Proving Correctness of Refactorings for Hybrid Simulink Models with Control Flow

Sebastian Schlesinger, Paula Herber, Thomas Göthel, and Sabine Glesner

Software and Embedded Systems Engineering
Technische Universität Berlin
Straße des 17. Juni 135, 10623 Berlin
{Sebastian.Schlesinger, Paula.Herber, Thomas.Goethel,
Sabine.Glesner}@tu-berlin.de

Abstract Hybrid models are highly relevant for the development of embedded systems because they cover both their continuous and discrete aspects. To master the increasing complexity of embedded systems design, transformation techniques such as automated refactoring play an important role, as they allow for simplifying (sub)models. In safety-critical environments, it is crucial to formally verify the behavioural equivalence of source and transformed target model. For data-flow models that contain control flow entities, this is a major challenge because small deviations of trigger values at control flow elements can yield diverging behaviour of the systems. In this paper, we present our approach that enables the semi-automated verification of the behavioural equivalence of hybrid MATLAB / Simulink models. To this end, we define a static analysis that derives proof obligations to estimate the worst case deviation between model and refactored model. Our approach can be applied to many practical applications such as in the automotive or aerospace industry where MATLAB / Simulink is a de-facto standard.

1 Introduction

Embedded systems are ubiquitous and continuously become more and more important in the near future. Highly advanced manufacturing techniques allow for the production of miniaturised, energy-efficient, highly inter-connected components with huge computing power. This offers the chance to model systems with a large functional range, but also means that the complexity is hard to manage. Especially in safety-critical environments, e.g. in the automotive, aerospace or train industries, it is crucial to fully understand what the system does in every possible scenario. This implies that designers for embedded systems must follow established software engineering techniques. Among these, model-driven engineering (MDE), refactorings and formal verification play an important role because they allow for analysing abstract systems that are safely refined later on. A *correct refactoring* is a model transformation that guarantees that the behaviour of the source model compared to the transformed model is equivalent. MATLAB/Simulink is a tool and the de facto standard for MDE in the above mentioned industries. One

of Simulink's important features is the modelling support for *hybrid* models. These are models containing both discrete and continuous parts together. Hybrid models allow for the design of the (continuous) environment and the (discrete) control or, in mixed systems, of analogous and digital parts of systems together. Consequently, they are of substantial help for the designer to understand the desired behaviour of the whole system comprehensively, especially in early design phases. However, establishing a formal proof for the behavioural equivalence between hybrid Simulink model and refactored counterpart is a major challenge. The main difficulty is that for the continuous parts, the simulation is only able to calculate approximations rather than performing exact operations. Consequently, equivalent behaviour cannot be guaranteed in the traditional sense in general. We therefore adapted the weaker notion of *approximate bisimulation* for Simulink and provided an approach to verify transformation correctness of discrete *or* continuous Simulink models in [22,23]. This means that we did not yet consider systems with both discrete and continuous modelling elements within the same model.

In this paper, we present a methodology to enable the verification of behavioural equivalence of hybrid Simulink models with control flow. This means in Simulink perspective, the models contain switch blocks with three input ports (two data ports and one control port) and a condition, which is connected with an incoming control signal. If the control input fulfills the condition, the first data signal is fed through, otherwise the second data signal. This case is comparable with the discrete jumps at guards of hybrid automata [17].

As a starting point to achieve our goal, we use the approach presented in [22,23], which enables the verification of refactorings of either solely discrete or solely continuous models using the concept of approximate bisimulation. This approach guarantees under certain conditions (i.e., proof obligations that the designer needs to verify) that the values of source and target model vary within an ε environment. The main challenge we need to deal with emerges if 1) the input signal at a switch is continuous and 2) a refactoring takes place in the subsystem impacting the input signal. In this case, small variations coming from the approximately equivalent behaviour in the control input may result in diverging behaviour at the outport of the switch and consequently at the observation.

Figure 1 shows an example. The control input signal is a sinusoid signal in source and target model that mathematically speaking (i.e., if the simulation step sizes would tend to 0) crosses the x-axis two times. The refactored control signal is the mathematical solution of the ODE in the source model. The data signals are just the constants 0 and 1 to illustrate the principle. The observation at the output of the switch differs as shown in the output graphs below the model. This observation depends on the sample step sizes and the precision ε from the approach [23] as we show in this paper. This means, if certain additional proof obligations can be verified, the diverging behaviour can be avoided, otherwise the designer has to accept a larger ε.

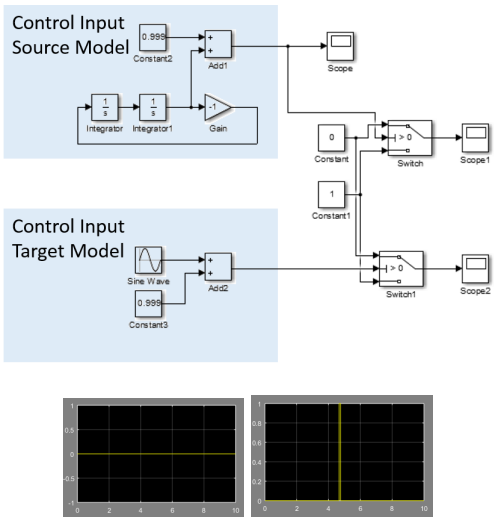

Figure 1. Simulink example for diverging behaviour: Output

With our approach, we provide an important first step to extend the approach [23] for purely time-discrete or time-continuous models to cope with the problem of verifying refactorings of hybrid Simulink models. In our approach, we deal with hybrid models that patch purely time-discrete and time-continuous models together via a Switch block. Note that with our approach many industrial applications, are already manageable, since 1) models with time-discrete control part do not pose a problem (as we also show), 2) models with time-continuous control part, but no refactoring in this part are supported and 3) time-continuous and time-discrete data input (refactored or not) are supported. However, we are confident to even extend our approach to be able to deal with mixed time-discrete and time-continuous models (e.g. Unit Delay and Integrator block in one system) in future work.

