Tool Support for Correctness-by-Construction

Tobias Runge[1] Ina Schaefer[2] Loek Cleophas[3] Thomas Thüm[4] Derrick Kourie[5] Bruce W. Watson[6]

Abstract: This work was published at the International Conference on Fundamental Approaches to Software Engineering (FASE) 2019 [Ru19]. We tackled a fundamental problem of missing tool support of the correctness-by-construction (CbC) methodology that was proposed by Dijsktra and Hall and revised to a light-weight and more amenable version by Kourie and Watson. Correctness-by-construction (CbC) is an incremental approach to create programs using a set of small, easily applicable refinement rules that guarantee the correctness of the program with regard to its pre- and postcondition specifications. Our goal was to implement a functional and user-friendly IDE, so that developers will adapt to the CbC approach and benefit from its advantages (e.g., defects can be easily tracked through the refinement structure of the program). The tool has a hybrid textual and graphical IDE that programmers can use to refine a specification into a correct implementation. The textual editor fits programmers that want to stay in their familiar environment, while the graphical editor highlights the refinement structure of the program to give visual feedback if errors occur, using KeY as verification backend. The tool was evaluated regarding feasibility and effort to develop correct programs. Here, slight benefits in comparison to a standard verification approach were discovered.

Keywords: correctness-by-construction; tool support; formal methods; verification

Overview

Correctness-by-Construction (CbC) [KW12] is a methodology to construct formally correct programs guided by a pre-/postcondition specification. Starting with an abstract program, formally verified refinement rules are applied to incrementally refine the program to a concrete implementation. In the literature [KW12, Wa16], CbC is described as having many benefits: The structured reasoning discipline that is enforced by the refinement rules reduces the appearance of defects. If defects do occur, they can be tracked through the refinement structure. Furthermore, the formal process increases trust in the program. To check these benefits, we implement tool support that enables CbC to be used by programmers. We want to compare CbC with the prevalent post-hoc verification approach, where program correctness is proven after construction.

[1] TU Braunschweig, Germany tobias.runge@tu-bs.de
[2] TU Braunschweig, Germany i.schaefer@tu-bs.de
[3] TU Eindhoven, The Netherlands and Stellenbosch University, South Africa l.g.w.a.cleophas@tue.nl
[4] Ulm University, Germany thomas.thuem@uni-ulm.de
[5] Stellenbosch University, South Africa derrick@fastar.org
[6] Stellenbosch University, South Africa bruce@fastar.org

In this work, we present CorC, an IDE that supports the CbC approach with a hybrid textual and graphical user interface. The IDE support users to apply refinement rules to an abstract program until the program is fully refined. In each refinement step, the correct application is guaranteed by using the theorem prover KeY [Ah16]. Each proof obligation can be immediately discharged during program development. In the textual editor, programmers enrich source code with specification and directly see if a refinement is unprovable. The graphical editor is useful to get an overview of all refinement steps and track errors in the program. To not burden programmers, they can switch automatically between both views. As CbC is not tailored to a specific host language, we implemented CorC in such a way that the language can be exchanged: Any imperative programming language with a specification language and an automatic verification tool can be integrated.

In our evaluation, we compared CorC with standard post-hoc verification using KeY as prover in both cases. We investigated the hypothesis whether the verification of algorithms is faster with CorC than with post-hoc verification. Therefore, we implemented seven algorithms with CorC and as plain Java code with specification. In each case, we measured that the verification time was lower for CorC, indicating a reduced proof complexity. The result is even statistically significant which provides empirical evidence for our hypothesis.

In summary, we extended the KeY ecosystem with tool support for the correctness-by-construction methodology. With CorC, programmers can utilize CbC to construct correct programs and use the results to bootstrap post-hoc verification as an additional check if necessary. They can reduce the verification time, as demonstrated in our evaluation, and benefit from synergistic effects of both approaches.

Bibliography

[Ah16] Ahrendt, Wolfgang; Beckert, Bernhard; Bubel, Richard; Hähnle, Reiner; Schmitt, Peter H; Ulbrich, Mattias: Deductive Software Verification–The KeY Book: From Theory to Practice, volume 10001. Springer, 2016.

[KW12] Kourie, Derrick G; Watson, Bruce W: The Correctness-by-Construction Approach to Programming. Springer Science & Business Media, 2012.

[Ru19] Runge, Tobias; Schaefer, Ina; Cleophas, Loek; Thüm, Thomas; Kourie, Derrick; Watson, Bruce W.: Tool Support for Correctness-by-Construction. In: Fundamental Approaches to Software Engineering. volume 11424 of Lecture Notes in Computer Science. Springer, pp. 25–42, 2019.

[Wa16] Watson, Bruce W; Kourie, Derrick G; Schaefer, Ina; Cleophas, Loek: Correctness-by-Construction and Post-hoc Verification: A Marriage of Convenience? In: International Symposium on Leveraging Applications of Formal Methods. volume 9952 of Lecture Notes in Computer Science. Springer, pp. 730–748, 2016.