Non-Standard Semantics of Hybrid Systems Modelers[☆]

Albert Benveniste[a,*], Timothy Bourke[a], Benoît Caillaud[a], Marc Pouzet[b]

[a] *INRIA-Rennes, Campus de Beaulieu, 35042 Rennes cedex, France*
[b] *Ecole Normale Supérieure, 45 rue d'Ulm, 75005 Paris, France*

Abstract

Hybrid system modelers have become a corner stone of complex embedded system development. Embedded systems include not only control components and software, but also physical devices. In this area, Simulink is a de facto standard design framework, and Modelica a new player. However, such tools raise several issues related to the lack of reproducibility of simulations (sensitivity to simulation parameters and to the choice of a simulation engine).

In this paper we propose using techniques from *non-standard analysis* to define a semantic domain for hybrid systems. Non-standard analysis is an extension of classical analysis in which infinitesimal (the ε and η in the celebrated generic sentence $\forall\varepsilon\exists\eta\ldots$ of college maths) can be manipulated as first class citizens. This approach allows us to define both a denotational semantics, a constructive semantics, and a Kahn Process Network semantics for hybrid systems, thus establishing simulation engines on a sound but flexible mathematical foundation. These semantics offer a clear distinction between the concerns of the numerical analyst (solving differential equations) and those of the computer scientist (generating execution schemes).

We also discuss a number of practical and fundamental issues in hybrid system modelers that give rise to non reproducibility of results, nondeterminism, and undesirable side effects. Of particular importance are cascaded mode changes (also called "zero-crossings" in the context of hybrid systems modelers).

Foreword: in memory of Amir Pnueli

When we were invited to contribute to this special issue in honor of Amir Pnueli, we first felt deeply honored. Then, we asked ourselves what we could contribute that would best fit the recollection our community has of him. Amir loved opening new avenues, rich in surprises. While he established himself as a leading figure in computer science, he had a profound mathematical background.

[☆] This work was supported by the SYNCHRONICS large scale initiative of INRIA.

[*] Corresponding author

Email addresses: Albert.Benveniste@inria.fr (Albert Benveniste), Timothy.Bourke@inria.fr (Timothy Bourke), Benoit.Caillaud@inria.fr (Benoît Caillaud), Marc.Pouzet@ens.fr (Marc Pouzet)

Finally, Amir is one of the founders of the area of hybrid systems from the perspective of the computer science community. These considerations led us to contribute to this special issue with a new approach to hybrid systems modelers that builds on the heterodox — but, we think, nonetheless useful — mathematical area of non-standard analysis.

1. Introduction

Hybrid system modelers have become in the last two decades the corner stone of complex embedded system development, with embedded systems involving not only control components or software, but also physical devices. Simulink[1] has become the de facto standard for physical system modeling and simulation. Noticeably, by building on top of the success of Simulink, The Mathworks was able to take over the market of embedded systems design, in many industries. This speaks for itself regarding the importance of such tools.

Hybrid system modelers mix discrete time reactive (or dynamical) systems with continuous time ones defined using Ordinary Differential Equations (ODE) or their extensions. In this paper we focus on general modelers, aimed at modeling and simulation of any type of hybrid system and we refer the reader to [14] for an overview of all tools related to hybrid systems modeling and analysis.

Besides Simulink with its state-based extension Stateflow, several such hybrid systems modelers have been developed. Scicos[2] is a free-ware developed by Ramine Nikoukhah at INRIA.

As quoted from the web site of its supporting association, Modelica[3] is a non-proprietary, object-oriented, equation based language to conveniently model complex physical systems containing, e.g., mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents. While[4] Modelica resembles object-oriented programming languages, such as C++ or Java, it differs in two important respects. First, Modelica is a modeling language. Equations do not describe assignment but equality. In Modelica terms, equations have no pre-defined causality. The simulation engine may (and usually must) manipulate the equations symbolically to determine their order of evaluation and which components in the equation are inputs and which are outputs. Said differently, Modelica not only manipulates functions and ODEs, but also equations and Differential Algebraic Equations (DAE) in which variables and their derivatives are involved in constraints. Commercial front-ends for Modelica include Dymola[5] from the Swedish company Dynasim AB (now part of Dassault Systèmes), MathModelica[6] from the Swedish company Math-

[1] http://www.mathworks.com/products/simulink/
[2] http://www-rocq.inria.fr/scicos/
[3] http://www.modelica.org/
[4] The following is quoted from Wikipedia
[5] http://en.wikipedia.org/wiki/Dymola
[6] http://en.wikipedia.org/wiki/MathModelica

Core Engineering AB, SimulationX[7] from the German company ITI GmbH, and MapleSim[8] from the Canadian company Maplesoft. Dassault Systèmes selected Modelica for their product CATIA[9] (CATIA is one of the major CAD systems). The goal of the OpenModelica[10] project is to create a complete Modelica modeling, compilation and simulation environment based on free software distributed in source code form intended for research purposes. The free simulation environment Scicos uses a subset of Modelica for component modeling. Support for a larger part of the Modelica language is currently under development. Recently, Mathworks has issued a similar tool dedicated to physical systems modeling, called Simscape.[11]

Hybrid systems modelers raise a number of difficult issues, both practical and fundamental. Some major practical issues are the following:

(i) Depending on the options selected by the user, simulation results may differ. Of course, simulation results are sensitive to the choice of the integration method — we discuss this unavoidable aspect later. Since simulations use a single, global, solver, the choice and tuning of the integration method is global to the system, which may have strange effects such as undesirable interactions between sub-systems that seemingly should not interact.

(ii) Mode changes occur in the considered hybrid systems by means of *zero-crossings,* which are mode switching boundaries where the dynamics experience a sudden change. The handling of zero-crossings is difficult for two reasons. First, zero-crossings are areas where maximal *stiffness* is encountered and the solvers must be very cautious not to miss them — variable step size integration methods are therefore mandatory. Second, mode changes triggered by zero-crossings can involve a combination of complex operations whose scheduling can be delicate. Indeed, the different simulation engines for Modelica sometimes give different results on identical programs. Of particular difficulty is the handling of *cascades* of zero-crossings, which are successive zero-crossings arising when a mode change leads to a next mode where the guard is immediately violated.

Other issues exist that are more fundamental:

(iii) How discrete is the semantics of the discrete part of a hybrid system modeler? Recall that, in earlier versions, Simulink saw everything as continuous time. In particular, discrete time flows were seen as piecewise constant continuous time signals.

[7] http://en.wikipedia.org/wiki/SimulationX
[8] http://en.wikipedia.org/wiki/MapleSim
[9] http://en.wikipedia.org/wiki/CATIA
[10] http://www.openmodelica.org/
[11] http://www.mathworks.com/products/simscape/

(iv) Physical systems often obey balance equations resulting from applying first principles. Such balance equations are better specified using non-directed systems of DAEs, with no pre-defined input/output roles. This observation goes back to the old work on Bond Graphs [35] as a modeling paradigm for physical systems from first principles, and lead to the development of Modelica. How can compilation techniques adapt to this more constraint oriented style of specification?

Efforts developed in the community of hybrid systems have improved the situation regarding issue (iii), see, e.g., the efforts made in the design of Scicos [13, 32]. Improvements have reduced the previously existing gaps between the results of simulation of a hybrid system and the simulation of its discrete time part in isolation for the purpose of generating code.

Issue (iv) seems solved in practice. However, the discrepancies sometimes observed between different tools for the execution of a same program of the standardized Modelica language reveal that difficulties still remain. Also, a closer investigation reveals that the part of Modelica language to handle mode changes is not compositional, which impairs full compositionality of Modelica as a whole.

Overall, we think that no fundamental answer has been provided to the difficulty raised by the following well justified but nevertheless contradictory requirements that underpin the development of a formally sound execution engine for hybrid systems modelers:

(a) *The semantic function*, mapping a hybrid systems specification to its executable mathematical model (its operational semantics), *should be statically definable*. Such a mapping is indeed the basis for designing formally sound compilation schemes. In particular, this semantic function should not get polluted by assumptions such as "f shall be Lipschitz over $[1, 2]$" or "boolean condition b shall not be Zeno", or "the system shall not be stiff", etc., as the above are typically value-dependent properties that can only be handled properly at run time.

(b) *Computers can only run according to discrete steps*, hence discretizing must be part of defining the semantic map. Indeed, early hybrid systems modelers such as MATRIXx [12] in its 90's generation had made the choice of using fixed step discretization for ODEs in order to achieve a clean combination of continuous time parts and mode changes or discrete time parts. Unfortunately, this design choice contradicted the next (essential) requirement.

(c) To achieve high computational quality with high flexibility, *the discretization scheme must be adaptive, meaning that it is determined at run time.* We recall later the background for this.

[12] http://www.ni.com/matrixx/

We strongly believe that lack of a theory providing adequate answers to the above seemingly contradictory requirements (a)–(c) has been the cause for some of the problems faced by hybrid systems modelers today. Our overall objective is to address these issues properly.

Contribution of the paper. First, we propose a semantics based on *non-standard analysis* after an original idea due to Bliudze and Krob [10]. Roughly speaking, non standard analysis is an extension of classical analysis in which infinitesimals (the ε and η in the celebrated generic sentence $\forall\varepsilon\exists\eta\ldots$ in college maths) can be manipulated as first class citizens. This provides a "synchronous-like" interpretation of the whole system where the base clock is an infinite sequence of infinitesimals — it is both dense and discrete. This interpretation clarifies the treatment of zero-crossings in modelers and provides a firm basis for rejecting or accepting programs.

Second, by building on top of this non-standard semantics, we develop two more semantics. The *constructive semantics* à la Berry [7, 8] allows for a sound definition of compilation schemes. The *Kahn semantics* provides the support for handling cascades of zero-crossings at compile-time and structuring the use of several ODE solvers.

Third, we discuss how to slice a hybrid systems language into its discrete part (for handling by off-the-shelf synchronous language compilers(and its continuous part (for handling by off-the-shelf numerical ODE solvers). Accordingly, hybrid systems appear as conservative extensions of discrete-time synchronous languages.

These contributions are substantiated in SimpleHybrid, a simple formalism that incorporates zero-crossings and parallel composition of discrete computations and ordinary differential equations. It is not intended to be a real language. In particular, it lacks essential features such as function definition and application. We make it minimal to focus on semantical issues. We report experiments with a prototype tool based on the material presented in the paper in Appendix A.

This paper does not address DAEs, supported by Modelica, for example. We restrict ourselves to a "functional" language in static single assignment form corresponding to, e.g., a subset of Simulink or a synchronous Lustre-like [20] language extended with ODEs.

Organization of the paper. In sections 2 to 5 the paper is motivated and the necessary background material is developed. In particular, section 2 presents some example programs that employ zero-crossings in subtle ways; these examples lead us to pose several motivating questions. Sections 3 and 4 present, respectively, background material on numerical integration and non-standard analysis. Section 5 develops the fundamental ideas behind the non-standard semantics and relates them to more standard models.

Sections 6 to 10 present the definition, semantics, and various properties of the SimpleHybrid language. The language is defined in section 6. A non-standard semantics follows in section 7. A complementary constructive seman-

tics is presented in section 8, and applied to the analysis of causality, including Bond Graphs [35, 29]. In section 9, a Kahn semantics is defined and applied to the analysis of zero-crossing cascades and multiple solver interactions. Some practical issues in the compilation of SimpleHybrid programs are outlined in section 10 and Appendix A contains some results from a related prototype implementation.

The final part of the paper comprises section 11, on related work, and section 12, some concluding remarks.

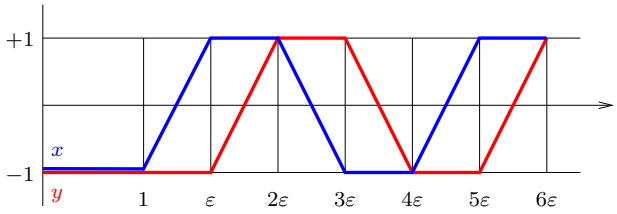
2. Issues raised by zero-crossings

The following examples illustrate some of the inherent subtleties of zero-crossings. In these examples, the resetting mechanisms involve tuples of zero-crossings. For instance, the statement "**reset** $[1,-1]$ **every up**$[x,-x]$" specifies that the signals x and $-x$ are monitored for upward crossings of zero (from <0 to ≥ 0), and further that the signal is reset to 1 when a zero-crossing occurs on x, and to -1 when a zero-crossing occurs on $-x$, with priority to the former if both events occur simultaneously. In Appendix A, examples are expressed in Simulink and a prototype implementation of SimpleHybrid.

Example 1.

$$\begin{array}{lllll} \dot{y} = & 0 \textbf{ init } & -1 \textbf{ reset } [1,-1] & \textbf{every up}[x,-x] \\ \dot{x} = & 0 \textbf{ init } & -1 \textbf{ reset } [-1,1,1] & \textbf{every up}[y,-y,z] \\ \dot{z} = & 1 \textbf{ init } & -1 & \end{array}$$

In this example, during the interval $[0,1]$, x and y remain steady (their slope is 0 with an initial value of -1) while z increases at constant speed 1. At $t=1$, there is a zero-crossing of z, which causes the reset of x to 1, which in turn causes a cascaded zero-crossing of x, which causes the value of y to be reset to 1, which causes a zero-crossing on y; this then causes a second zero-crossing on x, which then causes a second reset of the value of y to 1, and so on, unboundedly. These cascaded zero-crossings occur while time remains stalled at $t=1$. The following figure illustrates this behavior; $\varepsilon > 0$ is a "very small" step size, in that finitely many ε's still sum up to ≈ 0. The linear interpolation is only a convenience to make the diagram more readable.

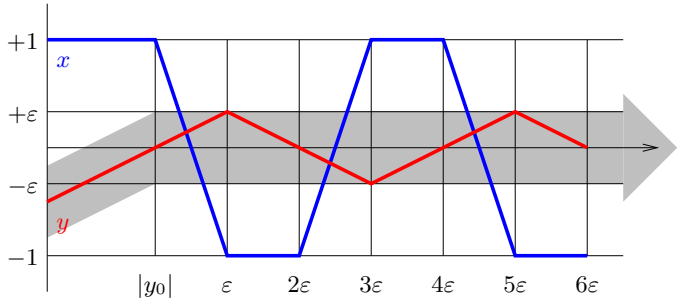


This example is certainly pathological. □

Example 2. In contrast,

$$\dot{x} = 0 \textbf{ init } -\operatorname{sgn}(y_0) \textbf{ reset } [-1, 1] \textbf{ every up}[y, -y]$$
$$\dot{y} = x \textbf{ init } y_0$$

is the simplest case of *sliding mode control* [19, 36]. Suppose $y_0 < 0$, and hence $x_0 > 0$. Then, y increases at constant speed until its first zero-crossing, just after time $t = |y_0|$. From then on, y chatters infinitesimally around 0 as its speed alternates between -1 and $+1$ with infinitesimal steps, as shown below with $y_0 < 0$.



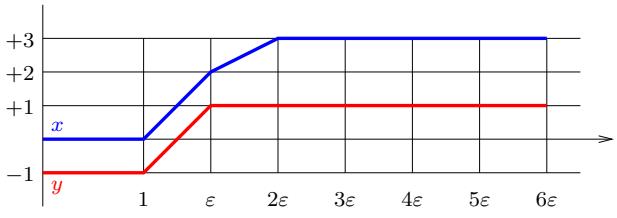
This simple example captures the behavior of systems like ABS in automobile brakes. An adequate interpretation of the behavior of y is *averaging* over time, thus resulting in the mean dynamics $\mathbf{y}$, depicted in the thick shaded dynamics of the figure, where:

$$\dot{\mathbf{y}} = \begin{cases} -\operatorname{sgn}(y_0), & \text{for the interval } [0, |y_0|\,) \\ 0 & \text{for } [\,|y_0|, \infty), \end{cases}$$

Example 3. For our third example, operator **last** (x), where x is a signal, delivers at instant t the left-limit $\lim_{s \nearrow t} x_s$:

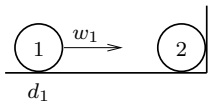
$$\dot{x} = 0 \textbf{ init } \quad 0 \textbf{ reset } [\textbf{last}\,(x) + 1, \textbf{last}\,(x) + 2] \textbf{ every up}[y, z]$$
$$\dot{z} = 1 \textbf{ init } -1$$
$$\dot{y} = 0 \textbf{ init } -1 \textbf{ reset } [1] \textbf{ every up}[z]$$

Signal z has a zero-crossing right after $t = 1$, which causes y to have a cascaded zero-crossing.



We illustrate the behavior of x that results if we consider the cascaded zero-crossings of z and y as successive "micro-steps": x has two successive jumps, of 2 and then 1. z is not shown. Alternatively, one could consider that the two zero-crossings occur simultaneously and then the zero-crossing of y preempts that of z (since y is listed first), which yields a single jump of 1 for x. Which semantics is best? □

Example 4. As a fourth example, we consider the case of two balls colliding next to a wall as shown below.



The figure shows the initial condition $w_1 > 0$ and $d_1 < d_2 = w_2 = 0$, meaning that ball 2 is motionless against the wall, and ball 1 is approaching it at a constant velocity. To simplify, we consider ideal balls of zero diameter. For convenience, the system is activated at initial time $t = -d_1/w_1$, so that the first hit occurs right after $t = 0$. The corresponding equations are:

$$\begin{array}{lcl} \dot{x}_1 & = & v_1 \textbf{ init } d_1 \\ \dot{x}_2 & = & v_2 \textbf{ init } d_2 \\ \dot{v}_1 & = & 0 \textbf{ init } w_1 \textbf{ reset last } (v_2) \textbf{ every up}[x_1 - x_2] \\ \dot{v}_2 & = & 0 \textbf{ init } w_2 \textbf{ reset } [\textbf{last } (v_1) , -\textbf{last } (v_2)] \textbf{ every up}[x_1 - x_2, x_2] \end{array}$$

Ideally, after the collision, ball 2 would still be motionless against the wall and ball 1 would be moving toward the left with velocity $-w_1$. But this state is only reached after a sequence of interactions between both balls and the wall, the details of which are presented in section 5.3. □

The above examples raise a number of issues:

- Can we propose a semantic domain for these examples?
- Can we use it
  - to identify example 1 as pathological, but not example 2?
  - to decide on the semantics of example 3?
  - to give a semantics to example 4?
- More generally, can we develop a semantic domain to serve as a mathematical basis for the management of (possibly cascaded) zero-crossings?

Some of the above questions have been addressed by mathematicians. Sliding mode control, of which example 2 is an instance, has been studied in control science [19, 36]. Similarly, dynamical systems involving subsystems of different time scales have been studied by mathematicians under the name of singular perturbations and addressed using averaging techniques [22]. We seek here, however, techniques based on analyses that compilers can support, not mathematical theories that can only be applied manually.

3. Background on numerical integration of ODEs

Throughout this paper, $\mathbb{N}$ (resp. $\mathbb{N}_+$) is the set of non-negative (resp. positive) integers; $\mathbb{Z}$ is the set of integers; $\mathbb{R}$ is the set of real numbers and $\mathbb{R}_+ = [0, +\infty)$.