The rest of this paper is structured as follows. In Section 2 we provide some background information that helps to understand the remainder of the paper. Particularly, we give a brief introduction to Simulink and to the approach for proving behavioural equivalence of solely discrete or solely continuous Simulink models presented in [22,23], which we extend in this paper. Section 3 provides an overview of our approach in this paper. In Section 4, we extend the Abstract Representation from [23], which is a set of equations that describes how the blocks modify their incoming signals, and provide a denotation. This denotation associates the signal lines with the function that expresses how the respective signal evolves over time - if this is possible (e.g. only if the underlying ODE is analytically solvable). This also enables us to observe signals from inside a Simulink model. This is covered in Section 5. In Section 6, we provide sufficient conditions to check if the behaviour of a source and a target model does not

diverge from each other - or in which time period this can happen - and provide a brief description of the resulting static analysis. In Section 7, we discuss related work. We conclude with a summary and an outline for future work in Section 8.

2 Background

In this section, we provide a short introduction to Simulink and briefly summarize our approach for the verification of Simulink refactorings presented in [22,23].

2.1 Simulink

Simulink is a widely used modelling language for dynamic systems. It is based on MATLAB, and both products are developed by The MathWorks [25]. In Simulink, dynamic systems are modelled as block diagrams. They can be simulated, and, with further software packages, it is also possible to automatically generate code.

In this paper, we consider six kinds of Simulink block types: unsampled or direct feed-through blocks (e.g. arithmetic blocks), discrete blocks (e.g. Unit Delay, Discrete Integrator), continuous blocks (e.g. Integrator), sink blocks (e.g. Scope), source blocks (e.g. Constant, Sine Wave, Ramp) and control flow blocks (e.g. switch). Each block can have inports and outports. These are the interfaces via which the blocks are connected.

In [6], Bouissou et al. defined an operational, synchronous semantics for Simulink. It deterministically calculates the values for the next time step for each signal line and each internal variable (for discrete and continuous blocks). There exist two modes for a simulation: fixed step and variable step size simulation. We provide a brief explanation for how the simulation is executed.

Initially, all outputs are calculated: direct feed-through blocks (e.g. arithmetic blocks) are evaluated, the values of internal variables are written on the outgoing signal line. Then, the discrete, internal variables are evaluated. Afterwards, the internal variables for the continuous blocks are evaluated by applying a chosen approximation method, e.g. Euler technique or one of the Runge-Kutta techniques [7]. After this, a *zero crossing detection* takes place to check if the output signal of discrete control elements, e.g. switches, needs to be changed. Finally, the next simulation step size is calculated (only if variable step size simulation is activated) and the simulation process starts again for the next sample step.

2.2 Proving Transformation Correctness for Discrete and Continuous Simulink Models

In [22,23], an approach to semi-automatically show transformation correctness of refactorings for Simulink systems that are solely discrete or continuous was presented. Figure 2 provides an overview over this approach. The idea is that the user extracts equations that describe the effect of all the blocks in a given model. This equation set is called the *Abstract Representation (AR)* of the given Simulink model. The respective interpretation of this AR is an output function

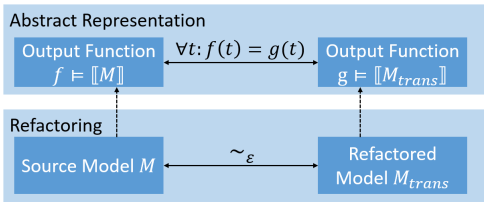

Figure 2. Approach for Correctness of Refactorings of Discrete or Continuous Simulink Models

(or several if there are multiple output blocks) that satisfies all equations of the AR. This is denoted by $f \models [\![M]\!]$. In [23], it is described that if the user is able to show that the interpretations of both models yield the same values in a given simulation interval $\mathfrak{I}$, i.e., $\forall t \in \mathfrak{I} : f(t) = g(t)$ with $f \models [\![M]\!], g \models [\![M_{trans}]\!]$, then the simulations of both models defined by the operational semantics for Simulink [6] are approximately bisimilar with a certain precision ε, denoted as $M \sim_\varepsilon M_{trans}$. The concept of approximate bisimulation is introduced, e.g. in [11,13]. In principle, it compares states from two labelled transition systems (LTS). The observations of the LTS must be in the same metric space. Two states are approximately bisimilar with precision ε if the distance of their observations is at most ε - and this also holds after one step in both LTS.

3 Proving Correctness of Refactorings for Hybrid Models with Control Flow

To achieve the extension of the approach [23] for solely discrete or continuous models to cover hybrid models with control flow, we proceed in three stages.

1. We extend the Abstract Representation (AR) to cover control flow elements. In this context, we provide a more concise version of the AR, which moves towards a denotational semantics.
2. Our approach is applicable whenever a given refactoring is located in the system that influences the control input of a control flow element (e.g. a switch). If the refactoring takes place at one of the data inputs, the approach [23] is already sufficient. We therefore need a notion that expresses what it exactly means that only the control input is influenced. Furthermore, we define a notion that allows us to observe the inner behaviour of a given Simulink model. In this course, we describe precisely the relations between Simulink models, expressions and interpreting functions.
3. We provide additional conditions and proof obligations that allow the user to verify the correctness of a given refactoring, i.e. approximate bisimulation with precision ε. The sufficient conditions for the approximately equivalent behaviour depend on several parameters that are either determined by parameters of a differential equation such as the Lipschitz constant or can be modified by the user, e.g. the step size.

