Static Analysis by Abstract Interpretation of Hybrid Systems

Olivier Bouissou (olivier.bouissou@cea.fr) and Matthieu Martel (matthieu.martel@cea.fr)
CEA - Recherche Technologique
LIST-DTSI-SOL-LSL
CEA F91191 Gif-Sur-Yvette Cedex, France

Abstract. In this article, we introduce a new way of modelling and analyzing hybrid systems. Our model insists on a clear separation between the discrete and continuous worlds, making it possible to use with little effort existing programs. The analysis, based on abstract interpretation, can prove safety properties concerning the use of floating point numbers instead of real numbers in hybrid systems. For this analysis, we developed a new domain, the domain of step-wise functions. Our abstraction consist of an abstraction of time in order to make the environments representable and of an abstraction of the continuous functions to make them computable.

Keywords: Hybrid systems, numerical precision, abstract interpretation, Runge-Kutta methods.

1. Introduction

Embedded softwares play a crucial role in our every day life, from driving assistance to power plant control programs. They are often safety-critical applications and run in a physical environment with which they continuously interact. Such dynamical systems whose evolution depends on both a discrete (e.g. the automatic pilot of a plane) and a continuous system (the physical environment) are called *hybrid systems*. The interest for modelling and analyzing such systems has experienced an impressing increase in the last few years, from both the academic and industrial worlds. Hybrid systems are mostly described using Hybrid Automatas (Henzinger, 1996), which combine discrete transition graphs with continuous dynamical systems. This framework makes it possible a quick modelling of hybrid systems and validation tools like HyTech (Henzinger et al., 1997) allow, in some basic cases, automatic verification, mostly using model checking techniques.

Static analyzers for critical embedded softwares (Blanchet et al., 2002; Goubault et al., 2002) often poorly abstract the physical environment in which, in practice, the embedded systems are run. To take an extreme example (more reasonable examples abound in articles dedicated to hybrid systems (Alur et al., 1995; Mosterman, 1999)),

the static analysis of avionic codes should abstract the plane environment, that is, the engines, the wings, and the atmosphere itself. This is because embedded software strongly interact with their environment, picking up physical values by means of sensors and modifying them via actuators. In practice, the sensors correspond to volatile variables in C programs and, at analysis time, the user assigns to these variables a range given by the minimal and maximal values the sensor can send. In this case, a static analyzer must assume that the value sent by the sensor may switch from its minimum to its maximum in arbitrary short laps of time, while, in practice, it follows a slow evolution. As a consequence, the results of the static analysis are significantly over-approximated. The abstraction of the physical environment is even more crucial for embedded systems that cannot be physically tested in their real environment, like space crafts whose safety only relies on verification tools.

In this article, we introduce a new formalism for modelling hybrid systems. This formalism is very close to imperative programming languages like C and includes a description of the physical environment and interaction mechanisms that model sensors and actuators. This makes it suitable for modelling and analyzing existing codes used in the industry. In a second time, we present a static analysis which tends to prove safety properties concerning the use of floating-point numbers in the programs in order to represent real numbers. Actually, even though these two domains have completely different properties, floating-point arithmetic is widely used in critical applications. As a consequence, it is crucial to check that the behaviour of the system is not affected by the loss of precision due to the representation of real numbers on a finite number of bits. The safety properties we want to prove may be formulated as follows: an hybrid system is said to be safe if its ideal execution in a model using real numbers and its implementation using floating-points have close behaviours. The notion of behaviour closeness means that if the model makes a critical action at a time t, then the implementation makes the same action at a time u near t, i.e. such that $u \in [t - \delta, t + \delta]$ for a given δ. The critical actions we consider are all the actions performed by the discrete program on its continuous environment (via actuators) and a set of specific user specified discrete actions. This safety property is graphically represented in Figure 1.

This new model and its concrete semantics are introduced in Section 2. An abstraction of the discrete evolution is proposed in Section 3 and an algorithm for abstracting the continuous evolution is quickly explained in Section 4.

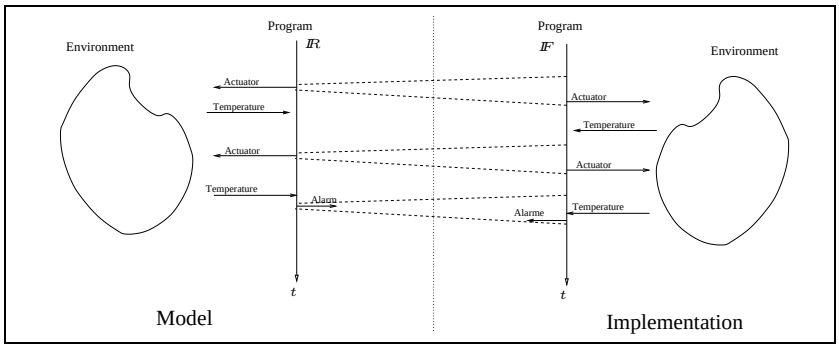

Figure 1. Safety Property for an Hybrid System.

Technical Contribution

Before introducing the language and its semantics, we first explain and motivate our choices for the concrete and abstract domains, which are the main contribution of this article. First of all, there is the problem of time. It is crucial to estimate with enough precision at what time every instruction is executed. Without information on time, we don't know when the action of a sensor is performed and we cannot be more precise for the value of the measured physical parameter than what is done for the moment, i.e. approximating it by a safe interval, like $[-1, 1]$. Fortunately, the syntax of programs gives explicit indications on time, via special delay commands. As other instructions are considered to be instantaneous, it is possible to evaluate the execution time of each instruction, by simply considering every environment as a pair $(Value\ of\ the\ variables, Time)$.

Secondly, we want to have a model which allows parallelism. This causes an issue as there are different processes acting at the same time on one physical environment, using the same actuators. So, the evolution of an actuator depends on the instructions found in every processes of the parallel system. Consequently, we define a global environment representing the evolution of all actuators. In addition to that, this global environment enables us to compute the evolution of continuous variables, which strongly depends on the evolution of actuators. So, yet another global environment for continuous variable must be defined.

Finally, we address the problem of the evaluation of continuous variables. This evaluation does not need to be carried out in parallel with the computation of the discrete semantics, but only when the discrete semantics needs it, i.e. only when asking for the value of a physical parameter via a sensor. Therefore, the value of a continuous variable may be lazily calculated. However, such a lazy evaluation has its drawback: it is necessary to remember all the past values of every ac-

tuators. Consider for instance the following sequence of actions, where an interrupter `I` controls the heating of a room and a sensor gives the temperature `t`:

```
Switch I on; Wait 1s; Switch I off; Wait 1s; Get t
```

When we evaluate the value of `t` at time 2, we must remember that the interrupter was on between $t = 0$ and $t = 1$ and off afterwards. So, considering the interrupter as a boolean value only is not sufficient, we must consider it as a step function from $\mathbb{R}^+$ to $\mathbb{B}$, thus all its past values are recorded.

2. Concrete Semantics

2.1. Syntax

In order to be able to reuse existing codes, we need to provide a model in which the discrete and the continuous parts are clearly separated and isolated. Therefore, we first separate the variables of the system which represent the environment from the variables which are used in the discrete part. Moreover, the program may have an influence on the continuous environment and may change its dynamics thanks to actuators. For this reason, we define special variables, called *shared variables*, which represent these actuators. These variables are used to simulate the closing of a door, the starting of the heater, etc. In other words, they are used to choose between two different continuous dynamics and, consequently, it is not restrictive to impose that they are only booleans. We can now define an hybrid system.

DEFINITION 1. Hybrid System
An hybrid system is a five-tuple $< C, S, X, P_E, P >$*, such that:*

- P_E *is the physical environment,*
- P *is the program controlling the environment,*
- C *is the set of* continuous *variables,*
- S *is the set of* shared *variables (actuators),*
- X *is the set of* discrete *variables that are used by* P*.*

The sets C*,* S *and* X *must be pairwise disjoint.*

The syntax for the environment P_E and for the program P is defined in the next two sections.

2.1.1. *Continuous system*
The environment models the physical system in which the program is run. Its dynamics is defined by differential equations which describe how the continuous variables $c \in C$ behave according to their derivatives $\dot{c}$. We assume that the functions defined by these equations are piecewise $\mathcal{C}^2$ which makes our approximation algorithm, described in Section 4, to work. To simulate the effect of actuators, shared variables from S may appear in these differential equations, such that if the program changes the value of one of these variables, the dynamics of the continuous system changes. As the environment may change its dynamics by itself, it is modeled as a guarded sum of differential systems.

DEFINITION 2. The Continuous Environment
The environment is a process P_E that may be written as a sum:

$$P_E = b_0 \rightarrow \begin{pmatrix} F_0 \\ I_0 \end{pmatrix} \Box \cdots \Box b_n \rightarrow \begin{pmatrix} F_n \\ I_n \end{pmatrix}$$

Each b_i is a conjunction of constraints $b_i = b_i^1 \wedge \cdots \wedge b_i^n$, such that $b_i^j = \{c \leq r\}$ or $b_i^j = \{c = r\}$ for some $c \in C$ and $r \in \mathbb{R}$. F_i is called a flow condition*, i.e. a function $\dot{C} \rightarrow \mathcal{C}^1(\mathbb{R}^{|C|\times|S|})$, where $\dot{C}$ is the set of all the derivatives of the continuous variables. I_i is an* invariant*, i.e. a predicate on the continuous variables whose syntax is the same as for the b_i .*

The flow condition describes the evolution of the continuous variables in a given state and the invariant states how long the system may stay in this state. As this guarded sum should represent the evolution of the real world, we shall assume that it never deadlocks, i.e. that there cannot be a situation in which none of the b_i's is true. Therefore, is must be true that $b_0 \vee \cdots \vee b_n \equiv true$.

Note. The differential equations F_i are autonomous, i.e. they do not depend on time. This is not a restriction, as it is a trivial matter to transform a differential system $\dot{Y} = F(Y, t)$ into an autonomous one $\dot{Z} = G(Z)$.

2.1.2. *The discrete system*
The *program* describes the discrete part of the hybrid system. We want to have a model in which one can easily use the programs that are already analyzed by Fluctuat (Goubault et al., 2002), but also a theoretical model with a formal semantics. Therefore, our formalism is inspired by the Timed CSP (TCSP, (Reed and Roscoe, 1988)), the

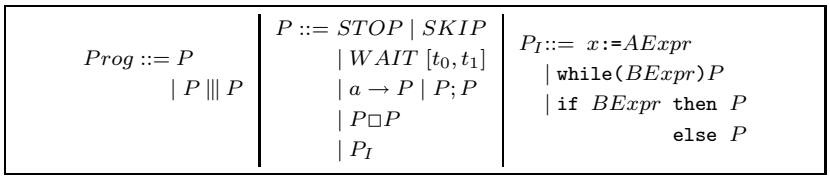

Figure 2. Syntax for a Program.

timed version of Communicating Sequential Processes (CSP, (Hoare, 1985)). We add imperative instructions and special actions to model the sensors and actuators. A program is therefore a parallel composition of sequential processes, built on a set of actions which are:

- $A_C = \bigcup_{c \in C} \{sens.c?x | x \in X\}$ which represents the effect of sensors. Such an action assigns the value of the continuous variable c to the discrete variable x,
- $A_S = \bigcup_{s \in S} \{act.s!0, act.s!1\}$ which represents the effect of actuators. Such an action sets the shared variable s to the boolean value 0 or 1,

The set of all actions is denoted $A = A_C \cup A_S$.

For simplicity reasons, we don't allow here communication between parallel processes P and Q (denoted $P|||Q$). However, as the continuous environment is common to all the processes, it can be used to simulate a synchronisation mechanism: P reads the value of a continuous variable c, whose dynamics is changed by Q. When P sees that the dynamics has changed, it understands that Q has performed an *act* action, so that both processes have synchronised. We assume that each process handles its own set of discrete variables, which are not known from the other processes. If the program P is the composition of n processes, $P = P_1 ||| \ldots ||| P_n$, then there exists pairwise disjoints sets $X_1, \ldots, X_n$ such that $X = X_1 \cup \ldots \cup X_n$ and such that X_i is the set of discrete variables occurring in P_i.

A process is a sequence of the following imperative instructions (see Figure 2): $SKIP$, which does nothing, or $STOP$ which blocks the process, $WAIT[u_0, u_1]$, which stops the process for a time $t \in [u_0, u_1]$, $P \square Q$, the non-deterministic external choice, $a \rightarrow P$, with $a \in A$, which executes the action a and behaves as P and imperative instructions: assignments, conditional tests, loops. For the sake of simplicity, we only allow arithmetic expressions ($AExpr$) using the four basic operators $+, -, *$ and $/$, and boolean expressions ($BExpr$) are limited to tests like $x \leq c$ or $x < c$, where c is a constant, as well as the two boolean constants $true$ and $false$. Note that this syntax is very close to imperative programs, the additional constructors (*act* and *sens* actions, $WAIT$ or

even parallel composition) are easy to identify in the industrial codes (or their comments) that are studied for the Fluctuat analyzer.

2.2. Equational Semantics

In this section, we define the concrete collecting semantics of our model. Classically, we later define an abstract domain and an abstract semantics in order to use abstract interpretation techniques. Our concrete semantics is defined as the least fix-point of a system of equations defining the relations between the environments before and after the execution of each control point of the program. These equations define the discrete evolution of the system as well as queries to compute the continuous dynamics. The continuous dynamics can be computed via an approximation algorithm which is explained in Section 4. The (concrete or abstract) semantics of the hybrid system therefore is a collaboration between two resolution methods: one for the discrete system and one for the continuous one. This separation is made possible by the complete separation between the discrete and the continuous sub-systems in our model.

