



Abstract Simulation: a Static Analysis of Simulink Models

Alexandre Chapoutot, Matthieu Martel

► To cite this version:

Alexandre Chapoutot, Matthieu Martel. Abstract Simulation: a Static Analysis of Simulink Models. International Conference on Embedded Software and Systems, May 2009, Zhejiang, China. pp.83-92, 10.1109/ICESS.2009.80 . hal-00332447

HAL Id: hal-00332447

<https://hal.science/hal-00332447>

Submitted on 21 Oct 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Abstract Simulation: a Static Analysis of Simulink Models

Alexandre Chapoutot¹ and Matthieu Martel²

¹ LIP6 - Université Pierre et Marie Curie
4, place Jussieu F75252 Paris Cedex 05 France
`alexandre.chapoutot@lip6.fr`

² Laboratoire ELIAUS-DALI - Université de Perpignan Via Domitia
52, avenue Paul Alduy F66860 Perpignan Cedex France
`matthieu.martel@univ-perp.fr`

Abstract. Simulink is one of the most widely used industrial tools to design embedded systems. Applying formal methods sooner in the cycle of development is an important industrial challenge in order to reduce the cost of bug fixing. In this article, we introduce a new method, called Abstract Simulation and based on abstract interpretation of Simulink models. Abstract Simulation uses several numerical domains such as a domain for Taylor forms or floating-point numbers with errors. These domains allow us to estimate errors introduced by numerical algorithms and by computations during simulations. As a result, our method makes it possible to validate numerical behaviors of embedded systems modeled in Simulink. A prototype has been implemented and experimental results are commented.

1 Introduction

In regard to the growth of complexity of embedded systems, software tools are needed at design-time. Simulink³ or Lustre/Scade⁴ are the main industrial tools used in this context. Despite of the numerous features added in both tools, such as simulation, test or code generation, Simulink is more often used due to its important system design expressiveness. It allows us to model and simulate continuous-time and discrete-time systems, as well as a mix of both. For embedded systems, Simulink offers a convenient way to model and specify both the embedded software and the physical environment. The application of formal methods on such specifications is an important challenge for the validation of embedded software [6].

In this article, we define a static analysis by abstract interpretation [7] of Simulink models. This static analysis is called *Abstract Simulation (A.S.)* and it has been shortly introduced in [4]. The aim of *A.S.* is to provide a correctness criterion for the executions of Simulink models because they are often used for

³ Trademarks of Mathworks

⁴ Trademarks of Esterel Technologies

the validation of systems. However, such simulations are closer to test-based verifications than to formal proofs and, consequently, they do not permit to validate, in regards to the specification, a system. *A.S.* provides a correctness criterion for numerical behaviors of the Simulink models in the sense that they mimic what happens in the real world.

Our contribution. We assume that the mathematical model encoded as a Simulink model is correct (the physical system is correctly modeled). We aim at automatically and conjointly compute an over-approximation of the mathematical behaviors and the simulation behaviors for all the possible inputs of the model. We can thus estimate the whole imprecision introduced by the simulation, *i.e.* numerical errors as truncation errors and round-off errors as well as sensor errors like quantization and sampling. The correctness criterion of continuous-time models is given by the distance between numerical integration algorithm (Simulink *solver*) and guaranteed numerical integration algorithm based on Taylor method [21]. The correctness criterion of discrete-time models is given by using the abstract numerical domain of floating-point numbers with errors [18, 11]. Furthermore, *A.S.* uses two abstract numerical domains, such that the domain of floating-point numbers with errors and the domain of Taylor forms, and *A.S.* is also based on an abstraction of sequences using set partitioning.

Related work. Static analysis has already been successfully applied to high level design languages such as Lustre [17, 1], Signal [10] or VHDL [15]. The main difference with our approach is that they validate the embedded software, using assumptions on the environment while we consider, thanks to Simulink, the software and its environment of execution. Some applications of formal methods to Simulink models have been proposed by Tiwari [22], based on hybrid automata and symbolic computation. An other approach for applying formal methods to Simulink models is to translate them in other paradigms, like Lustre [3]. However, all these methods focus on either the mathematical behaviors or the execution behaviors but they do not provide a way to compute both behaviors together and then a way to compare them. The strength of our method is then to validate the mathematical model, that is the specification, of the embedded systems and its robustness to a set of imprecision which generally arises in the final systems.

Outline. In Section 2, we give an overview of the different features of *A.S.*. In section 3, we introduce Simulink through a running example. We define the numerical domain of Taylor forms in Section 4.1 while, in Section 4.2, we recall the numerical domain of floating point with errors. We define the abstractions of sequences in Section 5 before defining *A.S.*, in Section 6. Finally, experimental results are presented in Section 7.

2 Overview of Abstract Simulation

Simulink offers a convenient way to describe dynamic systems, that is systems evolving during time. Such systems are mathematically represented by systems of equations that are differential equations in case of continuous-time systems, difference equations in case of discrete-time systems and a mix of both in case

of hybrid systems. The simulation of Simulink models is based on numerical algorithms, so the solution of systems of equations (*i.e.* the semantics of Simulink models) is given by sequences of values representing temporal functions.

As consequence, *A.S.* is mainly based on two abstractions of sequences. Equation (1) describes those abstractions with a sequential composition of Galois connections.

$$(\mathbb{N} \rightarrow \wp(\mathcal{V}), \preceq_{\subseteq}) \xleftrightarrow[\alpha_{\mathcal{V}}]{\gamma_{\mathcal{V}}} (\mathbb{N} \rightarrow \mathcal{V}^{\sharp}, \preceq_{\subseteq^{\sharp}}) \xleftrightarrow[\alpha_{\mu}]{\gamma_{\mu}} (D \rightarrow \mathcal{V}^{\sharp}, \preceq_{\subseteq^{\sharp}}) \quad (1)$$

A set of sequences over a numerical values \mathcal{V} is denoted by $\mathbb{N} \rightarrow \wp(\mathcal{V})$, we see this set as the complete lattice of total function space. The first abstraction, defined with the Galois connection $(\alpha_{\mathcal{V}}, \gamma_{\mathcal{V}})$, turns set of values into an abstract numerical domain. We will use two abstract numerical domains in this paper: the domain of Taylor forms \mathcal{T} , defined in Section 4.1, and the domain of floating-point numbers with errors [18], denoted by \mathcal{E} , we will briefly recall the definition of it in Section 4.2. The second Galois connection $(\alpha_{\mu}, \gamma_{\mu})$ transforms infinite length sequences into finite length ones (D is a finite set) thanks to the partition function μ . We will define this abstraction in Section 5. The definition of *A.S.* using the abstract domain of sequences is given in Section 6. *A.S.* closely follows the simulation process of Simulink which is adapted to use the guarantee numerical integration method [21].

A.S. aims at statically analyze Simulink models that is validate the numerical behaviors of the specifications of embedded control systems. The semantics of Simulink is given by numerical algorithms. In consequence, *A.S.* uses various abstract numerical domains in order to follow semantics of Simulink. These domains are tuned to be compatible with such numerical algorithms which is not necessary the case with the polyhedral domains [9, 19]. Furthermore, the inaccurate experimental results using interval domain, especially to deal with numerical integration algorithms, have lead to consider more elaborated domains, in particular the domain of Taylor forms.

