pyHybridAnalysis: a Package for ϵ-Semantics Analysis of Hybrid Systems

Alberto Casagrande
Dept. of Mathematics and Geosciences
University of Trieste, Trieste, Italy
e-mail: acasagrande@units.it

Tommaso Dreossi
Dept. of Mathematics and Computer Science
University of Udine, Udine, Italy
VERIMAG
Joseph Fourier University, Grenoble, France
e-mail: tommaso.dreossi@uniud.it

***Abstract*—Hybrid automata naturally represent systems that exhibit a mixed discrete-continuous behaviours. The undecidability of the reachability problem over them constrains the chances of punctually investigating this kind of formalism. Established that this negative result and the presence of artifacts, which do not correspond to any observable phenomena, are mainly due to the density of the continuous domain, a class of finite precision semantics, named ϵ-semantics, has been proposed to analyze hybrid automata. This paper presents a Python package, `pyHybridAnalysis`, that both implements the ϵ-semantics framework and allows to analyze hybrid automata.**

***Keywords*—Hybrid Automata, First-Order Theory, ϵ-Semantics**

I. Introduction

A wide range a phenomena manifests behaviors that are not completely describable neither with continuous models, nor with discrete models. These phenomena are often abstract in terms of *hybrid systems*, i.e., dynamical systems capable of evolving both continuously and discretely. *Hybrid automata* are frequently used to describe such systems. They consist mainly of a discrete finite automaton equipped with a set of continuous variables regulated by dynamical laws.

Hybrid automata have been used to model air traffic control systems [1], vehicle controls [2], or biological systems [3]. It is clear that the automatic deduction of properties for these models is an important part in the study and verification of hybrid systems. These concepts are strictly related to the *reachability problem* where, provided an hybrid automaton and an initial set of states, we ask whether there are evolutions of the system that lead to some final states. Unfortunately, it has been shown that such problem is not always decidable [4].

Numerous techniques have been proposed to address the undecidability related to the reachability problem. Syntactical restrictions [5], [6], approximations [7] and perturbations of the dynamics [8], relaxed (bi-)simulations [9] and abstractions [10], are just some examples of the various stratagems designed. Some of them have been exploited in the development of tools for the reachability analysis of hybrid systems, such as `d/dt` [11], `Ariadne` [12] and `Hsolver` [13].

In this work we present a new tool for the formal verification of hybrid systems, named `pyHybridAnalysis`, based on the manipulation of first-order logic formulæ. Starting from the assumption of working with hybrid automata whose components are describable by first-order formulæ, our tool translates the reachability problem into a formulæ satisfiability problem. The encoding process is enhanced by the reinterpretation of the standard semantics of the formulæ involved. In doing so, our framework is able of countering the undecidability of the reachability problem and enriches the descriptive power of hybrid automata taking into account the actions of the external agents that often perturb the univocal behaviors dictated by the dynamic laws. What we obtain is a tool based on symbolic computations that finds application in the analysis and verification of hybrid systems.

The paper is organized as follows. Section II recalls the basic definitions concerning logics and hybrid automata. Section III exposes the techniques of the semantics reinterpretation focusing on the theoretical definition. Section IV points out some relevant aspects of the implementation of our tool in Python, such as the modules designed to represent formulæ and hybrid automata. Finally, Section V ends the paper with some concluding remarks and general comments.

II. Hybrid Automata and Reachability

In this work we adopt the notions and definitions of logic, theories and hybrid automata given in [6]. Intuitively, a hybrid automaton is a tuple $H = \langle \mathbf{X}, \mathbf{X}', T, \mathcal{V}, \mathcal{E}, Inv, Dyn, Act, Res \rangle$ where, for any location $v \in \mathcal{V}$, the formula $Inv(v)$, called *invariant*, delimits the set of continuous values in which the variables of the automaton can evolve inside v. Such evolution is performed accordingly to the *dynamic* formula $Dyn(v)$. The discrete evolutions between the locations are regulated by the *activation* formulæ $Act(e)$, that identify the set of values from which the automaton can jump over an edge e, and the *resets* $Res(e)$, which consist in maps to be applied to the continuous variables after performing a discrete jump. If $Inv(v)$, $Dyn(v)$, $Act(e)$, and $Res(e)$ belong to the same logic theory $\mathcal{T}$, then we say that the hybrid automaton is a $\mathcal{T}$ hybrid automaton. In this work, since the polynomials allow us to capture a considerably part of the hybrid automata aimed at the formalization of real hybrid systems, we will focus on semi-algebraic automata, i.e., those hybrid automata whose components are describable by formulæ definable in the Tarski's theory.

