Denotational Semantics of Hybrid Automata*

Abbas Edalat[1] and Dirk Pattinson[2]

[1] Department of Computing, Imperial College London, UK
[2] Department of Computer Science, University of Leicester, UK

Abstract. We introduce a denotational semantics for non-linear hybrid automata, and relate it to the operational semantics given in terms of hybrid trajectories. The semantics is defined as least fixpoint of an operator on the continuous domain of functions of time that take values in the lattice of compact subsets of n-dimensional Euclidean space. The semantic function assigns to every point in time the set of states the automaton can visit at that time, starting from one of its initial states. Our main results are the correctness and computational adequacy of the denotational semantics with respect to the operational semantics and the fact that the denotational semantics is computable.

1 Introduction

A hybrid automaton [12, 2] is a digital, real-time system that interacts with an analogue environment. Hybrid automata are ubiquitous in all areas of modern engineering and technology. For example, the (digital) height control of an automobile chassis depends on and influences the (continuous) driving conditions of the vehicle [18]. Hybrid automata typically operate in safety critical areas, such as the highway control systems [17] and air traffic control [20]. They combine a finite set of control states with continuous dynamics. In every control state, the continuous variables evolve according to an ordinary differential equation and the system changes control states if the continuous variables reach certain thresholds.

One of the key concerns in the theory of hybrid automata is the algorithmic verification of safety critical properties. This problem is well understood for linear systems [3] and implemented in the model checker HyTech [13]. The situation for non-linear systems is, not surprisingly, much less satisfactory. While the approximation of non-linear hybrid automata by linear systems is asymptotically complete [14], it results in a huge blow-up in the number of discrete control states and associated state transitions, which limits the possibilities of algorithmic analysis.

This paper presents an alternative approach. Conceptually, we regard a hybrid automaton as the integration of two different types of systems: the evolution of a family of continuous systems, governed by differential equations, and the dynamics of a discrete system given by a generalised iterated function system (IFS), see [16]. We synthesise the domain-theoretic approach to solving differential equations [7, 10] and the domain-theoretic approach to obtain the attractor of an iterated function system [6] to develop a domain-theoretic semantics for general hybrid automata. The denotational semantics assigns to every time point t the set $[\![H]\!](t)$ of states that the automaton H

* This work has been partially supported by DFG (Germany) and the European Union.

L. Aceto and A. Ingólfsdóttir (Eds.): FOSSACS 2006, LNCS 3921, pp. 231–245, 2006.

can enter at time t. The semantic function $[\![H]\!]$ is obtained as the least fixpoint in the (continuous) domain of compact-set valued functions of a real variable. Our first main results are correctness and computational adequacy of this denotational semantics w.r.t. the operational semantics, given in terms of a labelled transition system. Moreover, standard techniques of domain theory allow us to actually compute this function. The implications are twofold: we obtain new results on the computability of trajectories in the domain theoretic model, and our analysis gives rise to a directly implementable algorithm that computes approximations to the semantic function $[\![H]\!]$ up to an arbitrary degree of accuracy. As the algorithm works on proper data types, defined e.g. over the dyadic numbers, this property is moreover guaranteed for implementations.

The paper is divided in two parts. In the first part, we focus on flow automata, where the behaviour of the continuous variables in every discrete control state is governed by flow functions, which behave like the solutions of ordinary differential equations. We impose two conditions on the automata under scrutiny: first, we require that the ingredients of the automaton give rise to Scott continuous functions on the respective domains. In order to show that the least fixpoint precisely captures the reachable states, we assume that the automaton is separated, i.e. has no transient states which the automaton can leave immediately (after 0 time units) after entering. We discuss these restrictions by means of examples, and show that the semantic function associated with a flow automaton cannot be computable in absence of these properties.

In the second part of the paper, we transfer the results obtained to hybrid automata, where the trajectories of the continuous variables are given by a vector field. By instantiating earlier results on domain theoretic solutions of initial value problems, we reduce the problem of computing the semantic function of a hybrid automaton to that of a flow automaton. Taken together, the domain theoretic approach provides a new computational model for the analysis of hybrid systems, and gives rise to both new computability results, and directly implementable data types and algorithms for the analysis of non-linear systems.

Related Work. We have already mentioned symbolic techniques for the analysis of linear hybrid automata [3] and their implementation in the HyTech model checker [13]. The domain theoretic approach of this paper is related to the interval analysis approach of [15], where interval numerical methods are used to compute over-approximations of the set of reachable states. In contrast to *loc.cit.*, where outward rounding is required if the result of an arithmetic operation is not machine representable, the domain theoretic model of computation actually allows to compute the semantic function up to an arbitrary degree of accuracy.

2 Preliminaries and Notation

We use basic domain theoretic notions, see e.g. [1, 11]. In particular, our analysis employs the following domains defined over the real numbers: the domain of n-dimensional compact rectangles extended with a least element

$$\mathbf{I}\mathbb{R}^n = \{a \subseteq \mathbb{R}^n \mid a \text{ nonempty compact rectangle}\} \cup \{\mathbb{R}^n\},$$

ordered by reverse inclusion, and the *extended upper space*

$$\mathbf{U}^\top \mathbb{R}^n = \{c \subseteq \mathbb{R}^n \mid c \text{ compact}\} \cup \{\mathbb{R}^n\}$$

of compact subsets of $\mathbb{R}^n$, also ordered by reverse inclusion. Note that the extended upper space arises by extending the upper space [5] with the top element $\top = \emptyset$. A closed *semi rectangle* in $\mathbb{R}^n$ is of the form $a_1 \times \cdots \times a_n$, where the a_i are closed (not necessarily bounded) intervals in $\mathbb{R}$. If A is a semi-rectangle, we write $\mathbf{I}\mathsf{A} = \{\mathsf{A} \cap r \mid r \in \mathbf{I}\mathbb{R}^n\}$ and $\mathbf{U}^\top \mathsf{A} = \{\mathsf{A} \cap c \mid c \in \mathbf{U}^\top \mathbb{R}^n\}$ for the sub-domain of all elements above A. In particular, we will consider the domain $\mathbf{I}[0, \infty)$, whose bottom element is $\bot = [0, \infty)$. For a semi rectangle A, $\mathbf{I}\mathsf{A}$ is a continuous Scott domain and $\mathbf{U}^\top \mathsf{A}$ is a continuous lattice. We often consider $\mathbf{I}\mathsf{A} \subseteq \mathbf{U}^\top \mathsf{A}$ as a sub-domain without making this explicit; similarly, we identify $x \in \mathbb{R}^n$ with the degenerate hyper-rectangle $\{x\} \in \mathbf{I}\mathbb{R}^n \subseteq \mathbf{U}^\top \mathbb{R}^n$. We write $\bot = \mathsf{A}$ for the least element of both $\mathbf{I}\mathsf{A}$ and $\mathbf{U}^\top \mathsf{A}$, and $\top = \emptyset$ for the top element of $\mathbf{U}^\top \mathsf{A}$. Note that the way-below relation, both in $\mathbf{I}\mathsf{A}$ and $\mathbf{U}^\top \mathsf{A}$, is given by $a \ll b$ iff $b \subseteq a^o$, where a^o is the interior of a.

