Reliability Analysis in Symbolic Pathfinder: A brief summary*

Antonio Filieri[a], Corina S. Păsăreanu[b], and Willem Visser[c]

[a]Institute of Software Technology, University of Stuttgart, Stuttgart, Germany
[b]Carnegie Mellon Silicon Valley, NASA Ames, Moffet Field, CA, USA
[c]Stellenbosch University, Stellenbosch, South Africa

Abstract: Designing a software for critical applications requires a precise assessment of reliability. Most of the reliability analysis techniques perform at the architecture level, driving the design since its early stages, but are not directly applicable to source code. We propose a general methodology based on symbolic execution of source code for extracting failure and success paths to be used for probabilistic reliability assessment against relevant usage scenarios. Under the assumption of finite and countable input domains, we provide an efficient implementation based on Symbolic PathFinder that supports the analysis of sequential and parallel Java programs, even with structured data types, at the desired level of confidence. We validated our approach on both NASA prototypes and other test cases showing a promising applicability scope.

Design and implementation of software systems for critical applications is stressing the the need for methodologies and tools to assess and certify its reliability. Different definitions of reliability are introduced within different domains. In this paper we generically refer to reliability as the probability of the software to successfully accomplish its assigned task when requested (Che80). In reality most of the software we use daily is defective in some way, though it can most of the time do its job. Indeed, the presence of a defect in the code may never be realized if the input does not activate the fault (ALRL04). For this reason, the reliability of a software heavily depends on the actual *usage profile* the software is required to deal with.

Most of the approaches for software reliability assessment have been based on the analysis of formal models derived from architectural abstractions (GPMT01; IN08). Model-driven techniques have often been used to keep design models synchronized with the implementation (IN08) and with analysis models. To deal with code, black-box (Mus93) or some ad-hoc reverse engineering approaches have been proposed, e.g. (GPHP05).

In (FPV13), we proposed the systematic and fully automated use of symbolic execution (Kin76; APV07) to extract logical models of failure and successful execution paths directly from source code. Each execution path is fully characterized by a *path condition*, i.e. a set of constraints on the inputs that, if satisfied by the input values, make the execution follow the specific path through the code. In our approach, we label the (terminating) execution paths as either *success* or *failure*. The set of path conditions produced by symbolic

*This paper reports a summary of (FPV13). Please refer to the original paper for a complete exposition.

execution is a complete partition of the input domain (Kin76). Hence, given a probability distribution on the input values, the reliability of the software can be formalized as the probability of satisfying any of the successful path conditions. We take the probability distribution over the input domain as the formalization of the usage profile. Furthermore, we assume the inputs to account for all the external interactions of the software, i.e. with the users, external resources, or third-party applications. Non-termination in presence of loops or recursion is handled by bounded symbolic execution (APV07). In this case interrupted execution paths are labeled as *grey*. For an input satisfying a grey path condition we cannot predict success nor failure. Thus, the probability for an input value to satisfy a grey path condition can be used to define a precise confidence measure to assess the impact of the execution bounds and the consequent quality of the reliability prediction.

As for (FPV13), we focused on inputs ranging over finite domains. This restriction allows us to make use of model counting procedures for efficiently computing the probability of execution paths. Our implementation, based on Symbolic PathFinder (APV07), supports linear integer arithmetic, complex data-structures, loops, and also concurrency. For multi-threaded programs the actual reliability depends both on the usage profile and on the scheduling policy. In this case we identify the best and worst schedule for a given usage profile, that respectively lead to the highest and lowest reliability achievable for that usage.

We evaluated our approach on both examples from the Literature and on NASA's Onboard Abort Executive, a real-life complex software from the aerospace domain. Both the accuracy and the analysis time revealed a promising applicability scope for our approach.

References

[ALRL04] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secure Comput.*, 1(1):11—33, 2004.

[APV07] S. Anand, C. S. Păsăreanu, and W. Visser. JPF-SE: A Symbolic Execution Extension to Java PathFinder. volume 4424 of *LNCS*, pages 134—138. Springer, 2007.

[Che80] R.C. Cheung. A User-Oriented Software Reliability Model. *IEEE Trans. Soft. Eng.*, SE-6(2):118—125, 1980.

[FPV13] A. Filieri, C. S. Păsăreanu, and W. Visser. Reliability analysis in symbolic pathfinder. ICSE, pages 622—631. IEEE, 2013.

[GPHP05] K. Goseva-Popstojanova, M. Hamill, and R. Perugupalli. Large empirical case study of architecture-based software reliability. In *ISSRE*, pages 52—61, Nov 2005.

[GPMT01] K. Goseva-Popstojanova, A.P. Mathur, and K.S. Trivedi. Comparison of architecture-based software reliability models. In *ISSRE*, pages 22—31, 2001.

[IN08] A. Immonen and E. Niemela. Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Software and Systems Modeling*, 7:49—65, 2008.

[Kin76] J. C. King. Symbolic execution and program testing. *Commun. ACM*, 19(7):385—394, Jul 1976.

[Mus93] J. Musa. Operational Profiles in Software-Reliability Engineering. *IEEE Software*, 10(2):14—32, March 1993.