Proving Transformation Correctness of Refactorings for Discrete and Continuous Simulink Models

Sebastian Schlesinger, Paula Herber, Thomas Göthel, and Sabine Glesner
Software Engineering for Embedded Systems
Technische Universität Berlin
Email: {Sebastian.Schlesinger, Paula.Herber, Thomas.Goethel, Sabine.Glesner}@tu-berlin.de

Abstract—**MATLAB/Simulink is a state-of-the-art tool for model-driven engineering of embedded systems. Simulink enables engineers to model continuous and discrete parts of a system together in hybrid models. In such models, complexity reduction via refactoring plays an important role. However, formal verification of the equivalence between a hybrid Simulink model and its refactored counterpart is still an open problem. One challenge is that for many refactorings, equivalent behaviour can only be shown to be 'close' to each other rather than being the 'same'. To solve this problem, we propose a methodology to show behavioural equivalence based on approximate bisimulation. Our main contributions are a sound abstract representation for Simulink models that serves as a basis for proving equivalence, that we adapt the concept of approximate bisimulations to the operational Simulink semantics, and a methodology that enables the designer to prove transformation correctness. Our approach is applicable to both discrete and continuous Simulink models. With that, we provide an ideal starting point for the verification of hybrid models that integrate discrete and continuous model parts.**

Keywords–formal verification; transformation correctness; refactoring; MATLAB/Simulink; approximate bisimulation

I. Introduction

MATLAB/Simulink is a tool and de facto standard in model-driven engineering (MDE) in many industries, e.g., in the automotive and aerospace sectors. To control the complexity of Simulink models and to ensure their adherence to certain guidelines, *refactorings* are often used. Refactorings are model transformations that improve the inner structure of the model by keeping the behaviour of source and target model equivalent. Apart from reducing complexity and establishing compliance to defined guidelines for the models, refactorings can offer various perspectives of the model to the engineer - behaviourally equivalent, but different in other aspects. This helps the engineer to better understand the system and to find the proper refinement in forthcoming design phases to fulfil the requirements of the desired system. Other potential benefits of refactorings can be improved simulation performance or increased simulation precision after the transformation.

However, as Simulink is often used in safety-critical applications, it is crucial to ensure that refactorings preserve the intended behaviour of a Simulink model. While it is generally possible to validate the correctness of refactorings by simulation and testing, this does not provide any guarantees that the behaviour is preserved for all possible input scenarios. In this context, formal verification plays an important role. The ISO 26262 [1] states that it is highly recommended to formally prove system behaviour against the specification to cope with high Automotive Safety Integrity Levels (ASILs). However, formal verification of hybrid Simulink models, and in particular the formal verification of refactorings of hybrid models is a major challenge. The main difficulty is that for many refactorings, equivalent behaviour can only be shown to be 'close' to each other rather than being the 'same'.

In this paper, we present a methodology to enable the verification of behavioural equivalence of discrete or continuous Simulink models, in the sense that two models are close to each other. We are confident that our approach can be extended to actual hybrid models later in future work. The main challenge we address is that the standard concept for showing behavioural equivalence, namely bisimulation, is not applicable in cases where hybrid models are numerically approximated by Simulink. However, in such cases, we can prove that the values of the next simulation steps are *approximately* the same, i.e., they are located in an *'epsilon tube'* around the actual solution, which does not need to be known to the designer. To achieve this, we provide a new notion of equivalence for hybrid MATLAB/Simulink models by adapting the existing concepts for *approximate bisimulation* as described in, e.g., [2] to the operational Simulink semantics given in [3].

Our main contributions are the following.

1) We adapt the concept of approximate bisimulation for Simulink.
2) We provide an abstract and formally well-defined semantics for Simulink.
3) We propose a methodology for transformation correctness of hybrid Simulink models.

The main idea of our methodology to prove transformation correctness is that we provide an abstract representation for Simulink models and perform the verification on this abstract level using approximate bisimulation. We put the interpretation of the abstract representations on a firm footing by proving soundness with respect to an existing operational Simulink semantics [3]. To prove behavioural equivalence between a source model and its refactored counterpart, a designer can use our abstract representation to compute boundaries for the ranges in which the values are contained after each simulation step. From these boundaries, approximate bisimulation between the original Simulink models can be derived.

Note that in this paper, we focus on systems that are either discrete or continuous. It is part of future work to combine the findings to be able to handle fully hybrid models as well.

The rest of this paper is structured as follows: In Section II, we introduce the operational semantics of MATLAB/Simulink and the concept of approximate bisimulation. In Section III, we discuss related work. Section IV provides a general overview over our approach. In Section V, we provide our adaptation of approximate bisimulation to Simulink. In Section VI, we present our abstract representation and discuss its soundness with respect to the operational semantics. In Section VII, we examine how behavioural equivalence between Simulink models can be established using our abstract representation together with our adapted concept of approximate bisimulation. We conclude in Section VIII.

