HiFrog: SMT-based Function Summarization for Software Verification

Leonardo Alt[1(✉)], Sepideh Asadi[1(✉)], Hana Chockler[2(✉)], Karine Even Mendoza[2(✉)], Grigory Fedyukovich[3(✉)], Antti E.J. Hyvärinen[1(✉)], and Natasha Sharygina[1(✉)]

[1] Università della Svizzera italiana, Lugano, Switzerland
leonardoaltt@gmail.com, antti.hyvarinen@gmail.com, {sepideh.asadi,natasha.sharygina}@usi.ch

[2] King's College London, London, UK
{hana.chockler,karine.even_mendoza}@kcl.ac.uk

[3] University of Washington, Seattle, USA
grigory.fedyukovich@gmail.com

Abstract. Function summarization can be used as a means of incremental verification based on the structure of the program. HiFrog is a fully featured function-summarization-based model checker that uses SMT as the modeling and summarization language. The tool supports three encoding precisions through SMT: uninterpreted functions, linear real arithmetics, and propositional logic. In addition the tool allows optimized traversal of reachability properties, counter-example-guided summary refinement, summary compression, and user-provided summaries. We describe the use of the tool through the description of its architecture and a rich set of features. The description is complemented by an experimental evaluation on the practical impact the different SMT precisions have on model-checking.

1 Introduction

Incremental verification addresses the unique opportunities and challenges that arise when a verification task can be performed in an incremental way, as a sequence of smaller closely related tasks. We present an implementation of the incremental verification of software with assertions that uses the insights obtained from a successful verification of earlier assertions. As a fundamental building block in storing the insights we use function summaries known to provide speed-up through localizing and modularizing verification [12,13].

In this paper we describe the HiFrog verification tool that uses Craig interpolation [6] in the context of Bounded Model Checking (BMC) [4] for constructing function summaries. The novelty of the tool is in the unique way it combines function summaries with the expressiveness of satisfiability modulo theories (SMT). The system currently supports verification based on the quantifier-free theories of linear real arithmetics (QF_LRA) and uninterpreted

This work was supported by the SNF projects 153402 and 163001.

functions (QF_UF), in addition to propositional logic (QF_BOOL). Compared to our earlier propositional tool FunFrog [13], the SMT summaries are smaller and more efficient in verification. They are also often significantly more human-readable, enabling their easier reuse, as well as injection of summaries provided directly by the user. The difference is due to the propositional summaries being based on correctness proofs over circuit-level representation of arithmetic operations. Theory encoding uses instead directly arithmetic symbols in the summaries. In addition, the tool offers a rich set of features such as verification of recursive programs, different ways of optimizing the summaries with respect to both size and strength, efficient heuristics for removing redundant safety properties, and easy-to-understand witnesses of property violations that can be directly mapped to bugs in the source code.

The paper provides an architectural description of the tool, an introduction to its use, and experimental evidence of its performance. The tool together with a comprehensive demo is available at http://verify.inf.usi.ch/hifrog.

Related Work. Incremental verification is extensively researched in domains such as hardware verification, deductive verification, and model checking. Due to space constraints we provide only a brief review of recent related work. The CPAchecker tool is able to migrate predicates across program versions [3]. Deductive verification tools such as Viper and Dafny offer modular verification [11] and caching the intermediate verification results [9] respectively. CBMC is a symbolic bounded model-checker for C that to a limited extent exploits incremental capabilities of a SAT solver[1], but does not use or output any reusable information like function summaries. Similar to HiFrog, ESBMC also shares the CProver infrastructure and is based on an SMT solver. To the best of our knowledge, it does not support incremental verification [5].

2 Tool Overview

HiFrog consists of two main components *SMT encoder* and *interpolating SMT solver*; and the function *summaries* (see Fig. 1). The components are initially configured with the theory and the interpolation algorithms. The tool then processes assertions sequentially using function summaries when possible. The results of a successful assertion verification are stored as interpolated function summaries, and failed verifications trigger a refinement phase or the printing of an error trace. This section details the tool features.

Preprocessing. The source code is parsed and transformed into an intermediate *goto-program* using the goto-cc symbolic compiler. The loops are unwound to the pre-determined number of iterations. HiFrog identifies the set of assertions from the source code, reads the user-defined function summaries (if any) in the smtlib2-format, and makes them available for the subsequent analysis.

