A computable and compositional semantics for hybrid automata

A computable and compositional semantics for hybrid automata

Davide Bresolin
Università di Padova
Padova, Italy
davide.bresolin@unipd.it

Pieter Collins
University of Maastricht
Maastricht, The Netherlands
pieter.collins@maastrichtuniversity.nl

Luca Geretti
Università di Verona
Verona, Italy
luca.geretti@univr.it

Roberto Segala
Università di Verona
Verona, Italy
roberto.segala@univr.it

Tiziano Villa
Università di Verona
Verona, Italy
tiziano.villa@univr.it

Sanja Živanović Gonzalez
Barry University
Miami Shores, Florida
SZivanovic@barry.edu

ABSTRACT

Hybrid Systems are systems having a mixed discrete and continuous behaviour that cannot be characterized faithfully using either only discrete or only continuous models. A good framework for hybrid systems should support their *compositional* description and analysis, since commonly systems are specified by a composition of smaller subsystems, to cope with the complexity of their monolithic representation. Moreover, since the reachability problem for hybrid systems is undecidable, one should investigate the conditions that guarantee *approximate* computability of composition, when only approximations to the exact problem data are available.

In this paper, we propose an automata-based formalism (HIOA) for hybrid systems that is compositional and for which the evolution can be computed approximately. The main results are that the composition of compatible HIOA yields a pre-HIOA; a dominance result on the composition of HIOA by which we can replace any component in a composition by another one that exhibits the same external behaviour without affecting the behaviour of the composition; finally, the key result that the composition of two compatible upper(lower)-semicontinuous HIOA is a computable upper(lower)-semicontinuous pre-HIOA, which entails that the evolution of the composition is upper(lower)-semicomputable. A discussion on how compositionality/computability are handled in state-of-art libraries for reachability analysis closes the paper.

CCS CONCEPTS

• **Theory of computation → Timed and hybrid models**; **Computability**.

KEYWORDS

Hybrid Automata, Composition, Computability

ACM Reference Format:
Davide Bresolin, Pieter Collins, Luca Geretti, Roberto Segala, Tiziano Villa, and Sanja Živanović Gonzalez. 2020. A computable and compositional semantics for hybrid automata. In *23rd ACM International Conference on Hybrid Systems: Computation and Control (HSCC '20), April 22–24, 2020, Sydney, NSW, Australia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3365365.3382202

HSCC '20, April 22–24, 2020, Sydney, NSW, Australia

ACM ISBN 978-1-4503-7018-9/20/04…$15.00
https://doi.org/10.1145/3365365.3382202

1 INTRODUCTION

Hybrid Systems are systems having a mixed discrete and continuous behavior that cannot be characterized faithfully using either discrete or continuous models only. Such systems typically consist of a discrete control part that operates in a continuous environment, and are used in many application domains, like automotive, robotics, avionics, autonomous vehicles, process control, real-time and mobile computing systems [30].

Since they are usually made of different components, like controllers, sensors, actuators, computer software, communication devices, etc., interacting in a complex environment, they can be very difficult to describe in a faithful way. This problem motivates the need for rigorous mathematical models and algorithms to describe and analyze hybrid systems.

A good framework for hybrid systems should support the *compositional* description and analysis of systems, since they can be too complex to be understood all at once and must be decomposed. A key of this decomposition is that the framework includes a notion of *external behavior* of a component, describing the continuous and discrete interactions with the environment, and that of *composition* and *abstraction* of different components. Composition defines how components are put together to make complex systems out of simpler ones. Abstractions are used both to view the system with different levels of detail, starting from a high-level abstraction of the system with only some relevant properties down to a low-level description with all the implementation details, and to show which properties of the system are preserved between the different levels. The most relevant compositional frameworks and languages for hybrid systems proposed in the literature are Hybrid Reactive Modules [3], Hybrid I/O Automata [28], and the languages Charon [2], Masaccio [23], HyDI [10], HRELTL [11], the Metropolis Metamodel Interchange Format [31], and the Compositional Interchange Format [1, 37]. Heterogeneous modeling frameworks like Ptolemy [36] are also relevant, since they support hybrid systems by combining continuous-time models with discrete logic. Compositional frameworks are useful not only to build complex systems from a network of simple components, but also to simplify the treatment of monolithic systems [8]. A recent example is [7], where abstraction and compositionality are exploited to approximate the

dynamics of nonlinear hybrid systems with linear relations that can be handled by tools for the verification of discrete-time systems.

However, while all these formalisms address the system decomposition issue, and solve it with different approaches, they pose little attention to the problem of the algorithmic analysis of hybrid systems. It is well known that the reachability relation for hybrid systems is undecidable, except for the class of timed automata and some generalizations [24]. Since several questions about the behavior of a hybrid system can be rephrased in terms of reachable sets and reachability relations, many papers propose approximation techniques to estimate the reachable set [4, 6, 17, 22, 26, 34]. Unfortunately, even the computation of approximations to the reachable set is not straightforward; indeed, it may not even be possible to compute a sequence of over-approximations convergent to the reachable set [12].

Other approaches introduce a notion of robustness in the interpretation of the verification problem, that can capture the distance ε from unsatisfiability of a given property [19, 20]. Robustness allows one to solve the problem of testing temporal logic formulas for continuous-time systems [20] and piecewise linear systems [19] using standard discrete-time analysis. However, it is known that it is not sufficient to recover decidability for arbitrary hybrid systems [25]. The works in [19, 20] share three main differences with the approach described in this paper. First, they focus on verification problems for temporal logic formulas, while we are interested in computing the evolution of a hybrid system. Second, they are not compositional: the system under verification is assumed to be monolithic, while we allow the composition of smaller components into a larger system. Finally, and most importantly, they do not explicitly discuss computation, but give sufficient conditions to reduce the continuous or hybrid system under verification into a discrete-time system, whereas our approach gives conditions under which the evolution of a system can be computationally approximated with an arbitrary small and known accuracy. A technical difference between our approach and [20] is that they require uniform closeness of trajectories over an infinite time domain, whereas our convergence notion is uniform on compact subdomains only; global uniform accuracy is sensitive to arbitrarily small perturbations and cannot be computed.

This leads to some fundamental questions that guide the focus of our paper. How can we compose systems into larger objects that are amenable to algorithmic analysis? What restrictions can we impose on components to ensure that their composition can be analyzed? What are some minimal requirements to ensure compositionality of the ability to analyze a system? In this paper, we base our computability results on the theory of *computable analysis* [38] by Weihrauch and co-workers, and extend the results of [13, 16], which describe how to solve approximately the reachability problem for systems specified as single monolithic hybrid automata. We focus on the problem of computing the set of behaviors of a hybrid system, and choose to use the model of Hybrid I/O Automata [28] (HIOAs) because it can be used as a semantic framework for several existing formalisms and is equipped already with a theory for compositional analysis; we restrict the non-discrete dynamics of HIOAs to be continuous solutions of differential equations, thus enabling the use of the theory of computable functions of [13, 16, 38], and then define appropriate topologies on the set of hybrid traces of a HIOA to lead to the notions of upper and lower semicontinuous automata, for which the set of hybrid traces are upper and lower semicomputable, respectively. As a consequence we have a formalism for non-linear hybrid systems over which the reachability problem is upper or lower approximable. We then show that upper and lower semicontinuity is preserved by parallel composition (Theorem 5.10). Thus, for instance, if we work with dynamics expressed in terms of differential equations based on locally Lipschitz functions with linear growth, we can safely build complex automata that are amenable to compositional analysis and that are upper or lower semicontinuous, thus supporting upper or lower approximations for problems like reachability analysis. Along the way we show how to build the best upper or lower semicontinuous approximation of a HIOA with dynamics expressed by locally Lipschitz functions with linear growth.

In summary we tackle the problem of building compositionally large computable hybrid systems by combining two existing theories: the theory of computable functions of Weihrauch and co-workers, which is used already within tools like Ariadne, and the theory of Hybrid I/O Automata, which is used already successfully and shows how to build complex hybrid systems amenable to compositional analysis once the underlying continuous dynamics satisfy appropriate restrictions. We focus on minimizing the modifications to both of the existing theories so that the enhancements due to compositional computability do not hinder any of their features. Therefore, the HIOA automata model of this paper is exactly the original HIOA automata model, with the very same definition of parallel composition, where we impose that continuous dynamics are described by means of differential equations. This allows us to impose restrictions in the style of the theory of computable analysis and verify that they satisfy the appropriate conditions. We also use stuttering actions, that appear in some variants of HIOAs, to identify the points where continuous dynamics change due to external discrete transitions. This is important to enable computability. Notice that in this paper we focus mainly on the theoretical aspects of the problem. Future work includes the demonstration of this proposed framework within the library Ariadne; yet, our general approach is meant to be independent of the actual underlying models, and thus we aim to see it applied to other frameworks as well.

The rest of the paper is organized as follows. Section 2 gives some preliminary notions and notations. Section 3 introduces the formalism of HIOAs, with dynamic behavior constrained to be described by differential equations, and its compositional results. Section 4 gives a brief descriptions of the known results from computability analysis that are needed in this paper. Section 5 presents our computability results for HIOAs. Section 6 describes the progress in the implementation of the results of this paper within the tool Ariadne and compares the resulting framework with other existing tools. Section 7 contains some concluding remarks.

2 PRELIMINARY NOTIONS

Throughout the paper we fix the *time axis* to be the set of non-negative real numbers $\mathbb{R}^+$. An *interval* I is any convex subset of $\mathbb{R}^+$, denoted as usual with $[t_1, t_2]$, (t_1, t_2), $[t_1, t_2)$, and $(t_1, t_2]$, where a parenthesis corresponds to an open endpoint and a square bracket

to a closed endpoint. For any interval I and $t \in \mathbb{R}^+$, we define $I + t$ as the interval $\{t' + t : t' \in I\}$.

We also fix a countable universal set $\mathcal{V}$ of *variables*, where every variable $v \in \mathcal{V}$ has a type $\mathrm{Type}(v)$ which defines the domain over which the variable ranges. Elementary types include *booleans*, *integers* and *reals*. A type is *discrete* if it has a decidable equality. Given a set of variables $V \subseteq \mathcal{V}$, a *valuation* over V is a function that associates every variable in V with a value in its type. We often refer to valuations as *states*, and we denote them as $\mathbf{x}, \mathbf{y}, \mathbf{z}, \ldots$. The set $\mathrm{Val}(X)$ is the set of all valuations over X. Given a valuation $\mathbf{x}$ and a subset of variables $Y \subseteq X$, we denote the *restriction* of $\mathbf{x}$ to Y as $\mathbf{x}|Y$. The restriction operator is extended to sets of valuations in the usual way. Symmetrically, given a set of valuations S and a superset of variables $Z \supseteq X$, the *extension* of S to Z is denoted as $S \uparrow Z$ and defined as the set $\{\mathbf{z} \in \mathrm{Val}(Z) : \mathbf{z}|X \in S\}$.

