Robustness of Model-based Simulations

Georgios E. Fainekos, Sriram Sankaranarayanan, Franjo Ivančić, and Aarti Gupta
NEC Laboratories America, 4 Independence Way, Princeton, NJ 08540, USA
Email: {fainekos, srirams, ivancic, agupta}@nec-labs.com

***Abstract*—This paper proposes a framework for determining the correctness and robustness of simulations of hybrid systems. The focus is on simulations generated from model-based design environments and, in particular, Simulink. The correctness and robustness of the simulation is guaranteed against floating-point rounding errors and system modeling uncertainties. Toward that goal, self-validated arithmetics, such as interval and affine arithmetic, are employed for guaranteed simulation of discrete-time hybrid systems. In the case of continuous-time hybrid systems, self-validated arithmetics are utilized for over-approximations of reachability computations.**

***Keywords*-Robustness; Simulation; Hybrid systems; Model Validation and Analysis; Floating-point arithmetic;**

I. Introduction

Model-based design has vastly simplified the development of embedded systems. Design tools such as Matlab/Simulink, MapleSim and Scade have improved the process of system design. Tools for automatically generating source code from these designs have vastly shorten the *time to market*. However, verification of these models remains just as complex and costly. Currently, extensive testing under different operating conditions is performed until the design team is satisfied that their design adheres to some coverage requirements.

In practice, however, testing does not explore all critical inputs. In addition, differentiating between an *interesting* and a *non-interesting* test-case is not straightforward. This paper considers the problem of evaluating tests in order to showcase failures of the *robustness property*. That is, we determine test trajectories that under some small perturbation – usually due to uncertainty or internal computation errors – could cause the system to diverge significantly from the expected behavior, and potentially lead to failures. Note that computational errors due to floating/fixed-points are frequently ignored by many verification techniques for real-time and hybrid systems. Furthermore, our approach ensures that the test cases that exhibit non-robust behavior will be of interest to the system designer.

The Patriot missile defense system failure [5] is a well-known example of the type of bugs that we target in this paper. Therein, the software, which estimated the future position of the incoming missile based on the velocity and the last position as returned by the radar, modeled time in increments of $0.1\,\mathrm{sec}$. However, 0.1 itself could not be accurately represented in the underlying 24-bit fixed-point register. As a result, the accumulated errors caused a drift in the computed times when the system was operational for many hours. Ultimately, this led to a failure to track and intercept incoming missiles.

Another – more subtle – example, is provided by Loh et al. [23] and earlier by Rump [29]. Consider the function $f(a,b) = (333.75-a^2)b^6+a^2(11a^2b^2-121b^4-2)+5.5b^8+\frac{a}{2b}$ with $a = 77617$ and $b = 33096$. Using round-to-nearest IEEE-754 arithmetic [31], the f function evaluates to (under different precision)

1.172604	(32-bit)
1.1726039400531786	(64-bit)
$1.172603940053178631858834904520183 8$	(128-bit)

By increasing the precision, we seem to derive more accurate results. However, this is deceiving since not even the sign is correct in the above computations. The actual result, within one unit of the last digit, is

$$f(a,b) = -0.827396059946821368141165095479816$$

These two examples demonstrate that our simulations could very well have undetected functional errors without the existence of any correctness errors (bugs) in the model.

For such computation errors and, also, for model uncertainties, our contribution in this paper is a framework for detecting the robustness of Simulink simulations. In detail, we developed a suit of tools called RobSim in Matlab for testing the robustness of discrete and continuous-time Simulink models. The advantage of our approach is that we do not have to translate the Simulink model into some other equivalent formalism, but we operate directly on the Simulink model itself. Thus, we remain faithful to Simulink semantics. Towards this objective, we have also produced the following results and implementations. First, we have built AALab (Affine Arithmetic Laboratory). AALab is an Affine Arithmetic (AA) [10] toolbox in Matlab that allows AA object manipulation with floating-point rounding control in a transparent way to the user. Second, we have refined the theory of linear system reachability in order to guarantee the correctness of Simulink numerical simulations under model uncertainties and, most importantly, under floating-point rounding uncertainties. The next section provides further details on the exact problem that we address in this paper.

II. Problem Definition

Our focus is on the investigation of the robustness of Simulink model simulations with respect to numerical and floating-point rounding errors under possible parametric uncertainties. Before proceeding on giving details on the problem itself and the proposed solutions, we need to define what we mean by robustness. The notion of robustness can have different meanings under different contexts.

Intuitively, in the context of system simulation and testing, a robust trajectory would be one that under small perturbations would exhibit the same qualitative behavior [17]. In certain cases, such questions can be answered by control theory even under the presence of floating-point rounding or quantization errors [27]. However in general, this problem cannot be addressed analytically in systems where the low-level system dynamics are governed by complex high-level logic such as in hybrid automata [17].

Thus, we approach the problem from a testing perspective. That is, given a model of a system and some initial conditions and parameter values, we would like to detect points in time where the simulation trajectory of the system does not capture the intended behavior. In particular, we first study the following problem.

Problem 1: Given a discrete-time Simulink model $\mathcal{D}$ with uncertain initial conditions X_0 and parameters P, determine points in time when the Simulink simulation trajectory is not robust.

In this problem, the Simulink model $\mathcal{D}$ already contains the initial conditions x_0 and parameter values p as well as the initial t_0 and final t_n simulation times and sampling step. Formally speaking, system $\mathcal{D}$ can be captured by a system of difference equations f_j^d, $j \in J_d$ such that

$$x(k+1) = f_j^d(x(k), u(k), p, k) \tag{1}$$

where $k \in \mathbb{N}$ is the current point in time, $x : \mathbb{N} \to \mathbb{R}^n$ is the solution of system (1) (we assume that a unique solution exists), $u : \mathbb{N} \to \mathbb{R}^q$ is an input to the system and $p \in P \subseteq \mathbb{R}^r$ is a vector modeling the parameters of the system. Here, $\mathbb{N}$ denotes the set of the natural numbers and $\mathbb{R}$ the set of reals. In system (1), at each point in time k, the system dynamics are governed by a unique difference equation f_j^d, i.e., the system is deterministic. Without going into details, the system dynamics f_j^d are chosen according to a relation G_j^d on $x(k)$, $x(k+1)$ and $u(k)$. Each solution $x(\cdot)$ of system (1) has a corresponding trajectory of indices $d_x : \mathbb{N} \to J_d$ indicating which system dynamics were active at the current point in time.

Assume that the set of initial conditions as well as the parameters and the inputs to the system are not precisely known, that is, $\mathbf{x}_0 \subseteq \mathbb{R}^n$, $\mathbf{p} \subseteq P$ and $\mathbf{u} \subseteq \mathbb{R}^q$ respectively. Then, theoretically for some $k_0 \leq k_f \in \mathbb{N}$ and a fixed $j \in J_d$, we could compute a set of trajectories $\mathbf{x} : [k_0, k_f] \to \mathcal{P}(\mathbb{R}^n)$, which represent the evolution of the system under the system dynamics f_j^d for all the possible parameter values and, moreover, take into account computation errors.

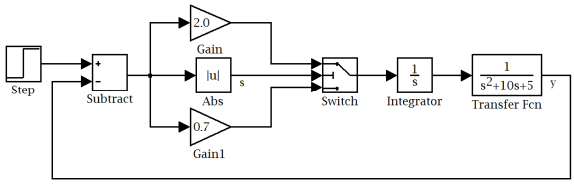

Figure 1. A continuous-time Simulink model.