If $(C_i)_{i \in I}$ is a family of compact subsets $C_i \subseteq \mathbb{R}^{n_i}$, we identify $(x_i)_{i \in I} \in \prod_{i \in I} \mathbf{U}^\top C_i$ with the set $\{(i, y) \mid i \in I, y \in x_i\}$ for convenience of notation. Note that this induces a membership predicate and subset relation, which are explicitely given by

$$(j, z) \in (x_i)_{i \in I} \iff z \in x_j \text{ and } (x_i)_{i \in I} \subseteq (y_i)_{i \in I} \iff \forall i \in I. x_i \subseteq y_i$$

where $(x_i)_{i \in I}$ and $(y_i)_{i \in I} \in \prod_{i \in I} \mathbf{U}^\top C_i$, $j \in I$ and $z \in C_j$. Moreover, we obtain two continuous maps $\bigcap, \bigcup$, whose explicit definition reads

$$\diamond : (\prod_{i \in I} \mathbf{U}^\top C_i)^2 \to \prod_{i \in I} \mathbf{U}^\top C_k, \quad ((x_i)_{i \in I}, (y_i)_{i \in I}) \mapsto (x_i \diamond y_i)_{i \in I}$$

where $\diamond \in \{\bigcap, \bigcup\}$. Note that, domain theoretically, $\bigcap$ is the least upper bound and $\bigcup$ gives us the greatest lower bound of two elements of $\prod_{i \in I} \mathbf{U}^\top C_i$. We always consider sub-domains of the upper space or the interval domain as equipped with the Scott topology.

The symbol $\Rightarrow$ is used for the continuous function space. In particular, for semi rectangles A, B, we consider the set $(\mathsf{A} \Rightarrow \mathbf{U}^\top \mathsf{B})$ of functions $f : \mathsf{A} \to \mathbf{U}^\top \mathsf{B}$ which are continuous with respect to the Euclidean topology on A and the Scott topology on $\mathbf{U}^\top \mathsf{B}$. Similarly, $(\mathbf{U}^\top \mathsf{A} \Rightarrow \mathbf{U}^\top \mathsf{B})$ denotes the set of functions that are continuous w.r.t. the Scott topology on $\mathbf{U}^\top \mathsf{A}$ and $\mathbf{U}^\top \mathsf{B}$; the same applies to the interval domain.

We extend the ordinary arithmetical operations to the upper space without further mention. In particular, we write $a \diamond b = \{x \diamond y \mid x \in a, y \in b\}$, where $\diamond \in \{+, -, *, /\}$ and $a, b \in \mathbf{U}^\top \mathbb{R}^n$. (We adopt the standard convention that $a/b = \bot$ if $0 \in b$.)

It is a straightforward exercise to see that Scott continuous functions of type $\mathsf{A} \to \mathbf{U}^\top \mathsf{B}$ are precisely the semi continuous functions of set-valued analysis [4]. More concretely, we have that $f : \mathsf{A} \to \mathbf{U}^\top \mathsf{B}$ is Scott continuous, iff

$$\forall x \in A \forall \epsilon > 0 \exists \delta > 0 \forall x' \in B_\epsilon(x). f(x') \subseteq f(x) + B_\delta$$

where $B_\epsilon(x) = \{x' \in \mathsf{A} \mid \|x - x'\| < \epsilon\}$ and $B_\delta = B_\delta(0)$. Note that we have the Scott continuous *extension mapping*

$$\mathcal{E} : (A \Rightarrow \mathbf{U}^\top \mathsf{B}) \to (\mathbf{U}^\top \mathsf{A} \Rightarrow \mathbf{U}^\top \mathsf{B}), f \mapsto \lambda x. \bigsqcap_{y \in x} f(y),$$

and it is an easy exercise to show that this greatest lower bound is actually given by direct image, i.e. $\mathcal{E}(f)(x) = \bigcup\{f(y) \mid y \in x\}$.

3 Flows and Flow Automata

We begin our study of hybrid automata by first discussing *flow automata*, where the continuous evolution in every control state is an explicitly given flow function. This will subsequently be shown to be equivalent to the case that the continuous evolution is specified by a vector field in Section 6. For flow automata, every discrete control state comes with a flow function that behaves like the solution of an initial value problem, and governs the evolution of the continuous variables in that state.

Definition 1. *A* flow *is a continuous function $f : \mathbb{R}^n \times [0,\infty) \to \mathbb{R}^n$, which is continuously differentiable w.r.t. its last argument, such that $f(x,0) = x$ and $f(x, s+t) = f(f(x,s),t)$ for all $x \in \mathbb{R}^n$ and $s,t \in [0,\infty)$.*

That is, a flow $f : \mathbb{R}^n \times [0,\infty) \to \mathbb{R}^n$ behaves like the solution of an initial value problem $\dot{f}(t) = v(f(t)), f(0) = x$, where v is defined on the whole of Euclidean space $\mathbb{R}^n$. Note that flows typically arise as solutions of initial value problems:

Lemma 1. *Suppose $v : \mathbb{R}^n \to \mathbb{R}^n$ is a Lipschitz vector field. If $f(x,\cdot)$ denotes the (unique) solution of the initial value problem $f(x,t) = v(f(x,t)), f(x,0) = x$, then f is a flow. We say that f is the* flow induced by *v.*

We now introduce continuous flow automata.

Definition 2. *A* flow automaton *in $\mathbb{R}^n$ is a tuple $F = (Q, \mathrm{inv}, \mathrm{flow}, \mathrm{res}, \mathrm{init})$ where*

- *Q is a finite set of discrete control states*
- $\mathrm{inv} = (\mathrm{inv}(q))_{q\in Q}$ *is a family of* state invariants *with* $\mathrm{inv}(q) \subseteq \mathbb{R}^n$
- $\mathrm{flow} = (\mathrm{flow}(q))_{q\in Q}$ *is a family of* flow functions $\mathrm{flow}(q) : \mathbb{R}^n \times [0,\infty) \to \mathbb{R}^n$
- $\mathrm{res} = (\mathrm{res}(p,q))_{p,q\in Q}$ *is a family of* reset relations *with* $\mathrm{res}(p,q) : \mathrm{inv}(p) \to \mathcal{P}(\mathrm{inv}(q))$
- $\mathrm{init} = (\mathrm{init}(q))_{q\in Q}$ *is a family of* initial states *with* $\mathrm{init}(q) \subseteq \mathrm{inv}(q)$.

We call a flow automaton compact*, if $\mathrm{inv}(q), \mathrm{init}(q) \in \mathbf{U}^\top\mathbb{R}^n$ are compact for all $q \in Q$ and $\mathrm{res}(p,q)(x) \in \mathbf{U}^\top\mathrm{inv}(q)$ is a compact subset of $\mathrm{inv}(q)$ for all $p,q \in Q$ and all $x \in \mathrm{inv}(q)$. A* state *of a flow automaton is a tuple (q,x) with $q \in Q$ and $x \in \mathrm{inv}(q)$. We write $S_F = \{(q,x) \mid q \in Q, x \in \mathrm{inv}(q)\}$ for the state space of F and $i_F = \{(q,x) \in S \mid x \in \mathrm{init}(q)\}$ for the set of initial states.*

Although our interest in the flow function will be restricted to values $\mathrm{flow}(q)(x,t)$, where $x \in \mathrm{inv}(q)$, the flow function is defined on the whole of Euclidean space for convenience.

