A continuous-time approach to modelling and validating Simulink Models

Chunqing Chen [*]

School of Computing
National University of Singapore
{chenchun}@comp.nus.edu.sg

Abstract. Our research focuses on applying formal methods to elevate the design quality of Simulink. As a modelling and simulation tool, Simulink is deficient when coping with the increasing requirements of high-level assurance and timing analysis. We propose a systematic approach to translate Simulink models to Timed Interval Calculus (TIC), a continuous-time based formal language for specifying and reasoning about real-time systems. The translation preserves the functional and timing aspects of Simulink models. Our approach can increase the design space by specifying timing related requirements, in particular safety and liveness, exactly on the model after the translation. Using TIC reasoning rules, we can formally validate that the design satisfies the requirements.

1 Introduction

Simulink is a graphical environment for modelling and simulating various systems. It adopts the continuous-time semantics, and discrete-time models behave piecewise constantly continuously. A Simulink model is constructed by connecting blocks with wires. The simulation characterizes the system behavior for particular circumstances.

Despite the efficiency of simulation in system design, formal verification is foreseen to complement the simulation to cope with the increasing requirements of high-level assurance. Moreover, timing analysis becomes necessary, though Simulink is currently difficult to specify and check timing constraints. We propose to apply a real-time formal notation, Timed Interval Calculus (TIC) [6] to elevate the design quality of Simulink.

TIC is based on the set theory and reuses the Z mathematical and schema notations. It utilizes continuous functions of time to model systems and contains a rich set of reasoning rules for verification. The similar notations are Duration Calculus with its extensions that describe system behavior without explicit references of absolute time points, so they are inappropriate to represent constraints which restrict the values of interval endpoints to specific intervals.

We aim at taking advantage of TIC to rigorously model and formally validate Simulink models. The approach is based on the same angle chosen by TIC and Simulink where they model systems in terms of continuous time. Due to the informal description of Simulink library blocks, we focus on capturing the mathematical functions denoted by the blocks (as their denotational semantics) in TIC. To ensure the consistency, we develop a strategy to preserve the functional and timing aspects of Simulink models in

[*] Supervised by Professor DONG Jin Song

the translation. The approach can increase the design space by allowing timing related requirements, especially safety and liveness, to be specified exactly on the model using the translated TIC specifications. With all the information represented in TIC, we can formally prove that the design meets the requirements by deduction. Optimizing the automation in the translation and reasoning processes is one of the goals as well.

2 Approach and proposed solutions

The approach consists of three steps.

Defining TIC library: Simulink library blocks are templates to generate elementary blocks, the units of Simulink models, by the parameterization technique. Correspondingly, we define a TIC library function to model a library block. The function accepts a set of arguments and returns a TIC schema. In this way, the arguments relate to the parameters of the library block, for example, the *sample time*, and the TIC schema characterizes the generated elementary block by declaring the block's inputs and outputs as total functions over time, as well as describing the block's the mathematical function in terms of time intervals. So far, we have rigorously modelled 25 often used library blocks of 8 categories including *continuous, discrete* and *discontinuous* libraries. We also identified that the library blocks of *Ports and Subsystems* category cannot be expressed precisely in this step and they can be handled in the translation step.

Translating Simulink models: Simulink models are constructed in a hierarchical way, so the translation is bottom up. Each elementary block is translated into a TIC schema that represents the functional behavior of the block. The schema is generated by applying a TIC library function to relevant block parameter values in Simulink. The primary parameter *BlockType* and the operator parameters compose the criteria to choose the proper TIC library. Each wire is depicted by an equivalence relationship between two continuous-time functions denoting the interface of blocks. The equivalence captures the Simulink communication feature, i.e. the destination block receiving the same value produced by the source block simultaneously. Each (sub)system is expressed by a TIC schema. The schema preserves the (sub)system structure by declaring its components as state variables and describing the connection as a set of equations. To keep the timing aspect, we developed an algorithm to derive sample time values of elementary blocks in particular models. Moreover, the conditionally executed subsystems in Simulink are taken into account. Currently, we have implemented the translation strategy using JAVA and successfully translated a continuous system and a hybrid system. [1]

