Timing constraints and symbolic execution for a hybrid synchronous language

Timothy Bourke and Marc Pouzet

November 1, 2013

Recent work in the Inria/ENS team Parkas, in collaboration with Esterel Technologies, has resulted in a compiler for a language called Zélus that combines discrete Lustre-like programs [7] with continuous dynamics modelled as Ordinary Differential Equations (ODEs) with resets. Such hybrid systems modellers are used not only in the high-level design and simulation of complex embedded systems, but also as development platforms in which the same source is used for formal verification, testing, simulation, and the generation of target executables. The quintessential example is the Simulink/Stateflow suite which is widely used in Industry. Zélus is distinguished by a novel type system [3], a completely integrated treatment of both discrete and continuous automata [4], and a semantic theory based on non-standard analysis [5]. A compiler [6] and example programs are available at http://zelus.di.ens.fr.

An ODE with reset is written in Zélus as an equation:

\[
\text{der } x = e \text{ init } e_i \text{ reset } z_1 \rightarrow e_1 | z_2 \rightarrow e_2
\]

which can be seen as a very simple hybrid automaton:

At present such behaviours are simulated by approximation with Runge-Kutta algorithms and external numeric solvers [9]. Triggering events, i.e., \( z_1 \) and \( z_2 \), have been limited to zero-crossing functions or discrete signals emitted by other parts of a program.

This project considers equations of a restricted form:

\[
\text{der } x = 1.0 \text{ init } 0.0 \text{ reset } z_1 \rightarrow 0.0 | z_2 \rightarrow 0.0
\]

where all continuous variables have a slope of 1 and may only be reset to 0, and where triggering events are combinations of discrete signals and constraints of the forms ‘\( x_1 R c’ \) or ‘\( x_1 - x_2 R c’ \), where \( R \in \{<,\leq,=,\geq,>\} \), \( x_1 \) and \( x_2 \) are continuous variables defined with \text{der}, and \( c \) is a constant in \( \mathbb{Q}_{\geq 0} \). These restrictions are more or less those of Timed Automata [1], to which ‘location invariants’ can also be added to express necessity and

\[1\text{http://www.mathworks.com/products/simulink/}\]
urgency [8]. Data structures, called Difference Bound Matrices (DBMs), are able to represent the states of such models symbolically. They are used in tools like Uppaal [2] to generate symbolic simulations and to perform model checking.

In this project, we propose extending the Zélus language, type system, compilation chain, and run-time to incorporate features of timed automata and to generate symbolic traces—i.e., equivalence classes of clock values having the same potential branching behaviour—of models which satisfy the constraints described above. This work will involve questions of language design: a) How should such timing constraints and concomitant non-determinism be integrated into a dataflow language? b) How should the type system be adjusted to identify programs that can be treated in this way? c) Which programming paradigms are the most convenient for modelling such systems? And also questions of implementation: a) Are existing DBM libraries suitable for use in Zélus? b) How should they be integrated into a run-time that will eventually need to combine several different back-ends for simulation and verification? c) What is the best way to present simulation results and to resolve non-determinism? Finally, it will be necessary to model typical timed models in the extended language and to test them in the prototype system.

Besides its intrinsic scientific interest, the ideas and implementation developed during this project will contribute to a larger project on modelling and implementing real-time embedded systems. Furthermore, we believe that this approach has several potential practical advantages. The instantiation of processes and use of parallel composition is restricted in Uppaal to the top-level of a program, whereas Zélus allows much more liberal combinations and thus potentially more satisfying models. Uppaal is a rich modelling language that combines graphical automata and a rich imperative language for specifying effects. Such a sophisticated system may itself contain bugs. We think that the architecture of source-to-source transformations applied in modern dataflow compilers minimises the risk of such bugs and makes for a good target for eventual verification. Also, the causality and other restrictions of dataflow program languages help programmers to create correct models by imposing strong disciplines of definition and bounded behaviour.

References