The above definition of flow automata, while slightly different, is equivalent to the standard definition given e.g. in [3]. While our control states are in one-to-one correspondence to the control locations of *loc.cit.*, the transitions between control states are

modelled in terms of a finite multiset $V \subseteq Q \times Q$ of transitions, and an action predicate $\mathrm{act}(v) \subseteq \mathbb{R}^n \times \mathbb{R}^n$ is assigned to every transition $v \in V$. In this terminology, the automaton can change its state, say from state (q, x) to state (q', x') iff there exists a transition $(q, q') \in V$ with $(x, x') \in \mathrm{act}(v)$. In our terminology, this can be modelled by the reset relation $\mathrm{res}(q, q') = \lambda x.\{y \in \mathrm{inv}(q) \mid \exists (q, q') \in V.(x, y) \in \mathrm{act}(q, q')\}$.

For the remainder of the paper, we assume that all flow automata are compact. Our main interest lies in the comparison of the denotational semantics and the operational semantics of a flow automaton. The latter is given in terms of a labelled transition system, where a label is either a non-negative real numbers, that signifies time, or τ, indicating that the automaton is changing its discrete control state.

Definition 3. *Suppose $F = (Q, \mathrm{inv}, \mathrm{flow}, \mathrm{res}, \mathrm{init})$ is a flow automaton and let $\Sigma = [0, \infty) \cup \{\tau\}$. The* associated transition system *T_F is the tuple $(S_F, \rightarrow)$, where S_F is the state space of F and $\rightarrow \subseteq S \times \Sigma \times S$ is defined by the following two clauses:*

flow transitions $(q, x) \rightarrow^t (q', x')$ *iff* $q = q'$, $\mathrm{flow}(q)(x, t_0) \in \mathrm{inv}(q)$ *for all* $t_0 \in [0, t]$ *and* $\mathrm{flow}(q)(x, t) = x'$

jump transitions $(q, x) \rightarrow^\tau (q', x')$ *iff* $x' \in \mathrm{res}(q, q')(x)$

For states $s, s' \in S$, we write $s \rightarrow_*^t s'$ if there is a finite sequence of states $s_1, \ldots, s_k$ with $s \rightarrow^{\delta_1} s_1 \rightarrow^{\delta_2} \cdots \rightarrow^{\delta_k} s_k = s'$ with $\delta_1, \ldots, \delta_k \in \Sigma$ and $\sum_{\delta_k \in [0,\infty)} \delta_k = t$. We write $\mathrm{init} \rightarrow_*^t s$ iff there exists $i \in i_F$ with $i \rightarrow_*^t s$.

An F-trajectory *is a finite or infinite sequence $(t_i, q_i, f_i)_{i<N}$ where $N \in \mathbb{N} \cup \{\infty\}$ such that $(t_i)_{i<N}$ is non-decreasing in $[0, \infty)$, $(q_i)_{i<N}$ is a sequence in Q and $f_i : [t_{i-1}, t_i] \rightarrow \mathbb{R}^n$ is a function (we use the convention that $t_{-1} = 0$) that, for all $i < N$, satisfies*

- *$f_0(t_{-1}) \in \mathrm{init}(q_0)$ and $(q_i, f_i(t_{i-1})) \rightarrow^t (q_i, f_i(t_{i-1} + t))$ for all $t \in [t_{i-1}, t_i]$*
- *$(q_i, f_i(t_i)) \rightarrow^\tau (q_{i+1}, f_{i+1}(t_i))$.*

We denote the set of possible states of the automaton F at time t by $R_F(t)$ and the set of all states the automaton can visit up to time t by $V_F(t)$, formally defined by

$$R_F(t) = \{s \in S_F \mid \mathrm{init} \rightarrow_*^t s\} \quad \textit{and} \quad V_F(t) = \bigcup \{R_F(s) \mid s \leq t\}$$

where $t \in [0, \infty)$.

Note that by assumption, $\mathrm{flow}(q_i)(f_i(t_{i-1}), t) = f_i(t_{i-1} + t)$. Compared with the definition of trajectories in [2], it is straightforward to verify that, under the correspondence outlined after Definition 2, our definition of trajectories gives rise to the same semantics.

We now turn to the main issue of the present paper and describe the necessary ingredients needed to perform domain theoretic analysis of a flow automaton F. Our main goal is to define a domain theoretic semantic function $[\![F]\!] : [0, \infty) \rightarrow \prod_{q \in Q} \mathbf{U}^\top \mathrm{inv}(q)$. The function $[\![F]\!]$ associated to every time point $t \in [0, \infty)$ an element of $\prod_{q \in Q} \mathbf{U}^\top \mathrm{inv}(q)$. That is, to every point in time t we associate a family $(s_q)_{q \in Q}$, with $s_q \subseteq \mathrm{inv}(q)$, of compact sets such that $\{(q, x) \mid x \in s_q\} = R_F(t)$. Having computed R_F, it is easy to derive a mechanism for computing the possibly visited states $V_F(t)$ at time t by unfolding the definition of V_F. We demonstrate later, that it is also possible to obtain V_F directly as a fixed point.

The goal of the construction is to give a *continuous* semantics of flow automata: if the automaton is *effectively given*, i.e. both flow and res arise as limits of sequences of finitary approximations with $\mathrm{flow} = \bigsqcup_{k\in\mathbb{N}} f_k$ and $\mathrm{res} = \bigsqcup_{k\in\mathbb{N}} r_k$, then we can effectively obtain $\sigma_k : [0,\infty) \to \prod_{q\in Q} \mathrm{inv}(q)$ such that $[\![F]\!] = \bigsqcup_{k\in\mathbb{N}} \sigma_k$. This provides us with three important properties:

1. Every σ_k is a *conservative approximation* of the semantics of F, for $k \geq 0$.
2. The semantics of F can be computed up to an arbitrary degree of accuracy.
3. The algorithm for computing σ_k can be implemented on a digital computer without loss of precision.

Clearly, continuity of the semantics mapping $[\![\cdot]\!]$ can only be achieved if we restrict attention to flow automata, whose components are continuous. This motivates the next definition.

Definition 4. *A flow automaton* $F = (Q, \mathrm{inv}, \mathrm{flow}, \mathrm{res}, \mathrm{inv})$ *is* continuous*, if* $\mathrm{res}(p,q) : \mathrm{inv}(p) \to \mathbf{U}^{\top}\mathrm{inv}(q)$ *is Scott continuous for all* $p, q \in Q$*. We say that* F *is* separated*, if*

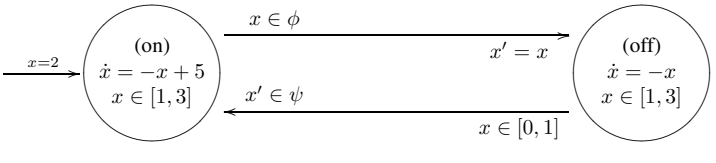
- $x \in \mathrm{res}(p,q)(y)$ *implies that* $\mathrm{res}(q,r)(x) = \emptyset$ *for all* $p, q, r \in Q$ *and* $y \in \mathrm{inv}(p)$
- $x \in \mathrm{init}(q)$ *implies that* $\mathrm{res}(q,r)(x) = \emptyset$ *for all* $q, r \in Q$

While the continuity condition on res is clearly enforced by our goal to be able to approximate the semantics of flow automata, the separation condition tells us that there are no transient states, i.e. the automaton cannot perform state changes from q_0 to q_1, and subsequently from q_1 to q_2 without remaining in state q_1 for a non-zero amount of time.

We will see later that separation and continuity imply that the automaton under scrutiny is non-zeno. While we believe that all of our results can be established even for non-separated automata under the additional assumption that the automata are non-zeno, the main benefit of the separation property is that it is very easy to verify.