3 Presentation of Simulink

This section is organised as follows. First, we present the main aspects of the Simulink language through an example issued from an industrial case study. Next, we define the generation of the semantic equations related to Simulink models.

3.1 Simulink

Simulink is a tool to model and to simulate systems evolving during time as critical embedded systems are. In Figure 1, an example of Simulink model is given. It represents a simple braking detector and it is split in three sub-systems. A continuous-time sub-system, given in Figure 1(a), represents a braking pedal as a mass-spring-damper mechanical system. A discrete-time sub-system, given in Figure 1(b), detects when the pressing force on the pedal becomes greater

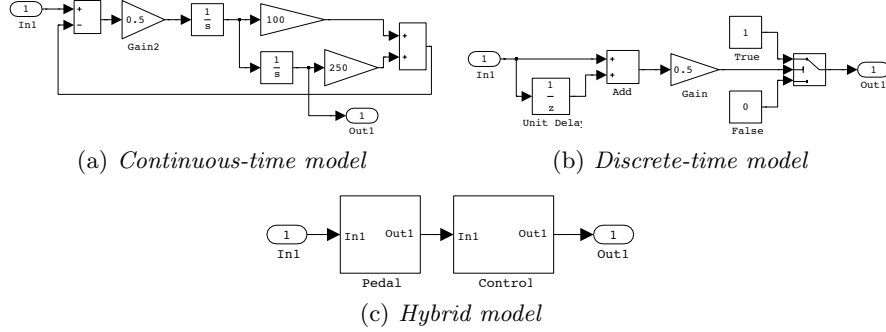


Fig. 1. A breaking pedal detector.

than a given threshold and then detects the braking action. Finally, the main system, given in Figure 1(c), has one input, the pressing force, and one output, the detection result. This last system is a simple composition of the continuous-time sub-system and of the discrete-time sub-system.

In regards to our example, we can see that Simulink models are composed of blocks connected by wires (named *signals*). These blocks are either elementary blocks such as arithmetic operations or sub-system blocks which are the composition of elementary blocks. We have listed in Figure 2 the main blocks used in this article. *Integrator* and *UnitDelay* are special because they introduce the notion of time (t represent the continuous-time variable and k represents the discrete-time one). When an *Integrator* is used, the model is called *continuous-time* and the operation associated to *Integrator* is a mathematical integration over time. Conversely, a model using *UnitDelay* is called *discrete-time*. A mix of these two operators yields an hybrid model defined in a fully dataflow way: signals are functions of time which are either continuous-time functions or discrete-time functions. We remark that a model which does not contain such blocks can be either continuous-time or discrete-time; we assume for the rest of the paper we know the type of such systems.

3.2 Semantic Equations

In this section, we describe the generation of the semantic equations for the blocks defined in the second column of Figure 2. $\mathcal{E}\{M\}$ denotes the semantic equations associated to the Simulink model M . We annotate each wire of M by a unique label ℓ . Moreover, each block *Integrator* and *UnitDelay* is also annotated by a label η . We denote by \mathcal{L} the finite set of labels. Semantic equations define the relation between the inputs and the outputs of a block. The equation associated to each block is given in the last column of Figure 2.

The system of equations associated to a Simulink model is described by a pair whose first argument is the set of semantic equations needed to compute the outputs of the model and the second argument is the set of semantic equations











Name	Block	Description	Equations
Input		System input	$\{\ell_1 = \text{In}(t), \emptyset\}$
Constant		Constant value	$\{\ell_1 = c, \emptyset\}$
Output		System output	$\{\text{Out}(t) = \ell_1, \emptyset\}$
Add		Addition	$\{\ell_3 = \ell_1 + \ell_2, \emptyset\}$
Sub		Subtraction	$\{\ell_3 = \ell_1 - \ell_2, \emptyset\}$
Gain		Multiplication by constant	$\{\ell_2 = g \times \ell_1, \emptyset\}$
Switch		Conditional statement	$\{\ell_4 = \text{if}(p(\ell_1), \ell_2, \ell_3), \emptyset\}$
Integrator		Continuous-time integration	$\{\ell_2(t) = \eta(t), \dot{\eta}(t) = \ell_1(t)\}$
UnitDelay		Discrete-time delay	$\{\ell_2(k) = \eta(k), \eta(k+1) = \ell_1(k)\}$
SubSystem		Sub-system	$\{\ell_2 = f(\ell_1), \emptyset\}$ with $f = \mathcal{E}\{S\}$

Fig. 2. Simulink blocks with their graphical representation and their equation.

computing the states of the model. Intuitively, states are values (*e.g.* previous outputs) of one model needed to compute its current output. We denote by $\mathcal{E}^o\{M\}$ the set of output semantic equations of M and by $\mathcal{E}^s\{M\}$ the set of state semantic equations of M . Then $\mathcal{E}\{M\} = \mathcal{E}^o\{M\} \cup \mathcal{E}^s\{M\}$. The system of semantic equations provides a mathematical description of the Simulink models close to the mathematical state-space formalism [14]. Remark that the *Integrator* blocks generate a first order ordinary differential equation. This is a key point of our semantics: for an integrator, we state that the output $\ell_2(t)$ is $\eta(t)$ which is constrained by the input $\ell_1(t)$ by the relation $\dot{\eta}(t) = \ell_1(t)$ where $\dot{\eta}$ denotes the first order derivative of η with respect to t . For the *UnitDelay* block, the semantic equation is a recursive relation of order one. Only these two blocks generate state semantic equations. When block generate only one set of semantic equations, the set of state semantic equation is empty (denoted by \emptyset). The semantic equations of the arithmetic operation blocks are very intuitive. For the *Switch* block, p is a predicate which is either $\ell_1 > c$ or $\ell_1 \geq c$ with ℓ_1 the first input signal of the block and c a constant value. A sub-system block S associates a set of equations $\mathcal{E}\{S\}$ named f and it generates one output equation $\ell_2 = f(\ell_1)$.

Remark that this subset of Simulink language is enough to deal with more complex blocks such as, for example, *State-Space* block which is a block asso-

ciated to set of linear differential equations or *Saturation* block which may be represented by two conditional statements.

4 Numerical Domains

We present the two abstract numerical domains used in the definition of *Abstract Simulation*. The domain of Taylor forms is defined in Section 4.1 and the domain of floating-point numbers with errors is defined in Section 4.2.

4.1 Domain of Taylor Forms

The aim of the domain of Taylor forms of order d is to compute, by an interval method, the image of an interval X by a real function \mathbf{f} . \mathbf{f} has to be d -times continuously differentiable and X has to be bounded. Domain of Taylor forms are used to extend numerical integration method, such as *Euler* or *Runge-Kutta*, to deal with interval values.