Now, we can formalize our proposed notion of robustness.

Definition 1: A trajectory x of a Simulink model is robust if all the trajectories x' under internal (model parameters) and external (input) uncertainties and computation errors have the same trajectory of indices as x, that is, $d_x = d_{x'}$.

Informally, the above definition states that the system should exhibit the same high-level behavior under all given conditions and computation errors. Our goal is to ascertain points in the system trajectory which are not robust. The following example, which is the running example of this paper, presents the basic idea behind simulation non-robustness.

Example 1: As an example consider the discretized system of the continuous-time model in Fig. 1. The model contains a switch block which determines which gain is going to be used in the feedback loop. If the switch signal (signal s in figure) is larger than 0.4, then "Gain" is selected otherwise "Gain1" is selected. In this simple case, we want to detect situations where the numerical simulation computes a signal with value greater or equal to 4, e.g., for some k, $s(k) = 4.2$, but the guaranteed simulation gives bounds on s which include 4, e.g., $[s(k)] = [3.9, 4.5]$. This means that it could be the case that at that point in time the system could exhibit different dynamical behavior from the one used in the numerical simulation. Such cases are reported to the user.

The solution of Problem 1 is presented in Section IV-A. In this paper, we also present some results toward the solution of the above problem for continuous-time Simulink models.

Problem 2: Given a continuous-time Simulink model $\mathcal{C}$ with piece-wise continuous dynamics and a set of uncertain initial conditions X_0 and parameters P, determine points in time when the Simulink simulation trajectory is not robust.

Similar to the discrete-time case, system $\mathcal{C}$ can be captured by a system of linear ordinary differential equations (ODEs):

$$\dot{x}(t) = A_j(p(t))x(t) + B_j(p(t))u(t) + h_j(p(t)) \tag{2}$$

where $j \in J_c$ is a set of indices, $t \in \mathbb{R}^+$ (non-negative reals) is the current point in time, $x : \mathbb{R}^+ \to \mathbb{R}^n$ is the solution of system (2) (again, we assume that a unique solution exists), $u : \mathbb{R}^+ \to \mathbb{R}^q$ is an input to the system, $p : \mathbb{R}^+ \to P \subseteq \mathbb{R}^r$ is a continuous function modeling the parameters of the system and A_j, B_j, h_j are appropriately

sized matrices (which are continuous with respect to the parameter vector p). At each point in time t, the system dynamics are governed by a unique differential equation $\dot{x} = A_j(p(t))x(t) + B_j(p(t))u(t) + h_j(p(t))$. As before, the system dynamics are chosen according to whether a relation G_j^c on $x(t)$, $\dot{x}(t)$ and $u(t)$ is true. Each solution $x(\cdot)$ of system (2) has a corresponding trajectory of indices $d_x : \mathbb{R}^+ \rightarrow J_c$ indicating which system dynamics were active at each point in time.

Analogous to the discrete-time case, we might want to study the behavior of the continuous-time model under uncertain initial conditions $\mathbf{x}_0 \subseteq \mathbb{R}^n$, parameters $\mathbf{p}(t) \subseteq P$ and/or inputs $\mathbf{u}(t) \subseteq \mathbb{R}^q$. Then, theoretically for some $t_0 \leq t_f$ and a fixed $j \in J_c$, we could compute a set of trajectories $\mathbf{x} : [t_0, t_f] \rightarrow \mathcal{P}(\mathbb{R}^n)$, which represent the evolution of the system under the system dynamics (2) for all the possible parameter values and, moreover, take into account computation errors.

Definition 1 also holds for continuous-time systems. However, one additional issue that appears in the case of continuous-time systems is the correctness of the computed simulation trajectory. In detail, the algorithm employed for the numerical solution of the ODEs affects the quality of the resulting simulation and, in some cases, the solution might even diverge from the real solution. Therefore, in the continuous-time case, we also report such cases. The details of the proposed approach appear in Section IV-B.

The solutions to Problems 1 and 2 enable also the robust simulation of mixed-signal Simulink models. Such models contain both continuous and discrete-time components. In Section IV-C, we present some preliminary results toward the solution of the following problem.

Problem 3: Given a mixed-signal Simulink model $\mathcal{M}$ with discrete and continuous-time components that satisfy the assumptions of Problems 1 and 2, respectively, and a set of uncertain initial conditions X_0 and parameters P, determine points in time when the Simulink simulation trajectory is not robust.

As a by-product of the solution to the above problem, we obtain a reachability algorithm for mixed-signal systems under input, parameter and computation uncertainties. To the best of our knowledge, this is the first time that this problem is addressed.

III. Theoretical Background

In this section, we review some basic results and notation that are necessary for presenting the results in this paper.

A. Interval Arithmetic

Interval arithmetic (IA) [25] is one of the first computation models that enabled in an efficient way self-validating computations. IA has proven to be a valuable tool in a variety of engineering applications [16]. More related to our work is the application of IA to reachability problems [14], [28].

In IA every quantity $[x]$ is essentially an interval $[x] = [\underline{x}, \overline{x}]$, where the bounds $\underline{x} = \inf[x]$ and $\overline{x} = \sup[x]$ range over a particular field and $\underline{x} \leq \overline{x}$. In this paper, we will consider the fields of the real numbers $\mathbb{R}$ and an "abstract" field of floating point numbers $\mathbb{F}$. The corresponding sets of intervals will be denoted by $\mathbb{IR}$ and $\mathbb{IF}$, respectively. The purpose of each such interval is to model potential uncertainties in the quantities and to capture internal computation errors such as rounding errors.

All the standard arithmetic operations have a corresponding operation in IA. For example, if $x, y \in \mathbb{IR}$, then we have $[\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$ and $[\underline{x}, \overline{x}][\underline{y}, \overline{y}] = [\min(\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}), \max(\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\})]$. Whenever we perform an operation of an interval with a real number a, we implicitly assume that the real number is a degenerate interval, i.e., $[a] = [a, a]$.

In this paper, we will also be using interval vectors and interval matrices. For example, an n-ordered tuple of intervals $[[\underline{x}_1, \overline{x}_1], \ldots, [\underline{x}_n, \overline{x}_n]]^T$ is an interval vector $[\mathbf{x}]$. In the following, we will also be using the notation $\square x$ to denote the interval vector $[-x, x]^n$ for some $n \geq 1$. Similarly, we define $m \times n$ interval matrices as members of $\mathbb{IR}^{m \times n}$ or $\mathbb{IF}^{m \times n}$. The IA operations on vectors and matrices are the natural extensions over the real arithmetic operations.

The concept of norm plays an important role in the analysis of properties of vectors and matrices. In order to define the infinity norm for interval matrices, we need to first define the absolute value of an interval number: $|[x]| := \max(\{|\underline{x}|, |\overline{x}|\})$. Note that the properties of the absolute value are preserved, i.e., for $x, y \in \mathbb{IR}$, we have $|[x] + [y]| \leq |[x]| + |[y]|$ and $|[x][y]| = |[x]||[y]|$. Recall that in the case of matrices, the induced norm for an $m \times n$ matrix A is defined as $\|A\|_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^{n} |a_{ij}|$. The extension of the definition of the infinity norm to interval matrices is simply $\|[\mathbf{A}]\|_\infty = \|\max(\{|\underline{A}|, |\overline{A}|\})\|_\infty$, where the absolute value and the maximum are considered element-wise. Finally, we should remark that both the absolute value and the norm are not interval numbers.

