Synchronous Data-flow Programming in ML

M. Pouzet
LIP6 *

Synchronous data-flow languages like LUSTRE have been specially developed for programming real-time systems. These languages manage time indexed sequences which are used for representing input/outputs signals of a real-time system. The synchronous aspect of LUSTRE and its first-order restriction guaranty that programs can be compiled into finite automata. Since LUSTRE is a functional stream language, it is natural to investigate the links with general purpose functional languages in order to extend it with notions of higher-order, type polymorphism and data-structures, that is, to incorporate some classical features of ML languages or, in a more pragmatic way, to integrate the synchronous data-flow paradigm in an existing ML language.

In this talk, we shall discuss about the problem of embedding LUSTRE inside a general purpose ML language, while keeping at the same time the efficiency due to synchrony and the expressiveness of ML. We show that such an integration calls for specific static analysis and compilation methods which must be extended to deal with higher-order features. This discussion is done with the support of LUCID SYNCHRONE, an ML extension of LUSTRE. LUCID SYNCHRONE is built above OBJECTIVE CAML, keeping its syntax and run-time system. It manages functions as first-class citizens, it has type and clock inference, a simple module system and programs are compiled into transition systems.

In the first part of the talk, we present some typical examples of programs written in LUCID SYNCHRONE, illustrating the use of type polymorphism, higher-order and the communication with the host language OBJECTIVE CAML.

In the second part, we present the semantics and implementation of the language. We show that an encoding of streams in term of lazy data-structures leads to a very inefficient execution and does not preserve reactivity as soon as sampling functions are considered. Thus, we present the two main techniques used in the compiler:

- a static analysis, named a *clock calculus* whose purpose is to reject non synchronous programs. This calculus is based on an inference type system and may be applied to higher-order programs.
- a compilation method producing some imperative OBJECTIVE CAML code. This method is based on a characterization of synchrony in term of co-iteration and it generalization to higher-order.

This work has been done in collaboration with Paul Caspi from VERIMAG, Grenoble. Informations about LUCID SYNCHRONE can be found at `www-spi.lip6.fr/lucid-synchrone`.

*Laboratoire d'Informatique de Paris 6, Equipe SPI, Case 168, 4 place Jussieu, 75252 Paris Cedex 05, France. E-mail: `Marc.Pouzet@lip6.fr`. Web: `www-spi.lip6.fr`