Recent results on computable and compositional semantics for hybrid systems

Davide Bresolin[1,*], Pieter Collins[2], Luca Geretti[3], Roberto Segala[3] and Tiziano Villa[3]

[1]University of Padova, Italy
[2]Maastricht University, The Netherlands
[3]University of Verona, Italy

Abstract

Hybrid systems combine discrete and continuous behaviors and are prevalent in applications such as automotive, robotics, and avionics, where precise interaction between control logic and dynamic environments is critical. This paper summarizes the results of [1], where a computable and compositional semantics for hybrid systems was first proposed. The formalism supports the composition of smaller subsystems and enables algorithmic analysis using computable functions. The approach addresses the challenge of undecidability in hybrid system reachability by employing approximate decision procedures. A key feature is the use of multifunctions to handle nondeterministic systems and ensure computability is preserved during system composition and the introduction of atomic interfaces to avoid circular dependencies in the composition. Several classes of computable multifunctions are identified to guarantee computational feasibility. This framework provides a scalable, compositional approach to the modeling and verification of complex hybrid systems.

Keywords

Hybrid Automata, Composition, Computability, Computable Analysis

1. Introduction

Hybrid systems—systems characterized by both discrete and continuous behaviors—are pervasive in real-world applications, such as automotive control systems, robotics, avionics, and process control, where precise interactions between a continuous environment and discrete control logic are critical. The intrinsic complexity of these systems is compounded by their size and interconnectivity, involving various components like controllers, sensors, actuators, and software, all interacting within a dynamic environment. This emphasizes the need for a *rigorous and compositional* approach to model and analyze hybrid systems effectively [2].

By decomposing complex systems into smaller, more manageable parts, a *compositional description* of a system allows each subcomponent to be analyzed independently and allows for *abstractions* at various levels of detail. For instance, high-level abstractions might focus on key system properties, while low-level abstractions may include more detailed implementation specifics. A compositional method aids in both understanding and verifying system behavior.

A central challenge in hybrid systems is the *reachability problem*, which involves determining whether a particular system state can be reached. Unfortunately, this problem is undecidable for most hybrid systems, except for timed automata and some generalizations [3]. Consequently, many approaches focus on *approximate decision procedures* and techniques for computing *approximations* of reachable sets [4, 5].

In [1], we have proposed a novel *behavioral formalism* for hybrid systems, which is both *computable* and *compositional*. This formalism is abstract and general, allowing it to serve as a semantic framework

OVERLAY 2024, 6th International Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis, November 28–29, 2024, Bolzano, Italy

[*] Corresponding author.

davide.bresolin@unipd.it (D. Bresolin); pieter.collins@maastrichtuniversity.nl (P. Collins); luca.geretti@univr.it (L. Geretti); roberto.segala@univr.it (R. Segala); tiziano.villa@univr.it (T. Villa)
0000-0003-2253-9878 (D. Bresolin); 0000-0002-8896-9603 (P. Collins); 0000-0001-6889-0706 (L. Geretti); 0000-0001-5586-3362 (R. Segala); 0000-0002-9671-8804 (T. Villa)

for various other modeling languages and formalisms proposed in the literature, including Hybrid Reactive Modules [6], Hybrid I/O Automata (HIOA) [7], and the Compositional Interchange Format [8]. The goal of this work was to support the *algorithmic analysis* of hybrid systems, which has been somewhat neglected in other formalisms that focus primarily on system decomposition.

The formalism emphasizes two key features: *compositionality* and *computability*. Compositionality refers to the ability to describe a large system as the *composition* of smaller, independently analyzed components. Computability refers to the use of *computable analysis*—a mathematical theory developed by K. Weihrauch and co-workers—to describe functions with a formalism that represents them with an increasing level of precision as computation evolves [9]. This theory is particularly useful in approximating the behavior of complex systems and ensuring that the results of such approximations are *computationally feasible*.

2. Behavioural Formalism

Our formalism extends the Hybrid I/O Automaton (HIOA) model by integrating it with *computable analysis*. The core idea of computable analysis is to describe the *evolution* of a system through functions that increase in precision as the computation progresses, allowing for approximate solutions to problems like reachability. A key feature of this approach is the introduction of *multifunctions*, which generalize traditional functions by mapping inputs to *sets of outputs* rather than single outputs. This is particularly useful for handling *nondeterministic systems*, where a single input might lead to multiple possible outcomes, and enables the *parallel composition* of multiple automata without losing computability.

