CoCoSim: an automated analysis framework for Simulink/Stateflow *

Applying V&V techniques for safety requirement on Simulink/Stateflow models.

Hamza Bourbouh[†‡] Guillaume Brat[‡] Pierre-Loïc Garoche[§]

Abstract

Performing verification and validation (V&V) early in the development cycle of critical systems can help reduce the cost and time of detecting and fixing errors. Thus, performing V&V at the design level helps eliminate potential problems before the software is fully implemented. Our objective is to enable the verification of Simulink (a graphical dataflow modeling language widely used in the design of flight control systems) models with respect to formal properties that represent system requirements. In this paper, we present the CoCoSim toolbox: an open source framework for specifying and verifying user-defined requirements on Simulink models. The open architecture of the tool enables the integration of multiple analyses (ours and promising ones in the research community for instance) in a bid to truly enable the application of formal verification methods to Simulink/Stateflow models.

We believe that model-based system engineering combined with tools supporting both code development and V&V activities could make a huge impact on the fast development of space systems. In addition, the open-source feature of the framework eases the integration of state of the art tools and methods from academia, enabling their uses by the industry practionner.

1 Introduction

1.1 Context

Safety-critical systems design requires a thorough development process including formal verification and correct by construction behaviour. In that area, Model-Based Design has been widely used for software development. Such an approach offers the refinement of a system from High Level Requirements down to the embedded code while having an executable model at different stages. Matlab/Simulink[1] from MathWorks, is a *de facto* model-based design standard in industry, offering verification and code generation means.

Nonetheless, other development frameworks are used in addition in some industries, such as aeronautic, railways or space. Indeed, control/command applications have received a particular attention over the years and several synchronous programming languages such as Esterel [1], Lustre [7] or Signal [15] have been defined to help their design. Scade [9] is an industrial and DO 178C qualified Lustre-based framework that provides strong guarantees and proofs well appreciated, in particular for certification.

Offering frameworks linking Simulink and synchronous approaches is thus appealing. CoCoSim belongs to this category as it is an *open source* tool that translates Simulink specification in Lustre while preserving semantics and providing many associated traceability or test capabilities.

This paper gives a brief overview of the CoCoSim architecture and its current capabilities. While Simulink models are more general and could manipulate both continuous time and discrete time systems, their semantics is not as formally defined as it is for a language such as Lustre. In our work we restrict Lustre to the discrete-time subset of Simulink constructs, which is a reasonable assumption when considering models that will be auto-coded into embedded devices.

2 Overview of CoCoSim

CoCoSim is a highly automated framework for verification and code generation of Simulink/Stateflow models. It consists of an open architecture, allowing the integration of different analyses. CoCoSim is structured as a compiler, sequencing a series of translation steps leading, eventually to either the production of

*The work was partially supported by projects ANR-17-CE25-0018 and NASA Contracts No. NNX14AI09G and NNA14AA60C.

[†]KBR Inc.

[‡]NASA Ames Research Center

[§]ENAC, Université de Toulouse, France

[1]https://www.mathworks.com/products/simulink.html

source code, or to the call to a verification tool. By design, each phase is highly parametrizable through an API and could then be used for different purposes depending on the customization. The Figure 1 outlines the different steps.

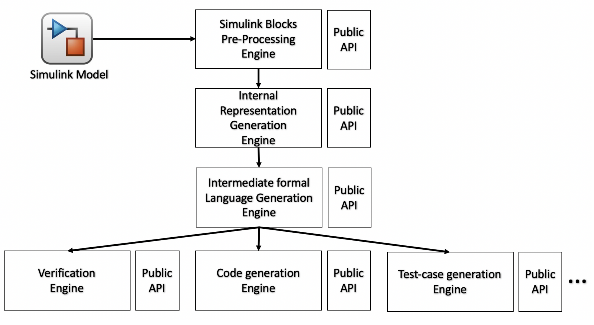
Simulink Model
Simulink Blocks Pre-Processing Engine
Public API
Internal Representation Generation Engine
Public API
Intermediate formal Language Generation Engine
Public API
Verification Engine
Public API
Code generation Engine
Public API
Test-case generation Engine
Public API
...

Figure 1: CoCoSim framework

2.1 Formal semantic

CoCoSim provides a **formal semantic** of a well defined subset of Simulink/Stateflow blocks. This formal representation will permit the use of formal verification methods and code generation.

CoCoSim starts first by simplifying some complex blocks into a set of basic blocks. Then an internal representation of the model is generated containing all information needed for code generation. Based on the work of Caspi et al. [6], Gene-Auto [13, 16] and P [3] projects, CoCoSim translates modularly the pre-processed mono-periodic Simulink model into an equivalent Lustre model. The generated Lustre model has the same hierarchy as the original Simulink model and preserves the initial semantic.

CoCoSim is **customizable** and **configurable**. Indeed, it supports most of frequently used Simulink blocks libraries (around 100 blocks) and new blocks can be easily supported.

2.2 Supported analyses

Once a formal representation of Simulink model is generated, CoCoSim is connected to a set of external tools to provide code generation, formal verification or test case generation. The toolchain is **highly automated** as all the steps of verification or code generation are automated.

The external tools are introduced and linked to the platform in a very generic way. While CoCoSim is built mainly around a specified set of tools, additional ones can be easily locally linked or even distributed as extensions.

All CoCoSim analyses are performed on the compiled artifact and the results are expressed back at Simulink level thanks to **traceability** information. We sketch here the features of the connected tools. At the current moment all tools are open-source and freely available. It **scales** well with large models, therefore various verification techniques and compositional reasoning can be used.