Limitations and Assumptions

As representatives for discrete and continuous blocks, we currently support Unit Delay and Integrator blocks respectively. All Unit Delays must have the same sample times. Currently, we support fixed sample step sizes only. Feedback loops, i.e., cycles in the Simulink graph, are only permitted with at least one Unit Delay or Integrator in the cycle. This means that we do not support *algebraic loops*, which is a minor restriction because they are practically irrelevant. As control flow elements, we currently only support switch blocks directly. However, the approach is transferable to other control flow elements such as If Action subsystems. Our approach is also only applicable to systems that do not contain time-discrete and time-continuous parts together in one submodel (e.g. Unit Delay and Integrator in the same subsystem).

4 Denotational Abstract Representation of Hybrid Simulink Models with Control Flow

In this section, we extend our Abstract Representation (AR) presented in [23] with a denotation that enables the precise, formal description of the behaviour of hybrid Simulink models with control flow. To this end, we introduce a *denotation* for each signal line, which also defines the behaviour at control flow elements (exemplified using a switch block). To achieve this, we firstly extend some of the notions we already introduced in [23] to obtain a more handy notation.

Definition 1 (Simulink Model Syntax) *A Simulink model is a tuple* $M = (B, E, I, O, S, c, s, t)$ *with* B *a finite set of blocks,* E *a finite set of signal lines,* $I, O, S \subseteq B$*, respectively are sets of input, output blocks and blocks carrying a state variable (Integrator and Unit Delay),* I, O, S *pairwise disjoint.* $c : B \times \mathbb{N} \to E$ *(for connection) provides the signal line at the* $n-th$ *port of a block (the order of the incoming signal line matters),* $s : E \to B$ *assigns a source block to a signal line,* $t : E \to B$ *assigns a target block to a signal line. The subtuple* (B, E, s, t) *is a graph.*

Next, we enhance the AR and define a denotation for each signal line, which expresses how the signal at this line evolves over time. Note that due to spacing constraints, we move some definitions of the cases in the case distinction to the bottom of the respective definition. Also note that in the following definition, we use terms (set $Term$) in a straightforward way: $in(t)$ to denote functions for incoming signals, $l(t)$ for state variables etc.

Definition 2 (Denotation for Simulink Models) *Let* M *be a Simulink model. We assign each signal line* $l \in E$ *an expression* $expr_M : E \to Term, l \mapsto$

$$\begin{cases} in_i(t) \text{ if } s(l) \in I \\ l(t) \text{ if } s(l) \in S \\ f(expr_M(c(s(l), 1), ..., c(s(l), n))) \\ case(expr_M(c(s(l), 2)) \diamond \xi, expr_M(c(s(l), 1)), expr_M(c(s(l), 3))) \end{cases}$$

Note that in the first case ($l \in I$), also specific functions are allowed, e.g. $\sin(t)$ *or even just* 1 *(constant). The third case applies for function blocks, f stands for the function associated with the respective block $s(l)$. The latter case is applicable if the block $s(l)$ is a switch. The condition in the latter case is extracted as parameter from the Simulink model (the part $\diamond\xi$, $\diamond$ can be $<, \leq, >, \geq$). The denotation of a Simulink model M, $[\![.]\!] : E \to \mathbb{R}^{\mathfrak{I}} \cup \{\perp\}$ ($\mathfrak{I}$ being the simulation interval) is defined as*

$$[\![l]\!] = \begin{cases} \varphi \\ in_i \text{ if } s(l) \in I \\ case([\![c(s(l),2)]\!] \diamond \xi, [\![c(s(l),1)]\!], [\![c(s(l),3))]\!]) \\ f([\![c(s(l),1)]\!], ..., [\![c(s(l),n)]\!]) \\ \perp \text{ otherwise} \end{cases}$$

The first case is applicable if φ is the unique solution of the ODE $\frac{d}{dt}l(t) = expr_M(c(s(l),1)), l(t_0) = init$ if $s(l)$ is an Integrator or the solution of the difference equation $l(t+h) = expr_M(c(s(l),1)), l(t_0) = init$ if $s(l)$ is a Unit Delay. The third case is applicable if $s(l)$ is a switch. The fourth case is applicable for function blocks.

The function $case : Pred \times \mathbb{R}^{\mathfrak{I}} \times \mathbb{R}^{\mathfrak{I}}$ is defined as $(b, f, g) \mapsto f$ if b is fulfilled, g otherwise.

To keep the notation simple, we associate variables l_i with the respective signal lines connected at the output of discrete or continuous blocks in this paper. Definition 2 is sound with the AR from [23]. We omit the proof, which is straightforward, due to space constraints.