In this work, we are particularly interested in using IA in order to capture floating-point rounding errors. For example, consider the number 0.1 which is not exactly representable in machine arithmetic. IA allows us to capture this number by defining an interval $[\downarrow(0.1), \uparrow(0.1)]$, where $\downarrow(\cdot)$ and $\uparrow(\cdot)$ are the rounding-down and up operations respectively as defined in IEEE Floating-Point Standard [31]. In the following, we will also be using the notation $\updownarrow(x) = [\downarrow(x), \uparrow(x)]$ in order to bound the real value of a number. In a slight abuse of notation, we will also use $\updownarrow(\texttt{exp})$ to imply that the operations in the expression $\texttt{exp}$ are performed using IA even though the operands are floating-point numbers. Moreover, when we perform IA operations over floating-point numbers, we have to control the rounding mode accordingly in order to enclose the real value of the operation. For example, if $x, y \in \mathbb{IF}$, then we have $[\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] = [\downarrow(\underline{x} + \underline{y}), \uparrow(\overline{x} + \overline{y})]$. For IA

computations, we use the IntLab library [30] in Matlab.

IA exhibits some of the usual properties, i.e., it is associative and commutative. Moreover, it is *inclusion monotone*. Namely, if $[x'] \subseteq [x]$ and $[y'] \subseteq [y]$, then $[x'] \circ [y'] \subseteq [x] \circ [y]$ where $\circ \in \{+, -, \times, \div\}$. However, IA has some important deficiencies. In detail, it is *sub-distributive*, i.e., $[x]\,([y] + [z]) \subseteq [x]\,[y] + [x]\,[z]$ and, also, it has *no additive* and *no multiplicative inverse*. In general, in IA, there is no information maintained concerning the relationship of two symbolic values. Consider for example the interval $[x] = [-0.1, 0.1]$, then $[x] - [x] = [-0.2, 0.2]$. Obviously, this is too coarse an overapproximation. The tightest possible bound is the interval $[0, 0]$. In feedback systems, such overapproximations will result in a quick divergence of the solution enclosure. Fortunately, symbolic computations using affine arithmetic can resolve this issue.

B. Affine Arithmetic

Affine Arithmetic (AA) [10] is also a self-validated computation model that permits reasoning over ranges. AA maintains linear dependencies between AA quantities in order to reduce some of the overapproximation issues that are present in IA. Linear operations on AA quantities preserve such linear dependencies. However, when nonlinear operations are encountered, for example multiplication, then some small overapproximation of the actual range is introduced.

In detail, AA quantities represent ranges over some field. In the following, we will be using angular brackets to denote symbolic variables that represent ranges in AA. Each AA variable $\langle x \rangle$ is the summation of a *central value* $x_0 \in \mathbb{R}$ and a number of *affine terms* $x_i\varepsilon_i$, where $x_i \in \mathbb{R}$ is a coefficient and ε_i is a variable that ranges over $[-1, 1]$. Formally, we write $\langle x \rangle = x_0 + \sum_{i=1}^{k} x_i\varepsilon_i$. The interpretation of $\langle x \rangle$ is that for any $x \in \langle x \rangle$, for all $i = 1, \ldots, k$, there exist $\tilde{\varepsilon}_i \in [-1, 1]$ such that $x = x_0 + \sum_{i=1}^{k} x_i\tilde{\varepsilon}_i$. Finally, the interval represented by $\langle x \rangle$ is simply $[\langle x \rangle] := [x_0 - \sum_{i=1}^{k} |x_i|, x_0 + \sum_{i=1}^{k} |x_i|]$ (we should appropriately control the rounding mode in the case of floating-point numbers). For convenience, we will denote the set of all AA quantities over the reals by $\mathbb{AR}$ and over the floating-point numbers by $\mathbb{AF}$.

As mentioned earlier, linear operations, such as addition, are straightforward to define, e.g., for $x, y \in \mathbb{AR}$, we have $\left(x_0 + \sum_{i=1}^{k} x_i\varepsilon_i\right) + \left(y_0 + \sum_{i=1}^{k} y_i\varepsilon_i\right) = (x_0 + y_0) + \sum_{i=1}^{k}(x_i + y_i)\varepsilon_i$. On the other hand, nonlinear operations, such as multiplication, must be defined in such a way that they maintain as many linear terms as possible and, in addition, overapproximate the nonlinear terms by generating a new affine term. For example, for $x, y \in \mathbb{AR}$, we have $\langle x \rangle \langle y \rangle = x_0y_0 + \left(\sum_{i=1}^{k}(x_0y_i) + \sum_{i=1}^{k}(x_iy_0)\right)\varepsilon_i + z\varepsilon_{k+1}$, where z is such that $|z| \geq \left|\left(\sum_{i=1}^{k} x_i\varepsilon_i\right)\left(\sum_{i=1}^{k} y_i\varepsilon_i\right)\right|$. One of the interesting problems in AA is how to compute good bounds on the nonlinear terms [9]. Finally, we should mention that whenever we mix interval and affine quantities in an expression we implicitly assume that the interval variable is converted into an affine variable (see [9]).

Similar to IA, we can define affine vectors and matrices as elements of $\mathbb{AR}^n$ ($\mathbb{AF}^n$) and $\mathbb{AR}^{m\times n}$ ($\mathbb{AF}^{m\times n}$), respectively. Furthermore, the infinity norm of an affine matrix is simply the infinity norm of the corresponding interval matrix, i.e., $\| \langle \mathbf{A} \rangle \|_\infty := \|[\langle \mathbf{A} \rangle]\|_\infty$.

Since one of our goals is to account for computation errors, we must take into account floating-point rounding errors. As opposed to IA, considering floating-point rounding errors in AA is a little bit more involved. We could just round outward each of the coefficients in a new affine quantity, but then the dependencies of the affine quantity on the rounding errors would be lost. In order to maintain as much information as possible, we can instead compute the rounding errors for each affine term separately and, then, we can accumulate the errors into a new affine term. For example, for $x, y \in \mathbb{AR}$, we have $\langle x \rangle + \langle y \rangle = (x_0 + y_0) + \sum_{i=1}^{k}(x_i + y_i)\varepsilon_i + z\varepsilon_{k+1}$, where ε_{k+1} is a new affine term and z captures all the floating-point rounding errors.

In other words, we can associate an affine term with the computation error that occurs at each operation. These errors and, also, the affine terms which model uncertain parameters in the system can be tracked along a computation sequence. Therefore, at each point in time, we can be cognizant on how each uncertainty affects the current state of the system. But more importantly, the propagation of the affine terms helps to avoid the problems of IA due to lost dependencies between the variables. Note, however, that it is not always the case that AA computes better bounds than IA.

Since there is no publicly available affine arithmetic toolbox for Matlab, we developed our own package AALab (Affine Arithmetic Laboratory). AALab provides a class for AA quantities and overloads most of the standard functions and operators in Matlab in order to provide AA in a transparent way to the user. In addition, AALab provides the option to take into account or ignore floating-point rounding errors. It also offers various levels of approximations for nonlinear functions (e.g., trigonometric functions) and, finally, it implements several ways of aggregating affine terms when their number becomes too large. We should remark that even though [32] also studies an implementation of AA in Matlab, that work does not support the aforementioned features.

IV. Detecting Non-Robustness in Simulations

In the previous sections, we provided a short overview on how self-validated computations can be realized within Matlab. In this section, we use these tools in order to present a framework for detecting non-robustness in discrete-time, continuous-time and mixed-signal Simulink models.

