A Hybrid Denotational Semantics for Hybrid Systems – Extended Version

Olivier Bouissou, Matthieu Martel

A Hybrid Denotational Semantics for Hybrid Systems

Olivier Bouissou and Matthieu Martel

[1] CEA LIST - Laboratoire MeASI
F-91191 Gif-sur-Yvette Cedex, France
Olivier.Bouissou@cea.fr
[2] ELIAUS-DALI Laboratory - Université de Perpignan Via Domitia
66860 Perpignan Cedex
Matthieu.Martel@univ-perp.fr

Abstract. In this article, we present a model and a denotational semantics for hybrid systems. Our model is designed to be used for the verification of large, existing embedded applications. The discrete part is modeled by a program written in an extension of an imperative language and the continuous part is modeled by differential equations. We give a denotational semantics to the continuous system inspired by what is usually done for the semantics of computer programs and then we show how it merges into the semantics of the whole system. The semantics of the continuous system is computed as the fix-point of a modified Picard operator which increases the information content at each step.

1 Introduction

The importance and usefulness of static analysis techniques [9] for software validation is no longer to be outlined. Their application to highly critical programs has become a major challenge for many industries. Such programs are usually automatically generated, imperative programs which are embedded into a heterogeneous system. They mostly behave as follows: they capture information from the physical environment via sensors, treat this information using numerical computations and consequently modify the environment via actuators. The analysis of such programs requires either to over-approximate the physical environment, which often leads to an imprecise analysis, or to analyze the whole hybrid system made of the continuous environment and the discrete program [7, 17]. We use this approach.

The analysis of hybrid systems requires as a starting point a formal description of their behavior. Thus, we need to give a coherent interpretation of both the discrete and the continuous subsystems. The formalization of a continuous system using the same notions as for a computer program is already a challenge of its own. The continuous variables move along a continuous function of the real time while the discrete system is defined, in a denotational semantics approach, as a function between discrete environments [32]. In this article, we propose a

formalism for modeling hybrid systems together with a description of their behavior as a *hybrid denotational semantics*: the evolution of the hybrid system is a function between hybrid environments (i.e. containing a discrete *and* a continuous part) and this function is computed as the least fix-point of a sequence of approximations.

Our model for hybrid systems is designed for an implementation level and ensures a clear separation of the discrete and the continuous subsystems. They are actually modeled in two different formalisms (see Section 3.1 and 3.2) which allows of the analysis of one program within various environments for example. Despite this heterogeneity, we give a unique description of the behavior of the hybrid system. To build the hybrid semantics of the whole system, we first give a coherent description of both parts. As a first step, we suppose that the discrete part is completely determined and we give a semantics $[\![\kappa]\!]$ for the continuous part (Section 4). It is computed as the fix-point of an operator Γ which acts on partially defined functions and we show that this fix-point is actually the limit of Tarski's iterates [30]. Then, we suppose that the continuous part is perfectly known and we define a denotational semantics $[\![\Delta]\!]$ for the discrete part only (Section 5). For this, we add to the standard semantics of imperative languages denotations for some hybrid actions that represent sensors and actuators. Finally, we define the semantics $[\![\Omega]\!]^{\mathcal{H}}$ of the hybrid systems as a combination of both discrete and continuous semantics (Section 6).

1.1 Running Example

We will illustrate this article with a simplified version of the well-known two tanks problem [22]. The system consists of *one* water tank (see Figure 1.1) which is filled by a constant flow i and which has two evacuation tubes: one at the bottom and one at height h. The evacuation tube at the bottom has a valve v which can be open or closed. The continuous system is the height x of the water in the tank, whose evolution is governed by the ordinary differential equation of Figure 1(b) and the discrete part is a controller whose goal is to maintain x between safe bounds by closing/opening the valve. A model of this system using hybrid automata requires 4 states: one for each possible equation for x. The switches between these 4 states have two natures: either they are directed by the discrete system (the valve is turned off/on), or by the continuous system (x becomes bigger/lower than h).

1.2 Related Work

The modeling of hybrid systems with hybrid automata was initiated by Henzinger [19] as an extension of timed automata [2]. They are finite state automata to which we add at each location a *flow equation* describing the continuous dynamics at this point. Their operational semantics was introduced in the early papers and their analysis using model checking techniques has been well studied, either for linear [1] or non-linear [20] cases. A denotational semantics for these models was recently proposed by Edalat [14] and proved to be equivalent with

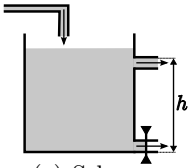
the operational semantics. Since the first results, many models for hybrid systems and verification methods were proposed. These include hybrid process algebra like HyPa [11] or Hybrid Chi [31]. Meanwhile, Hybrid-CC [18] introduced hybrid components to the concurrent constraints theory. All these models are generally used as high level abstract formalisms to reason about the principles of hybrid systems. However, when the verification of industrial size, critical systems is at stake, they are not fully sufficient. First, for safety reasons, the analysis of the embedded source code is always necessary. Secondly, for industrial size problems, it is necessary to have a clear distinction between discrete and continuous states to allow the modeling process of the both parts to be executed by different engineers. Most of the models we cited are not well-suited for these requirements, although some advances have been made for the separation issue [3].

On the other hand, hybrid systems may be seen as switched, dynamical systems, i.e. continuous systems whose dynamics changes in some regions [23]. In this formalism, the discrete part is very limited and the question of the stability of the system [6] is of great importance, while the verification is often done via successive simulations [15]. The behavior of such systems is however not always formally defined, although some advances in that direction have recently been made [33].

The main difficulty in the formalization of hybrid systems is to give a coherent interpretation of the continuous and the discrete parts. Edalat et al. proposed a formalization of differential calculus and of the solution of differential equations in the theory of Scott domains, both for the mono-variate [12] and multi-variate [13] cases. Their theory was one of the main starting points for this work and we use some of their results here.

1.3 Notations

In the rest of the article, we will write $\mathbb{R}$ for the set of real numbers, $\mathbb{R}_+$ for the set of non-negative real numbers and $\mathbb{N}$ denotes the natural integers. For two elements $a \leq b$ of a domain, we write $[a, b]$ the interval containing all elements $a \leq x \leq b$. The set of all intervals over $\mathbb{R}$ (resp. $\mathbb{R}_+$) will be denoted $\mathbb{I}(\mathbb{R})$ (resp. $\mathbb{I}(\mathbb{R}_+)$). For $i \in \mathbb{I}(\mathbb{R})$, we write i^- is lower bound and i^+ its upper bound. We define its width $w(i) = i^+ - i^-$ and its midpoint $mid(i) = \frac{i^+ + i^-}{2}$. We will denote $\mathbb{F}$ the set of floatting point numbers and for a given real number $x \in \mathbb{R}$,



(a) Scheme.

$$\dot{x} = \begin{cases} i - k_1\sqrt{x} - k_2\sqrt{x-h} & \text{if } x \geq h \text{ and } v \text{ open} \\ i - k_2\sqrt{x-h} & \text{if } x \geq h \text{ and } v \text{ closed} \\ i - k_1\sqrt{x} & \text{if } x \leq h \text{ and } v \text{ open} \\ i & \text{otherwise} \end{cases}$$

(b) Continuous System.

Fig. 1.1. One Tank Example.

we will write $\uparrow_\sim (x)$ the closest floatting point number. This casting operation from real to floating point numbers will be necessary for the semantics of sensors that transform a real variable (the physical value) into a floating point one (the discrete measurement).

2 Basics of Differential Equations Theory

In this Section, we present the basic notions of differential equations theory that are necessary for the understanding of this paper. They are not much explained, we invite the curious reader to refer to any differential equations book (for example [21]) for more details.

Definition 1 (*Ordinary Differential Equation*). *Let y be a vector-valued function of the positive real variable t, $y : \mathbb{R}_+ \to \mathbb{R}^d$. Let $y^{(i)}$ be its i^{th} derivative in t. Then, an* ordinary differential equation *(ODE) of order n and dimension d is a relation of the form $F\big(y, y^{(1)}, \ldots, y^{(n-1)}, t\big) = y^{(n)}$. The properties of the function F define the properties of the solution of the ODE. When the function F does not depend on t, the ODE is said to be* autonomous.
ODEs of dimension 1 are particularly important and we are used to writing them $\dot{y} = F(y, t)$, where $\dot{y} = y^{(1)}$.

Proposition 1. *Every differential equation of dimension d and order n is equivalent to an ODE of order 1 and dimension $d + n$.*

Proof. We introduce an auxiliary variable y_i for the ith derivative of y, such that $\dot{y_i} = y_{i+1}$. Then, the ODE is equivalent to the ODE of order one $\dot{z} = g(z)$ where $z = (y, y_1, \ldots, y_n, t)^T$ and $g(z) = (y_1, \ldots, y_{n-1}, F(t, y, y_1, \ldots, y_{n-1}), t)^T$.

Thus, from now on, we will only consider ordinary differential equations of order 1. For the simplicity of the presentation, we will limit this paper to ODEs of dimension 1. All the results we present can be easily extended to ODEs of arbitrary dimension (and thus differential equations of any order). An *initial value problem* (IVP) is an ODE $\dot{y} = F(y, t)$ together with an initial condition $y(t_0) = y_0$. We write it:

$$\dot{y} = F(y, t), \quad y(t_0) = y_0 \tag{1}$$

Depending on the function F and the initial condition, the behavior of the implicitly defined function y varies a lot: it may exist on a small open interval around t_0, on the whole real line or not exist at all. Let us recall some results which give conditions for the existence and uniqueness of solutions to Equation (1).