[1] http://www.cprover.org.

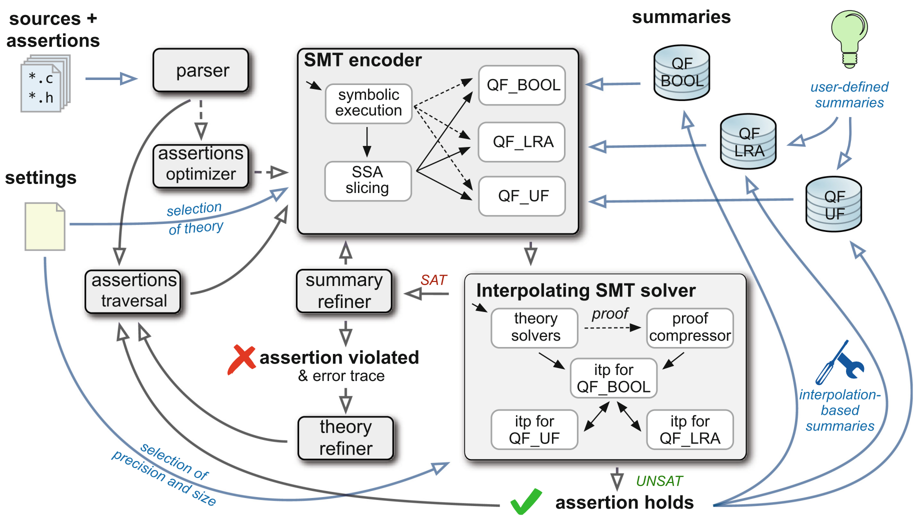

Fig. 1. HiFrog overview. Grey and black arrows connect different modules of the tool (dashed - optional). Blue arrows represent the flow of the input/output data. (Color figure online)

SMT Encoding and Function Summarization. For a given assertion, the goto-program is *symbolically executed* function-per-function resulting in the "modular" Static Single Assignment (SSA) form of the unwound program, i.e., a form where each function has its own isolated SSA-representation. To reduce the size of the SSA form, HiFrog performs *slicing* that keeps only the variables in the SSA form that are syntactically dependent on the variables in the assertion.

When the SSA form is pruned, HiFrog creates the SMT formula in the pre-determined logic (QF_BOOL, QF_UF or QF_LRA). The modularity of the SSA form comes in handy when the function summaries of the chosen logic (either user-defined, interpolation-based, or treated nondeterministically) are available. If this is the case, the call to a function with the available summary is replaced by the summary. The final SMT formula is pushed to an SMT solver to decide its satisfiability.

Due to over-approximating nature of function summaries, the program encoded with the summaries may contain spurious errors. The *summary refiner* identifies and marks summaries directly involved in the detected error, and HiFrog returns to the encoding stage to replace the marked summaries by the precise (up to the pre-determined logic) function representations. Note that due to refinement, HiFrog reveals nested function calls (including recursive ones) which are again replaced by available summaries. For an unsatisfiable SMT formula, HiFrog extracts function summaries using interpolation. The extracted summaries are serialized in a persistent storage so that they are available for other HiFrog runs. For a more detailed description we refer to [13].

Theories. HiFrog supports three different quantifier-free theories in which the program can be modelled: bit-precise QF_BOOL, QF_UF and QF_LRA. The use of theories beyond QF_BOOL allows the system to scale to larger problems since encoding in particular the arithmetic operations using bit-precision can be very expensive. As the precise arithmetics often do not play a role in the correctness of the program, substituting them with linear arithmetics, uninterpreted functions, or even nondeterministic behavior might result in a significant reduction in model-checking time (see Sect. 3). If a property is proved using one of the light-weight theories QF_UF and QF_LRA, the proof holds also for the exact BMC encoding of the program. However, the loss of precision can sometimes produce spurious counterexamples due to the over-approximating encoding. The light-weight theories therefore need to be refined (i.e., using *theory refiner*) to QF_BOOL if the provided counter-example does not correspond to a concrete counterexample.

