Concolic Testing for Analysis of Hybrid Systems

CS477 Project Report

Umang Mathur Atul Sandur

November 9, 2015

1 Overview

Cyberphysical systems are engineered systems that are built from, and depend upon, the seamless integration of computational algorithms and physical components. These systems have made entered into every aspect of our life, and have become an indenspensible part of modern life. Indeed, the applications of these systems are widespread. Example include engineered pacemakers and robotic arms in healthcare, autopilots in passenger aircrafts and spacecrafts, controllers for chemical plants, self driving cars, railways systems, critical systems like nuclear reactors. Clearly the list is endless.

The framework of Hybrid automata—introduced by Alur, Courcourbetis, Henzinger, and Ho [1]—provides a formal modeling and analysis environment to analyze the interaction between the discrete and the continuous parts of cyberphysical systems. Hybrid automata can be considered as generalizations of finite state automata augmented with a finite set of real-valued variables whose dynamics in each state is governed by a system of ordinary differential equations. Moreover, the discrete transitions of hybrid automata are guarded by constraints over the values of these real-valued variables, and enable discontinuous jumps in the evolution of these variables.

Given the immense integration of these systems in modern applicances, it is natural to ask questions such as whether a given system is free from erros. Concretely, if a given hybrid system reaches some *error* state, it can lead to loss of lives and property. For safety critical systems such as nuclear power plants, it becomes quite important to verify that the system does not crash.

As one can expect, there is a rich literature on verification of hybrid systems to tackle this problem. However, it turns out that the general model of Hybrid automata is quite expressive and not surprisingly, very simple questions like reachability are undecidable. Decidable subclasses for hybrid automata do exist, but most of them rely on a finite bisimulation to check for reachability, and do not scale very well.

As part of this project, we intend to develop systematic heurisitcs for answering questions such as reachability of some state in hybrid systems, using concepts from automated test input generation. The main advantage for using

these techniques is scalability, that is, from the literature, these techniques look promising, and seem to scale for decently large systems. Another advantage is that such techniques also give information about concrete input values that lead the system to a target state.

We will use the model of Singular Hybrid Automata (or *Multi rate automata* as it was originally called), introduced by Alur et. al., to demonstrate the applicability of our approach. While the model is quite simple to describe, the reachability problem is easily undecidable for very simple subclasses (namely when the number of continuous variables exceed 2).

The rest of the document is organized as follows. Section 2 describes the details for Singular Hybrid Automata (referred to as SHA from now on). Section 3 briefly introduces the framework for concolic testing and concolic walk. Section 4 descibes our approach, and specific details about the heuristics we use. Section 5 describes implementation details about our tool `Hybrolic`. The report concludes with Section 6 .

2 Singular Hybrid Automata

In this section, we describe the syntax and semantics for singular hybrid automata. Later we mention results about the rechability problems in SHA.

Most of the notations described in this report have been derived from [5].

2.1 Preliminaries

Let $\mathbb{R}$ be the set of real numbers. Let X be a finite set of real-valued variables. A *valuation* on X is a function $\nu : X \to \mathbb{R}$. We assume an arbitrary but fixed ordering on the variables and write x_i for the variable with order i. This allows us to treat a valuation ν as a point $(\nu(x_1), \nu(x_2), \ldots, \nu(x_n)) \in \mathbb{R}^{|X|}$ equipped with the standard *Euclidean norm* $\|\cdot\|$. Abusing notations slightly, we use a valuation on X and a point in $\mathbb{R}^{|X|}$ interchangeably.