2.2.1. *The concrete domains*

The concrete semantics of an hybrid system entails two kinds of environments: the local ones, defining the discrete dynamics of the system and the global ones, used to compute the continuous dynamics, both of them being linked by a compatibility relation. Let $\mathcal{L}$ be the set of all the control points of the program; the control points of the i^{th} process are written as $i - n$, with $n \geq 1$. For each $l \in \mathcal{L}$, two local environments are defined: χ^l and $\dot{\chi}^l$ denoting the environments containing the value of the variables just before and just after the execution of l. Above, the global environments are denoted χ_S^0 for the shared variables and χ_G^0 for the continuous ones. To define the semantics of the global environments we introduce intermediary environments, χ^{i-0}, denoting the global evolution of shared variables inside the i-th process. In the following, we first define local environments and then global ones. These environments strongly depend on the notion of step function, which is defined below.

DEFINITION 3. Step Functions
Let $Env_{SF}(D)$ *be the set of all finite step functions with values in* D. $f \in Env_{SF}(D)$ *if and only if* $\exists\ 0 = t_1 \leq t_2 \leq \cdots \leq t_n = t$ *such that* f *is constant on* $[0, t_1[, [t_1, t_2[, \cdots, [t_{n-1}, t_n[, [t_n, +\infty[$.

A step function can be written as a finite conjunction of constraints, as proposed in (Bertrane, 2005): $f = \langle t_1, t_2 \rangle : \alpha_1 \wedge \ldots \wedge \langle t_{n-1}, t_n \rangle : \alpha_n \wedge \langle t_n, +\infty \rangle : \alpha_{n+1}$.

DEFINITION 4. Step Functions Denotation
Every function $f \in Env_{SF}(\mathcal{D})$ can be written as the conjunction of constraints $\langle u, v \rangle : \alpha$, with $u, v \in \mathbb{R}^+$ and $\alpha \in \mathcal{D}$. $\langle u, v \rangle : \alpha$ means that $\forall t \in [u, v[, f(t) = \alpha$.

We now define the local environment related to a control point l, which maps every discrete variable to its value and every shared variable to a step function with values in $\mathbb{B}$. In addition the environment must contain the time at which the control point is run. Let $\mathbb{D}_{\mathbb{X}}$ be the numerical domain for discrete variables $x \in X$. We assume that $\mathbb{D}_{\mathbb{X}}$ is equipped with the four basic arithmetic operations $\oplus_{\mathbb{D}_{\mathbb{X}}}, \otimes_{\mathbb{D}_{\mathbb{X}}}, \oslash_{\mathbb{D}_{\mathbb{X}}}, \ominus_{\mathbb{D}_{\mathbb{X}}}$ as well as a cast from real values $v \in \mathbb{R}$ into $\mathbb{D}_{\mathbb{X}}$: $[\![v]\!]_{\mathbb{D}_{\mathbb{X}}}$.

DEFINITION 5. Local Environments
A local environment $\rho \in \mathcal{D}_{\mathcal{L}}$ is a triple (ρ_X, ρ_S, ρ_t) such that $\rho_X : X \rightarrow \mathbb{D}_{\mathbb{X}}$ is an environment for discrete variables, $\rho_S : S \rightarrow Env_{SF}(\mathbb{B})$ is an environment for actuators and $\rho_t \in \mathbb{R}^+$ is a time. In addition to that, we require that:

$$\begin{cases} \forall \alpha \in S, \rho(\alpha) = c_0 \wedge \ldots \wedge c_n \\ \quad \textit{with } c_n = \langle t_n, \infty \rangle : \beta \textit{ and } t_n \leq \rho_t \end{cases} \tag{1}$$

The condition of Equation 1 assesses that no action arises at a time greater than the current time.

To deal with the non-determinism if the language (the $\square$ or $WAIT[u, v]$ instructions), semantic equations use sets of such environments. So, we have for every control-point $l \in \mathcal{L}$:

$$\chi^l, \dot{\chi}^l \in \mathcal{P}(\mathcal{D}_{\mathcal{L}})$$

The usual set operators $(\cup, \cap)$ give a lattice structure to the local domain $\mathcal{P}(\mathcal{D}_{\mathcal{L}})$. Let $\chi^l \in \mathcal{P}(D_l)$, we write χ^l_S the related local shared environment defined by: $\chi^l_S = \left\{ \rho_S \mid \rho = (\rho_X, \rho_S, \rho_t) \in \chi^l \right\}$.

As mentioned in the introduction, we are mostly interested in computing the trace left by a program, i.e. the sequence of observable actions done during its execution. Thus, a trace, for an actuator $s \in S$, is a word of the language $\mathbb{T}$, defined in by:

DEFINITION 6. The set of traces
Let $\mathbb{T}$ be the set of all words $< t_1, s, \alpha_1 > . \cdots . < t_n, s, \alpha_n >$ such that $t_i \in \mathbb{R}^+$, $s \in S$ and $\alpha_i \in \mathbb{B}$ for all $i \in [1, n], n \in \mathbb{N}^*$.

Then, the trace left by a control point $l \in \mathcal{L}$ is defined by the following equations.

$$trace(c_0 \wedge \ldots \wedge c_n, s) = < t_1, s, \alpha_1 > . \cdots . < t_n, s, \alpha_n > \mid c_i = \langle t_i, t_{i+1} \rangle : \alpha_i \quad (2)$$

$$trace(\rho_S) = \{trace(\rho_S(s), s) \mid s \in dom(\rho_S)\} \quad (3)$$

$$trace(\chi) = \bigcup_{(\rho_X, \rho_S, \rho_t) \in \chi} trace(\rho_S) \quad (4)$$

$$trace(l) = trace(\chi^l) \cup trace(\dot{\chi}^l) \quad (5)$$

We now define the global environments which describe the evolution of shared and continuous variables. As already mentioned, we define for each process an environment which represents the evolution of shared variables within this process. This environment, denoted χ^{i-0} for the i^{th} process, is called an *intermediary* environment as it is intermediary between local and global environments. The global environment for shared variables is then computed as the interleaving of all such intermediary environments.

Intuitively, to compute χ^{i-0}, we only need to know the maximal values of each shared variable in order to compute the global evolution, which strongly reduces the size of the global environments. The notion of maximality is here given by a specific order on step functions, given in Equation 8, which may be seen as a kind of prefix order.

DEFINITION 7. The partial order $\sqsubseteq_{SF}$.

$$\langle t_1, \infty \rangle : \alpha \sqsubseteq_{SF} \langle t_1', t_2' \rangle : \alpha' \wedge C \iff t_1 = t_1' \text{ and } \alpha = \alpha' \quad (6)$$

$$\langle t_1, t_2 \rangle : \alpha \sqsubseteq_{SF} \langle t_1', t_2' \rangle : \alpha' \iff t_1 = t_1' \text{ and } \alpha = \alpha' \text{ and } t_2 \leq t_2' \quad (7)$$

$$c_1 \wedge \ldots \wedge c_n \sqsubseteq_{SF} c_1' \wedge \ldots \wedge c_m' \iff c_1 \sqsubseteq_{SF} c_1' \text{ and } c_2 \wedge .. \wedge c_n \sqsubseteq_{SF} c_2' \wedge .. \wedge c_m' \quad (8)$$

Using the order $\sqsubseteq_{SF}$, we next define a pre-order on local shared environments:

DEFINITION 8. The pre-order $\sqsubseteq$.
Let A and B be two sets $A, B \in \mathcal{P}((S \rightarrow Env_{SF}(\mathbb{B})))$. Then:

$$A \sqsubseteq B \Longleftrightarrow \forall a \in A, \exists b \in B \mid \forall s \in S,\ a(s) \sqsubseteq_{SF} b(s) \quad (9)$$

This pre-order defines an equivalence relation $\equiv_{SF}$. Then, the domain for intermediary environments is given by Definition 9.

DEFINITION 9. Intermediary Environments
Let $\mathcal{P}_{\mathcal{GL}} = (\mathcal{P}(\mathcal{D}_{\mathcal{GL}}))_{/\equiv_{SF}}$ be the domain for intermediary environments, with $\mathcal{D}_{\mathcal{GL}} = (S \rightarrow Env_{SF}(\mathbb{B}))$. We use the order $\sqsubseteq$ on this domain.

For any set $A \in \mathcal{P}(\mathcal{D}_{\mathcal{GL}})$, its equivalence class $[A]$ is represented by the set of smallest cardinality, which is clearly uniquely defined as the set

containing only $\sqsubseteq_{SF}$-maximal, not comparable elements. The set $\mathcal{P}_{\mathcal{GL}}$ is then given a lattice structure via the following operators:

$$[A] \sqcup [B] = [A \cup B] \; and \; [A] \sqcap [B] = [A \cap B] \tag{10}$$

We may now define the two global environments: χ_S^0 for shared variables and χ_G^0 for continuous ones. χ_S^0 unifies the local evolutions of each shared variable computed separately in each process and contained in χ^{i-0}. This unification may be understood as follows: if a process P performed an action $act.s!0$ at time $t = 0$ and P' performed the action $act.s!1$ at time $t = 1$, then χ_S^0 must have a function which contains these two actions, for example $\langle 0, 1\rangle : 0 \wedge \langle 1, +\infty\rangle : 1$. Actions coming from different parallel processes are combined. However, if we only merge functions, we obtain inexact behaviours, as shown in Example 1.

Example 1. Let us consider the two programs: P_1 defined by

```
act.s!0->WAIT [1,1];
{
  act.s!1->WAIT[0.5,0.5];       WAIT[2,2];
  act.s!0->WAIT[1.5,1.5];  []  act.s!1->sens.c?x->x:=x+1;
  act.s!1->SKIP
}
```

and P_2 defined by:

```
act.s!0->WAIT [1,1];
act.s!1->WAIT[0.5,0.5];  |||  act.s!0->WAIT [3,3];
act.s!0->WAIT[1.5,1.5];       act.s!1->sens.c?x->x:=x+1;
act.s!1->SKIP
```

P_1 has two global evolutions for shared variables, corresponding to the two sides of the non-deterministic choice, namely; $\langle 0, 3\rangle : 0 \wedge \langle 3, +\infty\rangle : 1$ and $\langle 0, 1\rangle : 0 \wedge \langle 1, 1.5\rangle : 1 \wedge \langle 1.5, 3\rangle : 0 \wedge \langle 3, +\infty\rangle : 1$. The global evolution of P_2 is: $\langle 0, 1\rangle : 0 \wedge \langle 1, 1.5\rangle : 1 \wedge \langle 1.5, 3\rangle : 0 \wedge \langle 3, +\infty\rangle : 1$. However, when executing the *sens* action in P_1, the only first action ($\langle 0, 3\rangle : 0 \wedge \langle 3, +\infty\rangle : 1$) should be considered, whereas in P_2 the second one must be used. So, a way to distinguish the two programs must be provided.

This example shows that more information must be collected in the global environment and a relation between global and local environments must be established. This is done by adding to each step of the functions of χ_S^0 a vector of booleans, $B \in \mathbb{B}^n$, such that the ith element of B is true if and only if the step may have been created by the ith process. With this information, we may compute the trace left by the ith process in the global environment and then define a compatibility relation between local and global environments using this trace.

DEFINITION 10. Global Shared Environment
Let $(\mathcal{P}(\mathcal{D}_\mathcal{S}))_{/_{\equiv_{SF}}}$ *be the domain for* χ^0_S*, with* $\mathcal{D}_\mathcal{S} = S \rightarrow Env_{SF}(\mathbb{B}\times\mathbb{B}^n)$*. This domain is given a lattice structure using the order* $\sqsubseteq$ *(see Equation 2.7) and the operators* $\sqcup$ *and* $\sqcap$*.*

The domain for continuous variables may now be defined. A continuous variables $c \in C$ represents the dynamics of a physical parameter such as temperature, speed, height etc. Therefore, it should be a continuous function. We require this function to be piecewise $\mathcal{C}^2$, for our approximation algorithm to work (see Section 4). As it is difficult to compute even an approximation of the solution of differential equations when time goes to infinity, this function will only be defined on a finite interval $[0, t]$, with $t \in I\!R^+$. Eventually, we need to have a compatibility relation between a local and a global environment. This compatibility relation may be understood as follows : a local environment ρ and a global continuous one ρ_G are compatible if and only if ρ_G results from a global shared environment ρ_S that is compatible with ρ. So, we must associate to the continuous function denoting the evolution of the continuous variable the global shared environment that created this evolution.

DEFINITION 11. Global Continuous Environment
Let $\mathcal{P}(\mathcal{D}_\mathcal{G})$ *be the domain for* χ^0_G*, with*
$\mathcal{D}_\mathcal{G} \subseteq (I\!R^+ \rightarrow I\!R^{|C|}) \times (S \rightarrow Env_{SF}(\mathbb{B} \times \mathbb{B}^n))$ *and*

$$\rho_G \in \mathcal{D}_\mathcal{G} \Leftrightarrow \rho_G = (f_c, f_s) \left| \begin{array}{l} \exists t \in I\!R^+ | \forall u > t, f_c(u) = \bot \\ \exists 0 = t_0 < t_1 < \ldots < t_n = t \mid \forall i \in [0, n-1], f_{c_{|[t_i,t_{i+1}]}} \text{ is } \mathcal{C}^2 \\ f_s \in \mathcal{D}_\mathcal{S} \end{array} \right.$$

For $\rho_G = (f_c, f_s) \in \mathcal{D}_\mathcal{G}$, ρ_{G_t} denotes the minimal time $t \in I\!R^+$ such that $\forall u > t, f_c(u) = \bot$. We also define the trace left by the ith process in the global environment:

$$trace(c_0 \wedge . \wedge c_n, s, i) = < t_1, s, \alpha_1 > . \cdots . < t_n, s, \alpha_n > \left| \begin{array}{l} c_j = \langle t_j, t_{j+1} \rangle : (\alpha_j, b_j) \\ b_j(i) = 1 \end{array} \right. \quad (11)$$

$$trace(g, i) = \{trace(g(s), s, i) \mid s \in dom(g)\} \quad (12)$$

$$trace(\rho_G, i) = trace(g, i) \mid \rho_G(u) = \begin{pmatrix} f(u) \\ g(u) \end{pmatrix} \text{ and } g \in Env_{SF}(\mathbb{B} \times \mathbb{B}^n) \quad (13)$$

$$trace(\chi_G, i) = \bigcup_{\rho_G \in \chi_G} trace(\rho_G, i) \quad (14)$$

2.2.2. *Concrete Semantics : Equations*
We now define the semantic equations, which are statically built, as the program is parsed. A set of equations is associated to each construction of the language, via an operator $[\![.]\!]$, which is defined by equations $(E-1)$ to $(E-11)$.