A notion that plays an important role in the paper is the one of *continuous trajectory*. A continuous trajectory over a set of variables X is a continuous function $\tau : I \mapsto \mathrm{Val}(X)$, where I is a left-closed interval with left endpoint equal to 0. In the following we call a continuous trajectory over X simply an "X-trajectory". With $\mathrm{dom}(\tau)$ we denote the domain of τ, while with $\mathrm{lasttime}(\tau)$ (the *limit time* of τ) we define the supremum of $\mathrm{dom}(\tau)$. The *first state* of a trajectory is $\mathrm{firststate}(\tau) = \tau(0)$, while, when $\mathrm{dom}(\tau)$ is right-closed, the *last state* of a trajectory is defined as $\mathrm{laststate}(\tau) = \tau(\mathrm{lasttime}(\tau))$. We denote with $\mathrm{Trajs}(X)$ the set of all trajectories over X. Given a subset $Y \subseteq X$, the *restriction* of τ to Y is denoted as $\tau|Y$ and it is defined as the trajectory $\tau' : \mathrm{dom}(\tau) \mapsto \mathrm{Val}(Y)$ such that $\tau'(t) = \tau(t)|Y$ for every $t \in \mathrm{dom}(\tau)$. With a little abuse of notation, if $I = [t, t']$ is a subinterval of $\mathrm{dom}(\tau)$, we denote with $\tau|I$ the trajectory τ' such that $\mathrm{dom}(\tau') = [0, t' - t]$ and $\tau'(t'') = \tau(t'' + t)$ for every $t'' \in \mathrm{dom}(\tau')$. The restriction operators are extended to sets of trajectories in the usual way.

A trajectory τ' is a *prefix* of another trajectory τ if and only if $\mathrm{lasttime}(\tau') \leqslant \mathrm{lasttime}(\tau)$ and $\tau'(t) = \tau(t)$ for every $t \in \mathrm{dom}(\tau')$. Conversely, we say that τ' is a *suffix* of τ if there exists $t \in \mathbb{R}^+$ such that $\mathrm{lasttime}(\tau') = \mathrm{lasttime}(\tau) - t$ and $\tau'(t') = \tau(t' + t)$ for every $t' \in \mathrm{dom}(\tau')$. Given two trajectories τ_1 and τ_2 such that $\mathrm{laststate}(\tau_1) = \mathrm{firststate}(\tau_2)$, their concatenation $\tau_1 \cdot \tau_2$ is the trajectory with domain $\mathrm{dom}(\tau_1) \cup (\mathrm{dom}(\tau_2) + \mathrm{lasttime}(\tau_1))$ such that $\tau_1 \cdot \tau_2(t) = \tau_1(t)$ if $t \in \mathrm{dom}(\tau_1)$, $\tau_1 \cdot \tau_2(t) = \tau_2(t - \mathrm{lasttime}(\tau_1))$ otherwise. We sometimes omit the $\cdot$ operator and write $\tau_1\tau_2$ to denote the concatenation $\tau_1 \cdot \tau_2$. Concatenation is extended to countable sequences of trajectories in the usual way.

3 A COMPOSITIONAL FORMALISM

The starting point of the discussion is the formalism of Hybrid I/O Automata (HIOAs) defined by Lynch, Segala, and Vandraager in [28, 29] and the definition of Hybrid System given in [14]. The key point of this section is to show how to revise the definition of HIOAs from [28] without losing the power of the model and yet enable computability of the reachable set as in [14].

More specifically, our definition of Hybrid Automata is based on the early definition in [29], which features the notion of *environment action* (representing the occurrence of an unobservable discrete transition outside the system, or a discontinuity in the input), later removed from [28] to simplify the formalism. However, the explicit presence of an environment action is needed to achieve computability of the formalism (as we will show in Section 5), and thus we choose to reintroduce it, at the price of adding some more axioms to deal with it. Comparing with [28], where transitions and trajectories are defined explicitly as sets, here we define continuous trajectories and discrete transitions implicitly in an operational way, where the former are solutions of differential equations and the latter are defined implicitly in term of guards and reset functions (so we do not need some related axioms needed in [28]); moreover, we do not introduce explicitly initial states (same as all states are initial) and define an invariant set. Notice that in our definition, as in [29], transitions affect internal and external variables, whereas in [28] they affect only internal variables.

Definition 3.1. *A* Hybrid Input/Output Automaton (HIOA) *is a tuple* $\mathcal{A} = \langle U, X, Y, I, H, O, Inv, Act, Res, Dyn \rangle$*, where:*

- *U, X, and Y are three finite sets of* input, internal, *and* output *variables, disjoint from each other. Variables in $Z = X \cup Y$ are called* locally controlled, *while variables in $W = U \cup Y$ are called* external. *We define $V = U \cup X \cup Y$;*
- *I, H, O are three finite sets of* input, internal, *and* output *actions, disjoint from each other. We assume that I contains a special action ε, which represents the* environment action. *Actions in $L = H \cup O$ are called* locally controlled, *while actions in $E = I \cup O$ are called* external. *We define $A = I \cup H \cup O$;*
- *$Act \subseteq A \times \mathrm{Val}(V)$ is the* activation set. *We say that an action $a \in A$ is* enabled *in $\mathbf{v} \in \mathrm{Val}(V)$ if $(a, \mathbf{v}) \in Act$. Act respects the following property:*
  - **A1** input action enabling*: for every $a \in I$ and every $\mathbf{v} \in \mathrm{Val}(V)$, $(a, \mathbf{v}) \in Act$;*
- *$Inv \subseteq \mathrm{Val}(Z)$ is the* invariant set*;*
- *$Res : A \times \mathrm{Val}(U) \times \mathrm{Val}(Z) \rightrightarrows \mathrm{Val}(Z)$ defines a* reset map[1] *respecting the following property:*
  - **R1** environment action does not change state: *for every $\mathbf{u} \in \mathrm{Val}(U)$ and $\mathbf{z} \in \mathrm{Val}(Z)$, $Res(\varepsilon, \mathbf{u}, \mathbf{z}) = \{\mathbf{z}\}$;*
  - **R2** internal actions do not change outputs: *for every $h \in H$, $\mathbf{u} \in \mathrm{Val}(U)$, $\mathbf{z} \in \mathrm{Val}(Z)$, and $\mathbf{z}' \in Res(h, \mathbf{u}, \mathbf{z})$, $\mathbf{z}|Y = \mathbf{z}'|Y$;*
- *$Dyn : \mathrm{Val}(U) \times \mathrm{Val}(Z) \to \mathrm{Val}(Z)$ is a differential equation giving a* continuous dynamics *of the form $\dot{\mathbf{z}} = Dyn(\mathbf{u}, \mathbf{z})$. We require Dyn to respect the following property:*
  - **T1** input trajectory enabling: *for every $\mathbf{z} \in Inv$ and every $\nu \in \mathrm{Trajs}(U)$, there exists $\tau \in \mathrm{Trajs}(V)$ such that $\tau|U = \nu$, $\mathrm{firststate}(\tau|Z) = \mathbf{z}$; for every $t \in \mathrm{dom}(\tau)$, $\tau(t)|Z \in Inv$ and $\dot{\tau}|Z(t) = Dyn(\tau(t))$; and either (i) $\mathrm{dom}(\tau) = \mathrm{dom}(\nu)$, or (ii) $\mathrm{dom}(\tau)$ is closed, $\mathrm{lasttime}(\tau) \leqslant \mathrm{lasttime}(\nu)$ and some $l \in L$ is enabled in $\mathrm{laststate}(\tau)$.*[2]

The fact that the invariant includes only locally controlled variables guarantees that a HIOA cannot constrain the value of the input variables by restricting the set of admissible states. Condition **A1** is the usual input enabling property of I/O automata. Condition **R1** asserts that the environment action ε cannot change the value of locally controlled variables, while condition **R2** asserts that the internal actions affect only internal variables and cannot change

[1] In this paper we use the notation $f : A \rightrightarrows B$ to denote multivalued functions, that is, functions f from A to the powerset of B.

[2] Note that formally, by $\dot{\tau}|Z(t)$ we mean $\frac{d}{dt}[\tau|Z](t)$ and not $\frac{d\tau}{dt}|Z(t)$, since $\nu = \tau|U$ need not be differentiable.

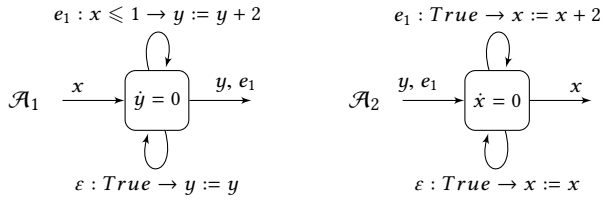

Figure 1: Two HIOAs.

the value of output variables. Condition **T1** establishes an input enabling condition on trajectories: for every input trajectory, either the automaton can follow it completely, or it must react with some locally controlled event. Violations of **T1** could potentially arise due to solutions of the differential equation leaving *Inv*, or becoming unbounded in finite time. The activation set *Act* and the reset map *Res* define the set of *discrete transitions* of the HIOA, namely, the set of triples $\mathbf{v} \xrightarrow{a} \mathbf{v}'$ such that $\mathbf{v}, \mathbf{v}' \in \mathrm{Val}(V)$, $a \in A$, $(a, \mathbf{v}) \in Act$ and $\mathbf{v}'|Z \in Res(a, \mathbf{v})$.

In the following we will sometime refer to an automaton that respects all properties of HIOA but condition **T1**: we call such an automaton *pre-HIOA*.

Example 3.2. *Figure 1 shows two HIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$. $\mathcal{A}_1$ has input variable x and output variable y with dynamics $\dot{y} = 0$. The dynamics of x is not specified since it is an input. Axiom **T1** is trivially respected. $\mathcal{A}_1$ has a single output action e_1 with guard $x \leqslant 1$ and reset $Res_1(e_1, y) = \{y + 2\}$. $\mathcal{A}_2$ has input variable y and output variable x with dynamics $\dot{x} = 0$. The action e_1 is an input action for $\mathcal{A}_2$: hence the corresponding discrete transition has guard $True$ (to respect the input enabling axiom **A1**), and reset $Res_2(e_1, x) = \{x+2\}$. Transitions labeled with the environment action ε are always active and do not change the value of the output variables. Notice that every HIOA defines the dynamics of its output variables and actions, while inputs dynamics are left unspecified. Similarly, an HIOA can constrain the activation of output actions but not of input actions.*

Given a set of actions A and a set of variables V, a *hybrid trajectory* over (A, V) (also called (A, V)*-trajectory*) is a possibly infinite sequence $\alpha = \tau_0 a_1 \tau_1 a_2 \tau_2 \ldots$ such that

(1) τ_i is a (continuous) trajectory over V, for every $i \geqslant 0$,
(2) a_i is an action in $A \cup \{\varepsilon\}$, for every $i \geqslant 0$,
(3) if α is finite then it ends with a trajectory, and
(4) if τ_i is not the last continuous trajectory of α, then $\mathrm{dom}(\tau_i)$ is right-closed.

