Clock Typing of n-Synchronous Programs

Louis Mandel Florence Plateau Marc Pouzet
Université Paris-Sud 11 and INRIA Saclay

Synchronous functional languages such as Lustre or Lucid Synchrone define a restricted class of Kahn Process Networks which can be executed without buffers. This condition is ensured by a dedicated type system, the clock calculus. Every stream is associated to a *clock* defining the instants where the stream is present. Every expression must in turn verify a type constraint such as:

$$\frac{H \vdash e_1 : ck_1 \mid C_1 \quad H \vdash e_2 : ck_2 \mid C_2}{H \vdash e_1 + e_2 : ck_3 \mid \{ck_1 \ = \ ck_2 \ = \ ck_3\} \cup C_1 \cup C_2}$$

which states that under the typing environment H, $e_1 + e_2$ has clock ck_3 if e_1 has clock ck_1, e_2 has clock ck_2 and $ck_1 \ = \ ck_2 \ = \ ck_3$. Constraints C_1 and C_2 are gathered during typing. Synchronous languages only consider equality constraints. An expression is well typed if its actual clock equals its expected clock and this means that no buffer will be necessary to store or delay it. n-Synchrony [1] relaxes these constraints by allowing to compose streams whose clocks are not equal but can be synchronized through the introduction of bounded buffers. It is obtained by extending the clock calculus with a *subtyping* rule which defines points where a buffer should be inserted. If a stream x with clock ck can be consumed later on a clock ck' using a bounded buffer, we shall say that ck is a subtype of ck' and we shall write $ck \ <: \ ck'$.

$$\frac{H \vdash e : ck \mid C}{H \vdash \texttt{buffer}(e) : ck' \mid \{ck \ <: \ ck'\} \cup C}$$

In term of sequence of values, $\texttt{buffer}(e)$ is equivalent to e but it may delay its input using a bounded buffer. The purpose of the extended clock calculus is to compute this bound. The designer can write $\texttt{buffer}(e)$ everywhere in the program as potential places where a buffer can be inserted. Then, the clock calculus automatically computes bounds for these buffers.

n-Synchrony can be defined for any language of clocks provided we are able to test equality ($ck_1 \ = \ ck_2$), subtyping ($ck_1 \ <: \ ck_2$) and $\texttt{size}(ck_1, ck_2)$ to compute the buffer to synchronize ck_1 and ck_2. An interesting case is the one where clocks are restricted to be *ultimately periodic binary sequences* for which all the above properties are decidable.

Last year, we presented how to abstract clocks in order to check the subtyping relation in an efficient manner [2]. This year, we shall present Lucy-n, the first implementation of a n-synchronous programming language. In this talk, we will describe the language as an extension of Lustre. Then, we will explain its clock calculus and the constraints resolution algorithm. Finally, we will illustrate it on a multimedia application.

References

[1] A. Cohen, M. Duranton, C. Eisenbeis, C. Pagetti, F. Plateau, and M. Pouzet. *N*-Synchronous Kahn Networks: a Relaxed Model of Synchrony for Real-Time Systems. In *ACM International Conference on Principles of Programming Languages*, January 2006.

[2] A. Cohen, L. Mandel, F. Plateau, and M. Pouzet. Relaxing synchronous composition with clock abstraction. In *Hardware Design using Functional languages*, pages 35–52, York, UK, 2009.