Definition 2 (*Solution of an IVP*). *A* solution *to the IVP* (1) *on the (open, semi-open or closed) interval (a, b) that contains t_0 is a continuous, continuously differentiable function $f : (a, b) \to \mathbb{R}$ such that:*

$$\forall t \in (a, b),\ \dot{f}(t) = F(f(t), t) \quad \textit{and} \quad f(t_0) = y_0$$

Definition 3 (*Maximal solution of an ODE*). *If two solutions f_1, f_2 of the IVP* (1) *defined over two intervals $i_1 \subseteq i_2$ are equal at some $t \in i_1$, then they are equal for all $t \in i_1$. f_2 is then said to be an* extension *of f_1.*
A solution of Equation (1) *is said to be* maximal *if it cannot be extended.*

An important class of IVPs are the ones where F is (at least locally) Lipschitz in y. We recall that a function $F : \mathbb{R} \times \mathbb{R}_+ \to \mathbb{R}$ is globally α-Lipschitz in its first coordinate if for all $y_1, y_2 \in \mathbb{R}$, it holds that $|F(y_1, t) - F(y_2, t)| \leq \alpha |y_1 - y_2|$. So, Lipschitz functions are functions with a bounded growth rate. For such functions, the existence of solutions to Equation (1) is (at least locally) guaranteed.

Theorem 1 (Existence and uniqueness theorem). *If the function F is locally Lipschitz in y on a neighborhood of (y_0, t_0), then the IVP has a unique maximal solution around t_0. If F is globally Lipschitz on $\mathbb{R}$, then the IVP has a unique solution on $\mathbb{R}$.*

Theorem 1 is very important as Lipschitz functions are common. The most interesting part of the theorem is without doubt its proof, as it proves the existence and uniqueness of the solution using a fix-point argument (in the sense of Banach's fix-point theory). The solution is constructed as the fix-point of the *Picard operator* [21] defined for an interval $I \in \mathbb{I}(\mathbb{R})$, a continuous function F and an initial condition $y_0 \in \mathbb{R}$ by:

$$P_I\big(F, y_0\big) : \begin{cases} \mathcal{C}^0\big(I\big) \to \mathcal{C}^0\big(I\big) \\ f \ \mapsto \lambda x. y_0 + \int_{I^-}^{x} F(f(s), s) ds \end{cases} \tag{2}$$

where $\mathcal{C}^0\big(I\big)$ is the set of continuous, real-valued functions defined on the interval I. It can be shown that if we chose carefully the interval I, then we can build a Banach space for which the Picard operator is contracting. It thus has a least fix-point. This fix-point is the solution of the IVP (1) on I, as shown by the following proposition:

Proposition 2. *A continuous, differentiable function f on (a, b) is a solution to the IVP* (1) *if and only if it satisfies the* integral equation*:*

$$\forall t \in (a, b),\ f(t) = y_0 + \int_a^t F(f(s), s) ds. \tag{3}$$

Theorem 2 (Convergence of the Picard iterates). *If F is globally Lipschitz on $\mathbb{R}$, the* Picard iterates *defined by:*

$$f_0 \in \mathcal{C}^0([a, b]), \quad f_{n+1} = P_{[a,b]}\big(F, y_0\big)(f_n)$$

converge uniformly on $[a, b]$, for all intervals $[a, b]$ of $\mathbb{R}$. So, whatever the choice of the first function f_0, if we iteratively compute $f_{n+1} = P_{[a,b]}\big(F, y_0\big)(f_n)$, the sequence converges toward the solution of (1) *on $[a, b]$.*

Proposition 2 gives a characterization of solutions of the IVP as solutions of a fix-point equation in the sense of Banach's fix-point theory. Theorem 2 shows

that this fix-point can be computed as the limit of the iterates of a contracting operator. Let us note that the convergence of the iterates is very quick: if we start two iterations with two functions f_0 and $\tilde{f}_0$, then we have for all $n \in \mathbb{N}$ and $t \in [a, b]$:

$$|f_n(t) - \tilde{f}_n(t)| \leq K.\frac{k^n(t-t_0)^n}{n!}$$

where K is a constant and k is the Lipschitz-constant of F.

3 Our Model for Hybrid Systems

In this Section, we present our model for hybrid systems. This model is designed for large, embedded, safety critical applications and is designed for being used at an implementation level, i.e. we want to analyze the programs that will actually be embedded. These programs continuously interact with their environment via sensors and actuators, the activity of which has thus to be carefully modeled. Our objectives for this new model are the following:

1. The discrete part remains close to existing embedded software.
2. The action of sensors and actuators is clearly identified.
3. Continuous and discrete systems are modeled separately.

The last requirement is necessary for two reasons. First, if we want to analyze the behavior of a controller in different physical environments without having to rewrite the entire system, the distinction between the plant (i.e. the discrete part) and the environment must be clear. Secondly, for existing industrial applications, the discrete part (i.e. the program) is already written, so we want a description of the hybrid systems that can use this program "as it is". An obvious solution would consist of building a cartesian product between the continuous states and the states of the program. For combinatorial reasons, our approach consists of first describing a model for continuous subsystems (Section 3.1) and then a model for discrete subsystems (Section 3.2).

3.1 Model for the Continuous Subsystem

The continuous part contains variables evolving continuously with time such as the water height in the tank or the temperature of the air. Their evolution is usually described by an *ordinary differential equation*; for example, the temperature x of a room with a heater is given by an ODE like $\dot{x} = 5 - 0.1x$. Let κ be the continuous model, its expressiveness depends on the set of functions F that we allow inside Equation (1). We need to capture two phenomena: a change in the dynamics due to the environment itself and a change due to the discrete program. The first arises for example when the water passes above the tube (see Equation (4)) while the second appears when the valve is closed.

To capture the changes induced by the discrete system via actuators, we allow the function F to have *boolean parameters*. Thus, we will have $F = F(y, t, \boldsymbol{k})$, where $\boldsymbol{k}$ is a vector of boolean values. We write $F_k(y, t) = F(y, t, k)$ for every

possible value of k. To capture the changes induced by the environment itself, we let each F_k be a continuous, piecewise Lipschitz function, as on Figure 3.1. Thus, F_k behaves differently in different regions of the space, which is precisely the kind of changes we wanted to model. We recall that a function g is piecewise Lipschitz if there exist finitely many real numbers $x_0 < x_1 < \cdots < x_n$ such that the restriction of g to $[x_i, x_{i+1}]$ is Lipschitz. The theory of differential equations remain unchanged with such functions, except that the solutions are now continuous but only piecewise differentiable functions. Especially, the Picard iterates still converge uniformly on every interval.

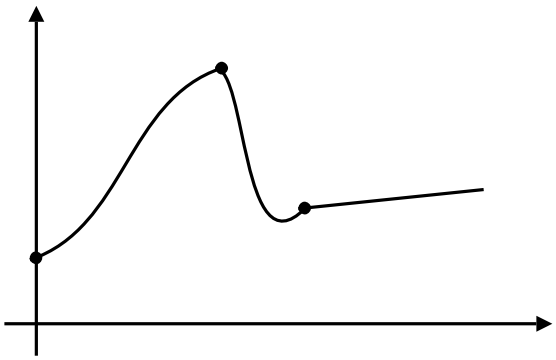


Fig. 3.1. Admissible function for an ODE.

The continuous model κ is a triple $\kappa = (F, \left(F_k\right)_{k\in\boldsymbol{k}}, y_0)$ where $\left(F_k\right)_{k\in\boldsymbol{k}}$ is the set of possible modes. We write $F_{\boldsymbol{k}}$ for $\left(F_k\right)_{k\in\boldsymbol{k}}$. F is the function defining the IVP and is such that there exists $t_0 < t_1 < \cdots < t_n < \ldots$ such that the restriction of F to $[t_i, t_{i+1}]$ is equal to one of the F_k. The model representing the evolution of the liquid height in the one-tank system is $(F, \{F_0, F_1\}, y_0)$ where (F_0, F_1) are given by Equation (4).

$$F_k(x) = \begin{cases} i - k * k_1\sqrt{x} - k_2\sqrt{x-h} \text{ if } & x \geq h \\ i - k * k_1\sqrt{x} & \text{otherwise} \end{cases} \tag{4}$$

Let us make a parallel between this model and a computer program as the representation of a dynamical system. A source code is a model for a discrete dynamical system: the compiled binary program. Every instruction explains how the state of the system varies from instant t (the current instruction) to instant $t+1$ (the next one). For a continuous function, the ODE plays that role: it links the state of the system at time $t+dt$ with the state of the system at time t via its derivatives. So, it is natural to consider an ODE as a program for continuously varying systems.

$$
\begin{aligned}
\mathit{stmt} &:= \text{v} = \mathit{exp} \mid \textbf{if}(\mathit{bool})\ \textbf{then}\ \mathit{stmt}\ \textbf{else}\mathit{stmt} \mid \textbf{while}(\mathit{bool})\ \mathit{stmt} \\
&\quad \mid \mathit{stmt};\mathit{stmt} \mid \mathit{hyb\_stmt} \\
\mathit{exp} &:= \text{c} \mid \mathit{exp}+\mathit{exp} \mid \mathit{exp}\text{-}\mathit{exp} \mid \mathit{exp}^{*}\mathit{exp}\ \ldots \\
\mathit{bool} &:= \text{v}<\text{exp} \mid \text{v}>\text{exp} \mid \mathit{bool} \vee \mathit{bool} \mid \ldots \\
\mathit{hyb\_stmt} &:= \textbf{sens}.\text{y?x} \mid \textbf{act}.\text{k!c} \mid \textbf{wait}\ \text{c}
\end{aligned}
$$

Table 1. Statements for the discrete system.

3.2 Model for the Discrete Subsystem

We want the discrete model Δ to remain close to existing embedded software in order to be used for large critical applications. We thus start with a set of standard statements which are common to any imperative language (*stmt* in Table 1). We have assignements, **if** statements, **while** loops, arithmetic and boolean expressions. This core language can be extended to more complex statements without perturbing the semantics of the hybrid system as they represent purely discrete actions.

In addition to standard statements, we have three hybrid actions:

- **sens** to model the sensors. The action of **sens.y?x** is to bind the variable **x** to the value of the continuous variable **y** at the time the action is executed,
- **act** to model the actuators. The action of **act.k!c** is to change the continuous dynamics by choosing the function F_c among all the possible dynamics $\{F_k\}$,
- **wait** to model the passing of time. We suppose that all discrete and hybrid actions are instantaneous and we model the fact that they were not by explicitly adding these **wait** statements. The effect of **wait c** is to move time forward by **c** seconds.