Concrete domain. We define the arithmetic operations over Taylor forms. We assume that we know how to compute the coefficients of Taylor forms. Automatic differentiation techniques [12] may be used for this purpose. We denote by \mathcal{T}^d the set of Taylor forms with remainder of order d . These terms are defined in Equation (2), $\mathbf{f} : \mathbb{R} \rightarrow \mathbb{R}$ is a d -times continuously differentiable function over the interval $[x_1, x_2]$, R_d is the Lagrange remainder and $h = |x_2 - x_1|$.

$$\mathbf{f}(x_2) = \mathbf{f}(x_1) + \sum_{i=1}^{d-1} \frac{h^i}{i!} \mathbf{f}^{(i)}(x_1) + R_d \quad (2)$$

$\mathbf{f}^{(i)}$ is the i -th derivative of \mathbf{f} and R_d is the remainder. This definition can be easily extended to deal with vectorial values. We represent Taylor forms by a vector of real values of length $d + 1$ where the latest element represent the remainder. That is, for a function \mathbf{f} at point x , its d -th order Taylor representation $T^d(\mathbf{f})(x)$ is:

$$T^d(\mathbf{f})(x) = \left(\mathbf{f}(x), h\mathbf{f}^{(1)}(x), \dots, \frac{h^{d-1}}{(d-1)!} \mathbf{f}^{(d-1)}(x), R_d \right) \quad (3)$$

We denote by $(T^d(\mathbf{f})(x))_i$ the i -th coefficient of the d -th order Taylor form of function \mathbf{f} at point x with $(T^d(\mathbf{f})(x))_0 = \mathbf{f}(x)$. We define the arithmetic of Taylor forms in Equation (4). We denote by $+_{\mathcal{T}}, -_{\mathcal{T}}, \times_{\mathcal{T}}$ the addition, the subtraction and the multiplication of Taylor forms. We restrict our presentation of the operations on Taylor forms to the ones needed for this article but such operations exist for division and for transcendental functions like \sin , \cos , \exp , etc. [12].

$$\begin{aligned} T^d(\mathbf{f}_1)(x) +_{\mathcal{T}} T^d(\mathbf{f}_2)(x) &= \forall i, (T^d(\mathbf{f}_1)(x))_i + (T^d(\mathbf{f}_2)(x))_i \\ T^d(\mathbf{f}_1)(x) -_{\mathcal{T}} T^d(\mathbf{f}_2)(x) &= \forall i, (T^d(\mathbf{f}_1)(x))_i - (T^d(\mathbf{f}_2)(x))_i \\ T^d(\mathbf{f}_1)(x) \times_{\mathcal{T}} T^d(\mathbf{f}_2)(x) &= \forall i, \sum_{j=0}^i (T^d(\mathbf{f}_1)(x))_j (T^d(\mathbf{f}_2)(x))_{i-j} \end{aligned} \quad (4)$$

We define, in Equation (5), the concrete semantics $\llbracket \cdot \rrbracket_{\mathbb{T}}$ based on Taylor forms by induction on the structure of expressions (we consider arithmetic and comparison expressions appearing in the right hand side of the equations in Figure 2). We denote by θ the environment function which maps variable to Taylor forms. A constant value c is associated to a constant function and all its derivatives are equal to zero. \diamond stands for an arithmetic operation among $\{+, -, \times\}$ and $\diamond_{\mathcal{T}}$ is the equivalent operation in the Taylor arithmetic. The comparison between Taylor forms and constant values c , Δ stands for $>$ or \geq , uses the function *eval* which evaluates Taylor forms into real values.

$$\begin{aligned}\llbracket c \rrbracket_{\mathbb{T}}(\theta) &= (c, 0, \dots, 0) \\ \llbracket \ell \rrbracket_{\mathbb{T}}(\theta) &= \theta(\ell) \\ \llbracket e_1 \diamond e_2 \rrbracket_{\mathbb{T}}(\theta) &= (\llbracket e_1 \rrbracket_{\mathbb{T}}(\theta)) \diamond_{\mathcal{T}} (\llbracket e_2 \rrbracket_{\mathbb{T}}(\theta)) \\ \llbracket e_1 \Delta c \rrbracket_{\mathbb{T}}(\theta) &= \text{eval}(\llbracket e_1 \rrbracket_{\mathbb{T}}(\theta)) \Delta c\end{aligned}\tag{5}$$

Abstract domain. We define the abstract domain associated to Taylor forms and based on the domain of intervals [7]. However, in our case, the straightforward abstraction by intervals yields very frequently imprecise results which are useless in practice. Instead, we use an abstraction called *Taylor inclusion function* [16] and based on the extension of the mean-value theorem.

For all function $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}$, $(d-1)$ -continuously differentiable on the interval $[a, b]$ and d -differentiable on the interval $]a, b[$, the Taylor-Lagrange form is:

$$\forall u \in [a, b], \exists z \in]a, b[, \mathbf{f}(u) = \sum_{i=0}^{d-1} \frac{u-c}{i!} \mathbf{f}^{(i)}(c) + \mathbf{f}^{(d)}(z)(u-c)$$

c stands for the center of $[a, b]$ and $\mathbf{f}^{(i)}$ denotes the i -th derivative of \mathbf{f} . Then,

$$\mathbf{f}([a, b]) \subseteq \sum_{i=0}^{d-1} \frac{u-c}{i!} \mathbf{f}^{(i)}(c) + [\mathbf{f}^{(d)}]([a, b])([a, b] - c)$$

$[\mathbf{f}^{(d)}]$ stands for the evaluation of $\mathbf{f}^{(d)}$ using interval arithmetic and it is a bound of the highest derivative on the interval $[a, b]$. Then, we obtain a new abstraction based on the interval domain which produces better bounds in practice. Let $(\alpha_{\mathbb{I}}, \gamma_{\mathbb{I}})$ be the Galois connection between the complete lattice $\langle \wp(\mathbb{R}), \subseteq \rangle$ and the complete lattice of intervals $\langle \mathbb{I}, \sqsubseteq_{\mathbb{I}} \rangle$. We write $m([a, b])$ the center of the interval $[a, b]$. $\text{eval}^{\sharp}((T^{d\sharp}(\mathbf{f}))(x))$ evaluates the centered interval Taylor forms at point x into interval values.

Proposition 1. $\langle \mathcal{T}^{d\sharp}, \sqsubseteq_{\mathcal{T}}^{\sharp}, \sqcup_{\mathcal{T}}^{\sharp}, \sqcap_{\mathcal{T}}^{\sharp} \rangle$ as defined below is a complete lattice:

$$\begin{aligned}T^{d\sharp}(\mathbf{f}_1)(x) \sqsubseteq_{\mathcal{T}}^{\sharp} T^{d\sharp}(\mathbf{f}_2)(x) &\stackrel{\text{def}}{=} (\text{eval}^{\sharp}(T^{d\sharp}(\mathbf{f}_1)(x))) \sqsubseteq_{\mathbb{I}}^{\sharp} (\text{eval}^{\sharp}(T^{d\sharp}(\mathbf{f}_2)(x))) \\ T^{d\sharp}(\mathbf{f}_1)(x) \sqcup_{\mathcal{T}}^{\sharp} T^{d\sharp}(\mathbf{f}_2)(x) &\stackrel{\text{def}}{=} (z_0^{\sharp}, z_1^{\sharp}, \dots, z_d^{\sharp}) \text{ where } \forall i < d, \\ z_i^{\sharp} &= m\left(\left[\left(T^{d\sharp}(\mathbf{f}_1)(x)\right)_i, \left(T^{d\sharp}(\mathbf{f}_2)(x)\right)_i\right]\right) \text{ and } z_d^{\sharp} = \left(T^{d\sharp}(\mathbf{f}_1)(x)\right)_d \sqcup_{\mathbb{I}} \left(T^{d\sharp}(\mathbf{f}_2)(x)\right)_d \\ T^{d\sharp}(\mathbf{f}_1)(x) \sqcap_{\mathcal{T}}^{\sharp} T^{d\sharp}(\mathbf{f}_2)(x) &\stackrel{\text{def}}{=} (z_0^{\sharp}, z_1^{\sharp}, \dots, z_d^{\sharp}) \text{ where } \forall i < d, \\ z_i^{\sharp} &= m\left(\left[\left(T^{d\sharp}(\mathbf{f}_1)(x)\right)_i, \left(T^{d\sharp}(\mathbf{f}_2)(x)\right)_i\right]\right) \text{ and } z_d^{\sharp} = \left(T^{d\sharp}(\mathbf{f}_1)(x)\right)_d \sqcap_{\mathbb{I}} \left(T^{d\sharp}(\mathbf{f}_2)(x)\right)_d\end{aligned}$$

The comparison \sqsubseteq_T^\sharp compares the interval values associated to the Taylor forms. The join operation \sqcup_T^\sharp computes the centered values of the convex hull of the $d-1$ first elements, $[r_1, r_2]$ denotes the convex hull of the points r_1 and r_2 , and computes the interval join result of the d -th elements. The meet operations \sqcap_T^\sharp is similar to \sqcup_T^\sharp .