For a continuous flow automaton, the family $\mathrm{res}(p,q)_{p,q\in Q}$ induces a generalised IFS on the extended upper spaces of $\mathrm{inv}(p)$, for $p \in Q$, as we will see in Definition 5 later on. The following example discusses the requirements introduced in Definition 4.

Example 1. We consider the following variant F of a thermostat automaton, see e.g. [14]. Let $Q = \{\mathrm{on}, \mathrm{off}\}$ with $\mathrm{inv}(q) = [1,3]$ for $q = \mathrm{on}, \mathrm{off}$. The flow functions are given by the differential equations $\mathrm{flow}(\mathrm{on})(x_0, \cdot) =$ the unique solution of $\dot{x} = -x + 5, x(0) = x_0$, and similarly, $\mathrm{flow}(\mathrm{off})(x_0, \cdot) =$ the unique solution of $\dot{x} = -x, x(0) = x_0$, with initial state $(\mathrm{on}, 2)$. We fix two subsets $\phi, \psi \subseteq [1,3]$ and let $\mathrm{res}(\mathrm{on}, \mathrm{off})(x) = \{x\} \cap \psi$. The function $\mathrm{res}(\mathrm{off}, \mathrm{on})$ is given by $x \mapsto \psi$, if $x \in [0,1]$, and $x \mapsto \emptyset$ otherwise. Graphically, the automaton can be displayed as follows, where x' denotes the value of x after the change of control states.



We now discuss several alternatives for the sets ϕ and ψ, and relate them to continuity of the induced automaton.

1. Suppose $\psi = (1, 2)$. Then $\mathrm{res}(\mathrm{off}, \mathrm{on})$ does not take values in $\mathbf{U}^\top[1, 3]$, as $(1, 2)$ is not compact, hence $\mathrm{res}(\mathrm{off}, \mathrm{on})$ is not a well defined function of type $[1, 3] \to \mathbf{U}^\top[1, 3]$.
2. Suppose $\phi = (2, 3]$. Then the F is not continuous, as for $x = 2$ and $\epsilon > 0$, we fail to find δ s.t. for all $x' \in B_\delta(x)$ we have $\mathrm{res}(\mathrm{on}, \mathrm{off})(x') \in \mathrm{res}(\mathrm{on}, \mathrm{off})(x) + B_\epsilon$.
3. If both ϕ and ψ are compact, then F is continuous.
4. We have that F is separated, iff $\phi \cap [0, 1] = \phi \cap \psi = \emptyset$ and $\phi \cap \{2\} = \emptyset$.

To verify continuity of the reset functions in practice, note that Scott continuity is preserved by function composition, hence all combinations of Scott continuous functions will be Scott continuous. In particular, we note that the following functions are Scott continuous, and thus can be used as building blocks for reset functions.

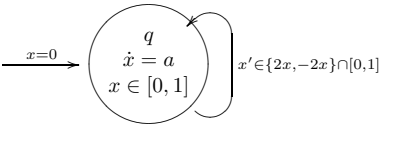
Proposition 2. *Suppose* $\mathsf{A}, \mathsf{B} \in \mathbf{U}^\top\mathbb{R}^n$.

1. *All step functions*
$$a \searrow b : \mathsf{A} \to \mathbf{U}^\top\mathsf{B}, \quad x \mapsto \begin{cases} b & x \in a^o \\ \bot & \textit{otherwise} \end{cases}$$
are continuous for $a \in \mathbf{U}^\top\mathsf{A}, b \in \mathbf{U}^\top\mathsf{B}$*, where* a^o *denotes the interior of* a*.*
2. *All co-step functions*
$$a \nwarrow b : \mathsf{A} \to \mathbf{U}^\top\mathsf{B}, \quad x \mapsto \begin{cases} b & x \in a \\ \top & \textit{otherwise} \end{cases}$$
are continuous for $a \in \mathbf{U}^\top\mathsf{A}, b \in \mathbf{U}^\top\mathsf{B}$*.*
3. *All functions*
$$\bowtie b : \mathsf{A} \to \mathbf{U}^\top\mathsf{B}, \quad x \mapsto \{x\} \cap b$$
are continuous for $b \in \mathbf{U}^\top\mathsf{B}$
4. *If* $f_1, f_2 : \mathsf{A} \to \mathbf{U}^\top\mathsf{B}$ *are continuous, then so is* $f_1 \cup f_2 : \mathsf{A} \to \mathbf{U}^\top\mathsf{B}, x \mapsto f_1(x) \cup f_2(x)$*.*
5. *If* $(f_i)_{i \in I}$ *is directed (w.r.t. the pointwise ordering), then* $\bigsqcup_{i \in I} f_i : \mathsf{A} \to \mathbf{U}^\top\mathsf{B}, x \mapsto \bigsqcup_{i \in I} f_i(x)$ *is continuous.*

The previous proposition gives some general construction principles for continuous hybrid automata, and can be applied to show that a large class of flow automata are actually continuous. We now turn to the separation property. The following example, which is a variation of the bouncing ball automaton [19] shows, that the separation property is vital for the computability of the semantic function associated with a flow automaton.

Example 2. Consider the automaton $F = (Q, \mathrm{init}, \mathrm{flow}, \mathrm{res}, \mathrm{inv})$ with

- $Q = \{q\}$
- $\mathrm{inv}(q) = [0, 1]$
- $\mathrm{flow}(r, t) = r + a \cdot t$
- $\mathrm{res}(x) = \{2x, -2x\} \cap [0, 1]$
- $\mathrm{init}(q) = \{0\}$

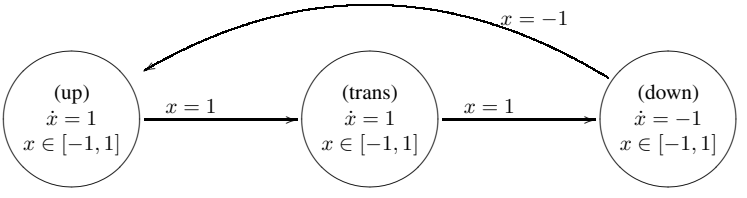


where $a \in \mathbb{R}$ is a computable real number, as depicted on the right above. Suppose we can effectively find a sequence of functions $R_k : [0,1] \to \mathbf{U}^\top[0,1]$ such that $\bigsqcup_{k\in\mathbb{N}} R_k = R_F$. Then clearly $R(1) = \{0\}$ iff $a = 0$, and $R(1) \cap [1/2, 1] \neq \emptyset$ iff $a > 0$. As $R(1) = \bigcap_{k\in\mathbb{N}} R_k(1)$, this implies that we can semi-decide whether $a = 0$. Together with a semi decision procedure for $a \neq 0$, we arrive at a decision procedure for $a = 0$, which is impossible, see e.g. [21].

Recall that a flow automaton is *zeno*, if it admits a trajectory $(t_i, q_i, f_i)_{i<\infty}$ with $\sup_{i<\infty} t_i < \infty$. The key consequence of separation, which makes it possible to compute the semantic function associated with a flow automaton, is that separated automata are non-zeno. This is the content of the next proposition.

