A Hybrid Denotational Semantics for Hybrid Systems

Olivier Bouissou[1] and Matthieu Martel[2]

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

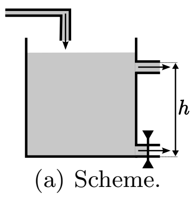
1 Introduction

The importance of static analysis techniques [6] 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 often 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 it using numerical computations and then 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 hybrid system made of the continuous environment and the discrete program [5, 14]. We use this approach. The analysis of hybrid systems requires as a starting point a formal description of their behavior. 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 [24]. 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 (containing a discrete *and* a continuous part) which 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 modeled in two different formalisms (see Sects. 2.1 and 2.2) which allows 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. First, we suppose that the discrete part is completely determined and we give a semantics $[\![\kappa]\!]$ for the continuous part (Sect. 3). 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 [22]. The semantics $[\![\Delta]\!]$ of the purely discrete part of the system is computed using the standard semantics of imperative languages (as in [24]). We add denotations for some hybrid actions that represent sensors and actuators, and show how these are combined to $[\![\kappa]\!]$ to form the hybrid semantics $[\![\Omega]\!]^{\mathcal{H}}$ (Sect. 4).

Running Example. We will illustrate this article with a simplified version of the well-known two tanks problem [18]. It consists of *one* water tank (Fig. 1.1(a)) filled by a constant flow i with two evacuation tubes: one at the bottom, which has a valve v than can be open or closed, and one at height h. The continuous system is the height x of the water in the tank, whose evolution is governed by the ordinary differential equation of Fig. 1(b). The discrete part is a controller whose goal is to maintain x between safe bounds by closing/opening the valve.

Related Work. The modeling of hybrid systems with hybrid automata was initiated by Henzinger [16]. They are finite state automata to which we add at each node 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 [12, 17]. A denotational semantics for these models was recently proposed by Edalat [11] and proved to be equivalent to the operational semantics. Since the first results, many models for hybrid systems and verification methods were proposed. These include hybrid process algebra like HyPa [8] or Hybrid Chi [23]. Meanwhile, Hybrid-CC [15] introduced hybrid components to the concurrent constraints theory. All these models are generally used as high level abstract formalisms to reason about the



(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.

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 [1]. The main difficulty in the formalization of hybrid systems is to give a coherent meaning to the continuous and the discrete parts. Edalat et al. proposed a formalization of differential calculus and of the solutions of differential equations in the theory of Scott domains, both for the mono-variate [9] and multi-variate [10] cases. We used their theory as a starting point for our work to define the denotational semantics of the continuous subsystem.

Notations and Mathematical Background. In this article, $\mathbb{R}$ denotes the set of real numbers, $\mathbb{R}_+$ the set of non-negative real numbers and $\mathbb{N}$ denotes the natural integers. The set of compact intervals over $\mathbb{R}$ is $\mathbb{I}(\mathbb{R})$. For $i \in \mathbb{I}(\mathbb{R})$, we write $\underline{i}$ (resp. $\overline{i}$) its lower (resp. upper) bound. We define its width $w(i) = \overline{i} - \underline{i}$ and its midpoint $mid(i) = \frac{\overline{i}+\underline{i}}{2}$. In Sect. 3, we use some advanced techniques of the theory of *ordinary differential equations* (ODEs). We assume that the reader is familiar with the basics of this theory, and give here just the main results that we will use. A more detailled version of this paper with some background elements of ODE theory can be found in [3]. The main theorem that we will use concerns the iterates of the Picard operator $P_I\big(F, y_0\big)$. Given $I \in \mathbb{I}(\mathbb{R})$, a continuous function F and $y_0 \in \mathbb{R}$, $P_I\big(F, y_0\big)$ is a map between continuous functions defined by $P_I\big(F, y_0\big)(f) = \lambda x. y_0 + \int_{\underline{I}}^{x} F(f(s), s)ds$. It gives a characterisation of the solution of an initial value problem (IVP) as a fixpoint and it provides a way to compute it via successive approximation, as shown by Theorem 1.