We denote with $\mathrm{HTrajs}(A, V)$ the set of all hybrid trajectories over (A, V). If $\varepsilon \in A' \subseteq A$ and $V' \subseteq V$, then the (A', V')*-restriction* of α is the (A', V')-trajectory $\alpha' = \tau'_0 a'_1 \tau'_1 a'_2 \tau'_2 \ldots$ such that, for every $i \geqslant 0$, $\tau'_i = \tau_i|V'$ and $a'_i = a_i$ if $a_i \in A'$, $a'_i = \varepsilon$ otherwise.

We define $\mathrm{dom}(\alpha) = \bigcup_{n \in N} \{n\} \times \mathrm{dom}(\tau_n)$, $\alpha(n, t) = \tau_n(t)$ if $t \in \mathrm{dom}(\tau_n)$, and $\alpha(t)$ is the list of all $\tau_n(t)$ for which $t \in \mathrm{dom}(\tau_n)$.

(A, V)-trajectories are used to give the semantics of a HIOA in terms of *executions* and *traces*.

Definition 3.3. *An* execution *of a HIOA $\mathcal{A}$ from a state $\mathbf{z} \in \mathrm{Val}(Z)$ is a (A, V)-trajectory $\alpha = \tau_0 a_0 \tau_1 a_1 \tau_2 a_2 \ldots$ such that:*

(1) $\mathrm{firststate}(\tau_0)|Z = \mathbf{z}$*;*
(2) every τ_i is a solution of Dyn, namely, a trajectory on V such that $\dot{\tau}|Z(t) = Dyn(\tau(t))$ for every $t \in \mathrm{dom}(\tau)$;
(3) for every τ_i and $t \in \mathrm{dom}(\tau_i)$, $\tau_i(t)|Z \in Inv$;
(4) if τ_i is not the last trajectory in α, then $\mathrm{laststate}(\tau_i)|Z \xrightarrow{a_i} \mathrm{firststate}(\tau_{i+1})|Z$.

The corresponding trace*, denoted* $\mathrm{trace}(\alpha)$*, is the restriction of α to external variables and external actions.*

By axioms **A1** we have that the environment action ε is always active, while by axiom **R1** we have that ε events do not affect the state of the automaton. Hence, we can add an arbitrary number of ε events to the input trace of $\mathcal{A}$ without affecting its behavior, as long as those ε events do not change the value of input variables. This intuition is captured by the following definition of *stuttering equivalence* between hybrid traces.

Definition 3.4. *Let α and β be hybrid trajectories over (A, V). We say that α is* stuttering equivalent *to β, denoted by $\alpha \sim_\varepsilon \beta$, if it is possible to decompose α into $\alpha^0 \alpha^1 \alpha^2 \ldots$, and β into $\beta^0 \beta^1 \beta^2 \ldots$ such that, for every $j \geqslant 0$, either*

(i) $\alpha^j = \beta^j$*, or*
(ii) $\alpha^j = \alpha^j_1 \varepsilon \alpha^j_2$ *with* $\mathrm{laststate}(\alpha^j_1) = \mathrm{firststate}(\alpha^j_2)$ *and* $\beta^j = \alpha^j_1 \alpha^j_2$*, or*
(iii) $\beta^j = \beta^j_1 \varepsilon \beta^j_2$ *with* $\mathrm{laststate}(\beta^j_1) = \mathrm{firststate}(\beta^j_2)$ *and* $\alpha^j = \beta^j_1 \beta^j_2$.

We define $\mathrm{Execs}_{\mathcal{A}}$ to be the set of executions of $\mathcal{A}$, $\mathrm{Execs}_{\mathcal{A}}(\nu)$ to be the set of all executions with input ν, and $\mathrm{Execs}_{\mathcal{A}}(\nu, \mathbf{z}_0)$ the set of executions with input ν and initial state $\mathbf{z}_0$. The following lemma shows that the set of execution of $\mathcal{A}$ is closed under stuttering equivalence.

Lemma 3.5. *Let $\mathcal{A}$ be a HIOA, $\alpha \in \mathrm{Execs}_{\mathcal{A}}$ and $\beta \in \mathrm{HTrajs}(A, V)$ such that $\alpha \sim_\varepsilon \beta$. Then $\beta \in \mathrm{Execs}_{\mathcal{A}}$.*

Proof. Let $\alpha \in \mathrm{Execs}_{\mathcal{A}}$ and $\beta \in \mathrm{HTrajs}(A, V)$ such that $\alpha \sim_\varepsilon \beta$. Let $\alpha^0 \alpha^1 \alpha^2 \ldots$, and $\beta^0 \beta^1 \beta^2 \ldots$ be decompositions of α and β that respect conditions (i)–(iii) of Definition 3.4.

We prove that for every $j \geqslant 0$, β^j is an execution of $\mathcal{A}$. By Definition 3.4, three cases may arise.

(i) $\alpha^j = \beta^j$: trivial since α^j is an execution of $\mathcal{A}$;
(ii) $\alpha^j = \alpha^j_1 \varepsilon \alpha^j_2$ with $\mathrm{laststate}(\alpha^j_1) = \mathrm{firststate}(\alpha^j_2)$ and $\beta^j = \alpha^j_1 \alpha^j_2$. Since α^j_1 and α^j_2 are executions of $\mathcal{A}$ with $\mathrm{laststate}(\alpha^j_1) = \mathrm{firststate}(\alpha^j_2)$, then their concatenation β is an execution of $\mathcal{A}$;
(iii) $\beta^j = \beta^j_1 \varepsilon \beta^j_2$ with $\mathrm{laststate}(\beta^j_1) = \mathrm{firststate}(\beta^j_2)$ and $\alpha^j = \beta^j_1 \beta^j_2$. Since β^j_1 and β^j_2 are executions of $\mathcal{A}$, and $\mathrm{laststate}(\beta^j_1) \xrightarrow{\varepsilon} \mathrm{firststate}(\beta^j_2)$ is a valid discrete transition, then $\beta^j_1 \varepsilon \beta^j_2$ is an execution of $\mathcal{A}$.

Since β is a countable concatenation of executions of $\mathcal{A}$, it is an execution of $\mathcal{A}$. □

Traces of $\mathcal{A}$ are not necessarily closed under stuttering equivalence. Hence, we define the set $\mathrm{Traces}_{\mathcal{A}}$ to be the closure under stuttering equivalence of the set of traces of $\mathcal{A}$: $\mathrm{Traces}_{\mathcal{A}} = \{\beta \in \mathrm{HTrajs}(E, W) \mid \exists \alpha \in \mathrm{Execs}_{\mathcal{A}}, \mathrm{trace}(\alpha) \sim_\varepsilon \beta\}$. Similarly define $\mathrm{Traces}_{\mathcal{A}}(\nu)$ and $\mathrm{Traces}_{\mathcal{A}}(\nu, \mathbf{z}_0)$.

Composition is defined as a partial binary operation on hybrid automata, which corresponds to the usual parallel composition of automata adapted to our context.

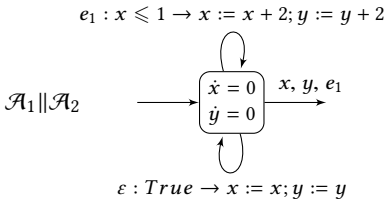

Figure 2: Composition of HIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ from Example 3.2.

Definition 3.6. *Given two HIOAs $\mathcal{A}_1 = (U_1, X_1, Y_1, I_1, H_1, O_1, Inv_1, Act_1, Res_1, Dyn_1)$ and $\mathcal{A}_2 = (U_2, X_2, Y_2, I_2, H_2, O_2, Inv_2, Act_2, Res_2, Dyn_2)$, we say that they are* compatible *if and only if*

(1) $H_1 \cap A_2 = H_2 \cap A_1 = \emptyset$, $X_1 \cap V_2 = X_2 \cap V_1 = \emptyset$ (disjointness of internal actions and variables), and
(2) $Y_1 \cap Y_2 = O_1 \cap O_2 = \emptyset$ (disjointness of output actions and variables).

We define the composition as follows.

Definition 3.7. *Given two compatible HIOAs $\mathcal{A}_1 = (U_1, X_1, Y_1, I_1, H_1, O_1, Inv_1, Act_1, Res_1, Dyn_1)$ and $\mathcal{A}_2 = (U_2, X_2, Y_2, I_2, H_2, O_2, Inv_2, Act_2, Res_2, Dyn_2)$, we define their composition $\mathcal{A}_1 \| \mathcal{A}_2$ as the structure $\mathcal{A} = (U, X, Y, I, H, O, Inv, Act, Res, Dyn)$ such that:*

(1) $Y = Y_1 \cup Y_2$, $U = (U_1 \cup U_2) \setminus Y$, and $X = X_1 \cup X_2$;
(2) $O = O_1 \cup O_2$, $I = (I_1 \cup I_2) \setminus O$, $H = H_1 \cup H_2$;
(3) $Inv = \{\mathbf{z} \in \mathrm{Val}(Z) : \mathbf{z}|Z_1 \in Inv_1 \wedge \mathbf{z}|Z_2 \in Inv_2\}$;
(4) for each $\mathbf{v} \in \mathrm{Val}(V)$ and $a \in A$, $(a, \mathbf{v}) \in Act$ iff for every $i = 1, 2$ either (i) $a \in A_i$ and $\mathbf{v}|V_i \in Act_i$, or (ii) $a \notin A_i$;
(5) for each $\mathbf{v} \in \mathrm{Val}(V)$, $\mathbf{z}' \in Res(a, \mathbf{v})$ iff for every $i = 1, 2$ either (i) $a \in A_i$ and $\mathbf{z}'|Z_i \in Res_i(a, \mathbf{v}|V_i)$, or (ii) $a \notin A_i$ and $\mathbf{z}'|Z_i = \mathbf{v}|Z_i$;
(6) for every $\mathbf{v} \in \mathrm{Val}(V)$, we have that $Dyn(\mathbf{v})|Z_1 = Dyn_1(\mathbf{v}|V_1)$ and $Dyn(\mathbf{v})|Z_2 = Dyn_2(\mathbf{v}|V_2)$.