Since all the components of our formalism are given in terms of first-order formulæ, it is possible to define a

This work has been partially supported by Istituto Nazionale di Alta Matematica (INdAM).

formula $Reach_H^i[\mathbf{X}, \mathbf{X}']$ that is satisfiable if and only if in the automaton H there exists a point $\mathbf{X}$ able to reach a point $\mathbf{X}'$ within i discrete transitions. Note anyway that since the state space of a hybrid automaton is potentially infinite, the exploration of its whole reachability set might require an infinite number of discrete steps. This kind of evolutions require formulæ composed of an infinite number of conjuncts, that is, formulæ practically intractable. In general, it is well known that the problem of determining the reachability set of a hybrid automaton is undecidable [4].

III. Changing Semantics

At the basis of the undecidability of the reachability problem there are the infinity and the density of the space provided by the continuous variables. One might suspect that placing constraints on these two characteristics, there could be some benefits on the reachability analysis. These constraints would correspond to working only with bounded invariants and finding a way to discretize the sets represented by the formulæ. In [14] the authors noticed that such process of discretization does not represent a loss of the descriptive power of hybrid automata, but which indeed, especially in the modeling of natural systems, it better reflects the real behavior of the observed systems. We therefore define a technique which allows the discretization of the sets represented by first-order formulæ. Since we want to alter the standard semantics of the considered formulæ and fix a threshold precision, we will define our discretization process in terms of a new semantics, called ϵ-*semantics*, where ϵ represents the granularity of the formulæ. Intuitively, this new semantics does not consider the formulæ which are "smaller" than an fixed ϵ. This reflects the concept of finite precision measurement accuracy that we need.

Definition 1 (ϵ-semantics [14])**.** *Let $\epsilon \in \mathbb{R}_{>0}$. For each formula $\varphi \in \mathcal{F}_n$ let $\{\!|\varphi|\!\}_\epsilon \subseteq \mathbb{R}^n$, be such that:*

(ϵ) $\{\!|\varphi|\!\}_\epsilon = \varnothing$ *or exists* $p \in \mathbb{R}^n$ *s.t.* $B(p,\epsilon) \subseteq \{\!|\varphi|\!\}_\epsilon$

$(\cap)$ $\{\!|\varphi_1 \wedge \varphi_2|\!\}_\epsilon \subseteq \{\!|\varphi_1|\!\}_\epsilon \cap \{\!|\varphi_2|\!\}_\epsilon$

$(\cup)$ $\{\!|\varphi_1 \vee \varphi_2|\!\}_\epsilon = \{\!|\varphi_1|\!\}_\epsilon \cup \{\!|\varphi_2|\!\}_\epsilon$

$(\forall)$ $\{\!|\forall X \varphi[X, \mathbf{X}]|\!\}_\epsilon = \{\!|\bigwedge_{r\in\mathbb{R}} \varphi[\![r, \mathbf{X}]\!]|\!\}_\epsilon$

$(\exists)$ $\{\!|\exists X \varphi[X, \mathbf{X}]|\!\}_\epsilon = \bigcup_{r\in\mathbb{R}} \{\!|\varphi[\![r, \mathbf{X}]\!]|\!\}_\epsilon$

$(\neg)$ $\{\!|\varphi|\!\}_\epsilon \cap \{\!|\neg\varphi|\!\}_\epsilon = \varnothing$

Any semantics $\{\!|\cdot|\!\}_\epsilon$ satisfying the above conditions is said to be an ϵ-semantics*.*

Let us notice that the above definition implicitly depends on the metric space in which we build the ball $B(p,\epsilon)$. Although the theory allows us to work in any metric space, unless otherwise stated, we adopt the Euclidean space.

