# Verified Model Refactorings for Hybrid Control Systems

Sebastian Schlesinger
Software and Embedded Systems Engineering
Technische Universität Berlin, Germany
Email: sebastian.schlesinger@tu-berlin.de

*Problem Statement*

The ever growing complexity in modern embedded systems requires to incorporate increasingly many functions into a single system. Model Driven Engineering (MDE) is a well-established technique for the development of such complex embedded systems. Refactoring is a common technique to reduce the complexity and establish compliance with industrial MDE guidelines via model transformations. However, model refactorings are often performed manually or partially automatically via tools without guarantee of correctness. Such refactorings therefore bear the risk of introducing unwanted or even erroneous behaviour. This is critical, especially in safety-critical applications, where an error may cause enormous costs or even endanger human lifes. To overcome this problem, I present an approach to guarantee behavioural equivalence of graph-based hybrid control systems.

*Challenges*

The key challenges we address with our approach are:

- *Hybrid Systems:* We consider models comprising time-discrete, time-continuous, and control flow elements together. The time-continuous part introduces an approximation error caused by the execution on machines with finite execution time and storage space. The challenges are firstly to properly consider this approximate nature of time-continuous model parts, secondly to consider the interplay of data flow and control flow in the models, and thirdly to consider discrete jumps in continuous behaviour.
- *Formal Foundation:* To provide guarantees for all possible input scenarios, a precise mathematical description is required. However, Simulink has only an informal, descriptive semantics. We require a precise and formal semantics and equivalence notion.
- *Automation:* A manual approach requires an unacceptable amount of effort for verification and its application is likely to be erroneous. A largely automated solution is therefore desired. However, this is challenging because the state space for hybrid control systems gets extremely large for arbitrary input signals.

*Main Contributions*

Our proposed solution is twofold. Firstly, we provide a framework for the formal verification of refactorings of data-flow oriented hybrid control models. We select MATLAB/ Simulink as most prominent representative of languages for such types of models. Secondly, we provide an automation of our framework for a clearly defined subset of Simulink models. Figure 1 provides an overview over our main contributions. We go through the main contributions as depicted in the figure.

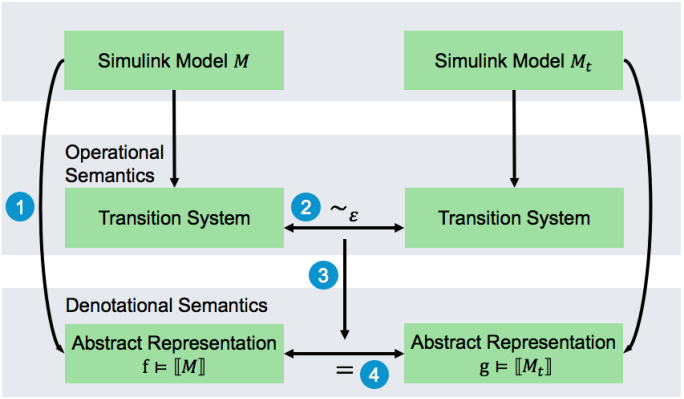


Fig. 1. Overview over the Equivalence Proof Framework

1) To obtain a *formal foundation*, we have adopted the approach [BC12], which describes an operational semantics for Simulink. In addition, we provide a semantics on denotational level, which we call the Abstract Representation (AR), and have shown that it is sound with respect to the operational semantics. We automatically extract the AR from a given Simulink model. The AR describes time-discrete parts in terms of difference equations, and time-continuous parts in terms of ordinary differential equations (ODEs). We prove the soundness of our AR with the operational semantics in the following sense: If we consider sequences of executions of the operational semantics with decreasing sample step size, these sequences converge uniformly to the interpretations or solutions determined by the AR. Our AR is formally well-defined and concise, and it enables us to formally relate traces or trajectories with respect to behavioural equivalence.
2) The simulation semantics of Simulink includes numerical approximations. This means that two *hybrid models* that are mathematically equivalent do not necessarily yield exactly the same trajectories if executed on a machine with finite sample step sizes. Therefore, traditional equivalence notions such as bisimulation are not applicable. We therefore adapt the concept of approximate bisimulation

to Simulink. Approximate bisimulation allows us to relate two models that are close to each other up to a maximal distance $\varepsilon$, the precision of the approximate bisimulation.

3) We provide a proof scheme, i.e., an approach for automated generation of proof obligations to show approximate bisimulation for time-discrete, time-continuous, and hybrid models. The proof obligations are provided on a denotational, mathematical level. We also provide means to automatically calculate the *precision* $\varepsilon$ of the approximate bisimulation.
4) We provide an approach for *semi-automated verification* of behavioral equivalence of data flow-oriented hybrid control systems expressed by linear difference and differential equations. Our automation approach is based on a calculation of all possible data flow paths in a given model. We call them Data Flow Models. Each Data Flow Model does not contain control flow elements, i.e., we eliminate control flow elements such as Switch blocks from the original model and calculate a set of models without control flow. Then, we partition the Data Flow Models to enable modular verification. The partitioning distinguishes between cycles or feedback loops and non-cyclic or directed cyclic graph (DAG) elements. We extract the ARs of the partitions for all Data Flow Models and verify their equivalence symbolically. In this step, we make use of Laplace- and z-transform to transform the system into simple algebraic expressions rather than complex ODEs. This enables largely automated symbolic equivalence verification with the help of a Computer Algebra System (CAS). The only manual interaction that is required is a mapping of partitions from the original to the refactored model.