Lemma 1 (Soundness with AR) *For a given Simulink model $M = (B, E, I, O, S, c, s, t)$ with finite simulation interval $\mathfrak{I}$, $[\![.]\!] : E \to \mathbb{R}^{\mathfrak{I}}$ is well-defined and if the model does not contain any switch blocks, the following holds: $\forall l \in E \exists b \in O : c(b,1) = l \Rightarrow [\![l]\!] \models [\![M]\!]$.*

Note that our denotation is not a denotational semantics, since not in every case the ODE or difference equation can analytically be solved, i.e., the unique solution exists and can be represented as expression, and therefore it is possible that some expressions at several signal lines remain unresolved. In this case, $[\![l]\!] = \perp$. In addition, the denotation does not take the approximations into account, i.e., it may map a Simulink model to the exact solution of an ODE. However, as we know from [23], the actual simulation differs from this exact function. As an example, consider the source model depicted in Figure 1. There, we have at the line l connecting the output (Scope1) $[\![l]\!] = case(\sin(t) + 0.999 > 0, 0, 1)$, since $sin(t)$ solves the ODE $\frac{d}{dt}l_1(t) = l_2(t), \frac{d}{dt}l_2(t) = -l_1(t), l_1(0) = 0, l_2(0) = 1$. So, in this case we have a fully denotational representation of the Simulink model.

5 Observing the Inner Behaviour of Regular Hybrid Control Flow Simulink Models

Our notion of a denotation for signals in a Simulink model is applicable for both discrete and continuous Simulink models. To achieve a verification methodology for proving the transformation correctness of hybrid Simulink models with control flow, we define a subset of hybrid Simulink models with control flow, namely *Regular Hybrid Control Flow Simulink Models*. This definition enables us to observe the inner behaviour of hybrid Simulink models, which forms the basis for our verification methodology presented in the next section.

Definition 3 (Regular Simulink Models) *We call an expression* $f(in_1(t), ..., in_n(t), l_1(t), ..., l_m(t))$ *continuous if* $\forall 1 \leq i \leq m : l_i$ *is associated with the signal line at the output of a continuous block in a Simulink model* M *and* $expr_M(l_i)$ *is also continuous. Analogously, the expression is called discrete if the same holds for the output of a discrete block and* $expr_M(l_i)$ *is also discrete. We call an expression mixed if it is neither continuous nor discrete.*

We call a Simulink model M *a regular hybrid control flow model if it contains only one discrete control block with outgoing signal* l *and none of the expressions* $expr_M(l_i)$ *in* $expr_M(l) = case(expr_M(l_2) \diamond \xi, expr_M(l_1), expr_M(l_3))$ *is mixed.*

Note that with this definition, we ensure that each of the incoming signals is purely discrete or continuous. It is not possible that one of the paths contains both discrete and continuous blocks. Such mixed systems are part of future work. However, a discrete control element, feedback loops, also taking signals after the control element and feeding back in one or multiple of the inputs of the control element are allowed - as long as the requirement that either only discrete or only continuous blocks occur on each path is fulfilled.

In Lemma 1, we linked the interpretation of the AR from [23] with the denotation $[\![l]\!]$ where l is a signal line at an outport (where the observation takes place). This can straightforwardly be extended to allow observation 'inside' a Simulink model.

Lemma 2 *Let* M, M_{trans} *be two Simulink models,* $l \in E \cap E_{trans}$ *a signal line appearing in both models, the expressions* $expr_M(l)$ *and* $expr_{M_{trans}}(l)$ *both not mixed and without case in it and* $\forall t \in \mathfrak{I} : [\![l]\!]_M(t) = [\![l]\!]_{M_{trans}}(t)$. *Let furthermore* M_l *and* M_{trans_l} *be models that evolve from* M *and* M_{trans} *respectively by transferring only edges and blocks from* M *and* M_{trans} *that affect* l. *Then there exists an* ε *given by the approach [23] such that* $M_l \sim_\varepsilon M_{trans_l}$. *We briefly write* $[\![l]\!]_M \sim_\varepsilon [\![l]\!]_{M_{trans}}$.

The lemma can be presented more formally by defining more precisely what 'transferring only edges and blocks that affect l' means. However, we omit this here together with a proof due to space constraints.

Lemma 2 can equivalently be interpreted as follows. Consider we take a look at a signal line l and observe the function $\varphi : \mathfrak{I} \to \mathbb{R}$ evolving over time

at this signal line (after evaluation by Simulink). If the simulation step sizes go to 0, we would observe $[\![l]\!]_M = f : \mathfrak{I} \to \mathbb{R}$. Consider furthermore another Simulink model with a signal line, simulated behaviour $\psi(t)$ and if simulation step sizes go to 0, we had $[\![l]\!]_{M_{trans}} = g : \mathfrak{I} \to \mathbb{R}$. If $\forall t \in \mathfrak{I} : f(t) = g(t)$, i.e., the observations are mathematically (if simulation step sizes go to 0) equal, then for the simulated observations φ, ψ the relation $||\varphi(t) - \psi(t)|| \leq \varepsilon$ or equivalently $\psi(t) \in [\varphi(t) - \varepsilon, \varphi(t) + \varepsilon]$ holds.

We now have the prerequisites to derive the sufficient conditions for approximately equivalent behaviour in the next section.

6 Proof obligations for behavioural equivalence of regular hybrid Simulink models with control flow

We have now prepared all prerequisites to investigate the conditions under which approximately equivalent behaviour can be guaranteed.

As we already mentioned in the introduction in Section 1, the behaviour compared between source and target model cannot be guaranteed to remain approximately equivalent with the same precision for the whole simulation interval in every case for Simulink models with control flow. In some cases, time delays in the switch are unavoidable, which increases the precision ε then. To be able to express the time-dependent changes of approximate behaviour as precisely as possible, we write $M \sim_{\varepsilon}^{[t_1,t_2]} M_{trans}$. That means that $M \sim_{\varepsilon} M_{trans}$ in the simulation interval $[t_1, t_2] \subseteq \mathfrak{I}$. It is clear that if we have a sequence of n intervals that cover $\mathfrak{I}$ and $\varepsilon_1, ..., \varepsilon_n$, then this sums up to $M \sim_{\max(\varepsilon_1,...,\varepsilon_n)} M_{trans}$.