The ϵ-semantics can be used within an algorithm that computes the reachability set of hybrid automata. The basic idea is to explore a hybrid automaton until the difference between the sets of states reached by two consecutive discrete transitions is smaller than an ϵ-sphere. When the automaton reaches for the first time a set which is smaller than the granularity of the ϵ-semantics, the algorithm halts and returns the formula characterizing the automaton reachability set.

Algorithm 1 is a summary of the complete algorithm presented in [14]. Intuitively, at the i-th iteration of the algorithm,

Algorithm 1 Reachability$(H, I[\mathbf{X}], \{\!|\cdot|\!\}_\epsilon)$

1: $R[\mathbf{X}] \leftarrow I[\mathbf{X}]$
2: $N[\mathbf{X}] \leftarrow \bot$
3: **repeat**
4: $\quad R[\mathbf{X}] \leftarrow R[\mathbf{X}] \vee N[\mathbf{X}]$
5: $\quad N[\mathbf{X}] \leftarrow \exists \mathbf{X}'(R[\![\mathbf{X}']\!] \wedge Reach_H^{\leq 1}[\![\mathbf{X}', \mathbf{X}]\!])$
6: **until** $\{\!|N[\mathbf{X}] \wedge \neg R[\mathbf{X}]|\!\}_\epsilon \neq \varnothing$
7: **return** $R[\mathbf{X}]$

the variable $R[\mathbf{X}]$ represents the reachability formula which encodes i discrete steps through the automaton H. The variable $N[\mathbf{X}]$ consists of the formula stored in $R[\mathbf{X}]$ extended by a single discrete step. The ϵ-semantics find application in the halting criterion of the algorithm (line 6), where it is verified whether the difference between the sets represented by $R[\mathbf{X}]$ and $N[\mathbf{X}]$ is smaller than an ϵ-sphere. An important aspect of this algorithm is that, in the case of hybrid automata with bounded invariants, its termination is guaranteed [14].

Definition 1 settles a generic framework of ϵ-semantics. This means that in the Algorithm 1 we have the freedom to instantiate any kind of semantics that respects the constraints imposed by the definition of ϵ-semantics. In this regard, we now expose an example of ϵ-semantics. The *sphere semantics* $(\!|\cdot|\!)_\epsilon$ [14] is an ϵ-semantics which is neither an over nor an under-approximation semantics. The set $(\!|\varphi|\!)_\epsilon$, where $\epsilon \in \mathbb{R}_{>0}$, is defined by structural induction on φ as follows:

- $(\!|t_1 \circ t_2|\!)_\epsilon \stackrel{def}{=} B(\{\!|t_1 \circ t_2|\!\}, \epsilon)$, for $\circ \in \{=, <\}$
- $(\!|\varphi_1 \wedge \varphi_2|\!)_\epsilon \stackrel{def}{=} \bigcup_{B(p,\epsilon)\subseteq (\!|\varphi_1|\!)_\epsilon \cap (\!|\varphi_2|\!)_\epsilon} B(p,\epsilon)$
- $(\!|\varphi_1 \vee \varphi_2|\!)_\epsilon \stackrel{def}{=} (\!|\varphi_1|\!)_\epsilon \cup (\!|\varphi_2|\!)_\epsilon$
- $(\!|\forall X \varphi[X, \mathbf{X}]|\!)_\epsilon \stackrel{def}{=} \bigcup_{B(p,\epsilon)\subseteq \bigcap_{r\in\mathbb{R}} (\!|\varphi[\![r,\mathbf{X}]\!]|\!)_\epsilon} B(p,\epsilon)$
- $(\!|\exists X \varphi[X, \mathbf{X}]|\!)_\epsilon \stackrel{def}{=} \bigcup_{r\in\mathbb{R}} (\!|\varphi[\![r, \mathbf{X}]\!]|\!)_\epsilon$
- $(\!|\neg\varphi|\!)_\epsilon \stackrel{def}{=} \bigcup_{B(p,\epsilon)\cap (\!|\varphi|\!)_\epsilon = \varnothing} B(p,\epsilon)$

IV. A Python Package

With the intent of using the techniques presented in Section III and verifying properties of hybrid systems, we developed a Python package named `pyHybridAnalysis`. Beside the well-known features of this language, we chose Python to reduce the effort required to describe a hybrid automaton. As a matter of fact, the chance of handling functions, types, and data uniformly enables us to use a syntax almost identical to the one introduced in Section II.