Theorem 1 (Properties of the Picard operator). *Let $\dot{y} = F(y),\ y(0) = y_0$ be an IVP. A continuous, differentiable function f on (a, b), with $0 \in (a, b)$, is a solution to the IVP if and only if it satisfies:*

$$\forall t \in (a, b),\ f(t) = P_{(a,b)}\big(F, y_0\big)(y)(t)\ . \tag{1}$$

If F is globally Lipschitz on $\mathbb{R}$, the Picard iterates *defined by $f_0 \in \mathcal{C}^0([a, b]),\ \ f_{n+1} = P_{[a,b]}\big(F, y_0\big)(f_n)$ converge uniformly on (a, b). So, whatever the choice of 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 the IVP on (a, b).*

2 Our Model for Hybrid Systems

Our goals for this model of hybrid systems are the following. First, the discrete part should remain close to existing embedded software. Secondly, the action of sensors and actuators must be clearly identified. Finally, we want the continuous and discrete systems to be modeled separately for two reasons. First, to analyze the behavior of a controller in different physical environments without rewriting 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 model 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 (Sect. 2.1) and then a model for discrete subsystems (Sect. 2.2).

2.1 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 y of a room with a heater is given by an ODE like $\dot{y} = 5 - 0.1y$. Let κ be the continuous model, its expressiveness depends on the set of functions F that we allow to define the IVP $\dot{y} = F(y),\ y(0) = y_0$. 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 (2)) while the second appears when the valve is closed.

To capture the changes due to the actuators, we allow F to have *boolean parameters*. We 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. 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.

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 (2).

$$F_k(x) = \begin{cases} i - k * k_1\sqrt{x} - k_2\sqrt{x - h} & \text{if } \ x \geq h \\ i - k * k_1\sqrt{x} & \text{otherwise} \end{cases} \tag{2}$$

2.2 Model for the Discrete Subsystem

We want the discrete model $\mathit{\Delta}$ to remain close to existing embedded software. We thus start with a set of standard statements which are common to any imperative

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

Fig. 2.1. Statements for the discrete system.

language (*stmt* in Fig. 2.1): assignemnts, **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, we have three hybrid actions. First, a **sens** action for 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. Then, a **act** action for 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 $F_{\boldsymbol{k}}$. Finally, a **wait** action for 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 programs already contain, as comments, the hybrid statements. For example, the loops of 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. 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 closing the valve takes two seconds, so the controller must predict the height of the water two seconds later (via the function `anticipate`) and start the opening if this predicted value is too high.

```
int main() {
  sensor x;          // sensors declaration
  actuator k;        // actuators declaration
  while (true) {
    sens.x?h;
    if (h>h_max)
      act.k!1; throw( alarm );
    h_in_2_secs = anticipate(h);
    if (h_in_2_secs > h_max)
      act.k!1;
    wait (0.01);    // delay action
  }
}
```

Listing 1. Controller for a one-tank system.

This model for hybrid systems conforms to our three requirements, and we designed it such that it prohibits physically impossible phenomena like continuous 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 discrete program is finite), which prohibits Zeno phenomena. We now give a formal, denotational semantics for this model of hybrid systems. The semantics is defined separately for the continuous system (Sect. 3) and then we merge it to the denotational semantics of imperative languages to form the semantics of the hybrid system (Sect. 4).

3 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 $\mathbb{R}$ to each ODE $\dot{y} = F_k(y,t)$. We first give the intuition for the continuous semantics and then we describe the lattice structure that we manipulate (Sect. 3.1) and the computation of the semantics as a fix-point (Sect. 3.2).

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:

$$\dot{y} = F(y,t),\ y(0) = y_0\ . \tag{3}$$

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 (3). 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. 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 1. We translate this fix-point computation into Tarski's fix-point theory: $[\![\kappa]\!](y)$ is computed as the fix-point of an operator Γ and 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 (3)) as the limit of an approximations sequence, each approximation being a partially defined, interval-valued function.

3.1 The Lattice of Interval-Valued Functions

We now define the set of partially defined, interval-valued functions. We also define an order and shows that this order provides a lattice structure.