This formalism is very close to existing imperative languages and, in most cases, the generated programs already contain, as comments, the hybrid statements. For example, the loops of embedded industrial programs are usually precisely cadenced and we often see in the codes comments indicating their frequency such as "this loop runs at 8kHz". Thus, adding a **wait** command at the end of the loop to model its cadence is easy. Clearly, every program that uses data sampling *must* know the sampling rate to ensure the meaning of the data given by the sensor. It is thus reasonable to assume that we know the cadence of the incoming data.

```
1  int main() {
2    sensor x;          // sensors declaration
3    actuator k;        // actuators declaration
4    while (true) {
5      sens.x?h;
6      if (h>h_max)
7        act.k!1; throw( alarm );
8      h_in_2_secs = anticipate(h);
9      if (h_in_2_secs > h_max)
10       act.k!1;
11     wait (0.01);    // delay action
12   }
13 }
```

Listing 1. Controller for a one-tank system.

Using this syntax, we can write a controller for the one tank system that measures the height x of the water with a sensor and open the valve if x is too high (see Listing 1). We suppose that the closing operation takes two seconds, so the controller must predict (using for example linear interpolation) the height of the water two seconds later (this is the role of the function `anticipate`) and starts the closing if this predicted value is too high.

This model for hybrid systems conforms to our three requirements:

1. Modeling the discrete system requires few additions to existing programs.
2. Sensors and actuators are clearly modeled.
3. Continuous and discrete systems are modeled using two different formalisms.

Moreover, we designed it so that it prohibits impossible phenomena like contuous state jumps or Zeno effects. Actually, time is driven by the discrete subsystem through the **wait** statements, thus there must exist a minimum time between two mode switchings (because the program is finite), which prohibits Zeno phenomena.

Let us now give a formal, denotational semantics for this model of hybrid systems. The semantics is defined separately for the continuous (Section 4) and the discrete (Section 5) systems and both are then joined to define the semantics for the complete, hybrid system. We define the semantics of the continuous system with the assumption that the discrete system is fully known and vice-versa for the discrete part. This allows of several simplifications when specifying both parts, those simplifications are then removed by the semantics of the whole system.

4 Continuous Semantics

In this section, we give a formal, denotational semantics of the continuous model. Let us recall that the continuous part of an hybrid system is represented as $\kappa = \big(F, F_{\boldsymbol{k}}, y_0\big)$ where $F_{\boldsymbol{k}}$ is a family of piecewise Lipschitz continuous functions and $y_0 \in \mathbb{R}$ is the initial condition (we suppose $t_0 = 0$). Each F_k is supposed

to be globally α-Lipschitz on $\mathbb{R}$, so that there exists a unique maximal solution on $\mathbb{R}$ (as defined in Section 2) to each ODE $\dot{y} = F_k(y,t)$. Our goal is to have a coherent formalism for describing both the continuous and the discrete part of the hybrid system. We first give the intuition for this semantics (Section 4.1) and then we describe in detail the lattice structure that we manipulate (Section 4.2) and the computation of the semantics as a fix-point (Section 4.3).

4.1 Intuitive Idea of the Semantics

In an analogy with standard denotational semantics, we want to express the semantics of κ as a function mapping an initial environment to a final value. If we know the behavior of the discrete part of the system, we know the times at which the parameters $k \in \boldsymbol{k}$ switch. Thus, we know completely the function F and the semantics of κ maps an initial value to the semantics of the IVP

$$\dot{y} = F(y,t),\ y(0) = y_0 \tag{5}$$

Basically, the semantics of the IVP is its maximal solution, i.e. a piecewise differentiable, continuous function $y : \mathbb{R}_+ \to \mathbb{R}$ which satisfies (5). Thus, the semantics of κ is a function $[\![\kappa]\!]$ mapping an initial *environment* (i.e. the initially available information y) to the solution of the IVP. This is very close to the denotational semantics of a program which is a function mapping the environment before the execution of the program with the environment at the end.

This is very comparable to the semantics of a computer program. Actually, the most concrete semantics of a computer program is a (potentially infinite) sequence $\big((l_i, v_i)\big)$ where $l_i \in L$ is a set of unique labels and v_i is the value of the variables of the program at the label l_i. A label represent a single point in the execution of the program and are linked to the usual notion of control points: they represent the control points once all the loops have been unrolled and the different instructions differentiated. It is thus a function from the *discrete* set of labels L to the domain V of the variable values. The main aspects of this formalism are:

1. clear separation of all the control points.
2. there is a countable number of labels.
3. the value of the variable at some label l_i only depends on the value of the variables at the previous labels.

The computation of the sequence $\big((l_i, v_i)\big)$ generally requires the computation of the least fix point of some monotone operator on a complete lattice [32]. This fix-point computation corresponds to upgrading the already computed values, i.e. updating the available information. Generally, the operator takes the already computed sequence $\big((l_0, v_0), \cdots, (l_n, v_n)\big)$ and extends it to the right, i.e. returns $\big((l_0, v_0), \cdots, (l_{n+1}, v_{n+1})\big)$.

The semantics of a continuous model κ, i.e. the solution of the corresponding IVP, is a function from an infinite set of labels ($\mathbb{R}_+$) to the domain $\mathbb{R}$. It thus

respects points 1 and 3 of the trace semantics of a computer program. Clearly, point 2 is no longer valid as we jumped from a discrete to a continuous world.

The computation of $[\![\kappa]\!](y)$ requires the computation of a fix-point, in the sense of Banach's fix-point theory, as shown by Theorem 2. We translate this fix-point computation into Tarski's fix-point theory and compute $[\![\kappa]\!](y)$ as the fix-point of an operator Γ. Then, we prove this is the supremum of the iterates $\Gamma^n(\bot)$. Γ is defined on elements with partial information and it updates them by increasing their information content. Our notion of partial information is the following: a function has only partial information if it is defined on a finite interval $[0, X]$ for some $X \in \mathbb{R}_+$ and its value at each point is bounded, i.e. is an interval. Thus, the maximal elements are the real-valued functions defined on $\mathbb{R}_+$ and our semantics will construct one of these (the solution of (5)) as the limit of an approximations sequence, each approximation being a partially defined, interval-valued function.

4.2 The Lattice of Interval-Valued Functions

We now define the set of partially defined, interval-valued functions, i.e. continuous functions with only partial information. We also define an order and shows that this order provides a lattice structure.

Definition 4 (*Partial, interval-valued functions*). *Let X be a non negative real number. $\mathcal{IF}_X$ is the set of interval-valued functions defined on $[0, X]$:*

$$\mathcal{IF}_X = \{f : [0, X] \to \mathbb{I}(\mathbb{R})\}$$

For such a function, we define its upper f^+ *and* lower f^- *functions as the two real-valued functions such that* $\forall x \in [0, X],\ f(x) = [f^-(x), f^+(x)]$.

When f^- (respectively f^+) is right-continuous (respectively left-continuous), f is (Scott) continuous and write $\mathcal{IF}^0_X$ the set of all *continuous*, partial, interval-valued functions. We recall that a function f is right-continuous if, for all x, we have $\lim_{t \to x, t > x} f(t) = f(x)$, i.e. if t tends toward x from above, $f(t)$ tends toward $f(x)$; the left-continuity is the opposite. We provide the set $\mathcal{IF}^0_X$ with a *complete partial order* structure with the pointwise reverse order:

$$f \sqsubseteq_X g \Leftrightarrow \forall x \in [0, X],\ g(x) \subseteq f(x) \tag{6}$$

This order means that at every point in $[0, X]$, g is more informative than f. Clearly, $(\mathcal{IF}^0_X, \sqsubseteq_X)$ is a CPO (actually, it is a continuous Scott domain [12]). The left-(resp. right) continuity of f^+ (resp. f^-) is a necessary condition for f to be Scott-continuous [12] *and* for $\mathcal{IF}^0_X$ to be a CPO; consider for example the piecewise linear functions $f_n \in \mathcal{IF}^0_1$ defined by $f_n(x) = [0, 1]$ if $x \in [0, \frac{1}{2}]$, $f_n(x) = [0, 1 - \frac{n}{2}(x - \frac{1}{2})]$ if $x \in [\frac{1}{2}, \frac{1}{2} + \frac{1}{n}]$ and $f_n(x) = [0, \frac{1}{2}]$ otherwise. Clearly, the supremum $f = \bigsqcup_n f_n$ is not continuous in $\frac{1}{2}$, while each f_n is. The right-continuity condition imposes that $f^+(x) = 1$ for $x \in [0, \frac{1}{2}[$ and $f^+(x) = \frac{1}{2}$ for $x \in [\frac{1}{2}, 1]$. $\mathcal{IF}^0_\infty$ is the natural extension of $\mathcal{IF}^0_X$ to the case of interval-valued

functions on $\mathbb{R}_+$. We now build the set of interval functions defined over arbitrary intervals of $\mathbb{R}$.

Definition 5 (*Arbitrary long, interval-valued functions*). *The set of all continuous, interval-valued functions defined over any interval* $[0, X]$ *is* $\mathcal{D}^0 = \left(\bigcup_{X \in \mathbb{R}_+} \mathcal{IF}^0_X\right) \cup \mathcal{IF}^0_\infty$. *For* $f \in \mathcal{D}^0$, *we note* X_f *the upper bound of its domain :* $X_f = sup(dom(f))$. *The value* X_f *is the maximum time until which* f *is defined; if* f *is defined on* $\mathbb{R}_+$, *then* $X_f = \infty$.

