M. Chechik. UFO: Verification with Interpolants and Abstract Interpretation - (Competition Contribution). In N. Piterman and S. A. Smolka, editors, *TACAS*, volume 7795 of *Lecture Notes in Computer Science*, pages 637–640. Springer, 2013.
2. A. Belov and J. Marques-Silva. MUSer2: An Efficient MUS Extractor. *JSAT*, 8(1/2), 2012.
3. R. Brummayer and A. Biere. Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In *TACAS*, 2009.
4. L. M. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS*, 2008.
5. N. Eén and N. Sörensson. An Extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.
6. V. Ganesh and D. L. Dill. A Decision Procedure for Bit-Vectors and Arrays. In *CAV*, 2007.
7. A. Gurfinkel, A. Belov, and J. Marques-Silva. Synthesizing Safe Bit-Precise Invariants. In *TACAS*, 2014.
8. K. Hoder and N. Bjørner. Generalized Property Directed Reachability. In *SAT*, 2012.
9. A. Komuravelli, A. Gurfinkel, S. Chaki, and E. M. Clarke. Automatic Abstraction in SMT-Based Unbounded Software Model Checking. In *CAV*, 2013.
10. C. Lattner and V. S. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *CGO*, pages 75–88. IEEE Computer Society, 2004.
11. F. Merz, S. Falke, and C. Sinz. LLBMC: Bounded Model Checking of C and C++ Programs Using a Compiler IR. In *VSTTE*, 2012.
12. G. C. Necula, S. McPeak, S. P. Rahul, and W. Weimer. CIL: Intermediate Language and Tools for Analysis and Transformation of C Programs. In R. N. Horspool, editor, *CC*, volume 2304 of *Lecture Notes in Computer Science*, pages 213–228. Springer, 2002.