Poster: HyRG: A Random Generation Tool for Affine Hybrid Automata

Luan Viet Nguyen
University of Texas at Arlington, USA

Christian Schilling
Albert-Ludwigs-Universität
Freiburg, Germany

Sergiy Bogomolov
Albert-Ludwigs-Universität
Freiburg, Germany

Taylor T. Johnson
University of Texas at
Arlington, USA

ABSTRACT

In this poster, we present methods for randomly generating hybrid automata with affine differential equations, invariants, guards, and assignments. Selecting an arbitrary affine function from the set of all affine functions results in a low likelihood of generating hybrid automata with diverse and interesting behaviors, as there are an uncountable number of elements in the set of all affine functions. Instead, we partition the set of all affine functions into potentially interesting classes and randomly select elements from these classes. For example, we partition the set of all affine differential equations by using restrictions on eigenvalues such as those that yield stable, unstable, etc. equilibrium points. We partition the components describing discrete behavior (guards, assignments, and invariants) to allow either time-dependent or state-dependent switching, and in particular provide the ability to generate subclasses of piecewise-affine hybrid automata. Our preliminary experimental results with a prototype tool called HyRG (Hybrid Random Generator) illustrate the feasibility of this generation method to automatically create standard hybrid automaton examples like the bouncing ball and thermostat.

Categories and Subject Descriptors

D.2.5 [**Testing and Debugging**]: Testing tools

General Terms

Theory, verification

Keywords

Hybrid automata, program generation

1. INTRODUCTION

In this poster, we present methods to randomly generate hybrid automata with affine (linear) differential equations, invariants, guards, and assignments implemented in a prototype tool called HyRG. While random generation of affine vector fields (i.e., continuous linear systems) has been used to evaluate reachability algorithms [1], to the best of our knowledge, there has been no effort to randomly generate hybrid automata with more complex discrete structure. Existing methods for generating continuous linear systems are relatively unsophisticated.[1] There are many tools and methods for random program generation in various languages (C, Java, etc.) [4]. Random generation of models is useful for: (a) evaluating reachability algorithms, (b) testing various components (from parsers to analysis algorithms) in analysis tools, (c) testing translators from hybrid systems modeling languages to other tools like Mathworks Simulink/Stateflow (SLSF) [3], and (d) developing libraries of examples with diverse continuous and discrete behaviors.

ICCPS '15 April 14 - 16, 2015, Seattle, WA, USA

ACM 978-1-4503-3433-4/15/04
http://dx.doi.org/10.1145/2728606.2728650.

2. GENERATING HYBRID AUTOMATA

For brevity, we do not define the structure of hybrid automata, but refer to the semantics and syntax definitions of hybrid automata in [2]. In HyRG, we randomly generate sets of locations, continuous dynamics (flows), invariants, transition, guards, assignments, and initial conditions for a hybrid automaton. We denote a hybrid automaton that has been randomly generated by $\mathcal{A}_R$. Rather than picking only random matrices and vectors for the affine functions used in flows, guards, invariants, assignments, etc., we instead partition these affine functions into interesting classes. Due to space, we focus on describing how different classes of affine functions used in flows are randomly generated using eigenvalue constraints.

Randomly Generating Continuous Flows. A randomly generated affine hybrid automaton $\mathcal{A}_R$ has continuous dynamics defined as $\dot{x} = Ax + B, x \in \mathcal{R}^n$, where n is a random number of state variables, x is an n-vector of state variables, and $\dot{x}$ is an n-vector of the derivatives of these variables w.r.t. time. Furthermore, A is an $n \times n$-matrix of real coefficients and B is an n-vector of real constants. Using the eigen-decomposition theorem, a matrix A can be written as $A = vDv^{-1}$, where D is an $n \times n$ diagonal matrix whose diagonal elements correspond to n eigenvalues λ_i of the matrix A, and v is an $n \times n$-matrix where each column v_i is a corresponding eigenvector of A. If $\Psi(t) = e^{At}$ is a fundamental matrix of a linear system of differential equations $\dot{x} = Ax$, where t denotes time, so $\Psi(t) = e^{At}v = e^{vDv^{-1}t}v = ve^{Dt}v^{-1}v = ve^{Dt}$ is also a fundamental matrix

[1]For instance, MathWorks Matlab includes a function `rss` to generate random linear systems, http://www.mathworks.com/help/control/ref/rss.html.

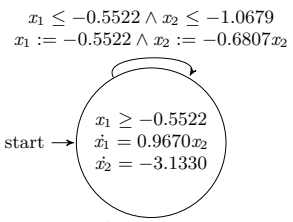


Figure 1: A randomly generated hybrid automaton from HyRG with similar behavior as the bouncing ball (BB) example.

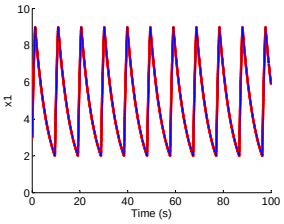
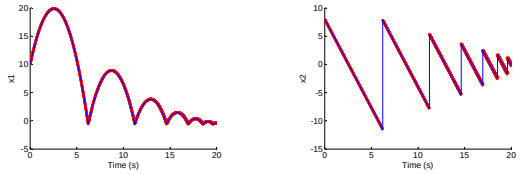


Figure 2: Reachable states of a randomly generated hybrid automaton from HyRG with similar behavior as the thermostat example.