Imperative Instructions $(E-1)$ *to* $(E-4)$
The semantics of imperative instructions is usual. We use an evaluation operator $[\![e]\!]_{\mathbb{D}_{\mathbb{X}}}\rho$ for arithmetic expressions, which computes the value of e using the operation of the numerical domain $\mathbb{D}_{\mathbb{X}}$, and $[\![e]\!]_b$ for boolean expressions.

$$\mathbb{L}(x := e)^l\mathbb{J} = \left\{ \dot{\chi}^l = \bigcup_{\rho\in\chi^l} \{\rho[x \mapsto [\![e]\!]_{\mathbb{D}_{\mathbb{X}}}\rho]\} \right\} \tag{$E-1$}$$

$$\mathbb{L}(P^{l_1}; P^{l_2})^l\mathbb{J} = \left\{ \chi^{l_1} = \chi^l \ ; \ \chi^{l_2} = \dot{\chi}^{l_1} \ ; \ \dot{\chi}^l = \dot{\chi}^{l_2} \right\} \cup \mathbb{L}P^{l_1}\mathbb{J} \cup \mathbb{L}P^{l_2}\mathbb{J} \tag{$E-2$}$$

$$\mathbb{L}(\texttt{if}(e)\ \texttt{then}\ P^{l_1}\ \texttt{else}\ P^{l_2})^l\mathbb{J} = \left\{ \begin{array}{l} \chi^{l_1} = \bigcup_{\rho\in\chi^l} \{\{\rho\} \cap [\![e]\!]_b\}\,; \\ \chi^{l_2} = \bigcup_{\rho\in\chi^l} \left\{\{\rho\} \cap \overline{[\![e]\!]_b}\right\}; \\ \dot{\chi}^l = \dot{\chi}^{l_1} \cup \dot{\chi}^{l_2} \end{array} \right\} \cup \mathbb{L}P^{l_1}\mathbb{J} \cup \mathbb{L}P^{l_2}\mathbb{J} \tag{$E-3$}$$

$$\mathbb{L}(\texttt{while}(e)P^{l_1})^l\mathbb{J} = \left\{ \begin{array}{l} \chi^{l_1} = \bigcup_{\rho\in\chi^l\cup\dot{\chi}^{l_1}} \{\rho\} \cap [\![e]\!]_b \\ \dot{\chi}^l = \bigcup_{\rho\in\chi^l\cup\dot{\chi}^{l_1}} \{\rho\} \cap \overline{[\![e, \rho(t)]\!]_b} \end{array} \right\} \cup \mathbb{L}P^{l_1}\mathbb{J} \tag{$E-4$}$$

Actions $(E-5)$
$act.s!b$ sets the value associated to the shared variable s to the boolean b. However, the past value of s must not be forgotten. So, the effect of $act.s!b$ is to add a new step to the function representing s. We therefore define the operator $\&$. Let $f = c_0 \wedge \cdots \wedge c_n \in Env_{SF}(\mathbb{B})$, $c_n = \langle t_n, +\infty\rangle : \alpha_n$, with $\alpha_n \in \mathbb{B}$. Let $t \geq t_n$ be the time where the action is executed and $b \in \mathbb{B}$. Then

$$f\&(b,t) = c_0 \wedge \cdots \wedge c_{n-1} \wedge \langle t_n, t\rangle : \alpha_n \wedge \langle t, +\infty\rangle : b$$

Let us remark that $f \sqsubseteq_{SF} f\&(t,b)$.

$$\mathbb{L}(act.s!b \rightarrow P^{l_1})^l\mathbb{J} = \left\{ \begin{array}{l} \dot{\chi}^l = \dot{\chi}^{l_1} \\ \chi^{l_1} = \bigcup_{\rho\in\chi^l} \{\rho[s \mapsto \rho(s)\&([\![b]\!]_{\mathbb{B}}\rho, \rho_t)]\} \end{array} \right\} \cup \mathbb{L}P^{l_1}\mathbb{J} \tag{$E-5$}$$

Delays $(E-6)$
$WAIT[u,v]$ delays the process for a non-deterministically chosen laps of time $\delta \in [u,v]$. Therefore, it just adds δ to the time entailed in the environment associated to this control point.

$$\mathbb{L}(WAIT[u,v])^l\mathbb{J} = \left\{ \dot{\chi}^l = \bigcup_{\delta\in[u,v],\rho\in\chi^l} \{\rho[\rho_t \mapsto \rho_t + \delta]\} \right\} \tag{$E-6$}$$

Choice ($E-7$)
The semantic equation for the external choice is straightforward.

$$[\![(P^{l_1} \square P^{l_2})^l]\!] = \left\{ \chi^{l_1} = \chi^l \ ; \ \chi^{l_2} = \chi^l \ ; \ \dot{\chi}^l = \dot{\chi}^{l_1} \cup \dot{\chi}^{l_2} \right\} \cup [\![P^{l_1}]\!] \cup [\![P^{l_2}]\!] \qquad (E-7)$$

Parallel composition ($E-8$)
The intermediary environment of the ith process is defined as the join (in the sense of $\sqcup$, see Equation 10) of all local environments contained in the ith process. The global shared environment is defined as the interleaving of all these intermediary environments. This interleaving is defined by the $|||$ operator (see below) which functions as follows: let $f = \langle t_0, t_1 \rangle : \alpha_1 \wedge \ldots \wedge \langle t_n, +\infty \rangle : \alpha_n$ and $f' = \langle t_0, t'_1 \rangle : \beta_1 \wedge \ldots \wedge \langle t'_m, +\infty \rangle : \beta_m$ be two step functions and $g = f \ ||| \ f'$ be their interleaving. Then g is the set of all functions that entail all the actions performed by f and by f'. If the two functions execute an action at the same time, i.e. there exists $t \in I\!R^+$ such that f and f' have a constraint of the form $\langle t, u \rangle : b$ for some $u \in I\!R^+$ and $b \in \mathbb{B}$, then the operator creates two functions: one where the action from f is executed first and then the action from f', and one with the opposite. Formally, g is defined as:

$$g = \left\{ \langle t_0, t''_1 \rangle : \gamma_1 \wedge \ldots \wedge \langle t''_l, +\infty \rangle : \gamma_l \;\middle|\; \begin{array}{l} t''_i \leq t''_{i+1}, \\ \{t''_k\}_{1 \leq k \leq l} = \{t_i\}_{1 \leq i \leq n} \cup \{t'_j\}_{1 \leq j \leq m}, \\ \gamma_k = (g_k, b_k) \wedge \left\{ \begin{array}{l} t''_k = t_i \Rightarrow \left\{ \begin{array}{l} b_k(1) = 1 \\ b_k(2) = 0 \end{array} \right. \\ t''_k = t'_j \Rightarrow \left\{ \begin{array}{l} b_k(2) = 1 \\ b_k(1) = 0 \end{array} \right. \\ g_k = \left\{ \begin{array}{c} \alpha_i \ if \ t''_k = t_i \\ or \\ \beta_j \ if \ t''_k = t_j \end{array} \right. \end{array} \right. \end{array} \right\}$$

It is a trivial matter to extend this definition to the parallel composition of n processes.

$$[\![P^1 ||| \ldots ||| \ P^n]\!] = \left\{ \begin{array}{l} \chi_S^{i-0} = \bigsqcup_{l \in i} [\chi^l] \cup [\dot{\chi}^l] \ ; \\ \chi_S^0 = \chi_S^{1-0} ||| \ldots ||| \ \chi_S^{1-n} \ ; \end{array} \right\} \cup \bigcup_{i \in [1,n]} [\![P^i]\!] \qquad (E-8)$$

Sens actions ($E-9$)
For *sens* actions, we introduce three notions. First, a compatibility relation which links a local environment with a global one, such that both are compatible. A local environment $\rho \in \mathcal{D}_{\mathcal{L}}$ coming from the ith process and a global one $\rho_{_G} \in \mathcal{D}_{\mathcal{G}}$ are compatible if all the actions performed by the ith process in $\rho_{_G}$ are performed by ρ and if all actions performed made by ρ occur in $\rho_{_G}$. Formally, we have:

$$\rho \ and \ \rho_{_G} \ are \ compatible \iff trace(\rho) = trace(\rho_{_{G\,|[0,\rho_t]}}, i)$$

Then, a compatibility relation between a global shared environment $\rho_S \in \mathcal{D}_\mathcal{S}$ and a global continuous environment $\rho_G \in \mathcal{D}_\mathcal{G}$. ρ_S and ρ_G are compatible if and only if the evolution of shared variables in ρ_G is the same as in the beginning of ρ_S. Formally, we have:

$$\rho_G = (f_c, f_s) \text{ and } \rho_S \text{ are compatible } \iff \forall s \in S,\ f_s \sqsubseteq_{SF} \rho_S$$

Finally, a fwd operator which computes the evolution of the continuous variables until the time at which the *sens* action is executed. Let $\rho_G \in \mathcal{D}_\mathcal{G}$ be a global continuous environment, ρ_S a compatible shared global environment and t the time until which we want to compute the evolution of the continuous part. Then, the extension of ρ_G under the constraint ρ_S is the continuous environment that behaves like ρ_G until ρ_{G_t} and then behaves like the solution of the differential equations with the constraint that the values of the shared variables are given by the step function ρ_S. Formally, we have:

$$fwd(\rho_G, \rho_S, t) = \rho' = \begin{pmatrix} f \\ g \end{pmatrix} \left| \begin{array}{l} \rho' \in \mathcal{D}_\mathcal{G},\ \rho'_t = t \\ \forall u \leq \rho_{G\,t}, \rho'(u) = \rho_G(u) \\ \forall u \in [0,t],\ \forall s \in S,\ g(u)(s) = \rho_S(s)(u) \\ f_{|[\rho_t,t]} = Sol(\Phi, \rho_S, f(\rho_t), \rho_t, t) \end{array} \right.$$

Next, let $\chi_G \in \mathcal{P}(\mathcal{D}_\mathcal{G})$ be a global environment. Then, we have:

$$fwd(\chi_G, \rho_S, t) = \bigcup_{\rho \in \chi_G\ ,\ \rho \text{ and } \rho_S \text{compatible}} fwd(\rho, \rho_S, t)$$

Finally, for a global shared environment χ_S, we have:

$$fwd(\chi_G, \chi_S, t) = \bigcup_{\rho_S \in \chi_S} fwd(\chi_G, \rho_S, t) \tag{15}$$

$$[\![(sens.c?x \to P^{l_1})^l]\!] = \left\{ \begin{array}{l} \dot{\chi}^l = \dot{\chi}^{l_1} \\ \chi^0_G = fwd(\chi^0_G, max\{\rho_t \mid \rho \in \chi^l\}) \\ \chi^{l_1} = \bigcup_{\substack{\rho \in \chi^l, \rho' \in \chi^0_G, \\ \rho \text{ and } \rho' \text{ compatible}}} \left\{ \rho \left[x \mapsto [\![f_c(\rho_t)(c)]\!]_{\mathbb{D}_\mathbb{X}} \right] \right\} \end{array} \right\} \cup [\![P^{l_1}]\!] \tag{E-9}$$

SKIP and STOP (E − 10 and E − 11)
Finally, $SKIP$ does nothing and $STOP$ blocks the process.

$$[\![SKIP^l]\!] = \left\{ \dot{\chi}^l = \chi^l \right\} \quad (E-10) \qquad [\![STOP^l]\!] = \left\{ \dot{\chi}^l = \emptyset \right\} \quad (E-11)$$

2.2.3. *Collecting and Non-Standard Semantics*
The semantics of a program is a solution to the equations of Section 2.2.2. Let P be a program and $[\![P]\!]$ a set of equations $\left\{E_{v_1}, \dot{E}_{v_1}, \cdots, E_{v_n}, \dot{E}_{v_n}, E_G, E_S\right\}$, where n is the number of control points in P. E_G is a set of equations for the global environment χ_G^0 and E_S is the equation for the global shared environment χ_S^0. Each equation E_i or $\dot{E}_i$ is written as $\chi^{l_i} = f_i(\chi^{l_1}, ., \chi^{l_n}, \dot{\chi}^{l_1}, ., \dot{\chi}^{l_n}, \chi_G^0, \chi_S^0)$ or $\dot{\chi}^{l_i} = \dot{f}_i(\chi^{l_1}, ., \chi^{l_n}, \dot{\chi}^{l_1}, ., \dot{\chi}^{l_n}, \chi_G^0, \chi_S^0)$, where f_i and $\dot{f}_i$ is the function representing the effect of a local or global operator. The solution to the semantic equations is then the least fix-point of the following function:

$$F : \begin{pmatrix} \chi^1, \\ \dot{\chi}^1, \\ \cdots, \\ \chi^n, \\ \dot{\chi}^n, \\ \chi^{1-0}, \\ \cdots, \\ \chi^{m-0}, \\ \chi_S^0, \\ \chi_G^0 \end{pmatrix} \rightarrow \begin{pmatrix} f_1(\chi^1, \ldots, \chi^n, \dot{\chi}^1, \ldots, \dot{\chi}^n) \\ \dot{f}_1(\chi^1, \ldots, \chi^n, \dot{\chi}^1, \ldots, \dot{\chi}^n) \\ \cdots, \\ f_n(\chi^1, \ldots, \chi^n, \dot{\chi}^1, \ldots, \dot{\chi}^n) \\ \dot{f}_n(\chi^1, \ldots, \chi^n, \dot{\chi}^1, \ldots, \dot{\chi}^n), \\ f_{1-0}(\chi^1, \ldots, \chi^n, \dot{\chi}^1, \ldots, \dot{\chi}^n), \\ \cdots, \\ f_{m-0}(\chi^1, \ldots, \chi^n, \dot{\chi}^1, \ldots, \dot{\chi}^n), \\ f_s(\chi^{1-0}, \ldots, \chi^{m-0}), \\ f_g(\chi^1, \ldots, \chi^n, \dot{\chi}^1, \ldots, \dot{\chi}^n, \chi_S^0) \end{pmatrix}$$

The domain of F is $F : \mathcal{D} \rightarrow \mathcal{D}$ with

$$\mathcal{D} = \mathcal{P}(\mathcal{D}_{\mathcal{L}})^{2n} \times {\mathcal{P}_{\mathcal{GL}}}^m \times \mathcal{P}(\mathcal{D}_{\mathcal{S}})_{/\equiv_{SF}} \times \mathcal{P}(\mathcal{D}_{\mathcal{G}})$$

where n is the number of local control point and m the number of parallel processes. $\mathcal{D}$ has a lattice structure using the corresponding order and operator for each component of the cartesian product. It is clear, according to the semantic equations, that each function f_i is monotone, and so is F. Then, according to Tarski's theorem, it has a least fix-point which may be computed using Kleen's sequence $(F^l(\perp))_{k \in I\!N}$.