To motivate our consideration of non-standard analysis, we find it useful to start with some classical background on the numerical integration of ODEs [12]. Given a continuous function $f : [0, 1] \to \mathbb{R}$, we wish to compute $\int_0^1 f(t)\, dt$. The k-stage quadrature formula is

$$\int_0^1 f(t)\, dt \quad \approx \quad \sum_{i=1}^{k} b_i f(c_i) \tag{1}$$

where c_i are the *knots* and b_i are the *weights*. Now, with $h = (b - a)/N$ and $t_j = a + jh$, we have

$$\begin{aligned} \int_a^b f(t)\, dt = \sum_{j=0}^{N-1} \int_{t_j}^{t_{j+1}} f(t)\, dt \quad &= \quad \sum_{j=0}^{N-1} h \int_0^1 f(t_j + th)\, dt \\ &\approx \quad \sum_{j=1}^{N} h \sum_{i=1}^{k} b_i f(t_j + c_i h) \end{aligned} \tag{2}$$

using (1). The quadrature formula is of *order* $p \in \mathbb{N}_+$ if the equality actually holds in (1) for f a polynomial of degree at most $p - 1$. This implies that, if f is q-times differentiable, then the approximation error in (2) is $h^{\min(p,q)}$. It is known that (1) has order p iff

$$\sum_{i=1}^{k} b_i c_i^{q-1} = 1/q \text{ holds for } 1 \le q \le p. \tag{3}$$

Fixing the knots in (3) yields a Vandermonde linear system with a unique solution for the b_i's, which yields an order $p = k$. So-called *superconvergence* can be reached, however, meaning that $p > k$. Techniques based on orthogonal polynomials allow reaching up to $p = 2k$ (e.g., with Gauss formulas).

Next, consider ODE $\dot{y} = f(t, y)$, $y(t_0) = y_0$ on interval $[t_0, t_0 + h]$, written

$$y(t_0 + h) = y_0 + \int_{t_0}^{t_0+h} f(u, y(u))\, du \tag{4}$$

Runge-Kutta (RK) methods for approximating (4) use the following formulas:

$$\begin{aligned} K_1 &= f(t_0, y_0) \\ K_2 &= f(t_0 + c_2 h, y_0 + h a_{2,1} K_1) \\ K_3 &= f(t_0 + c_3 h, y_0 + h(a_{3,1} K_1 + a_{3,2} K_2)) \\ &\dots \\ K_k &= f(t_0 + c_k h, y_0 + h(a_{k,1} K_1 + \ldots + a_{k,k-1} K_{k-1})) \\ y_1 &= y_0 + h(b_1 K_1 + \ldots + b_k K_k) \;\approx\; y(t_0 + h) \end{aligned}$$

They are obtained by applying quadrature formula (2) when approximating $\int_{t_0}^{t_0+h} f(u, y(u))\, du$ in (4). *Multistep* methods are also possible. They consist in using past values of y before t_0 in computing (4): (4) is replaced by

$$y(t_0 + h) = y_0 + \int_{t_0}^{t_0+h} p_0(u)\, du \tag{5}$$

where p_0 is a polynomial of degree $q - 1$ satisfying

$$p(t_j) = f(t_j, y_j) \text{ for } j = -1, \ldots, -q. \tag{6}$$

Here, $t_{-q} < t_{-q+1} < \ldots < t_{-1} < t_0$ are the previous instants where an approximation $y_{-q}, \ldots, y_0$ has already been computed and therefore the y_j's are known. RK formulas can then be used in solving (5). In all the above approximation methods, the step size h can be *adaptively* selected to satisfy given accuracy requirements by computing nested approximations with different values for h and comparing them when solving an ODE for a specified horizon.

The crux is that the generic form of an approximation formula for (4) is

$$y(t_0 + h) \quad = \quad y_0 + hF(t_0, y_{-q}, \ldots, y_0) \tag{7}$$

where h is small, $q \geq 0$, and $y_{-q}, \ldots, y_0$ are as in (6). In (7), h can vary adaptively, as can F (particularly if multi-step methods are used). The above discussion is also valid for systems of ODEs, i.e., when y takes values in $\mathbb{R}^n$.

The foregoing formulas are used to approximate the evolution of continuous variables over a time interval. In a hybrid system these evolutions may be interrupted by discrete events that result in mode changes or discontinuous jumps in variable values. The instants of occurrence of such events are usually expressed as a zero-crossing in some quantity $g(y)$, where y are variables constrained by ODEs and g yields a real value. As a solver progressively advances the simulation time to approximate integral values, it monitors the sign of the value of g, and if it changes from negative to positive from one approximated instant to the next the solver enters a phase where it searches for the precise instant where g crosses zero. When two (or more) zero-crossings that are being monitored both become very small, a solver will usually take particular care to find the one that first crosses zero as this could determine which mode is subsequently activated.

The slope of g may be used in determining the gap between successive instants and also in the iterative search, which is usually based on a variant of the secant method. That said, it is still sometimes necessary to bound the step size for a particular model to avoid missing zero-crossings where the function being monitored passes through zero an even number of times in rapid succession.

At this point, it should be clear that it is hopeless to include discretization schemes of ODEs or DAEs in our semantics. Which is, in any case, impossible for adaptive schemes where the discretization evolves at run time. We instead propose *non-standard analysis* [34, 17] as a semantic domain. In non-standard analysis, the statement $\dot{y} = x$ means, *by definition of the derivative of a function*:

$$\forall \partial \simeq 0 \quad : \quad \frac{y_{t+\partial} - y_t}{\partial} \simeq x_t \tag{8}$$

where expression "$u \simeq v$" is a *non-standard* expression that reads: *"$v - u$ is infinitesimal"*. The use of the heterodox symbol ∂ in (8) is deliberate. It intends to indicate that the real number referred to is *non-standard*.

4. Background on non-standard analysis

Non-standard analysis was proposed by Abraham Robinson in the 1960s to allow the explicit manipulation of "infinitesimals" in analysis [34, 17, 18]. Robinson's approach is axiomatic; he proposes adding three new axioms to the basic Zermelo-Fraenkel (ZFC) framework. There has been much debate in the mathematical community as to whether it is worth considering non-standard analysis instead of staying with the traditional one. We do not enter this debate. The important thing for us is that non-standard analysis allows the use of the non-standard discretization of continuous dynamics "as if" it was operational.

To our surprise, such an idea is indeed not new. Iwasaki et al. [23] first proposed using non-standard analysis to discuss the nature of time in hybrid systems. Bliudze and Krob [10, 9] have also used non-standard analysis as a mathematical support for defining a system theory for hybrid systems. They discuss in detail the notion of "system" and investigate computability issues. The formalization they propose closely follows that of Turing machines, with a memory tape and a control mechanism.

The introduction to non-standard analysis in [9] is very pleasant and we take the liberty to borrow it. This presentation was originally due to Lindstrøm, see [28]. Its interest is that it does not require any fancy axiomatic material but only makes use of the axiom of choice — actually a weaker form of it. The proposed construction bears some resemblance to the construction of $\mathbb{R}$ as the set of equivalence classes of Cauchy sequences in $\mathbb{Q}$ modulo the equivalence relation $(u_n) \approx (v_n)$ iff $\lim_{n\to\infty}(u_n - v_n) = 0$.

4.1. Motivation and intuitive introduction

We begin with an intuitive introduction to the construction of the non-standard reals. The goal is to augment $\mathbb{R} \cup \{\pm\infty\}$ by adding, to each x in the set, a set of elements that are "infinitesimally close" to it. We will call the resulting set ${}^\star\mathbb{R}$. Another requirement is that all operations and relations defined on $\mathbb{R}$ should extend to ${}^\star\mathbb{R}$.

A first idea is to represent such additional numbers as convergent sequences of reals. For example, elements infinitesimally close to the real number zero are the sequences $u_n = 1/n$, $v_n = 1/\sqrt{n}$ and $w_n = 1/n^2$. Observe that the above three sequences can be ordered: $v_n > u_n > w_n > 0$ where 0 denotes the constant zero sequence. Of course, infinitely large elements (close to $+\infty$) can also be considered, e.g., sequences $x_u = n$, $y_n = \sqrt{n}$, and $z_n = n^2$.

Unfortunately, this way of defining ${}^\star\mathbb{R}$ does not yield a total order since two sequences converging to zero cannot always be compared: if u_n and u'_n are two such sequences, the three sets $\{n \mid u_n > u'_n\}$, $\{n \mid u_n = u'_n\}$, and $\{n \mid u_n < u'_n\}$ may even all be infinitely large. The beautiful idea of Lindstrøm is to enforce

that *exactly one of the above sets is important and the other two can be neglected.* This is achieved by fixing once and for all a finitely additive positive measure μ over the set $\mathbb{N}$ of integers with the following properties:[13]

1. $\mu : 2^{\mathbb{N}} \to \{0, 1\}$;
2. $\mu(X) = 0$ whenever X is finite;
3. $\mu(\mathbb{N}) = 1$.

Now, once μ is fixed, one can compare any two sequences: for the above case, exactly one of the three sets must have μ-measure 1 and the others must have μ-measure 0. Thus, say that $u > u', u = u'$, or $u < u'$, if $\mu(\{n \mid u_n > u'_n\} = 1)$, $\mu(\{n \mid u_n = u'_n\}) = 1$, or $\mu(\{n \mid u_n < u'_n\}) = 1$, respectively. Indeed, the same trick works for many other relations and operations on non-standard real numbers, as we shall see. We now proceed with a more formal presentation.

4.2. Construction of non-standard domains

For I an arbitrary set, a *filter* $\mathcal{F}$ over I is a family of subsets of I such that:

1. the empty set does not belong to $\mathcal{F}$,
2. $P, Q \in \mathcal{F}$ implies $P \cap Q \in \mathcal{F}$, and
3. $P \in \mathcal{F}$ and $P \subset Q \subseteq I$ implies $Q \in \mathcal{F}$.

Consequently, $\mathcal{F}$ cannot contain both a set P and its complement P^c. A filter that contains one of the two for any subset $P \subseteq I$ is called an *ultra-filter*. At this point we recall Zorn's lemma, known to be equivalent to the axiom of choice:

Lemma 1 (Zorn's lemma). *Any partially ordered set $(X, \leq)$ such that any chain in X possesses an upper bound has a maximal element.*

A filter $\mathcal{F}$ over I is an ultra-filter if and only if it is maximal with respect to set inclusion. By Zorn's lemma, any filter $\mathcal{F}$ over I can be extended to an ultra-filter over I. Now, if I is infinite, the family of sets $\mathcal{F} = \{P \subseteq I \mid P^c \text{ is finite}\}$ is a *free* filter, meaning it contains no finite set. It can thus be extended to a free ultra-filter over I:

Lemma 2. *Any infinite set has a free ultra-filter.*

Every free ultra-filter $\mathcal{F}$ over I uniquely defines, by setting $\mu(P) = 1$ if $P \in \mathcal{F}$ and otherwise 0, a finitely additive measure[14] $\mu : 2^I \mapsto \{0, 1\}$, which satisfies

$$\mu(I) = 1 \text{ and, if } P \text{ is finite, then } \mu(P) = 0.$$

Now, fix an infinite set I and a finitely additive measure μ over I as above. Let $\mathbb{X}$ be a set and consider the Cartesian product $\mathbb{X}^I = (x_i)_{i \in I}$. Define $(x_i) \approx (x'_i)$ iff

[13] The existence of such a measure is non trivial and is explained later.

[14] Observe that, as a consequence, μ cannot be sigma-additive (in contrast to probability measures or Radon measures) in that it is *not* true that $\mu(\bigcup_n A_n) = \sum_n \mu(A_n)$ holds for an infinite denumerable sequence A_n of pairwise disjoint subsets of $\mathbb{N}$.

$\mu\{i \in I \mid x_i \neq x'_i\} = 0$. Relation $\approx$ is an equivalence relation whose equivalence classes are denoted by $[x_i]$ and we define

$$^\star\mathbb{X} = \mathbb{X}^I / \approx \tag{9}$$

$\mathbb{X}$ is naturally embedded into $^\star\mathbb{X}$ by mapping every $x \in \mathbb{X}$ to the constant tuple such that $x_i = x$ for every $i \in I$. Any algebraic structure over $\mathbb{X}$ (group, ring, field) carries over to $^\star\mathbb{X}$ by almost point-wise extension. In particular, if $[x_i] \neq 0$, meaning that $\mu\{i \mid x_i = 0\} = 0$ we can define its inverse $[x_i]^{-1}$ by taking $y_i = x_i^{-1}$ if $x_i \neq 0$ and $y_i = 0$ otherwise. This construction yields $\mu\{i \mid y_i x_i = 1\} = 1$, whence $[y_i][x_i] = 1$ in $^\star\mathbb{X}$. The existence of an inverse for any non-zero element of a ring is indeed stated by the formula: $\forall x\,(x = 0 \vee \exists y\,(xy = 1))$. More generally:

Lemma 3 (Transfer Principle). *Every first order formula is true over $^\star\mathbb{X}$ iff it is true over $\mathbb{X}$.*

4.3. Non-standard reals and integers

The above general construction can simply be applied to $\mathbb{X} = \mathbb{R}$ and $I = \mathbb{N}$. The result is denoted $^\star\mathbb{R}$; it is a field according to the transfer principle. By the same principle, $^\star\mathbb{R}$ is totally ordered by $[u_n] \leq [v_n]$ iff $\mu\{n \mid v_n > u_n\} = 0$. We claim that, for any finite $[x_n] \in {}^\star\mathbb{R}$, there exists a unique $st([x_n])$, call it the *standard part* of $[x_n]$, such that

$$st([x_n]) \in \mathbb{R} \quad \text{and} \quad st([x_n]) \approx [x_n]\,. \tag{10}$$

To prove this, let $x = \sup\{u \in \mathbb{R} \mid [u] \leq [x_n]\}$, where $[u]$ denotes the constant sequence equal to u. Since $[x_n]$ is finite, x exists and we only need to show that $[x_n] - x$ is infinitesimal. If not, then there exists $y \in \mathbb{R}, y > 0$ such that $y < |x - [x_n]|$, that is, either $x < [x_n] - [y]$ or $x > [x_n] + [y]$, which both contradict the definition of x. The uniqueness of x is clear, thus we can define $st([x_n]) = x$. Infinite non-standard reals have no standard part in $\mathbb{R}$.

It is also of interest to apply the general construction (9) to $\mathbb{X} = I = \mathbb{N}$, which results in the set $^\star\mathbb{N}$ of *non-standard natural numbers*. The non-standard set $^\star\mathbb{N}$ differs from $\mathbb{N}$ by the addition of *infinite natural numbers*, which are equivalence classes of sequences of integers whose essential limit is $+\infty$.

4.4. Integrals and differential equations: the standardization principle

Any sequence (g_n) of functions $g_n : \mathbb{R} \mapsto \mathbb{R}$ point-wise defines a function $[g_n] : {}^\star\mathbb{R} \mapsto {}^\star\mathbb{R}$ by setting

$$[g_n]([x_n]) = [g_n(x_n)] \tag{11}$$

A function $^\star\mathbb{R} \to {}^\star\mathbb{R}$ so obtained is called *internal*. Properties of and operations on ordinary functions extend point-wise to internal functions of $^\star\mathbb{R} \to {}^\star\mathbb{R}$. The *non-standard version* of $g : \mathbb{R} \to \mathbb{R}$ is the internal function $^\star g = [g, g, g, \ldots]$.

The same notions apply to sets. An internal set $A = [A_n]$ is called *hyperfinite* if $\mu\{n \mid A_n \text{ finite}\} = 1$; the *cardinal* $|A|$ of A is defined as $[|A_n|]$.

Now, consider an infinite number $N \in {}^\star\mathbb{N}$ and the set

$$T = \left\{ 0, \frac{1}{N}, \frac{2}{N}, \frac{3}{N}, \dots \frac{N-1}{N}, 1 \right\} \tag{12}$$

By definition, if $N = [N_n]$, then $T = [T_n]$ with

$$T_n = \left\{ 0, \frac{1}{N_n}, \frac{2}{N_n}, \frac{3}{N_n}, \dots \frac{N_n - 1}{N_n}, 1 \right\}$$

hence $|T| = [|T_n|] = [N_n + 1] = N + 1$. Now, consider an internal function $g = [g_n]$ and a hyperfinite set $A = [A_n]$. The *sum* of g over A can be defined:

$$\sum_{a \in A} g(a) \;=_{\text{def}}\; \left[\sum_{a \in A_n} g_n(a) \right]$$

If t is as above, and $f : \mathbb{R} \to \mathbb{R}$ is a standard function, we obtain

$$\sum_{t \in T} \frac{1}{N} {}^\star f(t) = \left[\sum_{t \in T_n} \frac{1}{N_n} f(t_n) \right] \tag{13}$$

Now, f continuous implies $\sum_{t \in T_n} \frac{1}{N_n} f(t_n) \to \int_0^1 f(t)\, dt$, so,

$$\int_0^1 f(t)\, dt = st \left(\sum_{t \in T} \frac{1}{N} {}^\star f(t) \right) \tag{14}$$

Under the same assumptions, for any $t \in [0, 1]$,

$$\int_0^t f(u)\, du = st \left(\sum_{u \in T, u \le t} \frac{1}{N} {}^\star f(t) \right) \tag{15}$$

Now, consider the following ODE:

$$\dot{x} = f(x, t), \quad x(0) = x_0 \tag{16}$$

Assume (16) possesses a solution $[0, 1] \ni t \mapsto x(t)$ such that the function $t \mapsto f(x(t), t)$ is continuous. Rewriting (16) in its equivalent integral form $x(t) = x_0 + \int_0^t f(x(u), u)\, du$ and using (15) yields

$$x(t) = st \left(x_0 + \sum_{u \in T, u \le t} \frac{1}{N} {}^\star f(x(u), u) \right) \tag{17}$$

The substitution in (17) of $\partial = 1/N$, which is positive and infinitesimal, yields $T = \{t_n = n\partial \mid n = 0, \dots, N\}$. The expression in parentheses on the right hand

side of (17) is the piecewise-constant right-continuous function ${}^\star x(t), t \in [0,1]$ such that, for $n = 1, \ldots, N$:

$$\begin{array}{rcl} {}^\star x(t_n) & = & {}^\star x(t_{n-1}) + \partial \times {}^\star f({}^\star x(t_{n-1}), t_{n-1}) \\ {}^\star x(t_0) & = & x_0 \end{array} \tag{18}$$

By (17), the solutions x, of ODE (16), and ${}^\star x$, as computed by algorithm (18), are related by $x = st({}^\star x)$. Formula (18) can be seen as a *non-standard operational semantics* for ODE (16); one which depends on the choice of infinitesimal step parameter ∂. Property (17), though, expresses the idea that all these non-standard semantics are equivalent from the standard viewpoint regardless of the choice made for ∂. This fact is referred to as the *standardization principle*.

5. Non-standard and standard semantics

The standardization principle will be further developed in this section to widen the class of hybrid systems for which a semantics (in the standard, usual, sense) can be given. We build on the seminal paper [1] and we study the class of hybrid systems defined there.

5.1. The Standardization Principle for Hybrid Systems

We first recall the *super-dense time* semantics of [30, 26, 27] for hybrid systems defined in [1]. Then, we give a non-standard semantics for hybrid systems and formulate the associated standardization principle. Following [1], a *hybrid system* is a tuple

$$\mathcal{H} \quad = \quad (Loc, Var, Edg, Act, Inv, Ini) \tag{19}$$

where

- Loc is a finite set of *locations*; whose representatives are denoted by ℓ;
- Var is a finite set of *variables*. A *valuation* v assigns, for each variable $x \in Var$, a real value $v(x) \in \mathbb{R}$. V denotes the set of valuations. A *state* is a pair $\sigma = (\ell, v)$;
- Edg is a finite set of *transitions* $e = (\ell, F, \ell')$ where $F \subseteq V \times V$.
- Act, the *continuous dynamics*, assigns to each location ℓ a set of ODEs over variables in Var; let O_ℓ denote the set of ODEs associated with location $\ell \in Loc$.[15] O_ℓ has the form $\dot{X} = f_\ell(s, X)$ where X is a vector containing all variables and s is the time index;
- Inv assigns to each location ℓ an *invariant* $G_\ell \subseteq V$;
- $Ini = (\ell_0, v_0) \in Loc \times V$ is the *initial condition*.