Validating Simulink models: Based on the generated TIC specifications, timing related requirements can be formalized easily on the system or a component. We have checked 9 requirements that cover *safety* and *liveness* of two non-trivial systems mentioned above. During the validation, we start with checking properties of subsystems, and the proved properties can hence act as lemmas for more complicated analysis of the higher-level system. In addition, the experiments show that open systems which are not checkable in Simulink can be verified in our approach. Furthermore, using common mathematical analysis, e.g. control theory, freely in TIC logic eases the proof.

[1] Details are available at: `www.comp.nus.edu.sg/~chenchun/report`

3 Related Work

Recently, there are a number of excellent works on translating Simulink into other formal notations or programming languages. [2] and [1] translate Simulink models into Z by specifying the functional behavior of one cycle. [4] extends the work by using Circus to capture functionality and concurrency of Simulink models. Their approaches aim to verify that Simulink models are correctly implemented in the programming language, and that is different from ours. Our goal is to validate that Simulink models satisfy different requirements. Moreover, they currently consider single-rate discrete systems, and the timing information is missing. Similarly, [3] focuses on only discrete Simulink blocks. [7] develops a tool CheckMate that can automatically verify customized Simulink models against the requirements that are in the format of linear inequality. It lacks of support to certain requirements that are represented by inequalities containing variables on both sides, though such requirements can be handled in ours. In short, our approach can cover a wider range of Simulink blocks, provide more accurate specifications and support more complex requirements.

4 Conclusion and future work

This paper describes the first attempt to apply a continuous-time formal specification language TIC to model and validate Simulink models. The approach is based on the similarity of two notations where they model systems in terms of continuous time. With the expressive power and the verification capability of TIC, we can increase the design space and elevate the confidence of Simulink models. The early experiment results demonstrate the feasibility of the approach. We list some future work below.

Currently we are extending the TIC library to cope with more library blocks of different categories. Supporting Stateflow is one of our goals as well. We intend to improve the algorithm to handle the sample time propagation in conditionally executed subsystems and examine the translator with more complex systems. So far, the validation is accomplished manually. We are currently studying the work [5] which formalized some TIC reasoning rules in the generic theorem prover Isabelle. We expect to encode TIC specifications into Isabelle/HOL logic for machine-assisted proof. Besides, we are investigating further heuristics to facilitate the validation.

References

1. M. M. Adams and P. B. Clayton. Clawz: Cost-effective formal verification for control systems. In *7th International Conference on Formal Engineering Methods*, pages 465–479, 2005.
2. R. D. Arthan, P. Caseley, C. O'Halloran, and A. Smith. ClawZ: Control laws in Z. In *3rd IEEE International Conference on Formal Engineering Methods*, pages 169–176. IEEE Press, 2000.
3. P. Caspi, A. Curic, A. Maignan, C. Sofronis, and S. Tripakis. Translating discrete-time Simulink to Lustre. In *EMSOFT'03*, Philadelphia, PA, USA, 2003.
4. A. Cavalcanti, P. Clayton, and C. O'Halloran. Control law diagrams in Circus. In *FM'05*.
5. J. E. Dawson and R. Goré. Machine-checking the timed interval calculus. In *AI'02*.
6. C. J. Fidge, I. J. Hayes, A. P. Martin, and A. Wabenhorst. A set-theoretic model for real-time specification and reasoning. In *MPC'98*, pages 188–206. Springer-Verlag, 1998.
7. S. Gupta, B. H. Krogh, and R. A. Rutenbar. Towards formal verification of analog designs. In *IEEE/ACM International Conference on Computer Aided Design*, pages 210 – 217, 2004.