Challenges and Tool Implementation of Hybrid Rapidly-Exploring Random Trees

Stanley Bak[1], Sergiy Bogomolov[2], Thomas A. Henzinger[3], and Aviral Kumar[4(✉)]

[1] Air Force Research Laboratory, Dayton, OH, USA
[2] Australian National Unviersity, Canberra, Australia
[3] IST Austria, Klosterneuberg, Austria
[4] Indian Institute of Technology Bombay, Mumbai, India
aviralkumar2907@gmail.com

Abstract. A Rapidly-exploring Random Tree (RRT) is an algorithm which can search a non-convex region of space by incrementally building a space-filling tree. The tree is constructed from random points drawn from system's state space and is biased to grow towards large unexplored areas in the system. RRT can provide better coverage of a system's possible behaviors compared with random simulations, but is more lightweight than full reachability analysis. In this paper, we explore some of the design decisions encountered while implementing a hybrid extension of the RRT algorithm, which have not been elaborated on before. In particular, we focus on handling non-determinism, which arises due to discrete transitions. We introduce the notion of important points to account for this phenomena. We showcase our ideas using heater and navigation benchmarks.

1 Introduction

Hybrid automata are mathematical models that combine discrete and continuous dynamics. This formalism can be used to analyze many real word systems. Hybrid automata analysis and particularly reachability analysis are computationally expensive and might be even intractable for large systems. Nondeterministic behavior, e.g. uncertain inputs, make analysis even more challenging [2,5,7,12]. Simulation based techniques [6] belong to a promising class of techniques to automatically detect system bugs. At the same time, these methods cannot provide any mathematical guarantees on the system safety.

Rapidly-exploring random trees (RRTs) [8,13] have been developed to address this problem. RRTs were originally proposed in the motion planning community in order to provide a fast and efficient way of to explore the search space towards given goal states. Bhatia and Frazzoli [4] and Esposito et al. [10] explored the application of RRTs for hybrid automata to provide an efficient way

DISTRIBUTION A. Approved for public release; Distribution unlimited. (Approval AFRL PA #88ABW-2016-4898, 30 SEP 2016).

A. Abate and S. Boldo (Eds.): NSV 2017, LNCS 10381, pp. 83–89, 2017.
DOI: 10.1007/978-3-319-63501-9_6

to generate a system simulations which provide some coverage guarantees. Plaku et al. [14] used motion planning techniques for falsification of hybrid automata.

In this paper, we discuss our experience with RRTs and particularly its adaptation to hybrid automata. We note that handling of system non-determinism poses a special challenge. For this purpose, we introduce a notion of *important points*, where an important point refers to a system state where either invariant is violated or a transition guard becomes enabled or disabled. Our algorithm uses sets of important points to find a sweet-spot between simulation time and state space coverage.

2 Preliminaries

In this paper, we consider systems modelled in terms of hybrid automata [1].

Definition 1. *A hybrid automaton is a tuple*

$$H = (S, Inv, E, G, J, U, f, I, F),$$

where

- Q *is a discrete and finite set, called modes*
- X *maps each mode to corresponding continuous state space, i.e.* $q \rightarrow X_q$, *where* $X_q \in \mathbb{R}^{dim(X_q)}$ *is the continuous state space associated with* $q \in Q$
- $S = Q \times X$ *is the Cartesian product of discrete and continuous state space*
- Inv *maps each mode to the corresponding continuous invariant, i.e.* $q \xrightarrow{Inv} Inv_q$ *where* $Inv_q \subseteq X_q$ *represents domain of the continuous variables associated with* $q \in Q$
- $E \subseteq Q \times Q$ *is the set of discrete transitions between modes*
- G *maps discrete transitions to guard conditions, i.e.,* $(q_i, q_j) \xrightarrow{G} G_{(q_i,q_j)}$, *where* $G_{(q_i,q_j)} \subseteq X_{q_i}$ *is the guard condition associated with* $(q_i, q_j) \in E$
- J *maps discrete transitions to reset functions, i.e.,* $(q_i, q_j) \xrightarrow{J} J_{(q_i,q_j)}$ *where* $J_{(q_i,q_j)} : G_{(q_i,q_j)} \rightarrow X_{q_j}$ *is the reset function associated with* $(q_i, q_j) \in E$
- U *maps each mode to the corresponding set of control input signals,* $q \xrightarrow{U} U_q$ *where* $q \in Q$ *and* $U_q \subseteq \mathbb{R}^{dim(U_q)}$
- f *maps each mode to the function that describes the associated continuous dynamics* $q \xrightarrow{f} f_q$, *where* $f_q : X_q \times U_q \rightarrow \dot{X}_q$

In the following, we briefly recall an algorithm to compute RRTs [9,13] (see Algorithm 1 for more details) for purely continuous systems. The algorithm incrementally constructs a tree of feasible trajectories. A RRT is represented by a graph $G = (V, E)$, where the set V consists of the end points of feasible trajectories and $E(v_i, v_j)$ represents the trajectory followed, and is labelled with $u \in U$, i.e. the control signal needed to reach v_j from v_i, within a time interval of the length T.