We now present conditions under which approximately equivalent behaviour for regular hybrid Simulink models with control flow can be guaranteed. The first theorem deals with refactorings where the control input is discrete.

Theorem 1 *Let M and M_{trans} regular hybrid control flow Simulink models, the switch block consisting of the control input l_2 and the data inputs l_1, l_3 and the output l in both systems. Let furthermore $[\![l_i]\!]_M \sim_{\varepsilon_i} [\![l]\!]_{M_{trans}}$ for $1 \leq i \leq 3$ due to performed refactorings in both systems. If $\varepsilon_2 = 0$, which means either no refactoring that reduces an Integrator (i.e., resolves an ODE) took place at the control input or the control input expression $expr(l_2)$ is discrete, then $M \sim_{\max(\varepsilon_1,\varepsilon_3)} M_{trans}$.*

Proof. Since the control input is not perturbed, the switching happens in both systems simultaneously and the approximately equivalent behaviour results directly from the application of the approach [23] and Lemma 2.

All subsequent theorems deal with refactorings of continuous control input, i.e., where the control input is perturbed by an $\varepsilon > 0$. The next theorem is about the case where the observation at the input is too far away from the threshold to cause a switch.

Theorem 2 *Let M and M_{trans} be regular hybrid control flow Simulink models, l, l_1, l_2, l_3 as in the above theorem. Let without loss of generality the condition*

at the switch be $\geq \xi$. *Let furthermore* $[\![l_i]\!]_M \sim_{\varepsilon_i} [\![l]\!]_{M_{trans}}$ *for* $1 \leq i \leq 3$ *due to performed refactorings in both systems,* $\varepsilon_2 > 0$ *(implying that* $expr(l_2)$ *is continuous),* $[\![l_i]\!]_M, [\![l]\!]_{M_{trans}} \neq \perp$ *(implying, the solution can be represented by a closed expression). Let the simulations of* M *and* M_{trans} *both be performed with the same fixed sample step size* h. *We also define* $\Sigma = \{\lambda h | \lambda \in \mathbb{N}\}$ *the set of sample steps. Note that we know that the values of the observations at* l_2 *for* M *and* M_{trans} *vary by at most* ε_2 *around the mathematical observation* $[\![l_2]\!]_M$ *(as pointed out in the previous section). We furthermore define a function* $Zero : \mathbb{R}^{\mathfrak{I}} \to \mathcal{P}(\mathbb{R})$, $Zero(f) = f^{-1}(0) = \{x \in \mathfrak{I} | f(x) = 0\}$ *the set of zero crossings of the function* f.

1. *If* $\forall t \in [t_1, t_2] \subseteq \mathfrak{I} : ||[\![l_2]\!]_M(t) - \xi|| > \varepsilon_2$, *then* $M \sim_{\varepsilon_1}^{[t_1,t_2]} M_{trans}$.
2. *Consider an interval* $\mathfrak{L} = [t_1, t_2] \subseteq \mathfrak{I}$ *where* $Zero([\![l_2]\!]_M - \varepsilon - \xi) \cap \mathfrak{L} =: \{\zeta_1, ..., \zeta_n\}$ *is finite, i.e., the zero crossings are all isolated (meaning around each* ζ_i *is an environment that contains no other* ζ_j*) and increasingly ordered. If* $\forall 1 \leq i < n : \forall t \in \Sigma : t \notin [\zeta_i, \zeta_{i+1}]$ *and* $\frac{d}{dt}[\![l_2]\!](\zeta_j) < 0$ *with* j *only the odd indices, then* $M \sim_{\varepsilon_1}^{\mathfrak{L}} M_{trans}$.

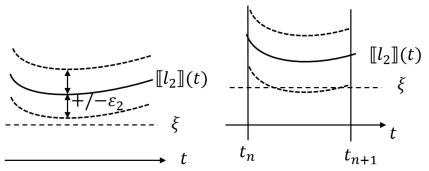

Figure 3. Illustration for the proof of Theorem 2

Proof. Figure 3 illustrates the proof idea for the lemma. The first case is represented by the picture on the left. The values of the simulations that are within the ε environment around $[\![l_2]\!]$ as described in [23] are too far from ξ to cause a zero crossing in both systems. The second case (depicted on the right hand side) eases this strict condition. In this case, it is possible for the values $[\![l_2]\!](t) - \varepsilon$ to cross the threshold ξ. However, this must not be recognised by the system. Hence, it may only happen between two sample steps. The condition $\frac{d}{dt}[\![l_2]\!](\zeta_j) < 0$ ensures that the crossing takes place starting with an observation from above ξ (note that $[\![l_2]\!]$ is indeed differentiable).

We now describe the cases where the control input signal crosses the threshold ξ completely. If this happens fast enough, we don't have a phase of increased ε, otherwise we have a time delay in the switching and therefore must accept a temporary increase of the error in the resulting behaviour.