[15] In [1], so-called *activities* are directly specified as trajectories; i.e., in our model, as the solution of a set of ODEs.

5.1.1. Super-dense time (standard) semantics

A super-dense time semantics was proposed by [30, 26, 27] for a hybrid system $\mathcal{H}$ involving cascaded transitions. We recall it here for the sake of completeness. The *super-dense time (standard) semantics* of a hybrid system $\mathcal{H}$ is defined as follows. The time index set is

$$\mathbb{S} \;=\; \mathbb{R}_+ \times \mathbb{N}$$

equipped with the lexicographic order: $(s,m) < (t,n)$ if either $s < t$ or $s = t$ and $m < n$. A *timeline* is a function $N : \mathbb{R}_+ \mapsto \mathbb{N}$. $N(s)$ indicates the number of additional instants that occur at a real date s, and each such timeline thus specifies a subset of super-dense time $\mathbb{S}_N = \{(s,n) \in \mathbb{S} \,|\, n \leq N(s)\}$. In particular, if N is the constant 0, then $\mathbb{S}_N$ is isomorphic to $\mathbb{R}_+$. A *run* of $\mathcal{H}$ is a finite or infinite sequence

$$\begin{array}{rcl} \rho & : & \sigma_{0,0} \mapsto_{O_0}^{(s_1,1)} \sigma_{1,1} \mapsto^{(s_1,2)} \sigma_{1,2} \ldots \mapsto^{(s_1,N_1)} \sigma_{1,N_1} \\ & & \quad \mapsto_{O_1}^{(s_2,1)} \sigma_{2,1} \mapsto^{(s_2,2)} \sigma_{2,2} \ldots \mapsto^{(s_2,N_2)} \sigma_{2,N_2} \ldots \end{array} \tag{20}$$

of states $\sigma_i = (\ell_i, v_i)$, non-negative reals $0 < s_1 < s_2 < \ldots$, positive integers N_i, and sets of ODEs O_i, such that

1. $O_0 = O_{\ell_0}$ is defined over the interval $(0, s_1]$ and v_0 is its initial condition;
2. $O_i = O_{\ell_i}$ is defined over the interval $(s_i, s_{i+1}]$, has v_i as its initial condition and possesses a solution that satisfies the invariant $G_i = G_{\ell_i}$; let v_i' be the valuation of the solution of O_i at time s_{i+1};
3. the state $\sigma_{i+1,0}$ is an *Edg*-successor of the state $\sigma_i' = (\ell_i, v_i')$; furthermore, when $N_i > 1$ and for $0 < n < N_i$, the state $\sigma_{i,n+1}$ is an *Edg*-successor of the state $\sigma_{i,n}$.

Observe that run ρ defined in (20) has timeline N equal to $N(s_i) = N_i$ and otherwise $N(s) = 0$.

5.1.2. Non-standard semantics

Fix a time base $\partial \in {}^\star\mathbb{R}, \partial > 0, \partial \approx 0$ and define the time index set

$$\mathbb{T}_\partial \quad = \quad \{t_n = n\partial \mid n \in {}^\star\mathbb{N}\}$$

The non-standard semantics of hybrid system $\mathcal{H}$ uses $\mathbb{T}_\partial$ as its time set and is defined as follows. A *non-standard run* of $\mathcal{H}$ is a finite or infinite non-standard sequence:

$${}^\star\rho(\partial) \quad : \quad {}^\star\sigma_0 \;\mapsto_\partial\; {}^\star\sigma_1 \;\ldots\; {}^\star\sigma_n \;\mapsto_\partial\; {}^\star\sigma_{n+1} \;\ldots \tag{21}$$

of states ${}^\star\sigma_n = (\ell_n, {}^\star v_n)$ for $n \in {}^\star\mathbb{N}$, such that one of two cases apply:

1. *ODE micro-step:* ${}^\star v_n \in G_{\ell_n}$ holds, which then implies $\ell_{n+1} = \ell_n$ and ${}^\star v_{n+1} = {}^\star v_n + \partial \times f_{\ell_n}(t_n, {}^\star v_n)$;

2. *location change micro-step:* ${}^\star v_n \notin G_{\ell_n}$ holds, which then implies that state ${}^\star\sigma_{n+1}$ is an *Edg*-successor of state ${}^\star\sigma_n$.

To highlight the dependence of ${}^\star\rho(\partial)$ on the time base ∂, we denote its successive states as ${}^\star\sigma_n(\partial) = (\ell_n(\partial), {}^\star v_n(\partial))$. Finally, we set

$$\forall t \in {}^\star\mathbb{R}_+ : {}^\star v_t(\partial) = {}^\star v_n(\partial) \text{ where } n = \min\{m \in {}^\star\mathbb{N} \mid t \leq t_m\}$$

Observe that we could give a non-standard semantics to a hybrid system whose dynamics, invariants, and edges, are defined using non-standard functions. We will not, however, develop the idea here; in the sequel, all hybrid systems we consider are defined in standard terms.

5.1.3. The Standardization Principle

The following theorem plays an essential role in defining the class of hybrid systems that can be given a semantics in the usual (standard) sense:

Theorem 1 (Standardisation Principle). *Assume the following properties:*

1. *for any location ℓ, the continuous dynamics O_ℓ have a unique solution v such that $s \mapsto f_\ell(v(s), s)$ is continuous while v satisfies G_ℓ;*
2. *when activated in any location ℓ, the continuous dynamics will continue to satisfy G_ℓ for some positive non-infinitesimal duration;*
3. *the transition relations F, arising in Edg, and guards G_ℓ are continuous;*[16]
4. *there are only finitely many successive cascaded location changes.*

Let (t_m) be the finite or infinite sequence of instants of zero-crossing, and set $t_\infty =_{\text{def}} +\infty$ if (t_m) is a finite sequence, and otherwise $t_\infty =_{\text{def}} \lim_{m\nearrow\infty} t_m \leq +\infty$. A hybrid system $\mathcal{H}$ possesses a unique (standard) solution v over $[0, t_\infty)$ such that

$$v \quad = \quad st({}^\star v(\partial)) \tag{22}$$

for any choice of time base ∂ in the non-standard semantics of $\mathcal{H}$.

Comments. Theorem 1 expresses that, under conditions *1–4*, the non-standard ∂-semantics is intrinsic in that its standardization does not depend on the choice of discretization step ∂.

Conditions 1–3 are smoothness conditions. Condition 1 requires a smooth continuous-time solution for the dynamics of each location — we used exactly the same condition while deriving the standardization principle (17) for ODEs. Condition 2 precludes the situation where an invariant is satisfied when a location is entered, only to be violated immediately when the ODE starts. Condition 3 prevents the reset valuation from having a discontinuity exactly at the instant of location change — this is trivially satisfied if, for example, reset values

[16] Formally, $x \mapsto \{y \mid F(x, y)\}$ is continuous and G_ℓ is closed.

are constant in each location. Sufficient conditions for Conditions 1–3 to hold are known but are typically beyond the reach of compilers.

In contrast, Condition 4 is interesting as sufficient conditions for it can be statically checked by a compiler, as shown in section 9.2.1. Additionally, Condition 4 only involves the discrete parts of a hybrid system; see sections 6 and 10.

Finally, note that defining the semantics over the whole of $\mathbb{R}_+$ using Theorem 1 requires that $t_\infty = +\infty$; a non-Zenoness condition. □

Proof. The proof is a rather technical extension of the proof of (17). We nevertheless provide the details for the sake of completeness. To simplify the proof, we assume that transition relation *Edg* gives raise to a deterministic function for the next location and valuation; this simplifying assumption prevents us from dealing with the non-determinism that would otherwise result and that would complicate the proof in a technical but unimportant way. The reasoning proceeds by induction on successive groups of cascaded zero-crossings.

Consider an instant $t(\partial) \in {}^\star\mathbb{R}_+$ that is at the head of a cascade of location changes for the non-standard semantics ${}^\star v(\partial)$, meaning that it does not directly follow any other location change. If no such instant exists, then the theorem is trivial. Hence, we focus on the other case. Our induction hypothesis consists of the following two conditions:

(H1) $t \ =_{\text{def}}\ st(t(\partial))$, which belongs to $\mathbb{R}_+$, does not depend on ∂;

(H2) The conclusion of Theorem 1 holds on $[0, t]$.

Let

$$\begin{array}{lcll} t(\partial) & , & t(\partial)+\partial & , \ldots , \quad t(\partial)+(n-1)\partial \\ \ell_1(\partial) & , & \ell_2(\partial) & , \ldots , \quad \ell_n(\partial) \\ v_{\ell_1(\partial)}\big(t(\partial)+\partial\big) & , & v_{\ell_2(\partial)}\big(t(\partial)+2\partial\big) & , \ldots , \quad v_{\ell_n(\partial)}\big(t(\partial)+n\partial\big) \end{array}$$

be the finite cascade of n successive instants of zero-crossing starting from $t(\partial)$, the corresponding successive visited locations, and the corresponding reset values. Using (H1),

$$\text{map } t(\partial), t(\partial)+\partial, \ldots, t(\partial)+n\partial \ \text{ to } \ (t,0),(t,1),\ldots,(t,n) \in \mathbb{R}_+ \times \mathbb{N}. \qquad (23)$$

Applying hypothesis (H2) to the guard and using Condition 3 proves that

$$\ell_0 \ =_{\text{def}}\ \ell(t(\partial)-\partial) \text{ does not depend on } \partial.$$

Next, using the definition (9) of non-standard reals as sequences of reals modulo $\approx$, write $t(\partial) = [t_n]$ and $\partial = [\delta_n]$. By (H1), $\lim_{k\to\infty} t_{n_k} = t$, where $\mathbb{N} - \{n_k \mid k \in \mathbb{N}\}$ is negligible, i.e., has μ-measure 0. Since ∂ is infinitesimal, $\lim_{j\to\infty} \delta_{n_j} = 0$, where $\mathbb{N} - \{n_j \mid j \in \mathbb{N}\}$ has μ-measure 0. Since μ is additive, the union of the above two subsets of $\mathbb{N}$ again has μ-measure 0. Reordering this set as a sequence $n_i, i \in \mathbb{N}$, we get $\lim_{i\to\infty} t_{n_i} = t$, $\lim_{i\to\infty} \delta_{n_i} = 0$, and

$\mu(\mathbb{N} - \{n_i \mid i \in \mathbb{N}\}) = 0$. Using hypothesis (H2) regarding F and condition 3 of Theorem 1, we obtain that

$$st(v_{\ell_1}(t(\partial) + \partial)) = \lim_{i \to \infty} v_{\ell_1}(t_{n_i} + \delta_{n_i}) = v_{\ell_1}(t) \text{ does not depend on } \partial.$$

Thus we can define, for $(t, 1)$ as in (23), $v(t, 1) = st(v_{\ell_1}(t(\partial) + \partial)) = v_{\ell_1}(t)$. Applying this reasoning inductively allows the definition of successive reset values

$$v(t, k) \quad = \quad st(v_{\ell_k}(t(\partial) + k\partial)) = v_{\ell_k}(t), \text{ for } k = 1, \ldots, n \tag{24}$$

Using Condition 2, we can then start the ODE in the final location ℓ_n and apply the reasoning proving (17) until the next location change. This extends the induction hypothesis and proves the theorem. □

Theorem 1 can be further refined by weakening its assumptions. Referring to the non-standard semantics of $\mathcal{H}$, call *infinitesimal* a micro-step that is an ODE step where neither ${}^\star v(t)$ nor ${}^\star v(t + \partial)$ violate the guard. Other micro-steps are called *non-infinitesimal*. Accordingly, an ODE step in which ${}^\star v(t)$ is the reset value resulting from a location change and the guard is violated by ${}^\star v(t + \partial)$ is non-infinitesimal. Non-infinitesimal micro-steps must be either location changes or ODE steps as above, in which the violation of the guard is immediate, i.e., occurs within one ∂-step.

Theorem 2 (Standardisation Principle, refined). *The conclusion of Theorem 1 still holds under the following weakened conditions:*

1. *for any location ℓ, the continuous dynamics O_ℓ have a unique solution v such that $s \mapsto f_\ell(v(s), s)$ is continuous while v satisfies G_ℓ;* [unchanged]
2. [suppressed]
3. *the transition relations F, arising in Edg, and guards G_ℓ are continuous;* [unchanged]
4. *successive non-infinitesimal micro-steps are always finitely many.* [new]

The suppression of Condition 2 is critical. It allows the treatment of situations like that of the colliding balls example (4), which are not addressed by Theorem 1. The proof of Theorem 2 is essentially the same as that of Theorem 1 but with a change to the last paragraph of the proof of the latter, where Condition 2 is invoked. After (24), we must distinguish two cases:

(i) After the last location change of the cascade, the system can run the ODE for a positive period of time; for this case, the proof terminates as in Theorem 1.

(ii) After the last location change of the cascade, the system performs one more non-infinitesimal micro-step by reaching a location change right after starting the ODE. Since only finitely many such micro-steps can occur by the new Condition 4, we are then back to case (i) after the cascade.

The reason for still considering Theorem 1 is that its Condition 4 can be checked at compile time, whereas Condition 4 of Theorem 2 can only be checked at run-time, since it requires checking that an active ODE violates the guard immediately upon starting.

Theorems 1 and 2 widen the class of hybrid systems for which the existence and uniqueness of a semantics can be statically checked by a compiler. A larger class of systems can probably be given a semantics through our approach, as evidenced by the analysis of sliding mode example 2 in section 5.3. Providing a systematic study of this type of system is left for future research.

5.2. Non-Standard Analysis as a semantic domain for hybrid systems

Applying non-standard analysis to define a semantic domain for hybrid systems has several advantages.

1. The time set $\mathbb{T}$ is both *discrete* — since each instant has unique previous and next instants, see formula (18) and section 7 — and *dense* in $\mathbb{R}$.
2. Since $\mathbb{T}$ is discrete, we can specify dynamical systems over $\mathbb{T}$ in full generality, without needing to refer to any kind of smoothness condition. Formula (18) is one such instance. As a result, the non-standard semantics presented in section 7 is simple and elegant.
3. Having a simple operational semantics allows us to develop a comprehensive *constructive semantics* for hybrid systems in their full generality. Constructive semantics is a mathematical basis from which sound execution schemes can be derived.
4. The problem with the smoothness condition does not miraculously disappear. But it is postponed to run time, thanks to Theorem 1 (Standardisation Principle): if, in each state of the hybrid system under consideration, the continuous dynamics have a unique solution in the usual mathematical sense, then the Standardisation of our operational semantics computes it.

Observe that the generation of execution schemes only depends on items 1–3, and not on item 4, which is related rather to issues of variable-step discretization. In other words, our approach separates the concerns of the computer scientist (defining a sound operational semantics and related execution schemes), from those of the numerical analyst (properly configuring numerical solvers).

5.3. Back to the examples

Figures in examples 1–3 plot the non-standard ∂-semantics of examples (1)–(3) using $\partial = \varepsilon$. We now discuss these examples in detail together with example 4.

Example 1. The mysterious behavior of example 1 can now be clarified: the first zero-crossing occurs at time $t = 1+\varepsilon$ (corresponding to 1_+ of Example 1). Then, zero-crossings occur repeatedly and forever with a period of 4ε, thus filling the timeline until $+\infty$. The non-standard time domain $\mathbb{T} = \{n\varepsilon \mid n \in {}^\star\mathbb{N}\}$ permits several successive zero-crossings each of zero duration, with time still diverging

eventually, since we can always find n infinitely large enough so that $n\varepsilon > t$ for any $t \in \mathbb{R}_+$. The key feature of example 1 is that, despite being well defined within a non-standard analysis framework, there is no possible standardization. In fact, this example does not satisfy condition 4 of Theorems 1 or 2.

Example 2. In contrast, consider example 2. We claim that the standardization $\mathbf{y} = st(y)$ exists and has the *averaged* dynamics given just before (3). To show this, we use a variation of the argument developed in analyzing formulas (12)–(14), see sections 4.3 and 4.4. Let (x, y) be the non-standard semantics of (2), i.e., given by Example 2. Again, let ε_n be the sequence of positive (standard) reals converging to 0 such that $\varepsilon = [\varepsilon_n]$. Consider the following sequence of (standard) dynamical systems y^n

$$\begin{aligned} \dot{x}^n &= \ 0 \ \textbf{init} \ -\operatorname{sgn}(y_0) \ \textbf{reset}\,[-1,1] \ \textbf{every up}[y^n - \varepsilon_n, -y^n + \varepsilon_n] \\ \dot{y}^n &= \ x^n \ \textbf{init} \ y_0 \end{aligned} \tag{25}$$

The behavior of y^n can again be seen in Example 2, with, however, ε_n substituted for ε. For any $k \in {}^\star\mathbb{N}$, we have $k\varepsilon = [k\varepsilon_n]$ and thus, since x alternates between -1 and $+1$ at multiples of ε (see Example 2), it follows that $x(k\varepsilon) = [x^n(k\varepsilon_n)]$, expressing that $x = [x^n]$, see (11). The same reasoning shows that $y = [y^n]$. On the other hand, using elementary arguments from standard analysis, (25) defines a sequence of functions $y^n_t, t \geq 0$ that converge uniformly to $\mathbf{y}$ defined just before (3) when $n \nearrow +\infty$.[17] The above analysis shows that $\mathbf{y} = st(y)$ where y is given by (2) and $\mathbf{y}$ is given just before (3). Still, this example is not easy in that it is neither covered by Theorem 1 nor by Theorem 2.

Example 3. This example satisfies conditions 1–4 of Theorem 1, which provides it with a standard semantics using super-dense time [26, 27].

Example 4. In the ∂-non-standard semantics, the colliding balls example behaves as follows:

1. At $t = \partial$, $x_1 = \partial \cdot w_1 > 0$, which causes a zero-crossing on $x_1 - x_2$.
2. As a result, at $t = 2\partial$ the balls exchange velocities: $v_1 = 0$ and $v_2 = w_1$.
3. At $t = 3\partial$, $x_1 = 2\partial \cdot w_1$ and $x_2 = \partial \cdot w_1$, and there is thus a zero-crossing on x_2. Observe that this zero-crossing is immediate; there is no "standard time" in which the ODEs can evolve.
4. Hence at $t = 4\partial$, $x_1 = x_2 = 2\partial \cdot w_1$, $v_1 = 0$ and $v_2 = -w_1$.
5. At $t = 5\partial$, $x_1 = 2\partial \cdot w_1$ and $x_2 = \partial \cdot w_1$, and $x_1 - x_2$ crosses zero.
6. Hence at at $t = 6\partial$, $x_1 = 2\partial \cdot w_1$, $x_2 = 0$, $v_1 = -w_1$ and $v_2 = 0$.

Then, ball 1 moves toward $-\infty$ according to the ODEs and no further zero-crossings occur. Due to the above step 3, Condition 1 of Theorem 1 is violated

[17] This is the part of the argument that cannot be invoked for example 1.

and thus Theorem 1 is not sufficient to give a semantics to the colliding balls example. Indeed, the successive steps 1–6 do not constitute a single cascade of zero-crossings, but rather of two successive cascades separated by an ODE phase of infinitesimal length, a situation not covered by Theorem 1, but covered by Theorem 2, provided that we prove that only finitely many non-infinitesimal micro-steps can occur. Executing the non-standard semantics symbolically indeed reveals that only steps 1–6 are non-infinitesimal.