DEFINITION 12. Concrete Collecting Semantics
The collecting semantics of an hybrid system is a function

$$f : \mathcal{L} \rightarrow \mathcal{P}(\mathcal{D}_{\mathcal{L}}) \cup \mathcal{P}_{\mathcal{GL}} \cup \mathcal{P}(\mathcal{D}_{\mathcal{S}}) \cup \mathcal{P}(\mathcal{D}_{\mathcal{G}})$$

such that $(f(l_1), f(\dot{l}_1), \cdots, f(l_n), f(\dot{l}_n), f((0)))$ *is the least fix-point of* F.

However, the safety properties we want to prove (see Section 1) are only interested in the traces left by the program, i.e. the sequences of actions it takes. So, we define the non-standard semantics as follows:

DEFINITION 13. Non-Standard Semantics
The non-standard semantics of an hybrid system SH is:

$$(\!|SH|\!) = trace\left(\bigcup_{l \in \mathcal{L}} f(l)\right)$$

$$[\![a]\!]_{\mathbb{E}} = f_a + e_a\overrightarrow{\varepsilon_e} \quad and \quad [\![b]\!]_{\mathbb{E}} = f_b + e_b\overrightarrow{\varepsilon_e}$$

$$\begin{array}{lcl}
[\![a+b]\!]_{\mathbb{E}} & = & \uparrow_{\sim}(f_a+f_b) + (e_a+e_b + \downarrow_{\sim}(f_a+f_b))\overrightarrow{\varepsilon_e} \\
[\![a-b]\!]_{\mathbb{E}} & = & \uparrow_{\sim}(f_a-f_b) + (e_a-e_b + \downarrow_{\sim}(f_a-f_b))\overrightarrow{\varepsilon_e} \\
[\![a\times b]\!]_{\mathbb{E}} & = & \uparrow_{\sim}(f_a\times f_b) + (e_af_b+e_bf_a+e_ae_b + \downarrow_{\sim}(f_a\times f_b))\overrightarrow{\varepsilon_e} \\
[\![\frac{1}{a}]\!]_{\mathbb{E}} & = & \uparrow_{\sim}\left(\frac{1}{f_a}\right) + \left[\ \sum_{i=1}^{+\infty}(-1)^i\frac{e^i}{f^{i+1}} + \downarrow_{\sim}\left(\frac{1}{f_a}\right)\ \right]\overrightarrow{\varepsilon_e} \\
[\![\frac{a}{b}]\!]_{\mathbb{E}} & = & [\![a\times\frac{1}{b}]\!]_{\mathbb{E}}
\end{array}$$

Figure 3. Arithmetic rules for the global error domain.

where f is the collecting semantics and trace the function defined in Equation 5.

At this point, we have not given any precise domain for discrete variables $x \in X$. We can use any numerical domain that supports the basic arithmetic operations as well as a cast from reals. For the purpose of the analysis of the numerical precision, we need to instantiate this non-standard semantics using pertinent domains. We therefore define:

- $(\!|SH|\!)_{I\!R}$ using $\mathbb{D}_{\mathbb{X}} = I\!R$ which is a theoretical instantiation used for comparison,
- $(\!|SH|\!)_{I\!F}$ using $\mathbb{D}_{\mathbb{X}} = I\!F$ the domain of floating point numbers, as defined by the IEEE-754 norm (ANSI and IEE, 1985), the realistic instantiation,
- $(\!|SH|\!)_{\mathbb{E}}$ using $\mathbb{D}_{\mathbb{X}} = \mathbb{E}$ the domain of global error, as defined in (Martel, 2005) and which arithmetics is given in Figure 3.

In order to define the safety properties, we need to introduce some operators on traces. Let $T \in \mathbb{T}$ be a trace, then:

- $act(T) = (a_0, s).\cdots.(a_n, s)$ such that $T = w_0.\cdots.w_n$ and $\forall i, w_i =< a_i, s, t_i >$ for some $s \in S$, computes the sequence of actions executed by this trace,
- $time(T, i) = t_i$ such that $T = w_0.\ldots.w_n$ and $w_i =< a_i, s, t_i >$, which returns the time at which the i^{th} action takes place in T,
- $T \leq T' \Longleftrightarrow act(T) = act(T') \wedge \forall i \in [0, |T|], time(T, i) \leq time(T', i)$. A trace T is smaller than T' if and only if it performs the same actions as T', in the same order, but at anterior times,
- $T \rhd t = w'_0.\cdots.w'_n$ if $T = w_0.\cdots.w_n$ and $w_i =< a_i, s, u_i > \Rightarrow w'_i =< a_i, s, u_i + t >$, which delays all actions in T by a time t.

Now, we formally define the safety property we intend to prove:

DEFINITION 14. Safety Property
An hybrid system SH satisfies the Δ-safety property if and only if

$$\forall T \in (\!|SH|\!)_{\mathbb{R}}, \exists T' \in (\!|SH|\!)_{\mathbb{F}} \mid (T \rhd (-\Delta)) \leq T' \leq (T \rhd \Delta)$$

3. Abstract Semantics

In this section, we propose a static analysis for hybrid systems based on the abstract interpretation framework (Cousot and Cousot, 1977; Cousot and Cousot, 1992). The general principle for abstract interpretation based analysis is to define an abstraction of the concrete semantics into an one in which the safety properties are computable, and to prove that this abstraction is safe. Our approach firstly consists of abstracting time by intervals and, next, to introduce new domains for step functions in order to keep precise information about variables despite the approximation on time. Actually, the non-determinism due to a $WAIT[u, v]$ creates an infinite, uncountable number of possible evolutions. Considering time as an interval $t^\sharp \in \mathbf{I}(\mathbb{R})$ rather than as a real $t \in \mathbb{R}$ solves this problem. However, this abstraction implies many changes, as the execution time of each instruction is no longer precisely known. The main drawback is that the precise time where the program changes the dynamics of the continuous system (via an *act* action) is undefined. This creates a set of possible curves from which it is difficult to find out the minimal and maximal in the general case, as shown by Figure 4. However, under some conditions, we are able to enclose all the possible dynamics precisely. Let the differential equations verify the following hypotheses:

1. variables should be separated, i.e. for all $c \in C$, $\dot{c} = f_c(c, S)$, where f_c only depends on c itself and S,
2. for all $c \in C$, the family of function $(x \mapsto f_c(x, s))_{s \in \mathbb{B}^{|S|}}$ is ordered.

The first condition states that two solutions of the differential equations with different initial values are parallel, i.e. their curves will never intersect. The second states that all the functions resulting from a change in the dynamics between time t and t' must start in the same direction (see Figure 5). Then, if the dynamics of the system changes between time t_0 and t_1, the resulting functions are enclosed by the dynamics computed when the change takes place at t_0 and at t_1. For the moment, we limit our analysis to such systems. We plan to investigate further on

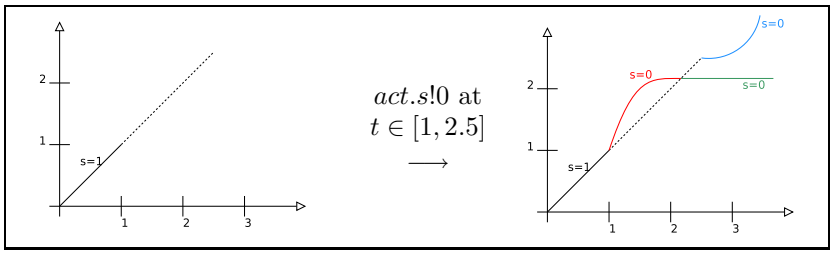

Figure 4. Problem caused by the uncertainty of time.

this issue and we believe that we will be able in the future to remove these restrictions. In the following, we introduce the abstract domain, then we build the concretisation map and the abstract semantics and we finally show that this abstraction is safe. As the abstract semantics is very similar to the concrete one, except that it uses abstract environments and abstract operators, we essentially focus on the abstract domains. Note that the abstract semantics we define still makes the difference between the different execution traces of the program. For instance, we distinguish the different runs of the body of a loop. This is necessary because we need to know precisely enough at what time the actions inside the loop are executed. So, the abstract semantics we propose is costly, but precise, and we will present at the end of this section a way of reducing the cost of the analysis using a widening operator.

3.1. Abstract Domains

We first abstract time into an interval of reals. This implies that we no longer know the exact time for an *act* action, thus the step function domain can no longer be used to describe the values of shared variables. We obtain a precise analysis by introducing another type of constraints of the form $[u, v]_k : \alpha$, which means that the function performs at most

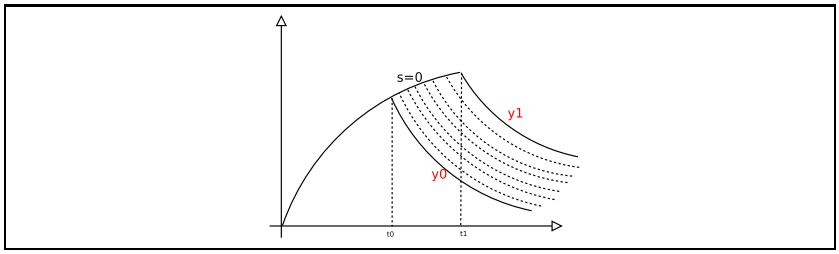

Figure 5. Possible enclosing when the conditions 1 and 2 are met. We assume that the program sets s to 1 between time t_0 and t_1.

k actions between time u and time v, but that its value remains in the interval α. For example, the constraint $[0,2]_2 : [0,1]$ represents the set of functions $\langle 0,k\rangle : \alpha \wedge \langle k,k'\rangle : \alpha' \wedge \langle k',2\rangle : \beta$, with $k,k' \in [0,2[$ and $\alpha,\alpha',\beta \in [0,1]$, i.e. the set of all functions whose value changes twice in the interval $[0,2]$, but always remain in $[0,1]$.

DEFINITION 15. Abstract Constraints
An abstract constraint can be either a type one *constraint,* $\langle u,v\rangle : \alpha$*, with* $u,v \in I\!R^+$ *and* $\alpha \in \mathbf{I}(\mathbb{B})$*, or a* type two *constraint,* $[u,v]_k : \alpha$*, with* $k \in I\!N$*.*

An abstract step function is now a conjunction of type one and type two constraints, with the restriction that between two type one constraints there is at least one type two constraint, because these should represent the time when the value of the function changes. Moreover, the last constraint must be a type one constraint to ensure that these functions are constant when time goes to infinity.

DEFINITION 16. Abstract Step Functions
Let $Env^\sharp_{SF}(\mathcal{D})$ *be the domain of abstract step functions with values in* $\mathcal{D}$*. Then,* $f \in Env^\sharp_{SF}(\mathcal{D})$ *if and only if* f *may be written as the conjunction of type one or type two constraints,* $f = c_1 \wedge \ldots \wedge c_n$*, such that*

1. c_n *is a type 1 constraint,*
2. *for all* $i \geq 2$*, if* c_i *is a type 1 constraint,* c_{i-1} *is type 2.*

We may now define the abstract domains, which strongly depend on this notion of abstract step functions.

3.1.1. *Local Abstract Domain*
A concrete local environment associates each discrete variable with a value of $\mathbb{D}_\mathbb{X}$, each shared variable with a concrete step function and time with a real number. Analogously, an abstract local environment associates each discrete variable with an interval of $\mathbf{I}(\mathbb{D}_\mathbb{X})$, each shared variable with an abstract step function with value in $\mathbf{I}(\mathbb{B})$ and time with an interval of real numbers.