II. Background

In this section, we briefly summarise the necessary background to understand the remainder of this paper.

A. Simulink

Simulink is a widely used modeling language for dynamic systems. It is based on Matlab, and both products are developed by The MathWorks [4]. In Simulink, dynamic systems are modelled as block diagrams. They can then be simulated, and, with further software packages, it is also possible to automatically generate code. Simulink provides a large library of predefined blocks. Additionally, user-defined libraries and block types can be added.

In this paper, we consider five kinds of Simulink blocks: unsampled or direct feed-through blocks (e.g., arithmetic blocks), discrete time blocks (e.g., Unit Delay, Discrete Integrator), continuous time blocks (e.g., Integrator), sink blocks (e.g., Scope) and source blocks (e.g., Constant, Sine Wave, Ramp). Each block can have inports and outports. These are the interfaces via which the blocks are connected. We speak of *discrete* models if the model consists only of discrete and unsampled blocks and *continuous* models if the model consists only of continuous and unsampled blocks. Finally, a *hybrid* model is a model with all block types.

Simulink Operational Semantics

To capture the behaviour of Simulink models, we use the operational semantics described by Bouissou and Chapoutot [3]. There, a state is a mapping $\sigma : V \to \mathbb{R}$ where $V = \{l_i | 1 \leq i \leq n_b\} \cup \{d_i | 1 \leq i \leq n_d\} \cup \{x_i | 1 \leq i \leq n_x\} \cup \{t, h\}$. The numbers n_b, n_d and n_x express the amounts of variables for the output of blocks, discrete and continuous respectively. Every block has a unique ouput variable l_i, discrete blocks have an internal variable d_i, and continuous blocks an internal variable x_i in addition. The variable t is used for the simulation time and h stands for the simulation step size. Parameters that are provided by the user are denoted by a function π. For instance, $\pi(t_0)$ expresses the simulation start time, $\pi(t_{end})$ the simulation end time, $\pi(h_0)$ the initial simulation step size. Bouissou and Chapoutot then assign each block with a set of equations, examples are $l_1 = l_2 + l_3$, $l_i =_S d; \bar{d} =_S l_1$, $\dot{x} = l_1, x(0) = init$. These equations are evaluated by the operational semantics, which is defined as a set of inference rules. In the following, we provide a brief summary of the most important inference rules.

The global simulation rule states that if the simulation time is not expired (in which case nothing happens), the simulation consists of three steps that are consecutively evaluated:

1) The $M-$transition (*major step* transition) evaluates equations of the form $l = e$ and $l =_S e$, i.e., unsampled equations and outputs of discrete (sampled) blocks.
2) The $u-$transition (*update* transition) evaluates the internal variables of discrete blocks, i.e., equations of the form $\bar{d} =_S l$.
3) The $s-$transition (*solver* transition) evaluates the internal variables of the continuous blocks, i.e., equations of the form $\dot{x} = l$.

During the evaluation of these transitions, the following rules apply:

- The evaluation of expressions and predicates is performed by $o-$transitions: variables l evaluate to the value in the current state $\sigma(l)$, constants to the constant value, functions are evaluated by applying them on the values of their arguments, predicates analogously.
- Equations with an index S, e.g., $l_o =_S f(l_1, ..., l_n)$ only lead to a state change if $\sigma(t) \in S$, i.e., the sample time of the block is reached by the simulation time. If $\sigma(t) \in S$, the expression on the right hand side is evaluated by $o-$transitions to calculate the new value for $\sigma(l_o)$.

The solver transitions to calculate equations of the form $\dot{x} = l$ ($\dot{x}$ denotes the derivative of x) are more complex. Simulink approximates the value x_{n+1} at point $t_n + h_n$ starting from the value x_n at time index t_n by applying an approximation technique defined by the user, e.g., Euler method or one of the Runge-Kutta methods [5]. The solver transition finishes with the calculation of the subsequent simulation step size. Please note that all blocks are simultaneously evaluated to calculate the succeeding state according to the operational semantics, i.e., it is a synchronous semantics.

B. Approximate Bisimulation

Our goal is to prove equivalent behaviour of source model and refactored target model. As indicated in Section II, in case a model contains continuous blocks, Simulink does not compute the actual solution, but an approximation. Hence, although intuitively source and target model may express the same solution function, it is not possible to show exact equality. We therefore require a notion of equivalence that relaxes equality. To this end, we use approximate bisimulation, as defined in [2], [6].