Throughout the paper, we fix a countable universal set of *variables*, where every variable has a type that defines the domain of that variable. Elementary types include booleans, integers, and reals. Given a set of variables V, a *valuation* over V is a function that associates every variable in V with a value in its type. We denote by $\mathrm{Val}(V)$ the set of all valuations over V. A *trajectory* over a set of variables V is a function $\zeta : I \mapsto \mathrm{Val}(V)$, where I is a left-closed interval with left endpoint equal to 0. We say that a trajectory is *continuous* if it is a continuous function, and we denote by $\mathrm{CTrajs}(V)$ the set of all continuous trajectories over V. To capture the mixed nature of hybrid systems, we represent their evolution by *hybrid trajectories* alternating continuous trajectories with discrete *events*. The special *stuttering* event τ is used to represent hidden events externally generated by the environment. $\mathrm{HTrajs}(A, V)$ is the set of all hybrid trajectories over the set of events A and the set of variables V.

Definition 2.1 (Hybrid input/output automaton)**.** A *hybrid input/output automaton* is a tuple $H_{\mathrm{i/o}} = (I, O, U, Y, R_{\mathrm{i/o}}, F_{\mathrm{i/o}})$ where:

1. I and O are disjoint sets of *input* and *output* events. We take $A = I \cup O$;
2. U and Y are disjoint sets of *input* and *output* variables. We take $V = U \cup Y$;
3. The *discrete input/output behaviour* is a multifunction
$$R_{\mathrm{i/o}} : \mathrm{Val}(V) \times (A \cup \{\tau\}) \times \mathrm{Val}(U) \rightrightarrows \mathrm{Val}(Y);$$
4. The *continuous input/output behaviour* is a multifunction
$$F_{\mathrm{i/o}} : \mathrm{Val}(Y) \times \mathrm{CTrajs}(U) \rightrightarrows \mathrm{CTrajs}(Y)$$
respecting some natural prefix- and suffix- closure conditions detailed in [1].

In Item 3 of the definition above, saying that $\mathbf{y} \in R_{\mathrm{i/o}}(\mathbf{v}, a, \mathbf{u})$ means that $\mathbf{y}$ is a valid output after the discrete transition from $\mathbf{v} \in \mathrm{Val}(V)$, with event a, and where $\mathbf{u} \in \mathrm{Val}(U)$ are the values of the input variables after the transition. In Item 4, saying that $\eta \in F_{\mathrm{i/o}}(\mathbf{y}, \upsilon)$ means that η is a valid output trajectory for input trajectory υ with initial value $\mathbf{y}$.

The *behaviour* of $H_{\mathrm{i/o}}$ is the map $B_{\mathrm{i/o}} : \mathrm{Val}(Y) \times \mathrm{HTrajs}(I, U) \rightrightarrows \mathrm{HTrajs}(O, Y)$ such that $\beta \in B_{\mathrm{i/o}}(\mathbf{y}, \alpha)$ if, and only if, β is an output of $H_{\mathrm{i/o}}$ for input α with initial value $\mathbf{y}$. In contrast to other formalisms, our approach *preserves computability* even when systems are composed of several interconnected components. In [1], we demonstrated that the *behavioral semantics* is compositional, that is, that the behaviour of the composition of two HIOAs is exactly the composition of the behaviours of the two components.

3. Challenges in System Composition: Circular Dependencies

When composing hybrid systems, one of the major challenges is avoiding *circular dependencies* between variables. Circular dependencies arise when two systems or components impose mutually dependent constraints on each other's variables, which can lead to *non-converging loops* and prevent meaningful system composition.