DEFINITION 17. Local Abstract Domain
Let $\mathcal{D}_\mathcal{L}{}^\sharp$ *be the set of local environments of the form* $\rho^\sharp = \left(\rho^\sharp_X, \rho^\sharp_S, \rho^\sharp_t\right)$*, with* $\rho^\sharp_X \in X \to \mathbf{I}(\mathbb{D}_\mathbb{X})$*,* $\rho^\sharp_S \in S \to Env^\sharp_{SF}(\mathbf{I}(\mathbb{B}))$ *and* $\rho^\sharp_t \in \mathbf{I}(I\!R^+)$*. Then,* $\rho^\sharp \in \mathcal{D}_\mathcal{L}{}^\sharp$ *if and only if* $\forall s \in S$*,* $\rho^\sharp_S(s) = c_0 \wedge \ldots \wedge c_n$*,* c_n *is a type one constraint* $c_n = \langle t_n, +\infty\rangle : \alpha_n$ *and the following conditions are satisfied:*
1. $\forall i \in [1,n-1],\ \alpha_i \cap \alpha_{i+1} \neq \emptyset$
2. $\exists j$ *s.t.* $\forall i \in [n-j,n-1], c_i$ *is type* 2*,* $c_i = [t_i,t_{i+1}]_{k_i} : \alpha_i$

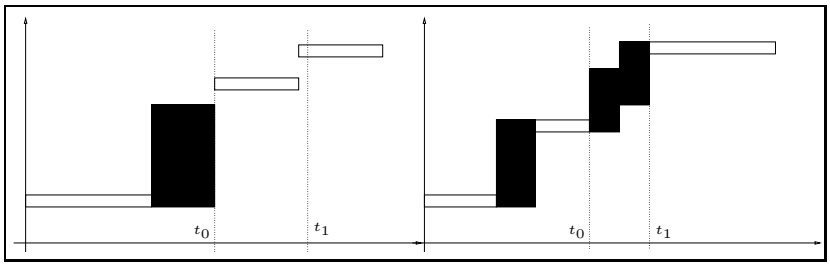

Figure 6. Illegal abstract environments.

3. $\forall i \in [n-j, n-1],\ \underline{\alpha_n} \subseteq \alpha_i$

4. $t_j \leq \underline{\rho_t^\sharp}$ *and* $t_n \leq \overline{\rho_t^\sharp}$

Condition 1 states that two consecutive steps must overlap, as the result of an action (i.e. a type one constraint) must intersect with the action itself (i.e. a type two constraint). Condition 2 states that there must be at least one type two constraint before the last one, because this is a type one constraint. So, these two conditions prevent a situation like the one on Figure 6 (left) to happen. Condition 3 and 4 ensure that actions cannot have been performed at a time in the future. They express that a situation like the one in Figure 6 (right) cannot happen, i.e. if the present time is between t_0 and t_1, then no action can start at a time greater than t_0.

3.1.2. *Global Abstract Domains*

First, we define the abstract order on $Env_{SF}^\sharp(.)$ (Equations 16 to 19), that translates $\sqsubseteq_{SF}$ to abstract step functions. Intuitively, a function $f^\sharp$ is smaller than $g^\sharp$ if and only if each action of $f^\sharp$ may be divided into a series of action of $g^\sharp$. In other words, the sequence of times where $f^\sharp$ makes an action is a sub-sequence of that of $g^\sharp$.

DEFINITION 18. The abstract Order $\sqsubseteq_{SF}^\sharp$

$$\langle t_1, t_2\rangle : \alpha \sqsubseteq_{SF}^\sharp c_1 \wedge \ldots \wedge c_n \iff n = 1,\ c_1 = \langle t_1, t_2\rangle : \alpha',\ \alpha \subseteq \alpha' \quad if\ t_2 < +\infty \tag{16}$$

$$\langle t_1, +\infty\rangle : \alpha \sqsubseteq_{SF}^\sharp c_1 \wedge \ldots \wedge c_n \iff \begin{cases} c_1 = \langle t_1, t\rangle : \alpha',\ \alpha \subseteq \alpha' \\ \quad or \\ c_1 = [t_1, t]_k : \alpha',\ \alpha \subseteq \alpha' \end{cases} \tag{17}$$

$$[t_1, t_2]_k : \alpha \sqsubseteq_{SF}^\sharp c_1 \wedge \ldots \wedge c_n \iff \begin{cases} \forall i \in [1, n],\ c_i = [t'_i, t'_{i+1}]_{k'_i} : \alpha'_i \\ t_1 = t'_1,\ t_2 = t'_{n+1},\ \alpha \subseteq \alpha'_i \end{cases} \tag{18}$$

$$\begin{aligned} c_1 \wedge \ldots \wedge c_n \sqsubseteq_{SF}^\sharp c'_1 \wedge \ldots \wedge c'_m \iff\ & \exists\ 1 = k_0 \leq k_1 \leq \ldots \leq k_n \leq k_{n+1} = m \text{ such that} \\ & \forall i \in [1, n],\ c_i \sqsubseteq_{SF}^\sharp c'_{k_{i-1}+1} \wedge \ldots \wedge c'_{k_i} \end{aligned} \tag{19}$$

As in the concrete case, this order is extended into a pre-order $\sqsubseteq^\sharp$ on sets of functions, which then defines an equivalence relation $\equiv_{SF}^\sharp$.

This equivalence relation lets us define the abstract domain for intermediary environments: $\mathcal{P}_{\mathcal{GL}^\sharp} = \left(\mathcal{P}\left(\mathcal{D}_{\mathcal{GL}^\sharp}\right)\right)_{/\equiv^\sharp_{SF}}$ with $\mathcal{D}_{\mathcal{GL}^\sharp} = (S \rightarrow Env^\sharp_{SF}(\mathbf{I}(\mathbb{B})))$. We use the order $\sqsubseteq^\sharp$ on this domain and the following meet and join operators:

$$[A] \sqcup^\sharp [B] = [A \cup B] \text{ and } [A] \sqcap^\sharp [B] = [A \cap B]$$

We may now define global abstract domains. For the shared variables, we still have to add to each step of the function an information about which process has performed the corresponding action. However, this information is no longer a vector of booleans but a vector of boolean sets, such that $1 \in b_i$ if and only if the step may have been created by the ith process.

DEFINITION 19. Abstract Global shared Domain
The abstract domain for χ^0_S *is* $\left(\mathcal{P}(\mathcal{D}_{\mathcal{S}^\sharp})\right)_{/\equiv^\sharp_{SF}}$, *with:*

$$\mathcal{D}_{\mathcal{S}^\sharp} = S \rightarrow Env^\sharp_{SF}(\mathbf{I}(\mathbb{B}) \times \mathcal{P}(\mathbb{B})^n)) \tag{20}$$

In Equation 20, n denotes the number of processes of the parallel system.

For continuous variables, we abstract a set of continuous real functions f by two floating-point step functions that encapsulate f. This is done via a safe approximation algorithm described in Section 4.

DEFINITION 20. Abstract Global Continuous Domain
Let $\mathcal{P}(\mathcal{D}_{\mathcal{G}^\sharp})$ *be the abstract domain for* χ^0_G, *with:*

$$\mathcal{D}_{\mathcal{G}^\sharp} \subseteq Env_{SF}(\mathbb{F}^{|C|}) \times Env_{SF}(\mathbb{F}^{|C|}) \times \mathcal{D}_{\mathcal{S}^\sharp}$$

An element $\rho^\sharp_G \in \mathcal{D}_{\mathcal{G}^\sharp}$ *is denoted* $\rho^\sharp_G = \left(f^\sharp_-, f^\sharp_+, f^\sharp_s\right)$ *and* $\rho^\sharp_G \in \mathcal{D}_{\mathcal{G}^\sharp}$ *if and only if* $f^\sharp_-$ *and* $f^\sharp_+$ *are defined on the same set of steps.*

3.1.3. *Abstract Traces*

The abstract traces $\mathbb{T}^\sharp$ are similar to concrete ones, except that the components are triple $< t, s, \alpha >$, with $t \in \mathbf{I}(\mathbb{R}^+)$, $s \in S$ and $\alpha \in \mathbf{I}(\mathbb{B})$. To compute these abstract traces, the type two constraints have to be located in the abstract step functions, as they represent the moments where the value of the function changes. So, for a step function $f = c_0 \wedge \ldots \wedge c_n \in Env^\sharp_{SF}(\mathbf{I}(\mathbb{B}))$, representing the evolution of the shared variable $s \in S$, the associated trace is:

$$tr(f, s) = \langle [t_{k_1}, t_{k_1+1}], s, \alpha_{k_1} \rangle . \cdots . \langle [t_{k_j}, t_{k_j+1}], s, \alpha_{k_j} \rangle \tag{$tr-1$}$$

where $\{k_1, \ldots, k_j\} = \left\{ p \in [1,n] \;\middle|\; \begin{array}{l} c_p \; is \; type \; 2 \\ c_p = [t_p, t_{p+1}]_{k_p} : \alpha_p \end{array} \right\}$. This function is then extended to local environments in the concrete case (Equation 5). For the global environment, the trace of the ith process is computed in the same way, except that only type two constraints that may come from this process must be handled. So, the trace function for an abstract step function $g = c_0 \wedge \ldots \wedge c_n \in Env^\sharp_{SF}(\mathbf{I}(\mathbb{B}) \times \mathcal{P}(\mathbb{B})^n)$ is:

$$tr(g, s, i) = \langle [t_{k_1}, t_{k_1+1}], s, \alpha_{k_1} \rangle . \cdots . \langle \left[t_{k_j}, t_{k_j+1} \right], s, \alpha_{k_j} \rangle \qquad (tr-2)$$

where $\{k_1, ., k_j\} = \left\{ p \in [1,n] \;\middle|\; \begin{array}{l} c_p \; is \; type \; 2 \\ c_p = [t_p, t_{p+1}]_{k_p} : (\alpha_p, b_p) \\ 1 \in b_p(i) \end{array} \right\}$. This function is readily extended to global environments. These two functions enable us to define the compatibility criterion in the abstract: a local environment $\rho^\sharp = \left(\rho^\sharp_X, \rho^\sharp_S, \rho^\sharp_t \right) \in \mathcal{D}_{\mathcal{L}^\sharp}$ is compatible with a global one $\rho^\sharp_G = \left(f^\sharp_-, f^\sharp_+, f^\sharp_s \right) \in \mathcal{D}_{\mathcal{G}^\sharp}$ if and only if, when $\rho^\sharp_S$ performs an action between time t and t', so does the global environment between time $u \leq t$ and $u' \geq t'$. Formally, $\rho^\sharp$ and $\rho^\sharp_G$ are compatible if and only if:

$$\forall T \in tr(\rho^\sharp), \exists T' \in tr(\rho^\sharp_{G\,|[0,\overline{\rho^\sharp_t}]}, i) \left| \begin{array}{l} \rho^\sharp = (\rho^\sharp_X, \rho^\sharp_S, \rho^\sharp_t), \;\; \rho^\sharp_t = [\underline{\rho^\sharp_t}, \overline{\rho^\sharp_t}] \\ T = w_1 . \cdots . w_n \\ T' = w'_1 . \cdots . w'_n \\ \forall i \in [1,n], \left\{ \begin{array}{l} w_i =< t_i, s, \alpha_i > \\ w'_i =< t'_i, s, \alpha'_i > \\ \alpha_i \subseteq \alpha'_i \\ t_i \subseteq t'_i \end{array} \right. \end{array} \right.$$

We now have an abstract version of all concrete local and global domains. Before defining the abstract operators and the abstract semantics, we define the concretisation map which establishes the relation between an abstract and a concrete environment.

3.2. Concretisation Maps

As there are four different domains, we must define four concretisation functions: γ_L for local environments, γ_{GL} for intermediary ones and γ_S, γ_C for global environments. We first define the concretisation function

$$\gamma_{SF} : \left(Env^\sharp_{SF}(\mathbb{B}) \right) \times \mathbf{I}\left(I\!R^+ \right) \rightarrow \mathcal{P}(Env_{SF}(\mathbb{B}))$$

for abstract step functions, performing actions until time δ:

$$\gamma_{SF}(\langle t, t' \rangle : i, \delta) \;=\; \left\{ \begin{array}{l} \{\langle \delta, +\infty \rangle : \alpha \mid \alpha \in i\} \;\; if \;\; \delta < t \\ \{\langle t, +\infty \rangle : \alpha \mid \alpha \in i\} \;\; if \;\; t \leq \delta < t' \\ \{\langle t, t' \rangle : \alpha \mid \alpha \in i\} \;\; if \;\; t' \leq \delta \end{array} \right. \qquad (21)$$

$$\gamma_{SF}([t,t']_k : i, \delta) = \left\{ \begin{array}{l} \emptyset \; if \; \delta < t \\ \left\{ \begin{array}{l} \langle t, t_1 \rangle : x_1 \wedge .. \wedge \langle t_{k-1}, \beta \rangle : x_k \; such \; that \\ \quad \left| \begin{array}{l} t < t_1 < .. < t_{k-1} < \beta \\ x_1, .., x_k \in i \end{array} \right. \end{array} \right\} \\ \\ with \; \beta = min(\delta, t') \; if \; \delta \geq t \end{array} \right. \quad (22)$$

$$\gamma_{SF}(c \wedge f, \delta) = \left\{ c^\natural \wedge f^\natural | c^\natural \in \gamma_{SF}(c, \delta) \; and \; f^\natural \in \gamma_{SF}(f, \delta) \right\} \quad (23)$$

$\gamma_{SF}(f, \delta)$ transforms the abstract step function f into a set P of concrete functions, defined until time δ, such that if f has a constraint $\langle u, v \rangle : \alpha$, each function $g \in P$ is constant on the interval $[u, v[$ with a value $x \in \alpha$. If f has a constraint $[u, v]_k : \alpha$, each function $g \in P$ performs at most k actions on the interval $[u, v[$.

