Semantics and Compilation of Synchronous Dataflow Languages

Pierre-Évariste Dagand
pierre-evariste.dagand@lip6.fr
Cnrs – Inria Paris – Lip6

Abstract

This internship offers to specify the semantics and formalize the compilation of a synchronous dataflow language. It will take place in the Whisper team of Inria Paris – LIP6, located at University Paris 6, and will be supervised by Pierre-Évariste Dagand (Cnrs).

Synchronous dataflow languages [2; 6] are commonly used to implement *reactive* systems. Unlike usual software, a reactive system interacts directly with the physical world: it continuously receives inputs from its environment (*e.g.* through sensors), to which it must react by performing actions (*e.g.* through actuators). A dataflow language offers abstractions suited to manipulate and produce infinite streams of events. As a result, a synchronous dataflow program must go through a series of compilation passes before yielding an imperative program that can be efficiently executed on a computer.

Internship objectives: This project aims at developing a certified Lustre compiler in the Coq proof assistant [7], taking inspiration from existing synchronous dataflow compilers [1] and building upon the Compcert [4] certified C compiler. It encompasses the following aspects:

- Specifying a synchronous dataflow semantics in Coq;
- Implementing compilation passes as Coq programs, translating dataflow programs to imperative ones;
- Proving that the semantics of dataflow programs is preserved through compilation.

Being broad in scope, this project offers many opportunities for in-depth experiments as well as in-the-large developments, depending on the student's interest.

This project is part of on-going work with the Parkas team at ENS Ulm (Lélio Brun, Timothy Bourke, Marc Pouzet) and Lionel Rieg at Yale University. It is partially funded by the Émergence(s) program of the City of Paris, thanks to which we can offer a stipend ("gratification") for the duration of the internship.

Student's profile: Acquaintance with an interactive theorem prover (Coq, or Isabelle) is recommended. Nonetheless, a motivated student with a strong background in functional programming (OCaml, or Haskell) could certainly learn to use Coq along the way [5; 3]. No prior knowledge of a synchronous programming paradigm is expected: the development of the formal semantics in a proof assistant shall provide many opportunities to deepen one's understanding of the formalism.

References

[1] D. Biernacki, J.-L. Colaço, G. Hamon, and M. Pouzet. Clock-directed modular code generation for synchronous data-flow languages. In *Conference on Languages, Compilers, and Tools for Embedded Systems*, LCTES'08, pages 121–130, 2008. doi:10.1145/1375657.1375674.

[2] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. Lustre: A declarative language for real-time programming. In *Principles of Programming Languages*, POPL'87, pages 178–188, 1987. doi:10.1145/41625.41641.

[3] A. Chlipala. *Certified Programming with Dependent Types - A Pragmatic Introduction to the Coq Proof Assistant.* MIT Press, 2013. ISBN 978-0-262-02665-9. URL `http://mitpress.mit.edu/books/certified-programming-dependent-types`.

[4] X. Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009. URL `http://gallium.inria.fr/~xleroy/publi/compcert-CACM.pdf`.

[5] B. C. Pierce, C. Casinghino, M. Gaboardi, M. Greenberg, C. Hriţcu, V. Sjoberg, and B. Yorgey. *Software Foundations.* Electronic textbook, 2015. `http://www.cis.upenn.edu/~bcpierce/sf`.

[6] A. Technologies. Scade suite, 2015. URL `http://www.esterel-technologies.com/products/scade-suite/`.

[7] The Coq development team. *The Coq proof assistant reference manual.* LogiCal Project, 2014. URL `http://coq.inria.fr`. Version 8.4pl6.