Proposition 3. *Suppose F is separated and continuous. Then F is non zeno.*

Note that, while the fact that an automaton is separated is sufficient for it being non-zeno, the separation property is not necessary. Consider for example the automaton



with reset relations $\mathrm{res}(\mathrm{up}, \mathrm{trans}) = \mathrm{res}(\mathrm{trans}, \mathrm{down}) = \lambda x.\{x\} \cap \{1\}$ and $\mathrm{res}(\mathrm{down}, \mathrm{up}) = \lambda x.\{x\} \cap \{-1\}$ and initial state $(\mathrm{up}, 0)$. Then clearly F is non-zeno, but F is not separated. This suggests that the separation property can be relaxed, and one just needs to require that there is no finite loop $(q_0, x_0), (q_1, x_1), \ldots, (q_l, x_l)$ with $x_{i+1} \in \mathrm{res}_{q_i,q_{i+1}}(x_i)$ and $x_0 \in \mathrm{res}_{q_l,q_0}(x_l)$, but we refrain from doing so, as the technical complications would obscure the techniques at the heart of our analysis.

4 Denotational Semantics of Continuous and Separated Automata

We now turn to the main objective of the present paper and describe a computational method for obtaining R_F for a continuous and separated flow automaton F. Our technique will compute the function R_F as least fixpoint of a functional of type $([0,\infty) \Rightarrow \mathcal{U}) \to ([0,\infty) \Rightarrow \mathcal{U})$, where $\mathcal{U} = \prod_{q\in Q} \mathbf{U}^\top \mathrm{inv}(q)$. We first introduce some terminology to make the notation more readable.

Definition 5. *Suppose $F = (Q, \mathrm{inv}, \mathrm{flow}, \mathrm{res}, \mathrm{init})$ is a flow automaton. The function*

$$f_F : \mathcal{U} \times \mathbf{I}[0,\infty) \to \mathcal{U},$$
$$((x_q)_{q\in Q}, \alpha) \mapsto (\{\mathrm{flow}(q)(y_q, t) \mid y_q \in x_q, t \in \alpha, \forall s \leq t.\mathrm{flow}(q)(y_q, s) \in \mathrm{inv}(q)\})_{q\in Q}$$

is called the extended flow function*, and*

$$r_F : \mathcal{U} \to \mathcal{U}, (x_q)_{q\in Q} \mapsto (\bigcup_{p\in Q} \mathcal{E}(\mathrm{res}(p, q))(x_p))_{q\in Q}$$

is the extended reset function. If the automaton F is clear from the context, we omit the corresponding subscript.

While the extended reset function collects all the functions $\mathrm{res}(p, q)$ in a single map, the rationale behind the definition of the extended flow function is moreover that we need to cut out those portions of the flows that leave or re-enter a state invariant. Pictorially, this leaves us with the shaded region displayed in Figure 1. It is easy to see that both

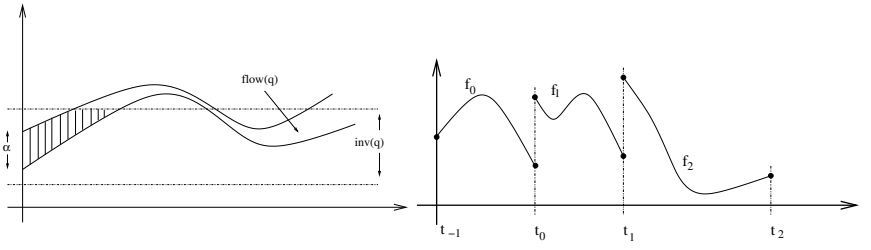


Fig. 1. The functions $f_F(\alpha, \cdot)$ (left) and $\rho^\sharp$ (right)

the extended flow function, and the extended reset function are Scott continuous.

Lemma 4. *If F is continuous, then both f_F and r_F are Scott continuous.*

With this notation, we are now ready to introduce the key concept of the present paper: the forward action associated with a flow automaton. As we will see later, the least fixpoint of this operator captures the set of of states the automaton can engage in at time t and, moreover, can be effectively computed.

Definition 6. *Suppose F is a flow automaton. The operator*

$$\Phi_F : ([0,\infty) \Rightarrow \mathcal{U}) \to ([0,\infty) \Rightarrow \mathcal{U}), \rho \mapsto \lambda t. f_F(i_F, t) \cup \bigcup_{s \leq t} f_F(r_F(\rho(s), t-s))$$

is called the forward action *associated with F.*

The forward action combines the discrete action and the continuous flow, and can be seen as a generalisation of the fixpoint operator associated with an IFS [6]. Our goal is to show that the least fixpoint of the forward action is precisely the function R_F that computes reachable states. In order to compute this fixpoint effectively, we first have to ensure that Φ_F is compatible with approximations, i.e. Φ_F is well-defined and Scott continuous.

Lemma 5. *Both $\Phi_F(\rho)$, for $\rho \in ([0,\infty) \Rightarrow \mathcal{U})$, and Φ_F are Scott continuous.*

Continuity of Φ_F now guarantees the existence of a least fixpoint of Φ_F, which we denote by $[\![F]\!]$ throughout. We now examine this fixpoint and show that it precisely captures the set of all F-trajectories.

In order to show soundness, it is convenient to formulate trajectories as maps into the upper space. In order to turn the trajectories into Scott continuous functions, we let the induced function take a non-singleton set as value whenever the discrete control state changes. Below, $\mathrm{sgn}(x) \in \{-1, 0, +1\}$ is the sign of $x \in \mathbb{R}$.

Lemma 6. *Suppose $f_{-1} : [-1,0] \to \mathbb{R}$ and $f_{+1} : [0,1] \to \mathbb{R}$ are continuous. Then the function $f \oplus g : [-1,1] \to \mathbf{U}^\top\mathbb{R}$, defined by $t \mapsto \{f_{\mathrm{sgn}(t)}(t)\}$, if $t \neq 0$, and $t \mapsto \{f(0), g(0)\}$ otherwise, is Scott continuous.*

For F-trajectories, we have the following corollary. Note that the condition on trajectories is automatic for continuous and separated automata.

Corollary 7. *Suppose F is a flow automaton $\rho = (t_i, q_i, f_i)_{i<N}$ is a F-trajectory with $\sup_i t_i = \infty$ in case $N = \infty$. Then*

$$\rho^\sharp : [0,\infty) \to \mathcal{U}, \quad t \mapsto \{(q_i, f_i(t)) \mid t \in [t_{i-1}, t_i]\}$$

where $q \in Q$, is Scott-continuous. Moreover, $R_F(t) = \bigcup\{\rho^\sharp(t) \mid \rho \text{ is an } F-\text{trajectory}\}$, if F is a flow automaton.

The function $\rho^\sharp$ is visualised on the right hand side of Figure 1. The next statement is a stepping stone for proving the soundness of our approach. We show, that applying Φ_F, we do not lose any trajectories; hence starting the fixpoint iteration from the everywhere undefined function, the least fixpoint is guaranteed to cover all trajectories.

Lemma 8. *Suppose F is separated and continuous, ρ is an F-trajectory and $\sigma \in ([0,\infty) \Rightarrow \mathcal{U})$ with $\sigma \sqsubseteq \rho^\sharp$. Then $\Phi_F(\sigma) \sqsubseteq \rho^\sharp$.*

Note that the proof of the previous theorem relies on separatedness. Using the above result, correctness of the fixpoint construction is very easy to show.

Corollary 9 (Correctness). *Suppose F is continuous and separated. Then $s \in [\![F]\!](t)$ if $\mathrm{init} \to_*^t s$ for all $s \in S_F$ and all $t \in [0,\infty)$.*

While the previous result asserts soundness, we now turn to computational adequacy of the construction, i.e. we show that $R_F = [\![F]\!]$, where $[\![F]\!]$ is the least fixpoint of Φ_F.