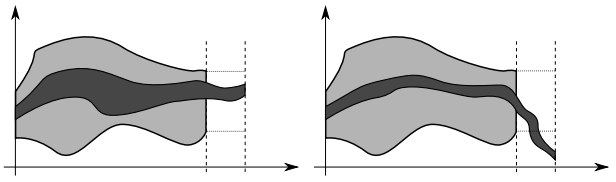
Note that for all $X \geq 0$, the set of continuous, real-valued functions $\mathcal{C}^0([0, X])$ is embedded into $\mathcal{D}^0$ by the function $\gamma : f \mapsto \lambda x.[f(x), f(x)]$. Thus, we will identify a map $f \in \mathcal{C}^0([0, X])$ with the map $\lambda x.[f(x), f(x)]$ and write $f \in \mathcal{D}^0$. We extend the order $\sqsubseteq_X$ to $\mathcal{D}^0$ by requiring that g is greater than f if it is more precise on a longer interval than f:

$$f \sqsubseteq g \Leftrightarrow X_f \leq X_g \text{ and } f \sqsubseteq_{X_f} g_{|_{[0,X_f]}} \text{ and } \forall x \in [X_f, X_g],\ g(x) \in f(X_f) \qquad (7)$$

where $g_{|_{[0,X_f]}}$ denotes the restriction of g to $[0, X_f]$. This order may be seen as a flexible prefix order: g is greater than f if it extends it, in a coherent way. Figure 4.1 gives an example of comparable functions (left, the dark one being bigger than the light one) and an example of incomparable functions (right). The third hypothesis in Equation (7) states that g remains bounded by the last value of f on $[X_f, X_g]$. It is necessary for $\mathcal{D}^0$ to be a CPO: in any increasing chain f_n, the functions f_n^+ and f_n^- are bounded, thus (f_n^-) is a bounded increasing sequence (with respect to the pointwise order for real-valued functions), so it has a limit f^-. Equivalently, (f_n^+) has a limit f^+, which proves the existence of $\bigsqcup_n f_n = [f^-, f^+]$.

We extend $(\mathcal{D}^0, \sqsubseteq)$ with a bottom $\bot$ and a top $\top$ element such that $\forall f \in \mathcal{D}^0,\ \bot \sqsubseteq f \sqsubseteq \top$. We also define the join and meet operators $\sqcup$ and $\sqcap$ as follows. Let $f, g \in \mathcal{D}^0$, with $X_f \leq X_g$. Then, $f \sqcup g \in \mathcal{IF}^0_{X_g}$ and $f \sqcap g \in \mathcal{IF}^0_{X_f}$ are defined by:

$$f \sqcup g(x) = \begin{cases} f(x) \cap g(x) & \text{if } x \in [0, X_f] \\ f(X) \cap g(x) & \text{otherwise} \end{cases}$$
$$f \sqcap g(x) = f(x) \cup g(x)$$



(a) Comparable functions. (b) Incomparable functions.

Fig. 4.1. Order on partially defined functions.

This definition of $f \sqcup g$ supposes that $\forall x \in [0, X_f],\ f(x) \cap g(x) \neq \emptyset$. If this is not true, $f \sqcup g = \top$. Figure 4.2 shows the effect of $\sqcup$: the join of the two gray functions (left) is the dark one (right).

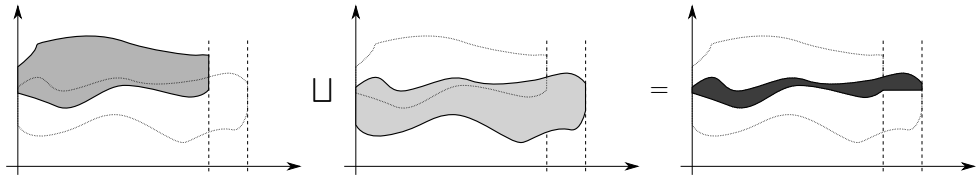


Fig. 4.2. Join on $\mathcal{D}^0$

Proposition 3. $(\mathcal{D}^0, \sqsubseteq, \top, \bot, \sqcup, \sqcap)$ *is a continuous lattice.*

Proof. Each underlying set $\mathcal{IF}^0_X$ is a continuous Scott domain (see [12] for a complete proof of that). Our extended order was specially designed so that it preserves this structure (see discussion of Equation (7)). Thus, $\mathcal{D}^0$ is a directed CPO. We need to prove that $\sqcup$ and $\sqcap$ do define a join and a meet operator to prove that $\mathcal{D}^0$ is a lattice.
We prove that $\sqcup$ is a join. Let $f, g \in \mathcal{D}^0$ with $X_f \leq X_g$, and let $h = f \sqcup g$. Let us suppose that $h \neq \top$. We have: $X_h = max(X_f, X_g)$, so $X_h \geq X_f$. Let now $x \in [0, X_h]$, then if $x \in [0, X_f]$, $h(x) = f(x) \cap g(x)$, so $h(x) \subseteq f(x)$, and if $x \in [X_f, X_h]$, $h(x) = f(X_f) \cap g(x)$, so $h(x) \subseteq f(X_f)$. So, we have $f \sqsubseteq h$. Equivalently, $g \sqsubseteq h$. Let now h' be such that $f \sqsubseteq h'$ and $g \sqsubseteq h'$, with $h' \neq \top$. We have $X_{h'} \geq X_f$ and $X_{h'} \geq X_g$, so $X_{h'} \geq X_h$. Let now $x \in [0, X_{h'}]$: if $x \in [0, X_f]$, then $h'(x) \in f(x)$ and $h'(x) \in g(x)$ so $h'(x) \in h(x)$; if $x \in [X_f, X_g]$, $h'(x) \in f(X_f)$ and $h'(x) \in g(x)$ so $h'(x) \in h(x)$; if $x \in [X_g, X_{h'}]$, $h'(x) \in f(X_f)$ and $h'(x) \in g(X_g)$, so $h'(x) \in f(X_f) \cap g(X_g) = h(X_h)$. So, we have $h \sqsubseteq h'$, so $\sqcup$ is a join.
Equivalently, we can prove that $\sqcap$ is a meet, so $\mathcal{D}^0$ is a lattice. □

Let us remark that $\mathcal{D}^0$ is a lattice and a CPO, so every increasing chain does have a supremum. It is however *not* a complete lattice as there exist infinite sequences without supremum. For example, let us consider the sequence of functions $\varphi_n \in \mathcal{IF}^0_{1-\frac{1}{n}}$ defined by $\varphi_n(x) = [-\frac{1}{1-x}, \frac{1}{1-x}]$. Clearly, this sequence does not have a supremum in $\mathcal{D}^0$ except $\top$, while there are infinitely many $f \in \mathcal{D}^0$ greater than f_n for all n (for example, the constant function with value 0). This lattice is a set of approximate functions, in the sense that a function $f \in \mathcal{D}^0$ only has partial information. We will extend and update these functions in order to get to the solution of Equation (5). We thus need to define some basic operations on $\mathcal{D}^0$ that adapt the classical operations on real-valued functions. The arithmetic operators $+, -, *, /$ are defined as an extension of the interval arithmetic. For $\odot \in \{+, -, *, /\}$ and $f, g \in \mathcal{IF}^0_X$, we define $f \odot g \in \mathcal{IF}^0_X$ as

$$\forall x \in [0, X],\ f \odot g(x) = \{y \odot z \mid y \in f(x) \text{ and } z \in g(x)\}$$

We next define the composition, primitive and width of functions in $\mathcal{D}^0$.

Definition 6 (*Function composition*). *The* composition *of a continuous, real-valued function $F : \mathbb{R} \to \mathbb{R}$ and a partial, interval-valued function $f \in \mathcal{IF}^0_X$ is the function $F \circ_X f \in \mathcal{IF}^0_X$ defined by:*

$$\forall x \in [0, X], (F \circ_X f)(x) = \{F(y) \ : \ y \in f(x)\}.$$

$F \circ_X f$ is well defined because F is continuous and $f(x)$ is an interval, so $F \circ f(x)$ is an interval for all x. We naturally extend the notion of function composition to $\mathcal{D}^0$ and define the operator $\circ$ as:

$$\forall F : \mathbb{R} \to \mathbb{R} \text{ and } f \in \mathcal{D}^0, \ F \circ f = F \circ_{X_f} f$$

Definition 7 (*Primitive*). *The* primitive *of a function $f \in \mathcal{IF}^0_X$ is $I_X(f) \in \mathcal{IF}^0_X$ defined by:*

$$\forall x \in [0, X], \ I_X(f)(x) = \left[\int_0^x f^-(s)ds, \int_0^x f^+(s)ds\right]$$

Again, the primitive operator is extended to $\mathcal{D}^0$ straightforwardly: for $f \in \mathcal{D}^0$, we set $I(f) = I_{X_f}(f)$

Definition 8 (*Width*). *The* width *of a function $f \in \mathcal{D}^0$ is computed as the maximum width of all intervals $f(x)$: $w(f) = \max_{x \in [0, X_f]} w(f(x))$*

Proposition 4. *The operator $\circ$ is monotone and continuous. The width w is a monotone, continuous function from $(\mathcal{D}^0, \sqsubseteq)$ to $([0, \infty[, \preceq)$ where $x \preceq y \Leftrightarrow y \leq x$.*

The proof of this proposition is straightforward: we use the monotonicity of functions with respect to set inclusion for $\circ$ and we note that for two intervals i_1, i_2, $i_1 \subseteq i_2 \Rightarrow w(i_2) \leq w(i_1)$, thus the monotonicity of w. The primitive operator is not monotone, as it does not preserve the third condition for the order $\sqsubseteq$ (Equation (7)). However, the second condition is preserved thanks to the monotonicity of the primitive for real-valued functions.

A measurement for $\mathcal{D}^0$

Among all the functions of $\mathcal{D}^0$, one is of special interest for us: y_∞, the maximal solution of (5).We will compute it by successive approximations and thus we need a way to measure the quality of our approximation. Following Keye Martin's measure theory [24, 25], a measurement is a continuous function μ from a CPO $\mathcal{D}$ into the set of positive real numbers with reverse ordering: $[0, \infty[^*$. The measurement of an element f reveals the distance of f to the maximal elements of $\mathcal{D}$, which should have measure 0. The measurement must also be coherent with the informational order on $\mathcal{D}$: the more informative f, the smaller its measure. Moreover, it must be the case that if we *measure* that the sequence of approximation f_n converges towards 0 ($\lim_{n \to \infty} \mu(f_n) = 0$), then the sequence f_n *does* converge towards a maximal element ($\bigsqcup_n f_n = f, \ \mu(f) = 0$). For a

formal definition of a measurement, we invite the reader to read [24], Chapter 2. In our case, the maximal elements of $\mathcal{D}^0$ are the real-valued functions defined on $\mathbb{R}_+$. These functions have a null width and an infinitely long domain of definition. Thus, a measurement must takes both aspects into account.