Theorem 3 *Let* M *and* M_{trans} *be regular hybrid control flow Simulink models,* $l, l_1, l_2, l_3, \xi, \varepsilon_i, \Sigma, Zero$ *as in the above theorems,* $\varepsilon_2 > 0$. *Let without loss of*

generality the condition at the switch $\geq \xi$. *Consider we have an isolated element* $\zeta^- \in Zero([\![l_2]\!]_M - \varepsilon_2 - \xi) =: Zero^-$. *Consider furthermore there exists an element* $\zeta^+ \in Zero([\![l_2]\!]_M + \varepsilon_2 - \xi) =: Zero^+$ *that follows directly after* ζ^- *(and no other element from* $Zero^-$ *between). Consider furthermore that we have a* t_0, t_n *such that* $M \sim_{\varepsilon_1}^{[t_0,t_n]} M_{trans}$ *and* $t_n \in \Sigma$ *is directly before* ζ^- *(meaning no other* $t \in \Sigma$ *between).*

1. *If* $\forall 1 \leq i \leq n : \forall t \in \Sigma : t \notin [\zeta_i^-, \zeta_i^+]$ *and* $\frac{d}{dt}[\![l_2]\!](\zeta_i^-) < 0$ *then* $M \sim_{\varepsilon_1}^{[t_0,t_n]} M_{trans}$ *and* $M \sim_{\varepsilon_3}^{[t_{n+1},t_{n+m}]} M_{trans}$ *for an* $m > 1$. *This means, the switch happens immediately in both models. Just the precisions of the relevant input lines must be updated.*
2. *If there are* $t \in \Sigma \cap [\zeta_i^-, \zeta_i^+]$ *(let us say* $t_{n+1}, ..., t_{n+\nu}$*) and* $\exists \tau$ *as in the previous item, then* $M \sim_{\varepsilon_1}^{[t_0,t_n]} M_{trans}$, $M \sim_{\varepsilon_3}^{[t_{n+\nu+1},t_{n+m}]} M_{trans}$ *for an* $m > \nu$. *However, in the interval* $[t_{n+1}, t_{n+\nu}] =: \mathfrak{L}$, *we only have* $M \sim_{\varepsilon_{\mathfrak{L}}}^{\mathfrak{L}} M_{trans}$ *with* $\varepsilon_{\mathfrak{L}} := ||[\![l_1]\!]_M - [\![l_3]\!]_M|| + \varepsilon_1 + \varepsilon_3$. *This means, the switching may be time-delayed.*

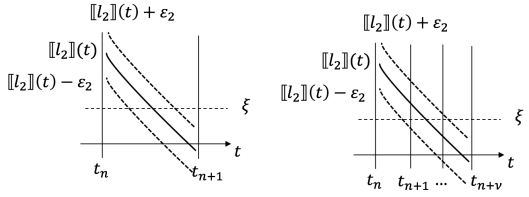

Figure 4. Illustration for the proof of Theorem 3

Proof. Figure 4 illustrates the proof.

The left image shows the first case in the theorem. The idea is that due to the fact that the values for the simulations at the control input l_2 may vary in the area of $\pm\varepsilon_2$ around $[\![l_2]\!]$, the crossing of the threshold ξ must take place for both curves $[\![l_2]\!] - \varepsilon_2$ and $[\![l_2]\!] + \varepsilon_2$ before another sample step is reached. If this cannot be guaranteed (right hand side picture), the switch may happen time-delayed and therefore an approximation $\varepsilon_{\mathfrak{L}}$, which consists of the sum of all differences, can only be guaranteed.

The complexity of the proof obligation can be reduced to make it easier for verification.

Corollary 1 *The first condition of Theorem 3 is fulfilled if* $[\![l_2]\!](t_n) - \varepsilon_2 - \xi > 0 \wedge \forall t \in [t_n, t_{n+1}] : \frac{d}{dt}[\![l_2]\!](t) < -\frac{2\varepsilon_2}{h}$. *The second condition is fulfilled if the same term is* $< -\frac{2\varepsilon_2}{\nu h}$.

Proof. For the first condition it is sufficient to check if at sample step t_n just before the crossing, we are still above the threshold and if the distance to the

threshold, namely $[\![l_2]\!](t_n) - \varepsilon_2 - \xi > [\![l_2]\!](t_{n+1}) + \varepsilon_2 - \xi$, since the second term (after $>$ needs to be below the threshold 0). This inequality can be transformed to $-\frac{2\varepsilon_2}{h} > \frac{[\![l_2]\!](t_{n+1}) - [\![l_2]\!](t_n)}{h}$, which is fulfilled if the derivative $\frac{d}{dt}[\![l_2]\!](t) < \frac{-2\varepsilon_2}{h}$ in the whole interval between t_n and t_{n+1}. The second condition follows analogously.

This corollary is especially helpful if M_{trans} expresses the analytic solution of M because in this case, we can calculate $\frac{d}{dt}[\![l_2]\!]$ easily.

Finally, we provide a criterion if we have non-isolated zero crossings.

Theorem 4 *Let M and M_{trans} be regular hybrid control flow Simulink models, $l, l_1, l_2, l_3, \xi, \varepsilon_i, \Sigma, Zero$ as in the above theorems, $\varepsilon_2 > 0$. Let without loss of generality the condition at the switch $\geq \xi$. If $Zero([\![l_2]\!] - \varepsilon_2 - \xi)$ consists of an interval $[t_1, t_2]$ (i.e., non-isolated zero crossings) and $\forall t \in \Sigma : t \notin [t_1, t_2]$, then $M \sim_{\varepsilon_1} M^{\mathfrak{L}}_{trans}$ for an interval $[t_1, t_2] \subseteq \mathfrak{L} \subseteq \mathfrak{I}$.*

Proof. The proof is analogous to the proof of Theorem 2. If the zero crossing does not coincide with a sample step, it is just not being recognised and therefore the switch does not happen and the precision remains.