A. Discrete-Time Simulink Models

The main challenge in trying to perform self-validated computations within the Simulink environment is that

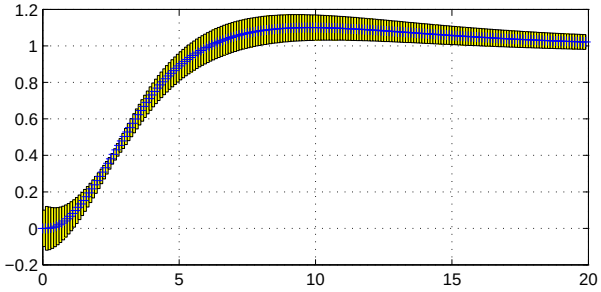

Figure 2. Guaranteed simulation of Example 2. The bars indicate the bounds on the guaranteed simulation for each time step of $0.1\,\text{sec}$, while the line inside is the result of the Simulink simulation.

Simulink accepts only its built-in data types, e.g., double, signal, int8, fixed-point etc. One way to overcome this restriction is by monitoring Simulink during a simulation. Namely, we first instrument Simulink with callback functions (listeners) using Simulink's Application Programming Interface (API). Then, we record the sequence of accesses and the corresponding events (such as "update" or "output") sent to the blocks during one Simulation step. Also, we maintain information about the input signals to the blocks. In the following, we refer to such sequences as *block execution traces*. The advantage of inserting callbacks is that the execution order semantics of Simulink is enforced. Finally, we follow the block execution trace, but now each mathematical operation and all the relation checks are performed using self-validating computation theories such as IA or AA.

We have implemented the above procedure into the software toolbox RobSimDt (ROBust SIMulations in Discrete-Time) within the Matlab programming environment. RobSimDt takes as input a discrete-time Simulink model $\mathcal{D}$ and can perform guaranteed discrete-time simulation using the parameters provided in $\mathcal{D}$. Since AA provides in general better bounds on the validated solution than IA, the AA computation theory is chosen by default. If the user wants to use uncertain initial conditions, parameters and/or inputs, then she or he can do so by providing them to RobSimDt. In this case, the self-validated computation model employed is the one that corresponds to the data-type of the initial conditions. The output of RobSimDt is the guaranteed simulation trace and a structure which holds information on which Simulink blocks, time and input signal values the computation might be non-robust. The following result is immediate from the properties of self-validated arithmetics.

Theorem 1: Let $\mathcal{D}$ be a discrete-time Simulink model as in Problem 1. Let $\tilde{x}(k)$ be the state vector of the Simulink simulation and $\langle \mathbf{x} \rangle (k)$ be the state vector enclosure returned by RobSimDt at sample k. Then, for all samples k of the Simulink simulation, we have $\tilde{x}(k) \in \langle \mathbf{x} \rangle (k)$.

Example 2: Let us consider the continuous-time model of Fig. 1. The model is discretized with the Simulink Model Discretizer using zero-order hold (zoh) and a sample step of $0.1\,\text{sec}$. We would like to determine the robustness of the simulation with initial conditions $[-0.1, 0.1]^3$ and variation of $\%1$ in the parameter values. The Simulink simulation is performed with initial conditions $[0\ 0\ 0]^T$ and the parameter values as appear in Fig. 1. The result of the computation for the signal y appears in Fig. 2. RobSimDt detected 5 points where the simulation is non-robust. Below, we present how RobSimDt displays one of the warnings of possible non-robustness (recall that the threshold is 0.4):

```
Warning 5: In the block 'Switch' at
time 3.6, the value of the input signal
to the block was 0.3537 while the range
of the input signal computed by RobSim
is [ 0.2948, 0.4126].
```

Next, we demonstrate how self-validated computations and robustness checks are performed. Due to space limitations, we are only going to discuss briefly the procedures of two blocks of our running example: the *discrete-time integrator* and the *switch*. Also, the following presentation focuses on the use of affine arithmetic as the underlying computation model which is the default option in RobSimDt. However, the application of other self-validating computation models is immediate.

Discrete-Time Integrator. In the current version, RobSimDt supports only the forward Euler integration (or accumulation) method (see the on-line Simulink help documentation). When a discrete-time integrator is accessed through an "update" event, it returns the value of the expression $\langle x \rangle (k) + \langle K \rangle \updownarrow (dt) \langle u \rangle (k)$ which is evaluated using self-validated arithmetics. Here, $\langle x \rangle (k)$ is the (guaranteed) state of the integrator at time k, $\langle u \rangle (k)$ is the (guaranteed) input to the block at time k, K is the gain parameter of the block and dt is the sample step size. Note that we enclose the real value of dt using interval arithmetic. This is important since the exact value of dt might not be representable in machine arithmetic and, thus, if floating-point rounding errors are not accounted for, they could cause major inaccuracies in the subsequent computations (recall the case of the Patriot missile incident in the introduction). The same holds for the gain K. However, we also allow for the case that the value of the gain K is uncertain and, moreover, it might be dependent on other system parameters. Such dependencies can be modeled by letting $\langle K \rangle$ share affine terms with other system parameters. Finally, in the case where an "output" event is received, the discrete-time integrator simply returns the current state $\langle x \rangle (k)$.

Switch Block. The switch block can only be accessed through an "output" event. Let us assume that the threshold option in the block has been set to the option "u2 >= Threshold". This option means that the input signal u2 should be greater or equal to the threshold and that if this is true, then the upper input signal "u1" is allowed to pass through, otherwise the lower input signal "u3" is allowed to pass through. Now, if the formula $\phi_s(k) = (\langle \mathtt{u2} \rangle (k) \geq$

$\texttt{Threshold}) \vee (\langle \texttt{u2} \rangle (k) < \texttt{Threshold})$ evaluates to false (note that this is not a tautology since the comparisons are in IA or AA), then a robustness violation has occurred. This is a violation since either input signal $\langle \texttt{u1} \rangle (k)$ or $\langle \texttt{u3}(k) \rangle$ could have been chosen by the switch as output. Thus, in this case, we record the time, block and signal values. Moreover, since we are trying to study the robustness of a particular Simulink simulation, we let the switch chose the output signal according to the value of the non-guaranteed input signal $\texttt{u2}(k)$. On the other hand, if either subformula of ϕ_s is true at time k, then the corresponding input signal is allowed to pass through.

B. Continuous-Time Simulink Models

Problem 2 is substantially more challenging then Problem 1. The main obstacle is that replacing the floating-point arithmetic with self-validating methods is not enough. Namely, the model of the system does not consist any more of ordinary difference equations, but of ordinary differential equations which are solved using a numerical integration algorithm. Numerical integration algorithms can only solve the system of equations approximately and, moreover, they are also themselves bound to the problem of floating-point round errors. One way to address this problem is by resorting to a guaranteed integration method [24], [6]. However, these integration methods require in turn the explicit representation of the underlying differential equations of the system.

In general, such a system of mathematical equations cannot be derived automatically due to the complex nature of the Simulink models, e.g., switch and saturation blocks, nonlinearities etc. Moreover, linearization methods do not always extract an exact representation of the linear system even if such an exact representation exists. In order to circumvent the deficiencies of any linearization method, we symbolically compute the derivatives of a Simulink model whose dynamics are affine for a given operating point and time. For that purpose, we use our toolbox SL-Trace, which was initially developed for the symbolic execution and systematic testing of Simulink models.