Definition 9 (*The measurement* μ). *Let $f \in \mathcal{D}^0$. Then, its measurement is:*

$$\mu(f) = w(f) + \frac{1}{X_f}$$

Clearly, $\mu(f)$ is null if and only if f is maximal, so in particular $\mu(y_\infty) = 0$.

Proposition 5. *μ is a measurement, i.e.:*

(i) *it is a Scott continuous map from $(\mathcal{D}^0, \sqsubseteq)$ into $[0, \infty[^*$.*
(ii) *for all $f \in \mathcal{D}^0$ such that $\mu(f) = 0$ and all sequences $f_n \ll f$, we have $\lim_{n\to\infty} \mu(f_n) = 0 \Rightarrow \sqcup_n f_n = f$*

We recall that the far away *relation $f \ll g$ means that for every increasing chain φ_n with a supremum greater than g, the elements φ_n must become greater than f at some $N \in \mathbb{N}$.*

Proof. Point **(i)** is a straightforward consequence of Proposition 4. The proof of point **(ii)** relies on two observations runs as follows. Let $f \in \mathcal{D}^0$ be such that $\mu(f) = 0$ and let f_n be a sequence such that $\forall \in \mathbb{N}$, $f_n \ll f$ and $\lim_{n\to\infty} \mu(f_n) = 0$. As $f_n \ll f$, it is true that $f_n \sqsubseteq f$; moreover, it holds that $\lim_n(X_{f_n}) = \infty$ and $\lim_n \big(w(f_n)\big) = 0$. Let now $x \in \mathbb{R}_+$; there exists $N \in \mathbb{N}$ such that $X_{f_N} \geq x$, and $\forall n \geq N$, we have $f(x) \in f_n(x)$. In addition, $lim_n w(f_n(x)) = 0$, so the sequence of intervals $f_n(x)$ converges toward the singleton $f(x)$, and always contain $f(x)$. So, $\lim_n f_n^-(x) = \lim_n f_n^+(x) = f(x)$ and consequently $\bigsqcup_n f_n = f$. □

Let us recall the results of this Section. We have built a lattice $\mathcal{D}^0$ and defined three operators on it: I, $\circ$ and w. Furthermore, we have a measurement μ on $\mathcal{D}^0$ which characterizes the maximal elements of $\mathcal{D}^0$, i.e. the real-valued functions defined on $\mathbb{R}_+$. This measurement will be used as follows: the semantics $[\![\kappa]\!]$ of the continuous model is defined as the fix-point of an operator Γ_{F,y_0} on $\mathcal{D}^0$ (see Section 4.3) and this operator is such that the iterates $f_n = \Gamma^n_{F,y_0}(\bot)$ verify $f_n \ll y_\infty$ and $\lim_n \mu(f_n) = 0$. By the definition of the measurement, we conclude that $\bigsqcup f_n = y_\infty$. This intuition is formalized in the next section.

4.3 The Semantics

The denotational semantics $[\![\kappa]\!]$ of a continuous model κ is a function mapping an initial state $y \in \mathcal{D}^0$ to the solution of Equation (5). $[\![\kappa]\!](y)$ is computed as the least fix-point of the operator $\Gamma_{F,y_0} : \mathcal{D}^0 \to \mathcal{D}^0$. This operator acts as follows: given some partial information (i.e. a function $f \in \mathcal{IF}^0_X$), it first updates the available information by bringing each $f(x)$ closer to $y_\infty(x)$ and then it extends the function to the right by assigning a value to $f(x)$ for $x \in [X, X+1]$. The first step uses an iteration of the Picard operator (Equation (2)) while the second step

extends the function in such a way that if f encloses the solution at X, then the extension encloses y_∞ on $[X, X+1]$. This is possible because F is α-Lipschitz, so y_∞ cannot grow faster than $e^{\alpha x}$. We recall that the Picard operator is defined as

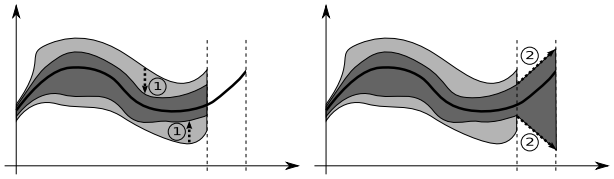
$$P_{[0,X_f]}\big(F, y_0\big)(f) = \lambda x.y_0 + \int_0^x F(f(s))ds = y_0 + I(F \circ f).$$

Let us now give the formal definition of $[\![\kappa]\!]$.

Definition 10 (*Updating operator*). *Let $f \in \mathcal{D}^0$, we suppose $X_f < \infty$. Let F be a continuous, globally α-Lipschitz function and $y_0 \in \mathbb{R}$. Then, $\Gamma_{F,y_0}(f) \in \mathcal{IF}^0_{X_f+1}$ is defined by:*

$$\Gamma_{F,y_0}(f)(x) = \begin{cases} P_{[0,X_f]}\big(F, y_0\big)(f)(x) & \text{if } x \leq X_f \\ J + F(J) * [-e^\alpha, e^\alpha] * (x - X), & \\ \quad \text{with } J = P_{[0,X_f]}\big(F, y_0\big)(f)(X) & \text{otherwise} \end{cases}$$

If $f \in \mathcal{IF}^0_\infty$, $\Gamma_{F,y_0}(f) = P_{[0,\infty[}\big(F, y_0\big)(f)$. $\Gamma_{F,y_0}(\bot)$ is the function defined on $[0,0]$ with value y_0.



(a)Update the information (b)Extends the information

Fig. 4.3. The updating operator (two steps).

An example of the effect of Γ_{F,y_0} on a partial function is shown on Figure 4.3. The black line represents y_∞; the left part of the figure shows the updating mechanism, while the right part is the extension. The operator Γ_{F,y_0} is not monotone on $\mathcal{D}^0$, but we know that it has a fix-point: y_∞. We will show in the following that this fix-point can be computed as the supremum of the Γ_{F,y_0} iterates, i.e. $y_\infty = \bigsqcup_n \Gamma^n_{F,y_0}(\bot)$.

Proposition 6. *Let $f \in \mathcal{IF}^0_X$. Γ_{F,y_0} verifies the invariant:*

$$\forall x \in [0, X], y_\infty(x) \in f(x) \Rightarrow \forall x \in [0, X+1], y_\infty(x) \in \Gamma_{F,y_0}(f)(x)$$

Proof. Let $f \in \mathcal{D}^0$ be such that $\forall x \in [0, X_f]$, $y_\infty(x) \in f(x)$. Let $f' = \gamma_{F,y_0}(f)$ and $x \in [0, X_f + 1]$. If $x \in [0, X_f]$, we have $f'(x) = P_{[0,X_f]}(F, y_0)(f)(x)$. Clearly, $P_{[0,X_f]}(F, y_0)$ is monotone, so $P_{[0,X_f]}(F, y_0)(y_\infty)(x) \subseteq P_{[0,X_f]}(F, y_0)(f)(x)$ and y_∞ is a fixpoint of the Picard operator, so $y_{infty}(x) \in f'(x)$. Now, if $x \in [X_f, X_{f'}]$, we know that as F is k-Lipschitz, it holds that

$$\|y_\infty(t) - y_\infty(t_0)\| \leq e^{k.|t-t_0|}.F(y_\infty(t_0)).(t - t_0)$$

for all $t,\ t_0$. If we apply this to $t_0 = X_f$ and $t = X_{f'}$, we have $y_\infty(t_0) \in P_{[0,X_f]}(F, y_0)(f)(X_f)$ and we thus get that $f_\infty(x) \in f'(x)$. □

The iterates $f_{n+1} = \Gamma_{F,y_0}(f_n)$, starting from $f_0 = \bot$, form a sequence of approximation of y_∞: they enclose it and their width converge toward 0. Figure 4.4 represent the first 3 iterates of f_n when the IVP is Equation (4), with $i = 2$, $k_1 = k_2 = 1$, v open and $x \leq h$. The black curve is the real solution and the gray lines represent respectively f_1, f_2 and f_3. The semantics of the continuous subsystem $\kappa = \left(F, \{F_k\}, y_0\right)$ is the function mapping $f \in \mathcal{D}^0$ with the least fix-point of Γ_{F,y_0} starting from f: $[\![\kappa]\!](f) = \bigsqcup_n \Gamma^n_{F,y_0}(f)$. We now give the main result of this Section.

Theorem 3. *The solution y_∞ of (5) is a fix-point of Γ_{F,y_0} and*

$$[\![\kappa]\!](\bot) = Fix(\Gamma_{F,y_0}) = \bigsqcup_n \Gamma^n_{F,y_0}(\bot) = y_\infty.$$

Proof. By induction on n, we show that $\forall n,\ f_n \in \mathcal{IF}^0_{[0,n]}$ and $\forall x \in [0,n],\ y_\infty(x) \in f_n(x)$. Based on that we prove that $f_n \ll y_\infty$. Next, we must prove that $\lim_{n\to\infty} \mu(f_n) = 0$. Clearly, $\lim \frac{1}{X_{f_n}} = 0$, so we must prove that $\lim w(f_n) = 0$. Let $x \in \mathbb{R}_+$, we prove that $\lim w(f_n(x)) = 0$ using the property of the Picard iterates (Theorem 2). Thus, for all $x \in \mathbb{R}_+$, $w(f_n(x))$ tends toward 0, so $\lim_n w(f_n) = 0$.