We denote points in this state space by $\overline{x}, \overline{y}$, vectors by $\vec{r}, \vec{v}$, and the i-th coordinate of point $\overline{x}$ and vector $\vec{r}$ by $\overline{x}(i)$ and $\vec{r}(i)$, respectively. We write $\vec{0}$ for a vector with all its coordinates equal to 0; its dimension is often clear from the context. The distance $\|\overline{x}, \overline{y}\|$ between points $\overline{x}$ and $\overline{y}$ is defined as $\|\overline{x}-\overline{y}\|$. We denote a *closed ball* of radius $d \in \mathbb{R}_{\geq 0}$ centered at $\overline{x}$ as $B_d(\overline{x}) = \{\overline{y} \in \mathbb{R}^n : \|\overline{x}, \overline{y}\| \leq d\}$. We say that a set $S \subseteq \mathbb{R}^n$ is *bounded* if there exists $d \in \mathbb{R}_{\geq 0}$ such that for all $\overline{x}, \overline{y} \in S$ we have $\|\overline{x}, \overline{y}\| \leq d$.

We define a constraint over a set X as a subset of $\mathbb{R}^{|X|}$. We say that a constraint is *polyhedral* if it is defined as the conjunction of a finite set of linear constraints of the form

$$a_1 x_1 + \cdots + a_n x_n \bowtie k,$$

where $k \in \mathbb{Z}$, for all $1 \leq i \leq n$ we have that $a_i \in \mathbb{R}, x_i \in X$, and $\bowtie \in \{<, \leq, =, >, \geq\}$. Every polyhedral constraints can be written in the standard form $A\overline{x} \leq \vec{b}$ for some matrix A of size $k \times n$ and a vector $\vec{b} \in \mathbb{R}^k$. We call a bounded polyhedral constraint a *convex polytope*. For a constraint G, we write $[\![G]\!]$ for

the set of valuations in $\mathbb{R}^{|X|}$ satisfying the constraint G. We write $\top$ (resp., $\bot$) for the special constraint that is true (resp., false) in all the valuations, i.e. $[\![\top]\!] = \mathbb{R}^{|X|}$ (resp., $[\![\bot]\!] = \emptyset$). We write poly(X) for the set of polyhedral constraints over X including $\top$ and $\bot$.

2.2 Syntax of Singular Hybrid Automata

Singular hybrid automata can be considered as extensions of finite state-transition graphs with a finite set of real-valued variables that grow with state-dependent constant-rates. The transitions of the automata are guarded by predicates on the valuations of the variables, and the syntax allows discrete update of the value of the variables.

Definition 2.1 (Singular Hybrid Automata)**.** A singular hybrid automaton is a tuple $\mathcal{H} = (M, M_0, \Sigma, X, \Delta, I, F)$ where

- M is a finite set of control *modes* including a distinguished initial set of control modes $M_0 \subseteq M$,
- Σ is a finite set of *actions*,
- X is an (ordered) set of *variables*,
- $\Delta \subseteq M \times \text{poly}(X) \times \Sigma \times 2^X \times M$ is the *transition relation*,
- $I : M \to \text{poly}(X)$ is the mode-invariant function, and
- $F : M \to \mathbb{Q}^{|X|}$ is the mode-dependent *flow function* characterizing the rate of each variable in each mode.

For computation purposes, we assume that all real numbers are rational and represented in the standard way by writing down the numerator and denominator in binary.

For all $\delta = (m, G, a, R, m') \in \Delta$ we say that δ is a *transition* between the modes m and m' with *guard* $G \in \text{poly}(X)$ and reset set $R \in 2^{|X|}$. For the sake of notational convenience, we assume that an action $a \in \Sigma$ uniquely determines a transition (m, G, a, R, m'), and we write $G(a)$ and $R(a)$ for the guard and the reset set corresponding to the action $a \in \Sigma$. This can be assumed without loss of generality, since, in this paper, we do not study language-theoretic properties of an SHA, and assume that the non-determinism is resolved by the controller.

Example 2.1 (Singular Hybrid Automata)**.** As an example of a SHA consider the automaton shown in Figure 2.2 with modes $\{m_0, m_1, m_2, m_3\}$ and variable set $\{x_1, x_2\}$. We represent modes using rounded rectangles and we write the invariant of a mode below the rectangle for the mode. We show a transition (m, G, a, R, m') as an arrow between modes m and m' labeled by triplet G, a, R. If we omit the guard of a transition or the invariant of a mode, it is implicitly assumed to be $\top$. Similarly, if we omit the reset set for a transition, it is assumed to be an empty set. To avoid confusion we write the flow of the variables as separate rate for each variable using dot notation to depict the first-order derivative.