Another problem that prevents us from directly employing guaranteed integrators for the solution of Problem 2 is the fact that these methods only apply to state vectors represented in interval arithmetic. However, in feedback systems it is paramount to use state vectors with symbolic numbers in affine arithmetic. Thus, in general, the dependencies between the different variables during computation are preserved and, in turn, we reduce overapproximations through cancellations. We addressed the problem of integration by developing RobSimCt.

RobSimCt (Robustness of Simulations in Continuous-time) is a toolbox built in Matlab programming environment which addresses Problem 2. Similar to RobSimDt, it instruments the input Simulink model with listeners in order to record the sequence of accesses to Simulink blocks in the model. Then, it executes the Simulink simulation with the default parameters of the model in order to produce a block execution trace. Note that for the simulation itself, Simulink can use any built-in numerical integration method. This is important since we don't have to restrict our analysis only to fixed-step algorithms.

For every time step $\Delta t_i = t_{i+1} - t_i$ in the simulation, we pass the corresponding part of the block execution trace to SL-Trace. In turn, SL-Trace returns the symbolic derivatives of the linear (or affine) system for that particular time step and, also, the constraints on the state space of the model that must be satisfied (see [18] for a related approach on deriving symbolic traces). Using the symbolic derivatives, we can compute an enclosure of the state vector for all time between the two sampling points. Then, the enclosure is compared with the constraints to detect possible non-robustness points in the simulation.

Now, let us assume that the system dynamics as returned by SL-Trace are given by the interval matrices $[\mathbf{A}]$, $[\mathbf{B}]$ and $[\mathbf{h}]$. The interpretation of these matrices is that there exists a parameter function $p : [t_i, t_{i+1}] \to P$ such that for all $t \in [t_i, t_{i+1}]$, we have $A^*(t) = A(p(t)) \in [\mathbf{A}]$, $B^*(t) = B(p(t)) \in [\mathbf{B}]$ and $h^*(t) = h(p(t)) \in [\mathbf{h}]$. Then, for any $u : [t_i, t_{i+1}] \to U$, the behavior of the system for the time interval $[t_i, t_{i+1}]$ is defined by the solution of the equation

$$\dot{x}(t) = A^*(t)x(t) + B^*(t)u(t) + h^*(t) \tag{3}$$

Recall that our goal in this section is twofold. First, we need to verify that the Simulink simulation at time t_{i+1} starting from time t_i is not false. Second, we have to check that between time t_i and time t_{i+1} there was no potential violation of the current model invariants. The former requires a method that computes an inclusion of the state of system (3) at any time t for all possible system parameters and inputs.

Correctness of Integration. Let us first consider the case where there are no external inputs to the system, i.e., the state-space representation of the system reduces to $\dot{x}(t) = A^*(t)x(t)$. Then, the solution is $x(t) = \Phi(t, t_i)x(t_i)$, where

$$\Phi(t, t_i) = I + \int_{t_i}^{t} A^*(s_1)ds_1 + \int_{t_i}^{t} A^*(s_1) \int_{t_i}^{s_1} A^*(s_2)ds_2 ds_1 + \dots$$

is the Peano-Baker series for the transition matrix. Here, I is the identity matrix. Since each entry in A^* is continuous with respect to t and the interval $[t_i, t]$ is compact, by repeated applications of the mean value theorem for integrals, there exist matrices $A_1^*, A_{21}^*, A_{22}^*, \dots \in [\mathbf{A}]$ such that

$$\Phi(t, t_i) = I + A_1^*(t - t_i) + A_{21}^* A_{22}^* \frac{(t - t_i)^2}{2} + \dots$$

Note that the mean value theorem cannot be applied to A^*, but to each of its entries. Hence, by the properties of IA, we derive the inclusion

$$\Phi(t, t_i) \in I + (t - t_i)\,[\mathbf{A}] + \frac{(t - t_i)^2}{2}\,[\mathbf{A}]^2 + \dots. \tag{4}$$

Moreover, in Section IV in [26], it was proven that the above series converges to the exponential of the interval matrix:

$$e^{[\mathbf{A}](t-t_i)} \triangleq I + (t-t_i)\,[\mathbf{A}] + \frac{(t-t_i)^2}{2}\,[\mathbf{A}]^2 + \dots. \quad (5)$$

Thus, we have

$$\Phi(t,t_i) \in e^{[\mathbf{A}](t-t_i)}. \quad (6)$$

Of course, we are now faced with the computation of the exponential of an interval matrix. Unfortunately, the exponential of an interval matrix cannot be computed exactly. In order to overapproximate the range of the exponential of a matrix, we use the Taylor expansion in a way similar to [1] and [26] such that it also takes into account floating-point rounding errors. Let $A \in \mathbb{IR}^{n\times n}$, then the exponential can be obtained by the Taylor series $e^{[\mathbf{A}]} = I + [\mathbf{A}] + \frac{1}{2}[\mathbf{A}]^2 + \frac{1}{3}[\mathbf{A}] + \dots$. In order to compute a bound on the solution of $e^{[\mathbf{A}]}$, we must first compute a number m of Taylor terms and then conservatively bound the remainder term. In this work, we bound the reminder term using the relation provided in [21] (which was also employed in [1] and [26]). The next theorem, which is immediate from the properties of IA, provides an overapproximation of the exponential of the interval matrix.

Theorem 2: Given $A \in \mathbb{IR}^{n\times n}$, an overapproximation of the matrix exponential $e^{[\mathbf{A}]}$ of order m can be obtained by:

$$\left\lceil e^{[\mathbf{A}]} \right\rceil = I + \sum_{k=1}^{m} \updownarrow\!\left(\frac{1}{k!}\right)[\mathbf{A}]^k + \square \sup \updownarrow(E([\mathbf{A}])) \quad (7)$$

where $E([\mathbf{A}]) = \frac{\|[\mathbf{A}]\|_\infty^{m+1}}{(m+1)!}\frac{1}{1-\epsilon}$ and $\epsilon = \frac{\|[\mathbf{A}]\|_\infty}{m+2} < 1$.

Remark 1: The condition that $\epsilon < 1$ is an important one. If it is not satisfied, then we should increase the order of the Taylor terms m. Since we are actually concerned with the product of the system dynamics $[\mathbf{A}]$ with the integration step Δt_i, we can alternatively decrease the size of Δt_i.

Thus, a safe inclusion of the unforced propagation of the state under rounding errors can be computed as:

$$\langle \mathbf{x} \rangle (t) = \left\lceil e^{[\mathbf{A}]\updownarrow(t-t_i)} \right\rceil \langle \mathbf{x} \rangle (t_i) \quad (8)$$

If the state vector $\tilde{x}$ computed by the numerical integration algorithm at time t_{i+1} is not contained in $\langle \mathbf{x} \rangle (t_{i+1})$, i.e., $\tilde{x}(t_{i+1}) \notin \langle \mathbf{x} \rangle (t_{i+1})$, then we know that an integration error occurred. On the other hand, if $\tilde{x}(t_{i+1}) \in \langle \mathbf{x} \rangle (t_{i+1})$, then we cannot be sure whether the integration is correct or not.

Remark 2: Both numbers t_i and t_{i+1} are machine representable floating-point numbers since they are computed during the Simulink simulation. Nevertheless, their difference $\Delta t_i = t_{i+1} - t_i$ might not be representable in machine arithmetic, hence in (8) we bound $t - t_i$ using IA.