Example 3.8. *Consider the HIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ from Example 3.2: they are compatible, since y is an output variable of $\mathcal{A}_1$ but not of $\mathcal{A}_2$, x is an output of $\mathcal{A}_2$ but not of $\mathcal{A}_1$ and the action e_1 is an output action of $\mathcal{A}_1$ but not of $\mathcal{A}_2$. Figure 2 shows the composition $\mathcal{A}_1 \| \mathcal{A}_2$: it has two output variables x, y, with dynamics $\dot{x} = 0$ and $\dot{y} = 0$, and no inputs. The action e_1 is an output action for the composition: the activation is $x \leqslant 1$ and the reset is $Res(e_1, x, y) = \{(x + 2, y + 2)\}$, which corresponds to the composition of the two resets from $\mathcal{A}_1$ and $\mathcal{A}_2$. In this particular example the composed system is closed: all the variables have a defined dynamics, and all actions are controlled by the system, except for the environment action ε.*

In the original definition of [28], the composition of two HIOAs is not necessarily a HIOA. Indeed, the authors of [28] show that all properties but **T1** are preserved by composition. To show that property **T1** is not preserved by composition, they give an example of two HIOAs whose composition does not respect **T1**. Even though the counter-example given in [28] does not fit in our definition of a HIOA since the dynamics are not continuous, there are counter-examples fitting in our definition with continuous dynamics, as demonstrated below:

Example 3.9. *Consider the HIOA $\mathcal{A}$ with no events, input variables u, v and output variable y with dynamics $\dot{y} = u^2$. The input variable v does not affect the dynamics of the system. Then the continuous trajectories satisfy **T1** with $y(t) = y(0) + \int_0^t u(t)^2 dt$.*

*Now consider identical copies $\mathcal{A}_{1,2}$, coupled by $u_1 = y_2$ and $u_2 = y_1$ (v_1 and v_2 are not coupled). The composed system $\mathcal{A}_1 \| \mathcal{A}_2$ has input variables v_1, v_2 and output variables y_1, y_2. With the initial conditions $y_1(0) = y_2(0) = 1$, the executions of $\mathcal{A}_1$ and $\mathcal{A}_2$ are identical, so they satisfy $\dot{y} = y^2$ with $y(0) = 1$, which has solution $y(t) = 1/(1 - t)$ for $t \in [0, 1)$. Hence, the composed system does not satisfy **T1**: since the executions are only defined on a bounded time interval, the system cannot follow input trajectories defined over an interval bigger than $[0, 1)$.*

However, composition still yields a pre-HIOA:

Theorem 3.10. *If $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible HIOAs then $\mathcal{A}_1 \| \mathcal{A}_2$ is a pre-HIOA.*

Proof. Let $\mathcal{A} = \mathcal{A}_1 \| \mathcal{A}_2$. We have to prove that it respects all properties of Definition 3.1 but **T1**.

Disjointness of U, X, and Y follows from disjointness of U_1, X_1, and Y_1, from disjointness of U_2, X_2, and Y_2, and compatibility. Similarly, disjointness of I, H, and O follows from disjointness of I_1, H_1, and O_1, from disjointness of I_2, H_2, and O_2, and compatibility. Nonemptiness of Inv follows from nonemptiness of Inv_1 and Inv_2.

Now, we verify the **A1** property. Consider $\mathbf{v} \in \mathrm{Val}(V)$ and $a \in I$: three cases may arise.

(1) $a \in I_1 \cap I_2$. Since $\mathcal{A}_i$ respects **A1**, we have that $(a, \mathbf{v}|V_i) \in Act_i$ for $i = 1, 2$. By definition of composition, $(a, \mathbf{v}) \in Act$.
(2) $a \in I_1 \setminus I_2$. Since $\mathcal{A}_1$ and $\mathcal{A}_2$ respect **A1**, we have that $(a, \mathbf{v}|V_1) \in Act_1$ and $(\varepsilon, \mathbf{v}|V_2) \in Act_2$. Therefore, by definition of composition, $(a, \mathbf{v}) \in Act$.
(3) $a \in I_2 \setminus I_1$. This case is symmetric to the previous one and thus is skipped.

To conclude the proof, we have to show that **R1** and **R2** are verified. Let $\mathbf{u} \in \mathrm{Val}(U)$, $\mathbf{z}, \mathbf{z}' \in \mathrm{Val}(Z)$ be such that $\mathbf{z}' \in Res(\varepsilon, \mathbf{u}, \mathbf{z})$, and let $\mathbf{v} = \mathbf{u} \cup \mathbf{z}$. By the definition of composition, we have that $\mathbf{z}'|Z_i \in Res(\varepsilon, \mathbf{v}|V_i)$, for $i = 1, 2$. By **R1** applied to $\mathcal{A}_i$ we have that $Res(\varepsilon, \mathbf{v}|V_i) = \{\mathbf{v}|Z_i\}$ and hence that $\mathbf{v}|Z_i = \mathbf{z}'|Z_i$. Since $Z = Z_1 \cup Z_2$, we can conclude that $\mathbf{v}|Z = \mathbf{z} = \mathbf{z}'$, and thus that $Res(\varepsilon, \mathbf{u}, \mathbf{z}) = \{\mathbf{z}\}$ as needed.

To prove **R2**, let $h \in I$, $\mathbf{u} \in \mathrm{Val}(U)$, $\mathbf{z}, \mathbf{z}' \in \mathrm{Val}(Z)$ be such that $\mathbf{z}' \in Res(h, \mathbf{u}, \mathbf{z})$, and let $\mathbf{v} = \mathbf{u} \cup \mathbf{z}$. Since $h \in I = I_1 \cup I_2$, by the disjointness of I_1 and I_2 we have that either $h \in I_1$ or $h \in I_2$. Suppose w.l.o.g. that $h \in I_1$: by the definition of composition, we have that $\mathbf{z}'|Z_1 \in Res(h_1, \mathbf{v}|V_i)$, and that $\mathbf{z}'|Z_2 \in Res(\varepsilon, \mathbf{v}|V_2)$. By **R2** applied to $\mathcal{A}_1$ we have that $\mathbf{z}'|Y_1 = \mathbf{v}|Y_1$. By **R1** applied to $\mathcal{A}_2$ we have that $\mathbf{z}'|Z_2 = \mathbf{v}|Z_2$, which implies that $\mathbf{z}'|Y_2 = \mathbf{v}|Y_2$. Since $Y = Y_1 \cup Y_2$, we can conclude that $\mathbf{z}|Y = \mathbf{z}'|Y$, as needed. □

This definition of a HIOA is *compositional* in the sense that it respects the following fundamental properties. First of all, the following *projection* results establish that the traces of a composition project down to traces of the individual components. Proofs are omitted to save space and will be included in a forthcoming journal paper. Similar results, with complete proofs, can also be found in [28].

Lemma 3.11. *Let $\mathcal{A} = \mathcal{A}_1 \| \mathcal{A}_2$ and let α be an execution fragment of $\mathcal{A}$. Then $\alpha|(A_1, V_1)$ and $\alpha|(A_2, V_2)$ are execution fragments of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively.*

Lemma 3.12. *Let $\mathcal{A} = \mathcal{A}_1 \| \mathcal{A}_2$ and let α be an execution fragment of $\mathcal{A}$. Then* $\mathrm{trace}(\alpha)|(E_i, W_i) = \mathrm{trace}(\alpha|(A_i, V_i))$.

The following theorem relates the set of traces of a composition to the sets of traces of the components. It is expressed in terms of equality between two sets of traces. Set inclusion in one direction follows from the previous lemmas. Set inclusion in the other direction expresses the idea that traces of components can be "pasted together" to yield a trace of the composition.

Theorem 3.13. *Let $\mathcal{A} = \mathcal{A}_1 \| \mathcal{A}_2$. Then the set of traces of $\mathcal{A}$ is exactly the set of (E, W)-trajectories whose restriction to $\mathcal{A}_1$ and $\mathcal{A}_2$ are traces of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. That is,* $\mathrm{Traces}_{\mathcal{A}} = \{\beta \in \mathrm{HTrajs}(E, W) \mid \exists \beta' \sim_\varepsilon \beta \text{ such that } \beta'|(E_i, W_i) \in \mathrm{Traces}_{\mathcal{A}_i}, i = 1, 2\}$

Axiom **R2** is crucial to prove the opposite direction of Theorem 3.13. Indeed, if we allow internal events to change output variables, then we can have (E, W)-trajectories β that project down to traces of $\mathcal{A}_1$ and $\mathcal{A}_2$ but are not included in $\mathrm{Traces}_{\mathcal{A}}$.

Example 3.14. *Consider the following variant of the two HIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ from Example 3.2. The variables x and y, and their dynamics are unchanged. The action e_1 is replaced in $\mathcal{A}_1$ by an internal action h_1 with the same guard $x \leqslant 1$ and the same reset $Res_1(h_1, y) = \{y + 2\}$. In $\mathcal{A}_2$ the input action e_1 is replaced by an internal action h_2 with guard $y \leqslant 1$ and reset $Res_2(h_2, x) = \{x + 2\}$. Notice that in this case $\mathcal{A}_2$ can control the activation of h_2 since it is an internal action. The new HIOAs respect all conditions of the Definition 3.1 except **R2** (the internal events change the values of outputs).*

The composition $\mathcal{A}_1 \| \mathcal{A}_2$ has output variables x, y with dynamics $\dot{x} = 0$ and $\dot{y} = 0$. The internal actions h_1 and h_2 have the same guards and resets as above, with the additional constraint that h_1 does not change the value of x and h_2 does not change the value of y.

Now, take a trace $\beta = \tau_1 \varepsilon \tau_2$ where τ_1 is a constant trajectory with $x = 0$ and $y = 0$ and τ_2 is a constant trajectory with $x = 2$ and $y = 2$. If you project β to $\mathcal{A}_1$ you obtain an execution where ε is replaced by h_1. If you project down to $\mathcal{A}_2$, ε is replaced by h_2.

However, no execution of $\mathcal{A}_1 \| \mathcal{A}_2$ can generate β, or any other trace $\beta' \sim_\varepsilon \beta$: when $x = 0$ and $y = 0$ both h_1 and h_2 are active in $\mathcal{A}$. If h_1 is executed first, then x is kept at 0 while y is reset to 2 and h_2 is deactivated. Then, the only possible trajectories that can follow h_1 are constant trajectories with $x = 0$ and $y = 2$. The same if h_2 is executed first: x is reset to 2 and y kept at 0, deactivating h_1. In this case the only possible trajectories after h_2 have $x = 2$ and $y = 0$.

