Science of Computer Programming 77 (2012) 1151–1177

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

Refinement-oriented models of Stateflow charts

Alvaro Miyazawa *, Ana Cavalcanti

Department of Computer Science, The University of York, York, YO10 5GH, United Kingdom

ARTICLE INFO

Article history:
Available online 3 August 2011

Keywords:
Simulink
Circus
Formal semantics
Verification
Tools

ABSTRACT

Simulink block diagrams are widely used in industry for specifying control systems, and of particular interest and complexity are Stateflow blocks, which are themselves defined by separate charts. To make formal reasoning about diagrams and charts possible, we need to formalise their semantics; for the formal verification of their implementations, a refinement-based semantics is appropriate. An extensive subset of Simulink has been formalised in a language for refinement, namely, *Circus*, and here, we propose an approach to cover Stateflow charts. Our models are distinctive in their operational nature, which closely reflects the informal description of the Stateflow (simulation) semantics. We describe, formalise, and automate a strategy to generate our *Circus* models. The result is a solid foundation for reasoning based on refinement.

1. Introduction

MATLAB Simulink [1] is a graphical notation that supports the specification of control systems at a level of abstraction convenient for engineers; it is, for example, widely used in the avionics and automotive industries. Stateflow [2] is part of Simulink, and consists of a statechart notation used to define Simulink blocks. Simulink diagrams are typically used to specify aspects of a system that can be modelled by differential equations relating inputs and outputs. On the other hand, Stateflow charts frequently model the control aspects, like, for instance, changes in modes of operations triggered by events and conditions. Both Simulink and Stateflow provide extensive tool support to handle diagrams; there are facilities for simulation and analysis [1,2], verification, validation and testing [3,4], code generation [5,6], and prototyping [7].

Many of the systems specified and designed using Simulink diagrams and Stateflow charts are safety-critical systems, and various certification standards [8,9] recommend the use of formal methods for the specification, design, development and verification of software. This suggests that formal techniques that support graphical notations like Simulink and Stateflow are extremely useful, if not essential.

We are concerned with the assessment of the correctness of implementations of Stateflow charts: we want to be able to assert that a program correctly implements a chart. This has been frequently dealt with by approaches based on the verification of automatic code generators [10–12]. The approach that we pursue is orthogonal, and can be used in situations where code generators are not applicable or convenient. For instance, frequent updates to the generator have a heavy impact on the cost of its verification. In addition, customised hardware or performance requirements often impose the need for changes in code generated.

In this paper, we provide a formal semantics of Stateflow charts suitable for reasoning based on refinement. It is written in a way that facilitates validation, and integration of models of Simulink diagrams. With this, we set the foundation not only for analysing complex diagrams, but also for verifying the correctness of systems specified using both standard Simulink blocks and Stateflow charts. Although we do not tackle program verification here, the models that we present can be

* Corresponding author.
E-mail addresses: alvarohm@cs.york.ac.uk, alvarohm@gmail.com (A. Miyazawa), Ana.Cavalcanti@cs.york.ac.uk (A. Cavalcanti).

0167-6423/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2011.07.007

the starting point, for instance, to extend the refinement-based verification technique for implementations of Simulink diagrams presented in [13].

Our semantics of Stateflow charts is based on the *Circus* notation [14], which is a refinement language that combines Z [15], CSP [16], Dijkstra's language of guarded commands [17], and Morgan's specification statement [18]. *Circus* has a formal semantics given in terms of the Unifying Theories of Programming [19], and a refinement strategy that integrates different theories of refinement (action and process refinement). *Circus* supports the abstract specification of state-rich reactive systems and provides a refinement calculus [20] that allows the verification of implementations. It is supported by various tools, such as a type-checker [21], a refinement editor [22], a translator from *Circus* to Java [23], and a theorem prover [24].