6. The SimpleHybrid Formalism

In this section we develop a minimalist language for hybrid systems called SimpleHybrid which is designed primarily to facilitate mathematical manipulations and focus on the semantics of hybrid systems. This formalism has some essential features of a language — a small set of primitive entities and statements plus a composition operator — but lacks essential features such as function definition and application (i.e., modularity). This is addressed in paper [5]. The statements of SimpleHybrid are equations of the form:

$$\begin{array}{ll} Eq_1: & y = f([x]) \\ Eq_2: & y = \textbf{last}\,(x) \\ Eq_3: & \zeta = \textbf{up}(x) \\ Eq_4: & \dot{y} = x\ \textbf{init}\ y_0\ \textbf{reset}\ z \\ Eq_5: & y = [x]\ \textbf{every}\ [\zeta]\ \ \textbf{init}\ y_0 \\ Eq_6: & y = \textbf{pre}\,(x)\ \ \textbf{init}\ y_0 \end{array} \tag{26}$$

Formally, SimpleHybrid consists of the following program kernel:

$$\begin{array}{rcl} Eq & ::= & Eq_1 \mid Eq_2 \mid Eq_3 \mid Eq_4 \mid Eq_5 \mid Eq_6 \\ S & ::= & Eq \ \mid S \parallel S \end{array} \tag{27}$$

A system S is an equation of the form Eq_1–Eq_6 or a parallel composition of systems. It resembles a Static Single Assignment (SSA) form with intermediate values stored in variables; a fact which simplifies subsequent mathematical developments.

Symbols $x, y, z, u, \ldots$ denote *variables*, taken from an underlying set $\mathcal{X}$ of variables, and having respective domains D_x, D_y, etc. $[x] = [x^1, \ldots, x^n]$ is a tuple of variables. Symbols x_0, y_1, etc. denote immediate values (e.g., 42, 1.5). A dotted variable $\dot{y}$ denotes a derivative. The symbol ζ denotes a *zero-crossing* variable taken from a set $\mathcal{T} \subset \mathcal{X}$ of clock variables (generically denoted by the symbol τ). Clock variables take their values from the set of all *clocks*, where a clock is any subset of $\mathbb{R}_+$. Equations Eq_1–Eq_6 define dynamical systems, or, equivalently, sets of behaviors with time index set $\mathbb{R}_+ = [0, +\infty)$. For example, an equation $y = x$ (form Eq_1, taking f as the identity function) means $\forall t \in \mathbb{R}_+ : y_t = x_t$. Hybrid systems are specified via sets of equations of the form Eq_1–Eq_6, taken conjunctively.

Well formation rules. Any system S made by composing equations Eq_1—Eq_6 must verify the following constraints:

1. An equation Eq_2 ($y = \mathbf{last}\,(x)$) is well formed if the variable x is defined by an equation of the form Eq_4 or Eq_5.
2. An equation Eq_4 ($\dot{y} = x$ **init** v_0 **reset** z) is well formed if the variable z is defined by an equation of the form Eq_5.
3. An equation Eq_6 ($y = \mathbf{pre}\,(x)$ **init** v_0) is well formed if the variable x is defined by an equation of the form Eq_5.

In the following we give an informal explanation of the language primitives, without making explicit the necessary continuity and smoothness assumptions for them to make sense. A more precise mathematical semantics will be given in the next section.

We identify any clock τ with the boolean predicate it defines (the same convention also applies to zero-crossings):

$$\tau_t \quad = \quad \text{if } t \in \tau \text{ then } \textsc{t} \text{ else } \textsc{f} \tag{28}$$

For $X \subseteq \mathcal{X}$ finite, a *state* over X is an element $s \in D_X$ where $D_X = \prod_{x \in X} D_x$ and a *behavior* over X is an element $\sigma \in (\mathbb{R}_+ \to D_X)$. For all $x \in X$, let $\sigma(x) \in (\mathbb{R}_+ \to D_x)$ be the x-coordinate of σ, termed a *signal*. By abuse of notation, and as no confusion will result, we write x_t instead of $\sigma \to \sigma(t)(x)$ and ζ_t instead of $\sigma \to \sigma(t)(\zeta)$. We now briefly review the primitives listed in (26).

Eq_1: means that $y_t = f(x_t^1, \ldots, x_t^n)$ holds for all t, where f is a total function over its domain; and the tuple $[x] = [x^1, \ldots, x^n]$;.

Eq_2: means $y_t = x_{t\_} \;=_{\text{def}}\; \lim_{s \nearrow t} x_s$, i.e., y_t is the *left-limit* of x_s when s approaches t from below.

Eq_3: defines the clock ζ such that, using convention (28):

$$\zeta_t \quad = \quad [b_{t\_} = \textsc{f}] \wedge [b_t = \textsc{t}] \text{ where } b_t = [z_t \geq 0]$$

Thus ζ selects the instants t at which z_t crosses zero from below, we call such a clock a *zero-crossing*. We consider tuples $[\zeta^1, \ldots, \zeta^k]$ of zero-crossings, denoted by the symbol $[\zeta]$.

Eq_4: Given two signals x and y, a value y_0, and a *discrete* signal z (see below), Eq_4 states that ODE $\dot{y}_t = x_t$ holds with initial condition y_0 and is reset to the value given by z at each instant of the discrete clock of z.

Eq_5: Given a signal y, a value $y_0 \in \mathbb{R}^n$, two matching[18] tuples of zero-crossings and signals $[\zeta] = [\zeta^1, \ldots, \zeta^k]$ and $[x] = [x^1, \ldots, x^k]$, Eq_5 states that $y_t = y_0$

[18] $[x^1, \ldots, x^k]$ and $[y^1, \ldots, y^l]$ are *matching* if $k = l$.

> for $t < t_1$, the first instant of ζ that y has a clock $\zeta = \bigcup_{i=1}^{k} \zeta^i$, and that for every $t \in \zeta^i \setminus (\bigcup_{j<i} \zeta^j)$, $z_t = (x^i)_t$. That is, when there are simultaneous zero-crossings, priority is given to the one with the lowest index.

Eq_1–Eq_5 are sufficient to define systems of ODEs with mode changes and reset conditions. The statement Eq_6 allows the embedding of discrete time systems. Before explaining it, we must first clarify what we mean by *discrete time*.

Signals are typed discrete or continuous. For each signal x, we assume a clock τ_x such that x is guaranteed constant on the complement of τ_x. We call τ_x the *clock of* x and take the following convention:

> A signal is termed *discrete* if it has been declared as such, or if its clock is a zero-crossing. Otherwise it is termed *continuous*.

Thus, Eq_3 defines a discrete clock. Eq_5 and Eq_6 define discrete signals. Well formation rules are a weak form of typing constraints: for being properly defined, the reset argument z from Eq_4 must be discrete, hence the sufficient condition that it must be defined by an equation Eq_5. Note that y defined by equation Eq_5 is discrete but $[x]$ is possibly continuous. Finally, x in Eq_6 must be discrete, hence the constraint that it is defined by an equation Eq_5.

Appendix A.3 gives an example in Simulink where a discrete signal is used where a continuous is expected and conversely. This kind of wrongly typed program leads to a strange behavior. The well-formation rule is a simple syntactic criteria to properly separate discrete from continuous signals.

While mathematically a clock is discrete if its restriction to any bounded interval of $\mathbb{R}_+$ is finite, the property of being discrete or not cannot be statically checked in general. Hence the *declaration or zero-crossing* definition of "discrete":

(i) It is a syntactic criterion and can thus be statically enforced;
(ii) it usually corresponds with the mathematical definition.

For instance, the set of zero-crossings of a continuous function f:

$$\mathit{zero}(f) \ =_{\text{def}} \ \{t \in \mathbb{R}_+ \mid f(t_-) < 0 \wedge f(t) \geq 0\}$$

is a closed subset of $\mathbb{R}_+$. If, furthermore, all instants belonging to $\mathit{zero}(f)$ are isolated, i.e., if a non-empty interval separates each pair of adjacent instants, then $\mathit{zero}(f)$ is either a finite set or a diverging sequence; in either case it is discrete in the mathematical sense. Functions f from which zero-crossings are constructed would typically possess such properties. Of course, property (ii) is not guaranteed in all cases; the sets of zero-crossings of certain, tricky signals may very well be Zeno, Cantor, or even an interval of the reals (see example 2). Statically checking whether a clock is discrete in the pure mathematical sense is simply not possible. With this definition of discrete clocks, we can now describe Eq_6:

Eq_6**:** assumes that x is discrete and defines y as the delayed version of x; i.e. the clock of y equals that of x, $\tau_y = \tau_x$. At the first instant of this clock y takes the initial value of x, and thereafter the nth value of y equals the $(n-1)$th value of x. Initially, i.e., before any discrete instant, y equals y_0.

Remark. Compound statements are expressed in SimpleHybrid as the conjunction of equations. For example, combining $\dot{y} = y'$ **init** u_0 **reset** z' (form Eq_4), with $y' = f(x, y)$ (form Eq_1), $z' = [v]$ **every** $[\zeta]$ **init** u_0 (form Eq_5) and $\zeta^i = \mathbf{up}(z^i)$ (form Eq_3) yields the ODE:

$$\dot{y} = f(x, y) \textbf{ init } u_0 \textbf{ reset } [v] \textbf{ every up } [z] \tag{29}$$

which means that $\dot{y}_t = f(y_t, v_t)$ holds with initial condition $y_0 = u_0$ and that it is reset to the value v_i each time zero-crossing ζ^i occurs on z^i.

7. Non-standard semantics of SimpleHybrid

The following defines a semantics based on non-standard analysis for SimpleHybrid. We fix the infinitesimal base step as $\partial \approx 0$. Without loss of generality, we assume that $\partial = [\varepsilon_n]$ for some decreasing sequence ε_n of reals converging to 0, see section 4.3. Following [10], as our universal time base we replace $\mathbb{R}_+$ by the non-standard set:

$$\mathbb{T} \quad = \quad \{t_n = n\partial \mid n \in {}^\star\mathbb{N}\} \tag{30}$$

For $t \in \mathbb{T}$, define

$${}^\bullet t = \max\{s \mid s \in \mathbb{T}, s < t\} \qquad t^\bullet = \min\{s \mid s \in \mathbb{T}, s > t\} \tag{31}$$

Thus ${}^\bullet t_n = t_{n-1}$ and $t_n^\bullet = t_{n+1}$. The most important characteristic of $\mathbb{T}$ is that for every $u \in \mathbb{R}_+$ there exists a unique $t \in \mathbb{T}$ such that ${}^\bullet t < u \leq t$ and $t - u$ is infinitesimal. Thus although $\mathbb{T}$ is dense in $\mathbb{R}_+$,[19] it can still be treated as discrete and totally ordered.

A *hybrid system* is a pair $S = (X, \Sigma)$ of a finite set of variables $X \subseteq \mathcal{X}$ and a set of behaviors Σ over X. The variables include a subset $T \subset X$ of clock variables, i.e., $T \subseteq \mathcal{T}$. For $Y \supseteq X$, we can lift Σ to Y, written $\Sigma^{\uparrow Y}$, by taking all behaviors over Y whose projection onto X is in Σ. Then, for $S_i = (X_i, \Sigma_i), i = 1, 2$, we define the *parallel composition*

$$S_1 \parallel S_2 = \left(X_1 \cup X_2, {\Sigma_1}^{\uparrow X_1 \cup X_2} \cap {\Sigma_2}^{\uparrow X_1 \cup X_2}\right). \tag{32}$$

The non-standard semantics of SimpleHybrid is given in Table 1, mid column. Note the semantics of $\zeta = \mathbf{up}(z)$ is a "weak preemption" since a

[19] "$\mathbb{T}$ is dense in $\mathbb{R}_+$" means that, for every $t \in \mathbb{R}_+$ and every $\epsilon > 0$, the interval $(t-\epsilon, t+\epsilon)$ intersects $\mathbb{T}$.

statement	**non-standard semantics**	**transition relation**
$y = f([x])$	$y_t = f([x_t])$	$y = f([x])$
$y = \mathbf{last}\,(x)$	$y_t = x_{\bullet_t}$	$y = {}^{\bullet}x$
$\zeta = \mathbf{up}(x)$	$\zeta_{t^\bullet} = [x_{\bullet_t} < 0] \wedge [x_t \geq 0]$	$\zeta^{\bullet} = [{}^{\bullet}x < 0] \wedge [x \geq 0]$
$\dot{y} = x$ **init** y_0 **reset** z	$t \in \tau \setminus \tau_z \Rightarrow y_t = y_{\bullet_t} + \partial \times x_{\bullet_t}$ $t \in \tau_z \Rightarrow y_t = z_t$	on $\tau \setminus \tau_z : y = {}^{\bullet}y + \partial \times {}^{\bullet}x$ on $\tau_z : y = z$
$y = [x]$ **every** $[\zeta]$ **init** y_0	$t < \min(\bigcup_i \zeta_i) \Rightarrow y_t = y_0$ $t \in \zeta_i \setminus (\bigcup_{j<i} \zeta_j) \Rightarrow y_t = x_{i,t}$	before $\min(\bigcup_i \zeta_i) : y = y_0$ on $\zeta_i \setminus (\bigcup_{j<i} \zeta_j) : y = x_i$
$y = \mathbf{pre}\,(x)$ **init** y_0	$\tau_y = \tau_x$ $t < \min(\tau_y) \Rightarrow y_t = y_0$ $t \in \tau_y \Rightarrow y_t = x_{\bullet_t}$	$\tau_y = \tau_x$ before $\min(\tau_y) : y = y_0$ on $\tau_y : y = {}^{\bullet}x$
$S_1 \parallel S_2$ $S_1 = (X_1, \Sigma_1)$ $S_2 = (X_2, \Sigma_2)$	$\big(X, {\Sigma_1}^{\uparrow X} \cap {\Sigma_2}^{\uparrow X}\big)$ where $X = X_1 \cup X_2$	conjunction

Table 1: Non-standard semantics of SimpleHybrid.

change in the sign of z at instant t does not result in a zero-crossing until the next instant $t^\bullet$.

This non-standard semantics defines a transition system acting on $\mathbb{T}$, which is obtained by abstracting away the time index t from the non-standard semantics of Table 1, mid column. To this end, for each variable $x \in X$ in a given system $S = (X, \Sigma)$, we augment X with the two auxiliary variables ${}^{\bullet}x$ and $x^{\bullet}$, such that, for every t

$$ {}^{\bullet}x_t = x_{\bullet_t} \text{ (assuming } t > 0), \text{ and } x^{\bullet}_t = x_{t^\bullet} \tag{33} $$

Using auxiliary these variables, the transition relation is obtained by abstracting away index t from Table 1, mid column. The result is shown in Table 1, right column. In this column, statement "on τ" is an abstraction for "$\forall t \in \tau$" and statement "before $\min(\tau)$" is an abstraction for "$\forall t < \min(\tau)$", where τ is seen as a subset of $\mathbb{T}$.

An important characteristic of this non-standard semantics is that, unlike for a fixed step-size (standard) semantics, it does not suffer from overshoot problems for zero-crossings, or even Zenoness, or any need for mentioning continuity properties, since steps are infinitesimal but "discrete". Yet the semantics is still statically defined, as was desired.

8. Constructive semantics of SimpleHybrid

A *constructive semantics* [7] formalizes how a synchronous reaction should be executed, that is, how actions should be scheduled within an instant while

respecting the causality constraints of a program. G. Berry [7] advocates using a Scott domain with an extra value "undefined", to be interpreted as "not executed yet"; the domain of values is made a flat partial order by setting "undefined is less than any other value". Esterel reactions are encoded as sets of equations over the domain and the minimal fix-point is sought by iterating from the configuration where all variables and signals are undefined. If all variables are uniquely defined in the fix-point then the reaction is deterministic and can be executed. An earlier approach was proposed by F. Boussinot [11] based on micro-step automata, which are automata describing the allowed schedules, and the decomposition of a reaction into micro-steps of atomic operations. These two approaches were also developed and shown equivalent for SIGNAL [3].

8.1. The constructive semantics

statement	**constructive semantics**
$y = f([x])$	$[x] \rhd y$
$y = \mathbf{last}\,(x)$	
$\dot{y} = x$ **init** y_0 **reset** z	$[\,\tau_z, z\,] \rhd y$
$y = [v]$ **every** $[\zeta]$ **init** y_0	$[\,[\zeta]\,, [v], y_0\,] \rhd y$
$y = \mathbf{pre}\,(x)$ **init** y_0	$\tau_x \rhd y$
$S_1 \parallel S_2$	conjunction

Table 2: Constructive semantics of SIMPLEHYBRID.

In this section we develop a Scott semantics for SIMPLEHYBRID. Let $\perp$ be a special value not belonging to any domain D_x, to be interpreted as "not evaluated yet".[20] For $x \in \mathcal{X}$, let $D_x^\perp = D_x \cup \{\perp\}$, and write $x = \top$ to mean that $x \neq \perp$. Let $\rhd$ be the *scheduling constraint* that relates any two variables u and v, that have, respectively, domains $D_u^\perp$ and $D_v^\perp$:

$$u \rhd v \quad =_{\text{def}} \quad [u = \top] \ \vee \ [v = \perp] \tag{34}$$

i.e., $u \rhd v$ means $[v = \top] \Rightarrow [u = \top]$, which formalizes that "v cannot be evaluated strictly before u". In particular, for any clock τ,

$$\forall t \in \tau \Rightarrow x_t \rhd y_t \quad =_{\text{def}} \quad [x_t = \top] \vee [y_t = \perp] \vee [\tau_t = \text{F}]$$

where τ_t is defined in (28). Observe that statements of the form $v = f(u)$, where f is a function, are abstracted to $u \rhd v$ since v can always be replaced

[20] This notation deviates from the historically established use of the symbol $\perp$ in synchronous languages to denote absence. Signal absence is a well-defined status obtained during the calculation of a reaction. Thus "absence" and "not evaluated yet" should not be confused.

by $f(u)$. The relation $\triangleright$ expresses causality constraints within systems of equations. The constructive semantics is obtained from the non-standard semantics by 1/ replacing any statement of the form $y_t = exp$ where expression exp involves variables x_t, u_t, τ_t for $t \in \mathbb{T}$, by the more abstract scheduling constraints $x_t \triangleright y_t$, $u_t \triangleright y_t$, and $\tau_t \triangleright y_t$, 2/ abstracting away any context (such as induced by **reset** , **init** , **every**), and 3/ abstracting away dummy time index t. The result is shown in Table 2, where $[u, v] \triangleright x$ means the conjunction of $u \triangleright x$ and $v \triangleright x$.

Remark. It is tempting to extend SimpleHybrid with the statement $x \triangleright y$, which would be treated similarly to Eq_1. Doing so would make it possible to express the causality analysis of a SimpleHybrid program, and even additional scheduling constraints that the programmer may want to enforce, in the language itself. Such ideas already exist, in fact, in the Signal synchronous language [4].

8.2. Various uses of the constructive semantics

In this section we develop various uses of the constructive semantics.

8.2.1. Avoiding causality cycles

Using the above abstraction, the transitive closure of relation $\triangleright$ is a *pre-order* on X, which we will also, by abuse of notation, call $\triangleright$. If S is such that $\triangleright$ is a *partial order*, then S is free of causality cycles and its variables can be evaluated according to any order compatible with $\triangleright$. The only possible cause of cycles in relation $\triangleright$ is through sets of statements of form Eq_1. This justifies the requirement that programs must not have any delay-free, derivative-free, data-flow cycles. If this condition is satisfied, topological sorting yields the required scheduling. In other words, the classical technique used in synchronous languages (e.g., Lustre, Lucid Synchrone, Scade) also applies here. For the remainder of this section, we assume that this requirement is met.