Note that if the interval of non-isolated zero-crossings contains sample steps, we cannot guarantee that the precisions of approximately equivalent behaviour remains at ε_1 or ε_3 in general. In this case, only the precision that sums up all distances could be guaranteed.

The statements in the theorems can be used to perform a static analysis to obtain the global precision ε for the approximately equivalent behaviour of the hybrid systems straightforwardly. The conditions can be verified with the help of a Computer Algebra System (CAS), e.g. the Reduce and Resolve commands in Mathematica. Note that since the precision measures ε are actually functions of the form $\varepsilon(t, L, h)$, the user has the option to decrease the simulation step size in order to obtain a better result and consequently yield approximately equivalent behaviour of a desired precision or at least reduce the interval for the time-delayed diverging behaviour to an acceptable period.

As an example, consider the simple Simulink model from Figure 1 without the part +0.999 for simplicity, the threshold at the switch being 0. The ODE associated with the control input is $\frac{d}{dt}l_2(t) = l_1(t), \frac{d}{dt}l_1(t) = -l_2(t), l_1(0) = 1, l_2(0) = 0$. The solution is $l_2(t) = \sin(t)$. This is expressed in the transformed model. The perturbation ε_2 depends on the selected approximation method chosen in Simulink. The set $Zero := Zero([\![l_2]\!]) = Zero(\sin(t)) = \{k\pi | k \in \mathbb{Z}\}$. The set $Zero(\sin(t) - \varepsilon_2)$ shifts these zero crossings a bit to the left (depending on the actual value of ε_2, which depends on the chosen approximation method). The user must then verify that the sample step sizes $nh \notin Zero$ or potentially weaken this and find out how many sample step sizes are in $Zero \cap \Sigma$ to apply one of the cases of Theorem 3 with the help of a Computer Algebra System.

7 Related Work

For the design and application of refactorings in Simulink, several approaches exist. In [18], a taxonomy of model mutations is defined. In [3], a normalisation of

Simulink models is presented, which is used for clone detection in [9]. In [27], the authors present Simulink refactorings that allow subsystem and signal shifting in arbitrary layers. None of these approaches consider behavioural equivalence or correctness of the transformation.

There exists a broad variety of verification approaches for hybrid models, for a detailed introduction see for example [13],[19],[11],[12],[28]. In these papers, the notion of approximate bisimulation is introduced and utilised. Furthermore, in [4] abstractions of hybrid systems are introduced. There, hybrid systems are abstracted to purely discrete systems. [26] provides an overview over abstractions on hybrid systems. However, these works consider hybrid systems in general, with no reference to the semantics of Simulink and its characteristics. Another interesting approach for conformance in hybrid models is [1]. There, the authors introduce a new notion of approximately equivalent behaviour, which takes time step distances besides the distances of values into account. Furthermore, the authors calculate the conformance, i.e., approximately equivalent behaviour, based on running tests, e.g. between simulated model and generated code. However, they do not aim at refactorings at model level and do not consider behavioural equivalence on symbolic expressions on model level.

To reason about correctness for Simulink refactorings, a clear understanding of the Simulink semantics is required. However, the Mathworks documentation [25] defines the Simulink semantics only informally. Existing approaches for the formal verification of Simulink models typically overcome this problem by using transformations into some well-established formal language. For example, in [15], Simulink models are mapped to UCLID and the SMT (satisfiability modulo theories) solver UCLID is used for verification. In [8], this is done with the synchronous data flow language LUSTRE. In [5], an approach for contract based verification is presented. In this approach, the semantics is described via synchronous data flow graphs. In [21], the authors use Boogie, a verification framework developed at Microsoft Research, for verification. In [2], Simulink models are translated to hybrid automata. This enables further investigation of hybrid automata semantics, e.g., in [10] or [17]. However, none of these approaches provides a comprehensive semantics of Simulink. To the best of our knowledge, only [6] provides a direct and comprehensive formalisation of the Simulink semantics. In [22], [23], a first methodology to show correctness of refactorings of Simulink models that are solely discrete or continuous was provided. It is based on the formal operational semantics from [6]. There, a set of syntactical equations as an abstract representation from the Simulink models is derived, which describes the changes of signals over time on an abstract level. To deal with the challenge that traditional equivalence notions such as bisimulation are not suitable, the authors adapted the concept of approximate bisimulation.

Since there are no methodologies showing the correctness of Simulink refactorings, also to the best of our knowledge no approach dealing with Simulink models with control flow exists. There are some theories that are related with this topic, e.g. [20], [16], [14]. They all deal with stability and robustness, i.e., under which conditions hybrid systems do not show diverging behaviour under perturbation.

The basis of these works is the Lyapunov stability theory [24]. However, they neither aim at Simulink nor deal with the behavioural equivalence of the models.

8 Conclusion and Future Work

In this paper, we have presented a methodology for proving the correctness of refactorings for hybrid Simulink models with control flow. We have achieved this by extending the approach from [22,23]. In particular, we have sharpened the notion of the abstract representation and have defined a denotation, which expresses the result of the simulation as a function at each signal. This denotation also allows us to observe inner behaviour in the Simulink model. We also have defined which kind of models we support. Finally, we have provided sufficient conditions for approximate equivalent behaviour for regular Simulink models with control flow. Our approach is able to cover a large variety of industrially relevant models.

