Tutorial: The Correctness-by-Construction Approach to Programming Using CorC

Ina Schaefer
TU Braunschweig
Braunschweig, Germany
i.schaefer@tu-bs.de

Tobias Runge
TU Braunschweig
Braunschweig, Germany
tobias.runge@tu-bs.de

Loek Cleophas
TU Eindhoven
Eindhoven, The Netherlands
Stellenbosch University
Stellenbosch, South Africa
l.g.w.a.cleophas@tue.nl

Bruce W. Watson
Centre for AI Research
School for Data-Sci. & Comp. Thinking
Stellenbosch University
Stellenbosch, South Africa
bruce@fastar.org

2021 IEEE Secure Development Conference (SecDev) | 978-1-6654-3170-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/SecDev51306.2021.00012

***Abstract*—The Correctness-by-Construction tutorial focuses on a structured programming approach for correct software development. Besides functional correctness, also non-functional properties such as security properties can be guaranteed using the CbC approach. In this tutorial, the participants learn a good practice to develop software that is midway between formal approaches and a "hack into correctness" style.**

***Index Terms*—tutorial, formal methods, correctness-by-construction**

I. Introduction

The purpose of the tutorial is to influence the way participants approach the task of developing algorithms, with a view to improving code quality. The tutorial features: a step-by-step explanation of how to derive provably correct algorithms using small and tractable refinements rules; a detailed illustration of the methodology through a set of carefully selected examples of increasing complexity; a demonstration of how practical non-trivial algorithms have been derived. The focus is on bridging the gap between two extreme methods for developing software. On the one hand, some approaches are so formal that they scare off all, but the most dedicated theoretical computer scientists. On the other, there are some who believe that any measure of formality is a waste of time, resulting in software that is developed by following gut feelings and intuitions.

II. Correctness-by-Construction

Correctness-by-Construction (CbC) [KW12], [Mor94] is an approach to incrementally create formally correct programs guided by pre- and postcondition specifications. A program is created using refinement rules that guarantee the resulting implementation is correct with respect to the specification.

The CbC approach to program development begins with a Hoare triple comprising a precondition, an abstract statement, and a postcondition. Such a triple should be read as a total correctness assertion, if the precondition holds, and its abstract statement executes then the execution will terminate, and its postcondition will hold. This triple can be refined by using a set of refinement rules, i.e., the statement is replaced by more concrete statements. For example, a loop is introduced, or an abstract statement is replaced by an assignment. If no abstract statement remains, the code is fully specialized.

III. CbC by Example

The tool CorC [RSC+19] is a hybrid textual and graphical IDE for the development of functional correct programs using CbC. In the tool, a starting Hoare triple specification $\{P\}S\{Q\}$ with a precondition P, a postcondition Q, and an abstract program S is refined stepwise to a correct implementation. The refinement is done by applying refinement rules which guarantee that the refined program still satisfies its starting specification. CorC checks the side-conditions of each applied refinement automatically such that the programmer gets direct feedback whether a refinement cannot be proven.

In Fig. 1, we construct a linear search algorithm. Each node represents one Hoare triple. An arrow is the application of a refinement, connecting a Hoare triple with the refined Hoare triple. The linear search algorithm has a precondition $P := appears(a, x, 0, a.length)$. This predicate states that the element x appears in the array a between the boundaries 0 and the array length. The postcondition $Q := modifiable(i);\ a[i] = x$ states that the element at index i in array a is equal to the searched element x. With `modifiable` we specify that only the variable i can be altered. In CorC, the used parameters and local variables are declared in a node on the top right. Below this node, global conditions are expressed. These conditions (e.g., invariants) have to be true in each step of the program.

To find the element in a linear search, we traverse the array from back to front. As invariant, we specify ! $appears(a, x, i + 1, a.length)$. We split the array in two parts: the part which we already examined and where x is not found; and the part that should still be examined. The program is constructed with four refinement steps. First, the composition rule [KW12] splits the starting Hoare triple into two triples with an intermediate condition which is equal to our invariant. The first statement *statement1* is refined to an assignment to start at the end of the array. The second statement *statement2* is refined to a loop. We repeat the loop until we found x. The loop body is refined in the last step. Here, we decrease the variable i. Each refinement is verified automatically by CorC to prove that the refined program satisfies the starting specification. For example, in the last refinement step, we prove that the invariant is preserved in

978-1-6654-3170-5/21/$31.00 © 2021, Tobias Runge. Under license to IEEE.
DOI 10.1109/SecDev51306.2021.00012

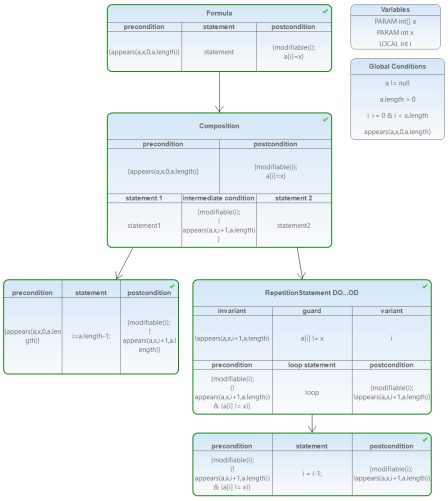

Fig. 1. Linear Search algorithm constructed in CorC

each iteration. The green borders of the nodes indicate the proven program refinements.

