University of Pennsylvania
ScholarlyCommons

Departmental Papers (CIS) | Department of Computer & Information Science

4-1-2007

Robust Test Generation and Coverage for Hybrid Systems

Agung Julius
University of Pennsylvania, agung@seas.upenn.edu

Georgios E. Fainekos
University of Pennsylvania, fainekos@seas.upenn.edu

Madhukar Anand
University of Pennsylvania, anandm@cis.upenn.edu

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

George Pappas
University of Pennsylvania, pappasg@cis.upenn.edu

Postprint version. Published in *Lecture Notes in Computer Science*, Proceedings of the 10th International Conference on Hybrid Systems: Computation and Control (HSCC), April 2007, pgs 329-342.

This paper is posted at ScholarlyCommons. http://repository.upenn.edu/cis_papers/336
For more information, please contact repository@pobox.upenn.edu.

Robust Test Generation and Coverage for Hybrid Systems

Abstract
Testing is an important tool for validation of the system design and its implementation. Model-based test generation allows to systematically ascertain whether the system meets its design requirements, particularly the safety and correctness requirements of the system. In this paper, we develop a framework for generating tests from hybrid systems' models. The core idea of the framework is to develop a notion of robust test, where one nominal test can be guaranteed to yield the same qualitative behavior with any other test that is close to it. Our approach offers three distinct advantages: 1) It allows for computing and formally quantifying the robustness of some properties; 2) it establishes a method to quantify the test coverage for every test case; and 3) the procedure is parallelizable and therefore, very scalable. We demonstrate our framework by generating tests for a navigation benchmark application.

Comments
Postprint version. Published in *Lecture Notes in Computer Science*, Proceedings of the 10th International Conference on Hybrid Systems: Computation and Control (HSCC), April 2007, pgs 329-342.

This conference paper is available at ScholarlyCommons: http://repository.upenn.edu/cis_papers/336

Robust Test Generation and Coverage for Hybrid Systems

A Agung Julius[1], Georgios E. Fainekos[2], Madhukar Anand[2], Insup Lee[2], and George J. Pappas[1]

[1] University of Pennsylvania, Department of Electrical and Systems Engineering, 200 South 33rd Street, Philadelphia PA-19104, USA.
{agung,pappasg}@seas.upenn.edu

[2] University of Pennsylvania, Department of Computer and Information Sciences, 200 South 33rd Street, Philadelphia PA-19104, USA.
{fainekos,anandm,lee}@seas.upenn.edu

Abstract. Testing is an important tool for validation of the system design and its implementation. Model-based test generation allows to systematically ascertain whether the system meets its design requirements, particularly the safety and correctness requirements of the system. In this paper, we develop a framework for generating tests from hybrid systems' models. The core idea of the framework is to develop a notion of robust test, where one nominal test can be guaranteed to yield the same qualitative behavior with any other test that is close to it. Our approach offers three distinct advantages. 1) It allows for computing and formally quantifying the robustness of some properties, 2) it establishes a method to quantify the test coverage for every test case, and 3) the procedure is parallelizable and therefore, very scalable. We demonstrate our framework by generating tests for a navigation benchmark application.

1 Introduction

As engineering systems gain more functionality and complexity, there is a need for sound discipline in their design, development and deployment. In particular, ensuring the safety and correctness of these large and complex systems is becoming increasingly hard. In recent years, a slew of model-based design efforts have been developed to address these problems. The promise of the model-based design paradigm is to develop design models and subject them to analysis, simulation, and validation prior to their implementation. Performing analysis early in the development cycle allows one to detect and fix design problems sooner and at a lower cost. There has been a lot of work [1–8] in the hybrid systems community toward analysis, validation and verification of systems developed from hybrid control models. The list [1–8] is by no means exhaustive. However, it does capture a broad spectrum of techniques that have been developed in the community to answer the reachability and verification problems.

Testing has been used in practice to check the conformance of an implementation to its specification. Although testing cannot provide formal guarantees on

correctness and reachability as it is possible with verification, disciplined use of testing, coupled with coverage criteria can be a great aid to system verification and validation.

Testing amounts to running or simulating the operation of the system for a finite period of time. It is comparable to taking a snapshot of the operation of the system. As we are interested in gaining some information about the system, testing is done repetitively with varying *testing parameters*, so as to simulate as many scenarios of operation as possible. By testing parameters, we mean the parameters that characterize the run of a test. For example, if we have an autonomous system whereof we can only influence the initial condition, then the testing parameter is the initial condition. If we have more degrees of freedom in influencing the execution of the system, for example, if we can also adjust some parameters in the system, then these parameters can be regarded as testing parameters as well. The ultimate goal of testing is to cover the entirety of the set of testing parameters.

When the set of testing parameters is an infinite set, it is obvious that we cannot exhaustively test each of the testing parameters. However, it is possible that one testing parameter is representative of many others. A testing parameter is said to be *robust* if a slight (quantifiable) perturbation of the parameter is guaranteed to result in a test with the same qualitative properties (for example, safety and correctness). It is obvious that robustness can lead to a significant reduction in the set of testing parameters. In fact, ideally, we would like to be able to reduce an infinite set of testing parameters into a finite set, and quantify the coverage by the performed tests. In this paper, we develop a framework where the robustness of a test can be formally quantified and computed. The framework is then applied to test a navigation benchmark problem [9].