Formal Verification: SMT-based model checking Once requirements have been expressed using CoCoSim library and attached to the Simulink model, different tools can perform SMT-based model checking and check their validity. In case the property supplied is falsified, CoCoSim provides means to simulate the counterexample trace in the Simulink environment. Currently, CoCoSim is connected to Kind2 [8] a powerful tool that implement multiple algorithms including k-induction [14] and IC3/PDR [4] as well as on-the-fly invariant generation. All of these can be performed with various SMT solvers: CVC4, Z3, Yices.

Code generation: Some of CoCoSim backends provide code generation. Eg. LustreC [12] is an implementation of the modular compilation scheme [2] used in Scade. It preserves the hierarchy of the initial model, easing the checking of traceability between Lustre and generated C code.

Test cases generation: CoCoSim generates test cases based on two different methods. In the first method a coverage criteria such as MC-DC is used. The second approach relies on the notion of mutants. A good test suite distinguishes valid program from mutants.

3 Experiments

Since we started this effort of applying Lustre-level analyses to Simulink models, we have had the opportunity to evaluate the approach and the applicability of CoCoSim on reasonably large examples. Among them the NASA Transport Class Model (TCM) [5], the model describing the attitude and orbital control system (AOCS) of the Space Shuttle, the nominal mode of the AOCS of a French scientific satellite (DEMETER) or on other industry-provided examples such as publicly available[2] Lockheed Martin Cyber Physical Systems (LMCPS) challenges [10, 11] which is a set of aerospace-inspired examples provided as text documents specifying the requirements along with associated Simulink models. Examples range from a basic integrator to complex autopilots. The complete case study and analysis results are presented in our technical report.[3]

[2]https://github.com/hbourbouh/lm_challenges

[3]`https://drive.google.com/drive/u/1/folders/1GsKiu_O9_OSK_5XcLZZefi6g9MDAeOCC`

References

[1] Gérard Berry and Georges Gonthier. “The Esterel synchronous programming language: design, semantics, implementation”. In: *Science of Computer Programming* 19.2 (1992), pp. 87–152. ISSN: 0167-6423.

[2] Dariusz Biernacki, Jean-Louis Colaço, Grégoire Hamon, and Marc Pouzet. “Clock-directed modular code generation for synchronous data-flow languages”. In: *LCTES’08*. 2008.

[3] Matteo Bordin, Tonu Naks, Marc Pantel, and Andres Toom. “Compiling heterogeneous models: motivations and challenges”. In: *Proceedings of the 6th International Congress Embedded Real Time Software (ERTS’12)*. 2012.

[4] Aaron R. Bradley. “IC3 and beyond: Incremental, Inductive Verification”. In: *CAV’12*. 2012.

[5] Guillaume Brat, David H. Bushnell, Misty Davies, Dimitra Giannakopoulou, Falk Howar, and Temesghen Kahsai. “Verifying the Safety of a Flight-Critical System”. In: *FM 2015: Formal Methods - 20th International Symposium, Oslo, Norway, June 24-26, 2015, Proceedings*. 2015, pp. 308–324.

[6] Paul Caspi, Adrian Curic, Aude Maignan, Christos Sofronis, and Stavros Tripakis. “Translating Discrete-Time Simulink to Lustre”. In: *Third International Conference on Embedded Software EMSOFT*. 2003, pp. 84–99.

[7] Paul Caspi, Daniel Pilaud, Nicolas Halbwachs, and John Plaice. “Lustre: A Declarative Language for Programming Synchronous Systems”. In: *POPL’87*. 1987, pp. 178–188.

[8] Adrien Champion, Alain Mebsout, Christoph Sticksel, and Cesare Tinelli. “The Kind 2 Model Checker”. In: *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*. 2016, pp. 510–517.

[9] Jean-Louis Colaço, Bruno Pagano, and Marc Pouzet. “SCADE 6: A formal language for embedded critical software development (invited paper)”. In: *11th International Symposium on Theoretical Aspects of Software Engineering, TASE 2017, Sophia Antipolis, France, September 13-15, 2017*. 2017, pp. 1–11.

[10] Chris Elliott. “An Example Set of Cyber-Physical V&V Challenges for S5, Lockheed Martin Skunk Works”. In: *Safe & Secure Systems and Software Symposium (S5), 12-14 July 2016, Dayton, Ohio*. Ed. by Air Force Research Laboratory. 2016.

[11] Chris Elliott. “On Example Models and Challenges Ahead for the Evaluation of Complex Cyber-Physical Systems with State of the Art Formal Methods V&V, Lockheed Martin Skunk Works”. In: *Safe & Secure Systems and Software Symposium (S5), 9-11 July 2015, Dayton, Ohio*. Ed. by Air Force Research Laboratory. 2015.

[12] Pierre-Loïc Garoche, Temesghen Kahsai, and Xavier Thirioux. *LustreC*. `https://github.com/coco-team/lustrec`.

[13] Ana-Elena Rugina, David Thomas, Xavier Olive, and G. Veran. “Gene-Auto: Automatic Software Code Generation for Real-Time Embedded Systems”. In: *Proceedings of DASIA 2008 Data Systems In Aerospace*. 2008.

[14] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. “Checking Safety Properties Using Induction and a SAT-Solver”. In: *FMCAD’00*. 2000, pp. 127–144. ISBN: 978-3-540-40922-9.

[15] “Synchronous programming with events and relations: the SIGNAL language and its semantics”. In: *Science of Computer Programming* 16.2 (1991), pp. 103–149. ISSN: 0167-6423.

[16] Andres Toom, Tonu Naks, Marc Pantel, M Gandriau, and Indrawati. “Gene-Auto: an Automatic Code Generator for a safe subset of Simulink/Stateflow and Scicos”. In: *Proceedings of the 4th International Congress Embedded Real Time Software (ERTS’08)*. 2008.