Definition 1 (***Partial, interval-valued functions***)**.** *Let $X \in \mathbb{R}_+$. $\mathcal{IF}_X$ 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 $\overline{f}$ *and* lower $\underline{f}$ *functions as the two real-valued functions such that $\forall x \in [0, X],\ f(x) = [\underline{f}(x), \overline{f}(x)]$.*

When $\underline{f}$ (respectively $\overline{f}$) 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 when 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 point-wise reverse order: $f \sqsubseteq_X g \Leftrightarrow \forall x \in [0, X],\ g(x) \subseteq f(x)$. 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 [9]). The left-(resp. right) continuity of $\overline{f}$ (resp. $\underline{f}$) is a necessary condition for f to be Scott-continuous [9] *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, $f = \bigsqcup_n f_n$ is not continuous in $\frac{1}{2}$, while each f_n is. The right-continuity condition imposes that $\overline{f}(x) = 1$ for $x \in [0, \frac{1}{2}[$ and $\overline{f}(x) = \frac{1}{2}$ for $x \in [\frac{1}{2}, 1]$.
$\mathcal{IF}^0_\infty$ is the natural extension of $\mathcal{IF}^0_X$ to functions defined on $\mathbb{R}_+$. We now build the set of interval functions defined over arbitrary intervals of $\mathbb{R}$.

Definition 2 (***Arbitrary long, interval-valued functions***)**.** *The set of all continuous, partial, interval-valued functions 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:

$$f \sqsubseteq g \Leftrightarrow X_f \leq X_g \text{ and } f \sqsubseteq_{X_f} g_{|[0,X_f]} \text{ and } \forall x \in [X_f, X_g],\ g(x) \subseteq f(X_f) \tag{4}$$

where $g_{|[0,X_f]}$ denotes the restriction of g to $[0, X_f]$. Figure 3.1 gives an example

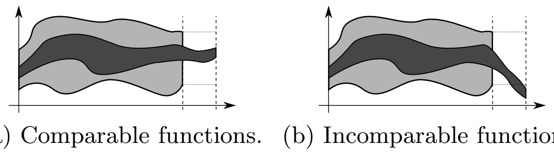

(a) Comparable functions. (b) Incomparable functions.

Fig. 3.1. Order on partially defined functions.

of comparable functions (left, the dark one being bigger than the light one) and an example of incomparable functions (right). The third hypothesis in (4) 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 $\overline{f_n}$ and $\underline{f_n}$ are bounded, thus $(\underline{f_n})$ is a bounded increasing sequence (with respect to the pointwise order for real-valued functions), so it has a limit $\underline{f}$. Equivalently, $(\overline{f_n})$ has a limit $\overline{f}$, which proves the existence of $\bigsqcup_n f_n = [\underline{f}, \overline{f}]$. 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:

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

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, $f \sqcup g = \top$.

Proposition 1. $(\mathcal{D}^0, \sqsubseteq, \top, \bot, \sqcup, \sqcap)$ *is a continuous 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).We next 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 3 (*Function composition, Primitive and Width*).
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 composition operator $\circ$ as: $\forall F : \mathbb{R} \to \mathbb{R}$ and $f \in \mathcal{D}^0,\ F \circ f = F \circ_{X_f} f$.*
The primitive *of a function $f \in \mathcal{IF}^0_X$ is $I_X(f) \in \mathcal{IF}^0_X$ defined by: $\forall x \in [0, X],\ I_X(f)(x) = \left[\int_0^x \underline{f}(s)ds, \int_0^x \overline{f}(s)ds\right]$. This primitive operator is extended to $\mathcal{D}^0$ straightforwardly: for $f \in \mathcal{D}^0$, we set $I(f) = I_{X_f}(f)$.*
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 2. *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, $i_2 \subseteq i_1 \Rightarrow w(i_2) \leq w(i_1)$, thus the monotonicity of w. The primitive operator is not monotone, as it does not preserve the third condition for the order $\sqsubseteq$ (Equation (4)). However, the second condition is preserved thanks to the monotonicity of the primitive for real-valued functions.