Cavalcanti et al. [25] define a semantics for Simulink diagrams in *Circus*. It builds upon the Z-based approach described in Arthan et al. [26], and Adams and Clayton [27], and extends it to cover a larger subset of Simulink, but not Stateflow blocks. Therefore, a *Circus* model of Stateflow charts is a natural extension of previous work, allowing for the verification of a broader variety of control law diagrams. By using *Circus* as a specification language, we provide support to reason formally about the model, to verify code of proposed implementations, and to integrate the model with existing models of Simulink diagrams.

As far as we know, our modelling technique is unique in that it covers a wide range of Stateflow constructs that have not been treated before, such as history junctions, unrestricted transitions, and multi-dimensional data. Additionally, the models are specified in a formal notation that has been extensively used for the verification of programs, namely *Circus*. It is possible to apply the *Circus* refinement calculus to reason about our models and their implementations. Finally, our models can be integrated with *Circus* models of Simulink diagrams. All this caters for a very comprehensive coverage of the Simulink/Stateflow notation.

Because the established semantics of Stateflow is only available through simulation or in an informal description in the *Stateflow User's Guide* [2], we provide a formal model based on the expected behaviour of charts during simulation. There is no way of formally comparing our model to the semantics encoded in the simulator (without access to the simulator's code), and this is a problem inherent to the formalisation of any language that does not have a well established formal model already.

In order to circumvent this issue, we have validated our model through alternative approaches. We have used inspection: systematic comparison of our formal model to the informal descriptions found in the Stateflow documentation. In particular, to facilitate validation, our model is constructed in a way that provides a direct correspondence with these descriptions. We have also used simulation: comparing traces of the simulation tool to traces of our models. Defining and implementing rules to translate Stateflow charts to *Circus*, and applying them to case studies, has also provided further validation. Finally, we are currently applying the *Circus* refinement calculus to verify chart implementations based on our model; this effort validates the modelling technique and its appropriateness for reasoning.

The main contributions of this work are an approach to modelling Stateflow charts using a state-rich process algebra like *Circus*, the formalisation of a strategy to translate Stateflow to *Circus* models, and the *s2c* tool that implements this strategy and generates *Circus* models automatically. These models are partitioned in two components that capture separately the semantics of Stateflow charts and the structure of a particular chart. The formalisation of the translation strategy is encoded in Z and consists of a formal syntax of Stateflow charts, well-formedness conditions, an embedding of *Circus* in Z, and a set of translation rules defined as functions from well-formed elements of the syntax of Stateflow charts to elements of the *Circus* notation. The *s2c* tool takes a textual representation of one or more charts, and automatically produces a *Circus* specification containing the corresponding models.

This article is structured as follows. Section 2 introduces the Stateflow and *Circus* notations. Section 3 presents our approach to construct formal models of Stateflow charts; Section 4 introduces the formalisation of the translation rules; Section 5 discusses the implementation of the translation rules in the *s2c* tool, and describes the validation of the proposed approach. Section 6 reviews our contributions, discusses the limitations of our work, examines related work, and proposes future directions for this work.

2. Preliminaries

In this section, we introduce the notations that form the basis for this work: Stateflow and *Circus*.

2.1. Stateflow

A Simulink diagram consists of blocks and wires connecting the inputs and outputs of the blocks. Its execution is carried out in a cyclic fashion, in which the blocks are executed in an order determined by the wiring of the diagram, and at a time determined by simulation time steps. Stateflow provides a new type of Simulink block, namely a Stateflow chart. This is a graphical notation that supports the specification of state transition systems. It is a variant of Harel's statecharts [28], which extends standard state-transition systems by introducing a number of features, such as hierarchy and parallelism.

Fig. 1 shows a Stateflow chart adapted from an example supplied with the Simulink/Stateflow tool; it is part of the model of an automatic transmission controller. This chart specifies a system that monitors the speed of a vehicle, and changes gears appropriately. To decide whether or not to change gears, it calculates upper and lower thresholds based on the current gear and the engine throttle.

A Stateflow chart is the root for the execution of the corresponding Stateflow model; it encapsulates the states, transitions, junctions, functions, data and events that characterise it. The only means of interaction with the Simulink