As a direct consequence of the previous results, we can substitute any of the two components of $\mathcal{A}_1 \| \mathcal{A}_2$ with another one that exhibits the same external behavior without affecting the behavior of the whole composition.

Theorem 3.15. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two hybrid automata such that $\mathcal{A}_1 \leqslant \mathcal{A}_2$ (the set of traces of $\mathcal{A}_1$ is contained in the set of traces of $\mathcal{A}_2$), and let $\mathcal{B}$ be a hybrid automaton compatible with both $\mathcal{A}_1$ and $\mathcal{A}_2$. Then $\mathcal{A}_1 \| \mathcal{B} \leqslant \mathcal{A}_2 \| \mathcal{B}$.*

4 COMPUTABLE ANALYSIS

In the theory of computable analysis, computation is performed by Turing Machines acting on (infinite) streams of data. These data streams encode a *sequence of approximations* to some quantity (such as a region of the state space, or a function describing the system) with *guarantees on the accuracy*. A given function or operator is *computable* if, given data streams encoding the inputs, it is possible to calculate a data stream encoding the output. Finite computations can be obtained by terminating when a given accuracy criterion is satisfied. The theory of computable analysis therefore allows to determine whether the solution to a certain problem can be approximated to *any* desired and *known* accuracy.

Even though the model of computation is based on ordinary Turing Machines, the main purpose of computable analysis is to deal with approximations. The fact that input data are interpreted to be approximate can drastically change the computability properties of problems, as shown by the following simple example.

Example 4.1. *Consider the problem of testing whether a guard $p(x) \geqslant 0$ is true or not, where p is some polynomial with rational coefficients. If the value of x is a rational number that is described exactly, the exact value for $p(x)$ can be computed and the problem is easily solvable. However, if the* only *information that we have about x is a sequence of approximations that converges to x at a known rate, then the problem becomes semi-decidable. From an approximation $\tilde{x}$ to x with error known to be at most ϵ_x, it is possible to compute an approximation $\tilde{y}$ to $y = p(x)$ and an error bound ϵ_y. When $p(x) < 0$ or $p(x) > 0$ it is possible to find a sufficiently accurate $\tilde{x}$ that allows us to prove that the guard is false in the former case and true in the latter case. On the other hand, when $p(x) = 0$, no matter how accurate the current approximation $\tilde{x}$ is, we cannot exclude neither the possibility that $p(x) < 0$, nor that $p(x) > 0$, and thus we cannot give a definite answer to the problem.*

The representations used in computable analysis to encode inputs and outputs are related to a particular topology on the set of objects under consideration. Although a given space may have many representations, most spaces, including the real numbers $\mathbb{R}$ have a unique "natural" representation (up to computable equivalence). We call a space with a representation a *type*, so e.g. the *real number type* is the space of real numbers encoded by a natural representation.

We also have natural representations for product spaces $X_1 \times X_2$ and spaces of continuous functions $C(X, Y)$. The projections $\pi_i : X_1 \times X_2 \to X_i$ are computable, as is the evaluation $C(X, Y) \times X \to Y$ with $f, x \mapsto f(x)$. Further, to *compute* $f \in C(X, Y)$, it suffices to show that one can effectively evaluate f for every argument. If $\mathbb{X}$ is a locally-compact Hausdorff space, then the topology induced by the natural representation on $C(X, Y)$ is the *compact-open* topology, of uniform convergence on compact sets.

A fundamental theorem is that *only continuous functions and operators can be computable* with respect to a given representation and to the corresponding topology [38]. Hence, if we can prove that a certain function is discontinuous, then it is uncomputable.

The converse is not generally true, but as a guiding principle, any "naturally-defined" continuous operator *is* computable.[3]

Since the only functions from a connected space to the Booleans are trivial, Boolean logic with values $\top$ (true) and $\bot$ (false) does not suffice. Logical predicates may also take the *indeterminate* value $\uparrow$, so to yield a *Kleenean* result $\mathbb{K} = \{\top, \uparrow, \bot\}$. As a topological space, $\mathbb{K}$ has open sets $\{\}, \{\top\}, \{\bot\}, \{\top, \bot\}, \{\top, \uparrow, \bot\}$. Unlike in discrete computation, the value $\uparrow$ is not considered an error, but an indication that the result of the predicate is highly sensitive to the arguments. When we merely aim to test the truth of a predicate, we consider a *Sierpinskian* result $\mathbb{S} = \{\top, \uparrow\}$. We say a predicate p taking values in $\mathbb{K}$ is *quasidecidable*, and one taking values in $\mathbb{S}$ is *verifiable* (semidecidable). A predicate taking values in $\{\uparrow, \bot\}$ is *falsifiable*.

We will also need to consider representations of sets of objects. A natural representation of a subset S of a space X is by its membership predicate $x \in S$. This should take values in the Kleeneans $\mathbb{K}$, with $x \in S$ yielding $\uparrow$ for the borderline case $x \in \partial S$. We call sets represented in this way *regular*, denoted $\mathcal{R}(X)$, since we can verify $x \in \mathrm{int}(S)$ and $x \notin \mathrm{cl}(S)$. When we are only interested in verifying $x \in S$, we are led to the natural representation of *open* sets $\mathcal{O}(X)$ describing open U by a membership predicate $U \ni x$ taking values in the Sierpinskians $\mathbb{S}$. Similarly, *closed* sets $\mathcal{A}(X)$ are described by a Sierpinskian-valued non-membership predicate $S \not\ni x$. The open and closed set types are both subtypes of the regular set type, i.e. the regular set representation yields more information.

We often wish to consider universal and existential quantification over sets. If $p : X \to \mathbb{S}$ is a verifiable predicate (i.e. is true on some open set U), then verifying $\forall x \in S$, $p(x)$ is equivalent to testing $S \subset U$. It turns out that sets S for which the predicates $S \subset U$ for open U are continuous are precisely the compact sets, yielding a representation for $\mathcal{K}(X)$. The underlying topology is the *upper Vietoris topology*. Similarly, verifying $\exists x \in S$, $p(x)$ is equivalent to testing whether S intersects U, denoted $S \mathbin{\not\emptyset} U$. Predicates $S \mathbin{\not\emptyset} U$ are continuous for all *separable* sets S (which includes all subsets of $\mathbb{R}^n$), and since $S \mathbin{\not\emptyset} U \iff \mathrm{cl}(S) \mathbin{\not\emptyset} U$, sets are described up to their closure. The underlying topology is the *lower Fell topology*. In the literature, the type of separable sets $\mathcal{V}(X)$ is also sometimes called the *overt* set type. The type of *located* sets $\mathcal{L}(X)$ supports both universal and existential verifiable predicates, so these sets are both compact and separable. The underlying topology is the *Vietoris* topology, which is that of convergence in the Hausdorff metric.

We will say S is *upper-semicomputable* if it is computable as a compact set, and *lower-semicomputable* if it is computable as a separable set. Concretely, if S is upper-semicomputable, then we can compute a sequence of over-approximations A_n (say, as a finite union of rational boxes) that converges to S from outside i.e. $S \subset \mathrm{int}(A_n)$. Similarly, if S is lower-semicomputable, then we can compute a sequence of approximations A_n converging to S "from below" i.e. such that $A_n \subset N_{2^{-n}}(F(x))$.

Important computable operations on sets are

i. Singleton $\{x\}$ is computable as a located set, and hence also as a compact set or a separable set. i.e. $x \mapsto \{x\}$ is computable $X \to \mathcal{L}(X), X \to \mathcal{K}(X)$ and $X \to \mathcal{V}(X)$.
ii. Intersection $S, R \mapsto S \cap R$ of a located set and a regular set is computable as a located set i.e. $\mathcal{L}(X) \times \mathcal{R}(X) \to \mathcal{L}(X)$. Further, intersection of a compact and a closed set is computable i.e. as $\mathcal{K}(X) \times \mathcal{A}(X) \to \mathcal{K}(X)$; and intesection of a separable and an open set is computable i.e. as $\mathcal{V}(X) \times \mathcal{O}(X) \to \mathcal{V}(X)$.
iii. Union $S_1, S_2 \mapsto S_1 \cup S_2$ is computable for all set types i.e. $\mathscr{S}(X) \times \mathscr{S}(X) \to \mathscr{S}(X)$ for $\mathscr{S} = \mathcal{R}, \mathcal{O}, \mathcal{A}, \mathcal{L}, \mathcal{K}, \mathcal{V}$.

For subsets of the real line, we have the following additional operations:

iv. If T is a regular (respectively open, closed) subset of $\mathbb{R}^+$ containing 0, then the maximum interval of T containing 0 is computable as a regular (respectively open, closed) set.
v. If $T_0 = [t_0, t_1]$ and $T_1 = [t_1, t_2]$ are closed intervals, then t_1 can be computed from T_0 and T_1.

Multivalued functions $X \rightrightarrows Y$ can be represented as types of functions returning sets. Given a function $F : X \rightrightarrows Y$, we say that F is *upper-semicomputable* if, given a point x, it is possible to compute $F(x)$ as a compact set. This means $F \in \mathcal{C}(X, \mathcal{K}(Y))$. Similarly, F is *lower-semicomputable* if it is possible to compute $F(x)$ as a separable set i.e. $F \in \mathcal{C}(X, \mathcal{V}(Y))$. A multivalued function $F : X \rightrightarrows Y$ is *computable* if and only if it is both upper and lower semicomputable i.e. computable as a located set. Similarly, partial functions can be considered as multivalued functions returning singleton or empty sets; in particular, the type $\mathcal{C}_{\mathcal{A}}(X, Y)$ of partial functions with closed domain is a subtype of compact-valued functions $\mathcal{C}(X, \mathcal{K}(Y))$.

The forward image of located, compact and separable sets is computable under multivalued maps i.e. $F(S)$ is computable in $\mathscr{S}(Y)$ for $F \in \mathcal{C}(X, \mathscr{S}(Y))$ and $S \in \mathscr{S}(X)$ for $\mathscr{S} = \mathcal{L}, \mathcal{K}, \mathcal{V}$.

Note that if F is upper-semicomputable, then it is *upper-semicontinuous*, meaning that for x close to x_0, $F(x)$ is a subset of a small neighborhood of $F(x_0)$. If F is lower-semicomputable, then it is *lower-semicontinuous*.

Important non-computable operations (since these are not continuous) are forward-image of closed sets $f, A \mapsto f(A)$, verifying $A \subset U$ for a closed set A in an open set U, and intersection of separable sets.

It is worth emphasizing that a function that is uncomputable with respect to given representations of the arguments and results can be computable with respect to representations based on a different topology. This corresponds to requiring more information on the inputs, or to requiring less information on the output of the function.