8.2.2. Single-assignment condition

We say that a system S obeys the single-assignment condition if no variable of S sits on the left-hand side of two or more equations. The following lemma is instrumental in obtaining the correct schedulings for executing a SimpleHybrid system.

Lemma 4. *If S is free of causality cycles and obeys the single-assignment condition, then it defines a deterministic input-output transition system and the partial order $\triangleright$ specifies all correct schedulings for the execution of S.*

The correct schedulings are obtained by applying topological sorting on the graph defined by the relation $\triangleright$ used in the constructive semantics of Table 2.

8.2.3. Bond Graph's causality analysis

Our constructive semantics can be enhanced to encompass the "causality analysis" performed in Modelica or in Bond Graphs [35, 29]. Suppose that the causal equality of Eq_1 were replaced by a more general form of constraint:

$$\begin{array}{rcl} Eq_{1'} & : & C(X), \text{ where} \\ & & C \text{ is solvable for } Y \subset X \end{array} \tag{35}$$

$C(X)$ denotes a constraint that relates the tuple of variables belonging to X. The additional clause states that, for every variable $y \in Y$, there exists a function f_y of the set of variables $X_{/y} = X \setminus \{y\}$, such that $C(X)$ holds if and only if $y = f_y(X_{/y})$. Typical examples are the constraints defined by the *series* and *parallel junctions* in Bond Graphs [35, 29], where, respectively, so-called "efforts" and "flows" sum to zero. Consider $x + y + z = 0$, for example. We have $X = Y = \{x, y, z\}$, since we can rewrite the junction as $x = -y - z$, $y = -x - z$, or $z = -x - y$. This kind of junction arises, for example, from the application of Kirchoff's laws in electrical circuits. Of course, solving such constraints requires a suitable rewriting engine.

In the constructive semantics, statement (35) becomes

$$Eq_{1'} \quad : \quad \bigvee_{y \in Y} (X - \{y\}) \rhd y \tag{36}$$

which consists of a disjunction of causality constraints. Then, the causality analysis "à la Bond Graph" consists in computing the disjunction of all compatible conjunctions of causality constraints selected from each instance of equation of the form $Eq_{1'}$ in the SimpleHybrid system. If, furthermore, inputs to the system (called "sources" in the terminology of bond graphs) are specified by the designer, then the subset of solutions compatible with this specification can be inferred.

9. Kahn semantics of SimpleHybrid

The theory of Kahn Process Networks [24] provides a semantics for networks of dataflow actors. We show how an extension of this theory [15] can be applied to SimpleHybrid by considering primitive equations as dataflow actors. The semantics we obtain in this way refines the constructive semantics of Section 8. The resulting Kahn semantics will be useful in managing cascades of zero-crossings as well as multiple ODE solvers.

9.1. The Kahn semantics

To simplify we assume a unique domain D for all variables $x \in \mathcal{X}$ and we reuse the extended domain $D^\perp = D \cup \{\perp\}$ introduced in section 8. We define an ordering $\leq$ on $D^\perp$ such that for all $u, v \in D$, $\perp \leq v$, and otherwise $u \leq v$ iff $u = v$. Let $(\mathbb{T}, \leq)$ be a partial order of instants. Consider the set $\mathbf{X}^\perp$ of total functions:

$$\mathbf{X}^\perp \quad = \quad \mathbb{T} \mapsto D^\perp$$

We define a partial order relation on $\mathbf{X}^\perp$: for $\mathbf{x}, \mathbf{y} \in \mathbf{X}^\perp$, $\mathbf{x}$ is a *prefix* of $\mathbf{y}$, written $\mathbf{x} \sqsubseteq \mathbf{y}$, if

$$\forall t \in \mathbb{T}: \qquad \mathbf{y}(t) \neq \mathbf{x}(t) \;\Rightarrow\; \forall t' \geq t: \; \mathbf{x}(t') = \perp \tag{37}$$

Note that (37) only requires $\mathbb{T}$ to be partially ordered. Note also that the definition allows "gaps" in the defined values. For instance:

$$\mathbf{x} : 1, \perp, 2, \perp, \perp, \perp, \ldots \quad \sqsubseteq \quad \mathbf{y} : 1, \perp, 2, \perp, 3, \perp, \ldots$$

For $u \in \mathbb{T}$ we denote by $\mathbf{x}_u$ an arbitrary stream such that $\mathbf{x}(v) = \perp$ for every $v \geq u$. An increasing chain of such functions is denoted by $\{\mathbf{x}_u\}_{u \in \mathbb{T}}$. Adapting the reasoning of [15], Proposition 1, we now show that

Lemma 5. *$(\mathbf{X}^\perp, \sqsubseteq)$ is a Complete Partial Order (CPO).*

Proof. There is a least element of $\sqsubseteq$ in $\mathbf{X}^\perp$: $\perp(t) = \perp$, for all $t \in \mathbb{T}$. Given $\mathbf{x} \sqsubseteq \mathbf{y}$ in $\mathbf{X}^\perp$, then $\mathbf{x}(t) \leq \mathbf{y}(t)$ holds for all $t \in \mathbb{T}$. Thus, if $\{\mathbf{x}_u\}_{u \in \mathbb{T}}$ is a chain in $\mathbf{X}^\perp$, then, for any $t \in \mathbb{T}$, $\{\mathbf{x}(t) \mid \mathbf{x} \in \{\mathbf{x}_u\}_{u \in \mathbb{T}}\}$ is a chain in D and we can define $\big(\bigsqcup_{u \in \mathbb{T}} \mathbf{x}_u\big)(t) = \bigsqcup_{u \in \mathbb{T}} \mathbf{x}_u(t)$. □

From now on we further assume that $\mathbb{T}$ is totally ordered. Consider the subset of $\mathbf{X} \subset \mathbf{X}^\perp$ of *streams* consisting of those elements of $\mathbf{X}^\perp$ having no gap:

$$\mathbf{X} \quad = \quad \{\mathbf{x} \in \mathbf{X}^\perp \mid \mathbf{x}(t) \neq \perp \Rightarrow \forall s \leq t, \mathbf{x}(s) \neq \perp\}$$

Since $\mathbb{T}$ is totally ordered, streams consist of a (possibly infinite) prefix of defined values, followed by a tail of $\perp$s. $\mathbf{X}$ is closed in $\mathbf{X}^\perp$ under supremum, hence

Lemma 6. *$(\mathbf{X}, \sqsubseteq)$ is also a CPO.*

A stream is equally well characterized by its prefix of non-$\perp$ entries (its *defined prefix*), and this is the representation we will use in the sequel. Accordingly, $\sqsubseteq$ is simply the prefix order on $\mathbf{X}$. A function $f : \mathbf{X} \mapsto \mathbf{X}$ is *order-preserving* if, for any two streams $\mathbf{x}$ and $\mathbf{y}$, $\mathbf{x} \sqsubseteq \mathbf{y}$ implies $f(\mathbf{x}) \sqsubseteq f(\mathbf{y})$. Furthermore, f is *continuous* if

$$\bigsqcup_{u \in \mathbb{T}} f(\mathbf{x}_u) = f\big(\bigsqcup_{u \in \mathbb{T}} \mathbf{x}_u\big)$$

for any chain $\{\mathbf{x}_u\}_{u \in \mathbb{T}}$ of streams. The fix-point theorem for CPO's states that *any continuous function has a unique least fix-point* $\mathbf{x}$ such that $\mathbf{x} = f(\mathbf{x})$, we call it the *Kahn Process Network semantics* (KPN semantics) of f. It is known that products of CPO are CPO and that KPN compose.

We now apply this general framework to our case where $\mathbb{T}$ is given by (30).

Theorem 3. *If a* SimpleHybrid *system $S = (X, \Sigma)$ is free of causality cycles and obeys the single-assignment condition (see Lemma 4), then it possesses a KPN semantics S^{K}. Furthermore, if $S = S_1 \parallel S_2$ holds, then $S^{\mathrm{K}} = S_1^{\mathrm{K}} \parallel^{\mathrm{K}} S_2^{\mathrm{K}}$ follows, where $\parallel^{\mathrm{K}}$ denotes the composition of KPN.*

statement S	**Kahn actor network** $[\![S]\!]^{\mathrm{K}}$
$y = f([x])$	$[x] \multimap\!\!\square\!\!\to y$
$y = \mathbf{last}\,(x)$	${}^{\bullet}x \multimap\!\!\square\!\!\to y$
$\dot{y} = x$ **init** y_0 **reset** z	$[\,\tau_z, {}^{\bullet}x, y_0, z\,] \multimap\!\!\square\!\!\to y$
$y = [v]$ **every** $[\zeta]$ **init** y_0	$[\,[\zeta]\,, [v]\,, y_0\,] \multimap\!\!\square\!\!\to y$
$y = \mathbf{pre}\,(x)$ **init** y_0	$[\,\tau_x, {}^{\bullet}x, y_0\,] \multimap\!\!\square\!\!\to y$
$S_1 \parallel S_2$	$[\![S_1]\!]^{\mathrm{K}} \cup [\![S_2]\!]^{\mathrm{K}}$

Table 3: Kahn actors of SimpleHybrid.

Proof. Partition the set X of variables into its inputs and outputs: $X = X^{\mathrm{in}} \uplus X^{\mathrm{out}}$, where input variables are those which do not appear on the left hand side of any statement of S. By Lemma 4, S is a deterministic input-output transition system, whose transition function

$$F_S \quad : \quad D^{{}^{\bullet}X^{\mathrm{out}}} \times D^{X^{\mathrm{in}}} \mapsto D^{X^{\mathrm{out}}} \tag{38}$$

is obtained by applying the rules of Table 1, right column. Consequently, system S performs a non-terminating while loop of steps of the form (38) indexed by $t \in \mathbb{T}$ and starting from some given initial condition for the output variables; whence the following model for S:

$$S \quad : \quad D^{{}^{\bullet}X^{\mathrm{out}}} \times (D^{X^{\mathrm{in}}})^{\mathbb{T}} \mapsto (D^{X^{\mathrm{out}}})^{\mathbb{T}} \tag{39}$$

Model (39) expresses that S maps a pair, consisting of an initial condition for X^{out} and an input stream of vectors X^{in}, to an output stream of vectors X^{out}. Using Curry isomorphism, we can equivalently regard S as a function mapping an input vector of streams to an output vector of streams, thus obtaining in this way its KPN semantics:

$$S^{\mathrm{K}} \quad : \quad D^{{}^{\bullet}X^{\mathrm{out}}} \times (D^{\mathbb{T}})^{X^{\mathrm{in}}} \mapsto (D^{\mathbb{T}})^{X^{\mathrm{out}}} \tag{40}$$

Now, if S decomposes as $S = S_1 \parallel S_2$, we claim that

$$S^{\mathrm{K}} \quad = \quad S_1^{\mathrm{K}} \parallel^{\mathrm{K}} S_2^{\mathrm{K}} \tag{41}$$

where $\parallel^{\mathrm{K}}$ denotes the composition according to the Kahn Process Network (KPN) semantics. To show this, note that S^{K} is a KPN semantics of the system of equations defined by the pair $(S_1^{\mathrm{K}}, S_2^{\mathrm{K}})$. Since, by the fix-point theorem for CPOs, this KPN semantics is unique, equality (41) holds. □

Using Theorem 3, we can interpret Table 1, right column, as KPN nodes, also called *actors*. Networks of actors for SimpleHybrid statements are given in Table 3. Notation $[u, x] \multimap\!\!\square\!\!\to y$ denotes an actor having u and x as input streams and y as output stream. The symbol $\cup$ denotes the union of KPNs, seen as graphs.

statement	$[\![S]\!]_{\text{ZC}}$: **zero-crossing calculus**
$y = f([x])$	$y = f([x])$
$y = \mathbf{last}\,(x)$	$y = \mathbf{last}\,(x)$
$\zeta = \mathbf{up}(z)$	$\zeta = \mathbf{up}(z)$
$\dot{y} = x$ **init** y_0 **reset** z	$y = z$
$y = [v]$ **every** $[\zeta]$ **init** y_0	$y = [v]$ **every** $[\zeta]$
$y = \mathbf{pre}\,(x)$ **init** y_0	$y = \mathbf{pre}\,(x)$
$S_1 \parallel S_2$	$[\![S_1]\!]_{\text{ZC}} \parallel [\![S_2]\!]_{\text{ZC}}$

Table 4: The zero-crossing calculus of SimpleHybrid.

9.2. Various uses of the Kahn semantics

We now review various uses of the Kahn semantics, for the detection of unbounded cascades of zero-crossings and for the management of multiple ODE solvers.

9.2.1. Bounding cascades of zero-crossings

Theorem 1, the Standardisation Principle, requires that cascades of successive zero-crossings contain only finitely many zero-crossings (Condition 4). In this section we develop an abstraction to check whether a program satisfies this condition. The principle is to erase, in every primitive statement of SimpleHybrid, the fields that relate either to initial conditions or to the progress of an ODE. Doing so yields the zero-crossing calculus of Table 4. Since initial conditions are constants, there is no need to keep them in the zero-crossing calculus. This calculus can be applied to the examples of section 2, as we now show.

Example 1. The zero-crossing calculus yields:

$$\begin{array}{cccc} & y = 1 \quad \mathbf{every}\ \mathbf{up}(x) & \parallel & x = -1\ \mathbf{every}\ \mathbf{up}(y) \\ \parallel & y = -1\ \mathbf{every}\ \mathbf{up}(-x) & \parallel & x = 1 \quad \mathbf{every}\ \mathbf{up}(-y) \\ & & \parallel & x = 1 \quad \mathbf{every}\ \mathbf{up}(z) \end{array} \tag{42}$$

To calculate the Kahn actor semantics of (42) we make the various zero-crossings explicit by setting $\zeta_x^+ = \mathbf{up}(x)$, $\zeta_x^- = \mathbf{up}(-x)$, with corresponding definitions for $\zeta_y^\pm$ and ζ_z. Using these notations, the Kahn semantics of (42) is:

$$\begin{array}{cccccccccc} & x \multimap\!\!\square\!\!\to \zeta_x^{-\bullet} & \cup & x \multimap\!\!\square\!\!\to \zeta_x^{+\bullet} & \cup & \zeta_x^- \multimap\!\!\square\!\!\to y & \cup & \zeta_x^+ \multimap\!\!\square\!\!\to y \\ \cup & y \multimap\!\!\square\!\!\to \zeta_y^{-\bullet} & \cup & y \multimap\!\!\square\!\!\to \zeta_y^{+\bullet} & \cup & \zeta_y^- \multimap\!\!\square\!\!\to x & \cup & \zeta_y^+ \multimap\!\!\square\!\!\to x \\ \cup & z \multimap\!\!\square\!\!\to \zeta_z^{\bullet} & \cup & \zeta_z \multimap\!\!\square\!\!\to x & & & & \end{array} \tag{43}$$

which exhibits a cycle of zero-crossings: $\zeta_x^\pm \multimap\!\!\square\!\!\to \zeta_y^{\pm\bullet} \multimap\!\!\square\!\!\to \zeta_x^{\pm\bullet\bullet}$. Therefore, there is a risk that condition 4 of theorem 1 does not hold. We can refine

this analysis by considering (42) directly. As soon as x and y are initialized to negative values, the following cascade of zero-crossings repeats for ever and is triggered by the occurrence of ζ_z: $\zeta_x^+, \zeta_y^+, \zeta_x^-, \zeta_y^-, \zeta_x^+, \ldots$, and such an infinite periodic cascade indeed occurs in the non-standard semantics.

Example 2. The zero-crossing calculus yields:

$$x = -1 \textbf{ every up}(y) \quad \| \quad x = 1 \textbf{ every up}(-y) \tag{44}$$

The Kahn semantics becomes

$$y \multimap\!\!\Box\!\!\to \zeta_y^{+\bullet} \cup \zeta_y^+ \multimap\!\!\Box\!\!\to x \cup y \multimap\!\!\Box\!\!\to \zeta_y^{-\bullet} \cup \zeta_y^- \multimap\!\!\Box\!\!\to x$$

which, in turn, exhibits no cycles of zero-crossings. And thus, condition 4 of theorem 1 holds. Condition 1 does not, however, hold for this sliding mode example, showing that the analysis of zero-crossing cycles is no panacea.

Example 3. The zero-crossing calculus yields:

$$\begin{array}{ll} & x = \textbf{last}\,(x) + 1 \textbf{ every up}(y) \quad \| \quad y = 1 \textbf{ every up}(z) \\ \| & x = \textbf{last}\,(x) + 2 \textbf{ every up}(z) \end{array} \tag{45}$$

which, in the Kahn semantics, becomes

$$\begin{array}{llllll} & y \multimap\!\!\Box\!\!\to \zeta_y^{\bullet} & \cup & \zeta_y \multimap\!\!\Box\!\!\to x & \cup & {}^{\bullet}x \multimap\!\!\Box\!\!\to x \textbf{ every } \zeta_y \\ \cup & z \multimap\!\!\Box\!\!\to \zeta_z^{\bullet} & \cup & \zeta_z \multimap\!\!\Box\!\!\to x & \cup & {}^{\bullet}x \multimap\!\!\Box\!\!\to x \textbf{ every } \zeta_z \\ \cup & z \multimap\!\!\Box\!\!\to \zeta_z^{\bullet} & \cup & \zeta_z \multimap\!\!\Box\!\!\to y & & \end{array}$$

and the lack of zero-crossing cycles implies that condition 4 of theorem 1 holds for this example.

Example 4. It is not covered by this analysis and requires Theorem 2.

An autonomous ODE with reset. In addition, consider the following example:

$$\dot{x} = f(x) \textbf{ init } 0 \textbf{ reset } g(\textbf{last}\,(x)) \textbf{ every up}(x - \lambda)$$

where λ is a real parameter. The zero-crossing calculus yields

$$x = g(\textbf{last}\,(x)) \textbf{ every up}(x - \lambda) \tag{46}$$

which, in the Kahn semantics and with some obvious notation, becomes

$$x \multimap\!\!\Box\!\!\to \zeta_x^{\lambda\bullet} \cup \zeta_x^{\lambda} \multimap\!\!\Box\!\!\to x \cup \textbf{last}\,(x) \multimap\!\!\Box\!\!\to x \textbf{ every } \zeta_x^{\lambda}$$

which contains a path $\zeta_x^{\lambda} \multimap\!\!\Box\!\!\to \zeta_x^{\lambda\bullet}$. But, there will be no cascaded zero-crossings because for x to exceed λ requires a strictly non-zero period of time, whatever the value of λ.

A criterion. A risk of cascading zero-crossings can be detected for a system S, whose set of variables of zero-crossing is denoted Z_S, by considering, for $\zeta, \zeta' \in Z_S$, the relation:

$$\begin{array}{l}\zeta \overset{\bullet}{\rightarrow} \zeta' \text{ if} \\ \zeta \neq \zeta' \text{ and } \zeta \mathrel{-\!\!\square\!\!\rightarrow^*} {\zeta'}^{\bullet} \text{ holds in the abstract Kahn semantics of } [\![S]\!]_{\text{ZC}}\end{array} \tag{47}$$

where superscript * indicates iteration. An occurrence of ζ followed immediately, that is, at the next ∂-step, by an occurrence of ζ' indicates the possibility of a cascade of zero-crossings. Whence the following criterion:

Lemma 7 (cascaded zero-crossings). *Let $\mathcal{Z}_S$ be the directed graph collecting all relations of the form (47). If $\mathcal{Z}_S$ contains no cycles, then all cascades of successive zero-crossings of S are provably finite.*

The study of the examples shows that, when $\mathcal{Z}_S$ does contain a cycle, a refined study can be performed by statically analyzing $[\![S]\!]_{\text{ZC}}$. The corresponding developments are not detailed here.