5 Discrete Semantics

We now give a denotational semantics of the discrete subsystem Δ. We recall that Δ is an imperative program to which some hybrid actions (**sens**, **act** and **wait**)

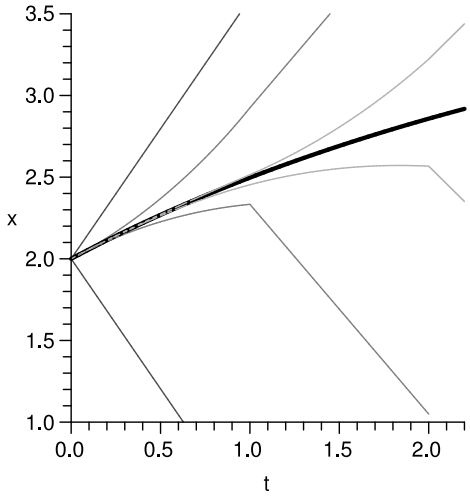


Fig. 4.4. Example of iterations of Γ_{F,y_0}.

were added. In this section, we suppose that the semantics of the continuous system is already computed, i.e. we know exactly the solutions of each IVP $\dot{y} = F_k(y),\ y(t_0) = y_0$ for each possible dynamics F_k and each initial condition (t_0, y_0). These solutions are $[\![\dot{y} = F_k(y),\ y(t_0) = y_0]\!](\bot)$, i.e. the semantics of the corresponding continuous system. The semantics of the discrete program Δ is given as a denotational semantics, i.e. a function $[\![\Delta]\!] : \Sigma \to \Sigma$ between environments. The environments bind every variable with its value, but also carry information about time. Actually, it is important to know the absolute time at which an instruction is executed, as the denotation of **sens** or **act** action strongly depend on it. We first present the environments that we consider (Section 5.1), then the effect of discrete statements on them (Section 5.2) and finally the effect of hybrid actions (Section 5.3).

5.1 The Environments

Let Var be the set of all variables of the program and Val be the set of values that these variables may take. For example, Val may be the set of floating-point numbers. We suppose that there exist two extra variables $time, y \notin Var$ that represent respectively the continuous time and the continuous environment. An environment σ binds every variable with a value, as well as it binds $time$ with a positive real number and y with a continuous function. The set of all environments is:

$$\Sigma = \left\{ \left(Var \to Val\right) * \left(\{time\} \to \mathbb{R}_+\right) * \left(y : \mathbb{R}_+ \to \mathbb{R}\right) \right\}.$$

For $\sigma \in \Sigma$, we denote $\sigma.x$ the value of the variable $x \in Var \cup \{time, y\}$. Thus, the instantaneous value of the physical environment within σ is $\sigma.y(\sigma.time)$. We also note $\sigma[x \mapsto v]$ the environment σ where the value of the variable x has been changed to v.

5.2 Denotations of Discrete Statements

The effect of the purely discrete statements on the environments is the same as for any imperative language. We quickly present the semantics of expressions, booleans and statements.
The semantics of an expression $e \in exp$ is a function mapping an environment to a value: $[\![e]\!] : \Sigma \to Val$. We write the functions $\Sigma \to Val$ as sets of pair (σ, n) with $\sigma \in \Sigma$ and $n \in Val$. Some of the denotations of expressions are given in Table 2. The semantics of a boolean $b \in bool$ is a function mapping an environment to a boolean value: $[\![b]\!] : \Sigma \to \{true, false\}$. The denotations for the comparison and for the $\vee$ operator are shown in Table 2. Finally, the semantics of a statement $s \in stmt$ is a function mapping an environment to another: $[\![s]\!] : \Sigma \to \Sigma$. Table 2 also presents the denotations for the statements (assignments, **if**, **while** and **;** sequence).

$$
\begin{aligned}
[\![n]\!] &= \{(\sigma, n) \mid \sigma \in \Sigma\} \\
[\![x]\!] &= \{(\sigma, n) \mid \sigma \in \Sigma \text{ and } n = \sigma.x\} \\
[\![e1 + e2]\!] &= \{(\sigma, n_1 + n_2) \mid (\sigma, n_1) \in [\![e1]\!] \text{ and } (\sigma, n_2) \in \sigma\} \\
[\![v < e]\!] &= \{(\sigma, true) : [\![v]\!]\sigma < [\![e]\!]\sigma\} \cup \{(\sigma, false) : [\![v]\!]\sigma \geq [\![e]\!]\sigma\} \\
[\![b_0 \vee b_1]\!] &= \{(\sigma, t_0 \vee t_1) \mid (\sigma, t_0) \in [\![b_0]\!] \text{ and } (\sigma, t_1) \in [\![b_2]\!]\}
\end{aligned}
$$

$$
\begin{aligned}
[\![i_1; i_2]\!] &= [\![i_2]\!] \circ [\![i_1]\!] \\
[\![v = e]\!] &= \big\{(\sigma, \sigma') \mid \sigma' = \sigma\,[v \mapsto n] \text{ and } (\sigma, n) \in [\![e]\!]\big\} \\
[\![\textbf{if } b \textbf{ then } i1 \textbf{ else } i2]\!] &= \big\{(\sigma, \sigma') \mid (\sigma, true) \in [\![b]\!] \text{ and } (\sigma, \sigma') \in [\![i1]\!]\big\} \\
&\quad \cup \big\{(\sigma, \sigma') \mid (\sigma, false) \in [\![b]\!] \text{ and } (\sigma, \sigma') \in [\![i2]\!]\big\} \\
[\![\textbf{while}(b)\ inst]\!] &= Fix(\Gamma) \text{ with} \\
\Gamma(\varphi) &= \big\{(\sigma, \sigma') \mid [\![b]\!]\sigma = true \text{ and } (\sigma, \sigma') \in \varphi \circ [\![inst]\!]\big\} \cup \{(\sigma, \sigma) \mid [\![b]\!]\sigma = false\}
\end{aligned}
$$

Table 2. Denotations for discrete statements.

5.3 Denotations of Hybrid Statements

The semantics of an hybrid statement *hyb_stmt* is a function that modifies an environment: $[\![hyb\_stmt]\!] : \Sigma \to \Sigma$. The modification is as follows:

- **sens.y?x** changes the value of x to floating point number closest to the value of the continuous variable y at the time the action is executed,
- **wait c** modifies the value of the variable *time* and adds **c** to it,
- **act.k!c** changes y so that, from the time t_0 the action is executed, it follows the solution of the ODE $\dot{y} = F_c(y)$. Thus, the new environment maps a time t to either the value of the previous environment (if $t \leq t_0$) or to the solution of $\dot{y} = F_c(y),\ y(t_0) = \sigma.y(t_0)$ if $t \geq t_0$.

Table 3 presents the denotations for hybrid instructions with the same conventions as for pure discrete ones.

6 Hybrid Semantics

Let us now give the semantics of the complete hybrid system. The hybrid model is a pair $\Omega = (\Delta, \kappa)$ consisting of a model Δ for the discrete system and a model κ for the continuous environment. These define two dynamical systems (Δ and κ) that run in parallel and, from time to time, communicate. The communication between them is of two kinds:

- data are passed from κ to Δ via the sensors. This communication requires that both dynamical systems reached the same time before the data is exchanged. The **sens** actions must thus be blocking.

$$[\![\textbf{sens}.\text{y?x}]\!] = \left\{(\sigma,\sigma')|\sigma' = \sigma[x \mapsto \uparrow_{\sim} (\sigma.y(\sigma.time))]\right\}$$

$$[\![\textbf{wait } c]\!] = \left\{(\sigma,\sigma') \mid \sigma' = \sigma[time \mapsto \sigma.time + c]\right\}$$

$$[\![\textbf{act}.\text{k!c}]\!] = \left\{(\sigma,\sigma')|\sigma' = \sigma\left[y \mapsto \lambda x.\begin{cases}\sigma.y(x) \; if \; x \leq \sigma.time\\ y_c(x) \quad else\end{cases}\right]\right\}$$
$$where \; y_c = [\![\dot{y} = F_c(y), \quad y(\sigma.time) = \sigma.y(\sigma.time)]\!](\bot)$$

Table 3. Denotations for hybrid statements.

– orders are passed from Δ to κ via the actuators. Indeed, the discrete system only indicates to the continuous system what its semantics will be, i.e. it chooses one of the possible functions F_k. This communication needs not to be blocking as it does not affect the value of the continuous variables but only their future behavior.

The hybrid denotations for **sens** and **act** respect these observations. The semantics $[\![\Omega]\!]^{\mathcal{H}}$ of Ω is a function between hybrid environments that are made of a discrete and a continuous environment. The discrete environment is altered by the discrete subsystem while the continuous one is computed only when necessary, i.e. when a **sens** is executed. We first define the hybrid environments, then the hybrid denotations for the discrete and hybrid statements and we finally define the semantics of the hybrid system using them.

6.1 Hybrid Environments

A hybrid environment consists of a pair made of a discrete and a continuous environment. The discrete environment σ_δ binds every discrete variable $v \in Var$ to a value and the time *time* to a positive real value. It also contains the function F that defines the semantics of the continuous variables. This function F is piecewisely defined by the discrete program through the **act** statements and thus storing F is equivalent to storing the sequence of all executed **act** actions. The discrete environment thus completely describes the discrete program: it has both the value of the variables and the execution time, as well as the sequence of modifications brought to the continuous system. We write Σ_Δ the set of all discrete environments:

$$\Sigma_\Delta = \left\{\left(Var \to Val\right) * \left(\{time\} \to \mathbb{R}_+\right) * \left(F : \mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}\right)\right\}.$$

The continuous environment σ_κ contains an approximation of the physical variables $y \in \mathcal{D}^0$ and the set of functions F_k defining the continuous dynamics. These are the set of possible continuous modes that are available for the discrete program to chose. We write Σ_κ the set of all continuous environments:

$$\Sigma_\kappa = \left\{\left(y \in \mathcal{D}^0\right) * \left(F_{\boldsymbol{k}} \mid F_k : \mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}\right)\right\}.$$

We write $\Sigma^{\mathcal{H}}$ the set of all hybrid environments:

$$\Sigma^{\mathcal{H}} = \left\{ \left(\sigma_\delta, \sigma_\kappa\right) \;\middle|\; \begin{array}{l} \sigma_\delta \in \Sigma_\Delta \text{ and } \sigma_\kappa \in \Sigma_\kappa \text{ and} \\ \exists (t_n), (c_n) \text{ s.t. } \forall i \in \mathbb{N},\ \forall t \in [t_i, t_{i+1}[, \\ \quad \sigma_\delta.F(t) = \sigma_\kappa.F_{c_i}(t) \end{array} \right\} \tag{8}$$

$\Sigma^{\mathcal{H}}$ contains all the pairs $\left(\sigma_\delta, \sigma_\kappa\right)$ that are compatible, i.e. such that the function $\sigma_\delta.F$ is made of fragments of functions from $\{\sigma_\kappa.F_k\}$. We write $\Pi_\delta : (\sigma_\delta, \sigma_\kappa) \mapsto \sigma_\delta$ and $\Pi_\kappa : (\sigma_\delta, \sigma_\kappa) \mapsto \sigma_\kappa$ the two projections of an hybrid environment into a discrete (resp. continuous) one. As before, we write $\sigma_\delta.X$ the the value of a variable $X \in Var \cup \{time, F\}$ in the discrete environment and $\sigma_\kappa.Y$ the value of a variable $Y \in \{y\} \cup \{F_k\}$ in the continuous one.