Proposition 2. (*Galois connection of centered interval Taylor forms*)

$$(\wp(\mathcal{T}_d), \subseteq) \xleftrightarrow[\alpha_T]{\gamma_T} (T^{d\sharp}, \sqsubseteq_T^\sharp)$$

with $\alpha_T (\{T^d(\mathbf{f})(x_j) : 1 \leq j \leq n\}) \stackrel{\text{def}}{=} (z_0^\sharp, z_1^\sharp, \dots, z_d^\sharp)$ where $\forall i < d$,

$$z_i^\sharp = m \left(\alpha_{\mathbb{I}} \left(\{(T^d(\mathbf{f})(x_j))_i : 1 \leq j \leq n\} \right) \right) \text{ and } z_d^\sharp = \alpha_{\mathbb{I}} \left(\{(T^d(\mathbf{f}_j)(x))_d : 1 \leq j \leq n\} \right)$$

and $\gamma_T (T^{d\sharp}(\mathbf{f})(x)) \stackrel{\text{def}}{=} \{(z_0, z_1, \dots, z) : z \in Z_d\}$ where $\forall i < d$,

$$z_i = (T^{d\sharp}(\mathbf{f})(x))_i \text{ and } Z_d = \gamma_{\mathbb{I}} \left((T^{d\sharp}(\mathbf{f})(x))_d \right)$$

The abstract semantics $\llbracket \cdot \rrbracket_{\mathbb{T}}^\sharp$ is obtained by abstracting sets of Taylor forms by centered interval Taylor forms. The semantics of arithmetic operations is straightforward and the semantics of comparisons uses the function $eval^\sharp$.

4.2 Domain of Floating-point Numbers with Errors

In this section, we briefly recall the definition of the second numerical abstract domains used in this article: the domain of floating-point numbers with errors. We aim at validating numerical behaviors of Simulink models and, more precisely, the influence of the finite precision on the simulation process. Intuitively, a floating-point number with error is a pair made of the same floating-point number manipulated by the computer and of an exact error term which is the distance between the value used by the machine and the exact result (computing with infinite precision) of the calculation. So, floating-point numbers with errors allows us to follow the control-flow induced by the computer arithmetic while tracking the errors due to finite precision.

In Equation (6), we recall the definition of the arithmetic operations of floating-point numbers with errors. A real constant value r is represented by the couple $(\uparrow_\circ(r), \downarrow_\circ(r))$ where \uparrow_\circ is the rounding function mapping real values to floating-point values and \downarrow_\circ is the rounding error function which gives the distance between the real value and its floating-point representation. So, $\downarrow_\circ(r) = r - \uparrow_\circ(r)$. A variable a is represented by a couple (f_a, e_a) with f_a the floating-point value and e_a the sum of all the errors arisen during the computation. We focus our presentation on the only operators used in this article. More details are given in [18].

$$\begin{aligned} a +_{\mathcal{E}} b &= (\uparrow_\circ(f_a + f_b), e_a + e_b + \downarrow_\circ(f_a + f_b)) \\ a -_{\mathcal{E}} b &= (\uparrow_\circ(f_a - f_b), e_a - e_b + \downarrow_\circ(f_a - f_b)) \\ a \times_{\mathcal{E}} b &= (\uparrow_\circ(f_a \times f_b), e_a f_b + e_b f_a + e_a e_b + \downarrow_\circ(f_a \times f_b)) \end{aligned} \tag{6}$$

We denote by ϕ the environment function which maps variable to floating-point with errors values. The concrete semantics $\llbracket \cdot \rrbracket_{\mathbb{E}}$ is given by extending the arithmetic operations to floating-point numbers with errors operations. The comparison operations is performed either on the floating-point part or on the real value in order to follow the simulation control flow or the mathematical control flow. \uparrow stands for the evaluation function which computes the floating-point value or the real value of the floating-point with errors value.

$$\begin{aligned}\llbracket c \rrbracket_{\mathbb{E}}(\phi) &= (\uparrow \circ (c), \downarrow \circ (c)) \\ \llbracket \ell \rrbracket_{\mathbb{E}}(\phi) &= \phi(\ell) \\ \llbracket e_1 \diamond e_2 \rrbracket_{\mathbb{E}}(\phi) &= (\llbracket e_1 \rrbracket_{\mathbb{E}}(\phi)) \diamond_{\mathcal{E}} (\llbracket e_2 \rrbracket_{\mathbb{E}}(\phi)) \\ \llbracket e_1 \Delta c \rrbracket_{\mathbb{E}}(\phi) &= \uparrow(\llbracket e_1 \rrbracket_{\mathbb{E}}(\phi)) \Delta \uparrow(\llbracket c \rrbracket_{\mathbb{E}}(\phi))\end{aligned}\tag{7}$$

The abstract semantics $\llbracket \cdot \rrbracket_{\mathbb{E}}^{\#}$ is obtained by abstracting the two elements of a floating-point number with error, that is the floating-point number and the rounding error, into intervals and by extending the arithmetic operations to these of the interval arithmetic.

5 Domain of Sequences

In this section, we define the two abstractions over sequences introduced in Equation (1). The first is a classical abstraction of total function space. The second is an abstraction over partitions of a set. We assume that \mathcal{V} is a numerical domain which is either the domain of Taylor forms, defined in Section 4.1, or the domain of floating-point numbers with errors, defined in Section 4.2.