Among all the functions of $\mathcal{D}^0$, one is of special interest for us: y_∞, the maximal solution of (3). We compute it by successive approximations and thus need to measure the quality of our approximation. Following Keye Martin's measure theory [19], a measurement is a continuous function μ from a CPO $\mathcal{D}$ into the set of nonnegative real numbers with reverse ordering: $[0, \infty[^*$ that reveals the distance of $f \in \mathcal{D}$ to the maximal elements of $\mathcal{D}$, which have measure 0. The measurement must be coherent with the informational order on $\mathcal{D}$: the more informative f, the smaller its measure. It must also be the case that if we *measure* that the sequence 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, please refer to [19], 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 4 (*The measurement* μ). *Let $f \in \mathcal{D}^0$. We let $\mu(f) = w(f) + \frac{1}{X_f}$.*

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

Proposition 3. *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 $N \in \mathbb{N}$.*

We thus have built a lattice $\mathcal{D}^0$ and defined three operators on it: I, $\circ$ and w. We also have a measurement μ on $\mathcal{D}^0$ which characterizes its maximal elements, i.e. the real-valued functions defined on $\mathbb{R}_+$. We use μ in the next section.

3.2 The Semantics

$[\![\kappa]\!](y)$ is computed as the least fix-point of the operator $\Gamma_{F,y_0} : \mathcal{D}^0 \to \mathcal{D}^0$ that acts as follows: 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 (Sect. **??**) 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 $P_{[0,X_f]}(F, y_0)(f) = \lambda x.y_0 + \int_0^x F(f(s))ds = y_0 + I(F \circ f)$.

Definition 5 (*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:*

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

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.*

An example of the effect of Γ_{F,y_0} on a partial function is shown on Fig. 3.2. The black line represents y_∞; Figure 3.2(a) shows the updating mechanism, while Fig. 3.2(b) 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 4. *Let $f \in \mathcal{IF}^0_X$. Γ_{F,y_0} verifies the invariant:*

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

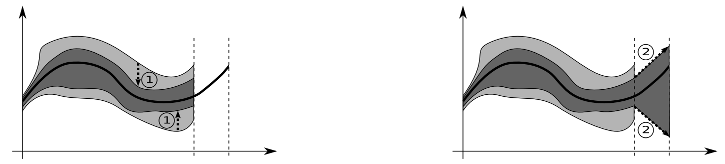
The iterates $f_{n+1} = \Gamma_{F,y_0}(f_n)$, starting from $f_0 = \bot$, form a sequence of approximation of y_∞: they enclose it and their width converge toward 0. On Table 1 the figures show how the iterates of Γ_{F,y_0} converge to a real valued function. The semantics of the continuous subsystem $\kappa = \big(F, F_{\boldsymbol{k}}, y_0\big)$ maps $f \in \mathcal{D}^0$ 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 2. *The solution y_∞ of* (3) *is a fix-point of Γ_{F,y_0} and*

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

4 Hybrid Semantics

Let us now give the semantics of the complete hybrid system. The hybrid model is a pair $\Omega = \big(\Delta, \kappa\big)$ consisting of a model Δ for the discrete system and a



(a) Update the information

(b) Extends the information

Fig. 3.2. The updating operator (two steps).

model κ for the continuous environment that define two dynamical systems that run in parallel and, from time to time, communicate. On the one hand,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. On the other hand, 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 facts. The semantics $[\![\Omega]\!]^{\mathcal{H}}$ of Ω is a function between hybrid environments. The discrete environment is altered by the discrete subsystem while the continuous one is computed only when needed, i.e. when a **sens** is found.

4.1 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 stores both the value of the variables, 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, $\Sigma_\Delta = \big\{\big(Var \to Val\big) * \big(\{time\} \to \mathbb{R}_+\big) * \big(F : \mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}\big)\big\}$. 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, i.e. 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, $\Sigma_\kappa = \big\{\big(y \in \mathcal{D}^0\big) * \big(F_{\boldsymbol{k}} \mid F_k : \mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}\big)\big\}$. As usual, we write $\sigma_\delta.X$ (resp. $\sigma_\kappa.Y$) the the value of a variable $X \in Var \cup \{time, F\}$ (resp. $Y \in \{y\} \cup F_{\boldsymbol{k}}$) in the discrete (resp. continuous) environment. We write $\Sigma^{\mathcal{H}}$ the set of all hybrid environments:

$$\Sigma^{\mathcal{H}} = \left\{ \big(\sigma_\delta, \sigma_\kappa\big) \;\middle|\; \begin{array}{l} \sigma_\delta \in \Sigma_\Delta \text{ and } \sigma_\kappa \in \Sigma_\kappa \text{ and} \\ \exists (t_n), (c_n) \text{ s.t. } \forall i \in \mathbb{N},\ \forall t \in [t_i, t_{i+1}[, \\ \qquad \sigma_\delta.F(t) = \sigma_\kappa.F_{c_i}(t) \end{array} \right\} . \tag{5}$$

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.