6.2 Hybrid Denotations

The denotation of the purely discrete parts of the language are left unchanged. The denotation of numerical (resp. boolean) expressions are functions from $\Sigma^{\mathcal{H}}$ to numerical (resp. boolean) values. These values are the ones computed by the discrete environment only (see Equation (9)).

$$\begin{array}{l} [\![exp]\!]^{\mathcal{H}}(\sigma_\kappa, \sigma_\delta) = [\![exp]\!](\sigma_\kappa) \\ [\![bool]\!]^{\mathcal{H}}(\sigma_\kappa, \sigma_\delta) = [\![bool]\!](\sigma_\kappa) \end{array} \tag{9}$$

The denotation of a statement *stmt* is a function from $\Sigma^{\mathcal{H}}$ to $\Sigma^{\mathcal{H}}$. Its definition is exactly the same as in the discrete case, except that the environments σ must be changed to pairs $(\sigma_\delta, \sigma_\kappa)$ in Table 2. The denotation of a **wait** remains as in the discrete semantics: the time is actually controlled by the discrete subsystem that imposes the sampling rate and thus the flow of time.
The denotation of an action **sens.y?x** is a function from $\Sigma^{\mathcal{H}}$ to $\Sigma^{\mathcal{H}}$ that modifies a pair $(\sigma_\delta, \sigma_\kappa)$ as follows: it first updates σ_κ to ensure that $\sigma_\kappa.y$ has a value at time $\sigma_\delta.time$ and then it binds x with this value in σ_δ. The first step is done by applying $\lfloor \sigma_\delta.time + 1 \rfloor$ times the operator Γ_{F,y_0} (see Section 4.3) to $\sigma_\kappa.y$ with $F = \sigma_\delta.F$ and $y_0 = \sigma_\kappa.y(0)$. The first equation of Table 4 formally defines this intuition.
The denotation of an action **act.k!c** is a function from $\Sigma^{\mathcal{H}}$ to $\Sigma^{\mathcal{H}}$ that modifies a pair $(\sigma_\delta, \sigma_\kappa)$ as follows: σ_κ is left unchanged and in σ_δ, the function F is modified so that it takes the value of $\sigma_\kappa.F_c$ for times greater than $\sigma_\delta.time$. The definition of $[\![\mathbf{act}.\text{k!c}]\!]^{\mathcal{H}}$ is given in the second equation of Table 4.

6.3 Hybrid Semantics

The semantics of the hybrid model $\Omega = \left(\Delta, \kappa\right)$ is a function between hybrid environments: $[\![\Omega]\!]^{\mathcal{H}} : \Sigma^{\mathcal{H}} \to \Sigma^{\mathcal{H}}$. $[\![\Omega]\!]^{\mathcal{H}}$ alters a pair $\left(\sigma_\delta, \sigma_\kappa\right)$ as follows. It first applies $[\![\Delta]\!]^{\mathcal{H}}$ and thus computes $\left(\sigma'_\delta, \sigma'_\kappa\right) = [\![\Delta]\!]^{\mathcal{H}}\left(\sigma_\delta, \sigma_\kappa\right)$. Then, two cases occur:

$$[\![\mathbf{sens}.\mathrm{y?x}]\!]^{\mathcal{H}}(\sigma_\delta,\sigma_\kappa) = (\sigma'_\delta,\sigma'_\kappa) \text{ with } \begin{cases} \sigma'_\kappa = \sigma_\kappa[y \mapsto \Gamma^n_{\sigma_\delta.F,y(0)}(y)] \\ \sigma'_\delta = \sigma_\delta[x \mapsto \uparrow_\sim (mid(\sigma'_\kappa.y(\sigma_\delta.time)))] \end{cases}$$

$$\text{where } n = \lfloor \sigma_\delta.time + 1 \rfloor$$

$$[\![\mathbf{act}.\mathrm{k!c}]\!]^{\mathcal{H}}(\sigma_\delta,\sigma_\kappa) = (\sigma'_\delta,\sigma_\kappa) \text{ with } \sigma'_\delta = \sigma_\delta \left[F \mapsto \lambda t, y. \begin{cases} \sigma_\delta.F(y,t) & \text{if } t \leq \sigma_\delta.time \\ \sigma_\kappa.F_c(y,t) & \text{otherwise} \end{cases} \right]$$

Table 4. Hybrid denotations for hybrid statements.

- $\sigma'_\kappa = \sigma_\kappa$. This means that the discrete program has no effect on the environment, i.e. either there are no **sens** statements in it, or they have no effect on σ_κ. This is the case only if $\sigma_\kappa.y$ is a fix-point of Γ_{F,y_0}, i.e. $\sigma_\delta.y = [\![\kappa]\!](\sigma_\delta.y)$. In this case, we have computed both the continuous semantics and the discrete one, so we set $[\![\Omega]\!]^{\mathcal{H}}(\sigma_\delta,\sigma_\kappa) = (\sigma'_\delta,\sigma'_\kappa)$.
- $\sigma'_\kappa \neq \sigma_\kappa$. The program has modified the environment and thus brought $\sigma_\delta.y$ closer to $[\![\kappa]\!](\sigma_\delta.y)$. σ'_δ (resp. σ'_κ) is only an approximation of the result of the discrete (resp. continuous) system and we must iterate the process to obtain a better approximation. We thus propagate σ'_κ into the discrete subsystem, i.e. we apply $[\![\Delta]\!]^{\mathcal{H}}$ to $(\sigma_\delta,\sigma'_\kappa)$ and repeat the operation.

The semantics $[\![\Omega]\!]^{\mathcal{H}}$ is computed as a fix-point of a function that applies $[\![\Delta]\!]$ consecutively until the semantics of the continuous environment has been computed. The formal definition of $[\![\Omega]\!]^{\mathcal{H}}$ is given in Table 5. Let us note that $[\![\Omega]\!]^{\mathcal{H}}$ is actually the only fix-point of the function $\Gamma^{\mathcal{H}}$ just like $[\![\kappa]\!]$ was the only fix-point of Γ_{F,y_0} in Section 4. $[\![\Omega]\!]^{\mathcal{H}}$ is compatible with the discrete semantics $[\![\Delta]\!]$ presented in the previous section and with the continuous semantics $[\![\kappa]\!]$ presented in Section 4. The continuous environment is finally computed as the fix-point of the operator Γ_{F,y_0} as in Section 4.3. In comparison with $[\![\Delta]\!]$, the hybrid semantics reveal the hidden fix-point computations. Actually, the denotation for **act** in the discrete semantics (see Table 3) requires that we compute the solution of an IVP, i.e. the fix-point of an operator Γ_{F,y_0}. These internal fix-point computations are made explicit in Table 5. Let us note that we could have chosen to place this fix-point computation inside the denotation of the **sens** action (i.e. $y \mapsto Fix(\Gamma_{\sigma_\delta.F,y(0)})$ instead of $y \mapsto \Gamma^n_{\sigma_\delta.F,y(0)}(y)$ in Table 4). This would have been more coherent with the discrete semantics, but requires an infinite computation between the execution of two statements, which is not compatible with the standard notion of denotational semantics.

Let us note that this semantics is also compatible with the standard denotational semantics of imperative languages: if Δ does not have any hybrid actions, then $[\![\Omega]\!]^{\mathcal{H}}$ is precisely the semantics of the discrete program as defined in [32] for example.

$$[\![\Omega]\!]^{\mathcal{H}} = Fix(\Gamma^{\mathcal{H}}) \text{ where}$$
$$\Gamma^{\mathcal{H}}(\varphi)(\sigma_\delta, \sigma_\kappa) = (\sigma'_\delta, \sigma'_\kappa) \text{ with } \begin{cases} \sigma'_\delta = \Pi_\delta\big([\![\Delta]\!]^{\mathcal{H}}(\sigma_\delta, \sigma'_\kappa)\big) \\ \sigma'_\kappa = \Pi_\kappa\big(\varphi(\sigma_\delta, \sigma^t_\kappa)\big) \text{ where } \sigma^t_\kappa = \Pi_\kappa\big([\![\Delta]\!]^{\mathcal{H}}(\sigma_\kappa, \sigma_\delta)\big) \end{cases}$$

Table 5. The semantics of the hybrid system.