Definition 1 (Approximate Bisimulation)**.** *Let* $T_1 = (Q_1, \Sigma, \to_1, Q_1^0, \Pi, \langle\langle . \rangle\rangle_1)$ *and* $T_2 = (Q_2, \Sigma, \to_2, Q_2^0, \Pi, \langle\langle . \rangle\rangle_2)$ *be two labelled transition systems (LTS), where* Q_i *are the sets of states,* $Q_i^0 \subseteq Q_i$ *the sets of initial states,* Σ *the sets of input alphabet (labels),* $\to_i \subseteq Q_i \times \Sigma \times Q_i$ *the transition relations,* Π *the set of observations and* $\langle\langle . \rangle\rangle_i : Q_i \to \Pi$ *mappings of states to observations (*$i \in \{1, 2\}$*; note that* Σ *and* Π *is in both LTS the same). Let furthermore* Π *be a metric space with metric* $d : \Pi \times \Pi \to \mathbb{R}$*. The metric allows us to measure distances in the set of observables.*

A bisimulation relation $B_\varepsilon \subseteq Q_1 \times Q_2$ *of precision* ε *is a relation fulfilling the following properties* $\forall (q_1, q_2) \in B_\varepsilon$*.*

1) $d(\langle\langle q_1 \rangle\rangle_1, \langle\langle q_2 \rangle\rangle_2) \leq \varepsilon$

2) $\exists q_1' \in Q_1 : q_1 \xrightarrow{\alpha} q_1' \Rightarrow \exists q_2' \in Q_2 : q_2 \xrightarrow{\alpha} q_2' \wedge (q_1', q_2') \in B_\varepsilon$
3) $\exists q_2' \in Q_2 : q_2 \xrightarrow{\alpha} q_2' \Rightarrow \exists q_1' \in Q_1 : q_1 \xrightarrow{\alpha} q_1' \wedge (q_1', q_2') \in B_\varepsilon$

The LTS are bisimilar with precision ε, denoted as $T_1 \sim_\varepsilon T_2$, if $B_\varepsilon \subseteq Q_1 \times Q_2$ exists such that

4) *B_ε is a bisimulation relation of precision ε,*
5) $\forall q_1 \in Q_1^0 \exists q_2 \in Q_2^0 : (q_1, q_2) \in B_\varepsilon$
6) $\forall q_2 \in Q_2^0 \exists q_1 \in Q_1^0 : (q_1, q_2) \in B_\varepsilon$

Before we come to the approach, we discuss related work in the next section.

III. Related Work

For the design and application of refactorings in Simulink, several approaches exist. In [7], a taxonomy of model mutations is defined. In [8], a normalisation of Simulink models is presented, which is used for clone detection in [9]. In [10], the authors present Simulink refactorings that allow subsystem and signal shifting in arbitrary layers. None of these approaches considers behavioural equivalence or transformation correctness.

There exists a broad variety of verification approaches for hybrid models, for a detailed introduction see for example [2], [6], [11]–[15]. In these papers, the notion of approximate bisimulation is introduced and utilised. However, none of them targets hybrid systems described in Simulink and it is a non-trivial challenge to transfer them to the specifics of the Simulink semantics and make use of its special characteristics such as deterministic behaviour. To reason about transformation correctness for Simulink models, a clear understanding of the Simulink semantics is required. However, the Mathworks documentation [4] 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 [16], Simulink models are mapped to the verification system UCLID and the satisfiability modulo theories (SMT) solver UCLID is used for verification. In [17], this is done with the synchronous data flow language LUSTRE. In [18], an approach for contract based verification is presented. In this approach, the semantics is described via synchronous data flow graphs. In [19], the authors use Boogie, a verification framework developed at Microsoft Research, for verification. For this, the semantics of discrete Simulink models is again translated to the input language of a verification tool, in this case Boogie. In [20], Simulink models are translated to hybrid automata. This enables further investigation of hybrid automata semantics, e.g., in [21] or [22]. However, none of these approaches provide a formal semantics for Simulink, as they all use some transformation into existing formal languages. This also constrains the supported subset of Simulink models in all cases.

To the best of our knowledge, only Bouissou and Chapoutot [3] provide a direct formalization of the Simulink semantics. The authors consider a Simulink model as a set of equations evaluated following an operational semantics, which is defined using a set of inference rules. In our approach, we use this operational semantics as a formal foundation to capture the behaviour of Simulink models. In contrast to Bouissou and Chapoutot, we utilise the formal semantics to derive a set of syntactical equations as an abstract representation, which describes the changes of signals over time on an abstract level. Together with the concept of approximate bisimulation, this enables the verification of approximate behavioural equivalence between a source and a target model.