`pyHybridAnalysis` provides support for representing first-order formulæ over the reals. Variables are defined as objects of the class `Variable` and all the functional and relational symbols used to define equalities and inequalities in the Tarski's theory extended with the exponential have been overloaded. Because of this, equalities and inequalities can be encoded by using the standard Python syntax. For instance, $\phi \stackrel{def}{=} X^2 + 3 * Y \geq 0$ and $\psi \stackrel{def}{=} X + Y = \frac{3}{5}$ are definable as:

```
from pyHybridAnalysis import *

X=Variable("X")
Y=Variable("Y")
```

```
phi=X**2+3*Y>=0
psi=X+Y==3/5
```

The boolean relational symbols cannot be overloaded in Python, hence, conjunctions, disjunctions, negations, and implications are built by constructors of the classes `And`, `Or`, `Not`, and `Impl`, respectively. The same treatment is reserved to existential and universal quantifications whose corresponding classes are `Exists` and `Forall`, respectively. In these latter cases, the first parameter should be a list of the variables that are meant to be quantified. A representation of the formula $\eta \stackrel{def}{=} \neg\exists Y\, phi \twoheadrightarrow psi$ can be easily obtained by the code:

```
eta=Not(Exists([Y],Impl(phi,psi)))
```

All the objects obtained by representing a formula as described above are instances of subclasses of the class `Formula`. This class provides methods to visually represent formulæ, identify formula variables, and establish free variables. We can print the existential closure of η by writing:

```
print Exists(eta.free_vars(),eta)
```

The notion of `SyntacticOperator` has been introduced to represent functions that operate recursively on the syntactic tree of formulæ. They take a first-order formula ϕ and recursively build a second formula whose standard semantics is equivalent to the ϵ-semantics of ϕ itself. Any ϵ-semantics is associated with a very specific translation and we have implemented the classes `SphereTranslator`, `TildeTranslator`, `EasyTranslator`, and `BottomTranslator` to provide translators for sphere semantics [14], tilde semantics, easy semantics, and bottom semantics [15]. Since all the ϵ-semantics are parametric on both ϵ and the underlying metric space, the translator constructors should receive as parameters a value for ϵ and, in case, a metric space different from the Euclidean one. For instance, in order to evaluate the sphere semantics of η with $\epsilon = 0.01$ in the maximum metric space, we may write:

```
print SphereTranslator(0.01,MaxSpace)(eta)
```

In addition to this, `pyHybridAnalysis` provides a easy-to-use interface to the quantifier elimination procedures of both `Redlog` and `QEPCAD B`. In particular, in order to use `Redlog` and eliminate the quantification of η, it is sufficient to invoke the following line:

```
gamma=Redlog()(eta)
```

Hybrid automata are represented as objects of the class `HAutomata`. This class is equipped with a single constructor which requires four parameters: the invariants, the dynamics, the resets, and the activations. The invariants and the dynamics are dictionaries from the set of locations of the automaton, while the resets and the activations are dictionaries from the set of edges. In order to better mime the syntax used in Section II, the values associated to these dictionaries are Python methods. If `Dyn` represents the dynamics and l is a location, `Dyn[l]` is a method that takes three parameters (i.e., $\mathbf{X}$, $\mathbf{X}'$, and T) and returns a Python representation of the formula $Dyn(l)[\mathbf{X}, \mathbf{X}', T]$. Analogously, if `Inv` represents the invariant, `Inv[l]` takes one parameter (i.e., $\mathbf{X}$) and produces the formula $Inv(l)[\mathbf{X}]$. This convention enables us to easily compute the formula $Dyn(l)[\![\langle 2,3\rangle, \langle X, Y\rangle, T]\!]$ by writing:

```
print Dyn[l]([2,3],[X,Y],Variable("T"))
```