We first define the concretisation for *one* local environment, it is then pointwise extended to sets. Let $\rho^\sharp = (\rho^\sharp_X, \rho^\sharp_S, \rho^\sharp_t) \in \mathcal{D}_{\mathcal{L}}{}^\sharp$. The concretisation of $\rho^\sharp$ is done as follows: a concretisation time $\delta \in \rho^\sharp_t$ is first chosen, then the abstract step function is concretised until time δ via γ_{SF}. Then, we have:

$$\gamma_L(\rho^\sharp) = \left\{ (\rho_X, \rho_S, \delta) \; \left| \begin{array}{l} \delta \in \rho^\sharp_t \\ \rho_X \in \rho^\sharp_X \\ \forall s \in S, \; \rho_S(s) \in \gamma(\rho_S(s)^\sharp, \delta) \end{array} \right. \right\} \quad (24)$$

The concretisation map for local environments is then:

$$\gamma_L(\chi) = \bigcup_{\rho^\sharp \in \chi} \{\gamma_L(\rho^\sharp)\}$$

Let $\rho^\sharp_{GL} \in (S \rightarrow Env^\sharp_{SF}(\mathbb{B}))$ be a intermediary environment. It is concretised using γ_{SF}, with a concretisation time set to infinity:

$$\gamma_{GL}(\rho^\sharp_{GL}) = \left\{ \rho_S \mid \forall s \in S, \; \rho_S(s) \in \gamma_{SF}(\rho_{GL}(s)^\sharp, +\infty) \right\}$$

Then, for $\left[\chi^\sharp_{GL}\right] \in \mathcal{D}_{\mathcal{GL}}{}^\sharp$,

$$\gamma_{GL}\left([\chi^\sharp_{GL}]\right) = \left[\bigcup_{\rho \in \chi_{GL}} \{\gamma_{GL}(\rho)\} \right]$$

It is clear that $\rho^\sharp \sqsubseteq^\sharp_{SF} \rho'^\sharp \Rightarrow \gamma_X(\rho^\sharp) \sqsubseteq \gamma_X(\rho'^\sharp)$, such that the function γ_{GL} is monotone.

The concretisation of global shared environments works exactly as for intermediary environments, except that we use a slightly different function, γ'_{SF}, for concretising the step functions, as their value now are in $\mathbf{I}(\mathbb{B}) \times \mathcal{P}(\mathbb{B})^n$. The function γ_{SF} changes a constraint $\langle u, v \rangle : \alpha$ into a constraint $\langle u, v \rangle : x$ with $x \in \alpha$. The new function γ'_{SF} changes a constraint $\langle u, v \rangle : (\alpha, B)$ into a constraint $\langle u, v \rangle : (x, b)$ such that $x \in \alpha$

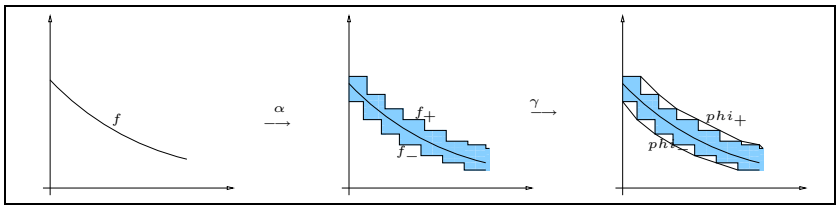

Figure 7. Over-Approximation of Step Functions.

and $\forall i, b(i) \in B(i)$. Then, the concretisation map for the global shared environment $[\chi_S^0] \in \left(\mathcal{P}(\mathcal{DS}^\sharp)\right)_{/\equiv^\sharp_{SF}}$ is:

$$\gamma_S([\chi_S^0]) = \left[\bigcup_{\rho_S^\sharp \in \chi_S^0} \{\rho_S \mid \forall s \in S, \rho_S(s) \in \gamma'_{SF}(\rho_S^\sharp(s))\} \right]$$

For the global continuous environment, the situation is slightly more complicated. Let $\rho_G^\sharp = (f_-^\sharp, f_+^\sharp, f_s^\sharp)$ be an abstract continuous environment. It is concretised as the set of all concrete environments (f_c, f_s) such that $\forall t, f_-^\sharp(t) \leq f_c(t) \leq f_+^\sharp(t)$ and $f_s \in \gamma'_S(f_s^\sharp)$, where f_c is piecewise $\mathcal{C}^2$. However, the upper limit of an infinite set of piecewise $\mathcal{C}^2$ function is not itself piecewise $\mathcal{C}^2$, so an over-approximation has to be performed and we may choose for example all the functions enclosed by the convex hull (ϕ_-, ϕ_+) of the steps of f_- and f_+ (see Figure 7). Then, the concretisation map for global continuous environments is:

$$\gamma_C(\rho_G^\sharp) = \left\{ (f_c, f_s) \;\middle|\; \begin{array}{l} f_c \in \mathcal{C}^2\left([0,t] \to I\!R^{|C|}\right), \\ \quad \forall u \in [0,t] \phi_-(u) \leq f_c(u) \leq \phi_+(u) \\ f_s \in \gamma_S\left(f_s^\sharp\right) \end{array} \right\}$$

3.3. Abstract Semantics

We now define the abstract semantics of an hybrid system, and next show that this abstract semantics is safe using the concretisation maps we defined in Section 3.2. The semantic equations are exactly the same as for the concrete case, except that they use the abstract operators defined below. Two equations are nevertheless different: for $WAIT$ and *sens* actions. Therefore, we do not present every abstract equations (only those two, see Equation 3.3.1 and Equation 3.3.1), but rather explain how abstract operators work.

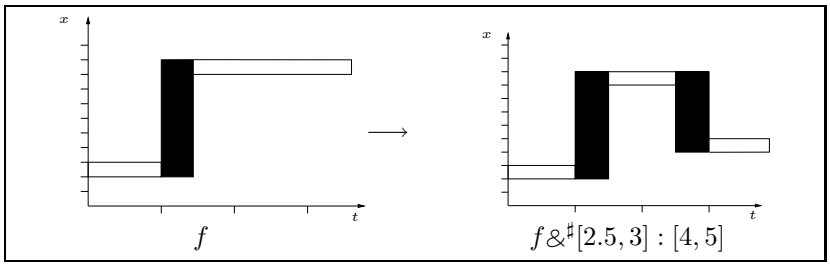

Figure 8. Effect of $\&^\sharp$.

3.3.1. *Abstract Operators*
For every concrete operator $(\&, [\![.]\!]_{\mathbb{D}_\mathbb{X}}, [\![.]\!]_b, |||)$ we must define an abstract equivalent. We present the intuition of all the abstract operators, but because of lack of place, we cannot give their entire, formal definition. This may be found in (Bouissou, 2005), together with the detailed proof of the correctness of the abstract semantics. For imperative instructions, we use an usual abstract semantics using the interval domain for numerical values (Moore, 1979). The concrete operators $[\![.]\!]_{\mathbb{D}_\mathbb{X}}$ and $[\![.]\!]_b$ are thus translated in the abstract domains.

Actions
We must define the $\&^\sharp$ operator, which adds a new step to an abstract step function. However, as the type two constraints are assumed to mark the times where the value of the function changes, adding the new step means first adding a new type two constraint and then the type one constraint until infinity. The action of this operator may be graphically represented by Figure 8 (type one constraints are printed in white, type two in black).

Delays
The equation for the $WAIT[u, v]$ instruction changes: $[u, v]$ is added to time of the local environment using interval arithmetic.

$$[\![(WAIT[u,v])^l]\!]^\sharp = \left\{ \dot{\chi}^l = \bigcup_{\rho \in \chi^l} \{\rho\big[t \mapsto \rho_t + [u,v]\big]\} \right\} \qquad (E-3-b)$$

Parallel composition
For parallel composition, we must combine abstract step functions coming from different processes. The problem mainly is to find conflicts, i.e. actions that may be done by the two processes at the same time. In the concrete case, this is easy as the execution time for all actions is known. Let f and f' be two abstract step functions that must be combined. A

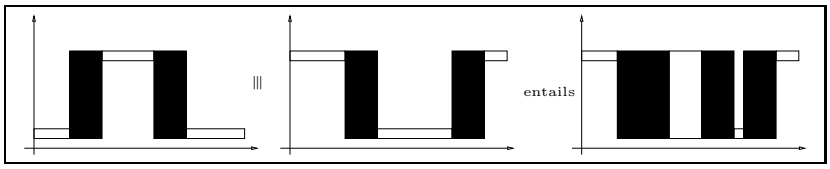

Figure 9. Abstract interleaving.

conflict may occur if the two functions have a constraint of the form $[u,v]_k : \alpha$ and $[u',v']_{k'} : \alpha'$ with $[u,v] \cap [u',v'] \neq \emptyset$. In this case, the combined function has a constraint $\langle([u,v] \cup [u',v'])_{k+k'} : \alpha \cup \alpha' \rangle$. An example of the application of $|||^\sharp$ is shown on Figure 9.

Sens actions
Finally, we need to define $fwd^\sharp$, the abstraction of the fwd operator in the abstract domain. This operator must compute the evolution of an abstract continuous environment $\rho^\sharp_G$ under the constraints given by an abstract shared environment $\rho^\sharp_S$. $\rho^\sharp_G$ and $\rho^\sharp_S$ must be compatible, i.e. $\rho^\sharp_S$ has precisely defined the evolution of $\rho^\sharp_G$ until the current time. To compute the forward evolution of $\rho^\sharp_G$, we assume the existence of a function α which, given a system of differential equations E, initial and final times t and t' as well as two initial values y_+ and y_-, computes two floating-point step functions f_- and f_+, defined on $[t,t']$, such that, if f is the real solution of E with initial value $y \in [y_-, y_+]$ at t, then $\forall u \in [t,t'], f_-(u) \leq f(u) \leq f_+(u)$. This algorithm is described in Section 4. The $fwd^\sharp$ operator then proceeds as follows: every time a constraint $[u,v]_k : I$ is found in $\rho^\sharp_S$, the α-algorithm is used with all $b \in I$, which gives us at most two solutions. The highest step function is then taken as the up-limit, the lowest as the down-limit. So, every time there is a doubt on the value of a shared variable $s \in S$, the two possibilities are distinguished, and the solutions compared. The conditions imposed on the dynamical systems are sufficient to ensure that this approximation is safe. An example of the use of $fwd^\sharp$ is shown on Figure 10. We may now define the semantics for *sens* actions:

$$[\![(sens.c?x \to P^{l_1})^l]\!]^\sharp = \left\{ \begin{array}{l} \dot{\chi}^l = \dot{\chi}^{l_1}; \\ \chi^0_G = fwd^\sharp(\chi^0_G, max\{\overline{\rho_t} \mid \rho \in \chi^l\}) \\ \chi^{l_1} = \bigcup_{\substack{\rho \in \chi^l, \rho' \in \chi^0_G, \\ \rho, \rho' \ compatible}} \left\{\rho[x \mapsto [\![c]\!]^\sharp \rho]\right\} \end{array} \right\} \cup [\![P^{l_1}]\!]^\sharp \qquad (E-9-b)$$

where $[\![c]\!]^\sharp \rho = [min\{f_c(t)(c) \mid t \in \rho_t\}, max\{f_c(t)(c) \mid t \in \rho_t\}]$.

The abstract semantics is then a function $f^\sharp$ mapping each control point to a local or global environment, such that $\{f(l) \mid l \in \mathcal{L}\}$ is the least fix-point of $F^\sharp$. The non-standard abstract semantics is then defined as the trace of this function:

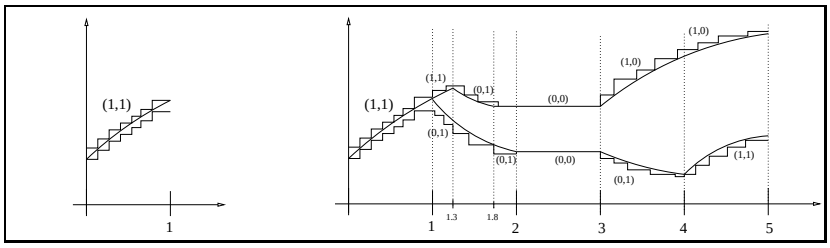

Figure 10. The effect of $fwd^\sharp$.

DEFINITION 21. Non-Standard Abstract Semantics
The non-standard abstract semantics of an hybrid system is:

$$(\!|SH|\!)^\sharp = tr\left(\bigcup_{l\in\mathcal{L}} f^\sharp(l)\right)$$

As for the concrete case, we define various semantics using various numerical domains for discrete variables: $(\!|.|\!)^\sharp_{I\!R}$, $(\!|.|\!)^\sharp_{I\!F}$ *and* $(\!|.|\!)^\sharp_{\mathbb{E}}$.

The calculation of the fix-point of $F^\sharp$ is however still problematic. Actually, the presence of time in a local environment forces us to completely unroll all the loops before reaching a fix-point. Since, from one execution of the loop body to another, the time has moved forward, the continuous variables are defined on a larger interval such that no fix-point can be found. A naive widening would be to set time to infinity, and, consequently, we would lose all information about also the continuous variables. This would not only lead to a very imprecise analysis, but the evolution of continuous variables, as time goes to infinity, would often be too hard to compute. Therefore, this approach cannot be used in practice. A solution consists of computing in advance the evolution of continuous variables, until a finite time T. This leads us to the following strong assumption: every controlled variable (i.e. a variable whose evolution may be modified by an actuator) is bound, and these variables would diverge without the program controlling them. We assume that the program aims at controlling its environment, it uses actuators in order to bond the continuous variables, such that the sequence of actions of a program is ultimately periodic. This assumption is nevertheless true in the practical systems we have studied. Consider for example a thermostat: it controls the temperature of a room via a heater connected to an interrupter, such that the temperature stays between 18 and 21 degrees. The interrupter has two positions: *on* and the temperature of the room increases, or *off* and the temperature of the room decreases. If the thermostat was correctly designed, it will, after a while, oscillate between the two positions, switching off the

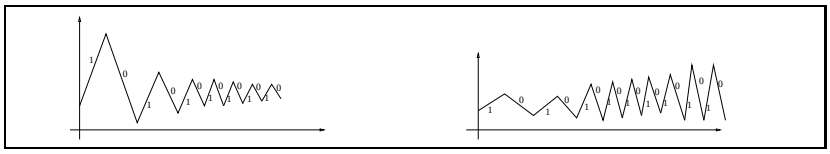

Figure 11. Examples of the effect of the widening operator

interrupter when temperature reaches 21 degrees, and on when it falls down to 18.

With this assumption, we can accelerate the detection of the fix-point. First, we must unroll the loop until we find a period for the actions, and then compute the evolution of continuous variables until a large time T. In this purpose, we use as values for shared variables the repetition of the period. Then, we find two cases. Either the different branches of the function we compute are included in the beginning of the function (Figure 3.3.1 left) and then we may find a fix-point for discrete variables as their inputs are bound. Or the continuous function gets wider and wider (Figure 3.3.1 right), and we must proceed to another widening: we increase the width of each branch and repeat this pattern.