Let $\{\mathbb{N} \rightarrow \wp(\mathcal{V})\}$ the complete lattice of total functions ordered by \preceq_{\subseteq} :

$$\forall \mathbf{f}_1, \mathbf{f}_2 \in \{\mathbb{N} \rightarrow \wp(\mathcal{V})\}, \mathbf{f}_1 \preceq_{\subseteq} \mathbf{f}_2 \Leftrightarrow \forall n \in \mathbb{N}, \mathbf{f}_1(n) \subseteq \mathbf{f}_2(n)$$

Even if we take $\{\mathbb{N} \rightarrow \wp(\mathcal{V})\}$ for our concrete semantic domain, many consider it as an abstraction of $\{\wp(\mathbb{N} \rightarrow \mathcal{V})\}$ the set of maps from \mathbb{N} to \mathcal{V} . If there is a Galois connection $(\alpha_{\mathcal{V}}, \gamma_{\mathcal{V}})$ such that:

$$(\wp(\mathcal{V}), \subseteq) \xleftrightarrow[\alpha_{\mathcal{V}}]{\gamma_{\mathcal{V}}} (\mathcal{V}^{\#}, \sqsubseteq^{\#})$$

then, we have the complete lattice of total functions $\{\mathbb{N} \rightarrow \mathcal{V}^{\#}\}$ ordered by $\preceq_{\sqsubseteq^{\#}}$:

$$\forall \mathbf{f}_1, \mathbf{f}_2 \in \{\mathbb{N} \rightarrow \mathcal{V}^{\#}\}, \mathbf{f}_1 \preceq_{\sqsubseteq^{\#}} \mathbf{f}_2 \Leftrightarrow \forall n \in \mathbb{N}, \mathbf{f}_1(n) \sqsubseteq^{\#} \mathbf{f}_2(n)$$

For each function \mathbf{f} in $\{\mathbb{N} \rightarrow \wp(\mathcal{V})\}$, we can define the Galois connection $(\alpha'_{\mathcal{V}}(\mathbf{f}), \gamma'_{\mathcal{V}}(\mathbf{g}))$ such that:

$$\begin{aligned}\alpha'_{\mathcal{V}}(\mathbf{f}) &= \alpha_{\mathcal{V}} \circ \mathbf{f} \\ \gamma'_{\mathcal{V}}(\mathbf{g}) &= \gamma_{\mathcal{V}} \circ \mathbf{g} \text{ with } \mathbf{g} = \alpha'_{\mathcal{V}}(\mathbf{f})\end{aligned}$$

This first abstraction offers a way to deal with set of behaviors of Simulink models induced by set of inputs. The second abstraction, introduced below, is a way to make an infinite length sequences fit into a finite length sequence using

the partitioning of the set \mathbb{N} . A partition of \mathbb{N} is given by a function $\mu : \mathbb{N} \rightarrow D$ which defines an equivalence relation \sim_μ such that:

$$\forall n_1, n_2 \in \mathbb{N}, n_1 \sim_\mu n_2 \Leftrightarrow \mu(n_1) = \mu(n_2)$$

Let $P_\mu(\mathbb{N})$ the partition associated to the equivalence relation \sim_μ , that is:

$$P_\mu(\mathbb{N}) = \{\{n : n \in \mathbb{N}, n \sim_\mu k\} : k \in D\}$$

Let $P_\mu(\mathbb{N})|_k$ the k -th element of the set $P_\mu(\mathbb{N})$, that is: $P_\mu(\mathbb{N})|_k = \{n : n \in \mathbb{N}, n \sim_\mu k\}$. As sequences are associated to temporal functions, we require the function μ to have its co-domain D to be a finite set because the size of D is associated to the number of elements in the partition that is the length of the sequences. An example of such function is:

$$\mu_1(n) = \begin{cases} n & \text{if } n \leq l \\ l+1 & \text{otherwise.} \end{cases} \quad (8)$$

This function separates the l first elements and it gathers all the other elements in one.

Theorem 1. $\{\mathbb{N} \rightarrow \mathcal{V}^\sharp, \preceq_\square^\sharp\} \xleftrightarrow[\alpha_\mu]{\gamma_\mu} \{D \rightarrow \mathcal{V}^\sharp, \preceq_\square^\sharp\}$ is a Galois connection. With,

$$\alpha_\mu(\mathbf{f}) = \begin{cases} D \rightarrow \mathcal{V}^\sharp \\ k \mapsto \bigsqcup_{n \in P_\mu(\mathbb{N})|_k}^\sharp \mathbf{f}(n) \end{cases} \quad \gamma_\mu(\mathbf{f}^\sharp) = \begin{cases} \mathbb{N} \rightarrow \mathcal{V}^\sharp \\ n \mapsto \mathbf{f}^\sharp(k) \text{ with } k \sim_\mu n \end{cases}$$

Proof. Straightforward use of the definitions of α_μ and γ_μ . More details are given in Appendix A. \square

The semantics of Simulink based in the domain of sequences is very intuitive. Each equation of the system of equations $\mathcal{E}\llbracket M \rrbracket$ associated to Simulink models M is split in two equations: one for the current instant, *e.g.* the equation associated to output and which is of the form $\ell(k) = e$; and one for the next instant, *e.g.* the equation associated to a state, of the form $\ell(k+1) = e$. k denotes the discrete-time variable. The concrete semantics $\llbracket \cdot \rrbracket_{\mathbb{K}}$ is defined as a substitution of the k -th or of the $(k+1)$ -th element of the sequences associated to the variable ℓ . The abstract semantics $\llbracket \cdot \rrbracket_{\mathbb{K}}^\sharp$ is based on the function μ and it is defined in Equation (9). We denote by σ^\sharp the function mapping variable to sequences over \mathcal{V}^\sharp , and σ_i^\sharp stands for the environment mapping variable to i -th element of the sequence in σ^\sharp . To each instant k is associated an abstract instant $\mu(k)$. At instant $\mu(k)$, the abstract sequences gathers all the values of the concrete sequences for each instant k in relation with $\mu(k)$ by \sim_μ .

$$\begin{aligned} \llbracket \ell(k) = e \rrbracket_{\mathbb{K}}^\sharp(\sigma^\sharp) &= \sigma_{\mu(k)}^\sharp \left[\ell \leftarrow \sigma_{\mu(k)}^\sharp(\ell) \sqcup_{\mathcal{V}}^\sharp \llbracket e \rrbracket_{\mathcal{V}}^\sharp(\sigma_{\mu(k)}^\sharp) \right] \\ \llbracket \ell(k+1) = e \rrbracket_{\mathbb{K}}^\sharp(\sigma^\sharp) &= \sigma_{\mu(k+1)}^\sharp \left[\ell \leftarrow \sigma_{\mu(k+1)}^\sharp(\ell) \sqcup_{\mathcal{V}}^\sharp \llbracket e \rrbracket_{\mathcal{V}}^\sharp(\sigma_k^\sharp) \right] \end{aligned} \quad (9)$$

6 Static Analysis of Hybrid Simulink Models

In this section, we define *Abstract Simulation* using domains of Sections 4.1, 4.2 and 5. The simulation process of Simulink gives an effective way to compute sequences of values. Such sequences are the approximate numerical solutions of the system of equations attached to Simulink models. Moreover, the approximations are dependent of the numerical integration algorithms used during the simulation. Then, we define a static analysis which is parametrized by the numerical integration algorithm. This makes our approach generic for the static analysis Simulink models.

Simulink uses a lot of features coming from compilation (*e.g.* typing, reduction of block number, ...) but if we only considered the simulation loop, we obtain these three steps: *i)* Computing the outputs of the model *ii)* Computing the states of the model and *iii)* computing the time at the next step. The main complex operations are in the step *ii)*, especially in the case of continuous-time models because this step involves numerical integration. But in the case of discrete-time models, our semantics $\langle \cdot \rangle_{\mathbb{D}}$ is straightforwardly given by the domain of floating-point numbers with errors and the semantics of sequences because step *ii)* does not need the use of complex numerical algorithms.

In Section 6.1, we define the semantics of continuous-time models, denoted by $\langle \cdot \rangle_{\mathbb{C}}$. We will show, in Section 6.2, that the semantics of hybrid Simulation models is a combination of the previous defined semantics.

6.1 Semantics of Continuous-time Models

The mathematical behavior of Simulink models is given using guaranteed numerical integration algorithms [21, 2] while the simulation behaviors is given by an abstraction of numerical integration algorithms with Taylor forms.