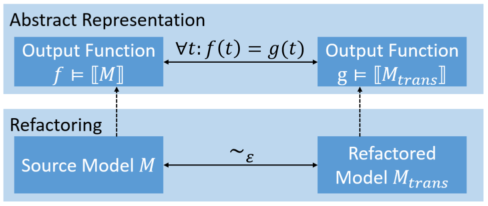
Abstract Representation
Output Function $f \vDash [\![M]\!]$
$\forall t: f(t) = g(t)$
Output Function $g \vDash [\![M_{trans}]\!]$
Refactoring
Source Model M
$\sim_\varepsilon$
Refactored Model M_{trans}

Figure 1. Overview over our approach

IV. Transformation Correctness Approach for Discrete and Continuous Simulink Models

In this paper, we present a methodology for proving transformation correctness of Simulink refactorings. Our approach is applicable to both discrete and continuous models. To keep the notation simple, we assume without loss of generality that there is only one output block (where the observation takes place) in the model. If there was more than one output block, our approach just needed to be repeated for each. One of the key ideas of our transformation correctness approach is that we define an abstract representation for Simulink models. The *Abstract Representation* (AR) links the outgoing with the incoming signals of a given Simulink model by describing the effect of the blocks via equations in a mathematical view. The resulting set of equations can be interpreted by an output function $f \models [\![M]\!]$ that fulfils the conjunction of all equations of the respective equation set. This function describes basically how the output (meaning the observation) is linked with the input. Using our abstract representation, we can show that two simulated models behave approximately equivalent up to a certain precision by proving that the output functions yield the same values for all possible simulation steps.

The overall concept is depicted in Figure 1. There, we assume that a source model M and its transformed refactoring M_{trans} are given. To prove approximate bisimulation ($\sim_\varepsilon$) between the source model and the refactoring, we compute abstract representations for both of them and then prove that $\forall t : f(t) = g(t)$ holds.

In the following sections, we first present our adaptation of approximate bisimulation for Simulink and our abstract representation together with its interpretation using the concept of an output function that captures the semantics of a given Simulink model. Then, we present our methodology for showing behavioural equivalence of two Simulink models using our abstract representation together with our adapted notion of approximate bisimulation. Note that in contrast to the operational semantics defined in [3], our abstract interpretation provides a mathematical interpretation, i.e., it describes the behaviour of a given Simulink model using an exact calculation rather than an approximation or simulation. Nevertheless, we can show that our abstract representation is sound with respect to the operational semantics defined in [3].

Limitations and Assumptions

As representatives for discrete and continuous blocks, we currently only support Unit Delay and Integrator respectively. All Unit Delays must have the same sample times because currently we do not support the rate transition block, which would be necessary otherwise. 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. Furthermore, the following Simulink items are currently not supported:

- buses (multiple signals on one signal line) and multi-dimensional signals
- atomic and virtual subsystems
- blocks that alter the control flow, e.g., Switches
- full hybrid systems, i.e., models containing both discrete and continuous parts.

With our current methodology, many industrially relevant models are already covered. Our approach can easily be extended to other blocks.

V. Approximate Bisimulation for Simulink

In this section, we adapt the notion of approximate bisimulation as introduced in Section II-B to Simulink. To this end, we define the semantics of Simulink as a labelled transition system (LTS) and provide a metric to measure observation distances.

First, we define the syntax of a Simulink model as follows.

Definition 2 (Simulink Syntax)**.** *A Simulink model is a tuple (B, E, I, O). B is a finite set of blocks, $E \subseteq B \times \mathbb{N} \times B$ the signal lines together with the arity of the incoming signal (the arity is important because some functions defined by blocks are not commutative), $I \subseteq B$ the set of sources (for which $\forall b' \in I \forall b \in B \forall n \in \mathbb{N} : (b, n, b') \notin E$ applies), $O \subseteq B$ the set of sinks (for which $\forall b \in O \forall b' \in B \forall n \in \mathbb{N} : (b, n, b') \notin E$ applies).*

Please note that we often abbreviate signal lines $(b_i, n, b') \in E$ by variables l_i with $i \in \mathbb{N}$. Using our Simulink syntax, we can now define the semantics as an LTS.

Definition 3 (Labelled Transition System of Simulink)**.** *The LTS for a Simulink model $M = (B, E, I, O)$ is*

- *$Q \subseteq \mathbb{R}^V$ - the set of states (each variable is assigned with a value)*
- *We do not need a set of inputs Σ.*
- *$\rightarrow \subseteq Q \times Q$ is defined by the operational semantics*
- *$Q^0 \subseteq Q$ assigns the initial values to each variable (given as parameter in the Simulink model)*
- *$\Pi \subseteq (\mathbb{R} \cup \{\perp\})^V$ - the set of observations*
- *$\langle\langle . \rangle\rangle : Q \rightarrow \Pi$, $\langle\langle \sigma \rangle\rangle(v) := \sigma(v)$ if v is a variable associated with a sink block $b \in O$, $\perp$ otherwise. (The observation map provides the value at the sink of the model - and an undefined symbol $\perp$ otherwise)*

To keep notation simple, we denote the unique output variable of a block $b_i \in B$ with v_i from now on. The metric we use for approximate bisimulation is defined as $d_\infty : \Pi \times \Pi \rightarrow \mathbb{R}$, $d_\infty(\sigma_1, \sigma_2) := \|\sigma_1 - \sigma_2\|_\infty$ with $\|.\|_\infty : \Pi \rightarrow \mathbb{R}$, $\|\sigma\|_\infty := \max_{v \in \bigcup_{b \in O} : \text{v variable of b}}(\sigma(v))$. It takes the biggest distance of all observations.