9.2.2. Managing Multiple ODE solvers

Let $\bowtie$ be the least equivalence relation on the set of variables of S such that

$$^{\bullet}x \mathrel{-\!\!\square\!\!\rightarrow^*} y \text{ and } {}^{\bullet}y \mathrel{-\!\!\square\!\!\rightarrow^*} x \quad \Rightarrow \quad x \bowtie y \tag{48}$$

Equivalence classes for $\bowtie$ are written between braces, e.g., $\{\{x\}\}, \{\{y\}\}$. We regard them as sets of variables and call them *clusters*. Cluster $\{\{x\}\}$ is such that every component of it causally depends on every component of ${}^{\bullet}\{\{x\}\}$. For two distinct clusters $\{\{x\}\}$ and $\{\{y\}\}$, let $\{\{x\}\} \preceq \{\{y\}\}$ if $u \mathrel{-\!\!\square\!\!\rightarrow^*} v$ or ${}^{\bullet}u \mathrel{-\!\!\square\!\!\rightarrow^*} v$ holds, for some pair (u, v) such that $u \in \{\{x\}\}, v \in \{\{y\}\}$. Relation $\preceq$ is a partial order if the system under consideration is free of causality cycles.

By the non-standard and Kahn semantics of SimpleHybrid, variables belonging to the same cluster must be handled as a system of interconnected ODEs with associated zero-crossings. For a system with two clusters such that $\{\{x\}\} \preceq \{\{y\}\}$, a first ODE solver could be given the variables belonging to $\{\{x\}\}$, and some of its output variables could serve as inputs for another ODE solver associated with $\{\{y\}\}$.

Managing solvers in this way can improve some aspects of variable step size tuning. More precisely, when using a single ODE solver to simulate a system, the choice of step size is governed by the sub-systems of strongest stiffness. This is undesirable when not all of the sub-systems actually interact, i.e., when some of them belong to different equivalence classes of $\bowtie$. When combined with loose handling of the distinction between continuous and discrete behaviours (as is the case in Simulink), this fact can result in the manifestation, in simulation results, of strange couplings between seemingly non-interacting sub-systems.

10. Off-the-shelf compilers

In this section we explain how to derive a SimpleHybrid compiler by reusing a legacy synchronous compiler in combination with a legacy ODE solver. The

synchronous language engine will regard the activation of ODE between two successive zero-crossings as just another (big) step, abstracting away from the fact that this step is managed by an external engine, namely the ODE solver. Throughout this section, the following assumption is in force:

Assumption 1. *The system S satisfies the assumptions of Theorem 1.*

The key idea is to structure SimpleHybrid systems in a specific way. Decompose every SimpleHybrid system S as

$$S \quad = \quad S_{\mathrm{ODE}} \parallel S_{\mathrm{noODE}}, \tag{49}$$

where S_{ODE} and S_{noODE} are constructed as shown in Table 5. This structuring is explained next.

statement of S	**Assigned to S_{noODE}**	**Assigned to S_{ODE}**
$y = f([x])$	on ζ_S : $y = f([x])$	outside ζ_S : $y = f([x])$
$y =$ **last** (x)	on ζ_S : $y =$ **last** (x)	outside ζ_S : $y =$ **last** (x)
$\zeta =$ **up**(x)		$\zeta =$ **up**(x)
$\dot{y} = x$ **init** y_0 **reset** z		$\dot{y} = x$ **init** y_0 **reset** z
$y = [x]$ **every** $[\zeta]$ **init** y_0	$y = [x]$ **every** $[\zeta]$ **init** y_0	
$y =$ **pre** (x) **init** y_0	$y =$ **pre** (x) **init** y_0	

Table 5: Structuring a system into its discrete and ODE parts. The added guards are shown in red.

Subsystem S_{ODE} is the part of S that is to be submitted to an ODE solver. Numerical solvers like the CVODE or CVODES solvers in the Sundials suite [21] solve systems of ODEs with initial conditions with respect to an independent variable (usually denoting time); they typically also include a feature for detecting and halting at zero-crossings Accordingly, S_{ODE} must include all equations of S of forms Eq_1–Eq_4. Indeed, Eq_4 ($\dot{y} = x$ **init** y_0 **reset** u) denotes an ODE with its initial condition and reset signal. The actual expression defining x in Eq_4 uses equations of the form Eq_1 (instantaneous function) and Eq_2 (last state). Since the ODE solver is in charge of detecting zero-crossings, equations of the form Eq_3 must also be included in S_{ODE}.

Subsystem S_{noODE} is the part of S to be handled by a synchronous language engine. S_{noODE} must include all those equations of S that have one of the forms Eq_1, Eq_2, Eq_5, or Eq_6. Equations of forms Eq_1 and Eq_2 are needed because the

variables they define may occur in expressions defining tuples of reset signals $[v]$. Equations of form Eq_5 describe the discrete parts of S, i.e. those parts triggered by zero-crossing tuples $[\zeta]$, that may incorporate discrete states defined by Eq_6.

Guards must be added to equations of forms Eq_1 and Eq_2. This addition is necessary to preserve the single-assignment property. Equations of forms Eq_1 and Eq_2 are considered as part of S_{noODE} at instants of zero-crossing, and otherwise as part of S_{ODE}. This is achieved by reusing the technique of guards already used in Table 1 for the definition of the transition relations associated to SimpleHybrid. For instance, statements

$$\text{“on } \zeta_S : y = f([x])\text{”} \quad \text{and} \quad \text{“outside } \zeta_S : y = f([x])\text{”}$$

respectively mean

$$\forall t \in \zeta_S : y_t = f([x_t]) \quad \text{and} \quad \forall t \notin \zeta_S : y_t = f([x_t]).$$

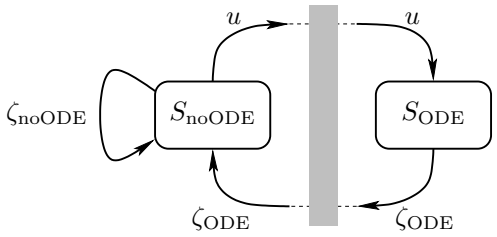


Figure 1: Two-mode automaton. Left: synchronous engine. Right: ODE solver.

The following observations can be formulated using decomposition (49) and Table 5, see Figure 1. Using Theorem 1, partition clock ζ_S into the instants ζ_{ODE} of zero-crossing triggered by the activation of the ODE part, and the instants ζ_{noODE} of cascaded zero-crossings. The discrete part S_{noODE} is activated on ticks of clock $\zeta_S = \zeta_{\text{noODE}} \vee \zeta_{\text{ODE}}$. S_{noODE} is responsible for setting the reset signals u of the ODE part S_{ODE}. The left mode is handled by the synchronous engine, whereas the right mode requires an activation of the ODE solver.

What happens if the assumptions of Theorem 1 are not satisfied? We will not provide a comprehensive solution for this case, but we can offer some hints for when the weaker assumptions of Theorem 2 apply — such as in the colliding balls example 4. Instead of S_{noODE} being triggered by (possibly cascaded) zero-crossings, we must consider that S_{noODE} is triggered by the larger set of so-called “non-infinitesimal” steps. This includes runs of the ODE part that violate the guard immediately upon starting, see step 3 of the non-standard semantics of Example (4). A new difficulty arises in this case, namely: no static typing seems to exist that can separate the duties of S_{noODE} from those of S_{ODE}. Of course, the handling of example 2 is even more difficult.

11. Related work

There are not very many studies on hybrid systems modelers in terms of programming language semantics. We discuss the few that we know of and consider relevant for comparison. First of all, we would like to recall the legacy work [2] by one of the authors. In fact, the agenda presented in that paper closely resembles the one we develop here. Except that the tool of non-standard analysis was not used. As a consequence, [2] suffers from some hand waving, as careful readers will notice. A very preliminary version of this work was presented in [6].

The Ptolemy project. Perhaps the work the most similar to ours is that of E.A. Lee and H. Zheng [26, 27] in the context of the Ptolemy project. One of the problems it addresses is the handling of discontinuities in hybrid systems modelers. For a typical situation where discontinuities occur consider an ODE $\dot{x} = f(x, u)$, where u is some input signal and the initial condition is discarded. Suppose that, at the first instant where $g(x) \leq 0$ for some real-valued function g, the above ODE is reset to $x = h(v)$ for some other input signal, and then it restarts and the same thing happens again. To properly handle this resetting mechanism, the following critical values of x must be considered: 1) the first x where $g(x) \leq 0$ holds in the ODE; and 2) the resetting value $x' = h(v)$ at the same instant. From the mathematical viewpoint, the two values for x occur at the same time, but they are causally ordered. This schizophrenia for x is unavoidable. Following the idea of *tagged signals* initially proposed in [25], the solution taken in [26, 27] is to *tag* events with an extended time index taken from index set $\mathbb{R}_+ \times \mathbb{N}$ equipped with the lexicographic order. In this approach, the two values for x would be indexed respectively as $x_{t,0}$ and $x_{t,1} = x'$, where $t \in \mathbb{R}_+$ is the common real instant at which the two events occur. The tag set $\mathbb{R}_+ \times \mathbb{N}$ is referred to by the authors as *super-dense time*.

Our present approach avoids using this mechanism of super-dense time, because non-standard index set $\mathbb{T}$ is both discrete and dense. The existence of previous instants ${}^\bullet t$ and next instants $t^\bullet$ was used in Table 2 as an alternative to the multi-dimensional instants $(t, 0)$ and $(t, 1)$ of [26, 27]. The operator $\mathbf{last}\,(x)$ of SimpleHybrid in particular uses the previous instant ${}^\bullet t$. Note that this complies in fact with the integration operator that is provided in Simulink: equation $(x, lx) = \texttt{integr}(x_0, x', r)$ defines x to be the integral of signal x with initial value x_0. In this equation, r is a reset condition. Then, lx stands for the *internal state*; lx equals x except at instants of reset, where it is used to avoid algebraic loops. Thus, lx corresponds to $\mathbf{last}\,(x)$ in SimpleHybrid.

On another aspect, the work [26, 27] is made complicated by issues of smoothness, Lipschitzness, existence and uniqueness of solutions, Zenoness, etc. This is particularly evident in Section 6 of [26] on "Ideal Solver Semantics" and Section 7 of [27] on "Continuous-Time Models". In our approach — compare section 7 of this paper with the above mentioned Sections of [26, 27] — these issues are postponed to the very end, after execution schemes have been built and the result is submitted to solvers. The problems do not disappear from

the whole process, rather they are postponed to run time, as per the desire expressed in the introduction. Our job, as computer scientists, is thus nicely isolated and cleanly separated from that of numerical analysts.

The work performed at The Mathworks. The work performed by P. Mosterman and his co-workers at The Mathworks [31] is also very interesting, in its attempt to establish the Simulink modeler on a solid semantic basis. The authors begin by acknowledging the need for variable step solvers. The contribution of this paper is to show how (a restricted class of) variable step solvers can be given a functional *stream* semantics [16].

To achieve this, the class of solvers is first restricted to those relying on *explicit schemes,* as *implicit* ones cannot directly be manipulated into explicit functional form. The second difficulty consists in the use of iterative solving in order to adapt the variable step size online — see our discussion following equation (6) in section 3. This mechanism, again, does not have a functional shape since several successive integrations with different step sizes are compared, for a single time interval, in order to select the appropriate step size. [31] proposes to re-cast the above procedure to a functional form by replacing a repeated integration with smaller step size, by its increment with respect to the previous integration. If explicit schemes are used, then an explicit form for this increment can be found and added to the previous integration. Observe that this technique requires using the mechanism of super-dense time since a single time interval is processed several times until an adequate step size is found. The same principle can be applied to the detection of zero-crossings, but this requires using the counterpart of explicit schemes, namely cautious zero-crossing detection by slowing down without overshoot. While this indeed provides a hybrid systems modeler with a stream semantics, this semantics is rather complex since it makes the discretization method explicit; in particular, changing the latter changes the semantics. This approach forbids the use of implicit schemes, although they are valuable from the numerical analysis point of view. We also believe that this method cannot easily support the kind of clock-configuration-dependent causality analysis like that provided by our constructive and Kahn semantics.

The work of Ramine Nikoukhah. In [33], R. Nikoukhah discusses cascaded zero-crossings. He advocates using a micro-step style of interpretation, where cascaded zero-crossings are interleaved non-deterministically. We prefer a synchronous interpretation in which the programmer makes explicit what to do when two zero-crossings occur. Then non-determinism arises solely from numerical solvers, and not from the semantics of a program. Because the effect of $\mathbf{up}(e)$ is delayed by one cycle of $\mathbb{T}$, a cascade of zero-crossing can last for several successive instants of $\mathbb{T}$. Note that the synchronous interpretation coincides with that of Simulink (see discussion in Appendix A) where zero-crossings have an immediate effect.

12. Conclusion

We have proposed a novel approach to defining the semantics of hybrid systems modelers with the following objectives: To leave the choice of integration method for ODEs totally free, as such a choice must be governed by numerical considerations only; To cast hybrid systems as a conservative extension of discrete time ones; To give semantic support for the following tasks:

- Scheduling different actions triggered by zero-crossings;
- Safely typing operators that are only defined in discrete contexts;
- Rejecting programs with causality cycles (or, at least, providing precise warnings); Supporting a causality analysis à la Bond Graph;
- Allowing the use of several local solvers instead of a single, global, one, with the objective of limiting side effects between non-interacting subsystems, due to step size adjustments.

Achieving these objectives was made possible thanks to the use of non-standard analysis as a semantic domain. The main point is that a non-standard semantics allows a clean separation between the tasks of the computer scientist (answering the above questions) and that of the numerical analyst (tuning solvers). Also, we believe that non-standard semantics is not a fancy thing for math addicts, but rather a very natural way of viewing continuous-time and hybrid systems in a syntactic manner, as computer scientists usually prefer. While the first author has been aware of non-standard analysis since the mid-eighties, it is only the presentation by Lindstrøm [28], as reported in [9], that allowed the authors to become familiar with the subject.

Are we done with semantic issues? Not quite. Our standardization principle (Theorem 1) provides an adequate answer for Examples 1 (no semantics) and 3 (super-dense time semantics). Example 4 (colliding balls) was covered by our extended standardization principle (Theorem 2). However, the example 2 (sliding mode control) is not covered by our theory. yet, it is physically relevant and we would like to give it a meaningful standard semantics, and, possibly, provide effective execution schemes for it.[21] This is left for future research. The study of DAE compliant hybrid systems modelers, such as Modelica, with the same objectives is also left as a future objective.

Acknowledgment

The authors are indebted to Ramine Nikoukhah and Sébastien Furic for detailed discussions regarding Modelica and to Daniel Krob and Simon Bliudze for comments on their work.

[21] Analyses of such examples can be performed manually by a mathematician. For example, sliding mode control is known to control scientists [19, 36]. Similarly, systems with time scales differing by orders of magnitude are known to mathematicians and referred to as singularly perturbed systems [22]. What we want instead are automatic compilation techniques able to handle such systems.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theor. Comput. Sci.*, 138(1):3–34, 1995.

[2] A. Benveniste. Compositional and uniform modelling of hybrid systems. *IEEE Trans. on Automatic Control*, 43(4):579–584, April 1998.

[3] A. Benveniste, B. Caillaud, and P. Le Guernic. Compositionality in dataflow synchronous languages: Specification and distributed code generation. *Inf. Comput.*, 163(1):125–171, 2000.

[4] A. Benveniste, P. Caspi, S.A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91(1):64–83, 2003.

[5] Albert Benveniste, Timothy Bourke, Benoit Caillaud, and Marc Pouzet. Divide and recycle: types and compilation for a hybrid synchronous language. In *ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools and Theory for Embedded Systems (LCTES'11)*, Chicago, USA, April 2011.

[6] Albert Benveniste, Benoît Caillaud, and Marc Pouzet. The fundamentals of hybrid systems modelers. In *CDC*, pages 4180–4185. IEEE, 2010.

[7] G. Berry. Constructive semantics of Esterel: From theory to practice (abstract). In *AMAST '96: Proceedings of the 5th International Conference on Algebraic Methodology and Software Technology*, page 225, London, UK, 1996. Springer-Verlag.

[8] Gérard Berry. *The Constructive Semantics of Esterel*. 2002. Draft book, current version 3.0.

[9] S. Bliudze. *Un cadre formel pour l'étude des systèmes industriels complexes: un exemple basé sur l'infrastructure de l'UMTS*. PhD thesis, Ecole Polytechnique, 2006.

[10] S. Bliudze and D. Krob. Modelling of complex systems: Systems as dataflow machines. *Fundam. Inform.*, 91(2):251–274, 2009.

[11] F. Boussinot. Une sémantique du langage Esterel . Technical Report 577, INRIA, 1986.

[12] J.C. Butcher. *Numerical Methods for Ordinary Differential Equations*. Wiley, 2003. ISBN 0-471-96758-0.

[13] Stephen L. Campbell, Jean-Philippe Chancelier, and Ramine Nikoukhah. *Modeling and Simulation in Scilab/Scicos*. Springer, 2006. ISBN 0-387-27802-8.

[14] L.P. Carloni, R. Passerone, A. Pinto, and A.L. Sangiovanni-Vincentelli. Languages and tools for hybrid systems design. *Foundations and Trends in Electronic Design Automation*, 1(1/2), 2006.

[15] P. Caspi, A. Benveniste, R. Lublinerman, and S. Tripakis. Actors without directors: A Kahnian view of heterogeneous systems. In Rupak Majumdar and Paulo Tabuada, editors, *Hybrid Systems: Computation and Control (HSCC)*, volume 5469 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2009.

[16] P. Caspi and M. Pouzet. A co-iterative characterization of synchronous stream functions. *Electr. Notes Theor. Comput. Sci.*, 11, 1998.

[17] F. Diener and G. Reeb. *Analyse non standard*. Hermann, 1989.

[18] N. Cutland (ed.). *Nonstandard analysis and its applications*. Cambridge Univ. Press, 1988.

[19] A.F. Filippov. *Differential Equations with Discontinuous Right-hand Sides*. Wiley, 1988. ISBN 978-9027726995.

[20] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[21] A.C. Hindmarsh, P.N. Brown, K.E. Grant, S.L. Lee, R. Serban, D.E. Shumaker, and C.S. Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software*, 31(3):363–396, September 2005.

[22] F. Hoppensteadt. Properties of solutions of ordinary differential equations with small parameters. *Comm. on Pure and Applied Math.*, 24:807–840, 1971.

[23] Y. Iwasaki, A. Farquhar, V.A. Saraswat, D.G. Bobrow, and V. Gupta. Modeling time in hybrid systems: How fast is "instantaneous"? In *IJCAI*, pages 1773–1781, 1995.

[24] G. Kahn. The semantics of a simple language for parallel programming. In *IFIP 74 Congress*, pages 471–475, 1974.

[25] E.A. Lee and A.L. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 17(12):1217–1229, 1998.

[26] E.A. Lee and H. Zheng. Operational semantics of hybrid systems. In *HSCC*, pages 25–53, 2005.

[27] Edward A. Lee and Haiyang Zheng. Leveraging synchronous language principles for heterogeneous modeling and design of embedded systems. In *EMSOFT*, pages 114–123, 2007.

[28] T. Lindstrøm. An invitation to nonstandard analysis. In N.J. Cutland, editor, *Nonstandard Analysis and its Applications*, pages 1–105. Cambridge Univ. Press, 1988.