Obtaining Summaries by Interpolation. HiFrog relies on different interpolation frameworks for the different theories it supports. As a result the generation of propositional, QF_UF and QF_LRA interpolants can be controlled with respect to strength and size by specifying an interpolation algorithm for a theory. For propositional logic we provide the *Labeled Interpolation Systems* [7] including the *Proof-Sensitive* interpolation algorithms [1]. Interpolation for QF_UF is implemented with *duality-based interpolation* [2], and a similar extension is applied to the interpolation algorithm for QF_LRA based on [10]. HiFrog also provides a range of techniques to reduce the size of the generated interpolants through removing redundancies in propositional proofs [12]: the algorithms RecyclePivotsWithIntersection and LowerUnits, structural hashing, and a set of local rewriting rules.

Assertion Optimizer. In addition to incremental verification of a set of assertions, HiFrog supports the basic functionality of classical model checkers to verify all assertions at once. For the cases when the set of assertions is too large, it can be optimized by constructing an *assertion implication relation* and exploiting it to remove redundant assertions [8]. In a nutshell, the assertion optimizer considers pairs of spatially close assertions a_i and a_j and uses the SMT solver to check if a_i conjoined with the code between a_i and a_j implies a_j (if there is any other assertion between a_i and a_j then it is treated as assumption). If the check succeeds then a_j is proven redundant and its verification can be safely skipped.

3 HiFrog Usage

We provide a Linux binary of HiFrog reading as input a C-program, assertions to be verified, a set of parameters and the interpolated or user-defined function summaries in the SMT-LIB2 format. HiFrog exploits the CProver framework and inherits some of its options (e.g., `--unwind` for the loop unrolling, `--show-claims` and `--claim` for managing the assertions checks); the ability for the user to declare and to use a `nondet_TYPE()` function of a specific numerical type (e.g., int, long, double, unsigned, in QF_LRA only) or add a `__CPROVER_assume()` statement to limit the domain to a specific range of values.

HiFrog uses QF_LRA by default but can be switched to QF_UF via the `--logic` option.[2] HiFrog uses a variety of interpolation and proof compression algorithms to control the the precision (with `--itp-uf-algorithm` option for QF_UF, `--itp-lra-algorithm` option for QF_LRA, and `--itp-algorithm` option for propositional interpolation) and the size (with `--reduce-proof`) of summaries. The summary storage is controlled using the `--save-summaries` and `--load-summaries` options. In between verification runs, the summaries contained in the corresponding files for QF_UF and QF_LRA might be edited manually. Note that due to the SMT encoding constraints HiFrog does not allow interchanging summaries between the theories. Finally, HiFrog supports the identification and reporting of redundant assertions with `--claims-opt`, a useful feature for some automatically generated assertions [8].

In the end of each verification run, HiFrog either reports `VERIFICATION SUCCESSFUL` or `VERIFICATION FAILED` accompanied by an error trace. An error trace presents a sequence of steps with a direct reference to the code and the values of variables in these steps. In most cases when QF_UF and QF_LRA introduce a spurious error, HiFrog outputs a warning, and thus the user is advised to use HiFrog with a more precise theory. HiFrog also reports the statistics on the running time and the number of the summary-refinements performed.

Experimental Results. We evaluated HiFrog on a large set of C programs coming from both academic and industrial sources such as SV-COMP. All benchmarks contained multiple assertions to be checked. To demonstrate the advantages of the SMT-based summarization, here we provide data for analysis of benchmarks containing 1086 assertions from which 474 were proven to hold using QF_BOOL (meaning that those properties satisfy the system specifications). Even despite the over-approximating nature of QF_UF and QF_LRA, our experiments witnessed a large amount of properties which were also proven to be correct by employing the light-weight theories of HiFrog (namely, 50.65% and 69.2% of validated properties out of 474 for QF_UF and QF_LRA respectively).

Furthermore, those experiments revealed that model checking using the QF_UF and QF_LRA-based summarization was extremely efficient. Figure 2

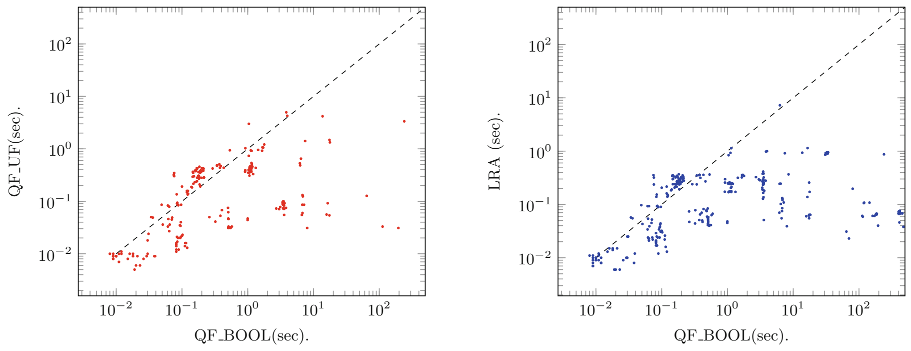

Fig. 2. Running time by QF_BOOL against QF_UF and QF_LRA.