TABLE I. EXAMPLE $eqExtract$ DEFINITIONS

Block type	$eqExtract(l)$
Input	$\{l(t) = in(t)\}$
Sine Wave	$\{l(t) = \sin(t)\}$
Math Operations	$\{l(t) = f(l_1(t), ..., l_n(t))\}$
Unit Delay	$\{l(t + h) = l_1(t), l(\pi(t_0)) = \pi(init_l)\}$
Integrator	$\{\frac{d}{dt} l(t) = l_1(t), l(\pi(t_0)) = \pi(init_l)\}$

VI. Abstract Representations and their Interpretation

In this section, we introduce the abstract representations (AR). Equations in the AR (we denote the set of AR equations as Eq_a) are of the form $l_i(t) = f(l_1(t), ..., l_n(t))$, $l_i(t + \lambda h) = l_1(t)$ or $\frac{d}{dt} l_i(t) = l_1(t)$, where f is an expression depending on other variables, h is the (synchronous) sample time, λ_i are integral coefficients.

Definition 4 (Abstract Representation)**.** *For a Simulink model $M = (B, E, I, O)$, the abstract representation is a function $[\![.]\!]$ that maps M to a set of equations $[\![.]\!] : SM \rightarrow \mathcal{P}(Eq_a)$, $[\![M]\!] = \bigcup_{l \in E} eqExtract(l)$, where SM is the set of Simulink models. The function $eqExtract : E \rightarrow \mathcal{P}(Eq_a)$ describes how the outgoing signal of a block is mathematically described depending on its incoming signals and must be defined according to the semantics of each block.*

Some examples for definitions of the function $eqExtract$ are shown in Table I. Note that Unit Delay and Integrator blocks both need an initial value, given as parameter. This is reflected by the second equations in these cases.

On the basis of our definition of an abstract representation of Simulink models, we can now define an output function that abstractly captures the semantics of a given Simulink model by interpreting our abstract representation.

Definition 5 (Interpretation of an Abstract Representations)**.** *Let $M = (B, E, I, O)$ a Simulink model with $\#O = 1$ (for simplicity) and simulation period $\mathfrak{I} := [\pi(t_0), \pi(t_{end})]$. Let furthermore $[\![M]\!] := \{eq_1, ..., eq_n\}$ the abstract representation. An interpretation of the abstract representation is a function $f : \mathfrak{I} \rightarrow \mathbb{R}$ that fulfils all equations of $[\![M]\!]$ for all $t \in \mathfrak{I}$. We denote with $f \models [\![M]\!]$ that f fulfils $\forall t : eq_1 \wedge eq_2 \wedge ... \wedge eq_n$.*

Please note that this function always exists if the input signals are continuous. For the uniqueness of the interpretation of a continuous model, Lipschitz-continuity must be guaranteed as well [23].

We now establish the soundness of the AR with the operational semantics. Due to space constraints, we only provide a proof sketch here.

Lemma 1 (Soundness of the Abstract Representation)**.** *Let M be a Simulink model that defines an LTS $(Q, \rightarrow, Q^0, \Pi, \langle\langle . \rangle\rangle)$. Then for all $\sigma \in Q$, the equations in $[\![M]\!]$ for unsampled and discrete blocks hold if we replace terms $l_i(t)$ by $\sigma(v_i)$, $l_i(t + h)$ by $\sigma'(v_i)$ (with $\sigma \rightarrow \sigma'$, $\sigma'(t) \leq \sigma(t) + \sigma(h)$) etc. For continuous blocks, we construct functions $l_{h \rightarrow 0}$. To define $l_{h \rightarrow 0}$ for a signal l, we define a function $l_{(t_n)_n} : E \rightarrow \mathbb{R}^{\mathfrak{I}}$ where $\mathfrak{I}$ is the simulation interval and $(t_n)_n$ is a sequence of simulation steps. $l_{(t_n)_n}(\tau)$ is defined as follows:*

$$l_{(t_n)_n}(\tau) = \frac{\sigma'(v) - \sigma(v)}{t_{i+1} - t_i}(\tau - t_i) + \sigma(v)$$

where $\sigma(t) = t_i, \sigma'(t) = t_{i+1}, t_i \leq \tau < t_{i+1}$.

