Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 17

Just test what you cannot verify![1]

Mike Czech[2] Marie-Christine Jakobs[3] Heike Wehrheim[4]

Abstract: Software verification is an established method to ensure software safety. Nevertheless, verification still often fails, either because it consumes too much resources, e.g., time or memory, or the technique is not mature enough to verify the property. Often then discarding the *partial verification*, the validation process proceeds with techniques like *testing*.

To enable standard testing to profit from previous, partial verification, we use a summary of the verification effort to simplify the program for subsequent testing. Our techniques use this summary to construct a *residual program* which only contains program paths with unproven assertions. Afterwards, the residual program can be used with standard testing tools.

Our first experiments show that testing profits from the partial verification. The test effort is reduced and combined verification and testing is faster than a complete verification.

Keywords:combination verification and validation, conditional model checking, static analysis, testing, slicing

1 Overview

Although automatic software verification and its tool support evolved in recent years, software verification still fails. The verified property may be beyond the capabilities of a tool or its verification requires too many resources, e.g., time and memory. Thus, verification cannot be applied in an "on-the-fly" context in which validation should be carried out in a small amount of time and probably on a device with restricted resources. To still gain confidence in the software, after a failed verification, further validation techniques like testing are applied which often discard the previous, partial verification results.

Within the Collaborative Research Centre SFB 901 at the University of Paderborn we developed two orthogonal approaches to combine verification and testing [CJW15]. Our idea is to consider the partial verification during testing and only test paths which have not been fully verified. To use standard testing techniques we build a new program for testing, the *residual program*, which contains only the non-verified paths. Both approaches start with a verification tool that keeps track of its (abstract) state space exploration in terms of an abstract reachability graph (ARG). If the verification tool stops with an uncomplete verification, it generates a condition as proposed in conditional model checking [Be12].

[1] This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901).

[2] Universität Paderborn, Institut für Informatik, Warburger Str. 100, 33098, Paderborn, mczech@mail.upb.de

[3] Universität Paderborn, Institut für Informatik, Warburger Str. 100, 33098, Paderborn, marie.christine.jakobs@upb.de

[4] Universität Paderborn, Institut für Informatik, Warburger Str. 100, 33098, Paderborn, wehrheim@upb.de

This condition is related to the ARG and describes in a graph manner which program paths are proven correct and which remain. Next, our approaches use the condition to construct the residual program. Afterwards, the residual program is tested.

Our first approach computes its residual program via a product combination of the program and condition, excluding paths of the condition which are proven correct. Thus, due to e.g. loop unwindings during verification, the residual program's structure may differ from the original program. It is only a semantical subprogram.

Our second approach constructs a syntactical subprogram which contains all statements that influence the assertions which have not been fully verified. These assertions are all assertions on the unexplored paths in the condition and become the slicing criteria for dependence based slicing. At last, dependence based slicing builds the residual program.

We can easily combine our two approaches. First, we apply the product construction technique to construct an intermediate residual program. Second, the set of all assertions in the intermediate residual program becomes our slicing criterion. Finally, we slice the intermediate residual program to obtain the final residual program for testing.

In our experiments, we used the verification tool CPACHECKER [BK11] for partial verification, Frama-C [Cu12] for slicing and the concolic test tool KLEE [CDE08]. On our small benchmark suite, the combination of verification and testing was mostly faster than complete verification. Additionally, the two slicing based approaches reduced the test effort (number of tests and program size) but none always outperformed the other.

Our proposed combinations of verification and testing demonstrate that testing benefits from previous partial verification.

References

[Be12] Beyer, Dirk; Henzinger, Thomas A.; Keremoglu, M. Erkan; Wendler, Philipp: Conditional Model Checking: A Technique to Pass Information Between Verifiers. In: FSE. FSE '12. ACM, pp. 1–11, 2012.

[BK11] Beyer, Dirk; Keremoglu, M.Erkan: CPAchecker: A Tool for Configurable Software Verification. In (Gopalakrishnan, Ganesh; Qadeer, Shaz, eds): CAV. volume 6806 of LNCS. Springer, pp. 184–190, 2011.

[CDE08] Cadar, Cristian; Dunbar, Daniel; Engler, Dawson: KLEE: Unassisted and Automatic Generation of High-coverage Tests for Complex Systems Programs. In: OSDI. OSDI'08. USENIX Association, pp. 209–224, 2008.

[CJW15] Czech, Mike; Jakobs, Marie-Christine; Wehrheim, Heike: Just Test What You Cannot Verify! In (Egyed, Alexander; Schaefer, Ina, eds): FASE, volume 9033 of LNCS, pp. 100–114. Springer Berlin Heidelberg, 2015.

[Cu12] Cuoq, Pascal; Kirchner, Florent; Kosmatov, Nikolai; Prevosto, Virgile; Signoles, Julien; Yakobowski, Boris: Frama-C. In (Eleftherakis, George; Hinchey, Mike; Holcombe, Mike, eds): SEFM. volume 7504 of LNCS. Springer, pp. 233–247, 2012.