To address this, we introduced a notion of *interfaces* between components, which explicitly describe the *dependencies between variables.* By enforcing a condition that ensures *compatibility* between composed automata (i.e., avoiding circular dependencies), our work demonstrates that it is possible to compute the *discrete* and *continuous dynamics* of a composed system without running into computational issues. We solved the problem of circular dependencies by the introduction of *atomic interfaces*, which break down interfaces into smaller, simpler components. This decomposition simplifies the process of checking for compatibility between systems and ensures that their composition is *well-structured* and *computationally feasible.*

An atomic component is unidirectional, meaning that its output variables may depend on its input variables, though none of its input variables may depend on any of its output variables. In other words, there is no feedback. Two components can be composed together (are *compatible*) if the composition does not introduce any circular dependencies. In particular, two atomic components $H_{\mathrm{i/o}_1}$, $H_{\mathrm{i/o}_2}$ can be composed together if for one of the two, say $H_{\mathrm{i/o}_2}$, its output variables do not intersect the input variables of the other. Then, the behavior of the composition can be obtained by computing first the output of $H_{\mathrm{i/o}_1}$, which depends only on the input of $H_{\mathrm{i/o}_1}$, and then computing the output of $H_{\mathrm{i/o}_2}$, which may depend on the output of $H_{\mathrm{i/o}_1}$ as well. We call this operation *dependent product.* It turns out that any collection of atomic components can be totally ordered so that its behavior can be computed by applying the dependent product to the first two components, then applying the dependent product to the result and to the third component, and so on. Therefore, everything reduces to computing the dependent product of two components. In particular, whenever we deal with computability, it suffices to show that the dependent product is a computable operator.

We say that a hybrid input/output automaton has an atomic decomposition if its behavior is the same as the behavior of the composition of a collection of atomic components, called an *atomic decomposition.* We say that two automata are compatible (can be composed) if they have compatible atomic decompositions. It turns out that the union of the atomic decompositions of two automata is an atomic decomposition of the composition of the two automata. Therefore, the behavior of any complex automaton can be computed in terms of dependent products of atomic decompositions. Furthermore, the atomic decompositions are known by construction whenever we compose automata with known atomic decompositions.

An important caveat is the definition of dependency. Intuitively, a variable y does not depend on a collection of variables U if there is no way to change the values of variables in U so that the value of y changes. This concept is straightforward for discrete transitions. When we move to trajectories, the dependencies involve time as well. Therefore, changing an input trajectory may affect output trajectories in the past, in the present, and in the future. Changing the past is unfeasible, but changing the present and the future is feasible. In particular, it is common within control theory to distinguish between input, state, and output variables (u, x, y) and specify the behavior of a system in terms of equations of the form $x' = f(u, x)$ and $y = g(x)$. In this case, the dependency between u and y is only on the future and is bound by a differential equation. We know that, under appropriate conditions on f, circular dependencies are not a problem in such cases. To address the issue, we distinguish between *algebraic* and *integral* dependencies, where informally, algebraic dependency deals with the present while integral dependency deals with the future. Our case above is an example of integral dependence.

Then, we impose no circularity only on the algebraic dependencies. Of course, this means showing how it is possible to compute a dependent product under integral dependencies. We do so via fixpoint operators.

4. Computable Multifunctions for Hybrid Systems

The second main contribution of our work is the identification of *classes of multifunctions* that are *computable* when systems are composed by establishing several conditions under which the dependent product and the fixpoint operators remain computable. These conditions include:

- *Compactness*: The functions are bounded in such a way that they can be handled with finite computational resources. We also considered the logically dual condition of *overtness* and that of *locatedness*, which rules out pathological counterexamples that exist for logical rather than topological reasons.
- *Lipschitz continuity*: The functions have a bounded rate of change, ensuring that small changes in input lead to proportionally small changes in output.
- *Linear growth*: The growth of the functions is limited by a linear function, ensuring that the computations remain tractable over time.
- *Upper Semicontinuity*: Prevents downward jumps of the function but allows upward jumps, ensuring that the functions can be approximated from above.
- *Lower Semicontinuity*: Prevents upward jumps of the function but allows downward jumps, ensuring that the functions can be approximated from below.

By focusing on specific combinations of these classes of functions, we showed that it is possible to build hybrid systems that are not only theoretically sound but also *computationally tractable.*

To prove computability of the composition of hybrid behaviours, we first showed that the composition of purely discrete behaviours can be computed via the discrete dependent product.