Next, we have to take into account the influence of potential inputs or noise to the system, i.e., the case where $B^*, h^* \neq 0$:

$$x(t) = \Phi(t,t_i)x(t_i) + \int_{t_i}^{t} \Phi(t,s)(B^*(s)u(s) + h^*(s))ds$$

For any $s \in [t_i, t]$, we have $B^*(s) \in [\mathbf{B}]$, $h^*(s) \in [\mathbf{h}]$, $u(s) \in [\mathbf{u}]$ and, also, from (6) and (7) by ignoring rounding control, we have $\Phi(t,s) \in \left\lceil e^{[\mathbf{A}](t-s)} \right\rceil$. By a change of variables, $\sigma = t - s$, we get $d\sigma = -ds$ and $\int_{t_i}^{t} \Phi(t,s)(B^*(s)u(s) + h^*(s))ds \in \int_0^{t-t_i} \left\lceil e^{[\mathbf{A}]\sigma} \right\rceil d\sigma([\mathbf{B}]\,[\mathbf{u}] + [\mathbf{h}])$. From (7), we also get (again, no rounding control)

$$\int_0^{t-t_i} \left(I + \sum_{k=1}^{m} \frac{1}{k!}[\mathbf{A}]^k \sigma^k + \square E([\mathbf{A}]\,\sigma) \right) d\sigma =$$

$$= I(t-t_i) + \sum_{k=1}^{m} \frac{(t-t_i)^{k+1}}{(k+1)!}[\mathbf{A}]^k + \int_0^{t-t_i} \square E([\mathbf{A}]\,\sigma) d\sigma$$

As it was indicated in [1], $E([\mathbf{A}]\,\sigma)$ is monotone (increasing) with respect to σ. Thus, $\int_0^{t-t_i} \square E([\mathbf{A}]\,\sigma)d\sigma \subseteq \square E([\mathbf{A}]\,(t-t_i)) \int_0^{t-t_i} d\sigma = \square E([\mathbf{A}]\,(t-t_i))(t-t_i)$. Therefore, we can compute a conservative enclosure for the effect of the external input u and the affine term h: $[\mathbf{\Gamma}]_m(\Delta t_i) = I \updownarrow (\Delta t_i) + \sum_{k=1}^{m} \updownarrow\!\left(\frac{\Delta t_i^{k+1}}{(k+1)!}\right)[\mathbf{A}]^k + \square E([\mathbf{A}] \updownarrow(\Delta t_i)) \updownarrow(\Delta t_i)$.

Constraint Violation. Next, we need to address the problem of possible violation of the location invariant constraints. This step is straightforward. Along with the symbolic derivatives, SL-Trace also returns the invariant constraints that hold at each point in time in the simulation. More formally, for each time step $[t_i, t_{i+1}]$, we have a set of constraints $\Pi_i = \{\alpha_j x + \beta_j u + \gamma_j \leq 0\}_{j\in J}$, which is indexed by the block id where the relation check took place. All we need to do then is to check that for all $t \in [t_i, t_{i+1}]$ and for all $j \in J$, the relation $\alpha_j x(t) + \beta_j u(t) + \gamma_j \leq 0$ is satisfied. From the computation of $\langle \mathbf{x} \rangle (t_{i+1})$, it is obvious that we cannot perform these checks analytically. Therefore, once more we must conservatively bound the state of the system between times t_i and t_{i+1}, that is, we have to compute a set $\mathbf{R}$ such that $\langle \mathbf{x} \rangle (t) \subseteq \mathbf{R}$ for all $t \in [t_i, t_{i+1}]$.

Toward that goal, we consider an initial state $x(t_i) \in \langle \mathbf{x} \rangle (t_i)$ and the final value of the trajectory at time t_{i+1}, i.e., $x(t_{i+1}) = \Phi(t_{i+1}, t_i)x(t_i)$. Similar to [11], for any $t \in [t_i, t_{i+1}]$, we consider the approximation of $\Phi(t,t_i)x(t_i)$ by the linear approximation $x(t_i) + \frac{t-t_i}{t_{i+1}-t_i}(\Phi(t_{i+1},t_i)x(t_i) - x(t_i))$. Let as denote the difference between the actual value and its approximation by $\delta(t)$, that is,

$$\delta(t) = \Phi(t,t_i)x(t_i) - x(t_i) - \frac{t-t_i}{t_{i+1}-t_i}(\Phi(t_{i+1},t_i)x(t_i) - x(t_i))$$

Using relation (6), the fact that $x(t_i) \in \langle \mathbf{x} \rangle (t_i)$ and the properties of AA, we get

$$\delta(t) \in \left(e^{[\mathbf{A}](t-t_i)} - I - \frac{t-t_i}{t_{i+1}-t_i}(e^{[\mathbf{A}](t_{i+1}-t_i)} - I) \right) \langle \mathbf{x} \rangle (t_i)$$

$$= \sum_{k=2}^{\infty} \frac{(t-t_i)((t-t_i)^{k-1} - (t_{i+1}-t_i)^{k-1})}{k!}[\mathbf{A}]^k \langle \mathbf{x} \rangle (t_i)$$

Now, we are in position to adapt Theorem 3 from [1] in order to conservatively enclose $\delta(t)$ in the set $[\mathbf{\Delta}]_m(\Delta t_i)\langle \mathbf{x} \rangle (t_i)$,

where $[\mathbf{\Delta}]_m(\Delta t_i) = \sum_{k=2}^{m}[\lambda(k,\Delta t_i),0]\,[\mathbf{A}]^k + \square E([\mathbf{A}]\updownarrow(\Delta t_i))$ and $\lambda(k,\Delta t_i) = \inf\updownarrow\Big((k^{\frac{-k}{k-1}} - k^{\frac{-1}{k-1}})\frac{\Delta t_i^k}{k!}\Big)$.

Up to now, we saw how to compute a bound on the divergence of $x(t)$ from the line that connects $x(t_i)$ and $x(t_{i+1})$. In order to account for all such trajectories between $\langle\mathbf{x}\rangle(t_i)$ and $\langle\mathbf{x}\rangle(t_{i+1})$, we simply have to compute the convex hull of the sets $\langle\mathbf{x}\rangle(t_i)$ and $\langle\mathbf{x}\rangle(t_{i+1})$. Such a set cannot be computed, thus previous authors [11], [1], who have worked with zonotope representations of the state vector, have over-approximated the convex hull with a new zonotope (for details see [11]). In our work, we instead compute the interval convex hull ($\mathcal{CH}$) of the two symbolic sets. This gives us a quick, but conservative, enclosure of the convex hull. Thus, the resulting set enclosure for the state vector between times t_i and t_{i+1} is

$$[\mathbf{R}] = \mathcal{CH}([\langle\mathbf{x}\rangle(t_i)],[\langle\mathbf{x}\rangle(t_{i+1})]) + [\mathbf{\Delta}]_m(\Delta t_i)[\langle\mathbf{x}\rangle(t_i)], \tag{9}$$

where for some $x, y \in \mathbb{R}$, we have $\mathcal{CH}([\mathbf{x}],[\mathbf{y}]) = [\min(\underline{x},\underline{y}),\max(\overline{x},\overline{y})]$. Now that we have an over-approximation of the reachable set between samples t_i and t_{i+1}, we can perform the check for guard violation by simply checking whether $\alpha_j[\mathbf{R}] + \beta_j[\mathbf{u}] + \gamma_j \leq 0$ for all $j \in J$. As mentioned before, if any of these checks fails, then we keep the corresponding information.

Summary. The above discussion can be summarized in the following theorem where internal computation errors are also taken into account.