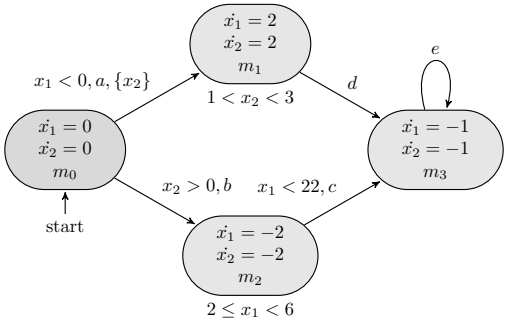

Figure 1: Singular Hybrid Automata with four modes and two variables.

2.3 Semantics of Singular Hybrid Automata

A *configuration* of a SHA $\mathcal{H}$ is a pair $(m, \nu) \in M \times \mathbb{R}^{|X|}$ consisting of a control mode m and a variable valuation $\nu \in \mathbb{R}^{|X|}$ such that that ν satisfies the invariant $I(m)$ of the mode m, i.e. $\nu \in [\![I(m)]\!]$. We say that the transition $\delta = (m, G, a, R, m')$ is *enabled* in a configuration (m, ν) when guard $G \in \text{poly}(X)$ is satisfied by the valuation, i.e. $\nu \in [\![G]\!]$. Moreover, the transition δ resets the variables in $R \in 2^{|X|}$ to 0. We write $\nu[R{:=}0]$ to denote the valuation resulting from substituting in valuation ν the value for the variables in the set R to 0, formally

$$\nu[R{:=}0](x) = \begin{cases} 0 & \text{if} x \in R \\ \nu(x) & \text{otherwise.} \end{cases}$$

A *timed action* of a SHA is the tuple $(t, a) \in \mathbb{R}_{\geq 0} \times \Sigma$ consisting of a time delay and discrete action. While the system dwells in a mode $m \in M$ the valuation of the system flows linearly according to the rate function $F(m)$, i.e. after spending t time units in mode m from a valuation ν the valuation of the variables will be $\nu + t \cdot F(m)$.

We say that $((m, \nu), (t, a), (m', \nu'))$ is a transition of a SHA $\mathcal{H}$ and we write $(m, \nu) \xrightarrow{t}_a (m', \nu')$ if (m, ν) and (m', ν') are valid configurations of the SHA $\mathcal{H}$, and there is a transition $\delta = (m, G, a, R, m') \in \Delta$ such that:

- all the valuations resulting from dwelling in mode m for time t from the valuation ν satisfy the invariant of the mode m, i.e. $(\nu + F(m) \cdot \tau) \in [\![I(m)]\!]$ for all $\tau \in [0, t]$ (observe that due to convexity of the invariant set we only need to check that $(\nu + F(m) \cdot t) \in [\![I(m)]\!]$);
- The valuation reached after waiting for t time-units satisfy the constraint G (called the guard of the transition δ), i.e. $(\nu + F(m) \cdot t) \in [\![G]\!]$, and
- $\nu' = (\nu + F(m) \cdot t)[R := 0]$.

A *finite run* of an SHA $\mathcal{H}$ is a finite sequence

$$r = \langle (m_0, \nu_0), (t_1, a_1), (m_1, \nu_1), (t_2, a_2), \ldots, (m_k, \nu_k) \rangle$$

such that $m_0 \in M_0$ and for all $0 \leq i < k$ we have that $((m_i, \nu_i), (t_{i+1}, a_{i+1}), (m_{i+1}, \nu_{i+1}))$ is a transition of $\mathcal{H}$. For such a run r we say that ν_0 is the *starting valuation*, while ν_k is the *terminal valuation*. An *infinite run* of an SHA $\mathcal{H}$ is similarly defined to be an infinite sequence

$$r = \langle (m_0, \nu_0), (t_1, a_1), (m_1, \nu_1), (t_2, a_2), \ldots \rangle$$

such that $((m_i, \nu_i), (t_{i+1}, a_{i+1}), (m_{i+1}, \nu_{i+1}))$ is a transition of the SHA $\mathcal{H}$ for all $i \geq 0$. We say that ν_0 is the starting configuration of the run. We say that an infinite run $r = \langle (m_0, \nu_0), (t_1, a_1), (m_1, \nu_1), \ldots \rangle$ is Zeno if $\sum_{i=1}^{\infty} t_i < \infty$. Zeno runs are physically unrealizable since they requires infinitely many mode-switches within a finite amount of time.