(a) (b)

Figure 3: The reachable states showing x_1 and x_2 computed by the LGG algorithm in SpaceEx (red) contain their SLSF simulation traces (blue) in Example 1.

of this system. Therefore, the general solution of a system of differential equations $\dot{x} = Ax$ is $x(t) = ve^{Dt}C$, where C is an n-vector of real values determined by the initial conditions of $x(t)$. If $x(t_0) = x_0$ is a vector of initial conditions, then $C = \Psi(t_0)^{-1}x_0$. For linear systems, the continuous dynamics may be described as an exponential function of eigenvalues, and the eigen-decomposition theorem allows us to generate a random matrix A from sets of arbitrarily given eigenvalues and eigenvectors. Thus, we first randomly generate a matrix of eigenvectors v as an arbitrarily non-singular $n \times n$-matrix, and then add constraints over the randomly generated $n \times n$ diagonal matrix of eigenvalues D. In other words, we can randomly generate many classes of continuous dynamics with different stability scenarios based on manipulating different sets of given eigenvalues.

3. HyRG TOOL AND RESULTS

We implemented the prototype HyRG tool in Java and Matlab and evaluated it in several scenarios.[2]. Suppose that randHA(m, n) is the main function in HyRG, where m is a number of locations and n is a numbers of variables, and the output is a hybrid automaton $\mathcal{A}_R$. We simulate $\mathcal{A}_R$ in SLSF and compute its reachable states in SpaceEx. The simulation trace is contained in the set of reachable states computed by SpaceEx, then validates the generation.

Example 1. For the input $m = 1$ and $n = 2$, if we repeatedly execute the function randHA enough times, we can randomly generate a hybrid automaton $\mathcal{A}_R$ whose discrete structure and trajectories are similar to the bouncing ball (BB) system. A randomly generated instance is shown in Figure 1. The initial values of its state variables are randomly generated as $x_1 = 10$, $x_2 = 8$. Figure 3 shows the SpaceEx reachability analysis and SLSF simulations[3] of $\mathcal{A}_R$, where x_1 and x_2 represent the position and velocity of the BB system. We also randomly generated 100 models similar to the BB system and collected data of the generation, shown in Table 1. When generating an automaton with a single location and two variables, on average we will generate a BB model after running randHA about 112 times. The average time to generate each BB model is about 17 seconds.

[2]The tool and examples are available online: http://www.verivital.com/hyrg/

[3]We use a converter from hybrid automaton models to SLSF models to perform the SLSF simulations [3].

Table 1: HyRG trial table for randomly generating 100 BB and thermostat/heater examples, where N denotes a number of trials and T is a generation time in seconds.

Example		Mean	Median	Std.Dev	Min	Max
BB	N	111.63	65	120.26	1	661
	T	17.022	9.824	18.615	0.0946	101.23
Heater	N	2126.5	1481	2152.2	24	10710
	T	216.35	152.13	219.15	2.4855	1091.5

Example 2. Again, if we run the function randHA long enough, this time with the input $m = 2$ and $n = 1$, we can randomly generate a hybrid automaton $\mathcal{A}_R$ that has discrete structure and trajectories similar to the thermostat system. The system $\mathcal{A}_R$ starts at location On, and an initial value of x_1 is equal to 3. The SpaceEx reachability analysis and SLSF simulation of $\mathcal{A}_R$ are shown in Figure 2, where x_1 represents the temperature of a thermostat system. Again, we generated 100 random hybrid models similar to the thermostat system. Table 1 shows the data collected from the generation process. The average number of unsuccessful trials before we get one hybrid model similar to the thermostat system is approximately 2127 trials, and an average generation time for this model is 216.35 seconds. By comparison with the trial data for generating the BB example in Table 1, we can see that the random generation function randHA runs more than ten times longer to produce the thermostat system. Since increasing the number of locations of $\mathcal{A}_R$ leads to a larger number of choices for other components (e.g., continuous dynamics, invariants, transitions, etc.), more instances of $\mathcal{A}_R$ needed to be generated.

Outlook and Future Work. Overall, these results indicate HyRG is able to generate known hybrid automaton models—such as BB and thermostat—that are of interest to the research community and not purely arbitrary. We highlight that *all* structural components of the automaton were selected randomly (i.e., the transitions and continuous dynamics), and not simply parameters. In ongoing and future work, we will investigate more sophisticated generation methods (such as using stochastic grammars) and analyze larger randomly generated examples.

4. REFERENCES

[1] M. Althoff, B. H. Krogh, and O. Stursberg. Analyzing reachability of linear dynamic systems with parametric uncertainties. In A. Rauh and E. Auer, editors, *Modeling, Design, and Simulation of Systems with Uncertainties*, volume 3 of *Mathematical Engineering*, pages 69–94. Springer Berlin Heidelberg, 2011.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[3] S. Bogomolov, T. T. Johnson, L. V. Nguyen, and C. Schilling. Translating hybrid automata to Simulink/Stateflow models. In *35th IFIP Joint International Conference on Formal Techniques for Distributed Systems*. Springer, 2015 (Under Review).

[4] X. Yang, Y. Chen, E. Eide, and J. Regehr. Finding and understanding bugs in c compilers. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 283–294, New York, NY, USA, 2011. ACM.