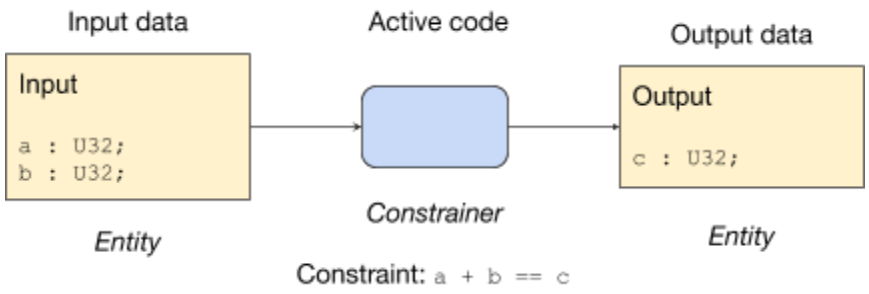
Master Thesis on Formal Verification Methods for the **TACC Framework**

Background

TACC is a programming language and framework that is widely used at Arista Networks to implement components of its Extensible Operating System (EOS). It is based on the *attribute-oriented* programming approach by Prof. David Cheriton. Its fundamental model of computation is based on defining and implementing *constrainers* which enforce constraints between input and output *data entities*. Changes in attributes of the input entities cause the activation of a *reactor* of the constrainer. A reactor is an imperative piece of code that modifies the attributes of output entities such that the constraint is re-established. For example, consider the following constrainer which may be used to enforce the constraint “a+b==c”. By declaring reactors to input attributes a and b, the constrainer can update output attribute c to re-establish the constraint.



The input and output relationships are described in the TACC language which the TACC compiler translates into C++ boilerplate code. Combined with reactor implementations that are written as C++ methods, a C++ compiler generates the final executable. A typical process in EOS consists of hundreds of distinct constrainers with dynamic lifetime.

Typical industry best practice embraced at Arista is to validate implementation correctness with extensive automated testing in a testing hierarchy, including unit tests, process-level, system-level and network-level integration tests. Software engineers are used to testing strategies, but the key drawback always remains that edge cases may be overlooked. This problem could be solved by applying formal verification.

Goals of the Thesis

The objective of this project is to explore applicability of formal verification methods to formally prove that constrainer implementations satisfy their specifications, for a relevant subset of the TACC language. The idea is to adapt or extend some of the standard formal verification tools

that have been developed in academia, for instance Frama-C and Why3, to analyze TACC code. A few simple examples as well as a real-world constrainer implementation together with their informal specifications will be provided by Arista, as well as an informal specification of relevant TACC semantics. Formal specifications will need to be formulated from those. At the end of the project, the given implementations will be formally proven correct against the formal specifications in a semi-automated way.

Relevant Background and Courses

Knowledge in compiler construction, formal methods, and logic will be required for this project. Those topics are partly covered in the following courses at Uppsala University; additional reading material will be provided to get a more complete background.

- 1DL321, Compiler Design I
- 1DT034, Programming Theory
- 1DL481, Algorithms and Data Structures III
- 1DL500, Automata and Logic (currently not given)

Practical Things

This project can be carried out by one or multiple students, and we will adapt the scope and project goals accordingly. Since the company Arista Networks is currently not able to host students, due to the Corona crisis, the work will be carried out at the IT Department, or by the students working from home. Supervision will be provided through regular meetings with supervisors at Arista Networks and at the IT Department:

- Contacts at IT Department: Zafer Esen (zafer.esen@it.uu.se) and Philipp Rümmer (philipp.ruemmer@it.uu.se)
- Contact at Arista Networks: Martin Stigge (mstigge@arista.com)

If you are interested in this project, please contact the above people; your application should include a CV.