6.4 Example

To illustrate our semantics and show that it really computes the behavior of the hybrid system, let us consider a simplified version of the one-tank controller. We only consider two iterations of the **while** loop and don't take into account the anticipation mechanism. For the example to be significant, we suppose that the **while** loop had a period of one second. The program representing this simplified version is given in Figure 6.1. The continuous system is still given by Equation (4), with $i = 2$, $k_1 = k_2 = 1$, $h = 3$, $h\_max = 2.9$, and the initial value for the height of water x is $x_0 = 2$. We suppose that the valve is initially closed. Let us now compute the semantics of this system. Table 6 shows the beginning

```
wait (1);
sens.x?h;
if (h>h_max)
  act.k!0; throw( alarm );
wait (1);
sens.x?h;
if (h>h_max)
  act.k!0; throw( alarm );
```

Fig. 6.1. Simplified version of the one-tank controller.

of this computation. The table is to read as follows: the left column shows the program, then each column corresponds to one iteration in the fix-point computation. For example, the second column correspond to the first computation of $[\![\Delta]\!]^{\mathcal{H}}$ on the initial environment. The first row presents the environment $(\sigma_\delta, \sigma_\kappa)$ before the computation, the last row the environment after, i.e. $[\![\Delta]\!]^{\mathcal{H}}(\sigma_\delta, \sigma_\kappa)$. Each intermediate row represents the changes in the hybrid environment due to the corresponding statement. We notice that the continuous environment has changed during this first iteration, so we compute $[\![\Delta]\!]^{\mathcal{H}}$ another time but with an updated initial environment (second column). We get better values for the **sens** actions, and a new continuous environment at the end. We repeat the iteration once more (third column), which gives us a more accurate result. Let us remark that during this third iteration, the second **act** statement was executed, which corresponds to the real behavior of the program. So, we see that the

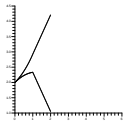
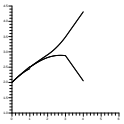
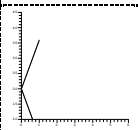
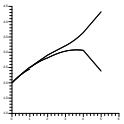
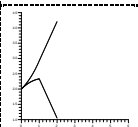
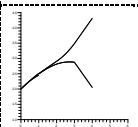
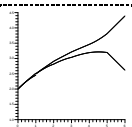
Statement	Iteration 1			Iteration 2			Iteration 3		
	t	h	x	t	h	x	t	h	x
Initial environment	0	$\perp$	$\perp$	0	$\perp$		0	$\perp$	
`wait(1);`	1			1			1		
`sens.x?h;`		2.0			2.45			2.48	
`if (h>h_max) act.k!1;`		$F \mapsto F_0$			$F \mapsto F_0$			$F \mapsto F_0$	
`wait(1);`	2			2			2		
`sens.x?h;`		2.8			2.85			2.95	
`if(h>h_max) act.k!1;`		$F \mapsto F_0$			$F \mapsto F_0$			$F \mapsto \lambda t.(t < 2)?F_0; F_1$	

Table 6. First three iterations of the semantics computation.

consecutive iterations compute an ever more accurate behavior, and eventually, the real semantics (i.e. as it would have been computed in Sections 4 and 5) is reached.

7 Conclusion

In this article, we presented a new approach to hybrid systems that can be used for the modeling and analysis of large critical embedded programs. Our model for hybrid systems is based on a clear separation of the discrete and the continuous systems. They are described in two different formalisms: ordinary differential equations with boolean parameters for the continuous system, an imperative language with hybrid statements for the discrete part. The emphasis has been placed on making this model as unintrusive as possible for existing software, so we believe that we can effectively use it for industrial size problems. We gave a denotational semantics of this model that unifies in a coherent formalism the semantics of the continuous and the discrete parts. The main difficulty was to express the solution of an IVP as the semantics of a continuous program, i.e. the underlying ODE. In [12], Edalat and Lieutier proposed a data type for differential calculus, we extended their results to consider the maximal solutions of IVP on $\mathbb{R}_+$ and in an analogy with imperative languages we presented the semantics of the continuous model as a function mapping the initial condition to the maximal

solution. The semantics of the discrete subsystem is an extension of the standard denotational semantics of imperative languages, we added information on time and the continuous environment to give a denotation to the actions of sensors and actuators. The semantics of the hybrid system is a composition of these two semantics and remains coherent with them.

This model for hybrid systems and especially its denotational semantics is a first step toward the validation of embedded software with their environment. The analysis of such systems using, for example, abstract interpretation techniques [9] requires two stages. First, the continuous system must be abstracted in a non-naive way. Our semantics for the continuous system already does that, as it constantly computes a partial interval-valued function that encloses the real-valued solution. This abstraction is however not computable and cannot be used for a static analyzer. The theory of guaranteed integration of ODE [29] brings us the adequate tools for the safe abstraction of the continuous system. Validated ODE solvers [5, 28] compute interval bounds that are proved to contain the solution. They can thus be used to construct piecewise constant interval-valued functions that enclose the solution at all time. For the analysis of the discrete part, the use of an implementation level model allows us to use existing methods (for example, the verification of the absence of run-time errors [10] or of the numerical precision of floating point computations [16]). These methods must however be completed so that they consider time. Actually, the main difficulty in the analysis of the discrete system is to carefully analyze the time at which every statement is executed (this is necessary for the sensor actions to be precise enough), but taking time into account makes it hard to find a suitable widening. This modification of standard static analysis techniques to our framework will be our main concern for future work.

Another interesting application of our approach for hybrid systems is to modify standard strictness [26, 27] or termination analysis [8, 4] so that they fit to our model. This could be used to solve, in an approximate way, the reachability problem of a discrete state in a hybrid system. It is well-known that this problem is undecidable [19] and several methods have been proposed for its simplification [20]. We believe that our approach may be efficiently used for its approximate solution as it benefits from all the static analysis based methods for programming languages.

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. *Theoretical Computer Science*, 138(1):3–34, 1995.
2. R. Alur and D.L. Dill. A theory of timed automata. *TCS*, 126:183–235, 1994.
3. R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specification of hybrid systems in charon. In *HSCC*, volume 1790 of *LNCS*. Springer-Verlag, 2000.
4. J. Berdine, A. Chawdhary, B. Cook, D. Distefano, and P. O'Hearn. Variance analyses from invariance analyses. In *POPL*, pages 211–224, 2007.

5. O. Bouissou and M. Martel. GRKLib: a guaranteed runge-kutta library. In *International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics*. IEEE, 2006.
6. M.S. Branicky. Multiple lyapunov functions and other analysis tools for switched and hybrid systems. *Automatic Control*, 43(4):475–482, April 1998.
7. P. Cousot. Integrating physical systems in the static analysis of embedded control software. In *APLAS*, volume 3780 of *LNCS*, pages 135–138, Tsukuba, Japan, November 3–5 2005. Springer, Berlin.
8. P. Cousot. Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. In *VMCAI*, volume 3385 of *LNCS*, pages 1–24, 2005.
9. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252. ACM Press, 1977.
10. P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Minéand D. Monniaux, and X. Rival. The ASTREÉ analyzer. In *ESOP*, volume 3444 of *LNCS*, 2005.
11. P. Cuijpers and M. Reniers. Hybrid process algebra. *Journal of Logic and Algebraic Programming*, 62(2):191–245, 2005.
12. A. Edalat and A. Lieutier. Domain theory and differential calculus. *Mathematical Structures in Computer Science*, 14(06):771–802, 2002.
13. A. Edalat, A. Lieutier, and D. Pattinson. A computational model for multi-variable differential calculus. In *FOSSACS*, volume 3441 of *LNCS*, 2005.
14. A. Edalat and D. Pattinson. Denotational Semantics of Hybrid Automata. In *FOSSACS*, volume 3921, pages 231–245, 2006.
15. A. Girard and G. J. Pappas. Verification using simulation. In J. P. Hespanha and A. Tiwari, editors, *HSCC*, volume 3927 of *LNCS*, pages 272–286. Springer, 2006.
16. E. Goubault, M. Martel, and S. Putot. Asserting the precision of floating-point computations: A simple abstract interpreter. In *ESOP*, volume 2305 of *LNCS*, pages 209–212. Springer, 2002.
17. E. Goubault, M. Martel, and S. Putot. Some future challenges in the validation of control systems. In *ERTS*, 2006.
18. V. Gupta, R. Jagadeesan, and V. Saraswat. Computing with continuous change. *Science of Computer Programming*, 30(1–2):3–49, January 1998.
19. T. A. Henzinger. The theory of hybrid automata. In *Symposium on Logic in Computer Science*, pages 278–292. IEEE Computer Society Press, 1996.
20. T. A. Henzinger, P. H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43:540–554, 1998.
21. E. Ince. *Ordinary Differential Equations*. Dover Publications, 1956.
22. S. Kowalewski, O. Stursberg, M. Fritz, H. Graf, I. H., J. Preuß, and et al. A case study in tool-aided analysis of discretely controlled continuous systems: the two tanks problem. In *Hybrid Systems V*, volume 1567 of *LNCS*. Springer, 1999.
23. D. Liberzon. *Switching in Systems and Control*. Birkhäuser, Boston, MA, 2003.
24. K. Martin. *A Foundation for Computation*. PhD thesis, Department of Mathematics, Tulane University, 2000.
25. K. Martin. The measurement process in domain theory. In *ICALP*, pages 116–126, London, UK, 2000. Springer-Verlag.
26. A. Mycroft. The theory and practice of transforming call-by-need into call-by-value. In *Fourth International Symposium on Programming*, volume 83 of *LNCS*, pages 269–281, 1980.

27. A. Mycroft. *Abstract interpretation and Optimizing Transformations for Applicative Programs*. PhD thesis, Department of Computer Science, University of Edinburgh, 1981.
28. N. S. Nedialkov and K. R. Jackson. The design and implementation of an object-oriented validated ODE solver, 2001.
29. N.S. Nedialkov, K.R. Jackson, and G.F. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Applied Mathematics and Computation*, 105(1):21–68, 1999.
30. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
31. D. van Beek, K. Man, M. Reniers, J. Rooda, and R. Schiffelers. Syntax and consistent equation semantics of hybrid chi. *Journal of Logic and Algebraic Programming*, 68(1-2):129–210, 2006.
32. G. Winskel. *The formal semantics of programming languages: an introduction*. MIT Press, Cambridge, MA, USA, 1993.
33. Haiyang Zheng. *Operational Semantics of Hybrid Systems*. PhD thesis, EECS Department, University of California, Berkeley, May 2007.