Example 4.2. *Let G be the set of points satisfying the condition $g(x) \geqslant 0$. Then G* cannot, *in general, be computed to arbitrary accuracy in the topology of the Hausdorff metric i.e. as a located set, but it* is *possible to compute a sequence of concrete over-approximations (as the complement of unions of rational boxes) which converges to G i.e. to compute G as a closed set. A counterexample is given by $g(x) = (x-a)^2(x-b) + e$ with $a < b$, for which G contains a neighborhood of a for $e > 0$, but not for $e < 0$, so testing $a \in G$ is undecidable for $e = 0$ if e is only known to arbitrary accuracy.*

5 COMPUTABILITY RESULTS

In [14, 16], it was shown how to apply the theory of computable analysis to find algorithmic solutions to the reachability problem for hybrid automata without inputs. In this section we will show

[3] An open case is the embedding of a countable product of compact sets in the product space, though even this is known to be computable for countably-based spaces.

how to extend these results to compute execution sets for hybrid input-output automata.

The use of an appropriate topology/representation is crucial when considering computability for hybrid automata, where discrete transitions can cause discontinuities in both space and time [14]. A natural description of the executions of a hybrid automaton is as a set of hybrid trajectories over (A, V). To obtain a representation of the space of (A, V)-hybrid trajectories $\alpha = \tau_0 a_0 \tau_1 \cdots$, we need to consider the continuous trajectories, which are partial functions $\tau : \mathbb{R}^+ \to \mathrm{Val}(V)$ defined on closed intervals i.e. $\tau \in C_{\mathcal{A}}(\mathbb{R}^+, \mathrm{Val}(V))$. In the corresponding topology, hybrid trajectories α_k converge to α_∞ with $N_\infty \in \mathbb{N} \cup \{\infty\}$ if (i) the time intervals $\mathrm{dom}(\tau_{k,i}) = [0, t_{k,i}]$ satisfy $t_{k,i} \to t_{\infty,i}$ if $i < N_\infty$, and $t_{k,N_\infty+1} \to \infty$ if $N_\infty < \infty$; (ii) for each $i < N_\infty$, the events $a_{k,i}$ equal $a_{\infty,i}$ for sufficiently large i, (iii) the $\tau_{k,i}$ converge uniformly to $\tau_{\infty,i}$ for $i < N_\infty$, and (iv) if $N_\infty < \infty$, τ_{k,N_∞} converge uniformly to τ_{∞,N_∞} on every compact subset $[0, t]$ of $[0, \infty)$.

Even for the simple class of *deterministic* hybrid systems without inputs, the solution trajectory has unavoidable discontinuities with respect to the initial conditions, so by the fundamental theorem of computable analysis it is uncomputable.

Theorem 5.1 ([14, Theorem 4.6]). *For any coherent semantics of evolution[4], the finite-time evolution of a hybrid system is uncomputable.*

The above theorem proves that it is impossible to regularize the evolution near the discontinuity points to make the solution computable. However, it does not in itself rule out the possibility of regularizing the evolution in some way so that the evolution becomes at least semi-computable.

One possible approach is to compute sets of trajectories. We cannot compute the executions as a compact set, since the set of allowable input (I, U)-trajectories is not compact. We could try to compute sets of trajectories as a closed set. While this is possible for a large class of systems, the resulting information is too weak to be used to prove safety properties, since testing inclusion of a closed set in an open set is undecidable.

Instead, we represent the executions by an input-to-state mapping taking (I, U)-trajectories to sets of (A, V)-trajectories. Theorem 5.1 implies that the set of (A, V)-trajectories is not in general computable as a located set. However, for appropriate system classes, we will be able to compute the set as either a compact set, or as a separable set, yielding two kinds of semicomputability.

Before defining our system classes, we first consider computability of solutions of the differential equation describing the continuous dynamics.

Definition 5.2. *The differential equation $\dot{\mathbf{z}} = Dyn(\mathbf{u}, \mathbf{z})$ has* unique, entire solutions *if for all continuous input functions $\mathbf{u} : \mathbb{R}^+ \to \mathrm{Val}(U)$ and initial conditions $\mathbf{z}_0 \in \mathrm{Val}(Z)$, there exists a unique differentiable function $\mathbf{z} : \mathbb{R}^+ \to \mathrm{Val}(Z)$ such that $\mathbf{z}(0) = \mathbf{z}_0$ and for all $t \in \mathbb{R}^+$, $\dot{\mathbf{z}}(t) = Dyn(\mathbf{u}(t), \mathbf{z}(t))$.*

[4]The formal definition of coherent semantics is given in [14]. Here it is sufficient to say that this condition eliminates all trivial approximations, like the one that takes the entire state space.

Well-known sufficient conditions for a differential equation to have unique, entire solutions are that it is locally-Lipschitz with linear growth.

Definition 5.3.

a) *The differential equation $\dot{\mathbf{z}} = Dyn(\mathbf{u}, \mathbf{z})$ has* linear growth *if there exists a constant C such that $\|\dot{\mathbf{z}}\| \leqslant C(\|\mathbf{u}\| + \|\mathbf{z}\| + 1)$, and* linear growth in state *if $\|\dot{\mathbf{z}}\| \leqslant C(\|\mathbf{z}\| + 1)$*
b) *The differential equation $\dot{\mathbf{z}} = Dyn(\mathbf{u}, \mathbf{z})$ is* locally-Lipschitz *if for every constant B there exists a constant L such that for all $\mathbf{u} \in \mathrm{Val}(U)$, $\mathbf{z}_1, \mathbf{z}_2 \in \mathrm{Val}(Z)$ with $\|\mathbf{u}\|, \|\mathbf{z}_1\|, \|\mathbf{z}_2\| \leqslant B$, we have $\left\|Dyn(\mathbf{u}, \mathbf{z}_1) - Dyn(\mathbf{u}, \mathbf{z}_2)\right\| \leqslant L\|\mathbf{z}_1 - \mathbf{z}_2\|$.*

The conditions in Definition 5.3 are standard in the literature. The locally-Lipschitz condition is sufficient for solutions to be unique, and linear growth is sufficient for solutions to be entire.

Definition 5.4. *Let $\mathcal{A} = \langle U, X, Y, I, H, O, Inv, Act, Res, Dyn \rangle$ be a HIOA.*

a) *$\mathcal{A}$ is said to be* upper-semicontinuous *if Inv and Act are* closed*, Res is an upper-semicontinuous compact-valued function, and Dyn has unique entire solutions.*
b) *$\mathcal{A}$ is said to be* lower-semicontinuous *if Inv is* open*, Res is a lower-semicontinuous separable-valued function, and Dyn has unique entire solutions.*

For such systems, the evolution is semicomputable:

Theorem 5.5. *Let $\mathcal{A}$ be a HIOA.*

a) *If $\mathcal{A}$ is upper-semicontinuous, then the set of executions for a given input trace and initial condition is upper-semicomputable i.e. computable as a compact set.*
b) *If $\mathcal{A}$ is lower-semicontinuous, then the set of executions for a given input trace and initial condition is lower-semicomputable i.e. computable as a separable set.*

To prove the theorem we use the following lemmas.

Lemma 5.6. *Let Dyn have unique, entire solutions, in particular, when Dyn is locally-Lipschitz with linear growth Then the continuous trajectory $\mathbf{z}$ satisfying Dyn is computable given the continuous input trajectory $\mathbf{u}$ and the initial state $\mathbf{z}_0$.*

Proof. By a theorem of Ruohonen [32], the solution of a differential equation with unique solutions is computable. □

Lemma 5.7.

a) *Let Res be a compact-valued upper-semicontinuous map and Act a closed set. Then the operator $Res|_{Act}$ is upper-semicomputable.*
b) *Let Res be a separable-valued lower-semicontinuous map and Act an open set. Then the map $Res|_{Act}$ is lower-semicomputable.*

Proof.

a) For any $x \in \mathrm{Val}(V)$, $Res|_{Act}(x) = Res(\{x\} \cap Act)$. Given x, $\{x\}$ is computable as a compact set. $\{x\} \cap Act$ is the intersection of a compact and a closed set, so it is computable as a compact set. Finally, $Res(\{x\} \cap Act)$ is the image of a compact set under a compact-valued function that is upper-semicomputable, so it is computable as a compact set.
b) For any $x \in \mathrm{Val}(V)$, $\{x\}$ is computable as a separable set, $\{x\} \cap Act$ is the intersection of a separable and an open set, so it is computable as a separable set. Then $Res(\{x\} \cap Act)$ is the image of

a separable set under a separable-valued lower-semicomputable function, so it is computable as a separable set. □

Proof of Theorem 5.5.

a) Given the input hybrid trajectory ν, we solve the differential equation $\dot{\mathbf{z}} = Dyn(\mathbf{u}_0, \mathbf{z}_0)$ up to the first event time ν to find the trajectory τ, by Lemma 5.6. We then compute the set of times for which $\tau(t) \in Inv$ as a closed set, and the maximum interval D containing 0 for which $\tau(D) \subset Inv$. We further consider the set of prefixes τ' of τ with $\text{lasttime}(\tau') \in D$. For each such trajectory τ' and each action a, we apply $Res_{Act}(a, \text{laststate}(\tau'))$ by Lemma 5.7 to obtain an initial state for the next trajectory.
b) Similarly to a), one can iteratively build a lower-approximation to the set of trajectories solving the system. □

General hybrid automata of the form given by Definition 3.1 are not necessarily upper or lower semicontinuous. In order to compute upper or lower approximations to the solution, we need to convert the automaton into either upper or lower semicontinuous form. We can do this by forcing dynamics and reset functions to be continuous and by regularizing invariants and guard sets to be open or closed.

Definition 5.8. *Let $\mathcal{A} = \langle U, X, Y, I, H, O, Inv, Act, Res, Dyn\rangle$ be a (pre-) HIOA such that Res is a continuous multivalued function with compact separable values, and Dyn has unique, entire solutions. Then*

a) α is an execution of $\mathcal{A}$ using upper-semantics *if α is an execution of the upper-semicontinuous automaton $\overline{\mathcal{A}} = \langle U, X, Y, I, H, O, \text{cl}(Inv), \text{cl}(Act), Res, Dyn\rangle$.*
b) α is an execution of $\mathcal{A}$ using lower-semantics *if α is in the closure of the set of executions of the lower-semicontinuous automaton $\underline{\mathcal{A}} = \langle U, X, Y, I, H, O, \text{int}(Inv), \text{int}(Act), Res, Dyn\rangle$.*

The following result is a direct consequence of Theorem 5.5.

