Translation validation for synchronous data-flow equations in a Lustre compiler

Marc Pouzet
Marc.Pouzet@ens.fr

Francesco Zappa Nardelli
Francesco.Zappa_Nardelli@inria.fr

Location: Département d'Informatique, École Normale Supérieure, 45 rue d'Ulm, 75230 Paris cedex 05.

Prerequisite: interest in the design and implementation of programming languages, some experience with functional programming, familiarity with a proof assistant (Coq or Isabelle) is a plus.

Research Context The synchronous data-flow language SCADE is the de-facto standard to implement reactive systems in critical domains, including nuclear energy, avionics, railways, and automotive (http://www.esterel-technologies.com). The SCADE compiler is written in OCaml and is "qualified" with the highest safety requirements (norm DO-178C, level A). This certification is instrumental for the SCADE success but imposes a major overhead to implement even simple modifications of the compiler.

Objective In this internship we will explore some directions towards a formal certification of the SCADE compiler, aided by the appropriate tool support. In particular, we propose to design a *translation validation* strategy between some intermediate phases of the SCADE compiler.

The idea behind translation validation [6] is to check the equivalence between the source code and the generated code at compilation time. This approach has been applied for checking the correctness of the code generation of a discrete subset of Simulink [8] and extended to deal with optimising C compilers [5, 7, 9, 10].

Our aim is to realise an independent tool that checks the equivalence between some of the intermediate steps realized by the SCADE compiler. Compared to the existing work, this task involves novel challenges: the semantics of SCADE programs is radically different from the semantics of C programs, and the SCADE compiler can be seen as sequence of source-to-source transformation applied to an internal representation of *clocked data-flow equations* with a final and rather simple translation to sequential imperative code [1]. Our tool will work directly on the intermediate clocked data-flow representation, and the first step will be defining a normal form for clocked equations and related manipulation functions. We will then determine which information the compiler must provide to make the semantic check simple to implement and to formalise, while being reasonably efficient. Depending on the interest of the candidate, we might rely on external SMT solvers or implement, and prove correct, a dedicated checking function in a theorem prouver. Even a tool not entirely proved correct will increase the safety of the compiler and easy its maintenance with

respect to the certication authorities. To make the project maneagable, we might start on a simple Lustre compiler (Lustre can be considered the core language of SCADE), considering only later the fancy features of SCADE (including the type based clock calculus [3], the initialisation analysis [4], hierarchical automata [2]), etc).

Long term goal The ambitious, long-term, goal of this project would be to provide tools to get a formal certification (in the mathematical sense) of the industrial compiler of SCADE 6. The succesfull student can pursue this work with a PhD thesis in the PARKAS group, in collaboration with the compiler group of Esterel-Technologies, for instance extending the techniques to a production compiler such as the one of SCADE 6. Possibility of CIFRE funding.

References

[1] Darek Biernacki, Jean-Louis Colaco, Grégoire Hamon, and Marc Pouzet. Clock-directed Modular Code Generation of Synchronous Data-flow Languages. In *ACM International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, Tucson, Arizona, June 2008.

[2] Jean-Louis Colaço, Bruno Pagano, and Marc Pouzet. A Conservative Extension of Synchronous Data-flow with State Machines. In *ACM International Conference on Embedded Software (EMSOFT'05)*, Jersey city, New Jersey, USA, September 2005.

[3] Jean-Louis Colaço and Marc Pouzet. Clocks as First Class Abstract Types. In *Third International Conference on Embedded Software (EMSOFT'03)*, Philadelphia, Pennsylvania, USA, october 2003.

[4] Jean-Louis Colaço and Marc Pouzet. Type-based Initialization Analysis of a Synchronous Data-flow Language. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(3):245–255, August 2004.

[5] George C. Necula. Translation validation for an optimizing compiler. In *Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation*, PLDI '00, pages 83–94, New York, NY, USA, 2000. ACM.

[6] Amir Pnueli, Ofer Shtrichman, and Michael Siegel. The code validation tool CVT: Automatic verification of a compilation process. *International Journal on Software Tools for Technology Transfer*, 2(2):192–201, 1998.

[7] Xavier Rival. Symbolic Transfer Functions-based Approaches to Certified Compilation. In *31st Symposium on Principles of Programming Languages (POPL'2004)*, Venice, January 2004. ACM.

[8] Michael Ryabtsev and Ofer Strichman. Translation validation: From simulink to c. In *CAV*, pages 696–701, 2009.

[9] Jean-Baptiste Tristan and Xavier Leroy. Verified validation of lazy code motion. In *PLDI*, pages 316–326, 2009.

[10] Jean-Baptiste Tristan and Xavier Leroy. A simple, verified validator for software pipelining. In *POPL*, pages 83–92, 2010.