Exchanging Verification Witnesses between Verifiers *

Dirk Beyer [1], Matthias Dangl [2], Daniel Dietsch [3], Matthias Heizmann [3]

[1] LMU Munich, Germany [2] University of Passau, Germany [3] University of Freiburg, Germany
http://www.sosy-lab.org/research/correctness-witnesses/

Abstract: Standard verification tools provide a counterexample to witness a specification violation. Since a few years, such a witness can be validated by an independent validator using an exchangeable witness format. This way, information about the violation can be shared across verifiers and the user can use standard tools to visualize and explore witnesses. This technique is not yet established for the correctness case, where a program fulfills a specification. Even for simple programs, users often struggle to comprehend why a program is correct, and there is no way to independently check the verification result. We recently closed this gap by complementing our earlier work on violation witnesses with correctness witnesses. The overall goal to make proofs available to engineers is probably as old as programming itself, and proof-carrying code was proposed two decades ago — our goal is to make it practical: We consider witnesses as first-class exchangeable objects, stored independently from the source code and checked independently from the verifier that produced them, respecting the principle of separation of concerns. At any time, the correctness witness can be used to reconstruct a correctness proof to establish trust. We extended two state-of-the-art verifiers, CPACHECKER and ULTIMATEAUTOMIZER, to produce and validate witnesses.

1 Introduction

The omnipresent dependency on software in society and industry makes it necessary to ensure reliable and correct functioning of the software. This trend will continue and become even more important in the future. During the last decade, various conceptual breakthroughs in verification research were achieved, and, as showcased by the annual TACAS International Competition on Software Verification (SV-COMP) [1] [Bey16], many successful software verifiers were developed.

Recently, the problem of false alarms that verification tools sometimes produce has been addressed [BDD+15]: Formerly, a verification tool reported found bugs as counterexample traces in a tool-specific manner; those counterexamples were often not readable and therefore hardly usable. Determining whether the reported bug was a false alarm or described an actual programming error that needed to be fixed was a tedious manual process for the user. Exchangeable violation witnesses resolve this issue, because the general syntax allows new tools for presentation to be developed and used [BD16]. Witnesses should be considered as first-class objects that have much more value than the actual verification result TRUE

*This is a summary of a full article on this topic that appeared in Proc. FSE 2016 [BDDH16].

[1] http://sv-comp.sosy-lab.org/

validation is fully automatic.

Our recent work [BDDH16] complements the work on violation witnesses [BDD+15] with a method for producing and validating *correctness witnesses*. The most recent edition of the competition on software verification [Bey16] revealed that soundness is a big issue: ten out of 13 participating verifiers in the category 'Overall' reported wrong correctness claims for verification tasks with known specification violations. One of the submissions was even claiming safety for 962 out of 2 348 verification tasks that were known to contain a bug. This rather embarrassing situation of the state-of-the-art in software verification can be fixed by producing correctness witnesses and letting a witness validator confirm the result. A verification result should be trusted only if it can be confirmed by at least one other verifier.

We propose that a verifier should be required to augment a verification result with a machine-readable and exchangeable witness, such that both, bug alarms and claims of safety, may be validated. With this technique, a trusted validator establishes trust in the verification results produced by an untrusted verifier, and even in the absence of a trusted validator the user's confidence in a verification result can be increased by applying different validators to a verification witness. Witnesses can be read by humans (perhaps using a visualization or inspection tool) or by a witness validator.

We use the standard concept of (non-deterministic) finite automata to represent correctness witnesses. A correctness-witness automaton observes the program locations (along the control flow) that the verifier explores and provides invariants that hold at the locations that the verifier visits. A correctness witness is valid if its predicates are invariants for the program, and a validator should reject witnesses with incorrect invariants. The strength of the invariants determines the quality of the witnesses, but no particular strength is required. Witness validation can be more efficient than verification because it might be easier to (re-) verify that invariants indeed hold, while the verification needs to come up with the invariants. The task of finding useful invariants is in general considered one of the key challenges in software verification. Generalizing this approach allows for a lot of flexibility, because the more helpful the candidate invariants are, the less work has to be performed by the validator.

References

[BD16] D. Beyer and M. Dangl. Verification-Aided Debugging: An Interactive Web-Service for Exploring Error Witnesses. In *Proc. CAV*, pages 502–509. Springer, 2016.

[BDD+15] D. Beyer, M. Dangl, D. Dietsch, M. Heizmann, and A. Stahlbauer. Witness Validation and Stepwise Testification across Software Verifiers. In *Proc. ESEC/FSE*, pages 721–733. ACM, 2015.

[BDDH16] D. Beyer, M. Dangl, D. Dietsch, and M. Heizmann. Correctness Witnesses: Exchanging Verification Results between Verifiers. In *Proc. FSE*, pages 326–337. ACM, 2016.

[Bey16] Dirk Beyer. Reliable and Reproducible Competition Results with BENCHEXEC and Witnesses. In *Proc. TACAS*, LNCS 9636, pages 887–904. Springer, 2016.

[BHKW12] D. Beyer, T. A. Henzinger, M. E. Keremoglu, and P. Wendler. Conditional Model Checking: A Technique to Pass Information between Verifiers. In *FSE*. ACM, 2012.