Function Summaries in Software Upgrade Checking[⋆]

Grigory Fedyukovich, Ondrej Sery, and Natasha Sharygina

University of Lugano, Formal Verification and Security Lab, Lugano, Switzerland
{grigory.fedyukovich,ondrej.sery,natasha.sharygina}@usi.ch
http://verify.inf.usi.ch/

We propose a new technique for checking of software upgrades based on an optimization of bounded model checking (BMC) with interpolation-based function summaries. In general, function summaries avoid duplicate actions during the verification process. We extract function summaries as an over-approximation of the actual function behavior after a successful model checker run and use it in the consecutive runs. It is useful in real life, when the same code gets analyzed multiple times for different properties. As a practical example of this situation, consider SLAM [1] which is used in a Static Driver Verifier to verify Windows device drivers. There the same code of the device driver is model checked repeatedly for different sets of predefined properties. In every run, function summaries could be generated and reused in the others to reduce the computational burden.

For generation of function summaries with respect to a given property (e.g., a user supplied or automatically generated assertion), we use Craig Interpolantion [3]. As in the standard BMC, we transform the program into a logical formula, via unwinding, conversion into the SSA form, followed by addition of the negated property, and bit-blasting. Then we check satisfiability of the resulting formula, passing it to a SAT solver. From UNSAT result (i.e., the property holds), the solver generates an interpolant for each function call, occurring in the formula. By construction, the interpolant is an over-approximation of the corresponding function, i.e., its summary. To use the extracted summary later, we substitute it in the formula instead of processing the entire function again.

Consider we check the program with respect to a different property. Since the 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 properties, leading to spurious errors. In such a case, our method requires refinement, in which we analyze the spurious error trace. It aims at identifying function calls substituted by summaries that occur on the error trace and influence the assertion (determined by dependency analysis). We repeat the check again without using these summaries, but keeping the rest. If no such summary is identified, the error is real. We describe the extraction, use, and refinement of summaries in details in [5], and implement the process in a tool called FunFrog[1].

As our contribution to upgrade checking, we propose to reuse the already extracted summaries to prevent re-verification of the entire code. Our upgrade

[⋆] This work is partially supported by the European Community under the call FP7-ICT-2009-5 — project PINCETTE 257647.

[1] http://verify.inf.usi.ch/funfrog

checking procedure verifies that a property of a program still holds after the program is modified. The proposed method is based on the following observation. There are two possibilities for the modified functions, either their old summaries are still a valid over-approximation or not. If they are valid, properties of the old version, for which the summaries are relevant, still hold also in the new version.

Our method identifies the functions that were modified using syntactic comparison of different versions. Then, for all modified functions, it checks if the summaries are still valid. If the check fails, we detect the calling context of the function and try again. It is important to note two things. First, the check compares the function with its summary and ignores the rest of the system. So the check is local and thus relatively cheap. Second, we can reuse the summaries of the unmodified functions when performing the check. Of course, this implies that refinement may be necessary.

Obviously, the cost of the upgrade check would depend on the extend of the change both syntactical (number of modified functions) and semantical (number of invalid summaries). Thus our approach is likely to benefit for smaller changes that do not violated the properties. In the worst case, the problem is refined to checking of the entire upgraded program, which, however, is rare in practice.

In [2], the authors propose an algorithm for containment and compatibility checks for upgrades of software components. They use predicate abstraction and the CEGAR loop to create and refine models of the new and old components. Though the goal is the same, the means are different. In the context of dynamic test generation, the authors of [4] purpose to use function summaries for testing upgrades. Their notion of a summary is a collection of concrete pairs of inputs and outputs, i.e., an under-approximation, and thus their check can be not sound.

In future work, we will implement the proposed upgrade checker in the FunFrog tool. In collaboration with the PINCETTE project validators, we plan to apply the tool to verify real-world applications. As shown in [5], the current release of FunFrog, despite not being final, was already applied to industrial benchmarks. Enlarging the set of benchmarks, will help us to analyze experimentally the benefits of our approach and to reveal directions for further optimizations. We expect to confirm experimentally that interpolant-based upgrade checking outperforms model checking of the entire upgraded program.

References

1. Ball, T., Rajamani, S.K.: The SLAM Project: Debugging System Software via Static Analysis. In: POPL 2002, pp. 1–3 (January 2002)
2. Chaki, S., Clarke, E., Sharygina, N., Sinha, N.: Dynamic Component Substitutability Analysis. In: Fitzgerald, J.S., Hayes, I.J., Tarlecki, A. (eds.) FM 2005. LNCS, vol. 3582, pp. 512–528. Springer, Heidelberg (2005)
3. Craig, W.: Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. J. of Symbolic Logic, 269–285 (1957)
4. Godefroid, P., Lahiri, S., Rubio-González, C.: Statically Validating Must Summaries for Incremental Compositional Dynamic Test Generation. In: Yahav, E. (ed.) SAS 2011. LNCS, vol. 6887, pp. 112–128. Springer, Heidelberg (2011)
5. Sery, O., Fedyukovich, G., Sharygina, N.: Interpolation-Based Function Summaries in Bounded Model Checking. In: Eder, K., Lourenço, J., Shehory, O. (eds.) HVC 2011. LNCS, vol. 7261, pp. 160–175. Springer, Heidelberg (2012)