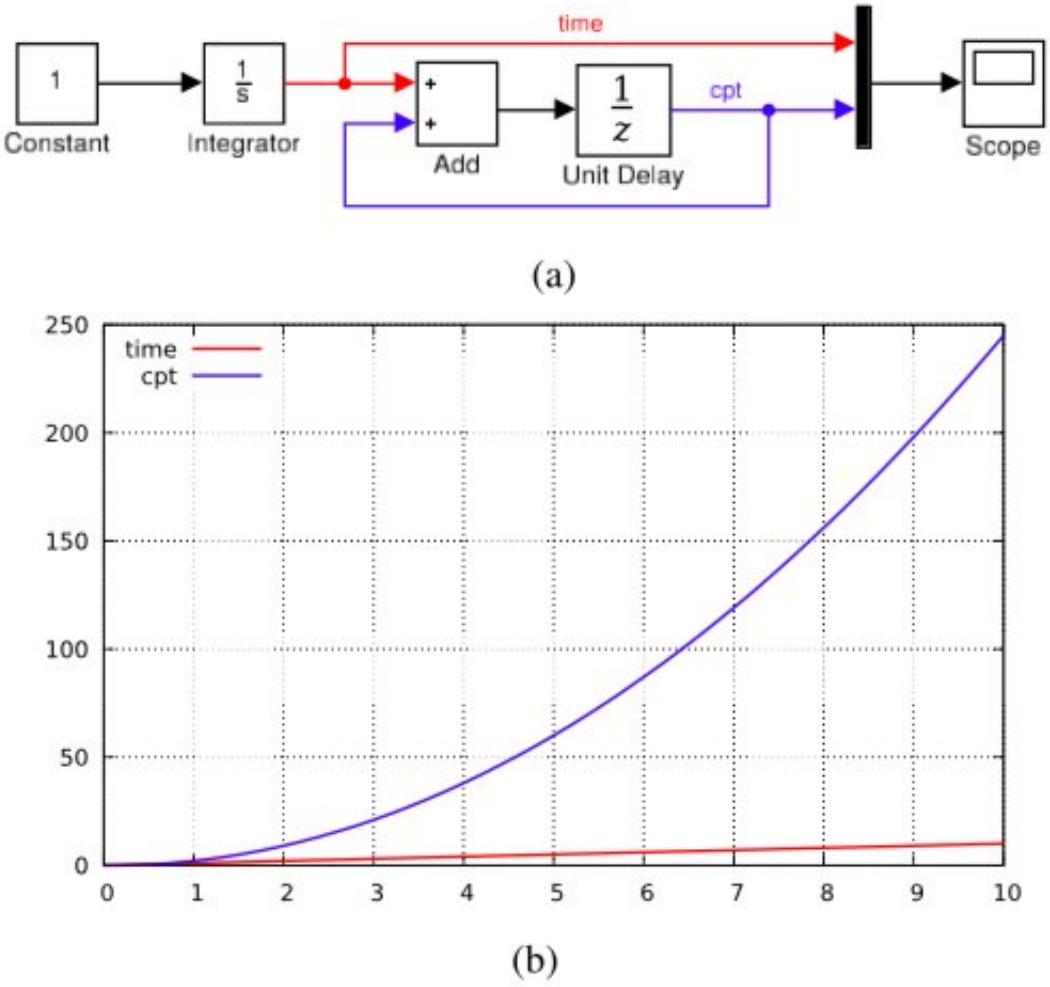
A new approach for designing and implementing a hybrid systems language

November 28 2018, by Ingrid Fadelli



Composition of models that mix discrete- and continuous-time blocks in Simulink (R2016b). (a) Basic model. (b) Simulation of basic model. Credit: Benveniste et al.

Hybrid systems are systems that exhibit both continuous and discrete dynamic behavior, allowing more flexibility in modeling dynamic phenomena. Hybrid systems modeling languages are widely used for the development of cyber-physical systems, in which control software interacts with physical devices.

Researchers at Inria and ANSYS/Esterel Technologies have recently presented a new approach to design and implement hybrid systems languages. Their method, outlined in a paper in *Proceedings of the IEEE*, is based on synchronous language principles and associated compilation techniques.

Hybrid system modeling tools have evolved from being mere interfaces to numeric solvers, then became fully fledged languages for programming executable models of dynamic systems. These models are generally simulated, tested, debugged and verified at different stages of their development chain.

In state-of-the-art methods, compilers typically check source models, produce intermediate representations and generate sequential code for either efficient simulation or execution on target platforms. However, these compilation steps are often difficult to design and implement.

The recent study focused on the design, semantics and implementation of hybrid systems modeling languages. It is based on the assumption that such languages are programming languages with hybrid systems semantics, hence presenting a number of new challenges.

"The bottom line is that the complexity of actual hybrid systems modeling languages makes the definition of a comprehensive formal static and dynamic semantics difficult to achieve," the researchers write in their paper. "Far from being abstract philosophical concerns, these difficulties have practical consequences."

To address these challenges, the researchers set out to identify a minimal language kernel of orthogonal programming constructs that is expressive enough to write realistic hybrid models. They also wished to define detailed static and dynamic semantics of this language, as well as its compilation steps.

"The result is a hybrid systems modeling language in which synchronous programming constructs can be mixed with ordinary differential equations (ODEs) and zero-crossing events, and a runtime that delegates their approximation to an off-the-shelf numerical solver," the researchers explain in their paper. "We propose an ideal semantics based on nonstandard analysis, which defines the execution of a hybrid model as an infinite sequence of infinitesimally small time steps."

The semantics framework proposed by the researchers can be used to specify and prove three essential compilation steps. First, it leads to a type system that guarantees that a continuous-time signal is never used in situations where a discrete-time signal is expected, and vice versa. In addition, it ensures the absence of combinatorial loops, as well as the generation of statically scheduled code for efficient execution.

"Our approach has been evaluated in two implementations: the academic language Zélus, which extends a language reminiscent of Lustre with Odes and zero-crossing events, and the industrial prototype Scade Hybrid, a conservative extension of Scade 6," the researchers write in their paper.

Compared to other tools and languages, such as Ptolemy, the approach used by the researchers favors the detection of unsafe models at compile time. The consequence of this is that some good models are rejected, mainly because the resulting type systems are not expressive enough. Further experimental studies could help to determine whether these type systems are overly constraining.

"The discovery of numerical difficulties is related to stiffness remains runtime, and rules out the need for overly restrictive programming disciplines in industrial contexts," the researchers write in their paper. "Performing rich analyses at compile time, while constraining the users, may detect errors in models early; it also allows for removing runtime checks and to statically schedule the computation of the step function and the reset actions, which leads to more efficient code."

More information: Albert Benveniste et al. Building a Hybrid Systems Modeler on Synchronous Languages Principles, *Proceedings of the IEEE* (2018). DOI: 10.1109/JPROC.2018.2858016