Interpolation-Based Model Checking for Efficient Incremental Analysis of Software

Grigory Fedyukovich, Antti E. J. Hyvärinen, and Natasha Sharygina
Faculty of Informatics, University of Lugano
Via Guiseppe Buffi 13, CH-6904 Lugano, Switzerland

***Abstract*—Verification based on model checking has recently obtained an important role in certain software engineering tasks, such as developing operating system device drivers. This extended abstract discusses how model checking can be made more efficient by using the structure from program function calls. We use this idea in two orthogonal ways, both of which fundamentally depend on automatically summarizing the relevant behavior of the function calls based on an earlier verification. The first approach assumes a piece of software needs to be verified with respect to a set of properties, whereas the second approach considers a case where an early version of a software has been verified but needs to be re-verified after an upgrade. These techniques have been implemented in tools FunFrog and eVolCheck for verifying C programs. Both of them have been tested on a range of academic and industrial benchmarks, and provide in many cases an order of magnitude speed-up with respect to the baseline. They seem to scale to programs with thousands of lines of code.**

I. Introduction

Model checking is a powerful technique for automatically verifying that the behavior of a system implementation conforms to its specifications. For instance, if the system being verified is a C program, its specifications could be a set of assertions written by the programmer or obtained by automatic means from the source code. The advantages of model checking are often shaded by the high consumption of computational resources, the *state-space explosion problem*. Many algorithms have been developed to efficiently cope with this problem. In particular, the state-space of a system is often expressed symbolically, typically as a Boolean formula, instead of explicitly as a graph. Other approaches include *Bounded Model Checking* (BMC) [1], and different types of automated abstraction, such as predicate abstraction [2], interpolation-based reasoning [3], and function summarization [4], [5], [6]. Most state-of-the-art model checking tools implement some combinations of these methods in order to deal with complex verification tasks.

In this extended abstract we discuss a technique for extracting reusable information about software to avoid repeating previous computation in an incremental verification task. The repetition occurs naturally when the same piece of code is verified multiple times with respect to different properties (for example, [7]), or when a previously verified software needs to be re-verified after a software upgrade. We report that the presented approach often scales to software of practical significance, and in particular to validation cases provided by VTT Technical Research Centre of Finland and ABB. The technology is being integrated to the CCRT collaborative code review tool developed at IBM.

II. Function Summaries

We follow the usual notion in symbolic model checking that the program p to be verified is transformed, along with conditions c to be verified, into a propositional formula $\phi_p \wedge \phi_c$ in conjunctive normal form that is satisfiable if and only if there is a violation of one of the verification conditions c in the program. The formula can then be solved with an efficient, general purpose solver for propositional satisfiability (SAT). In the transformation process the program is first converted into a *static single assignment* (SSA) form where the loops are unwound up to some fixed bound and then to a propositional formula via *bit-blasting*.

Function summarization can be used when $\phi_p \wedge \phi_c$ has been proved unsatisfiable. A *summary* for a function f is a propositional formula I_f corresponding to a *Craig Interpolant* [8] obtained from the proof and partitioning of the formula $\phi_p \wedge \phi_c$ to the function ϕ_f being summarized and the rest of the program ϕ'_p. By construction the summary I_f is an over-approximation of the formula ϕ_f in the propositional sense that still results in an unsatisfiable formula when conjoined with the rest of the formula. More technically, it is guaranteed that $\models \phi_f \rightarrow I_f$ and that $\phi'_p \wedge I_f$ is unsatisfiable. To use the extracted summary I_f in later verification rounds, we substitute the formula ϕ_f in $\phi_p \wedge \phi_c$ with I_f.

III. Sequential Processing of Verification Conditions

Assume the program has now been verified to work correctly with respect to a verification condition c and we would like to verify it with respect to another condition c'. If the summaries I_f produced in the proof for c are sufficiently strong, substituting them instead of the full functions ϕ_f results potentially in significant speed-up. However, since a summary is an over-approximation of a function, made for a specific property, it may contain spurious behaviors. These behaviors may be crucial for checking another property, leading to spurious errors. In such a case, our method requires refinement, in which we analyze the spurious error trace. The method aims at determining the function calls substituted by summaries that occur on the error trace and influence the verification condition. We repeat the check again without using these summaries, but keeping the rest. If no such summary is identified, the error is real. The extraction, use, and refinement of summaries are described in detail in [9].

IV. Software Upgrade Checking

Consider now a case where an earlier version of the program has been checked with respect to a set of verification

978-1-4673-6136-1/13/$31.00

conditions, and now an upgrade of the program needs to be verified. We propose to reuse the already extracted summaries to prevent re-verification of the entire code. Initially we first check whether the old summaries are valid for the functions of the upgraded program. Since this check considers only code of the function bodies, its old summary and potentially summaries of its callees, it is very local and thus it tends to be computationally inexpensive. If old summaries are valid, the upgrade is safe. Otherwise, the check is propagated to the callers of the modified functions. When the summary of the call tree root is shown to be violated, a real error is found and it is reported to the user along with an error trace. After each successful check, any invalidated summaries are regenerated so that they are ready for the check of the next version. In addition, our method implements a counter-example guided refinement to deal with too coarse summaries during the checks. This process is described in details in [10].