4.2 Hybrid Denotations

The denotation of the purely discrete parts of the language are defined as usual for imperative languages (see [24] for example). We have denotations for numerical (resp. boolean) expressions $[\![exp]\!]$ (resp. $[\![bool]\!]$) which are functions between a discrete environment and a numerical (resp. boolean) value. Every discrete statement *stmt* also has a denotation which is a function between discrete environmnents. We extend them to hybrid environments: $[\![exp]\!]^{\mathcal{H}}(\sigma_\delta, \sigma_\kappa) =$

$[\![exp]\!](\sigma_\delta)$, $[\![bool]\!]^{\mathcal{H}}(\sigma_\delta,\sigma_\kappa) = [\![bool]\!](\sigma_\delta)$, and $[\![stms]\!]^{\mathcal{H}}(\sigma_\delta,\sigma_\kappa) = [\![stms]\!](\sigma_\delta)$. The denotation of a **wait** is a function from $\Sigma^{\mathcal{H}}$ to $\Sigma^{\mathcal{H}}$ that modifies the value of time: $[\![\mathbf{wait}(c)]\!]^{\mathcal{H}}(\sigma_\delta,\sigma_\kappa) = \big(\sigma_\delta[time \mapsto \sigma_\delta.time + c, \sigma_\kappa\big)$. The denotation of an action **sens.y?x** (Equation (6) with $n = \lfloor \sigma_\delta.time + 1 \rfloor$) 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 Sect. 3.2) to $\sigma_\kappa.y$ with $F = \sigma_\delta.F$ and $y_0 = \sigma_\kappa.y(0)$.

$$[\![\mathbf{sens}.\mathrm{y?x}]\!]^{\mathcal{H}}(\sigma_\delta,\sigma_\kappa) = \begin{pmatrix} \sigma'_\kappa = \sigma_\kappa[y \mapsto \Gamma^n_{\sigma_\delta.F,y(0)}(y)], \\ \sigma'_\delta = \sigma_\delta[x \mapsto mid(\sigma'_\kappa.y(\sigma_\delta.time))] \end{pmatrix} . \tag{6}$$

The denotation of an action **act.k!c** (Equation (7)) is a function from $\Sigma^{\mathcal{H}}$ to $\Sigma^{\mathcal{H}}$ that modifies$(\sigma_\delta,\sigma_\kappa)$ 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 $\sigma_\delta.time$.

$$[\![\mathbf{act}.\mathrm{k!c}]\!]^{\mathcal{H}}(\sigma_\delta,\sigma_\kappa) = \left(\sigma_\delta \left[F \mapsto \lambda t, y. \begin{cases} \sigma_\delta.F(y,t) & \text{if } \ t \le \sigma_\delta.time \\ \sigma_\kappa.F_c(y,t) \text{ otherwise} \end{cases}\right], \sigma_\kappa\right) . \tag{7}$$

We can compute the hybrid semantics $[\![\Delta]\!]^{\mathcal{H}}$ of the discrete program by combining these denotations. This does not however compute the semantics of the continuous environment, this is the role of the semantics of the hybrid system.