For the sake of simplicity, we examine now how the guaranteed integration algorithms are added to the simulation loop. Recall that a Simulink model M is represented by a set of equations $\mathcal{E}\{M\} = \mathcal{E}^o\{M\} \cup \mathcal{E}^s\{M\}$. The output of the model is the result of the system of equations $\mathcal{E}^o\{M\}$ and the state evolution is given by the system of equations $\mathcal{E}^s\{M\}$. Let ρ_k the environment based on Taylor form and E_k the guaranteed environment, the loop iteration at instant k is given by:

- i)* Compute the outputs of the model that is evaluated $\mathcal{E}^o\{M\}$ in ρ_k ,
- ii)* Evaluate $\mathcal{E}^s\{M\}$ in ρ_k where each expression $\dot{\eta} = e$ are substituted by $\eta(k+1) = \eta(k) + he$ with h the integration step. In other words, differential equations are transformed into recurrence equations following *Euler* algorithm,
- iii)* Compute the guaranteed outputs of the model evaluating $\mathcal{E}^o\{M\}$ in E_k ,
- iv)* Compute the guaranteed states using the Taylor method [21] in E_k ,
- v)* Compute the distance between the result of steps *ii)* and step *iv)*.

Remark that the simulation loop, when using more sophisticated methods, such as *Runge-Kutta* is similar. The main changes are the evaluation of $\mathcal{E}^o\{M\}$ which

needs to be done several times, at different instants in order to apply the numerical integration method [13].

6.2 Semantics of Hybrid Models

We define now the semantics of hybrid Simulink models based on the semantics of continuous-time models $\langle \cdot \rangle_{\mathbb{C}}$ and the semantics of discrete-time models $\langle \cdot \rangle_{\mathbb{D}}$. $\langle \cdot \rangle_{\mathbb{C}}$ stands for the semantics based on the sequence domain over the Taylor domain and $\langle \cdot \rangle_{\mathbb{D}}$ stands for the semantics based on the sequence domain over the floating-point with errors domain. We assume that a Simulink model M is represented by a list L of sub-systems such as the model of Figure 1(c). In addition, we assume that we know the kind of each sub-system, that is, if it is a continuous-time or a discrete-time sub-system. Moreover, we assume that elements in the list are in the topological order. The evaluation of M is then the evaluation of the list L . The loop of simulation is simple. Assuming that initial values are sets, starting from the first element $L(1)$ of L , we apply, until the last element of L is reached, the following steps:

- If $L(i)$ is a continuous-time sub-system then apply $\langle \cdot \rangle_{\mathbb{C}}$,
 - if $L(i+1)$ is a continuous-time sub-system, continue with $\langle \cdot \rangle_{\mathbb{C}}$,
 - if $L(i+1)$ is a discrete-time sub-system, convert the results into floating-point numbers with errors and continue with $\langle \cdot \rangle_{\mathbb{D}}$;
- if $L(i)$ is a discrete-time sub-system then apply $\langle \cdot \rangle_{\mathbb{D}}$,
 - if $L(i+1)$ is a discrete-time sub-system, continue with $\langle \cdot \rangle_{\mathbb{D}}$,
 - if $L(i+1)$ is a continuous-time sub-system, either we take the floating-point value or the real value (that is the sum of the floating-point value and the error term) and continue with $\langle \cdot \rangle_{\mathbb{C}}$.

The choice made in the cast from discrete-time to continuous-time enables to cope with the combinatorial explosion due to the necessity to follow both possible paths in the analysis. So, we choose, before the analysis, to follow the computer path in the control flow or the mathematical one thanks to the function \uparrow (see Section 4.2).

7 Experimental Results

A static analyzer of Simulink programs has been implemented, including *Euler's* algorithm and using interval Taylor domain, floating-point numbers with errors and the abstraction of sequences. In this section, we consider two Simulink models: in Section 7.1, the one given in Figure 1 and, in Section 7.2, an electronic throttle control system. In all the experiments, we have used the validated numerical solver of differential equations VNODE[20] to compute the real behaviors of the continuous-time systems. Moreover, the cast of values v between continuous-time system and discrete-time system, that is the quantization activity of the sensor, is made by converting v in floating-point with a precision of 10 bits.

7.1 Breaking Pedal Detector

For this first experiment, we take the input function $u(t)$ depicted in Figure 3(a) and defined by:

$$u(t) = \begin{cases} [100, 200] \times t & \text{if } 0 \leq t < 1 \\ [100, 200] & \text{if } 1 \leq t < 1.5 \\ 0 & \text{otherwise} \end{cases}$$

The gray area represent the set of values and we represent with the black line

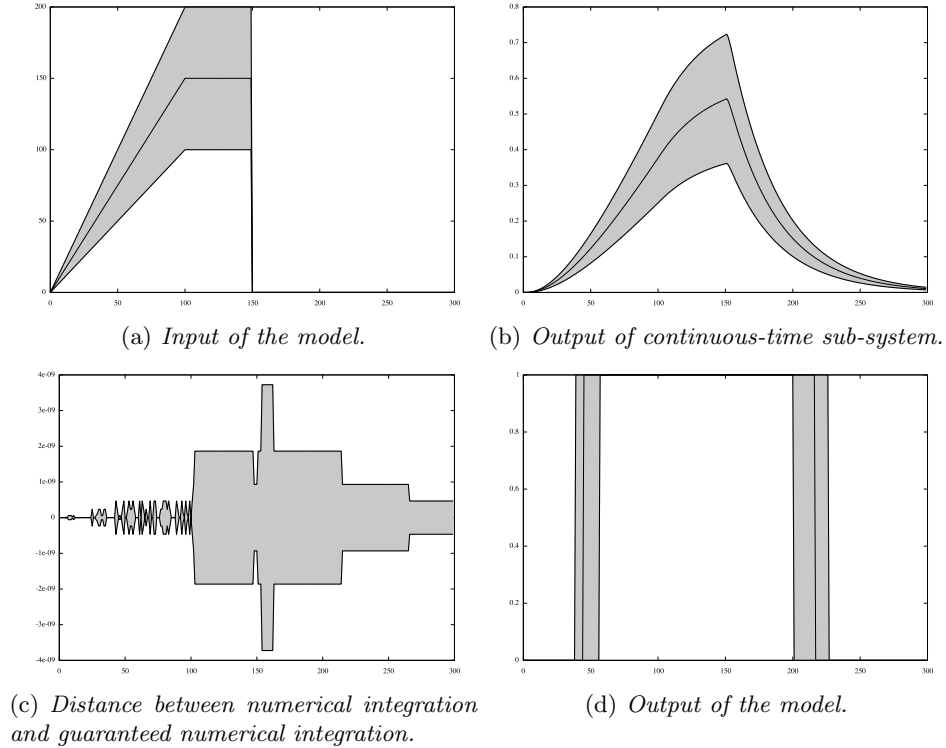


Fig. 3. Results analysis of system at Figure 1.

a particular trajectory computed with Simulink. We use the partition function μ defined in Equation (8) with $l = 3$. The output of the continuous-time system, given by interval *Euler* algorithm with the integration step $h = 0.01$, is represented in Figure 3(b). The correction criterion given by the maximal error between the simulation and the guarantee solution, in Figure 3(c), shows the numerical stability of the system design. The maximal error is less than $4e^{-9}$ for the all the analysis duration. The output of the system is given in Figure 3(d). The controller, with a threshold equal to 0.1, correctly detects the pressing force on the pedal for this set of inputs and the rounding-errors do not exceed

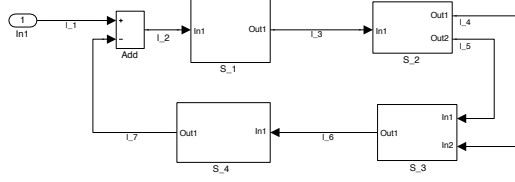


Fig. 4. Simulink model of an electronic throttle control system.