V. Implementation

The sequential processing of verification conditions is implemented in a model checker called FunFrog [11], an extension of the CBMC [12] model checker. The upgrade checking method is implemented in a tool called eVolCheck [13]. The tools have been evaluated using a set of industrial benchmarks. Our experimentation confirms that the incremental analysis of upgrades containing incremental changes is often orders of magnitude faster than analysis performed from scratch. Both tools use the OpenSMT solver [14] both for satisfiability checks and interpolation. Note that OpenSMT is used as a SAT solver, which gives us bit-precise reasoning[1].

VI. Conclusions and Further Work

The FunFrog and eVolCheck tools were validated on a wide range of various benchmarks inccluding validation cases provided by project collaborators of the Pincette EU project[2]. In particular, they were used to verify the C part of an implementation of the DTP2 robot controller, developed by VTT for the ITER fusion reactor. They were also applied to validation cases, provided by ABB, on a code taken from the project implementing a core of a feeder protector and controller. The code originates from an embedded software used in the ABB hardware module. This is a large scale project containing many sub-projects which implement various functions of the feeder device. The total number of lines in the overall code is in millions. The implementation details and a thorough experimental analysis is available in [13], [11], [10], [15], [9].

In addition to the stand-alone use of eVolCheck, we are currently exploring the use of the tool as a static analysis engine within the hybrid static/dynamic upgrade checking platform developed as part of the Pincette project. The platform has been applied to the analysis of software developed by Pincette's industrial partners VTT, ABB, and Israeli Aerospace Industries (IAI), the last developing a software for a stabilized optical device payload (MSEOS) of unmanned airborne vehicles. As part of the project, eVolCheck is also integrated in the CCRT platform[3], a collaborative code review tool developed at IBM.

[1] Specialized SAT solvers without proof construction generally outperform OpenSMT in the satisfiability checks though they lack the interpolant generation features.

[2] http:/www.pincette-project.eu/

[3] CCRT is a proprietary tool of IBM

References

[1] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *The 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS '99)*, ser. Lecture Notes in Computer Science, vol. 1579. Springer, 1999, pp. 193–207.

[2] S. Graf and H. Saïdi, "Construction of abstract state graphs with PVS," in *The 9th International Conference on Computer Aided Verification (CAV '97)*, ser. Lecture Notes in Computer Science, vol. 1254. Springer, 1997, pp. 72–83.

[3] K. L. McMillan, "Applications of Craig Interpolation in Model Checking," in *The 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '05)*, ser. Lecture Notes in Computer Science, vol. 3440, 2005, pp. 1–12.

[4] ——, "Lazy abstraction with interpolants," in *The 18th International Conference on Computer Aided Verification, (CAV '06)*, ser. Lecture Notes in Computer Science, vol. 4144. Springer, 2006, pp. 123–136.

[5] ——, "Lazy annotation for program testing and verification," in *The 22nd International Conference on Computer Aided Verification (CAV'10)*, ser. Lecture Notes in Computer Science, vol. 6174. Springer, 2010, pp. 104–118.

[6] A. Albarghouthi, A. Gurfinkel, and M. Chechik, "Whale: An interpolation-based algorithm for inter-procedural verification," in *The 13th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI '12)*, ser. Lecture Notes in Computer Science, vol. 7148. Springer, 2012, pp. 39–55.

[7] T. Ball and S. K. Rajamani, "Bebop: A symbolic model checker for boolean programs," in *The 7th International SPIN Workshop (SPIN '00)*, vol. 1885. Springer, 2000, pp. 113–130.

[8] W. Craig, "Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory," *Journal of Symbolic Logic*, vol. 22, no. 3, pp. 269–285, 1957.

[9] O. Sery, G. Fedyukovich, and N. Sharygina, "Interpolation-based function summaries in bounded model checking," in *Haifa Verification Conference (HVC 2011)*, ser. Lecture Notes in Computer Science, vol. 7261. Springer, 2012, pp. 160–175.

[10] ——, "Incremental upgrade checking by means of interpolation-based function summaries," in *12th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2012)*, to appear.

[11] ——, "FunFrog: Bounded model checking with interpolation-based function summarization," in *The 10th International Symposium on Automated Technology for Verification and Analysis (ATVA 2012)*, ser. Lecture Notes in Computer Science, vol. 7561. Springer, 2012, pp. 203–207.

[12] E. Clarke, D. Kroening, and F. Lerda, "A tool for checking ANSI-C programs," in *The 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, (TACAS '04)*, ser. Lecture Notes in Computer Science, vol. 2988. Springer, 2004, pp. 168–176.

[13] G. Fedyukovich, O. Sery, and N. Sharygina, "eVolCheck: Incremental upgrade checker for C," in *19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2013)*, ser. Lecture Notes in Computer Science, vol. 7795. Springer, 2013, pp. 292–307.

[14] R. Bruttomesso, E. Pek, N. Sharygina, and A. Tsitovich, "The OpenSMT solver," in *The 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '10)*, ser. Lecture Notes in Computer Science, vol. 6015. Springer, 2010, pp. 150–153.

[15] G. Fedyukovich, O. Sery, and N. Sharygina, "Function summaries in software upgrade checking," in *Haifa Verification Conference (HVC 2011)*, ser. Lecture Notes in Computer Science, vol. 7261. Springer, 2012, pp. 257–258.