4.3 Hybrid Semantics

The semantics of the hybrid model $\Omega = \big(\Delta, \kappa\big)$ is a function between hybrid environments: $[\![\Omega]\!]^{\mathcal{H}} : \Sigma^{\mathcal{H}} \to \Sigma^{\mathcal{H}}$. $[\![\Omega]\!]^{\mathcal{H}}$ alters a pair $\big(\sigma_\delta,\sigma_\kappa\big)$ as follows. It computes $\big(\sigma'_\delta,\sigma'_\kappa\big) = [\![\Delta]\!]^{\mathcal{H}}\big(\sigma_\delta,\sigma_\kappa\big)$ and two cases occur. If $\sigma'_\kappa = \sigma_\kappa$, 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 $[\![\Omega]\!]^{\mathcal{H}}\big(\sigma_\delta,\sigma_\kappa\big) = \big(\sigma'_\delta,\sigma'_\kappa\big)$. On the other hand, if $\sigma'_\kappa \ne \sigma_\kappa$, 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 $\big(\sigma_\delta,\sigma'_\kappa\big)$ 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 (8). 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 Sect. 3. $[\![\Omega]\!]^{\mathcal{H}}$ is compatible with the continuous semantics $[\![\kappa]\!]$ presented in Sect. 3: the continuous environment is finally computed as the fix-point of the operator Γ_{F,y_0} as in Sect. 3.2. It 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 [24] for example.

$$\begin{aligned} &[\![\Omega]\!]^{\mathcal{H}} = Fix(\Gamma^{\mathcal{H}}) \text{ where} \\ &\Gamma^{\mathcal{H}}(\varphi)(\sigma_\delta,\sigma_\kappa) = (\sigma'_\delta,\sigma'_\kappa) \text{ with } \begin{cases} \sigma'_\delta = \Pi_\delta\big([\![\Delta]\!]^{\mathcal{H}}(\sigma_\delta,\sigma'_\kappa)\big) \\ \sigma'_\kappa = \Pi_\kappa\left(\varphi(\sigma_\delta, \Pi_\kappa\big([\![\Delta]\!]^{\mathcal{H}}(\sigma_\kappa,\sigma_\delta)\big)\right) \end{cases} \end{aligned} . \tag{8}$$

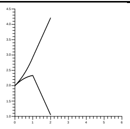
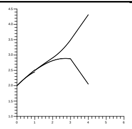
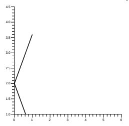
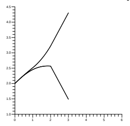
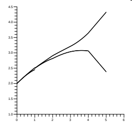
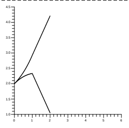
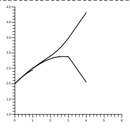
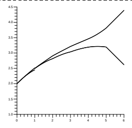
Statement	Iteration 1			Iteration 2			Iteration 3		
	t	h	x	t	h	x	t	h	x
Initial environment	0	$\perp$	$\perp$	0	$\perp$		0	$\perp$	
`wait(1);`	1			1			1		
`sens.x?h;`		2.0			2.45			2.48	
`if (h>h_max)` `act.k!1;`			$F \mapsto F_0$			$F \mapsto F_0$			$F \mapsto F_0$
`wait(1);`	2			2			2		
`sens.x?h;`		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 1. First three iterations of the semantics computation.

4.4 Example

To illustrate that our semantics really computes the behavior of the hybrid system, let us consider a simplified version of the one-tank controller (see the first column of Tab. 1). We only consider two iterations of the **while** loop (which has a period of one second) and forget about the anticipation mechanism. The continuous system is still given by (2), 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 have two possible continuous dynamics : F_0 (the valve is closed) and F_1 (the valve is open). Initially, the valve is closed, i.e. we start with the dynamic F_0. Table 1 shows the first three iterations of the computation of the semantics of the system. For each line of the program, we indicate how the variables are changed (t is the time, h the discrete variable and x the continuous one). For the **act** statement, we indicate how it changes the function F of the hybrid environment. The notation $\lambda t.(t < 2)?F_0; F_1$ means that $F(t) = F_0(t)$ if $t < 2$, and $F(t) = F_1(t)$ otherwise.

5 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 is based on a clear separation of the discrete and the continuous systems: ordinary differential equations with boolean parameters are used to model the continuous system, an imperative language with hybrid statements is used 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 use it for industrial size problems. We defined the semantics of our model in two steps: first, we extended results by Edalat and Lieutier [9] to consider the maximal solutions of IVP on $\mathbb{R}_+$ and we presented the semantics of the continuous model as a function mapping the initial condition to the maximal solution. The semantics of the hybrid system is then an extension of the standard denotational semantics of imperative languages in which actions of sensors and actuators are defined.