2.4 Reachability Problem

We next define the reachability problem for SHA, which is a fundamental problem for this model.

Definition 2.2 (Reachability Problem)**.** Given a singular hybrid automaton $\mathcal{H}$=$(M, M_0, \Sigma, X, \Delta, I, F)$, a starting valuation $\nu \in \mathbb{R}^{|X|}$, and a target polytope $\mathcal{T} \subseteq \mathbb{R}^X$, the reachability problem, $\textsc{Reach}(\mathcal{H}, \nu, \mathcal{T})$, is to decide whether there exists a finite run from ν_0 to some valuation $\nu' \in \mathcal{T}$.

The termination [6] problem for two-counter Minsky machines is known to be undecidable. By encoding the two counters as two vairables, and using another variable to do additional book-keeping, the termination problem for Minsky machines can be reduced to reachability problem for singular hybrid automata.

Hence the following result is immediate.

Theorem 2.1 (Undecidability [4, 3, 2])**.** *The reachability problem is undecidable for singular hybrid automata with three or more variables.*

We observe that the reachability problem for SHA with one variable remains decidable, and can be solved in linear time.

Theorem 2.2 (Decidability of 1 variable case [5])**.** *The reachability and the schedulability problems are* NL-Complete *for singular hybrid automata with single variable.*

3 Concolic Testing and Concolic Walk

Concolic testing [7] automates test input generation by combining concrete and symbolic executions of the system under test. So the concrete part constitutes

normal execution of the program while symbolic part collects symbolic constraints over the symbolic input values at each branch point encountered along concrete execution path. At the end of an execution, we have collected a set of path constraints that exercise an execution path in the system. For the next iteration, one of the constraints is negated and solved using constraint solvers to generate a new test input that directs the system along a different execution path. This process continues until all feasible distinct execution paths have been explored. When some of the symbolic constraints become too complicated for the constraint solver to generate concrete values, they are simplified by replacing some of the symbolic values with concrete values. Concolic testing is thus sound (infers real bugs) and also allows exploring previously unreachable paths in the system.

4 Approach

The key observation that will aid us in designing the testing framework is that eventhough the reachability problem asks for concrete values of time durations in any specific run, it suffices to only consider at the sequences of transitions taken and theremodes visited.

4.1 Symbolic Run

A *symbolic finite run* of an SHA $\mathcal{H}$ is a finite sequence

$$r = \langle m_0, a_1, m_1, a_2, \ldots m_k \rangle$$

such that $m_0 \in M_0$ and for all $0 \leq i < k$ we have that there are valuations ν_i and ν_{i+1}, and $t_{i+1} \geq 0$ such that $((m_i, \nu_i), (t_{i+1}, a_{i+1}), (m_{i+1}, \nu_{i+1}))$ is a transition of $\mathcal{H}$.

Observe that for every (finite or infinite) run

$$r = \langle (m_0, \nu_0), (t_1, a_1), (m_1, \nu_1), \ldots \rangle$$

of a WSHA we have that $\varrho(m_i) \leq \varrho(m_j)$ for every $i \leq j$. We define the type $\Gamma(r)$ of a finite run

$$r = \langle (m_0, \nu_0), (t_1, a_1), (m_1, \nu_1), \ldots, (m_k, \nu_k) \rangle$$

as a finite sequence of ranks (natural numbers) and actions $\langle n_0, b_1, n_1, \ldots, b_p, n_p \rangle$ defined inductively in the following manner:

$$\Gamma(r) = \begin{cases} \langle \varrho(m_0) \rangle & \text{if } r = \langle (m_0, \nu_0) \rangle \\ \Gamma(r') \oplus (a, \varrho(m)) & \text{if } r = r' :: \langle (t, a), (m, \nu) \rangle, \end{cases}$$

where :: is the cons operator that appends two sequences, while for a sequence $\sigma = \langle n_0, b_1, n_1, \ldots, b_p, n_p \rangle$, $a \in \Sigma$, and $n \in \mathbb{N}$ we define $\sigma \oplus (a, n)$ to be equal to σ if $n_p = n$ and $\langle n_0, b_1, n_1, \ldots, n_p, a, n \rangle$ otherwise. Intuitively, the type of

a finite run gives the (non-duplicate) sequence of ranks of modes and actions appearing in the run, where action is stored only when a transition to a mode of higher rank happens. We need to remember only these actions since transitions that stay in the modes of same rank do not reset the variables.

Since there are only finitely many ranks for a given WSHA, it follows that for every infinite run

$$r = \langle (m_0, \nu_0), (t_1, a_1), (m_1, \nu_1), \ldots \rangle$$

there exists an index i such that for all $j \geq i$ we have that $\varrho(m_i) = \varrho(m_j)$. With this intuition we define the type of an infinite run r as the type of the finite prefix of r till index i. Let $\Gamma_{\mathcal{H}}$ be the set of run types of a WSHA $\mathcal{H}$.

Given initial values of continuous variables and a concrete trace, it is easy to check if it is a valid trace that reaches some state q. However coming up with concrete values for time durations in that trace is non-trivial because the possible values of t_i's are uncountable. Therefore it makes sense to look at symbolic traces where t_i's are symbolic variables.

A very basic idea to solve reachability question using test generation concepts, is to collect constraints on invariants/guards in a given symbolic trace and feed it to a suitable solver to check for feasibility of the constraints. This will give concrete initial values for the input to the system. It is possible to avoid quantifiers from the constraints because the continuous variables evolve in a linear fashion relative to each other. Therefore we expect simple SMT solvers like Yices to be able to handle these constraints. The challenge is in determining correct symbolic traces that lead to a final goal state in hybrid sytems which have loops. This is because one can generate infinite number of symbolic traces as inputs to our algorithm. We will initially focus our attention on bounded step reachability. Another challenging aspect is to be able to deal with non-linear invariants/guards as constraints that most solvers cannot handle. We attempt to attack this problem by guiding symbolic executions using concrete values for symbolic variables. This is primarily where we will use concepts from concolic testing techniques. Further, such non-linear constraints result in quantified constraints which have one or more quanitified variables. This is again difficult for constraint solvers to handle. For hybrid systems where continuous variables evolve in a much complex fashion, such as when the rates of evolution are specified in terms of arbitrary differential equations, there are many more challenges that need to be addressed. We will be looking at such problems as part of future work.

5 Implementation

The key observation that will aid us in designing the testing framework is that eventhough the reachability problem asks for concrete values of time durations in any specific run, it suffices to only consider at the sequences of transitions taken and theremodes visited.

5.1 Symbolic Run

A *symbolic finite run* of an SHA $\mathcal{H}$ is a finite sequence

$$r = \langle m_0, a_1, m_1, a_2, \ldots m_k \rangle$$

such that $m_0 \in M_0$ and for all $0 \leq i < k$ we have that there are valuations ν_i and ν_{i+1}, and $t_{i+1} \geq 0$ such that $((m_i, \nu_i), (t_{i+1}, a_{i+1}), (m_{i+1}, \nu_{i+1}))$ is a transition of $\mathcal{H}$.