IV. Content of the Tutorial

In the tutorial, we want to introduce participants to the CbC approach to programming using the tool CorC. First, we provide the theoretical background of refinement rules. Afterwards, we apply the CbC approach to a series of examples. By using a tool CorC[1], the participants can try the CbC approach on their own during the tutorial without any installation burden. We present how CbC supports the construction of large-scale algorithmic families, and compare CbC against post-hoc verification. While CbC traditionally focuses on functional correctness (specifications), we will also consider the constructive development of non-functional properties, such as security properties, which can be formulated as information flow properties [SRK+18], [RKTS20]. The construction of secure programs by checking information flows will be the second part of the tutorial.

The audience for this tutorial is everyone interested in software development that wants to get insights into an alternative programming style that values a structured and correct programming process. The purpose of the tutorial is to influence the way the participants approach the task of developing algorithms. Instead of specifying a problem and solving the problem by coding with gut feelings and intuitions, we want to focus on the more formal CbC approach to construct programs. The expected learning outcome should be that the participants reflect on their coding style and find their best practice to construct formally correct programs [RTC+20], [WKSC16].

The tutorial relies on Kourie and Watson's easily accessible text book *The Correctness-by-Construction Approach to Programming* [KW12] and other publications by the tutorial presenters and co-authors [RSC+19], [RTC+20], [SRK+18], [RKTS20], [WKSC16]. These publications are additional material for interested participants.

[1]https://www.isf.cs.tu-bs.de/WebCorC/,https://github.com/TUBS-ISF/CorC

V. Agenda

1. *Introduction to CbC (Motivation and Foundations)*. We introduce Hoare logic, refinement rules, and supplementary specification such as loop invariants.
2. *CbC examples step-by-step*. Small algorithms like linear search and pattern matching are constructed stepwise to explain the CbC approach.
3. *Introduction to the CorC tool*. The participants can implement algorithms using CorC on their own.
4. *CbC for security (non-functional properties)*. We introduce how security-critical programs can be incrementally constructed using a refinement procedure. CbC is extended to track the information flow of the programs during each refinement step.
5. *Advanced CbC and ongoing research*.
   - *Construction of algorithm taxonomies*. We show how to construct an algorithmic family using taxonomies by stepwise refinements. A taxonomy describes the commonalities and differences of algorithms for the same algorithmic problem in a hierarchical structure.
   - *CbC in comparison to post-hoc verification*. We discuss the differences of both approaches to construct programs, but we will also show how to combine both approaches for synergistic effects.
   - *Discussion*. We close the tutorial with a discussion about the CbC approach and the introduced tool.

VI. Previous tutorials

The CbC tutorial was held four times. At the International Symposium on Formal Method (FM) in 2014 and 2019, at the International Conference on Software Quality, Reliability & Security (QRS) in 2017, and at the International Colloquium on Theoretical Aspects of Computing (ICTAC) in 2018.

References

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

[Mor94] Carroll Morgan. *Programming from Specifications*. Prentice Hall, 2nd edition, 1994.

[RKTS20] Tobias Runge, Alexander Knüppel, Thomas Thüm, and Ina Schaefer. Lattice-Based Information Flow Control-by-Construction for Security-by-Design. In *FormaliSE*, FormaliSE '20, page 44–54, New York, NY, USA, 2020. ACM.

[RSC+19] Tobias Runge, Ina Schaefer, Loek Cleophas, Thomas Thüm, Derrick G. Kourie, and Bruce W. Watson. Tool Support for Correctness-by-Construction. In *FASE*, pages 25–42. Springer, 2019.

[RTC+20] Tobias Runge, Thomas Thüm, Loek Cleophas, Ina Schaefer, and Bruce W. Watson. Comparing Correctness-by-Construction with Post-Hoc Verification—A Qualitative User Study. In *Refine*, pages 388–405. Springer, 2020.

[SRK+18] Ina Schaefer, Tobias Runge, Alexander Knüppel, Loek Cleophas, Derrick Kourie, and Bruce W. Watson. Towards Confidentiality-by-Construction. In *ISoLA*, pages 502–515. Springer, 2018.

[WKSC16] Bruce W Watson, Derrick G Kourie, Ina Schaefer, and Loek Cleophas. Correctness-by-Construction and Post-hoc Verification: A Marriage of Convenience? In *ISoLA*, pages 730–748. Springer, 2016.