[2] Currently the support for QF_BOOL needs to be specified at compile time.

presents two logarithmic plots for comparison of running times[3] of HiFrog with QF_BOOL to respectively QF_UF and QF_LRA. Each point represents a pair of verification runs of a holding assertion with the two corresponding theories using the interpolation-based summaries. Note that for most of the assertions, the verification with QF_UF and QF_LRA is an order of magnitude faster than the verification with QF_BOOL.

References

1. Alt, L., Fedyukovich, G., Hyvärinen, A.E.J., Sharygina, N.: A proof-sensitive approach for small propositional interpolants. In: Gurfinkel, A., Seshia, S.A. (eds.) VSTTE 2015. LNCS, vol. 9593, pp. 1–18. Springer, Heidelberg (2016). doi:10.1007/978-3-319-29613-5_1
2. Alt, L., Hyvärinen, A.E.J., Sharygina, N.: Duality-based interpolation for quantifier-free equalities and uninterpreted functions (2016). http://www.inf.usi.ch/postdoc/hyvarinen/euf-interpolation.pdf
3. Beyer, D., Löwe, S., Novikov, E., Stahlbauer, A., Wendler, P.: Precision reuse for efficient regression verification. In: ESEC/FSE, pp. 389–399. ACM (2013)
4. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: Cleaveland, W.R. (ed.) TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999). doi:10.1007/3-540-49059-0_14
5. Cordeiro, L.C., de Lima Filho, E.B.: SMT-based context-bounded model checking for embedded systems: challenges and future trends. ACM SIGSOFT Softw. Eng. Notes **41**(3), 1–6 (2016)
6. Craig, W.: Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. J. Symb. Log. **22**(3), 269–285 (1957)
7. D'Silva, V., Kroening, D., Purandare, M., Weissenbacher, G.: Interpolant strength. In: Barthe, G., Hermenegildo, M. (eds.) VMCAI 2010. LNCS, vol. 5944, pp. 129–145. Springer, Heidelberg (2010). doi:10.1007/978-3-642-11319-2_12
8. Fedyukovich, G., D'Iddio, A.C., Hyvärinen, A.E.J., Sharygina, N.: Symbolic detection of assertion dependencies for bounded model checking. In: Egyed, A., Schaefer, I. (eds.) FASE 2015. LNCS, vol. 9033, pp. 186–201. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46675-9_13
9. Leino, K.R.M., Wüstholz, V.: Fine-grained caching of verification results. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 380–397. Springer, Heidelberg (2015). doi:10.1007/978-3-319-21690-4_22
10. McMillan, K.L.: An interpolating theorem prover. Theor. Comput. Sci. **345**(1), 101–121 (2005)
11. Müller, P., Schwerhoff, M., Summers, A.J.: Viper: a verification infrastructure for permission-based reasoning. In: Jobstmann, B., Leino, K.R.M. (eds.) VMCAI 2016. LNCS, vol. 9583, pp. 41–62. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49122-5_2

[3] The timing results were obtained on an Ubuntu 14.04.1 LTS server running two Intel(R) Xeon(R) $E5620$ $CPUs$ @ 2.40 GHz and 16 GB RAM. We prepared a pre-compiled Linux-binary available at the Virtual Machine at http://verify.inf.usi.ch/hifrog/binary; our benchmarks set is available at http://verify.inf.usi.ch/hifrog/bench and can facilitate the property verification for other researchers.

12. Rollini, S.F., Alt, L., Fedyukovich, G., Hyvärinen, A.E.J., Sharygina, N.: PeRIPLO: a framework for producing effective interpolants in SAT-based software verification. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR 2013. LNCS, vol. 8312, pp. 683–693. Springer, Heidelberg (2013). doi:10.1007/978-3-642-45221-5_45
13. Sery, O., Fedyukovich, G., Sharygina, N.: FunFrog: bounded model checking with interpolation-based function summarization. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, vol. 7561, pp. 203–207. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33386-6_17