Observe that for every (finite or infinite) run

$$r = \langle (m_0, \nu_0), (t_1, a_1), (m_1, \nu_1), \ldots \rangle$$

of a WSHA we have that $\varrho(m_i) \leq \varrho(m_j)$ for every $i \leq j$. We define the type $\Gamma(r)$ of a finite run

$$r = \langle (m_0, \nu_0), (t_1, a_1), (m_1, \nu_1), \ldots, (m_k, \nu_k) \rangle$$

as a finite sequence of ranks (natural numbers) and actions $\langle n_0, b_1, n_1, \ldots, b_p, n_p \rangle$ defined inductively in the following manner:

$$\Gamma(r) = \begin{cases} \langle \varrho(m_0) \rangle & \text{if } r = \langle (m_0, \nu_0) \rangle \\ \Gamma(r') \oplus (a, \varrho(m)) & \text{if } r = r' :: \langle (t, a), (m, \nu) \rangle, \end{cases}$$

where :: is the cons operator that appends two sequences, while for a sequence $\sigma = \langle n_0, b_1, n_1, \ldots, b_p, n_p \rangle$, $a \in \Sigma$, and $n \in \mathbb{N}$ we define $\sigma \oplus (a, n)$ to be equal to σ if $n_p = n$ and $\langle n_0, b_1, n_1, \ldots, n_p, a, n \rangle$ otherwise. Intuitively, the type of a finite run gives the (non-duplicate) sequence of ranks of modes and actions appearing in the run, where action is stored only when a transition to a mode of higher rank happens. We need to remember only these actions since transitions that stay in the modes of same rank do not reset the variables.

Since there are only finitely many ranks for a given WSHA, it follows that for every infinite run

$$r = \langle (m_0, \nu_0), (t_1, a_1), (m_1, \nu_1), \ldots \rangle$$

there exists an index i such that for all $j \geq i$ we have that $\varrho(m_i) = \varrho(m_j)$. With this intuition we define the type of an infinite run r as the type of the finite prefix of r till index i. Let $\Gamma_{\mathcal{H}}$ be the set of run types of a WSHA $\mathcal{H}$.

Given initial values of continuous variables and a concrete trace, it is easy to check if it is a valid trace that reaches some state q. However coming up with concrete values for time durations in that trace is non-trivial because the possible values of t_i's are uncountable. Therefore it makes sense to look at symbolic traces where t_i's are symbolic variables.

A very basic idea to solve reachability question using test generation concepts, is to collect constraints on invariants/guards in a given symbolic trace and feed it to a suitable solver to check for feasibility of the constraints. This will give concrete initial values for the input to the system. It is possible to avoid

quantifiers from the constraints because the continuous variables evolve in a linear fashion relative to each other. Therefore we expect simple SMT solvers like Yices to be able to handle these constraints. The challenge is in determining correct symbolic traces that lead to a final goal state in hybrid sytems which have loops. This is because one can generate infinite number of symbolic traces as inputs to our algorithm. We will initially focus our attention on bounded step reachability. Another challenging aspect is to be able to deal with non-linear invariants/guards as constraints that most solvers cannot handle. We attempt to attack this problem by guiding symbolic executions using concrete values for symbolic variables. This is primarily where we will use concepts from concolic testing techniques. Further, such non-linear constraints result in quantified constraints which have one or more quanitified variables. This is again difficult for constraint solvers to handle. For hybrid systems where continuous variables evolve in a much complex fashion, such as when the rates of evolution are specified in terms of arbitrary differential equations, there are many more challenges that need to be addressed. We will be looking at such problems as part of future work.

6 Conclusion

References

[1] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-S. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, pages 209–229, 1992.

[2] E. Asarin and O. Maler. Achilles and the tortoise climbing up the arithmetical hierarchy. *Journal of Computer and System Sciences*, 57(3):389 – 398, 1998.

[3] E. Asarin, M. Oded, and A. Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138:35–66, 1995.

[4] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Comp. and Sys. Sciences*, 57:94–124, 1998.

[5] Shankara Narayanan Krishna, Umang Mathur, and Ashutosh Trivedi. Weak singular hybrid automata. In *Formal Modeling and Analysis of Timed Systems*, Lecture Notes in Computer Science. Springer International Publishing, 2014.

[6] Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[7] Koushik Sen. Concolic testing. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, pages 571–572, New York, NY, USA, 2007. ACM.