[29] L. Ljung and T. Glad. *Modeling of Dynamic Systems*. Prentice Hall Information and System Science Series, 1994.

[30] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J. W. de Bakker, Cornelis Huizing, Willem P. de Roever, and Grzegorz Rozenberg, editors, *REX Workshop*, volume 600 of *Lecture Notes in Computer Science*, pages 447–484. Springer, 1991.

[31] P.J. Mosterman, J. Zander, G. Hamon, and B. Denckla. Towards computational hybrid system semantics for time-based block diagrams. In *3rd IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'09)*, pages 376–385, Zaragoza, Spain, September 2009. keynote paper.

[32] M. Najafi and R. Nikoukhah. Implementation of Hybrid Automata in Scicos. In *IEEE Multi-conference on Systems and Control*, 2007.

[33] R. Nikoukhah. Hybrid dynamics in modelica: Should all events be considered synchronous? In *First Internation Workshop on Equation-Based Object Oriented Languages and Tools (EOOLT 2007)*, pages 37–48, Berlin, Germany, 2007.

[34] A. Robinson. *Non-Standard Analysis*. Princeton Landmarks in Mathematics, 1996. ISBN 0-691-04490-2.

[35] R.C. Rosenberg and D.C. Karnopp. *Introduction to Physical System Dynamics*. McGraw-Hill, New York, 1983.

[36] V. Utkin. Sliding mode control in mechanical systems. In *Industrial Electronics, Control and Instrumentation, 1994. IECON '94., 20th International Conference on*, volume 3, pages 1429 –1431 vol.3, September 1994.

Appendix A. Experimental results

We have modeled Examples 1 to 3 in both Simulink (version 7.7.0.471, R2008b) and a prototype tool based on the Sundials (version 2.4.0) CVODE library [21]. The results are presented and discussed in this appendix.

Appendix A.1. Using Simulink

Example 1. The implementation of Example 1 in Simulink is shown in Figure A.2a. It can be written as the set of equations:

$$
\begin{aligned}
y &= 1/s(iy, \mathit{updown}(lx), 0)\\
x, lx &= 1/s(ix, \mathit{updown}(z_x), 0)\\
z &= 1/s(-1, 1)\\
i_y &= \mathit{switchup}(lx, 1, -1)\\
i_x &= \mathit{switchup}(y, -1, \mathit{switchup}(-y, 1, \mathit{switchup}(z, 1, -1)))\\
z_x &= \mathit{switchup}(y, -1, \mathit{switchup}(-y, 1, \mathit{switchup}(z, 1, -1)))
\end{aligned}
$$

where:

$1/s(\mathit{init}, \mathit{zero}, \mathit{input})$ is the integral of a signal *input* with initial value *init* that is reset on the zero-crossing *zero* (which is sometimes omitted). This operator may return a second output; the so-called state port which corresponds to the left limit of signal x. The second output is written lx in the above equations.

$\mathit{updown}(r)$ outputs 1 when a rising or falling zero-crossing is detected on r otherwise it outputs 0; $\mathit{up}(r)$ is similar but only detects rising zero-crossings.

$\mathit{switchup}(x, e_1, e_2)$ returns the value of e_1 when x crosses zero, and the value of e_2 otherwise. In Simulink, $\mathit{switchup}(x, e_1, e_2)$ is implemented with a switch operator and a (rising) hit-crossing operator applied to x.

The zero-crossing handler $[-1, 1, 1]$ **every** $[y, -y, z]$ is encoded with two equations: the equation ix defines the initial value for x, and the equation z_x detects zero-crossings on y, $-y$ and z. The latter equation cannot simply be written

$$\mathit{switchup}(y, 1, \mathit{switchup}(-y, 1, \mathit{switchup}(z, 1, -1)))$$

because its value must alternate with each successive zero-crossing. In this case, as we know that zero-crossings occur first on z, then on y, and then on $-y$, we choose the values so as to go from -1 to 1 when there is a zero-crossing on z, then to -1 for a zero-crossing on y, and then to 1 for a zero-crossing on $-y$.

The simulation results are presented in Figure A.2b. Between time 0.0 and time 1.0 the value of z increases while those of x and y remain constant. When z crosses 0, just after time 1.0, the signal labeled $ixxx$ in Figure A.2a changes from -1 to 1, this value is passed through onto ixx, and then to ix and zx which causes the reset of the integrator for x to 1. The value of lx, the state port of the integrator for x, does not change from -1 to 1 immediately, but

rather on the subsequent step, this gives a rising zero-crossing which causes the reset of the integrator for y to 1. This reset is detected immediately as a zero-crossing on y that causes the reset of the integrator for x to -1. (Note that $ixxx$, and ixx both fall back to -1 during this step, but this does not cause any new zero-crossings.) A step later, the value of lx also changes from 1 to -1, which causes the reset of y to -1, which, in turn, causes a zero-crossing on $-y$, which causes ixx to become 1, which leads to a reset of the integrator for x to 1. The system is then locked into a pattern of interrelated oscillations of x, lx, and y. This is the expected behavior, but in contrast to our interpretation in the non-standard semantics section 2, and also to the implementation in Sundials, see Appendix A.2, the time in Simulink advances by a very small amount (1.7×10^{-14}) every step. Furthermore, after a certain number of oscillations, the simulation stops with an error message:

```
At time 1.000000000017115, simulation hits (1000) consecutive zero
crossings. Consecutive zero crossings will slow down the simulation
or cause the simulation to hang. To continue the simulation, you may
1) Try using Adaptive zero-crossing detection algorithm or 2)
Disable the zero crossing of the blocks shown in the following table.
 ------------------------------------------------------------------------
 No. of Consecutive Zcs |         Block type         |     Block path
 ------------------------------------------------------------------------
                   500                     HitCross     "Example1/Hit Crossing: up(lx)"
                   500                     HitCross     "Example1/Hit Crossing: up(y)"
                   499                     HitCross     "Example1/Hit Crossing: up(-y)"
                     1                     HitCross     "Example1/Hit Crossing: up(z)"
```

Whether a zero-crossing on an integral is detected instantaneously or only after a very small delay, depends on whether the output port or the state port, respectively, is monitored. In this example, the state ports of either or both x and y, rather than their output ports, must be included in the feedback loop to avoid the rejection of the model due to instantaneous dependencies.

Interestingly, if a fixed-step solver with a step size of 0.2 is used, the same values for x, y, and z are observed, but t becomes

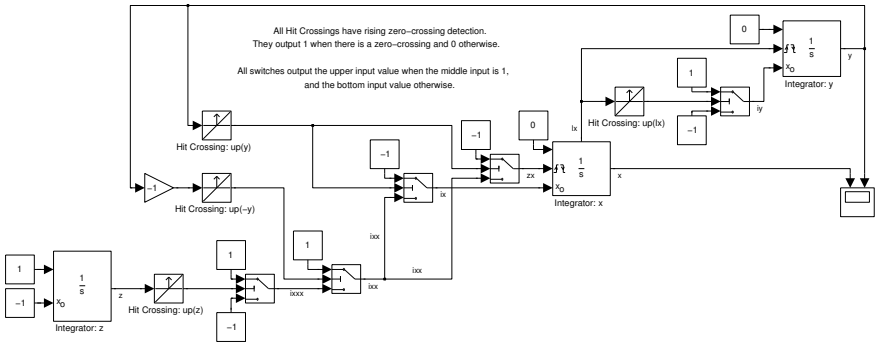
$$0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, \ldots$$

and there are no error messages since there are no zero-crossing options for fixed-step solvers. The values of t chosen by the variable-step solver are not affected by the *minimum step size* setting.

Example 2. The Simulink model is shown in Figure A.3a (with y_0 set to -1). The corresponding equations are:

$$\begin{aligned} x &= 1/s(ix, updown(y), 0) \\ y &= 1/s(y_0, x) \\ ix &= switchup(y, -1, switchup(-y, 1, -y_0)) \end{aligned}$$

In normal mode, the simulation fails with a similar error message as Example 1 because there are too many zero-crossings at instant $t = 1$. The results plotted in Figure A.3b were generated with adaptive zero-crossing detection enabled. The value of y hovers around 0 while x alternates between -1 and 1, resting at each for a period determined by the adaptive zero-crossings algorithm.



(a) Simulink model

t	x	y	z
0.000000000000000	-1.000000000000000	-1.000000000000000	-1.000000000000000
0.200000000000000	-1.000000000000000	-1.000000000000000	-0.800000000000000
0.400000000000000	-1.000000000000000	-1.000000000000000	-0.600000000000000
0.600000000000000	-1.000000000000000	-1.000000000000000	-0.400000000000000
0.800000000000000	-1.000000000000000	-1.000000000000000	-0.200000000000000
1.000000000000000	-1.000000000000000	-1.000000000000000	-0.000000000000000
1.000000000000017	1.000000000000000	-1.000000000000000	0.000000000000017
1.000000000000034	-1.000000000000000	1.000000000000000	0.000000000000034
1.000000000000051	1.000000000000000	-1.000000000000000	0.000000000000051
1.000000000000068	-1.000000000000000	1.000000000000000	0.000000000000068
1.000000000000085	1.000000000000000	-1.000000000000000	0.000000000000085
1.000000000000103	-1.000000000000000	1.000000000000000	0.000000000000103
1.000000000000120	1.000000000000000	-1.000000000000000	0.000000000000120
1.000000000000137	-1.000000000000000	1.000000000000000	0.000000000000137
1.000000000000154	1.000000000000000	-1.000000000000000	0.000000000000154
1.000000000000171	-1.000000000000000	1.000000000000000	0.000000000000171
1.000000000000188	1.000000000000000	-1.000000000000000	0.000000000000188
1.000000000000205	-1.000000000000000	1.000000000000000	0.000000000000205
1.000000000000222	1.000000000000000	-1.000000000000000	0.000000000000222
1.000000000000239	-1.000000000000000	1.000000000000000	0.000000000000239
1.000000000000256	1.000000000000000	-1.000000000000000	0.000000000000256
⋮	⋮	⋮	

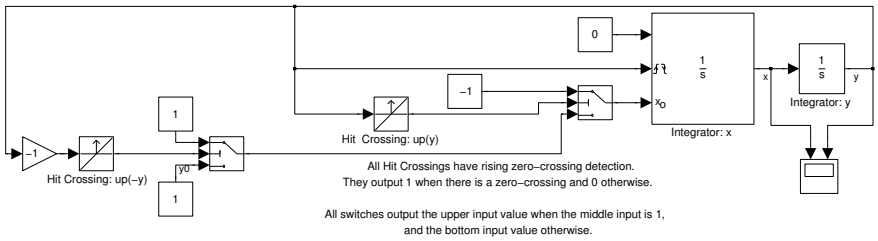
(b) Simulation results

Figure A.2: Example 1 in Simulink

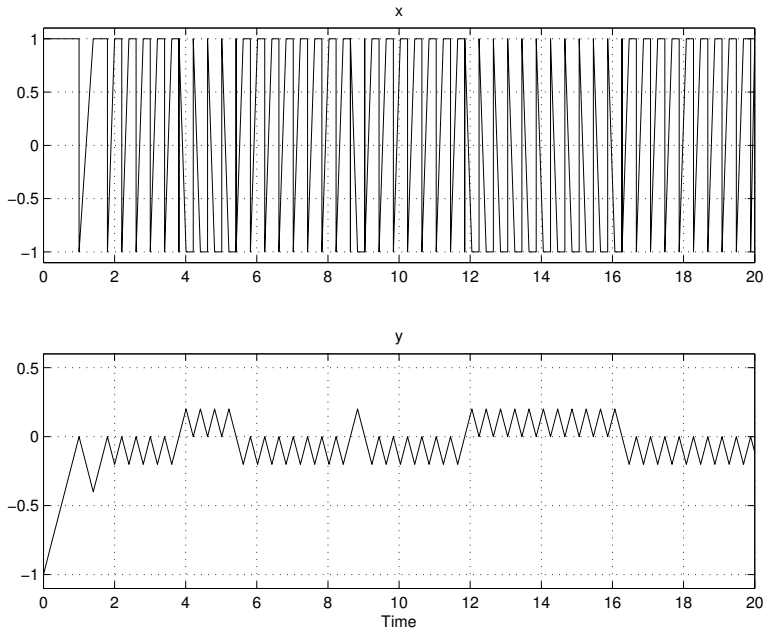
Example 3. The Simulink model is shown in Figure A.4a, and the corresponding set of equations is:

$$
\begin{aligned}
&x, lx = 1/s(ix, up(zx), 0)\\
&z = 1/s(-1, 1)\\
&y = 1/s(iy, up(z), 0)\\
&ix = switchup(y, lx + 1, switchup(z, lx + 2, 0))\\
&zx = switchup(y, 1, switchup(z, 1, -1))\\
&iy = switchup(z, 1, -1)
\end{aligned}
$$

The result of simulating this system is shown in Figure A.4b. The value of z increases from -1 until it crosses 0 at time 1. That triggers two hit crossings and the reset of the integrator for y. The top hit crossing causes the connected

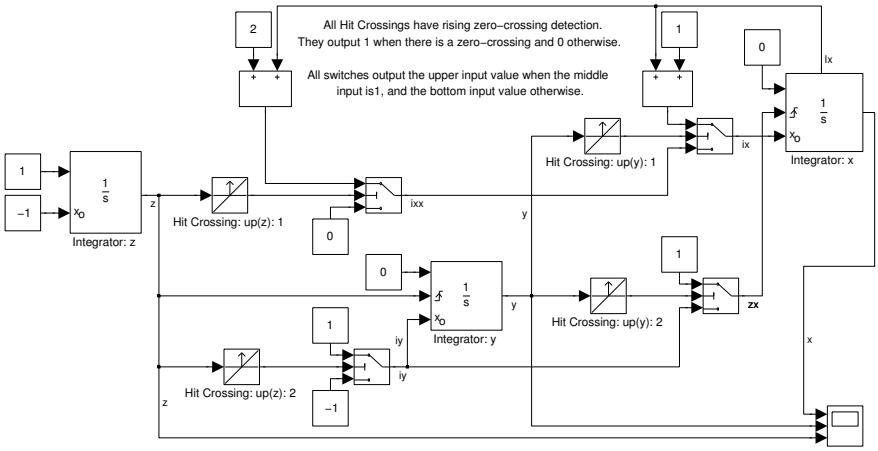


(a) Simulink model

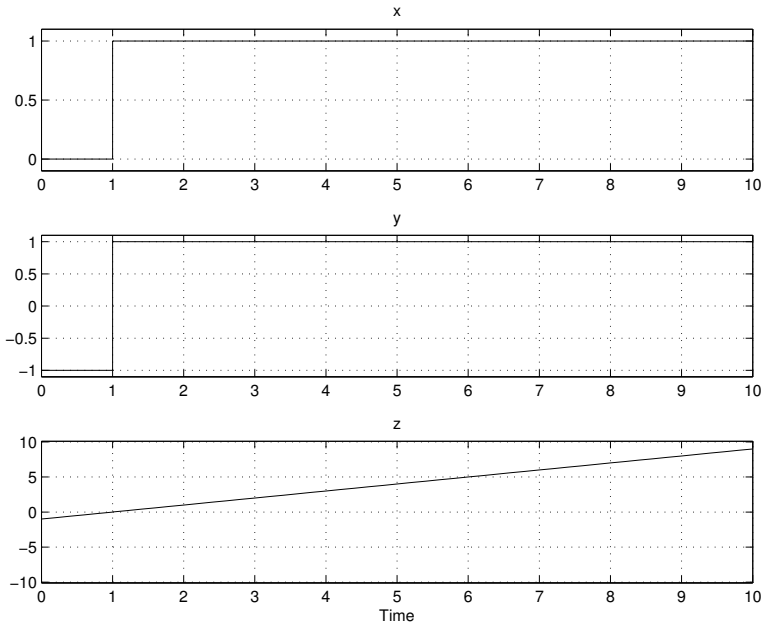


(b) Simulation results

Figure A.3: Example 2 in Simulink



(a) Simulink model



(b) Simulation results

Figure A.4: Example 3 in Simulink

switch output (labeled ixx in the figure) to become 2 ($lx + 2$). The bottom one causes the switch output (labeled iy) to become 1, which is taken as the new value of the integrator for y. The change in y from -1 to 1 triggers the two connected hit crossings. The top one causes the output of the connected switch (labeled ix) to become 1 ($lx + 1$) which overrides the value of ixx due to the zero-crossing on z. The other hit crossing sets the related switch output (labeled zx) to 1 and this zero-crossing resets the integrator for x to 1. Note that, in contrast to the non-standard interpretation explained in section 2, the zero-crossings on z and y are treated as simultaneous within Simulink. There is no delay between the two, even though the former causes the latter, and x immediately takes the value 1 rather than first taking the value 2 and then the value 3. That is, the definition of a zero-crossing in Simulink is effectively

$$\mathbf{up}(x)_t = [x_{\bullet t} \leq 0] \wedge [x_t > 0]$$

which means that the effect of a zero-crossing is instantaneous. This highlights a central benefit of using non-standard analysis as a model for reasoning about hybrid systems and the treatment of zero-crossings. The behavior of Simulink can be interpreted simply by changing the definition of $\mathbf{up}(.)$.

Appendix A.2. Using the Sundials-based Prototype

We have developed a prototype implementation of our language in Caml. It comprises a generic interface to the SUNDIALS CVODE library [21] (using serial vectors), and an implementation of the algorithm that alternates between continuous phases and discrete phases in response to zero-crossings. Each example was manually translated into a single Ocaml function that is called by Sundials during continuous phases, and by the algorithm directly during discrete phases.

Example 1. The results of running the prototype tool on Example 1 are shown in Table A.6. The first row ('I') shows the initial state values, it is followed by a series of executions of the CVODE solver ('C') during which the states evolve according to their derivatives, and then just after 1.0, a zero-crossing is detected ('Z'). The values of the continuous states at the time of the zero-crossing ('C′'), become *last* values during the subsequent discrete phase ('D'). The first zero-crossing occurs for $\mathbf{up}(z)$. It triggers an unbounded cascade of discrete phases, after each of which another (single and non-simultaneous) zero-crossing is detected. The sequence $\mathbf{up}(x)$, $\mathbf{up}(y)$, $\mathbf{up}(-x)$, $\mathbf{up}(-y)$ is repeated indefinitely without the continuous solver ever being re-invoked.