Theorem 10 (Computational Adequacy). *Suppose F is separated and continuous. Then $s \in [\![F]\!](t)$ iff $\mathrm{init} \to_*^t s$ for all $s \in S_F$ and all $t \geq 0$.*

The proof of the theorem in fact demonstrates, that any function $\rho \in ([0,\infty) \Rightarrow \mathcal{U})$ with $\rho \sqsubseteq [\![F]\!]$, that does not arise as an F-trajectory, necessarily leads to a violation of the separatedness property. Unfolding the definition of V_F, we also obtain computational means to obtain the states of a flow automaton F that can be visited up to time t, in terms of the least fixpoint $[\![F]\!]$ of the forward action associated with F. This then gives $V_F(t) = \bigcup_{s \leq t} R_F(s)$. However, we can also obtain V_F as a fixpoint of an operator in its own right.

Definition 7. *The operator*

$$\Psi_F : ([0,\infty) \Rightarrow \mathcal{U}) \to ([0,\infty) \Rightarrow \mathcal{U}), \rho \mapsto f_F(i_F, [0,t]) \cup \bigcup_{s \leq t} f_F(r_F(\rho(s), [0, t-s]))$$

is the visited states operator *associated with F.*

The properties of Ψ_F are similar to those of Φ_F, in particular, Ψ_F is Scott continuous, and the least fixpoint captures the set of visited states. More formally:

Theorem 11. *Suppose $\rho : [0,\infty) \to \mathcal{U}$ is the least fixpoint of Ψ_F. Then $\rho = V_F$.*

While Theorem 10 and Theorem 11 are important on their own, as they allow us to obtain the semantics of hybrid automata as a least fixpoint in a suitable function space, they also allow us to derive new results about the function R_F that yields the states reachable at time t for continuous and separated automata:

Corollary 12. *1. $R_F(t)$ and $V_F(t)$ are compact for every $t \in [0,\infty)$.*
2. R_F and V_F are Scott continuous.

5 Approximation of Flow Automata

We have seen that the semantics $[\![F]\!] : [0,\infty) \to \mathcal{U}$ of a flow automaton F can be computed as the least fixpoint of a functional on $([0,\infty) \Rightarrow \mathcal{U})$. While this gives a mathematical means of understanding the semantics, we now show, that this also induces a method to compute the semantics up to an arbitrary degree of accuracy.

To do this, we restrict attention to countable bases of the involved domains, that is, to finitely representable objects, that generate all of the involved domains by means of directed suprema. We show, that we can effectively compute the least fixpoint of the functional up to an arbitrary degree of accuracy, if we approximate all continuous ingredients of the automaton. We begin by introducing the bases of the domains we are interested in. For the remainder of the section, we fix a countable dense ordered subring $D = \{d_0, d_1, \dots\}$ with decidable equality and order, and computable ring operations. We put $D_k = \{d_0, \dots, d_k\}$. We only treat the case of computing R_F as a least fixpoint; the setup can be easily adapted to accommodate also V_F.

Definition 8. *We let, for an arbitrary set $S \subseteq \mathbb{R}$, $\mathbf{I}\mathbb{R}^n_S = \{[a_1,b_1] \times \cdots \times [a_n,b_n] \in \mathbf{I}\mathbb{R}^n \mid a_1,\dots,a_n,b_1,\dots,b_n \in S\} \cup \{\mathbb{R}\}$ denote the set of rectangles with endpoints in S, augmented with the least element $\mathbb{R}$. If $\mathsf{A} \subseteq \mathbb{R}^n$ is a semi rectangle, then $\mathbf{I}\mathsf{A}_S = \{\mathsf{A} \cap b \mid b \in \mathbf{I}\mathbb{R}^n_S\}$ denotes the set of rectangles $R \in \mathbf{I}\mathbb{R}^n$ that are contained in A and have corners in S, again with a bottom element. We distinguish two kinds of step functions:*

$$a \searrow^i b : \mathsf{A} \to \mathsf{B}, x \mapsto \begin{cases} b & x \in a^o \\ \bot & \textit{otherwise}, \end{cases} \quad \textit{and} \quad a \searrow b : \mathsf{A} \to \mathsf{B}, x \mapsto \begin{cases} b & a \ll x \\ \bot & \textit{otherwise} \end{cases}$$

where B is a dcpo with $b \in \mathsf{B}$ in both cases; $\mathsf{A} \subseteq \mathbb{R}^n$ is a semi rectangle with $a \in \mathbf{I}\mathsf{A}$ in the case of $a \searrow^i b$, and A is a dcpo with $a \in \mathsf{A}$ for $a \searrow b$. We use the following bases:

1. *If $\mathsf{A} \subseteq \mathbb{R}^n$ is a semi-rectangle with corners in $D \cup \{\pm\infty\}$, then the set $\mathbf{I}\mathsf{A}_D$ of rectangles contained in A and corners in D is called the* standard base *of $\mathbf{I}\mathbb{R}^n$.*
2. *If $\mathsf{A} \in \mathbf{I}\mathbb{R}^n_D$, then the set $\mathbf{U}^\top\mathsf{A}_D = \{\cup_{1\le i\le k} D_i \mid i \in \mathbb{N}, D_i \in \mathbf{I}\mathsf{A}_D\}$ of finite unions of rectangles with corners in D is the* rectangular base *of $\mathbf{U}^\top\mathsf{A}$.*
3. *If $(A_i)_D$ is a base of the dcpo A_i, then $(A_1 \times \cdots \times A_n)_D = (A_1)_D \times \cdots \times (A_k)_D$ is the base of $A_1 \times \cdots \times A_k$ induced by the components.*
4. *If A_D and B_D are bases of the dcpos A and B, respectively, then $(A \Rightarrow \mathsf{B})_D = \{\bigsqcup_{i\le i\le k} a_i \searrow b_i \in (\mathsf{A} \Rightarrow \mathsf{B}) \mid a_1,\dots,a_k \in \mathsf{A}_D, b_1,\dots b_k \in \mathsf{B}_D\}$ is the rectangular base of $(\mathsf{A} \Rightarrow \mathsf{B})$.*

5. *Finally, if* $\mathsf{A} \subseteq \mathbb{R}^n$ *is a semi rectangle with corners in* $D \cup \{\pm\infty\}$ *and* B_D *is a base of the dcpo* B*, then* $(\mathsf{A} \Rightarrow \mathsf{B})_D = \{\bigsqcup_{1\leq i\leq k} a_i \searrow^i b_i \in (\mathsf{A} \Rightarrow \mathsf{B}) \mid a_1, \ldots, a_k \in \mathbf{I}\mathsf{A}_D, b_1, \ldots, b_k \in \mathsf{B}_D$ *is the induced base of of* $(\mathsf{A} \Rightarrow \mathsf{B})$

where we indicate by $\bigsqcup_{1\leq i\leq k} a_i \searrow b_i \in (\mathsf{A} \Rightarrow \mathsf{B})$ *that we consider only consider consistent step functions [8, Section 2], similarly for* $\bigsqcup_{1\leq i\leq k} a_i \searrow^i b_i$.