Theorem 3: Consider the time interval $T_i = [t_i, t_{i+1}]$ and a continuous parameter function $p : T_i \to P$ such that for all $t \in T_i$ the matrices $A^*(t) = A(p(t)) \in [\mathbf{A}]$ and $B^*(t) = B(p(t)) \in [\mathbf{B}]$ and the vector $h^*(t) = h(p(t)) \in [\mathbf{h}]$ are continuous in each of their entries with respect to t. Moreover, let $\Delta t_i = t_{i+1} - t_i$ and m be the order of the Taylor terms. Then, for any continuous input function $u : T_i \to U$ such that for all $t \in T_i$ we have $u(t) \in [\mathbf{u}]$, the system (3) with initial conditions in $\langle\mathbf{x}\rangle(t_i)$ has a solution at time t_{i+1} which is guaranteed to lie in affine set $\langle\mathbf{x}\rangle(t_{i+1}) = \left[e^{[\mathbf{A}]\updownarrow(\Delta t_i)}\right]\langle\mathbf{x}\rangle(t_i) + [\mathbf{\Gamma}]_m(\Delta t_i)$ under floating-point arithmetic. Moreover, for any $t \in T_i$, from (9), we have $\langle\mathbf{x}\rangle(t) \in [\mathbf{R}]$ under floating-point arithmetic.

Example 3: Consider the model of Fig. 1 with parameter values as in Example 2. For the numerical integration, we use the stiff ODE 15 solver with variable step size with maximum step size $0.05\,\text{sec}$. The constraint on the step size is necessary in order to compute a better approximation on the reachable set. The order for the Taylor approximation used was 40. RobSimCt reported 317 possible constraint violation warnings and 0 integration errors. Out of the warnings, 17 were at the Switch block. The warnings generated at the Abs block can be safely ignored since the output of the Abs block in all these cases is always less than 0.4. Thus, it does not affect the behavior of the Switch.

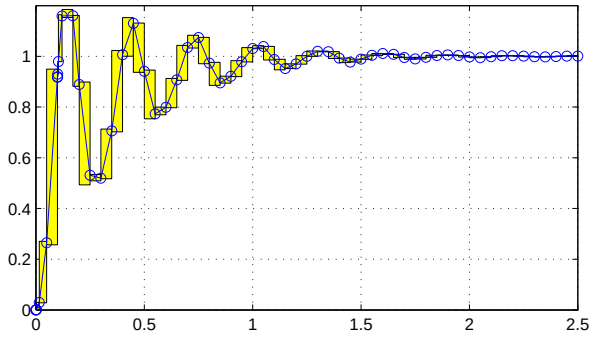

Figure 3. Guaranteed simulation of Example 4. The bars indicate the bounds on the guaranteed simulation at each time step, while the line with the circles is the result of the Simulink simulation.

C. Mixed-Signal Simulink Models

In this section, we present our recent progresses in solving Problem 3. That is, we briefly review the integration of RobSimDt and RobSimCt into a single toolbox, RobSim, which can test the robustness of mixed-signal models.

Currently, RobSim accepts models which have continuous and discrete-time components separated by Zero-Order Hold blocks. This is necessary both theoretically and implementation-wise. Zero-Order Hold blocks enforce the requirement that the discrete-time signals have a constant continuous value for the duration of the respective sample, while at the same time they act as translators between symbolic and self-validated computations.

As before, RobSim starts by first computing the block execution sequence of the given Simulink model. Then, the symbolic derivatives are derived or the discrete-time blocks are processed using self-validated arithmetics depending on which part of the model is updated. The process is repeated for each time step in the simulation.

Example 4 (Example 10-13 from [19]): Consider a discrete-time controller connected to the process $G_p(s) = \frac{50}{s(s+10)}$. The transfer function of the controller is $D(z) = \frac{5z^2-6z+2}{z^2}$ and the sampling rate is 10 Hz. Note that the model has 4 states: 2 continuous-time and 2 discrete-time state variables. We would like to study the correctness of the numerical integration. For the Simulink simulation, we use the ODE45 solver with no constraints on the step size. The resulting trajectory has 63 points (see Fig. 3). RobSim reports that the integration is not correct at 57 points. Note, though, that the graph (Fig. 3) indicates that the qualitative behavior of the simulation is correct. Therefore, the user might want to further relax the bounds in order to tolerate some of the integration errors as long as the qualitative behavior is correct.

V. Related Research and Discussion

Our work spans and adapts results from several (distinct) research areas. In the previous sections, we have cited

several papers which are related to the discussion in each section. Here, we summarize and discuss some works which are directly related to our proposed framework.

Affine arithmetic (AA) [10] has been a popular computation model for capturing floating-point rounding errors. In [13], the authors present a framework for the analysis of floating-point rounding errors in embedded control software and the corresponding toolbox FLUCTUAT. In brief, the underlying theory in [13] is based on abstract interpretation and on AA. Our work has different scope and goals from [13]. Besides the apparent differences in the systems that can be handled (e.g., discrete-time vs continuous-time), our aim is to study the robustness of a particular simulation with respect to internal and external uncertainties in the system, whereas FLUCTUAT computes system invariants and demonstrates how rounding errors propagate through the computation.

Another related research area is the reachability analysis [28] and verification [4] of dynamical systems with uncertain system parameters. Along this line of research, the works which are the closest related to ours appear in [15], [11] and [2]. In [15], the authors present a framework for performing simulation of discrete-time dynamical systems with uncertain parameters using semi-symbolic simulations. The systems are given in SystemC-AMS and the underlying computation model for the symbolic simulations is AA. The main difference between [15] and RobSimDt is that in [15] the authors compute the reachable set of a dynamical system without switching dynamics while RobSimDt determines the robustness of a trajectory of a discrete-time hybrid system. Moreover, RobSimDt can account for computation errors.

The papers [11] and [2] deal with the reachability analysis of continuous-time hybrid systems based on zonotope representations of the state vector. Zonotopes can be regarded as a special case of symbolic quantities in AA. In detail, [11] presents a reachability algorithm for linear (and hybrid) systems which do not have any uncertain system parameters. On the other hand, in [2], the authors extend their previous work in [1] to address the reachability problem of continuous-time hybrid systems with uncertain system parameters. Even though, our work in this paper subsumes the existence of a solution for the reachability problem of continuous-time uncertain hybrid systems, our results differ in two subtle points. That is, we have to take into account the possible numerical drift in the computation of time and, also, to model and capture computation errors. Finally, the goal of our work is different when compared to [2] and [1]. RobSimCt studies the robustness and correctness of a particular simulation.

Reachability analysis of mixed-signal circuits is addressed in [22] and [7]. However, these approaches do not handle uncertain parameters or computation errors. Finally, similar in spirit – but different in underlying theory – are the robust testing methods [12], [17], [20], [8]. In this line of work, a numerical simulation is performed which is representative of a neighborhood of trajectories of the system. Nonetheless, all the above approaches ignore computation errors.

VI. Conclusions

In this paper, we have presented a framework for reporting points where a simulation might not be robust. Our solution can model and capture both uncertainties in the model and internal computation errors. In addition, in the case of continuous-time Simulink models with piecewise affine dynamics, we can also detect points where the numerical integration is not correct.

One of the main advantages of our approach is that it is a non-intrusive, i.e, we let the simulation execute and, then, we analyze the resulting trajectories (both the Simulink trajectory and the block execution trace). Therefore, we overcome the key challenge of deciphering Simulink semantics and, moreover, we avoid translating the model into a formalism such as a hybrid automaton. To the best of our knowledge, this is the first time that robust simulations under many different sources of uncertainty are considered in Simulink and, most probably, in any model based development suite.