We obtain $l_{h\to 0}$ out of the family of functions $l_{(t_n)_n}$ with simulation step size $t_{n+1} - t_n \to 0$. Then the equations of the form $\frac{d}{dt}l(t) = l_1(t)$ hold if we replace all $l(t)$ by $l_{h\to 0}(t)$ etc.

The function $l_{(t_n)_n}$ assigns a value to each $t \in \mathfrak{I}$: If t is a sample time in the sequence $(t_n)_n$, i.e., if an i exists with $t = t_i$, then $l_{(t_n)_n}(t)$ is the result of the execution of the Simulink semantics. If t is between two sample steps, the value is on the line connecting the points $(t_i, \sigma(v))$ and $(t_{i+1}, \sigma'(v))$.

Proof sketch: The unsampled and discrete cases follow from the operational semantics because unsampled blocks are directly evaluated, and discrete blocks only update if sample time is reached. For the continuous case, the function $l_{h\to 0}$ exists because the family of functions of the form $l_{(t_n)_n}$ is equicontinuous, bounded and all functions of this family are defined on the compact interval $\mathfrak{I}$. Consequently, according to the theorem of Arzela-Ascoli [24], a limit function $l_{h\to 0}$ exists towards which the functions uniformly converge. According to the semantics, the integrator block performs an approximation technique such as Euler. This yields immediately that $\frac{d}{dt}l_{h\to 0}(t) = l_{1,h\to 0}(t)$ holds. ■

We have now established that our AR is sound with respect to the operational semantics defined in [3]. Especially, we have put syntactical constructs that were already introduced in [3] on a firm footage regarding the interpretation with respect to an output function that mathematically captures the semantics of a given model.

VII. Behavioural Equivalences

We have established a sound abstract representation for Simulink models. Its interpretation yields a concise form of an output function, i.e., it expresses what the Simulink model actually does in the form of equations. The idea of our approach is that the designer verifies that the abstract representations of source and target model, which are either discrete or continuous, yield the same values on the simulation interval. From this, according to the next theorem, follows the approximately equivalent behaviour of both models.

Theorem 1 (Behavioural Equivalence of Simulink Models)**.** *Let M and M_{trans} be Simulink models (cf. Section IV for the limitations, $\#O = 1$ for simplicity), $\mathfrak{I} = [\pi(t_0), \pi(t_{end})]$ the common simulation interval. Let furthermore the interpretations of the abstract representations $g \models [\![M]\!]$ and $h \models [\![M_{trans}]\!]$ be equal during the simulation interval, i.e., $\forall t \in \mathfrak{I} : g(t) = h(t)$ holds.*

Then $M \sim_0 M_{trans}$ if the models are unsampled or discrete. If M is continuous, $[\![M]\!]$ expressing an ordinary differential equation (ODE; after re-arrangement of the equations) $\frac{d}{dt}y(t) = f(t, y(t)), y(\pi(t_0)) = \xi$ with f being Lipschitz-continuous in y, i.e., $||f(t, y_1) - f(t, y_2)|| \leq L||y_1 - y_2||$ for all t, y_1, y_2. Then we have $M \sim_\varepsilon M_{trans}$ with

$$\epsilon = \sup_{n=0,\ldots} (|t_n^t - t_n^s|) + \sup_{n=0,\ldots} (||\varphi(t_n^t) - \varphi(t_n^s)||) + \varepsilon_s + \varepsilon_t,$$

where t_n^s is the sample time of M (source model) at the $n-th$ step, t_n^t analogously for M_{trans} (target model). ε_s and ε_t are the global errors between the approximations performed by Simulink for M and M_{trans} and the mathematical solution of the ODE, i.e., the unique function that actually solves the ODE rather than approximating it.

If source and target model are sampled with the same fixed step size, then the error reduces to $\varepsilon = \varepsilon_s + \varepsilon_t$.

Proof sketch: For unsampled and discrete models, the precision ε is 0, i.e., the systems are bisimilar because the ARs are sound (cf. Lemma 1) and the operational semantics evaluates the expressions by providing exact and deterministic values. In the continuous case, let us abbreviate the approximated value at the $n-$th time step in the source model by $\psi_s(t_n^s)$ and for the target model by $\psi_t(t_n^t)$ respectively and the mathematical solution by φ. Then, with triangular inequality follows $\varepsilon \leq |t_n^t - t_n^s| + ||\psi_s(t_n^s) - \psi_t(t_n^t)|| \leq ||\psi_s(t_n^s) - \varphi(t_n^s)|| + ||\psi_t(t_n^t) - \varphi(t_n^t)|| + ||\varphi(t_n^s) - \varphi(t_n^t)|| = |t_n^t - t_n^s| + \epsilon_s + \epsilon_t + ||\varphi(t_n^t) - \varphi(t_n^s)||$. ■

Note that y and f in the continuous case can be multidimensional, e.g., $y = (y_1, y_2)$, $f = (f_1(t, y_1(t), y_2(t)), f_2(t, y_1(t), y_2(t)))$. This occurs if the system consists of more than one Integrator block.