In words, if $\mathsf{A}, \mathsf{B} \subseteq \mathbb{R}^n$ are semi-rectangles, $\mathbf{I}\mathsf{A}_D$ is the set of rectangles contained in A with corners in D and $\mathbf{U}^\top\mathsf{A}_D$ is the set of finite unions of rectangles with corners in D. For the function space, $(\mathsf{A} \Rightarrow \mathbf{U}^\top\mathsf{B})_D$ is the induced base of the space of functions of one or more real variables; $(\mathbf{U}^\top\mathsf{A} \Rightarrow \mathbf{U}^\top\mathsf{B})_D$ is the induced base of the function space of a compact set valued variable.

It is easy to see that the sets introduced above are indeed bases of the corresponding domain. We now use these bases to show, that the fixpoint operator Φ_F associated to a flow automaton can be effectively computed, given approximations of the components of the automaton. In order to make assertions about the computability of functions in the domain theoretic model of computation, we have to fix an enumeration of the base of the involved domains. We do not do this explicitely here, but instead assume that the bases $(\cdot)_D$ above come with an effective enumeration $\iota : \mathbb{N} \to (\cdot)_D$, which is fix throughout. In particular, the enumeration gives rise to a notion of *effective sequence*: If A is a dcpo whose base is enumerated via $\iota : \mathbb{N} \to \mathsf{A}_D$, then a sequence $(a_k)_{k\in\mathbb{N}}$ in A_D is *effective*, if $a_k = \iota(f(k))$ for some total recursive function f.

First, note that composition of base functions yields a base function, and that the extension function is effectively computable.

Lemma 13. *Suppose* $f \in (\mathsf{A} \Rightarrow \mathbf{U}^\top\mathsf{B})_D$ *and* $g \in (\mathbf{U}^\top\mathsf{B} \Rightarrow \mathbf{U}^\top\mathsf{C})_D$*. Then*

1. $g \circ f \in (\mathsf{A} \Rightarrow \mathbf{U}^\top\mathsf{C})_D$ *and* $g \circ f$ *is effectively computable.*
2. $\mathcal{E}(f) \in (\mathbf{U}^\top\mathsf{A} \Rightarrow \mathbf{U}^\top\mathsf{B})_D$ *and* $\mathcal{E}(f)$ *is effectively computable.*

The next lemma gives a basis representation of subtraction, which is needed in the definition of the fixpoint functional Φ_F associated with F, see Definition 6.

Lemma 14. *The functions* $M_k : [0,\infty)^2 \to \mathbf{I}[0,\infty)$*, defined by* $M_k = \bigsqcup\{a \times b \searrow b - a \mid a, b \in \mathbf{I}[0,\infty)_{D_k}\}$ *satisfy* $M_k \in ([0,\infty)^2 \Rightarrow \mathbf{I}[0,\infty))_D$ *for all* $k \in \mathbb{N}$*, and* $\bigsqcup_{k\in\mathbb{N}} M_k = \lambda(x,y).y - x$.

Building on these basic facts, we can now show, that the least fixpoint of the operator Φ_F associated with a flow automaton is effectively computable. This of course hinges on the fact that the automaton is effectively given:

Definition 9. *Suppose* $F = (Q, \mathrm{init}, \mathrm{flow}, \mathrm{res}, \mathrm{inv})$ *is a flow automaton. We say that* F *is* effectively given *if it comes with*

- *an effective sequence* $(i_k^q)_{k\in\mathbb{N}}$ *in* $(\mathbf{U}^\top\mathrm{inv}(q))_D$ *with* $\bigsqcup_{k\in\mathbb{N}} i_k^q = \mathrm{init}(q)$
- *an effective sequence* $(f_k^q)_{k\in\mathbb{N}}$ *in* $(\mathbb{R}^n \times [0,\infty) \Rightarrow \mathbf{I}\mathbb{R}^n)_D$ *with* $\bigsqcup_{k\in\mathbb{N}} f_k^q = \mathrm{flow}(q)$
- *an effective sequence* $(r_k^{p,q})_{k\in\mathbb{N}}$ *in* $(\mathrm{inv}(p) \Rightarrow \mathbf{U}^\top\mathrm{inv}(q))_D$ *with* $\bigsqcup_k r_k^{p,q} = \mathrm{res}(p,q)$

for all $q \in Q$, resp. all $(p,q) \in Q^2$ and $\mathrm{inv}(q) \in \mathbf{U}^\top \mathbb{R}^n_D$ for all $q \in Q$. The family of sequences $(f^q_k)_{k\in\mathbb{N}}$, $(i^q_k)_{k\in\mathbb{N}}$ (where $q \in Q$) and $(r^{p,q}_k)_{k\in\mathbb{N}}$ (where $(p,q) \in Q^2$) are called an effective presentation *of F.*

That is to say that, for an effectively given flow automaton, the initial states, the flow functions and the reset functions are computable. It is easy to see that every effectively given flow automaton induces a computable extended flow function, and a computable extended reset function. Spelling this out, Lemma 13 and Lemma 14, together with the Scott continuity of Φ_F, allow us to prove our second main theorem:

Theorem 15. *Suppose F is an effectively given flow automaton. Then we can effectively obtain a sequence (σ_k) with $\bigsqcup_{k\in\mathbb{N}} \sigma_k = [\![F]\!]$.*

6 Hybrid Automata

In this section, we transfer our results on flow automata to hybrid systems, where the continuous behaviour of the system in every given control state is described directly by a vector field. This is achieved by associating the equivalent flow automaton to the hybrid automaton under consideration. If the hybrid automaton is effectively given, we show, that the same also holds for the induced flow automaton. We thus obtain an effective framework for the analysis of hybrid automata. The following is a variant of the standard definition of a hybrid automaton [12, 2].

Definition 10. *A* hybrid automaton *is a tuple $H = (Q, \mathrm{inv}, \mathrm{vect}, \mathrm{res}, \mathrm{init})$ where $Q, \mathrm{inv}, \mathrm{res}, \mathrm{init}$ are as in Definition 2, and $\mathrm{vect} = (\mathrm{vect}_q)_{q\in Q}$ is a family of vector fields $\mathrm{vect}(q) : \mathbb{R}^n \to \mathbb{R}^n$ where all $\mathrm{vect}(q)$ are assumed to be globally Lipschitz, i.e. $\|\mathrm{vect}(q)(x) - \mathrm{vect}(q)(y)\| \leq L\|x - y\|$, for all $q \in Q$, $x, y \in \mathbb{R}^n$ and some $L \in \mathbb{R}$.*

In contrast to the standard definition, the trajectories of the real variables are described by a differential equation rather than differential inclusion. We require this restriction in view of the domain theoretic treatment of differential equations [9], which in general gives a strict over-approximation to the solution of differential inclusion.

We recall from Lemma 1, that every Lipschitz vector field $v : \mathbb{R}^n \to \mathbb{R}^n$ induces a flow function $f : \mathbb{R}^n \times [0,\infty) \to \mathbb{R}^n$. The main reason for restricting attention to vector fields that are globally Lipschitz is that the induced flows are globally defined; we believe that similar results can be obtained for vector fields whose associated flows don't diverge. Replacing the vector field by the induced flow function, every hybrid automaton H induces a flow automaton F; in this case, we write $[\![H]\!]$ for $[\![F]\!]$.