The edges are represented as objects of the class `Edge` and they can be built by the constructor of `Edge` which mandatory takes as parameters the source and destination locations and, optionally, a label to distinguish two edges that share the same sources and destinations, but have different activations or resets. For instance, the code

```
e0=Edge(0,1,"First")
e1=Edge(0,1,"Second")
```

defines two edges from the location 0 to the location 1 which can be distinguished exclusively by their labels. Given an object `H` of the class `HAutomaton`, we can get the sets of its locations and edges by using the methods `H.locations()` and `H.edges()`, respectively, moreover, if `l` is a location of `H`, `H.edges(l)` returns the set of edges leaving `l`. The dynamics, invariants, activations, and resets of `H` are reachable as `Dyn(H)`, `Inv(H)`, `Act(H)`, and `Reset(H)`.

We can define a hybrid reach set as an object of the class `HybridReachSet`. This maps a location into the formula representing the continuous points reached in that location. The class `EpsilonReachability` implements Algorithm 1. In order to use it, it is necessary to instantiate an object of this class. Since the algorithm does not relay on a specific ϵ-semantics, but it can be applied to all of them, the constructor of `EpsilonReachability` requires as parameter an object of class `EpsilonTranslator` to specify the desired ϵ-semantics. A further parameter can be used to obtain a log of the execution, since no output is produced by default. Given an instance `er` of the class `EpsilonReachability`, the implementation of Algorithm 1 can be invoked by writing:

```
er.compute_reachset(H,init)
```

where `H` is a hybrid automaton and `init` is the initial hybrid reach set.

```
#!/usr/bin/python

import sys
from pyHybridAnalysis import *

epsilon=0.1
metric_space=MaxSpace

a=Variable("a")
b=Variable("b")

init=And(a==0.3,b==0)

slope=[3.05/4,1]

lbound=len(slope)*[0]
ubound=len(slope)*[1]

# Invariants
def Inv_v(vars):
    return in_itvl(vars,lbound,ubound)

# Dynamics
def Dyn_v(src,dst,t):
```

```
    return And(dst[1]==src[1]+t*slope[1],
               dst[0]==src[0]+t*slope[0])

# Activations
def Act_e0(vars):
    return vars[0]==ubound[0]

def Act_e1(vars):
    return vars[1]==ubound[1]

# Resets
def Reset_e0(src,dst):
    return And(dst[0]==lbound[0],
               dst[1]==src[1])

def Reset_e1(src,dst):
    return And(dst[0]==src[0],
               dst[1]==lbound[1])

Invs=dict()
Dyns=dict()
Resets=dict()
Acts=dict()

Invs[0]=Inv_v
Dyns[0]=Dyn_v

e0=Edge(0,0,"Reset a")

Resets[e0]=Reset_e0
Acts[e0]=Act_e0

e1=Edge(0,0,"Reset b")

Resets[e1]=Reset_e1
Acts[e1]=Act_e1

H=HAutomaton(Dyns,Invs,Resets,Acts)

init=HybridReachSet(H,init)

trans=EasyTranslator(epsilon,metric_space)
reval=EpsilonReachability(trans,log=sys.stderr)

print reval.compute_reachset(H,init)
```

V. Conclusion

This paper presents a Python package, named `pyHybridAnalysis`, that implements the ϵ-semantics framework developed in [14], [15], [3]. First of all, we recalled the notion of hybrid automaton, we described the reachability problem, and we cited the well-known undecidability result for it. After that, we reviewed the theory of ϵ-semantics and we showed how to reduce the evaluation of a particular ϵ-semantics to the evaluation of the standard one. We exposed many of the features of `pyHybridAnalysis` and we exhibit through examples that this package is really easy to use. The source code of the presented package has been released under the LGPL license and it can be freely downloaded from http://www.dmi.units.it/~casagran/pyHybridAnalysis.

Many of the criticisms to the ϵ-semantic framework concern the computational complexity of the quantifier elimination procedure needed to complete the emptiness test of Algorithm 1. This procedure, which is more than exponential in the number of variables or quantifier alternations, is the bottleneck of all the analysis. We are aware of the fact that such an instrument will not be able to treat automata with a high number of variables, nevertheless, `pyHybridAnalysis` proves that we have the chance of studing properties of real hybrid automata by adopting this technique.