Also note that the mathematical solution φ always exists and is unique under the assumptions of the theorem [23]. The ODEs do not need to be analytically solvable, i.e., our approach is applicable even though the solution may not be phraseable as a closed expression. The errors ε_s and ε_t depend on the approximation technique defined by the user in Simulink. For example, the Euler method yields

$$\varepsilon_{s/t} = \frac{1}{2}(\text{len}(\mathfrak{I}))e^{(\text{len}(\mathfrak{I})L)} \sup_{t\in\mathfrak{I}} \left(|\frac{d^2}{dt^2}\varphi(t)| \right) \sup_{n=0,\ldots} h_n^{s/t}$$

with the Lipschitz-constant L for f, $h_n^{s/t}$ the simulation step sizes at the $n-$th step, $\varphi : \mathfrak{I} \to \mathbb{R}$ the mathematical solution of the ODE and $\text{len}(\mathfrak{I}) = \pi(t_{end}) - \pi(t_0)$ the length of the simulation interval [23]. Euler has the worst approximation to the mathematical solution and consequently the precision ε can be improved by taking more advanced techniques available in Simulink into account. Also note that in case of fixed and equal sample step sizes on both models, the error reduces to just $\varepsilon = \varepsilon_s + \varepsilon_t$.

One important class of refactorings that immediately results from the theorem is if the ODE of M is analytically solvable and M_{trans} expresses the analytical solution directly. In this case, $\varepsilon_t = 0$ and M_{trans} is significantly less complex than M because usually feedback loops are removed. However, our approach is not restricted to this case. We also support refactorings that for instance keep the Integrator and transform something else mathematically equivalent.

It should also be mentioned that ε in the continuous case increases with the simulation interval size in the general case. However, we are confident to obtain certain classes of functions that allow to show that the global error of the approximation techniques is bounded for all $t \in [\pi(t_0), \infty[$.

Example

Let us go through a simple continuous example depicted in Figure 2 to demonstrate the approach. The model in the left is the source model M, on the right the target model M_{trans}. The labels and parameters are shown as well. Together with the models it is given that the expression for l_2 in M_{trans} solves the ODE in M. Suppose now that the designer wants to verify the behavioural equivalence of both models. In the first step, the abstract representations are extracted. $[\![M]\!] = \{\frac{d}{dt}l_1(t) =$

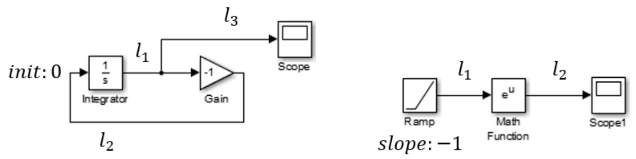

Figure 2. Refactoring Example

$l_2(t), l_1(0) = 1, l_2(t) = -l_1(t), l_3(t) = l_1(t)\}$, $M_{trans} = \{l_1(t) = -t, l_2(t) = e^t\}$. The observation at model M is $l_3(t)$, at model M_{trans} it is $l_2(t)$ because these signal lines are connected with an output block. The designer now has to establish that the interpretations $f \models [\![M]\!]$ and $g \models [\![M_{trans}]\!]$ fulfil $\forall t \in \mathfrak{I} : f(t) = g(t)$. To see that, the interpretations, i.e., the observations in both models must be equal. For M_{trans} we obtain this immediately by extracting $g(t) = e^{-t}$. For M, we need the solution of the ODE, which is given by M_{trans} as previously stated, i.e., we obtain $f(t) = e^{-t}$. This immediately yields $f(t) = g(t)$ for all t. However, to complete the proof, we finally need to establish that e^{-t} indeed solves the ODE in M (and the initial value problem $e^0 = 1$). This yields the final proof obligations $\frac{d}{dt}e^{-t} = -e^{-t}, e^0 = 1$, which hold. The designer consequently can apply our theorem and obtains the approximately equivalent behaviour. The precision ε depends on the chosen approximation technique.

VIII. Conclusion and Future Work

In this paper, we have presented a methodology for the verification of refactorings of discrete and continuous Simulink models. We have extended the operational semantics from [3] to an abstract representation of Simulink models, where the semantics is mathematically captured using a concise output function, and have shown its soundness with the operational semantics. The abstract representation is better suitable for automation than the operational semantics. Furthermore, we have adapted a suitable notion of behavioural equivalence, the approximate bisimulation, to Simulink models. This is especially useful for continuous models, since Simulink does not provide exactly the same values for the refactored model in comparison to the original model. Altogether, we have provided a methodology that enables a designer to verify the equivalence of discrete and continuous Simulink models and their refactored counterpart up to a certain precision.