Example 2. The results for Example 2 are shown in Table A.7. The value of y exceeds zero and triggers the zero-crossing $\mathbf{up}(y)$ just after $t = 1.0$. Then, the value of x is changed from 1.0 to -1.0 during the discrete phase, but as there are no further zero-crossings the continuous solver is called again. Another zero-crossing, $\mathbf{up}(-y)$, is discovered almost immediately and another discrete phase is triggered during which x is changed back to 1.0. This process is repeated indefinitely; time is advanced in small increments by the continuous solver,

phase	time	x	y	z
I	0.000000e+00	-1.000000e+00	-1.000000e+00	-1.000000e+00
C	1.000000e-01	-1.000000e+00	-1.000000e+00	-9.000000e-01
C	2.000000e-01	-1.000000e+00	-1.000000e+00	-8.000000e-01
C	3.000000e-01	-1.000000e+00	-1.000000e+00	-7.000000e-01
C	4.000000e-01	-1.000000e+00	-1.000000e+00	-6.000000e-01
C	5.000000e-01	-1.000000e+00	-1.000000e+00	-5.000000e-01
C	6.000000e-01	-1.000000e+00	-1.000000e+00	-4.000000e-01
C	7.000000e-01	-1.000000e+00	-1.000000e+00	-3.000000e-01
C	8.000000e-01	-1.000000e+00	-1.000000e+00	-2.000000e-01
C	9.000000e-01	-1.000000e+00	-1.000000e+00	-1.000000e-01
C	1.000000e+00	-1.000000e+00	-1.000000e+00	-2.235451e-14
C′	1.000000e+00	-1.000000e+00	-1.000000e+00	7.786350e-14
Z	1.000000e+00	up(z)		
D	1.000000e+00	1.000000e+00	-1.000000e+00	7.786350e-14
Z	1.000000e+00	up(x)		
D	1.000000e+00	1.000000e+00	1.000000e+00	7.786350e-14
Z	1.000000e+00	up(y)		
D	1.000000e+00	-1.000000e+00	1.000000e+00	7.786350e-14
Z	1.000000e+00	up(-x)		
D	1.000000e+00	-1.000000e+00	-1.000000e+00	7.786350e-14
Z	1.000000e+00	up(-y)		
D	1.000000e+00	1.000000e+00	-1.000000e+00	7.786350e-14

Table A.6: Log of example 1 (prototype tool)

phase	time	x	y
I	0.000000000000000e+00	1.000000e+00	-1.000000e+00
C	1.000000000000000e-01	1.000000e+00	-9.000000e-01
C	2.000000000000000e-01	1.000000e+00	-8.000000e-01
C	3.000000000000000e-01	1.000000e+00	-7.000000e-01
C	4.000000000000000e-01	1.000000e+00	-6.000000e-01
C	5.000000000000000e-01	1.000000e+00	-5.000000e-01
C	6.000000000000000e-01	1.000000e+00	-4.000000e-01
C	7.000000000000000e-01	1.000000e+00	-3.000000e-01
C	7.999999999999999e-01	1.000000e+00	-2.000000e-01
C	8.999999999999999e-01	1.000000e+00	-1.000000e-01
C	9.999999999999999e-01	1.000000e+00	-4.464441e-14
C′	1.000000000000100e+00	1.000000e+00	5.557360e-14
Z	1.000000000000100e+00	up(y)	
D	1.000000000000100e+00	-1.000000e+00	5.557360e-14
C′	1.000000000000175e+00	-1.000000e+00	-1.974954e-14
Z	1.000000000000175e+00	up(-y)	
D	1.000000000000175e+00	1.000000e+00	-1.974954e-14
C′	1.000000000000195e+00	1.000000e+00	9.288416e-18
Z	1.000000000000195e+00	up(y)	
D	1.000000000000195e+00	-1.000000e+00	9.288416e-18
C′	1.000000000000215e+00	-1.000000e+00	-1.972025e-14
Z	1.000000000000215e+00	up(-y)	
D	1.000000000000215e+00	1.000000e+00	-1.972025e-14
C′	1.000000000000234e+00	1.000000e+00	3.853879e-17
Z	1.000000000000234e+00	up(y)	
D	1.000000000000234e+00	-1.000000e+00	3.853879e-17
C′	1.000000000000254e+00	-1.000000e+00	-1.969104e-14
Z	1.000000000000254e+00	up(-y)	
D	1.000000000000254e+00	1.000000e+00	-1.969104e-14
C′	1.000000000000274e+00	1.000000e+00	6.770241e-17

Table A.7: Log of example 2 (prototype tool)

phase	time	x	y	z
I	0.000000e+00	0.000000e+00	-1.000000e+00	-1.000000e+00
C	1.000000e-01	0.000000e+00	-1.000000e+00	-9.000000e-01
C	2.000000e-01	0.000000e+00	-1.000000e+00	-8.000000e-01
C	3.000000e-01	0.000000e+00	-1.000000e+00	-7.000000e-01
C	4.000000e-01	0.000000e+00	-1.000000e+00	-6.000000e-01
C	5.000000e-01	0.000000e+00	-1.000000e+00	-5.000000e-01
C	6.000000e-01	0.000000e+00	-1.000000e+00	-4.000000e-01
C	7.000000e-01	0.000000e+00	-1.000000e+00	-3.000000e-01
C	8.000000e-01	0.000000e+00	-1.000000e+00	-2.000000e-01
C	9.000000e-01	0.000000e+00	-1.000000e+00	-1.000000e-01
C	1.000000e+00	0.000000e+00	-1.000000e+00	-1.500536e-16
C′	1.000000e+00	0.000000e+00	-1.000000e+00	1.000680e-13
Z	1.000000e+00	up(z)		
D	1.000000e+00	2.000000e+00	1.000000e+00	1.000680e-13
Z	1.000000e+00	up(y)		
D	1.000000e+00	3.000000e+00	1.000000e+00	1.000680e-13
C	1.100000e+00	3.000000e+00	1.000000e+00	1.000000e-01
C	1.200000e+00	3.000000e+00	1.000000e+00	2.000000e-01

Table A.8: Log of example 3 (prototype tool)

and the value of x is alternated between 1.0 and -1.0 by intervening discrete phases. The observed behavior thus approximates the ideal behavior; a small overshoot, which is proportional to step size chosen by the continuous solver, effectively simulates the ε of the non-standard semantics. Note that the time column is given with a greater precision than in the other examples. Without the extra significant figures, it appears as if the simulation iterates without bound at $t = 1.0$. As it is, time barely advances just as is in Simulink when the adaptive zero-crossing detection algorithm is not used.

Example 3. The results for Example 3 are shown in Table A.8. Both x and y are constant throughout the initial continuous phases, but z increases steadily from -1.0. The first zero-crossing, $\mathbf{up}(z)$, is triggered just after z crosses 0.0. The ensuing discrete phase sees x incremented by 2.0 and y set to 1.0. The latter update triggers the zero-crossing $\mathbf{up}(y)$, which causes another discrete phase to be executed at the same instant of time. During this second discrete phase, x is incremented by 1.0. The simulation then continues with an unbounded number of continuous phases. Note that, during a discrete phase, the effects of changes to variables on zero-crossing expressions are not detected immediately, rather any new zero-crossings are detected after the discrete phase, i.e. after variables have been reset as necessary, when the last values of zero-crossing expressions are compared with their new values. There is thus no question of priority in this example: $\mathbf{up}(x)$ occurs strictly before $\mathbf{up}(y)$, even though no simulation time elapses between them.

Appendix A.3. A 'badly-typed' example

We have argued that compositions of discrete and continuous expressions must be rejected when it is not clear how discrete (logical) instants are related to (absolute) continuous time. Even when such compositions are well defined

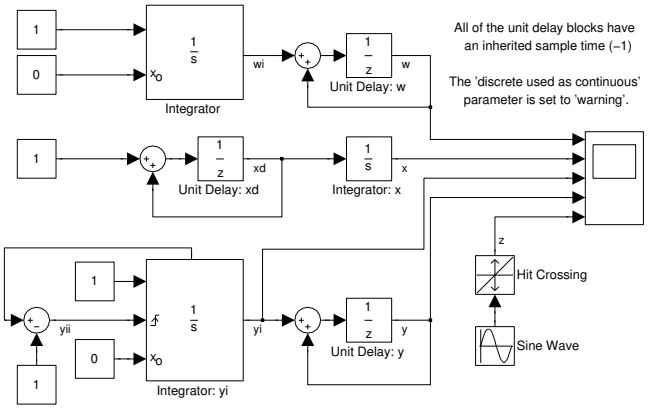


Figure A.5: A 'badly-typed' Simulink Example: model

in the non-standard semantics, they cannot be simulated by the usual numerical methods. It is instructive to model a strange combination of discrete and continuous blocks in Simulink and to observe the results.

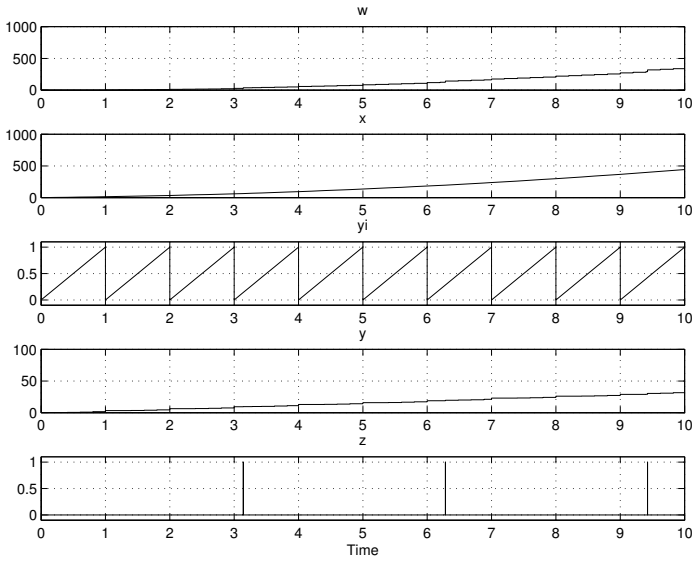
The model shown in Figure A.5 contains three strange combinations of discrete and continuous elements. At the top, the value of an integrator, wi, is fed into a discrete expression $w = (\mathbf{pre}\,(w)\ \ \mathbf{init}\ 0) + wi$. In the middle, the output of a similar discrete expression, xd, is fed into an integrator x. At the bottom, the value of an integrator with reset, yi, is fed into a discrete expression for y. All of the unit delays have an *inherited sample time*; their rate of execution is determined by those of their inputs and outputs. In parallel, a sinusoid is fed through a hit crossing block. We will see that changing the frequency of the sinusoid, changes the values of the three other signals (w, x and y).

In Simulink, there are a number of user-configurable analyses for deciding whether to accept a model, accept it with warnings, or reject it with an error message. The default settings are fairly liberal; Simulink usually prefers to execute problematic models after displaying a warning message. The model of Figure A.5 is, however, too much. By default, it is rejected with the message:

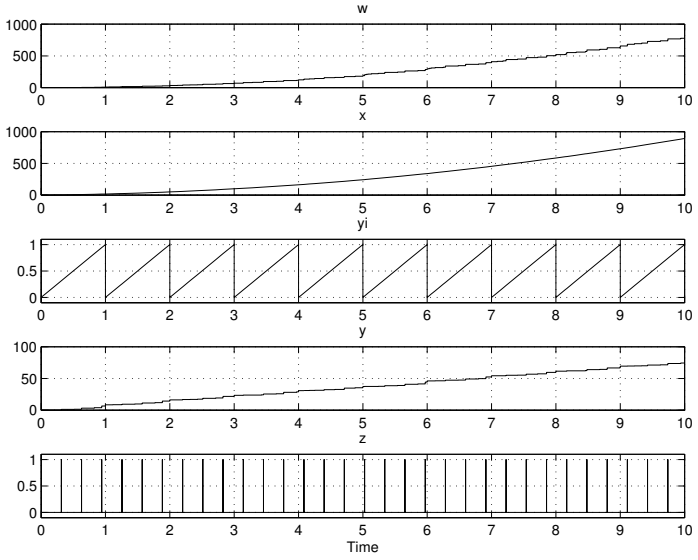
```
'Example4/Unit Delay: w' is discrete, yet is inheriting a continuous sample time; consider
replacing unit delay with a memory block. When a unit delay block inherits continuous
sample time, its behavior is the same as the memory block. Unit delay block's time delay
will not be fixed and could change with each time step. This might be unexpected behavior.
Normally, a unit delay block uses discrete sample time. You can disable this diagnostic by
setting the 'Discrete used as continuous' diagnostic to 'none' in the Sample Time group on
the Diagnostics pane of the Configuration Parameters dialog box.
```

The *discrete used as continuous* setting must be changed from 'error' to 'warning' before the model can be simulated. Even then, there is another warning that explicit conversions are required between the discrete and continuous blocks:

```
Warning: The configuration of the Unit Delay 'Example4/Unit Delay: w' is incorrect for
handling a rate transition. Consider using the Rate Transition block to handle the data
transfer between rates. Alternatively, you can control the diagnostic action for
unspecified rate transitions on the Sample Time Diagnostics pane of the Configuration
Parameters dialog box.
```



(a) Sine Wave frequency = 1 rad/s



(b) Sine Wave frequency = 10 rad/s

Figure A.6: A 'badly-typed' Simulink Example: results

t	w	x	yi	y	z
0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	1.000000000000000
0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000
0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000
0.000200950914521	0.000000000000000	0.000401901829042	0.000200950914521	0.000000000000000	0.000000000000000
0.000401901829042	0.000200950914521	0.001004754572604	0.000401901829042	0.000200950914521	0.000000000000000
0.000803803658083	0.000602852743562	0.002612361888770	0.000803803658083	0.000602852743562	0.000000000000000
0.001607607316166	0.001406656401645	0.006631380179185	0.001607607316166	0.001406656401645	0.000000000000000
0.003215214632332	0.003014263717812	0.016277024076182	0.003215214632332	0.003014263717812	0.000000000000000
0.006430429264665	0.006229478350144	0.038783526502508	0.006430429264665	0.006229478350144	0.000000000000000
0.012860858529329	0.012659907614808	0.090226960619824	0.012860858529329	0.012659907614808	0.000000000000000
0.025721717058658	0.025520766144137	0.205974687383786	0.025721717058658	0.025520766144137	0.000000000000000
0.051443434117316	0.051242483202795	0.463191857970367	0.051443434117316	0.051242483202795	0.000000000000000
0.102886868234632	0.102685917320112	1.029069633260845	0.102886868234632	0.102685917320112	0.000000000000000
0.205773736469265	0.205572785554744	2.263712052076434	0.205773736469265	0.205572785554744	0.000000000000000
0.405773736469265	0.411346522024009	4.863712052076434	0.405773736469265	0.411346522024009	0.000000000000000
0.605773736469265	0.817120258493274	7.663712052076435	0.605773736469265	0.817120258493274	0.000000000000000
0.805773736469265	1.422893994962538	10.663712052076436	0.805773736469265	1.422893994962538	0.000000000000000
1.000000000000000	2.228667731431803	13.771332268568196	1.000000000000000	2.228667731431803	0.000000000000000
1.000000000000014	3.228667731431803	13.771332268568424	0.000000000000000	3.228667731431803	0.000000000000000
1.200000000000014	4.228667731431817	17.371332268568423	0.200000000000000	3.228667731431803	0.000000000000000
1.400000000000014	5.428667731431831	21.171332268568424	0.400000000000000	3.428667731431803	0.000000000000000
1.600000000000014	6.828667731431845	25.171332268568424	0.600000000000000	3.828667731431803	0.000000000000000
1.800000000000014	8.428667731431858	29.371332268568423	0.800000000000000	4.428667731431803	0.000000000000000
2.000000000000014	10.228667731431871	33.771332268568422	1.000000000000000	5.228667731431803	0.000000000000000
2.000000000000045	12.228667731431885	33.771332268569140	0.000000000000000	6.228667731431803	0.000000000000000
2.200000000000045	14.228667731431930	38.571332268569137	0.200000000000000	6.228667731431803	0.000000000000000
2.400000000000046	16.428667731431975	43.571332268569137	0.400000000000000	6.428667731431803	0.000000000000000
2.600000000000046	18.828667731432020	48.771332268569140	0.600000000000000	6.828667731431803	0.000000000000000
2.800000000000046	21.428667731432064	54.171332268569138	0.800000000000000	7.428667731431803	0.000000000000000
3.000000000000004	24.228667731432111	59.771332268567946	0.999999999999957	8.228667731431804	0.000000000000000
⋮	⋮	⋮			

(a) Sine Wave frequency = 1 rad/s

t	w	x	yi	y	z
0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	1.000000000000000
0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000
0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000
0.000200950914521	0.000000000000000	0.000401901829042	0.000200950914521	0.000000000000000	0.000000000000000
0.000401901829042	0.000200950914521	0.001004754572604	0.000401901829042	0.000200950914521	0.000000000000000
0.000803803658083	0.000602852743562	0.002612361888770	0.000803803658083	0.000602852743562	0.000000000000000
0.001607607316166	0.001406656401645	0.006631380179185	0.001607607316166	0.001406656401645	0.000000000000000
0.003215214632332	0.003014263717812	0.016277024076182	0.003215214632332	0.003014263717812	0.000000000000000
0.006430429264665	0.006229478350144	0.038783526502508	0.006430429264665	0.006229478350144	0.000000000000000
0.012860858529329	0.012659907614808	0.090226960619824	0.012860858529329	0.012659907614808	0.000000000000000
0.025721717058658	0.025520766144137	0.205974687383786	0.025721717058658	0.025520766144137	0.000000000000000
0.051443434117316	0.051242483202795	0.463191857970367	0.051443434117316	0.051242483202795	0.000000000000000
0.102886868234632	0.102685917320112	1.029069633260845	0.102886868234632	0.102685917320112	0.000000000000000
0.205773736469265	0.205572785554744	2.263712052076434	0.205773736469265	0.205572785554744	0.000000000000000
0.314159265358979	0.411346522024009	3.672723927642722	0.314159265358979	0.411346522024009	0.000000000000000
0.314159265358985	0.725505787382988	3.672723927642797	0.314159265358985	0.725505787382988	1.000000000000000
0.314159265358992	1.039665052741973	3.672723927642907	0.314159265358992	1.039665052741973	0.000000000000000
0.514159265358992	1.353824318100966	6.872723927642907	0.514159265358992	1.353824318100966	0.000000000000000
0.628318530717959	1.867983583459958	8.813431438745333	0.628318530717959	1.867983583459958	0.000000000000000
0.628318530717969	2.496302114177917	8.813431438745503	0.628318530717969	2.496302114177917	1.000000000000000
0.628318530717980	3.124620644895885	8.813431438745727	0.628318530717980	3.124620644895885	0.000000000000000
0.828318530717980	3.752939175613866	12.813431438745727	0.828318530717980	3.752939175613866	0.000000000000000
0.942477796076923	4.581257706331846	15.210776011283528	0.942477796076923	4.581257706331846	0.000000000000000
0.942477796076938	5.523735502408769	15.210776011283837	0.942477796076938	5.523735502408769	1.000000000000000
0.942477796076954	6.466213298485707	15.210776011284210	0.942477796076954	6.466213298485707	0.000000000000000
1.000000000000000	7.408691094562661	16.591308905437309	1.000000000000000	7.408691094562661	0.000000000000000
1.000000000000016	8.408691094562661	16.591308905437696	0.000000000000000	8.408691094562661	0.000000000000000
1.057522203923062	9.408691094562677	18.086886207436887	0.057522203923046	8.408691094562661	0.000000000000000
1.115044407846108	10.466213298485739	19.639985713359124	0.115044407846092	8.466213298485707	0.000000000000000
1.230088815692200	11.581257706331847	22.861229133049690	0.230088815692183	8.581257706331799	0.000000000000000
⋮	⋮	⋮			

(b) Sine Wave frequency = 10 rad/s

Table A.9: A 'badly-typed' Simulink example: result detail

The simulation is performed anyway, and the result with a Sine Wave frequency of 1 rad/s is shown in Figure A.6a, and the result with a frequency of 10 rad/s is shown in Figure A.6b. The only value that is, more or less, the same is that of yi, the integrator with reset. That the value of z differs between the two runs is expected: increasing the frequency of the sine way increases the number of zero-crossings. All of the other values, however, finish larger in the run at 10 rad/s than in the run at 1 rad/s. Changing the behaviour of the Sine Wave block has changed the behaviour of unrelated components running in parallel!

This happens because the behaviour of a discrete component without explicit triggering depends on the discretization scheme chosen by the simulation engine. That is, the discrete components with inherited sample times that are connected to continuous components are triggered at every *major time step* of the simulation. When the number of zero-crossings is increased, by changing the sine wave frequency, there are more major time steps in the simulation and the unit delays accumulate samples at a higher rate. This can be seen in the detailed log extracts of Figure A.9.