Corollary 5.9. *Let $\mathcal{A} = \langle U, X, Y, I, H, O, Inv, Act, Res, Dyn\rangle$ be a HIOA such that Res is a continuous multivalued function with compact separable values, and Dyn has unique, entire solutions. Then the evolution of $\mathcal{A}$ using upper semantics is upper-semicomputable and the evolution of $\mathcal{A}$ using lower semantics is lower-semicomputable.*

Moreover, it turns out that $\overline{\mathcal{A}}$ is the "smallest" hybrid automaton for which the evolution is upper-semicomputable and that $\underline{\mathcal{A}}$ is the "greatest" hybrid automaton for which the evolution is lower-semicomputable. This means that, in general, the approximations given by the upper and by the lower semantics are the "best" approximations for computing the evolution of hybrid automata, unless additional information is provided.

Theorem 5.10 (**Main theorem**). *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two compatible upper-semicontinuous (resp., lower-semicontinuous) HIOAs. Suppose Dyn_1 and Dyn_2 are locally-Lipschitz functions with linear growth. Then the parallel composition $\mathcal{A}_1 \| \mathcal{A}_2$ is an upper-semicontinuous (resp., lower-semicontinuous) pre-HIOA that can be effectively computed.*

Proof. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two compatible upper-semicontinuous HIOAs, and let $\mathcal{A}$ denote $\mathcal{A}_1 \| \mathcal{A}_2$. By Theorem 3.10, $\mathcal{A}$ is a pre-HIOA. We have to prove that $\mathcal{A}$ is upper-semicontinuous and that it can be effectively computed.

Since the set of variables and of actions of a HIOA are finite, U, X, Y, I, H, O are trivially computable from U_1, X_1, Y_1, U_2, X_2, Y_2, and from I_1, H_1, O_1, I_2, H_2, O_2, respectively.

By definition of composition, we have that $Inv = (Inv_1 \uparrow Z) \cap (Inv_2 \uparrow Z)$. Since Inv_1 and Inv_2 are closed, projection is a continuous computable function, and the intersection of closed sets is a computable closed set, Inv is closed and computable from Inv_1 and Inv_2. Similarly, by definition of composition we have that $Act = Act_1 \uparrow (A, Z) \cap Act_2 \uparrow (A, Z)$. Again, since Act_1 and Act_2 are closed, projection is a continuous computable function, and the intersection of closed sets is closed and computable, Act is closed and can be computed from Act_1 and Act_2.

To prove that Res is an upper-semicomputable function, it is sufficient to observe that *(i)* when $a \in A_1 \cap A_2$ then $Res(a, \mathbf{z}) = Res_1(a, \mathbf{z}|Z_1) \uparrow Z \cap Res_2(a, \mathbf{z}|V_2) \uparrow Z$, *(ii)* when $a \in A_1 \setminus A_2$ then $Res(a, \mathbf{z}) = Res_1(a, \mathbf{z}|Z_1) \uparrow Z \cap Res_2(\varepsilon, \mathbf{z}|Z_2) \uparrow Z$, and, finally *(iii)* when $a \in A_2 \setminus A_1$ then $Res(a, \mathbf{z}) = Res_1(\varepsilon, \mathbf{z}|Z_1,) \uparrow Z \cap Res_2(a, \mathbf{z}|Z_2) \uparrow Z$. Upper-semicontinuity of Res follows from upper-semicontinuity of Res_1 and Res_2. Computability of Res as a compact-valued function follows from the computability of projection and intersection of compact sets.

Finally, consider the dynamics $Dyn : \text{Val}(U) \times Inv \mapsto \text{Val}(Z)$. Since $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible, we have that the set Z_1 of locally-controlled variables of $\mathcal{A}_1$ is disjoint from the set Z_2 of locally-controlled variables of $\mathcal{A}_2$, and that $Z = Z_1 \cup Z_2$. Hence, we have that $Dyn(\mathbf{v}) = (Dyn_1(\mathbf{v}|V_1), Dyn_2(\mathbf{v}|V_2))$. Since projection and cartesian product are computable, Dyn is computable from Dyn_1 and Dyn_2. Further, we see that Dyn is locally-Lipschitz with linear growth, so has computable solutions.

When $\mathcal{A}_1$ and $\mathcal{A}_2$ are two compatible lower-semicontinuous HIOAs, lower-semicontinuity and computability of $\mathcal{A}$ can be proved in a similar way. □

From Theorem 5.10 we can conclude that the evolution of the composition of two upper-semicontinuous (resp., lower-semicontinuous) HIOAs is upper-semicomputable (resp., lower-semicomputable).

Theorem 5.11. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two compatible HIOAs, and suppose that the continuous dynamics Dyn_1 and Dyn_2 are locally-Lipschitz functions with linear growth.*

a) If $\mathcal{A}_1$ and $\mathcal{A}_2$ are upper-semicontinuous, then the evolution of $\mathcal{A}_1 \| \mathcal{A}_2$ is upper-semicomputable.
b) If $\mathcal{A}_1$ and $\mathcal{A}_2$ are lower-semicontinuous, then the evolution of $\mathcal{A}_1 \| \mathcal{A}_2$ is lower-semicomputable.

Proof. Direct consequence of Theorems 5.5 and 5.10. □

The requirement that the dynamics have linear growth and are locally-Lipschitz in Theorem 5.10 cannot be dropped, as the following examples show:

Example 5.12. *Suppose $\mathcal{A}_1$ has continuous dynamics $\dot{x}_1 = x_1 x_2$, and $\mathcal{A}_2$ has continuous dynamics $\dot{x}_2 = x_1 x_2$, and that $x_1(0) = x_2(0) = 1$. Then if x_2 is a continuous entire function $\mathbb{R}^+ \to \mathbb{R}$, then so is x_1, since $x_1(t) = \exp\left(\int_0^t x_2(\tau) d\tau\right)$. However, clearly the composed system has $x_1(t) = x_2(t) = x(t)$ where $\dot{x} = x^2$ and $x(0) = 1$, which has solution $x(t) = 1/(1-t)$ which is only defined for $t \in [0, 1)$.*

Define $s : \mathbb{R} \to \mathbb{R}$ *by* $s(x) = \sqrt{x}$ *if* $x \geqslant 0$ *and* $s(x) = -\sqrt{-x}$ *if* $x \leqslant 0$. *Suppose* $\dot{x}_1 = s(x_1) - s(x_1 - x_2)$ *and* $\dot{x}_2 = s(x_2) - s(x_2 - x_1)$. *If* x_2 *is a continuous entire function* $\mathbb{R}^+ \to \mathbb{R}$, *then it can be shown that so is* x_1. *However, if* $x_1(0) = x_2(0) = 0$, *then the composed system has* $x_1(t) = x_2(t) = x(t)$ *where* $\dot{x} = s(x)$ *and* $x(0) = 1$, *which has multiple solutions e.g.* $x(t) = 0$ *and* $x(t) = t^2/4$.

6 COMPOSITIONAL IMPLEMENTATIONS

The computational advantage of a compositional approach is the ability to decompose a system into open subsystems with lower complexity, and analyze those subsystems separately. The number of variables involved is reduced since the inputs of a given subsystem come exclusively from the external variables of other subsystems. In addition, using approaches such as assume-guarantee reasoning (see [5]) we can abstract the behavior of such inputs, in particular by modeling input traces using (hybrid) differential inclusions. Regarding the advantage in scalability by using decomposition and time-varying inputs, see also the work of [9] applied to the tool Flow*; while the paper deals with continuous spaces only, the methodology can be easily extended to the hybrid domain.

In particular, the framework described in this paper has been implemented in the open-source Ariadne library [15] for the analysis of nonlinear hybrid systems. The computational kernel of Ariadne is founded on the principles of Computable Analysis, meaning that it is possible to rigorously compute converging approximations either from outside or from inside. The current 1.0 stable version, referenced in the official website [35], allows to model systems as HIOAs. In this version the composed system must be closed in order to compute its evolution. Composition is performed statically, which allows for checking the correct definition of the whole system before any evolution is computed, but conversely does not scale well for large systems.

The development version of Ariadne 2.0, also reachable from the official website, uses instead on-the-fly composition. The current automaton model does not have an explicit notion of input/output variables and events: instead, the input/output character is deduced based on the product of the components. Work is under way on the modeling of inputs using differential inclusions: evolution for the continuous case is already available [21], while the extension to the hybrid case is next on schedule. Consequently, as soon as differential inclusions are handled in the hybrid domain in Ariadne, it will be possible to perform system decomposition and to provide an experimental evaluation of the proposed methodology.

While this paper and the corresponding implementation in Ariadne are the only works that additionally address computability, compositional approaches can be found in other tools as well. The SpaceEx tool specifically uses HIOAs (see [18] for the details) and allows for modularity and hierarchy. The MATLAB library called CORA uses its own compositional format and performs both static and on-the-fly composition; [27] gives an application of the methodology, even though their syntax and semantics have not been published. Finally, we care to mention HyPro [33]. The current implementation allows to describe hybrid automata compositionally, but the actual analysis routines are still under development. The proposed approach would analyze the components individually, neither computing the full product automaton nor composing locations on-the-fly. Instead, it would analyze locally and synchronise the analysis processes whenever required.

7 CONCLUDING REMARKS

Our objective in this paper has been to provide a theory for computable composition of HIOA (missing in the literature), as a prerequisite to develop a library grounded on a sound theory that exploits these results. Specifically, we considered a class of hybrid input/output automata and studied a compositional approach to obtain sets of hybrid traces that are amenable to approximate computations. To obtain well-posedness, we restricted the continuous dynamics of ordinary HIOAs to be defined in terms of differential equations based on locally Lipschitz functions with linear growth. After identifying the appropriate topologies, we introduced a compositional concept of upper and lower semicontinuity for HIOAs, which leads to upper and lower semicomputability, thus allowing us to address effectively the reachability problem on a large class of non-linear hybrid systems. As a side result we also identified a construction for the best upper and lower approximations of a HIOA.

Although this paper shows in a concise and effective way how to deal compositionally with computability within hybrid systems, it is our goal to extend the results to more general classes of Hybrid I/O Automata, e.g., by removing restictions, by identifying other compositional classes of dynamics, and by dealing with differential inclusions. We also aim to find conditions under which composition can be performed at the level of hybrid trajectory sets. We expect the layout outlined in this paper to be maintained, though intertwined with several further details that, if addressed at this stage, would hide the main idea that makes compositionality work. It is our objective to simplify the details so that our generalizations continue to keep the clean mathematical structure that we have obtained so far.

Finally, implementation of reachability analysis routines for hybrid open systems is under way in Ariadne. Future work will describe the implementation and demonstrate the advantage of the proposed compositional approach from the experimental viewpoint.

8 ACKNOWLEDGEMENTS

This work was partially supported by MIUR, Project "Italian Outstanding Departments, 2018-2022" and by INDAM, GNCS 2019, "Formal Methods for Mixed Verification Techniques".