In future works, we will investigate the scalability of this tool and we plan to improve the efficiency of the emptiness test by interfacing `pyHybridAnalysis` to a more recent algorithm for the elimination of quantifiers (e.g,. see [16]).

References

[1] P. Prabhakar, V. Vladimerou, M. Viswanathan, and G. E. Dullerud, "Verifying tolerant systems using polynomial approximations," in *Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*, ser. RTSS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 181–190. [Online]. Available: http://dx.doi.org/10.1109/RTSS.2009.28

[2] J. Lygeros and N. Lynch, "Strings of vehicles: Modeling and safety conditions," *Hybrid Systems: Computation and Control*, pp. 273–288, 1998.

[3] A. Casagrande, T. Dreossi, and C. Piazza, "Hybrid automata and ϵ-analysis on a neural oscillator," in *Proceedings First International Workshop on Hybrid Systems and Biology (HSB 2012)*, ser. EPTCS, vol. 92, 2012, pp. 58–72.

[4] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" in *Proc. of ACM Symposium on Theory of Computing (STOCS'95)*, 1995, pp. 373–382.

[5] G. Lafferriere, G. J. Pappas, and S. Sastry, "O-minimal hybrid systems," *Mathematics of Control, Signals, and Systems*, vol. 13, pp. 1–21, 2000.

[6] A. Casagrande, C. Piazza, A. Policriti, and B. Mishra, "Inclusion dynamics hybrid automata," *Inform. and Comput.*, vol. 206, no. 12, pp. 1394–1424, 2008.

[7] S. Sankaranarayanan, T. Dang, and F. Ivancic, "Symbolic model checking of hybrid systems using template polyhedra," in *TACAS*, 2008, pp. 188–202.

[8] M. Fränzle, "Analysis of hybrid systems: An ounce of realism can save an infinity of states," in *Int. Workshop on Computer, Science, and Logic (CSL 99)*, ser. LNCS, J. Flum and M. Rodríguez-Artalejo, Eds., vol. 1683. Springer, 1999, pp. 126–140.

[9] A. Girard, A. A. Julius, and G. J. Pappas, "Approximate simulation relations for hybrid systems," *Discrete Event Dyn. Syst.*, vol. 18, no. 2, pp. 163–179, 2008.

[10] A. Tiwari and G. Khanna, "Series of Abstraction for Hybrid Automata," in *Proc. of Hybrid Systems: Computation and Control (HSCC'02)*, ser. LNCS, C. J. Tomlin and M. Greenstreet, Eds., vol. 2289. Springer, 2002, pp. 465–478.

[11] E. Asarin, T. Dang, and O. Maler, "The d/dt tool for verification of hybrid systems," in *Proc. of 14th International Conference on Computer Aided Verification (CAV'02)*, ser. LNCS, E. Brinksma and K. G. Larsen, Eds., vol. 2404, 2002, pp. 365–370.

[12] A. Balluchi, A. Casagrande, P. Collins, A. Ferrari, T. Villa, and A. L. Sangiovanni-Vincentelli, "Ariadne: a framework for reachability analysis of hybrid automata," in *In: Proceedings of the International Syposium on Mathematical Theory of Networks and Systems.(2006*. Citeseer, 2006.

[13] S. Ratschan and Z. She, "Safety verification of hybrid systems by constraint propagation based abstraction refinement," in *Proc. of Hybrid Systems: Computation and Control (HSCC'05)*, ser. LNCS, M. Morari and L. Thiele, Eds., vol. 3114. Springer, 2005, pp. 573–589.

[14] A. Casagrade, C. Piazza, and A. Policriti, "Discrete semantics for hybrid automata. avoiding misleading assumptions in systems biology." *Discrete Event Dynamic Systems*, vol. 19, no. 4, pp. 471–493, 2009.

[15] A. Casagrande, T. Dreossi, and C. Piazza, "Approximated symbolic computations over hybrid automata," in *Proceedings of the Third International Workshop on Hybrid Autonomous Systems (HAS 2013)*, ser. EPTCS, 2013, p. To appear.

[16] S. Basu, "New results on quantifier elimination over real closed fields and applications to constraint databases," *J. ACM*, vol. 46, no. 4, pp. 537–555, 1999.