Understanding Parameters of Deductive Verification: An Empirical Investigation of KeY

Alexander Knüppel[1] Thomas Thüm[1] Carsten Immanuel Pardylla[1] Ina Schaefer[1]

Abstract: As formal verification of software systems is a complex task comprising many algorithms and heuristics, modern theorem provers offer numerous parameters that are to be selected by a user to control how a piece of software is verified. Evidently, the number of parameters even increases with each new release. One challenge is that default parameters are often insufficient to close proofs automatically and are not optimal in terms of verification effort. The verification phase becomes hardly accessible for non-experts, who typically must follow a time-consuming trial-and-error strategy to choose the right parameters even for trivial pieces of software. To aid users of deductive verification, we apply machine learning techniques to empirically investigate which parameters and combinations thereof impair or improve provability and verification effort. We exemplify our procedure on the deductive verification system KeY 2.6.1 and specified extracts of OpenJDK, and formulate 53 hypotheses of which only three have been rejected. We identified parameters that represent a trade-off between high provability and low verification effort, enabling the possibility to prioritize the selection of a parameter for either direction. Our insights give tool builders a better understanding of their control parameters and constitute a stepping stone towards automated deductive verification and better applicability of verification tools for non-experts.

Keywords: Deductive Verification, Design by Contract, Formal Methods, Theorem Proving, KeY, Control Parameters, Automated Reasoning

Overview

Software verification is vital for safety-critical and security-critical applications applied in industry. Although deductive verification is a promising static analysis technique that targets program verification directly on source code level, it has not yet found its way into industry due to issues with the scalability in specification and verification. Indeed, specifying large-scale software systems for efficient verification still demands high effort and expertise, which constitutes a problem for typical software developers who are not trained in proof theory.

Our long-term goal is to make deductive verification accessible for mainstream software developers. In this regard, one often overlooked hurdle that inexperienced users face is parameterization. While parameterization of formal method tools comes with the promise to ease the process of automatic verification, we exhibited that setting the right values for

[1] TU Braunschweig, Germany

doi:10.18420/se2019-51

the ever growing amount of parameters is challenging. Moreover, default parameters are often insufficient to close proofs automatically and are typically not optimal in terms of verification effort. Hence, in a recent publication, we empirically investigated the influence of parameters of KeY 2.6.1 [Ah16], a deductive verification system for Java source code.

The main results of our empirical study have been presented at the 9th International Conference on Interactive Theorem Proving (ITP'18) in Oxford, United Kingdom [Kn18b]. Based on our experience and observations combined with studying the online documentation of KeY, numerous publications, all tool tips in the KeY front-end, and the KeY book [Ah16], we formulated a total of 38 assumptions on how options in KeY improve or impair provability and verification effort. We derived a total of 53 statistical hypotheses and empirically measured the effect of different parameter configurations by employing significance tests and machine-learning techniques.

For our verification targets, we formally specified parts of OpenJDK's Collection API with JML [Kn18a]. By not only testing our hypotheses, but also by employing *SPL Conqueror* [Si12] for learning parameter-influence models, we were able to identify options of parameters that should be prioritized regarding their impact on verification effort. Moreover, we even identified parameters whose options represent a trade-off between provability and verification effort. Our insights provide valuable recommendations to users on which parameters to prioritize given a verification requirement. Moreover, tool builders can utilize our insights to improve on the user experience. For instance, implementing a recommendation system for parameters based on our investigation would help users to verify software more easily. Furthermore, KeY may hide insignificant parameters in specific verification scenarios or fine-tune parameters automatically during proofs. Although we focused on KeY, the problem of choosing sufficient parameters is not limited to deductive verification tools alone. Thus, our proposed approach for empirically studying the influence of parameters is applicable to arbitrary verification tools.

References

[Ah16] Ahrendt, Wolfgang; Beckert, Bernhard; Bubel, Richard; Hähnle, Reiner; Schmitt, Peter H; Ulbrich, Mattias: Deductive Software Verification–The KeY Book: From Theory to Practice. Springer, 2016.

[Kn18a] Knüppel, Alexander; I. Pardylla, Carsten; Thüm, Thomas; Schaefer, Ina: Experience Report on Formally Verifying Parts of OpenJDK's API with KeY. In: Proceedings of the Fourth Workshop on Formal Integrated Development Environment. Springer, 2018.

[Kn18b] Knüppel, Alexander; Thüm, Thomas; Pardylla, Carsten I.; Schaefer, Ina: Understanding Parameters of Deductive Verification: An Empirical Investigation of KeY. In: Proc. Int'l. Conf. Interactive Theorem Proving (ITP). Springer, 2018.

[Si12] Siegmund, Norbert; Rosenmüller, Marko; Kuhlemann, Martin; Kästner, Christian; Apel, Sven; Saake, Gunter: SPL Conqueror: Toward optimization of non-functional properties in software product lines. Software Quality Journal, 20(3-4):487–517, 2012.