$5e^{-8}$. Furthermore, we remark that we fully enclose the Simulink results obtain by simulation with an input equals to the mean of the interval input.

7.2 Electronic Throttle Controller

The second example is an electronic throttle control system⁵ which is a common automobile feature used, for example, in fuel injection. The Simulink model is given in Figure 4. The system is made of four sub-systems in a feedback loop. S_1 is a proportional-integral controller (PI controller), S_3 is the mathematical model of the dynamic of the throttle, S_2 and S_4 are conversion operations. The dynamic of the throttle is described mathematically in Equation (10). A throttle is a valve described by its angular position θ and its angular velocity ω . It is controlled by an electric motor represented by its rotating direction (Direction) and its rotating speed (Speed).

$$\begin{cases} T(t) = \text{Direction} \times \text{Speed} \times 3 \\ \dot{\omega}(t) = 4081.6(-0.2(\theta(t) - 0.5223599) - 0.03\omega(t) + T(t)) \text{ with } 0 < \theta < \pi/2 \\ ((\theta < 0 \wedge \text{sign}(\dot{\omega}(t)) = -1) \vee ((\theta > \pi/2 \wedge \text{sign}(\dot{\omega}(t)) = 1))) \Rightarrow \dot{\omega}(t) = 0 \end{cases} \quad (10)$$

We set as input of the system a periodic function, depicted in Figure 5(a). The period of 2 seconds is split into two parts: the first second is an interval constant force between $[200, 600]$ and, in the last second, the function is equal to zero. We use in this case the partition function μ defined by considering the first period and componentwisely gathering all the elements of the next periods.

The output of the PI controller is given in Figure 5(c). Moreover, the rounding-errors are bounded in the interval $[-3e^{-15}, 3e^{-15}]$ (see Figure 5(d)). These results show that the PI controller is numerically stable and follows the dynamic of its input. The output of S_3 , given in Figure 5(b) with the integration step $h = 0.01$, is include in the interval $[0, \pi/2]$ and its dynamic follows the input of the systems. The static analysis of this Simulink model validates that for this set of inputs and for an infinite duration the electronic throttle controller is numerically stable despite computer arithmetic and sensor errors.

⁵ Issue from the Simulink course (SL01) given by Mathworks

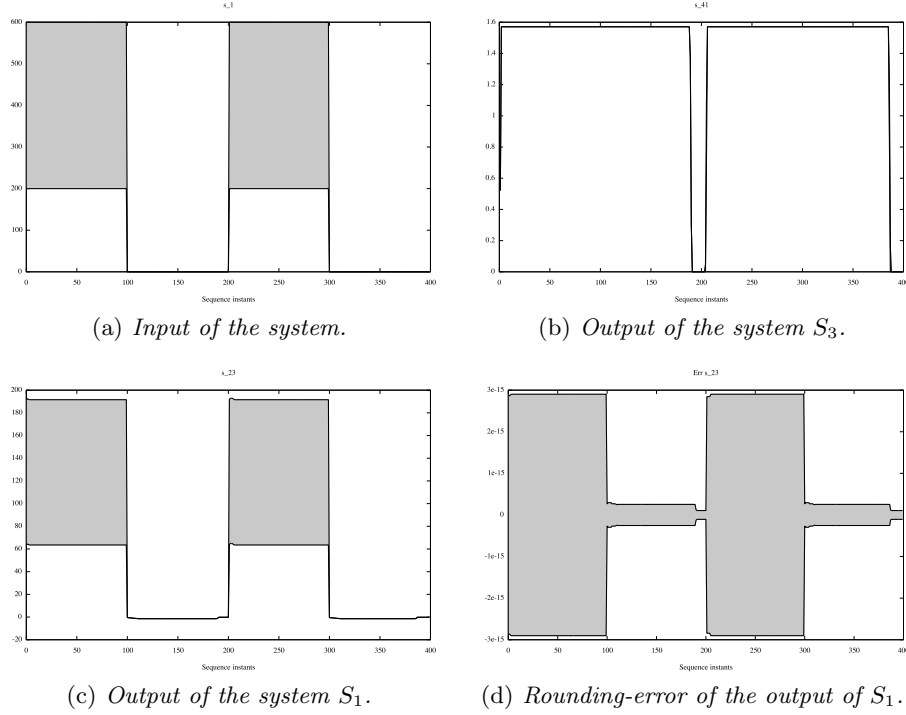


Fig. 5. Experimental results of the electronic throttle controller.

8 Conclusion

In this article, we have presented a new method, called *Abstract Simulation*, to validate Simulink programs by static analysis. A.S. enables one to analyze both the continuous-time and discrete-time sub-systems of a Simulink program. In addition, *Abstract Simulation* provides a correctness criterion for the numerical quality of large sets of simulations. A.S. computes the distance between the simulation result and the mathematical result of the continuous-time models and it computes the distance between the floating-point behaviors and the mathematical ones. A.S. is based on the new abstract numerical domain of Taylor forms and on an abstraction of sequences using set partitioning. Hence, we can validate the design of embedded systems and assert that a program may be embedded in a certain (physical) system.

Our experimental results highlight the advantages, for the validation of critical embedded systems, of applying static analysis on high-level design languages: first, we can take into account the physical environment during the analysis. Second, we can bound safely the approximation errors arising during a simulation. Third, we may treat in one pass a large class of inputs. We strongly believe that, as more and more complex systems are designed using Simulink-like tools, the

needs to detect bugs as soon as possible in the development process are going to increase hugely. We also believe that static analysis is the adequate framework to perform the validation required in this context. The results presented in this article are our first steps in this direction.

In future work, we will take into account more language features of Simulink such as Stateflow, the statecharts plug-in, that is the combination of dataflow language and automata like [5]. Moreover, the correctness criterion computed by *A.S.* offers a new perspective to validate more complex properties such as temporal ones. We currently investigate the use of the temporal abstract interpretation framework [8] to achieve this goal.