The authors would like to thank Goran Frehse from the SpaceEx research group, Niklas Kochdumper and Matthias Althoff from the CORA research group, Erika Abraham and Stefan Schupp from the HyPro research group for the valuable feedback on the compositional capabilities of their tools.

REFERENCES

[1] D.E. Nadales Agut, D.A. van Beek, and J.E. Rooda. 2013. Syntax and semantics of the compositional interchange format for hybrid systems. *The Journal of Logic and Algebraic Programming* 82, 1 (2013), 1 – 52. https://doi.org/10.1016/j.jlap.2012.07.001

[2] Rajeev Alur, Radu Grosu, Insup Lee, and Oleg Sokolsky. 2006. Compositional modeling and refinement for hierarchical hybrid systems. *Journal of Logic and Algebraic Programming* 68, 1-2 (2006), 105–128. https://doi.org/10.1016/j.jlap.2005.10.004

[3] Rajeev Alur and Thomas Henzinger. 1997. Modularity for timed and hybrid systems. In *CONCUR '97: Proceedings of the 8th International Conference on Concurrency Theory*. Lecture Notes in Computer Science, Vol. 1243. Springer, Berlin, Heidelberg, 74–88.

[4] E. Asarin, T. Dang, O. Maler, and O. Bournez. 2000. Approximate Reachability Analysis of Piecewise-Linear Dynamical Systems. In *Proceedings of Hybrid Systems: Computation and Control (HSCC'00) (Lecture Notes in Computer Science)*, Vol. 1790. Springer, Berlin, Heidelberg, 20–31.

[5] L. Benvenuti, D. Bresolin, P. Collins, A. Ferrari, L. Geretti, and T. Villa. 2014. Assume-guarantee verification of nonlinear hybrid systems with Ariadne. *Int. J. Robust. Nonlinear Control* 24, 4 (2014), 699–724. https://doi.org/10.1002/rnc.2914

[6] O. Botchkarev and S. Tripakis. 2000. Verification of Hybrid Systems with Linear Differential Inclusions Using Ellipsoidal Approximations. In *Proceedings of Hybrid Systems: Computation and Control (HSCC'00) (LNCS)*, Vol. 1790. Springer, Berlin, Heidelberg, 73–88.

[7] Xin Chen, Sergio Mover, and Sriram Sankaranarayanan. 2017. Compositional Relational Abstraction for Nonlinear Hybrid Systems. *ACM Trans. Embed. Comput. Syst.* 16, 5s, Article 187 (Sept. 2017), 19 pages. https://doi.org/10.1145/3126522

[8] X. Chen and S. Sankaranarayanan. 2016. Decomposed Reachability Analysis for Nonlinear Systems. In *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, New York, NY, USA, 13–24. https://doi.org/10.1109/RTSS.2016.011

[9] X. Chen and S. Sankaranarayanan. 2016. Decomposed Reachability Analysis for Nonlinear Systems. In *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, New York, NY, USA, 13–24.

[10] A. Cimatti, S. Mover, and S. Tonetta. 2011. HyDI: A Language for Symbolic Hybrid Systems with Discrete Interaction. In *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, New York, NY, USA, 275–278. https://doi.org/10.1109/SEAA.2011.49

[11] Alessandro Cimatti, Marco Roveri, and Stefano Tonetta. 2015. HRELTL: A temporal logic for hybrid systems. *Information and Computation* 245 (2015), 54 – 71. https://doi.org/10.1016/j.ic.2015.06.006

[12] P. Collins. 2005. Continuity and computability of reachable sets. *Theoret. Comput. Sci.* 341 (2005), 162–195.

[13] Pieter Collins. 2008. *Semantics and Computability of the Evolution of Hybrid Systems*. Research Report MAS-R0801. CWI, Amsterdam, The Netherlands. ISSN 1386-3703.

[14] P. Collins. 2011. Semantics and Computability of the Evolution of Hybrid Systems. *SIAM Journal on Control and Optimization* 49, 2 (2011), 890–925. https://doi.org/10.1137/080716955 arXiv:https://doi.org/10.1137/080716955

[15] P. Collins, D. Bresolin, L. Geretti, and T. Villa. 2012. Computing the Evolution of Hybrid Systems Using Rigorous Function Calculus. In *Proc. of the 4th IFAC Conference on Analysis and Design of Hybrid Systems (ADHS12) (IFAC Proceedings Volumes)*, Vol. 45, Issue 9. Elsevier, Amsterdam, The Netherlands, 284–290.

[16] Pieter Collins and John Lygeros. 2005. Computability of finite-time reachable sets for hybrid systems. In *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, New York, NY, USA, 4688–4693.

[17] T. Dang and O. Maler. 1998. Reachability Analysis via Face Lifting. In *Proceedings of Hybrid Systems: Computation and Control (HSCC'98) (LNCS)*, Vol. 1386. Springer, Berlin, Heidelberg, 96–109.

[18] A. Donze and G. Frehse. 2013. Modular, hierarchical models of control systems in SpaceEx. In *2013 European Control Conference, ECC 2013*. IEEE, New York, NY, USA, 4244–4251.

[19] Alexandre Donzé and Oded Maler. 2010. Robust Satisfaction of Temporal Logic over Real-Valued Signals. In *FORMATS 2010: Formal Modeling and Analysis of Timed Systems (LNCS)*, Vol. 6246. Springer, Berlin, Heidelberg, 92–106.

[20] Georgios E. Fainekos and George J. Pappas. 2009. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science* 410, 42 (2009), 4262 – 4291. https://doi.org/10.1016/j.tcs.2009.06.021

[21] L. Geretti, S. Živanović Gonzalez, P. Collins, D. Bresolin, and T. Villa. 2019. Rigorous Continuous Evolution of Uncertain Systems. In *Proc. of NSV 2019: Numerical Software Verification (LNCS)*, Vol. 11652. Springer International Publishing, Cham, 60–75.

[22] N. Halbwachs, Y.-E. Proy, and P. Raymond. 1994. Verification of Linear Hybrid Systems by Means of Convex Approximations. In *Static Analysis Symposium*. Springer-Verlag, Berlin, Heidelberg, 223–237.

[23] Thomas Henzinger, Marius Minea, and Vinayak Prabhu. 2001. Assume-Guarantee Reasoning for Hierarchical Hybrid Systems. In *Hybrid Systems: Computation and Control*. Lecture Notes in Computer Science, Vol. 2034. Springer, Berlin, Heidelberg, 275–290.

[24] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. 1998. What's Decidable about Hybrid Automata? *J. Comput. System Sci.* 57, 1 (1998), 94 – 124. https://doi.org/10.1006/jcss.1998.1581

[25] Thomas A. Henzinger and Jean-François Raskin. 2000. Robust Undecidability of Timed and Hybrid Systems. In *HSCC 2000: Hybrid Systems: Computation and Control (LNCS)*, Vol. 1790. Springer, Berlin, Heidelberg, 145–159.

[26] A. B. Kurzhanski and P. Varaiya. 2000. Ellipsoidal Techniques for Reachability Analysis. In *Proceedings of Hybrid Systems: Computation and Control (HSCC'00) (LNCS)*, Vol. 1790. Springer, Berlin, Heidelberg, 202–214.

[27] H. L. Lee, M. Althoff, S. Hoelldampf, M. Olbrich, and E. Barke. 2015. Automated generation of hybrid system models for reachability analysis of nonlinear analog circuits. In *20th Asia and South Pacific Design Automation Conference, ASP-DAC 2015*. IEEE, New York, NY, USA, 725–730.

[28] Nancy Lynch, Roberto Segala, and Frits Vaandrager. 2003. Hybrid I/O automata. *Information and Computation* 185, 1 (2003), 105 – 157. https://doi.org/10.1016/S0890-5401(03)00067-1

[29] Nancy Lynch, Roberto Segala, Frits Vaandrager, and H. Weinberg. 1996. Hybrid I/O automata. In *Hybrid Systems III. HS 1995 (LNCS)*, Vol. 1066. Springer, Berlin, Heidelberg, 496–510. http://dx.doi.org/10.1007/BFb0020971 10.1007/BFb0020971.

[30] P. Nuzzo, A. L. Sangiovanni-Vincentelli, D. Bresolin, L. Geretti, and T. Villa. 2015. A Platform-Based Design Methodology with Contracts and Related Tools for the Design of Cyber-Physical Systems. *Proc. IEEE* 103, 11 (2015), 2104–2132. https://doi.org/10.1109/JPROC.2015.2453253

[31] Alessandro Pinto, Luca Carloni, Roberto Passerone, and Alberto Sangiovanni-Vincentelli. 2006. Interchange Formats for Hybrid Systems: Abstract Semantics. In *Hybrid Systems: Computation and Control (LNCS)*, Vol. 3927. Springer, Berlin, Heidelberg, 491–506. http://chess.eecs.berkeley.edu/pubs/101.html

[32] K. Ruohonen. 1996. An effective Cauchy-Peano existence theorem for unique solutions. *Internat. J. Found. Comput. Sci.* 7, 2 (1996), 151–160.

[33] S. Schupp, E. Abraham, I. B. Makhlouf, and S. Kowalewski. 2017. HyPro: A C++ library of state set representations for hybrid systems reachability analysis. In *Proc. of NFM 2017: NASA Formal Methods (LNCS)*, Vol. 10227. Springer, Berlin, Heidelberg, 288–294.

[34] B. I. Silva, O. Stursberg, B. H. Krogh, and S. Engell. 2001. An Assessment of the Current Status of Algorithmic Approaches to the Verification of Hybrid Systems. In *Proceedings of the Fortieth IEEE Conference on Decision and Control (CDC '01)*. IEEE, New York, NY, USA, 2867–2874.

[35] The Ariadne Team. 2020. Ariadne: an open library for formal verification of cyber-physical systems. http://www.ariadne-cps.org

[36] Stavros Tripakis, Christos Stergiou, Chris Shaver, and Edward A. Lee. 2013. A modular formal semantics for Ptolemy. *Mathematical Structures in Computer Science* 23, 4 (2013), 834–881. https://doi.org/10.1017/S0960129512000278

[37] D. A. van Beek, P. J. Collins, D. E. Nadales Agut, J. E. Rooda, and Schiffelers R. R. H. 2009. New Concepts In The Abstract Format Of the Compositional Interchange Format. In *Proc. of the 3rd IFAC Conference on Analysis and Design of Hybrid Systems (IFAC Proceedings Volumes)*, Vol. 42, Issue 17. Elsevier, Amsterdam, The Netherlands, 250–255.

[38] Klaus Weihrauch. 2000. *Computable analysis*. Springer-Verlag, Berlin. x+285 pages.