**Tool Support**

**for Correctness-by-Construction**

Summary by <team members>

Correctness-by-Construction is an approach for developing correct programs based on formal specifications. By using this methodology, the defect rate of a program decreases, and the bugs can be tracked easier. Also, the programs can be better trusted since the development follows a formal process.

**Open problem**

Despite the benefits of using the CbC approach, this method is not prevalent among programmers. A major reason for this is the fact that there is no tool that supports this way of development.

Another reason is that the programmers are more familiar with another related approach, post-hoc verification. In this approach, a method is verified against its pre- and post-conditions. On the other hand, the CbC approach offers the possibility for a method to be gradually verified (instruction after instruction) against the pre- and post-conditions. CbC is beneficial even in domains of the post-hoc verification and reduces the proof complexity since a task is divided into smaller problems.

**Solution/Method**

CorC is a tool designed to help programmers in developing programs using the CbC approach. It is built on a post-hoc verifier, KeY to profit from its ecosystem and future features. It is integrated into KeY in order to offer the possibility for developers to adopt this approach.

CorC offers a graphical and a textual editor that can work interchangeably. The textual editor is like a normal code editor, which offers support for pre- and post-condition specifications. In the graphical editor, the programmer can view the code and specifications in a tree-like structure.

The CorC tool is realized as an Eclipse plug-in in Java. The Hoare triple verification is implemented by the program verification tool KeY. Programs are written as Hoare specifications (pre- and post-conditions, different types of statements). CorC has 8 rules for developing programs: skip, assignment, composition, selection, repetition, weakening precondition, strengthening postcondition, and subroutine. The specifications are propagated through the program automatically. CorC supports integers, chars, strings, arrays, and subroutine calls. Hoare triples are verified by KeY automatically.

CbC starts with the specification of a program as a Hoare triple consisting of a pre-condition, abstract statement and a post-condition. This triple can be read as follows “if T is in a state

where the precondition holds and its abstract statement is executed, then the execution will terminate and the postcondition will hold”.

The CorC language uses Java as the host language. A Hoare triple is refined by applying rules, which introduce CorC statements. A *skip* statement does not alter the state of a program. *Composition* statement divides one statement into two. *Selection* statement can be compared to a switch. *Repetition* statement works like a while loop. The *weaken pre condition* statement can weaken the precondition of a method if the initial implies the new one. *Strengthen postcondition* can be used to strengthen a postcondition. A *subroutine* is used to divide a program into smaller parts.

**Example (case study)**

In order to prove the benefits of CorC, the following study was conducted. The CbC approach was compared with the post-hoc verification. In the study were used eight algorithms from the book by Kourie and Watson. The verification time and proof nodes needed by KeY to close both approaches were measured.

**Results**

7 out of the 8 tests worked without problems. The *binary search* algorithm could not be verified using post-hoc verification. In the case of measured proof nodes, only the *maximum element* algorithm needed slightly fewer nodes using post-hoc verification. The other algorithms had proofs from 3% to 854% smaller when CbC was used.

In the case of verification time, the CbC approach was always faster. For *maximum element, exponentiation, logarithm and factorial* the verification time was between 22% and 60% less than post-hoc verification. The *Dutch flag* and *linear search* had a 137% and 176% difference respectively. The *pattern matching* needed nearly 1 minute, while the post-hoc verifications needed 24 minutes.

**Conclusions**

CorC was implemented to support the CbC style of developing programs. This approach reduces the proof verification with respect to the post-hoc verification. CorC offers a textual and graphical editor that can be used interchangeably. It was built on KeY in order to profit from its ecosystem and the programmers can benefit from both approaches.