   We provide an approach to automatically calculate the local precision $\varepsilon_l$, i.e., the disturbance resulting from the approximation at the partition where the refactoring took place. We propagate the local precision $\varepsilon_l$ to a global precision $\varepsilon_g$. The global precision is the overall precision of the approximate bisimulation of source and target model.

   Finally, we verify supplemental proof obligations such as conditions that exclude time-delayed behaviour at control flow elements, or conditions to guarantee injectivity of Laplace or z-transform.

We have published our framework for purely time-discrete and purely time-continuous models in [SHGG15], [SHGG16b] and for hybrid control systems in [SHGG16a]. We have demonstrated the practical applicability of our approach with various case studies, comprising industrial Simulink examples from The MathWorks, Inc. as well as case studies from our industrial partners from the automotive industry.

### *Related Work*

There are various articles concerning formal Simulink verification, e.g. [RG14], [HRB13], [TSCC05]. In each of them, the authors map only a subset of Simulink to formal languages. Some work has been published to formalise semantics of causal block diagrams in general or Simulink in particular, e.g. [BC12], [GDV16], [DPT16]. However, they do not consider formal verification of behavioural equivalence. We have chosen and extended [BC12] as our formal foundation. There exists a large variety of works about hybrid systems verification, e.g. [FMQ$^+$15], [Alu11], and on behavioural equivalence of hybrid systems, e.g. [GP11]. However, they do not consider the specifics of data flow-oriented hybrid control modelling languages such as Simulink. Finally, there exist some works on refactorings of Simulink models, e.g. [TWD13]. However, they do not aim at verification of behavioural equivalence.

### *Conclusion*

In my thesis, I have provided a framework for formal verification of behavioural equivalence of data-flow oriented hybrid control systems. I have automated the approach for a clearly defined subsystem of the most prominent representative of such models, namely Simulink. With this approach, introduction of erroneous behaviour during refactorings is avoided. It therefore is a substantial contribution to increase safety in embedded systems design.

## REFERENCES


[Alu11] Rajeev Alur. Formal verification of hybrid systems. In *EMSOFT*, pages 273–278. IEEE, 2011.

[BC12] Olivier Bouissou and Alexandre Chapoutot. An operational semantics for simulink's simulation engine. *ACM SIGPLAN Notices*, 47(5):129–138, 2012.

[DPT16] Iulia Dragomir, Viorel Preoteasa, and Stavros Tripakis. Compositional semantics and analysis of hierarchical block diagrams. In *International Symposium on Model Checking Software*, pages 38–56. Springer, 2016.

[FMQ$^+$15] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. Keymaera x: an axiomatic tactical theorem prover for hybrid systems. In *International Conference on Automated Deduction*, pages 527–538. Springer, 2015.

[GDV16] Cláudio Gomes, Joachim Denil, and Hans Vangheluwe. Causal-block diagrams. Technical report, Technical report, 2016.

[GP11] Antoine Girard and George J. Pappas. Approximate bisimulation: A bridge between computer science and control theory. *European Journal of Control*, 17(5):568–578, 2011.

[HRB13] Paula Herber, Robert Reicherdt, and Patrick Bittner. Bit-precise formal verification of discrete-time matlab/simulink models using smt solving. In *EMSOFT*. IEEE Press, 2013.

[RG14] Robert Reicherdt and Sabine Glesner. Formal verification of discrete-time matlab/simulink models using boogie. In *Software Engineering and Formal Methods*, pages 190–204. Springer, 2014.

[SHGG15] Sebastian Schlesinger, Paula Herber, Thomas Göthel, and Sabine Glesner. Towards the verification of refactorings of hybrid simulink models. In Alexei Lisitsa, Andrei Nemytyk, and Alberto Pettorossi, editors, *VPT*, volume 199 of *EPTCS*, page 69, 2015.

[SHGG16a] Sebastian Schlesinger, Paula Herber, Thomas Göthel, and Sabine Glesner. Proving correctness of refactorings for hybrid simulink models with control flow. In *CyPhy*, 2016.

[SHGG16b] Sebastian Schlesinger, Paula Herber, Thomas Göthel, and Sabine Glesner. Proving transformation correctness of refactorings for discrete and continuous simulink models. In *ICONS, EMBEDDED*, pages 45–50, 2016.

[TSCC05] Stavros Tripakis, Christos Sofronis, Paul Caspi, and Adrian Curic. Translating discrete-time simulink to lustre. *ACM Transactions on Embedded Computing Systems (TECS)*, 4(4):779–818, 2005.

[TWD13] Quang Minh Tran, Benjamin Wilmes, and Christian Dziobek. Refactoring of simulink diagrams via composition of transformation steps. In *ICSEA*, pages 140–145, 2013.