Definition 11. *Suppose $H = (Q, \mathrm{inv}, \mathrm{vect}, \mathrm{res}, \mathrm{init})$ is a hybrid automaton and $\mathrm{flow}(q)$ is the flow induced by $\mathrm{vect}(q)$. The automaton $F = (Q, \mathrm{inv}, \mathrm{flow}, \mathrm{res}, \mathrm{init})$ is called the* flow automaton induced by H. *We say that H is continuous (resp. separated), if the induced flow automaton is continuous (resp. separated). We say that H is* effectively given *if it comes with*

- *an effective sequence $(i^q_k)_{k\in\mathbb{N}}$ in $(\mathbf{U}^\top \mathrm{inv}(q))_D$ with $\bigsqcup_{k\in\mathbb{N}} i^q_k = \mathrm{init}(q)$*
- *an effective sequence $(v^q_k)_{k\in\mathbb{N}}$ in $(\mathbf{I}\mathbb{R}^n \Rightarrow \mathbf{I}\mathbb{R}^n)_D$ with $\bigsqcup_{k\in\mathbb{N}} v^q_k = \mathrm{vect}(q)$*

– *an effective sequence* $(r_k^{p,q})_{k\in\mathbb{N}}$ *in* $(\mathrm{inv}(p) \Rightarrow \mathbf{U}^\top \mathrm{inv}(q))_D$ *with* $\bigsqcup_k r_k^{p,q} = \mathrm{res}(p,q)$

for all $q \in Q$*, resp. all* $(p,q) \in Q^2$ *and* $\mathrm{inv}(q) \in \mathbf{U}^\top \mathbb{R}^n_D$ *for all* $q \in Q$*. The family of sequences* $(i_k^q)_{k\in\mathbb{N}}$*,* $(v_k^q)_{k\in\mathbb{N}}$ *(where* $q \in Q$*) and* $(r_k^{p,q})_{k\in\mathbb{N}}$ *(where* $(p,q) \in Q^2$*) are called an* effective presentation *of* H*.*

We have seen in Theorem 15, that the function $[\![F]\!]$ associated with a flow automaton, which captures the states reachable by F at time $t \in [0,\infty)$, is effectively computable, if F is effectively given. In order to associate an effectively given flow automaton to an effectively given hybrid automaton, we therefore have to produce approximations $f_k \in (\prod_{q\in Q} \mathrm{inv}(q) \times [0,\infty) \Rightarrow \mathcal{U})$ of the flow function induced by a hybrid automaton. In other words, we have to solve the initial value problems defined by the vector field that defines the hybrid automation. This is achieved by instantiating results from [9, 10], where it is shown how to solve initial value problems in a domain theoretic framework.

Theorem 16. *Suppose* H *is an effectively given hybrid automaton. Then so is the induced flow automaton* F*. Moreover, we can construct an effective presentation of* F *from an effective presentation of* H*.*

Together with Theorem 15, we have now shown, that the semantic function $[\![H]\!]$, associated with an effectively given hybrid automaton, is computable:

Theorem 17. *Suppose* H *is effectively given, continuous and separated. Then the function* $[\![H]\!] : [0,\infty) \to \mathcal{U}$ *is effectively computable.*

Moreover, as all our constructions are based on bases of the domains involved, the algorithms underlying Theorems 17 and 15 are based on proper data types, and can be directly implemented on a digital computer: we choose the dyadic (or rational) numbers for D, and then define data types that directly represent the bases $[0,\infty)_D$ and $\mathcal{U}_D$, as well as the bases of the function space $([0,\infty) \Rightarrow \mathcal{U})_D$. Computing with dyadic (or rational) numbers then allows us to manipulate elements of the data types without any loss of arithmetical precision. Moreover, we have shown that the fixpoint operator, that gives rise to the semantic function $[\![H]\!]$ of a hybrid automaton, can be effectively computed on the described data types.

Conclusions and Future Work. Of course, much remains to be done. While the presentation in this paper is geared towards demonstrating that a domain theory has all the necessary tools to facilitate the algorithmic analysis of hybrid automata, we anticipate that major improvements will be made on the efficiency of the involved algorithms. In particular, we are working towards rigorous estimates of the convergence speed and the complexity of the described fixpoint algorithms in terms of the Hausdorff distance in $\mathcal{U}$, which will also allow us to make concrete assertions about the computational complexity of our method. For now, we have concentrated on computing the semantic function $[\![H]\!]$ associated with a hybrid automaton. Future work will bring a framework for computing the set of reachable states of a hybrid automaton, and a real time logic with associated model checking procedure for the automated verification of hybrid automata.

References

1. S. Abramsky and A. Jung. Domain Theory. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3. Clarendon Press, 1994.
2. R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoret. Comp. Sci*, 138(1):3–34, 1995.
3. R. Alur, T. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.
4. J.-P. Aubin. *Viability Theory*. Birkhäuser, 1991.
5. A. Edalat. Dynamical systems, measures and fractals via domain theory. *Information and Computation*, 120(1):32–48, 1995.
6. A. Edalat. Power domains and iterated function systems. *Information and Computation*, 124:182–197, 1996.
7. A. Edalat, M Krznarić, and A. Lieutier. Domain-theoretic solution of differential equations (scalar fields). In *Proceedings of MFPS XIX*, volume 83 of *Elect. Notes in Theoret. Comput. Sci.*, 2004.
8. A. Edalat and A. Lieutier. Domain theory and differential calculus (functions of one variable. *Math. Struct. Comp. Sci.*, 14, 2004.
9. A. Edalat and D. Pattinson. A domain theoretic account of picard's theorem. In *Proc. ICALP 2004*, number 3142 in Lect. Notes in Comp. Sci., pages 494–505, 2004.
10. A. Edalat and D. Pattinson. Domain theoretic solutions of initial value problems for unbounded vector fields. In M. Escardó, editor, *Proc. MFPS XXI*, Electr. Notes in Theoret. Comp. Sci., 2005. to appear.
11. G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. Scott. *Continuous Lattices and Domains*. Cambridge University Press, 2003.
12. T. Henzinger. The theory of hybrid automata. In M. Inan and R. Kurshan, editors, *Verification of Digital and Hybrid Systems*, volume 170 of *NATO ASI Series F: Computer and Systems Sciences*, pages 265–292. Springer Verlag, 2000.
13. T. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.
14. T. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43:540–554, 1998.
15. T. Henzinger, B. Horowitz, R. Majumdar, and H. Wong-Toi. Beyond HYTECH: Hybrid systems analysis using interval numerical methods. In *Proc. HSCC 2000*, volume 1790 of *Lect. Notes in Comp. Sci.*, pages 130–144. Springer, 2000.
16. J. E. Hutchinson. Fractals and self-similarity. *Indiana University Mathematics Journal*, 30:713–747, 1981.
17. J. Lygeros, D. Godbole, and S. Sastry. Verified hybrid controllers for automated vehicles. *IEEE Transactions on Automatic Control*, 43(4):522–539, 1998.
18. O. Müller and T. Stauner. Modelling and verification using linear hybrid automata – a case study. *Mathematical and Computer Modelling of Dynamical Systems*, 6(1):71–89, 2000.
19. S. Simic, K. Johansson, S. Sastry, and J. Lygeros. Towards a geometric theory of hybrid systems. In N. Lynch and B. Krogh, editors, *Proc. HSCC 2000*, volume 1790 of *Lect. Notes in Comp. Sci.*, pages 421–436, 2000.
20. C. Tomlin, G. Pappas, and S. Sastry. Conflict resolution for air traffic management : A study in muti-agent hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):509–521, 1998.
21. K. Weihrauch. *Computable Analysis*. Springer, 2000.