References

1. J. Bertrane. Proving the Properties of Communicating Imperfectly-Clocked Synchronous Systems. In *Static Analysis Symposium*, 2006.
2. O. Bouissou and M. Martel. GRKLib: a Guaranteed Runge-Kutta Library. In *Scientific Computing, Computer Arithmetic and Validated Numerics*, 2006.
3. P. Caspi, A. Curic, A. Maignan, C. Sofronis, and S. Tripakis. Translating Discrete-Time Simulink to Lustre. In *Embedded Computing Systems*, 2003.
4. A. Chapoutot and M. Martel. Static Analysis of Simulink Programs (short paper). In *Model-Driven High-Level Programming of Embedded Systems*, 2008.
5. J.-L. Colaço, G. Hamon, and M. Pouzet. Mixing Signals and Modes in Synchronous Data-flow Systems. In *Embedded Software*, 2006.
6. P. Cousot. Integrating physical systems in the static analysis of embedded control software. In *Asian Symposium on Programming Languages and Systems*, 2005.
7. P. Cousot and R. Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Principles of Programming Languages*, 1977.
8. P. Cousot and R. Cousot. Temporal Abstract Interpretation. In *Principles of programming languages*, 2000.
9. P. Cousot and N. Halbwachs. Automatic Discovery of Linear Restraints Among Variables of a Program. In *Principles of Programming Languages*, 1978.
10. A. Gamatié, T. Gautier, and L. Bensard. An Interval-Based Solution for Static Analysis in the Signal Language. In *Engineering of computer-based systems*, 2008.
11. E. Goubault, M. Martel, and S. Putot. Static Analysis-Based Validation of Floating-Point Computations. In *Numerical software with result verification*, 2004.
12. A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Society for Industrial and Applied Mathematics, 2000.
13. E. Hairer, S.P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer, 1993.
14. D. Hinrichsen and A.J. Pritchard. *Mathematical Systems Theory I : Modelling, State Space Analysis, Stability and Robustness*. Springer, 2005.
15. C. Hymans. Checking Safety Properties of Behavioral VHDL Descriptions by Abstract Interpretation. In *Static Analysis Symposium*, 2002.
16. L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis*. Springer, 2001.
17. B. Jeannet, N. Halbwachs, and P. Raymond. Dynamic Partitioning in Analyses of Numerical Properties. In *Static Analysis Symposium*, 1999.

18. M. Martel. Semantics of Roundoff Error Propagation in Finite Precision Computations. *Journal of Higher Order and Symbolic Computation*, 19(1):7–30, 2006.
19. A. Miné. The octagon abstract domain. *Journal of Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.
20. N.S. Nedialkov and K.R. Jackson. The design and implementation of an object-oriented validated ode solver, 2002.
21. N.S. Nedialkov, K.R. Jackson, and G.F. Corliss. Validated Solutions of Initial Value Problems for Ordinary Differential Equations. *Applied Mathematics and Computation*, 105(1):21–68, 1999.
22. A. Tiwari. Formal Semantics and Analysis Methods for Simulink Stateflow models. Technical report, SRI International, 2002.

A Proof of Theorem 1

Proof. (α_μ, γ_μ) is a Galois connection between $\{\mathbb{N} \rightarrow \wp(\mathcal{V}), \preceq_{\sqsubseteq^\#}\}$ and $\{D \rightarrow \mathcal{V}^\#, \preceq_{\sqsubseteq^\#}\}$ with $\mu : \mathbb{N} \rightarrow D$ a partition function.

– α_μ is monotone: $\forall \mathbf{f}_1, \mathbf{f}_2 \in \{\mathbb{N} \rightarrow \mathcal{V}^\#, \preceq_{\sqsubseteq^\#}\}$

$$\begin{aligned}
\mathbf{f}_1 \preceq_{\sqsubseteq^\#} \mathbf{f}_2 &\Leftrightarrow \forall n \in \mathbb{N}, \mathbf{f}_1(n) \sqsubseteq^\# \mathbf{f}_2(n) \\
&\Leftrightarrow \bigsqcup_{n \in P_\mu(\mathbb{N})|_1}^\# \mathbf{f}_1(n) \sqcup^\# \dots \sqcup^\# \bigsqcup_{n \in P_\mu(\mathbb{N})|_k}^\# \mathbf{f}_1(n) \sqsubseteq^\# \bigsqcup_{n \in P_\mu(\mathbb{N})|_1}^\# \mathbf{f}_2(n) \sqcup^\# \dots \sqcup^\# \bigsqcup_{n \in P_\mu(\mathbb{N})|_k}^\# \mathbf{f}_2(n) \\
&\Leftrightarrow \forall i \in D, \bigsqcup_{n \in P_\mu(\mathbb{N})|_i}^\# \mathbf{f}_1(n) \sqsubseteq^\# \forall i \in D, \bigsqcup_{n \in P_\mu(\mathbb{N})|_i}^\# \mathbf{f}_2(n) \\
&\Leftrightarrow \alpha_\mu(\mathbf{f}_1) \preceq_{\sqsubseteq^\#} \alpha_\mu(\mathbf{f}_2)
\end{aligned}$$

– γ_μ is monotone: $\forall \mathbf{f}_1^\#, \mathbf{f}_2^\# \in \{D \rightarrow \mathcal{V}^\#, \preceq_{\sqsubseteq^\#}\}$

$$\begin{aligned}
\mathbf{f}_1^\# \preceq_{\sqsubseteq^\#} \mathbf{f}_2^\# &\Leftrightarrow \forall i \in D, \mathbf{f}_1^\#(i) \sqsubseteq^\# \mathbf{f}_2^\#(i) \\
&\Leftrightarrow \forall i \in D, \forall n \in P_\mu(\mathbb{N})|_i, \mathbf{f}_1(n) = \mathbf{f}_1^\#(i) \sqsubseteq^\# \mathbf{f}_2(n) = \mathbf{f}_2^\#(i) \\
&\Leftrightarrow \forall n \in \mathbb{N}, \mathbf{f}_1(n) \sqsubseteq^\# \mathbf{f}_2(n) \Leftrightarrow \mathbf{f}_1 \preceq_{\sqsubseteq^\#} \mathbf{f}_2
\end{aligned}$$

– $\forall \mathbf{f} \in \{\mathbb{N} \rightarrow \mathcal{V}^\#\}$ and $\forall \mathbf{f}^\# \in \{D \rightarrow \mathcal{V}^\#\}$

• $\gamma_\mu(\alpha_\mu(\mathbf{f})) \stackrel{?}{\succeq}_{\sqsubseteq^\#} \mathbf{f}$

$$\begin{aligned}
\gamma_\mu(\alpha_\mu(\mathbf{f})) &\Rightarrow \gamma_\mu \left(\forall i \in D, \bigsqcup_{n \in P_\mu(\mathbb{N})|_i}^\# \mathbf{f}(n) \right) \\
&\Rightarrow \forall j \in \bigsqcup_{i \in D}^\# P_\mu(\mathbb{N})|_i = \mathbb{N}, \mathbf{g}(j) = \bigsqcup_{n \in P_\mu(\mathbb{N})|_i}^\# \mathbf{f}(n) \text{ with } j \sim_\mu i \\
&\Rightarrow \mathbf{g} \succeq_{\sqsubseteq^\#} \mathbf{f}
\end{aligned}$$

• $\alpha_\mu(\gamma_\mu(\mathbf{f}^\#)) \stackrel{?}{\preceq}_{\sqsubseteq^\#} \mathbf{f}^\#$

$$\begin{aligned}
\alpha_\mu(\gamma_\mu(\mathbf{f}^\#)) &\Rightarrow \alpha_\mu \left(\forall j \in \mathbb{N}, \mathbf{f}(j) = \mathbf{f}^\#(i) \text{ with } j \sim_\mu i \right) \\
&\Rightarrow \forall i \in D, \mathbf{g}^\#(i) = \bigsqcup_{n \in P_\mu(\mathbb{N})|_i}^\# \mathbf{f}(n) = \mathbf{f}^\#(i) \\
&\Rightarrow \mathbf{g}^\# \preceq_{\sqsubseteq^\#} \mathbf{f}^\#
\end{aligned}$$