Our approach offers many possibilities for further research. Currently we are working on the automation of our approach. This includes an algorithm that extracts expressions from Simulink models and sets up proof obligations that are then being verified with the assistance of a computer algebra system. In future work, we plan to extend our approach to hybrid models that integrate discrete and continuous model parts. To achieve this, we plan to investigate the behaviour 1) of models that contain both continuous and discrete blocks within the same subsystem and 2) of models that contain control flow blocks, e.g., switches. For the former, we plan to make use of delay differential equations theory. The challenge for the latter is that small changes at the switch input can yield diverging behaviour. Hence, additional proof obligations must be considered to establish the approximate behaviour in these cases.

References

[1] International Organization for Standardization, “ISO 26262 - Road vehicles – Functional safety.”

[2] A. Girard and G. J. Pappas, “Approximate bisimulation: A bridge between computer science and control theory,” European Journal of Control, vol. 17, no. 5, 2011, pp. 568–578.

[3] O. Bouissou and A. Chapoutot, “An operational semantics for simulink’s simulation engine,” ACM SIGPLAN Notices, vol. 47, no. 5, 2012, pp. 129–138.

[4] I. The Mathworks, “Simulink documentation website,” retrieved: Jan 2016. [Online]. Available: http://de.mathworks.com/help/simulink/

[5] J. C. Butcher, Numerical methods for ordinary differential equations, 2nd ed. Wiley, 2008.

[6] A. Girard, “Approximate bisimulations for constrained linear systems,” Automatica, 2005, pp. 1307–1317.

[7] S. Matthew, “Towards a taxonomy for simulink model mutations,” in ICSTW 2014, IEEE International Conference. IEEE, 2014, pp. 206–215.

[8] B. Al-Batran, B. Schätz, and B. Hummel, “Semantic clone detection for model-based development of embedded systems,” in Model Driven Engineering Languages and Systems. Springer, 2011, pp. 258–272.

[9] F. Deissenboeck, B. Hummel, E. Juergens, M. Pfaehler, and B. Schaetz, “Model clone detection in practice,” in Proceedings of the 4th International Workshop on Software Clones. New York: ACM, 2010, pp. 57–64.

[10] Q. M. Tran, B. Wilmes, and C. Dziobek, “Refactoring of simulink diagrams via composition of transformation steps,” in ICSEA 2013, The Eighth International Conference on Software Engineering Advances, 2013, pp. 140–145.

[11] G. J. Pappas, “Bisimilar linear systems,” Automatica, vol. 39, no. 12, 2003, pp. 2035–2047.

[12] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, “Discrete abstractions of hybrid systems,” Proceedings of the IEEE, vol. 88, no. 7, 2000, pp. 971–984.

[13] A. Girard, “Approximate bisimulations for nonlinear dynamical systems,” in European Control Conference 2005. IEEE, 2005, pp. 684–689.

[14] Van der Schaft, AJ, “Equivalence of dynamical systems by bisimulation,” Automatic Control, IEEE Transactions on, vol. 49, no. 12, 2004, pp. 2160–2172.

[15] A. Tiwari, “Abstractions for hybrid systems,” Formal Methods in System Design, vol. 32, no. 1, 2008, pp. 57–83.

[16] P. Herber, R. Reicherdt, and P. Bittner, “Bit-precise formal verification of discrete-time matlab/simulink models using smt solving,” in Proceedings EMSOFT ’13. IEEE Press, 2013.

[17] P. Caspi, “Translating discrete-time simulink to lustre,” in ACM Transactions on Embedded Computing Systems, New York, 2005, pp. 779–818.

[18] P. Boström, “Contract-based verification of simulink models,” in Formal Methods and Software Engineering. Springer, 2011, pp. 291–306.

[19] R. Reicherdt and S. Glesner, “Formal verification of discrete-time matlab/simulink models using boogie,” in Software Engineering and Formal Methods. Springer, 2014, pp. 190–204.

[20] A. Agrawal, G. Simon, and G. Karsai, “Semantic translation of simulink/stateflow models to hybrid automata using graph transformations,” Electronic Notes in Theoretical Computer Science, vol. 109, 2004, pp. 43–56.

[21] A. Edalat and D. Pattinson, “Denotational semantics of hybrid automata,” in Foundations of Software Science and Computation Structures, ser. Lecture Notes in Computer Science, vol. 3921, 2006, pp. 231–245.

[22] E. A. Lee and H. Zheng, “Operational semantics of hybrid systems,” in Hybrid Systems: Computation and Control. Springer, 2005, pp. 25–53.

[23] E. A. Coddington, An Introduction to Ordinary Differential Equations, ser. Dover Books on Mathematics. Dover Publications, 2012.

[24] J. B. Conway, A course in functional analysis, 2nd ed., ser. Graduate texts in mathematics. Springer, 2010, vol. 96.