To the best of our knowledge, this is the first attempt to integrate into the semantics of imperative languages the continuous environment that models the programs inputs. We are not aware of any equivalent, operationally defined models. We believe that our model is expressive enough to encode most of Henzinger's hybrid automata, but both models are based on very different asumptions (for example, we consider that time is driven by the discrete system) so that it is difficult to formally compare them.

This work is a first step toward the validation of embedded software with their environment. The analysis of such systems using, for example, abstract interpretation techniques [6] requires two stages. First, the continuous system must be abstracted in a non-naive way. The theory of guaranteed integration of ODE [21] brings us the adequate tools for the safe abstraction of the continuous system. Validated ODE solvers [4] compute interval bounds that are proved to contain the solution. This can be seen as a valid abstraction in the theory of abstract interpretation. 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 [7] or of the numerical precision of floating point computations [13]). These methods must however be completed so that they consider time: 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) 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 [20] or termination analysis [2] 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, which is known to be undecidable [16]. Several methods have been proposed for its simplification [17]; 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, 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.
2. J. Berdine, A. Chawdhary, B. Cook, D. Distefano, and P. O'Hearn. Variance analyses from invariance analyses. In *POPL*, pages 211–224, 2007.

3. O. Bouissou and M. Martel. A hybrid denotational semantics of hybrid systems - extended version. http://hal.archives-ouvertes.fr/hal-00177031/.
4. O. Bouissou and M. Martel. GRKLib: a guaranteed runge-kutta library. In *International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics*. IEEE, 2006.
5. P. Cousot. Integrating physical systems in the static analysis of embedded control software. In *APLAS*, volume 3780 of *LNCS*, pages 135–138. Springer, Berlin, 2005.
6. 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.
7. 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.
8. P. Cuijpers and M. Reniers. Hybrid process algebra. *Journal of Logic and Algebraic Programming*, 62(2):191–245, 2005.
9. A. Edalat and A. Lieutier. Domain theory and differential calculus. *Mathematical Structures in Computer Science*, 14(06):771–802, 2002.
10. A. Edalat, A. Lieutier, and D. Pattinson. A computational model for multi-variable differential calculus. In *FOSSACS*, volume 3441 of *LNCS*. Springer, 2005.
11. A. Edalat and D. Pattinson. Denotational Semantics of Hybrid Automata. In *FOSSACS*, volume 3921, pages 231–245, 2006.
12. R. Alur et al. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.
13. 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.
14. E. Goubault, M. Martel, and S. Putot. Some future challenges in the validation of control systems. In *ERTS*, 2006.
15. V. Gupta, R. Jagadeesan, and V. Saraswat. Computing with continuous change. *Science of Computer Programming*, 30(1–2):3–49, 1998.
16. T. A. Henzinger. The theory of hybrid automata. In *Symposium on Logic in Computer Science*, pages 278–292. IEEE Computer Society Press, 1996.
17. 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.
18. 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.
19. K. Martin. *A Foundation for Computation.* PhD thesis, Department of Mathematics, Tulane University, 2000.
20. A. Mycroft. *Abstract interpretation and Optimizing Transformations for Applicative Programs.* PhD thesis, University of Edinburgh, 1981.
21. N.S. Nedialkov, K.R. Jackson, and G.F. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Applied Mathematics and Computation*, 105(1):21–68, 1999.
22. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
23. 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.
24. G. Winskel. *The formal semantics of programming languages: an introduction.* MIT Press, Cambridge, MA, USA, 1993.

A Proofs

Proof of Proposition 1
Each underlying set $\mathcal{IF}_X^0$ is a continuous Scott domain (see [9] for a complete proof of that). Our extended order was specially designed so that it preserves this structure (see discussion of Equation (4)). 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. $\square$

Proof of Proposition 3
Point **(i)** is a straightforward consequence of Proposition 2. 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, $\lim_n \underline{f}_n(x) = \lim_n \overline{f}_n(x) = f(x)$ and consequently $\bigsqcup_n f_n = f$. $\square$

Proof of Proposition 4
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 $f_\infty(x) \in f'(x)$. $\square$

Proof of Theorem 2
This is a straightforward consequence of Propositions 3 and 4.