Initially, the RRT contains a single mode x_{init}. A random point $x_{rand} \in X$ is generated in each iteration (GENERATE-RANDOM), and the nearest neighbour $x_{near} \in V$ to x_{rand} is found in the tree where distance is defined by some distance metric ρ (NEAREST-NEIGHBOUR). Now, a new candidate to be added to the tree, x_{new} is found by applying control signal u for some time horizon T. The chosen control signal u minimizes the distance between x_{rand} and x_{new} (MIN-CONTROL-SIGNAL and EXTEND) The above mentioned steps are repeated until either we reach a ε neighbourhood of x_{target} or exceed number of iterations K_{max}.

3 RRT for Hybrid Automata

In this section, we discuss necessary changes to Algorithm 1 to account for mixed discrete-continuous nature of hybrid automata. We modify the procedure GENERATE-RANDOM to reflect the fact that, in the new setting, the state space consists of both discrete and continuous states. Therefore, in order to select a random point, we randomly select (1) a discrete mode $q \in Q$ and (2) a continuous point which satisfies the invariant Inv_q. In order to define a distance measure in the hybrid state space in the NEAREST-NEIGHBOUR procedure, we follow the approach suggested by Bhatia and Frazzoli [4]. In particular, the distance between two nodes $z_1 = (q, x_1)$ and $z_2 = (q, x_2)$ lying in the same discrete mode is given by $D(z_1, z_2) = d(x_1, x_2)$ where $d(.,.)$ is a standard L_1, L_∞ or Euclidean measure. For two states which belong to different discrete modes, distance is given by a tuple

$$D(z_1, z_2) = (\delta(q_1, q_2), \min_{\forall q \in q_{next}} d(x_1, G(q_1, q))),$$

where $\delta(q_1, q_2)$ denotes the number of nodes on the shortest path from mode q_1 to mode q_2 in graph induced by the discrete structure of the hybrid automaton H and

Procedure RRT(x_{init}, x_{target}, ε)
$V = x_{init}$, $E \leftarrow \emptyset$, $k = 1, x_{new} = +\infty$;
while $k \leq K_{max} \wedge d(x_{new}, x_{target}) > \varepsilon$ **do**
  x_{rand} = GENERATE-RANDOM();
  x_{near} = NEAREST-NEIGHBOUR(x_{rand}, V, ρ);
  u_{min} = MIN-CONTROL-SIGNAL(x_{rand}, x_{near});
  x_{new} = EXTEND(x_{near}, u_{min});
  $V \leftarrow V \cup x_{new}$;
  $E \leftarrow E \cup (x_{new}, x_{near}, u_{min})$;
  $k = k + 1$;
end

Algorithm 1: Algorithm to construct a RRT. The algorithm starts with the single mode x_{init} and grows the tree uniformly until x_{target} or a threshold on number of iterations K_{max} has been reached. The tuple (V, E) stores the current state of the RRT. The used distance measure is referred to as ρ.

$$q_{next} = \{q \in Q | (\delta(q, q_2) < \delta(q_1, q_2)) \wedge E(q_1, q)\}.$$

Geometrically speaking, q_{next} is the set of nodes which are at one edge distance from q_1 and which reduce the path length to q_2 from q_1 by one discrete jump. The procedure, MIN-CONTROL-SIGNAL finds the control input $u \in \mathcal{U}$ which minimizes the distance between x_{rand} and a state reachable from x_{near} within the time frame T. Now, we observe that due to possible *non-determinism* in the hybrid automaton behavior, the state reachable within the time frame T is not uniquely defined. In the next section, we discuss this issue in more details.