Even though our framework could be extended to perform bounded-time reachability computations for hybrid systems in Simulink, our goal is different. The RobSim suite is meant to be used as an analysis tool on top of a Simulink simulation and provide guarantees that the computation itself is correct against modeling uncertainties and floating-point rounding errors. Especially, the latter types of errors can be quite stealthy as we have indicated in the introduction. In addition, possible points of non-robustness in the simulation can be further explored by some lightweight method such as Monte-Carlo simulations. Beyond testing and verification, our framework could have other applications, too. For example, an efficient implementation could be used for detecting non-robust points for code generation frameworks as in [3].

Future research is targeting three problems. First, we are working on extending the framework to non-linear continuous models. This essentially requires a method to compute symbolically the derivatives of the system at each point in time. Second, we are going to provide full support for mixed-signal systems in RobSim and port our prototype Matlab implementation into C for more efficient computation. Currently, Matlab speed is a bottleneck for scaling the RobSim framework into larger systems. Finally, we plan to add methods for exploring the points of non-robustness in the simulations.

References

[1] M. Althoff, O. Stursberg, and M. Buss. Reachability analysis of linear systems with uncertain parameters and inputs. In *46th IEEE Conference on Decision and Control*, pages 726–732, Dec. 2007.

[2] M. Althoff, O. Stursberg, and M. Buss. Verification of uncertain embedded systems by computing reachable sets based on zonotopes. In *Proc. of the 17th IFAC World Congress*, 2008.

[3] M. Anand, S. Fischmeister, J. Kim, and I. Lee. Generating sound and resource-aware code from hybrid systems models. In *Model-Driven Development of Reliable Automotive Services,*, volume 4922 of *LNCS*, pages 48–66. Springer, 2008.

[4] G. Batt, C. Belta, and R. Weiss. Temporal logic analysis of gene networks under parameter uncertainty. *IEEE Transactions on Automatic Control*, 53:215–229, Jan. 2008.

[5] M. Blair, S. Obenski, and P. Bridickas. Patriot missile software problem. Technical Report GAO/IMTEC-92-26, United States General Accounting Office, 1992.

[6] O. Bouissou and M. Martel. Grklib: a guaranteed runge kutta library. In *Scientific Computing, Computer Arithmetic and Validated Numerics, International Symposium on*, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[7] T. Dang, A. Donzé, and O. Maler. Verification of analog and mixed-signal circuits using hybrid system techniques. In *Formal Methods in Computer-Aided Design*, volume 3312 of *LNCS*, pages 21–36. Springer, 2004.

[8] T. Dang, A. Donze, O. Maler, and N. Shalev. Sensitive state-space exploration. In *Proc. of the 47th IEEE Conference on Decision and Control*, pages 4049–4054, Dec. 2008.

[9] L. H. de Figueiredo and J. Stolfi. *Self-Validated Numerical Methods and Applications*. Brazilian Mathematics Colloquium monographs. IMPA/CNPq, Brazil, 1997.

[10] L. H. de Figueiredo and J. Stolfi. Affine arithmetic: Concepts and applications. *Numerical Algorithms*, 37(1-4):147–158, Dec. 2004.

[11] A. Girard. Reachability of uncertain linear systems using zonotopes. In *Hybrid Systems: Computation and Control*, volume 3414 of *LNCS*, pages 291–305, 2005.

[12] A. Girard and G. J. Pappas. Verification using simulation. In *Hybrid Systems: Computation and Control (HSCC)*, volume 3927 of *LNCS*, pages 272 – 286. Springer, 2006.

[13] E. Goubault, S. Putot, P. Baufreton, and J. Gassino. Static analysis of the accuracy in control systems: Principles and experiments. In *FMICS*, volume 4916 of *LNCS*, pages 3–20. Springer, 2007.

[14] T. A. Henzinger, B. Horowitz, R. Majumdar, and H. Wong-Toi. Beyond hytech: Hybrid systems analysis using interval numerical methods. In *Hybrid Systems: Computation and Control*, pages 130–144, London, UK, 2000. Springer-Verlag.

[15] W. Heupke, C. Grimm, and K. Waldschmidt. *Applications of Specification and Design Languages for SoCs*, chapter Modeling Uncertainty in Nonlinear Analog Systems with Affine Arithmetic, pages 155–169. Springer, 2006.

[16] E. P. Hofer and A. Rauh. Applications of interval algorithms in engineering. In *Proceedings of the 12th International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics*, page 1, 2006. IEEE Computer Society.

[17] A. A. Julius, G. E. Fainekos, M. Anand, I. Lee, and G. J. Pappas. Robust test generation and coverage for hybrid systems. In *Hybrid Systems: Computation and Control*, number 4416 in LNCS, pages 329–342. Springer, 2007.

[18] A. Kanade, R. Alur, F. Ivančić, S. Ramesh, S. Sankaranarayanan, and K. C. Shashidhar. Generating and Analyzing Symbolic Traces of Simulink/Stateflow Models. In *CAV*, pages 430–445. Springer, 2009.

[19] B. C. Kuo. *Digital Control Systems*. Oxford University Press, Inc., New York, NY, USA, 1992.

[20] F. Lerda, J. Kapinski, E. M. Clarke, and B. H. Krogh. Verification of supervisory control software using state proximity and merging. In *Hybrid Systems: Computation and Control*, volume 4981 of *LNCS*, pages 344–357. Springer, 2008.

[21] M. Liou. A novel method of evaluating transient response. *Proceedings of the IEEE*, 54(1):20–23, Jan. 1966.

[22] S. Little, N. Seegmiller, D. Walter, C. Myers, and T. Yoneda. Verification of analog/mixed-signal circuits using labeled hybrid petri nets. In *Proceedings of the 2006 IEEE/ACM CAD*, pages 275–282. ACM, 2006.

[23] E. Loh and G. W. Walster. Rumps example revisited. *Reliable Computing*, 8:245248, 2002.

[24] K. Makino and M. Berz. Cosy infinity version 9. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 558(1):346 – 350, 2006.

[25] R. E. Moore, R. B. Kearfott, and M. J. Cloud. *Introduction to Interval Analysis*. SIAM, 2009.

[26] E. P. Oppenheimer and A. N. Michel. Application of interval analysis techniques to linear systems. ii. the interval matrix exponential function. *IEEE Transactions on Circuits and Systems*, 35(10):1230–1242, Oct 1988.

[27] C. Phillips. Instabilities caused by floating-point arithmetic quantization. *IEEE Transactions on Automatic Control,*, 17(2):242–243, Apr 1972.

[28] N. Ramdani, N. Meslem, and Y. Candau. Reachability of uncertain nonlinear systems using a nonlinear hybridization. In *Proceedings of the 11th international workshop on Hybrid Systems*, pages 415–428, 2008. Springer.

[29] S. Rump. Algorithms for verified inclusions: Theory and practice. In R. E. Moore, editor, *Reliability in Computing: The Role of Interval Methods in Scientific Computing*, chapter 1, pages 109–126. Academic Press, 1988.

[30] S. Rump. INTLAB - INTerval LABoratory. In T. Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999.

[31] D. Stevenson. IEEE standard for binary floating-point arithmetic. Technical report, IEEE, August 1985.

[32] T. Yokozeki. An implementation of affine arithmetic on Matlab. Master's thesis, Waseda University, 2004.