Theorem 4.1. *Suppose that $H_{\mathrm{i}/\mathrm{o}_1}$ and $H_{\mathrm{i}/\mathrm{o}_2}$ are hybrid components with compatible atomic decompositions. Then the composition $R_{\mathrm{i}/\mathrm{o}_1} \| R_{\mathrm{i}/\mathrm{o}_2}$ of the discrete input/output behaviours is computable for the following classes of multifunctions.*

(a) Upper-semicontinuous compact-valued functions with nonempty values.
(b) Lower-semicontinuous overt-valued functions with nonempty values.
(c) Continuous located-valued functions with nonempty values.
(d) Continuous singleton-valued functions over closed (respectively, open) sets.

This follows because the parallel composition can be computed as the (discrete) dependent product of the input/output behaviours of the components. For the classes of multifunctions considered, the dependent product for multifunctions is computable (and yields a multifunction of the same class.)

Then, we showed that the composition of continuous systems can be computed using the continuous dependent product.

Theorem 4.2. *Suppose that $H_{\mathrm{i}/\mathrm{o}_1}$ and $H_{\mathrm{i}/\mathrm{o}_2}$ are hybrid components with compatible atomic decompositions. Then the composition $F_{\mathrm{i}/\mathrm{o}_1} \| F_{\mathrm{i}/\mathrm{o}_2}$ of the continuous input/output behaviours is computable for the following classes of multifunctions.*

(a) Compact-valued effectively bounded-equicontinuous upper-semicontinuous functions with nonempty values and locally linear growth.
(b) Overt-valued effectively locally Lipschitzian functions with nonempty values.
(c) Located-valued effectively locally Lipschitzian functions with nonempty values.
(d) Singleton-valued effectively locally-Lipschitzian functions.
(e) Singleton-valued effectively locally-Lipschitzian functions with locally linear growth and entire solutions.

This follows because the parallel composition of continuous input/output behaviours can be computed using only the operations of dependent product for multifunctions and the fixed-point operator for flows. For the classes considered, we proved that the dependent product for multifunctions and the fixed-point operator for flows is computable.

The two theorems above allowed us to prove our two main theorems. The first one showed that for classes of discrete and continuous systems, corresponding to the cases in Theorem 4.2, the composition of compatible discrete and continuous input/output behaviours can be computed, and thus that we can compute the hybrid behaviour of a hybrid component. The second main result proved that from the behaviour, we can extract the discrete and continuous parts, though some of the meta-information on the continuous dynamics may be lost.

In conclusion, since 1) the automata obtained by extracting the discrete- and continuous- behaviours of the corresponding atomic hybrid input-output automata are composable, and 2) the behaviour of the composed automaton is computable, we conclude that the resulting behaviour satisfies the same effectivity properties as the component behaviours, as these are preserved by each operation.

5. Conclusion and Future Work

In conclusion, in [1], we presented a *computable and compositional semantics* for hybrid systems, offering a powerful framework for describing and analyzing complex systems that combine discrete and continuous behaviors. By focusing on *computability* and *compositionality*, we provided a solution to the challenges of modeling and verifying large hybrid systems.

The formalism is general enough to be applied to various other hybrid system models, and it constitutes the theoretical foundation of the software package Ariadne, which we developed for the formal verification of hybrid systems [10, 11, 12]. To support the compositional description and analysis of systems, in Ariadne, a distinction between input, output, and internal variables and actions has been introduced. A component can define the flows of its internal and output variables only and the guards and reset functions of its internal and output actions only. This restriction guarantees that there are no algebraic loops in the definition of the flows nor in the definitions of guards and invariants and, thus, that the evolution of the composition can be computed.

Besides its application to Ariadne mentioned above, it should not be difficult to adapt our results to existing formalisms like [13, 14, 15, 16, 17, 18, 19, 20, 21].

We expect as well our compositionality results to be amenable to further extensions. Indeed, besides the minimal conditions required on the underlying operational model, part of the conditions are based on the way the set of continuous dynamics are specified and, in particular, how they are formulated by adapting or extending known results from differential analysis. In this paper, we focused on a specific way to formulate differential equations, and we relied on Lipschitz assumptions. However, other known theories can be adapted seemingly to our framework. In summary, we hope the framework introduced in this paper will be the basis for linking approaches to the analysis of hybrid systems that otherwise appear to be much further apart.