4 Non-determinism Handling

We recall that hybrid automata exhibit non-determinism due to discrete mode switches. In particular, for the discrete modes q and q', the mode switch is enabled as long as the system is in the set $Inv_q \cap G(q, q') \cap Inv_{q'}$. For example, in the heater example (see Fig. 1) the transition from mode q_{off} to q_{on} is enabled for the temperature interval $[18; 20.1]$. Therefore, the procedure MIN-CONTROL-SIGNAL might need to consider multiple trajectories while simulating the system behavior up to the time horizon of T time units. In more details, a decision on whether to stay in the current mode or take a transition has to be made for every sampled time moment during the simulation process where a guard is enabled. This observation leads to an exponential number of induced simulations in the worst case. In order to mitigate this problem and restrict the number of considered simulations, we suggest the notion of an *important point*.

Definition 2. *A state reachable along a simulation is* important *if the mode invariant is violated or guard becomes either enabled or disabled exactly at time moment this state has been reached.*

We illustrate the notion of important points on the automaton from Fig. 1. In particular, the simulation starting in the mode q_{off} at the temperature 21° will lead to two important points when the temperature reaches 20.1° (transition from q_{off} to q_{on} becomes enabled) and 18° (invariant of q_{off} gets violated). Therefore, by restricting simulations to important points only, we can drastically restrict our search space.

These ideas are formally summarized in Algorithm 2. The algorithm has two loops. The outer loop iterates over discretized (sampled) version of the input

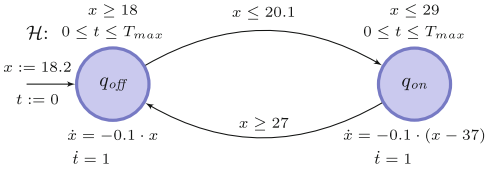


Fig. 1. Hybrid automaton for a heater [3].

set $\mathcal{U}$. Based on the chosen control input, the algorithm numerically integrates differential equations and looks up whether a reached state is important. For every important point, we add its *successor* states along enabled transitions to the set S of important points. Note that, if multiple transitions are enabled, we add *multiple* points in one step, i.e. all successor states along enabled transitions. In this way, we make sure to extend the RRT with nodes featuring new discrete modes for every discovered important point. At the same time, we always add the last simulation point to the set S to ensure it is not empty, even in case no important points have been discovered along the considered trajectory.

Procedure MIN-CONTROL-SIGNAL(x_{rand}, x_{near})
$S \leftarrow \emptyset$;
$q = \text{MODE}(x_{near})$;
$x = \text{CONT}(x_{near})$;
for $u \in \text{DISCRETIZE}(\mathcal{U})$ **do**
  $t = 0$;
  while $t \leq T$ **do**
    $x = \text{ODE-SOLVER}(q, x, \delta, u)$;
    if x *is important* **then**
      $S = S \cup (u, \text{SUCC}(x))$;
    end
    $t = t + \delta$;
  end
  $S = S \cup (u, x)$;
end
return u_{min} from $(u_{min}, x_{min}) = argmin_{u,x \in S}\ \rho(x, x_{near})$;

Algorithm 2: A version of the procedure MIN-CONTROL-SIGNAL which accounts for possible non-determinism in discrete switches. The function DISCRETIZE returns a discretized version of the input set $\mathcal{U}$. The functions MODE and CONT return the discrete and continuous parts of a state, respectively. ODE-SOLVER numerically integrates differential equations for provided mode, initial state, time step and control input. SUCC returns successor states reachable along enabled transitions. The set S stores tuples (u, x) of important points and corresponding control inputs.

5 Experimental Results

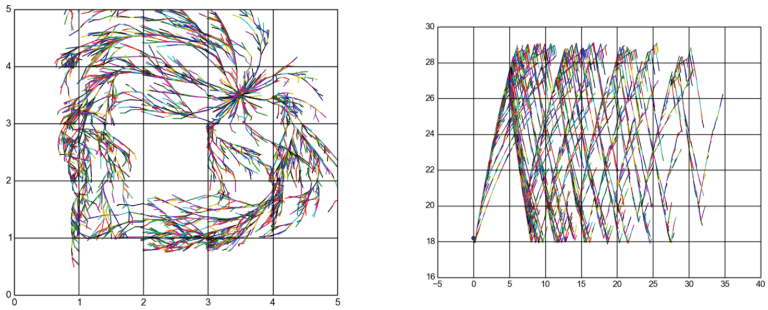
Our implementation uses an input format similar to the Pysim hybrid automaton simulator built in to Hyst [3]. This allows models to be created in SpaceEx [12] and then converted and used within our RRT tool with minimal manual modification. We evaluate our algorithms on the heater benchmark [3] and the navigation benchmark [11].