3.3.2. *Abstract Safety Properties*

We are now ready to formulate the abstract safety property. This is equivalent to concrete ones, except that the time at which actions are taken are intervals instead of reals. An hybrid system SH is said to verify the $\Delta^\sharp$-property if and only if

$$\forall T \in (\!|SH|\!)^\sharp_{\mathbb{R}}, \exists T' \in (\!|SH|\!)^\sharp_{\mathbb{F}} \;\left|\; \begin{array}{l} act(T) = act(T') \\ \forall i,\ time(T',i) - time(T,i) \subseteq [-\Delta, \Delta] \end{array}\right.$$

3.4. Safety Proof

In this section, we prove that *act* actions are safe. The rest of the semantics is proved safe in (Bouissou, 2005). Let us call $f_{s,b}$ the function representing the effect of the action $act.s!b$ on a concrete local environment and $f^\sharp_{s,b}$ the corresponding abstract function. Then, we have: $f_{s,b}(\chi) = \bigcup_{\rho\in\chi}\{\rho[s \mapsto \rho(s)\&([\![b]\!]_{\mathbb{B}}\rho, \rho_t)]\}$ and $f^\sharp_{s,b}(\chi^\sharp) = \bigcup_{\rho^\sharp\in\chi^\sharp}\{\rho^\sharp[s \mapsto \rho^\sharp(s)\&^\sharp(\rho^\sharp_t : [\![b]\!]_{\mathbb{B}}\rho^\sharp)]\}$. We must prove that, for all $\chi^\sharp \in \mathcal{P}(\mathcal{D}_{\mathcal{L}}{}^\sharp)$, we have:

$$f_{s,b}(\gamma(\chi^\sharp)) \subseteq \gamma(f^\sharp_{s,b}(\chi^\sharp)) \tag{25}$$

It is clear that it is sufficient to prove Equation 25 when $\chi^\sharp = \{\rho^\sharp\}$. Moreover, as $f_{s,b}$ has no effect on discrete variables, we can write $\rho^\sharp =$

$(\rho_S^\sharp, \rho_t^\sharp)$, with $\rho_S^\sharp(s) = c_0^\sharp \wedge \ldots \wedge c_n^\sharp$ and $\rho_t^\sharp = [\underline{t}, \overline{t}]$. Let $c_n^\sharp = \langle t_n, +\infty\rangle : I_n$. We know (Definition 17) that there exists j such that $\forall i \in [n-j, n-1]$, $c_i^\sharp = [t_i, t_{i+1}]_{k_i} : I_i$, $I_n \subseteq I_i$ and $\underline{t} \geq t_{n-j}$. We suppose here that $j = 1$, the other cases are similar. There are then two more cases:

If $t_n \leq \underline{t}$

In this case, we can compute $x^\sharp = \rho_S^\sharp(s) \&^\sharp \rho_t^\sharp : [\![b]\!]_b^\sharp \rho^\sharp$:

$$x^\sharp = c_0 \wedge \ldots \wedge c_{n-1} \wedge \langle t_n, \underline{t}\rangle : I_n \wedge [\underline{t}, \overline{t}]_1 : I_n \cup [\![b]\!]_b^\sharp \wedge \langle \overline{t}, +\infty\rangle : [\![b]\!]_b^\sharp$$

and the concretisation $\chi' = \gamma(f_{s,b}(\chi^\sharp)$ is (Equation 24):

$$\chi' = \bigcup_{\delta \in [\underline{t}, \overline{t}]} \left\{ \begin{array}{c} \big(c'_0 \wedge .. \wedge c'_{n-1} \wedge \langle t_n, \underline{t}\rangle : \alpha \wedge \langle \underline{t}, t\rangle : \beta \wedge \langle t, \delta\rangle : \beta' \wedge \langle \delta, +\infty\rangle : \alpha' \ , \ \delta\big) \\ \textit{such that } c_i \in \gamma_X(c_i, \delta), \alpha \in I_n, t \in [\underline{t}, \overline{t}], \beta, \beta' \in I_n \cup [\![b]\!]_b^\sharp, \alpha' \in [\![b]\!]_b^\sharp \end{array} \right\} \tag{26}$$

In addition, we can compute $\chi = \gamma(\chi^\sharp) = \bigcup_{\delta \in [\underline{t}, \overline{t}]} \big\{ \big(c'_0 \wedge \cdots \wedge c'_n, \delta\big) : c'_i \in \gamma_X(c_i, \delta) \big\}$.

As $t_n \leq \underline{t}$, $\forall \delta \in [\underline{t}, \overline{t}]$, $\gamma_X(c_n, \delta) = \{\langle t_n, +\infty\rangle : \alpha | \alpha \in I_n\}$. Let now $\rho = \big(c_0 \wedge \ldots \wedge c_n, t\big) \in \chi$. In particular, $c_n = \langle t_n, +\infty\rangle : v$, with $v \in I_n$. We need to show that $f_{s,b}(\rho) \in \chi'$. We have $f_{s,b}(\rho) = (c_0 \wedge .. \wedge c_{n-1} \wedge \langle t_n, \delta\rangle : v$ $\wedge \langle \delta, +\infty\rangle : [\![b]\!]_b \ , \ \delta)$. As $[\![b]\!]_b \in [\![b]\!]_b^\sharp$, if we take in Equation 26 $\alpha = \beta = \beta = v$ and $\alpha' = [\![b]\!]_b$ and $t = \delta$, we get that $f_{s,b}(\rho) \in \chi'$.

If $t_{n-1} \leq \underline{t} < t_n \leq \overline{t}$

In this case, we have:

$$\begin{array}{rcl} x^\sharp = c_0 \wedge \ldots \wedge c_{n-2} & \wedge & [t_{n-1}, \underline{t}]_{k_{n-1}} : I_{n-1} \wedge [\underline{t}, t_n]_{k_{n-1}+1} : I_{n-1} \cup [\![b]\!]_b^\sharp \\ & \wedge & [t_n, \overline{t}]_1 : I_n \cup [\![b]\!]_b^\sharp \wedge \langle \overline{t}, +\infty\rangle : [\![b]\!]_b^\sharp \end{array}$$

We can now compute the concretisation of $\rho^\sharp$ and $x^\sharp$. We need to distinguish the cases where the concretisation time δ is chosen in $[\underline{t}, t_n]$ and in $[t_n, \overline{t}]$. In the first case, we have

$$\gamma(x^\sharp) = \bigcup_{\delta \in [t_1, t_n[} \left\{ \begin{array}{r} \Big(c'_0 \wedge \ldots \wedge c'_{n-2} \wedge \langle t_{n-1}, t_1^1\rangle : \alpha^1 \wedge \ldots \wedge \langle t_1^{k_{n-1}-1}, t_1\rangle : \alpha^{k_{n-1}} \\ \wedge \langle t_1, t_2^1\rangle : \beta^1 \wedge \ldots \wedge \langle t_2^{k_{n-1}}, \delta\rangle : \beta^{k_{n-1}+1} \\ \wedge \langle \delta, +\infty\rangle : \alpha \ , \ \delta\Big) \end{array} \right\}$$

and

$$\chi = \gamma(\chi^\sharp) = \bigcup_{\delta \in [t_1, t_n[} \left\{ \begin{array}{r} \Big(c'_0 \wedge \ldots \wedge c'_{n-2} \wedge \langle t_{n-1}, t_1^1\rangle : \alpha^1 \wedge \ldots \wedge \langle t_1^{k_{n-1}-1}, \delta\rangle : \alpha^{k_{n-1}} \\ \wedge \langle \delta, +\infty\rangle : \alpha \ , \ \delta\Big) \end{array} \right\}$$

If we choose carefully the coefficient and the times in these formulae, we see immediately that $\forall \rho \in \chi, f_{s,b}(\rho) \in \gamma(x^\sharp)$. The case $\delta \in [t_n, \overline{t}]$ is similar.

4. Approximation Algorithm

In this section, we briefly present our approximation algorithm for ordinary differential equations (ODE). A fully detailed description of this algorithm is given in (Bouissou, 2005). This algorithm is based on an order 5 Runge-Kutta iteration method, developed by Cash and Karp (Cash and Karp, 1990). The main advantage of this method is that it lets the algorithm control the error, by dynamically changing the iteration step. We modified the method to control the step size and introduced many numerical domains such as interval or global error in order to have a safe algorithm. The main advantage of our method, in comparison with classical interval methods, is that the upper and lower limits are calculated separately. Consequently, we avoid the well-known *wrapping effect* which is the main cause for instability of interval methods. In the following, we concisely present the order 5 Runge-Kutta-Cash-Karp method, then we explain the changes we made. Finally, we present some experimental results which compare our method to VNODE, a validating interval solver (Nedialkov and Jackson, 2001).

4.1. Runge-Kutta-Cash-Karp Method

The term Runge-Kutta method embraces a family of numerical integration iterative algorithms. These algorithms compute, given a differential equation $\dot{y} = f(y, x)$ and an initial value $y(x_0) = y_0$, an approximation of the value y_1 of y at $x_0 + h$. The principle is to compute a certain number of intermediary points in order to increase precision. Moreover, these methods give an approximation of the discretisation error. When this error is $O(h^n)$, the method is called of order n. The most popular method is the so-called RK4 method, of order 4 (Press et al., 1986).

The Cash-Karp method is an order 5 Runge-Kutta algorithm, based on the Fehlberg method (Fehlberg, 1969), which evaluates f in 6 points, such that another ordering of these points gives an order 4 algorithm. These two arrangements may be used to evaluate y_1 and then give an upper approximation of the error. A step control method can then be implemented, in order to reduce the step size if this error becomes too important. The central point of our algorithm is this step size control method, which uses different numerical domains in order to validate the approximation. In the following, we present this algorithm.

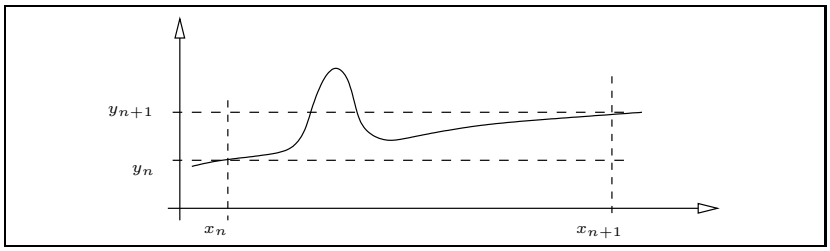

Figure 12. Kind of Irregularities we want to detect.

4.2. The step control method

The major problem with Runge-Kutta methods is that they only give a *local* approximation of the function, whereas our goal is to build a step function that *globally* overapproximates f. So, we must be sure that no irregularities occur between two evaluation points (see Figure 12. We present our algorithm for a differential system composed by a single equation $\dot{y} = f(y, x)$. The extension to multiple variable systems is immediate. Let f be a differentiable function, such that $\dot{y} = f(y, x)$, and ϵ be the maximal allowed error on the result. Assume we already computed an approximation y_n of y at the point x_n. The Runge-Kutta-Cash-Karp method gives an approximation y_{n+1} of y at the point x_{n+1}, as well as an overapproximation e_n of the error. The control method must decide whether this approximation is valid, i.e. it must test that the error e_n is smaller than the tolerance ϵ, and detect whether there is an irregularity between x_n and x_{n+1}. These irregularities can be detected, as the sign of the derivative of f changes in such cases. Therefore, if we compute $I = f([y_n, y_{n+1}], [x_n, x_{n+1}])$ using interval arithmetic, and if $0 \in I$, there is a potential irregularity. In this case, we must reject the step and reduce its size, until both points x_n and x_{n+1} are on the same side of the irregularity.

To get through the irregularity, another case has to be considered, as shown on Figure 13: when the derivative at x_n and at x_{n+1} are not of the same sign. In this case, $\dot{y}(x) = 0$ for some $x \in [x_n, x_{n+1}]$ and, as a consequence, evaluating f on this interval is not sufficient. However, in the case where there is an irregularity, the second derivative $\ddot{y}$ takes the value 0 in this interval. So, we first compute g such that $\ddot{y} = g(y, x)$ using standard multiple variable derivation formulae, and then compute $J = g(R)$, where R is the rectangle shown on Figure 13. This rectangle is obtained by using linear interpolation in order to find its maximal point α, such that the rectangle contains the highest and lowest point of the curve. If $0 \in J$, we reject the current step and reduce its size.

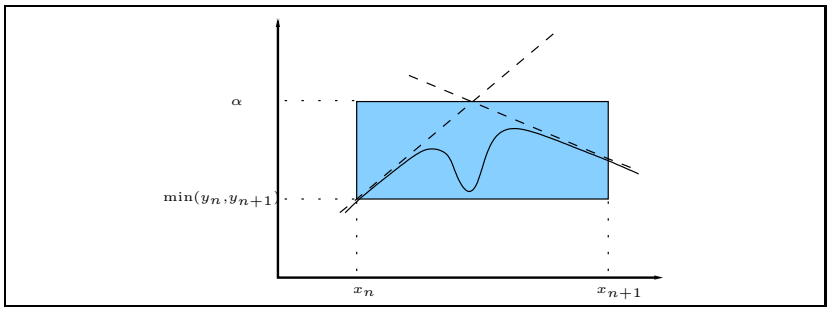

Figure 13. Case where the derivative has different sign in x_n and x_{n+1}.

