A Synchronous-Based Code Generator for Explicit Hybrid Systems Languages*

Timothy Bourke¹,³, Jean-Louis Colaço², Bruno Pagano², Cédric Pasteur², and Marc Pouzet⁴,³,¹

¹ INRIA Paris-Rocquencourt
² ANSYS/Esterel-Technologies, Toulouse
³ DI, École normale supérieure, Paris
⁴ Université Pierre et Marie Curie, Paris

Abstract. Modeling languages for hybrid systems are cornerstones of embedded systems development in which software interacts with a physical environment. Sequential code generation from such languages is important for simulation efficiency and for producing code for embedded targets. Despite being routinely used in industrial compilers, code generation is rarely, if ever, described in full detail, much less formalized. Yet formalization is an essential step in building trustable compilers for critical embedded software development.

This paper presents a novel approach for generating code from a hybrid systems modeling language. By building on top of an existing synchronous language and compiler, it reuses almost all the existing infrastructure with only a few modifications. Starting from an existing synchronous data-flow language conservatively extended with Ordinary Differential Equations (ODEs), this paper details the sequence of source-to-source transformations that ultimately yield sequential code. A generic intermediate language is introduced to represent transition functions. The versatility of this approach is exhibited by treating two classical simulation targets: code that complies with the FMI standard and code directly linked with an off-the-shelf numerical solver (Sundials CVODE).

The presented material has been implemented in the ŽELUS compiler and the industrial SCADÉ Suite KCG code generator of SCADÉ 6.

1 Introduction

Hybrid systems modeling languages allow models to include both software and elements of its physical environment. Such models serve as references for simulation, testing, formal verification, and the generation of embedded code. Explicit hybrid systems languages like SIMULINK/STATEFLOW¹ combine Ordinary Differential Equations (ODEs) with difference and data-flow equations, hierarchical automata in the style of Statecharts [15], and traditional imperative features.

* Examples in ŽELUS and the extension of SCADÉ 6 with hybrid features are available at http://zelus.di.ens.fr/cc2015/
http://mathworks.org/simulink

¹ Springer-Verlag Berlin Heidelberg 2015
DOI: 10.1007/978-3-662-46663-6_4
Models in these languages mix signals that evolve in both discrete and continuous time. While the formal verification of hybrid systems has been extensively studied, this paper addresses the different, but no less important, question of generating sequential code (typically C) for efficient simulations and embedded real-time implementations.

Sequential code generation for synchronous languages like Lustre has been extensively studied. It can be formalized as a series of source-to-source and traceable transformations that progressively reduce high-level programming constructs, like hierarchical automata and activation conditions, into a minimal data-flow kernel. This kernel is further simplified into a generic intermediate representation for transition functions, and ultimately turned into C code. Notably, this is the approach taken in the Scade Suite KCG code generator of Scade, which is used in a wide range of critical embedded applications.

Yet synchronous languages only manipulate discrete-time signals. Their expressiveness is deliberately limited to ensure determinacy, execution in bounded time and space, and simple, traceable code generation. The cyclic execution model of synchronous languages does not suffer the complications that accompany numerical solvers. Conversely, a hybrid modeling language allows discrete and continuous time behaviors to interact. But this interaction together with unsafe constructs, like side effects and while loops, is not constrained enough, nor specified with adequate precision in tools like Simulink/Stateflow. It can occasion semantic pitfalls and compiler bugs. A precise description of code generation, that is, the actual implemented semantics, is mandatory in safety critical development processes where target code must be trustworthy. Our aim, in short, is to increase the expressiveness of synchronous languages without sacrificing any confidence in their code generators.

Benveniste et al. recently proposed a novel approach for the design and implementation of a hybrid modeling language that reuses synchronous language principles and an existing compiler infrastructure. They proposed an ideal synchronous semantics based on non standard analysis for a Lustre-like language with ODEs, and then extended the kernel language with hierarchical automata and a modular causality analysis. These results form the foundation of Z´elus. This paper describes their validation in an industrial compiler.

Paper Contribution and Organisation Our first contribution is to precisely describe the translation of a minimal synchronous language extended with ODEs into sequential code. Our second contribution is the experimental validation in two different compilers: the research prototype Z´elus and the Scade Suite KCG code generator. In the latter it was possible to reuse all the existing infrastructure like static checking, intermediate languages, and optimisations, with little modification. The extensions for hybrid features require only 5\% additional lines of code. Moreover, the proposed language extension is conservative in that regular synchronous functions are compiled as before—the same synchronous code is used both for simulation and for execution on target platforms.

\[2\] \texttt{http://www.esterel-technologies.com/products/scade-suite/}