Navigation Benchmark. This benchmark describes the motion of an object on a 2D plane with differential equations of the form: $x' = v_x,\ y' = v_y, \mathbf{v} = (v_x, v_y), \mathbf{v}' = A(\mathbf{v} - \mathbf{v_d}) + \mathbf{u}$ where

$$A = \begin{bmatrix} -1.2 & -0.8 \\ -0.8 & -1.2 \end{bmatrix}$$

and $\mathbf{u} = (u_1, u_2)$ is the set of control inputs which perturb the differential equations and v_d is a target velocity constant, defined individually for every discrete mode. The control inputs are constrained by $u_1, u_2 \in [-0.1, 0.1]$. We ran our algorithms for a total of 10000 iterations on the 5×5 navigation benchmark instance. Figure 2a contains a generated RRT for this benchmark instance.

Heater Benchmark. The hybrid automaton for the heater benchmark is shown in Fig. 1. Again, we ran our algorithm for 10000 iterations, which resulted in Fig. 2b.



(a) (b)

Fig. 2. (a) RRT generated for a 5×5 instance of the navigation benchmark; (b) RRT for the heater benchmark. We use different colors to illustrate the tree growth in every iteration. (Color figure online)

For both the considered benchmark classes, we observe a rather uniform state space coverage, which confirms the validity of our implementation.

6 Conclusion

In this paper, we have described our experiences with RRTs for hybrid automata. In order to account for possible non-determinism due to discrete mode switches, we have introduced the notion of important points, which intuitively captures time moments where a mode invariant becomes invalid or transitions become enabled/disabled. The evaluation shows that our algorithms lead to a reasonable state space coverage.

Acknowledgment. This work was partly supported by the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE/SHiNE) and Z211-N23 (Wittgenstein Award) and by the ARC project DP140104219 “Robust AI Planning for Hybrid Systems”.

References

1. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.-H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoret. Comput. Sci. **138**(1), 3–34 (1995)
2. Bak, S., Bogomolov, S., Henzinger, T.A., Johnson, T.T., Prakash, P.: Scalable static hybridization methods for analysis of nonlinear systems. In: 19th International Conference on Hybrid Systems: Computation and Control (HSCC 2016), pp. 155–164. ACM (2016)
3. Bak, S., Bogomolov, S., Johnson, T.T.: HyST: a source transformation and translation tool for hybrid automaton models. In: 18th International Conference on Hybrid Systems: Computation and Control, Seattle, Washington. ACM, April 2015
4. Bhatia, A., Frazzoli, E.: Incremental search methods for reachability analysis of continuous and hybrid systems. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 142–156. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24743-2_10
5. Bogomolov, S., Frehse, G., Grosu, R., Ladan, H., Podelski, A., Wehrle, M.: A box-based distance between regions for guiding the reachability analysis of SpaceEx. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 479–494. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31424-7_35
6. Bogomolov, S., Greitschus, M., Jensen, P.G., Larsen, K.G., Mikucionis, M., Strump, T., Tripakis, S.: Co-simulation of hybrid systems with SpaceEx and Uppaal. In: 11th International Modelica Conference (Modelica 2015), Linköping Electronic Conference Proceedings, pp. 159–169. Linköping University Electronic Press, Linköpings universitet (2015)
7. Bogomolov, S., Mitrohin, C., Podelski, A.: Composing reachability analyses of hybrid systems for safety and stability. In: Bouajjani, A., Chin, W.-N. (eds.) ATVA 2010. LNCS, vol. 6252, pp. 67–81. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15643-4_7
8. Cheng, P., LaValle, S.M.: Resolution complete rapidly-exploring random trees. In: ICRA, pp. 267–272. IEEE (2002)
9. Dang, T., Nahhal, T.: Randomized simulation of hybrid systems for circuit validation. Technical report (2006)
10. Esposito, J.M., Kim, J., Kumar, V.: Adaptive RRTs for validating hybrid robotic control systems. In: Erdmann, M., Overmars, M., Hsu, D., van der Stappen, F. (eds.) Algorithmic Foundations of Robotics VI, pp. 107–121. Springer, Heidelberg (2005). doi:10.1007/10991541_9
11. Fehnker, A., Ivančić, F.: Benchmarks for hybrid systems verification. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 326–341. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24743-2_22
12. Frehse, G., et al.: SpaceEx: scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22110-1_30
13. Lavalle, S.M.: Rapidly-exploring random trees: a new tool for path planning. Technical report (1998)
14. Plaku, E., Kavraki, L., Vardi, M.: Hybrid systems: from verification to falsification by combining motion planning and discrete search. Formal Methods Syst. Des. **34**, 157–182 (2009)