Towards a denotational semantics of streams for a verified Lustre compiler

Timothy Bourke, Paul Jeanmaire, Marc Pouzet

DI ENS, École normale supérieure, Université PSL, CNRS, INRIA, Paris, France

Vélus [1] is a formally verified compiler for the Lustre synchronous programming language. It is developed in Coq and uses the CompCert C compiler as a back-end. The correctness theorem links the dataflow semantics of the source language to the semantics of the generated assembly code. Its proof is a composition of the individual proofs of each compilation pass.

For Vélus it has been proved that repeated execution of the generated assembly code faithfully implements the dataflow semantics of source programs. To facilitate the compilation correctness proof, the choice was made to model the input language with a relational-style semantics, as shown in the following statement.

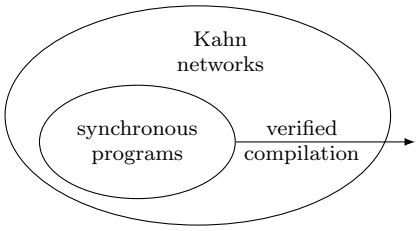
Theorem (compilation correctness, simplified)**.** *Given a Lustre node f, a list of input streams xs and a list of output streams ys, if f is successfully compiled to an assembly program P then:*

$$\mathsf{sem\_node}\ f\ xs\ ys \implies \exists T \in Traces(P),\ T \simeq (xs, ys).$$

Here the inductive predicate $\mathsf{sem\_node} : \mathsf{node} \to \mathsf{list\ Stream} \to \mathsf{list\ Stream} \to \mathsf{Prop}$ describes a relation whose elements are, for each node, the possible pairs of input and output streams. Since Lustre nodes denote deterministic stream functions, we were able to show that for all f, xs, ys and ys', $\mathsf{sem\_node}\ f\ xs\ ys \implies \mathsf{sem\_node}\ f\ xs\ ys' \implies ys \simeq ys'$.

While these definitions are well-tailored to establish compilation correctness, notably in the transition between dataflow and imperative languages, they do not give a procedure to build streams that satisfy the predicates. In particular, the determinism of nodes ensures that there is at most one possible output for a given input, but it does not guarantee the existence of such an output. Although unlikely, it could be the case that sem_node has no inhabitants, thus rendering void the main correctness theorem.

Since directly stating and proving the existence of a witness is very challenging due to the mutually recursive nature of equations in a Lustre node, it seems more appropriate to reason forward by defining a constructive interpretation of Lustre programs and then showing that the computed streams actually satisfy sem_node.



One possible approach is to consider the original definition of the language. The set of Lustre programs is naturally determined as a restricted class of Kahn networks [2] that can be executed *synchronously* and with bounded buffers. The ability to statically bound the required memory has long been exploited to design control software, especially in the certified development of safety-critical applications. In [3], Christine Paulin-Mohring describes how to

give a constructive denotational semantics to Kahn networks in Coq by means of a general library for CPOs, defining stream operations as the least fixed-points of continuous functions.

We are using this library to define a denotational semantics for Lustre. The aim is to provide a more natural front-end semantics for Vélus, closer to the one introduced in seminal articles [4]. There are two potential advantages to this approach. First, we believe it will facilitate the existence proof, because the the sem_node predicate is defined in a similar manner. A witness could also be constructed using a different style, for instance, using coiteration [5] which defines streams using iterated transition functions. We think, though, that it would be more difficult to relate the sequence global valuations so generated to the streams used in sem_node. There is also reason to believe that a denotational model à la Kahn may be the most convenient for interactively verifying Lustre programs that involve sampling since the absence of values is represented implicitly [6].

Mechanizing synchronous programs as Kahn networks in Coq challenges us to finely state the assumptions necessary to ensure that they compile correctly and execute safely. In the Kahn model, streams are built by iterating continuous stream functions, starting from the empty sequence. The first step is to ensure that the computed streams are indeed infinite, as required by sem_node. We are proving it by characterizing the class of *constructive* stream functions and exploiting a causality predicate required of source programs.

Finally, we obtain a denotation $[\![\cdot]\!]$ of Lustre components. For every node f, $[\![f]\!]$ is a total continuous function that maps infinite input streams to infinite output streams that may contain error values. We aim to show that these errors only arise from runtime exceptions (division by zero, integer overflow, etc.) which cannot be detected statically. We conjecture that if no such error occurs in output or local streams, then the relational predicate sem_node holds.

Conjecture (coherence of the relational semantics)**.** *Given a Lustre node f and a list of input streams xs, if $[\![f]\!](xs)$ is exempt of runtime errors then:*

$$\mathsf{sem\_node}\ f\ xs\ ([\![f]\!](xs)).$$

References

[1] T. Bourke, L. Brun, P.-E. Dagand, X. Leroy, M. Pouzet, and L. Rieg, "A formally verified compiler for Lustre," in *PLDI 2017 - 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, (Barcelone, Spain), ACM, June 2017.

[2] G. Kahn, "The semantics of a simple language for parallel programming," in *Information Processing, Proceedings of the 6th IFIP Congress 1974, Stockholm, Sweden, August 5-10, 1974* (J. L. Rosenfeld, ed.), pp. 471–475, North-Holland, 1974.

[3] C. Paulin-Mohring, "A constructive denotational semantics for Kahn networks in Coq," in *From Semantics to Computer Science* (Y. Bertot, G. Huet, J.-J. Lévy, and G. Plotkin, eds.), pp. 383–413, Cambridge University Press, 2009.

[4] P. Caspi and M. Pouzet, "Synchronous Kahn networks," *Proceedings of the ACM SIGPLAN International Conference on Functional Programming, ICFP*, vol. 31, 08 2001.

[5] P. Caspi and M. Pouzet, "A co-iterative characterization of synchronous stream functions," in *First Workshop on Coalgebraic Methods in Computer Science (CMCS'98)*, vol. 11 of *ENTCS*, (Lisbon, Portugal), pp. 1–21, Elsevier Science, Mar. 1998.

[6] C. Canovas-Dumas and P. Caspi, "A PVS proof obligation generator for Lustre programs," in *Logic for Programming and Automated Reasoning, 7th International Conference, LPAR 2000, Reunion Island, France, November 11-12, 2000, Proceedings* (M. Parigot and A. Voronkov, eds.), vol. 1955 of *Lecture Notes in Computer Science*, pp. 179–188, Springer, 2000.