Acknowledgments

This work was supported by the National Recovery and Resilience Plan (NRRP) Project "Securing sOftware Platforms - SOP", CUP H73C22000890001, and by the INdAM-GNCS 2024 Project “Certificazione, monitoraggio, ed interpretabilità in sistemi di intelligenza artificiale”, CUP E53C23001670001.

References

[1] D. Bresolin, P. Collins, L. Geretti, R. Segala, T. Villa, S. Živanović Gonzalez, A computable and compositional semantics for hybrid systems, Information and Computation 300 (2024) 105189. doi:https://doi.org/10.1016/j.ic.2024.105189.

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

[3] T. A. Henzinger, P. W. Kopke, A. Puri, P. Varaiya, What's decidable about hybrid automata?, Journal of Computer and System Sciences 57 (1998) 94 – 124. doi:10.1006/jcss.1998.1581.

[4] B. I. Silva, O. Stursberg, B. H. Krogh, S. Engell, 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, 2001, pp. 2867–2874.

[5] R. Alur, Formal verification of hybrid systems, in: Proc. of the 9th ACM International Conference on Embedded Software, EMSOFT '11, ACM, New York, NY, USA, 2011, pp. 273–278. doi:10.1145/2038642.2038685.

[6] R. Alur, T. Henzinger, Modularity for timed and hybrid systems, in: CONCUR '97: Proceedings of the 8th International Conference on Concurrency Theory, volume 1243 of *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 1997, pp. 74–88.

[7] N. Lynch, R. Segala, F. Vaandrager, Hybrid I/O automata, Information and Computation 185 (2003) 105 – 157. doi:10.1016/S0890-5401(03)00067-1.

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

[9] K. Weihrauch, Computable analysis, Texts in Theoretical Computer Science. An EATCS Series, Springer-Verlag, Berlin, 2000.

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

[11] S. Živanović Gonzalez, L. Geretti, D. Bresolin, T. Villa, P. Collins, A higher order method for input-affine uncertain systems, Nonlinear Analysis: Hybrid Systems 47 (2023) 101266.

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

[13] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, O. Maler, SpaceEx: Scalable Verification of Hybrid Systems, in: G. Gopalakrishnan, S. Qadeer (Eds.), Computer Aided Verification, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 379–395.

[14] O. Bouissou, A. Chapoutot, S. Mimram, Computing flowpipe of nonlinear hybrid systems with numerical methods, arXiv preprint arXiv:1306.2305 (2013).

[15] X. Chen, E. Ábrahám, S. Sankaranarayanan, Flow*: An analyzer for non-linear hybrid systems, in: N. Sharygina, H. Veith (Eds.), Computer Aided Verification, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 258–263.

[16] M. Althoff, J. M. Dolan, Online verification of automated road vehicles using reachability analysis, IEEE Transactions on Robotics 30 (2014) 903–918. doi:10.1109/TRO.2014.2312453.

[17] R. Ray, A. Gurung, Parallel State Space Exploration of Linear Systems with Inputs Using XSpeed, in: Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC '15, ACM, New York, NY, USA, 2015, pp. 285–286. doi:10.1145/2728606.2728644.

[18] E. Möhlmann, W. Hagemann, A. Rakow, Verifying a PI controller using SoapBox and Stabhyli, in: ARCH@ CPSWeek, 2016, pp. 115–125.

[19] S. Bak, P. S. Duggirala, Simulation-equivalent reachability of large linear systems with inputs, in: R. Majumdar, V. Kunčak (Eds.), Computer Aided Verification, Springer International Publishing, Cham, 2017, pp. 401–420.

[20] S. Bogomolov, M. Forets, G. Frehse, K. Potomkin, C. Schilling, JuliaReach: A toolbox for set-based reachability, in: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC '19, ACM, New York, NY, USA, 2019, pp. 39–44. doi:10.1145/3302504.3311804.

[21] S. Schupp, E. Ábrahám, T. Ebert, Recent developments in theory and tool support for hybrid

systems verification with HyPro, Information and Computation (2022) 104945. doi:https://doi.org/10.1016/j.ic.2022.104945.