Unfortunately, this form of the algorithm is not yet safe, as we considered that all numerical computations were carried out using real numbers. This is not the case when we want to implement it on a computer using floating-point numbers that conform to the IEEE754 standard (ANSI and IEE, 1985), or if we use multi-precision arithmetic domains ($\mathbb{M}$) (Daney et al., 2005). The loss of precision may cause the algorithm to believe that the error on y_{n+1} is smaller than ϵ, whereas, using real numbers, the error would have been bigger. To deal with this problem, we implement the computation of the next step of the iteration using the domain of multi-precision floating-point numbers with global error $\mathbb{M} + \mathbb{E}$. Recall that the definition of the global error domain is given in Section 2.2.3. Then, the result of the computation of y_{n+1} is a tuple (y, e, e_n) where y is a multi-precision floating-point approximating $y(x_{n+1})$, e_n an approximation of the error due to the approximation method and e is the error due to the use of floating-point numbers. We have $y_{n+1} \in [y - e, y + e]$, and, consequently, we must not compare e_n with ϵ, but $e_n + e$. So, if $e_n + e \leq \epsilon$, we are sure that the real result of the approximation algorithm is close enough to $y(x_{n+1})$, and we may validate y.

The main advantage of this method is that it uses interval arithmetic only for the validation of one iteration step. Then, the instability introduced by the wrapping effect disappears. Moreover, this algorithm gives a *global* enclosing of the real solution, where most integration algorithms only give *local* approximations. In addition to that, the use of the global error domain takes into account the round-off errors inherent to the representation of real numbers on a computer, which guarantees the safety of the algorithm.

4.3. Experimental Results

Our algorithm has been implemented in C++, using a JAVA class for computing the second derivative and we compare it to a validated

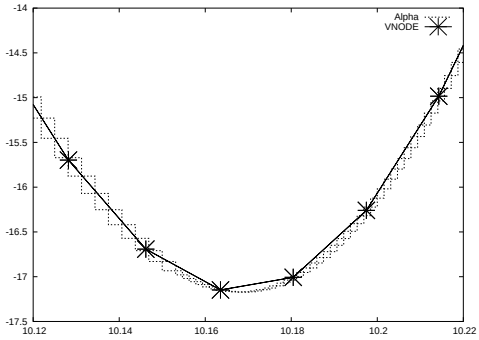

Figure 14. Comparison between VNODE and the α-algorithm for the Lorenz equations.

numerical integration algorithm which uses interval arithmetic in order to compute the iteration steps, VNODE (Nedialkov and Jackson, 2001). We use for this comparison three problems that are provided with the VNODE software: Lorenz Equations, the Tow-Body problem and Vander-Pol equations. We next present the result for the Lorenz Equations problem only, the two others are detailed in Appendix A.

The differential system we have to solve is the following:

$$\begin{cases} f' &= \sigma.(g-f) \\ g' &= f.(\rho-h)-g \\ h' &= f.g-\beta.h \end{cases} \qquad \begin{cases} f_0 &= 15.0 \\ g_0 &= 15.0 \\ h_0 &= 36.0 \end{cases}$$

Figure 14 shows the curves we obtain using our algorithm (called α), represented as the two dotted lines, and the one using VNODE, represented as the plain line, which is an extrapolation using the computed points represented by the crosses. This figure shows that our algorithm is more precise than VNODE in the sense that it computes more points, and gives an exact enclosing of the real curve between these points. It also shows that this algorithm is safe, as all points computed by VNODE are enclosed by the two step functions computed by the α-algorithm. Moreover, in this example, VNODE was unable to compute the solution after $x = 25$, as the error generated by the interval arithmetic exploded because of wrapping effect. On the contrary, our algorithm was able to perform the approximation much further, reducing gradually the step size. The fundamental difference between the two methods is that our algorithm computes the two bounds independently one from each other, which makes us avoid the wrapping effect. If this increases computation time, it also provides much more stable results than interval techniques.

5. Conclusion

The first contribution of this article is a new formalism for modelling hybrid systems. Our approach focuses on a complete separation between the continuous and discrete sub-systems, while formalizing their interactions via sensors and actuators. This new model is intended to be usable for representing large existing applications in an hybrid way. Next, we provide an analysis framework, based on abstract interpretation, for proving safety properties on hybrid systems. These properties concern the loss of numerical precision due to the use of floating-point numbers. We developed a new domain, based on the notion of step functions, and a specific approximation method for the evolution of the continuous part. This method takes into account the effect of actuators which dynamically modify the continuous evolution of the system. What is still missing is an extension of our approximation algorithm for a larger class of dynamical systems, mostly systems where the variables may be mixed. This raises the question of enclosing as precisely as possible the evolution of dynamical systems whose dynamics may change with time.

A prototype analyzer currently is under development. We plan to carry out some experiments with it, in order to confirm the interest of our domains and to improve them thanks to the first results. for instance, a better widening operator could be defined. We would like to find criterion in order to have a more powerful widening, where time is set to infinity. In this case, we need a safe upperapproximation of the evolution of continuous functions when time goes to infinity, which is complicated in the general case. There are some criteria to find, such as a limit (for example if the second derivative of the function is of constant sign after some point), but it is not clear what to do without any information on the continuous function. Another point that we would like to explore is the treatment of periodic functions, which, we believe, needs a special algorithm, as they clearly allow easy widening, even when time goes to infinity. We also aim at including, in our prototype analyzer, the direct computation of the safety properties described in Section 3.3.2 concerning the divergences of actions performed by an ideal model and its implementation.

To conclude let us insist on the fact that the hybrid system abstract interpretation framework we have developed in this article is a first step towards a fine analysis of the numerical precision of complex embedded systems. As mentioned in the introduction, precise properties about the numerical quality of codes requires a good abstraction of the (physical) inputs of programs. Additional examples illustrating this problem can be found in (Goubault et al., 2006).

References

Alur, R., C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine: 1995, 'The algorithmic analysis of hybrid systems'. *Theoretical Computer Science* **138**(1), 3–34.

ANSI and IEE: 1985, *IEEE Standard for Binary Floating-Point Arithmetic*. ANSI/IEEE Std 754-1985. American National Standards Institute and Institute of Electrical and Electronic Engineers.

Bertrane, J.: 2005, 'Static Analysis by Abstract Interpretation of the Quasi-synchronous Composition of Synchronous Programs.'. In: *VMCAI 05 : Proceedings of the 6th International Conference on Verification, Model Checking, and Abstract Interpretation*, Vol. 3385 of *Lecture Notes in Computer Science*. pp. 97–112.

Blanchet, B., P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival: 2002, 'Design and Implementation of a Special-Purpose Static Program Analyzer for Safety-Critical Real-Time Embedded Software'. In: T. Mogensen, D. A. Schmidt, and I. H. Sudborough (eds.): *The Essence of Computation: Complexity, Analysis, Transformation.*, Vol. 2566 of *Lecture Notes in Computer Science*. Springer-Verlag, pp. 85–108.

Bouissou, O.: 2005, 'Analyse statique par interpretation abstraite de systme hybrides discrets-continus'. Technical Report 05-301, CEA-LIST.

Cash, J. R. and A. H. Karp: 1990, 'A variable order Runge-Kutta method for initial value problems with rapidly varying right-hand sides'. *ACM Trans. Math. Softw.* **16**(3), 201–222.

Cousot, P. and R. Cousot: 1977, 'Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints'. In: *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. Los Angeles, California, pp. 238–252.

Cousot, P. and R. Cousot: 1992, 'Abstract Interpretation Frameworks'. *Journal of Logic and Computation* **2**(4), 511–547.

Daney, D., G. Hanrot, V. Lefvre, P. Plissier, F. Rouillier, and P. Zimmermann: 2005, 'The MPFR Library'.

Fehlberg, E.: 1969, 'Low-order classical Runge-Kutta formulas with stepsize control and their application to some heat transfer problems'. Technical Report TR-R-315, NASA.

Goubault, E., M. Martel, and S. Putot: 2002, 'Asserting the Precision of Floating-Point Computations: A Simple Abstract Interpreter'. In: *ESOP '02: Proceedings of the 11th European Symposium on Programming Languages and Systems*. pp. 209–212.

Goubault, E., M. Martel, and S. Putot: 2006, 'Some future challenges in the validation of control systems'. In: *European Congress on Embedded Real Time Sofware*.

Henzinger, T. A.: 1996, 'The Theory of Hybrid Automata'. In: *Proceedings of the 11th Annual Symposium on Logic in Computer Science*. pp. 278–292.

Henzinger, T. A., P. H. Ho, and H. Wong-Toi: 1997, 'HyTech: A model checker for hybrid systems'. In: *CAV 97: Computer-Aided Verification*, Lecture Notes in Computer Science 1254. Springer-Verlag, pp. 460–463.

Hoare, C. A. R.: 1985, *Communicating Sequential Processes*. Prentice Hall.

Martel, M.: 2005, 'An Overview of Semantics for the Validation of Numerical Programs.'. In: *VMCAI 05 : Proceedings of the 6th International Conference on*

Verification, Model Checking, and Abstract Interpretation, Vol. 3385 of *Lecture Notes in Computer Science*. pp. 59–77.

Moore, R.: 1979, *Methods and Applications of Interval Analysis*. Philadelphia, PA: SIAM.

Mosterman, P. J.: 1999, 'An Overview of Hybrid Simulation Phenomena and their Support by Simulation Packages'. In: *Hybrid Systems: Computation and Control*. pp. 165–177.

Nedialkov, N. S. and K. R. Jackson: 2001, 'The Design and Implementation of an Object-Oriented Validated ODE Solver'.

Press, W., B. Flannery, S. A. Teukolsky, and W. T. Vetterling: 1986, *Numerical Recipes: The Art of Scientific Computing*. Cambridge (UK) and New York: Cambridge University Press, 1st edition.

Reed, G. M. and A. W. Roscoe: 1988, 'A timed model for communicating sequential processes'. *Theor. Comput. Sci.* **58**(1-3), 249–261.

Appendix

A. Results

A.1. The Two-Body Problem

The so-called Two-Body problem consists in solving the following differential equations:

$$\begin{cases} f' &= h \\ g' &= l \\ h' &= -1.0.f/((f.f+g.g).sqrt(f.f+g.g)) \\ l' &= -1.0.g/((f.f+g.g).sqrt(f.f+g.g)) \end{cases} \quad \begin{cases} f_0 &= 0.1 \\ g_0 &= 0.0 \\ h_0 &= 0.0 \\ l_0 &= 4.36 \end{cases}$$

Figure 15 shows the results obtained with VNODE and the α-algorithm. Once again, our algorithm is globally more precise than VNODE, and more stable because VNODE was not able to compute the approximation of the function after $x = 56$. However, it should be noted that locally, the points computed by VNODE are more precise than the steps computed by the α-algorithm. This is a consequence of the wish of a global approximation of y: when y_{n+1} is computed, we do not only look for an approximation of $y(x_{n+1})$ but rather for a upperapproximation of $\{f(t) \mid t \in [x_n, x_{n+1}]\}$. Therefore, we must be locally less precise than VNODE, even if we are globally closer to the real curve.

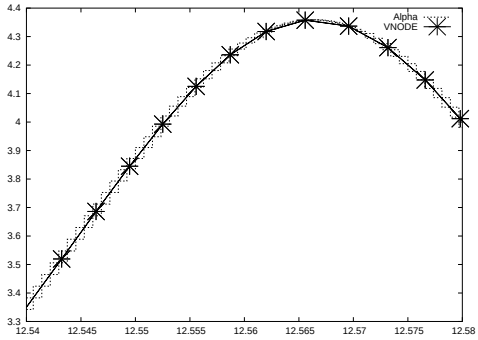

Figure 15. Comparison between VNODE and the α-algorithm for the Two-Body Problem.

A.2. Van der Pol Equations

The Van der Pol equations are:

$$\begin{cases} f' &= & h \\ g' &= & 2.(1-f^2).g - f \end{cases} \qquad \begin{cases} f_0 &= & 2.0 \\ g_0 &= & 0.0 \end{cases}$$

Figure 16 represents the curves obtained by VNODE and the α-algorithm. This example turned in favor of VNODE, as it is as stable as the α-algorithm, but almost twice as fast as our method. The reason why it is much quicker is that we avoid, in the α-algorithm, to extend the step size, whereas it could be possible without loss of local precision.

As the global precision would clearly suffer from it, we prefer to avoid such a refinement.

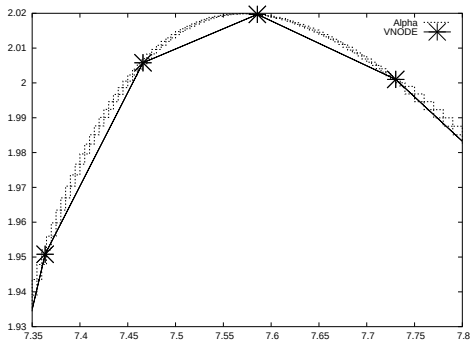

Figure 16. Comparison between VNODE and the α-algorithm for the Van der Pol Equations.

B. The Algorithm

Algorithm 1 Step size control algorithm

Require: yerr,**y[n+1]**=yout+e,eps,F,dydt,x,n,y
  if e+yerr $\geq$ eps **then**
    return false;
  end if
  dyout$\leftarrow$F(yout,x[n]+h_n)
  if yout $\leq$ y[n] **then**
    I$\leftarrow$F([y[n]-h_n/100,yout+h_n/100],[x[n],x[n]+h_n])
  else
    I$\leftarrow$F([yout-h_n/100,y[n]+h_n/100],[x[n],x[n]+h_n])
  end if
  if dydt[n]$\geq$0 **then**
    if dyout$\geq$0 **then**
      if I$\cap\mathbb{R}_*^- \neq \emptyset$ **then**
        if |min(I).h_n| $\geq$eps **then**
          return false
        end if
      end if
      return true
    else {dydt[n]$\leq$0 and dyout$>$0}
      $\alpha\leftarrow$(y[n+1]-y[n]-dydt[n+1]*x[n+1]+dydt[n]*x[n])/(dydt[n]-dydt[n+1]);
      dI$\leftarrow$dF([min(y[n],yout),α],[x[n],x[n]+h_n]);
      if dI$\cap\mathbb{R}_*^+ \neq \emptyset$ **then**
        return false
      else
        return true
      end if
    end if
  else {dydt[n]$>$0}
    ...
  end if