In future work, we aim at automating the approach. We were already successful in verifying correctness of transformations for purely discrete and continuous systems with the approach [23]. We plan to extend this to cover the approach in this paper as well. Furthermore, we plan to extend the presented approach by taking variable step size simulations into account. Our approach is already applicable for a broad variety of systems. However, for the continuous parts, it relies mainly on systems where the ODEs may be solved. We plan to strengthen our approach by providing assistance for systems with continuous parts that are not necessarily analytically solvable.

References

1. Abbas, H., Hoxha, B., Fainekos, G., Deshmukh, J.V., Kapinski, J., Ueda, K.: Conformance testing as falsification for cyber-physical systems. arXiv preprint arXiv:1401.5200 (2014)
2. Agrawal, A., Simon, G., Karsai, G.: Semantic translation of simulink/stateflow models to hybrid automata using graph transformations. Electronic Notes in Theoretical Computer Science 109, 43–56 (2004)
3. Al-Batran, B., Schätz, B., Hummel, B.: Semantic clone detection for model-based development of embedded systems. In: Model Driven Engineering Languages and Systems, pp. 258–272. Springer (2011)
4. Alur, R., Henzinger, T.A., Lafferriere, G., Pappas, G.J.: Discrete abstractions of hybrid systems. Proceedings of the IEEE 88(7), 971–984 (2000)
5. Boström, P.: Contract-based verification of simulink models. In: Formal Methods and Software Engineering, pp. 291–306. Springer (2011)
6. Bouissou, O., Chapoutot, A.: An operational semantics for simulink's simulation engine. ACM SIGPLAN Notices 47(5), 129–138 (2012)
7. Butcher, J.C.: Numerical methods for ordinary differential equations. Wiley, Chichester, 2. ed. edn. (2008), http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10232593
8. Caspi, P.: Translating discrete-time simulink to lustre. In: ACM Transactions on Embedded Computing Systems (TECS). pp. 779–818. New York (2005)

9. Deissenboeck, F., Hummel, B., Juergens, E., Pfaehler, M., Schaetz, B.: Model clone detection in practice. In: Proceedings of the 4th International Workshop on Software Clones. pp. 57–64. ACM, New York (2010)
10. Edalat, A., Pattinson, D.: Denotational semantics of hybrid automata. In: Foundations of Software Science and Computation Structures. Lecture Notes in Computer Science, vol. 3921, pp. 231–245 (2006)
11. Girard, A.: Approximate bisimulations for constrained linear systems. Automatica pp. 1307–1317 (2005)
12. Girard, A., Pappas, G.J.: Approximate bisimulations for nonlinear dynamical systems. In: Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. pp. 684–689. IEEE (2005)
13. Girard, A., Pappas, G.J.: Approximate bisimulation: A bridge between computer science and control theory. European Journal of Control 17(5), 568–578 (2011)
14. Goebel, R., Sanfelice, R.G., Teel, A.R.: Hybrid Dynamical Systems: modeling, stability, and robustness. Princeton University Press (2012)
15. Herber, P., Reicherdt, R., Bittner, P.: Bit-precise formal verification of discrete-time matlab/simulink models using smt solving. In: Proceedings EMSOFT '13 Proceedings of the Eleventh ACM International Conference on Embedded Software. IEEE Press (2013)
16. Lazar, M.: Model predictive control of hybrid systems: Stability and robustness. Dissertation, Technische Universiteit Eindhoven, Eindhoven (2006)
17. Lee, E.A., Zheng, H.: Operational semantics of hybrid systems. In: Hybrid Systems: Computation and Control, pp. 25–53. Springer (2005)
18. Matthew, S.: Towards a taxonomy for simulink model mutations. In: Software Testing, Verification and Validation Workshops (ICSTW), 2014 IEEE Seventh International Conference. pp. 206–215. IEEE (2014)
19. Pappas, G.J.: Bisimilar linear systems. Automatica 39(12), 2035–2047 (2003)
20. Pettersson, S., Lennartson, B.: Stability and robustness for hybrid systems. In: Proceedings of the 35th IEEE Conference on Decision and Control. IEEE (1996)
21. Reicherdt, R., Glesner, S.: Formal verification of discrete-time matlab/simulink models using boogie. In: Software Engineering and Formal Methods, pp. 190–204. Springer (2014)
22. Schlesinger, S., Herber, P., Göthel, T., Glesner, S.: Towards the verification of refactorings of hybrid simulink models. In: Lisitsa, A., Nemytyk, A., Pettorossi, A. (eds.) Proceedings of the Third International Workshop on Verification and Program Transformation. EPTCS, vol. 199, p. 69 (2015), http://eptcs.web.cse.unsw.edu.au/content.cgi?VPT2015#EPTCS199.5
23. Schlesinger, S., Herber, P., Göthel, T., Glesner, S.: Proving Transformation Correctness of Refactorings for Discrete and Continuous Simulink Models (2016)
24. Teschl, G.: Ordinary differential equations and dynamical systems, Graduate studies in mathematics, vol. 140. American Math. Soc, Providence, RI (2012)
25. The Mathworks, I.: Simulink documentation website, http://de.mathworks.com/help/simulink/
26. Tiwari, A.: Abstractions for hybrid systems. Formal Methods in System Design 32(1), 57–83 (2008)
27. Tran, Q.M., Wilmes, B., Dziobek, C.: Refactoring of simulink diagrams via composition of transformation steps. In: ICSEA 2013, The Eighth International Conference on Software Engineering Advances. pp. 140–145 (2013)
28. Van der Schaft, AJ: Equivalence of dynamical systems by bisimulation. Automatic Control, IEEE Transactions on 49(12), 2160–2172 (2004)