Prior work on generating tests from hybrid systems' models has mainly focused on randomized testing or monitoring to check whether an implementation conforms to its model. Esposito [10] and Branicky et. al. [11] use Rapidly exploring Random Trees (RRT) to generate test cases from hybrid systems models. Another paper by van Osch [12] describes testing for input-output conformance by providing inputs to the implementation and comparing its outputs to those of its model. In [13], the author presents a case-study that identifies a minimal set of test scenarios required to determine, with some confidence interval, if the system meets the specification by casting the test generation problem as an optimal control problem. This approach suffers from the drawback that it is only applicable in scenarios where the optimal control problem can be solved efficiently. Other publications in this area include [14], where the authors present an integrated framework to test and monitor code generated from hybrid models, and [15], where test generation from Extended Finite State Machines (EFSM) is developed in order to test temporal logic properties.

2 Problem formulation

In this paper, we consider a standard model of a hybrid automaton [16], $\mathcal{H} = (\mathcal{X}, \mathcal{L}, E, Inv, F)$, where $\mathcal{X}$ is the continuous state space of the system, $\mathcal{L}$ is the finite set of discrete states (locations), E is the set of transitions, $Inv : \mathcal{L} \to 2^{\mathcal{X}}$ is the invariant set of each location, and $F : \mathcal{X} \times \mathcal{L} \to \mathcal{X}$ is the vector field that defines the continuous dynamics in each location.

A transition $e \in E$ is a 4-tuple (l, l', g, r), where $l \in \mathcal{L}$ is the origin of the transition, $l' \in \mathcal{L}$ is the target of the transition, $g \subset \partial Inv(l)$ is the guard of the transition, which is a subset of the boundary of the invariant set of location l, and $r : g \to Inv(l')$ is the reset map that resets the continuous state at the new location. Here, we assume that the reset map r is continuous.

In this paper, we shall assume that the following statements hold. The state space is $\mathbb{R}^n$ and the invariant sets are closed. We denote the open interior of an invariant set as $\underline{Inv}(l)$ and we assume that the differential equation

$$\frac{dx}{dt} = F(x(t), l),$$

admits a unique solution for every location $l \in \mathcal{L}$, i.e. it satisfies the Lipschitz conditions. The transitions are deterministic[3] in the sense that the guards are forcing and all outgoing transitions from a location have disjoint guards. Finally, the system does not deadlock or possess Zeno behavior.

In analyzing the safety of the system, we assume that there is a subset $\texttt{Unsafe} \subset \mathcal{X} \times \mathcal{L}$ unsafe states. A trajectory of the hybrid system corresponds to an unsafe execution if it intersects with the unsafe set.

Example 1 (Navigation Benchmark [9]). As a case study in this paper, we consider a slightly modified version of the navigation benchmark proposed by Fehnker and Ivancic [9]. The benchmark studies a hybrid automaton $\mathcal{H}$ with 3×3 discrete locations and 4 continuous variables x_1, x_2, v_1, v_2 that form the state vector $x = [x_1\ x_2\ v_1\ v_2]^T$. We refer to the vectors $[x_1\ x_2]^T$ and $[v_1\ v_2]^T$ as the position and the velocity of the system, respectively. The structure of the hybrid automaton can be better visualised in Fig. 1. The invariant set of every (i, j) location is an 1×1 box that constraints the position of the system, while the velocity can flow unconstrained. The guards in each location are the edges and the vertices that are common among the neighboring locations.

Each location has affine constant dynamics with drift. In detail, in each location (i, j) of the hybrid automaton, the system evolves under the differential equation $\dot{x} = Ax - Bu(i, j)$ where the matrices A and B are

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1.2 & 0.1 \\ 0 & 0 & 0.1 & -1.2 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ -1.2 & 0.1 \\ 0.1 & -1.2 \end{bmatrix}$$

and the input in each location is $u(i, j) = [\sin(\pi C(i, j)/4)\ \ \cos(\pi C(i, j)/4)]^T$. The array C is one of the two parameters of the hybrid automaton that the user

[3] We limit the discussion in this paper to deterministic guards. However, the framework presented here is also applicable to nondeterministic guards.

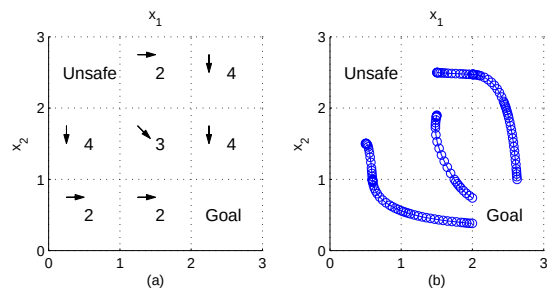

Fig. 1. A graphical representation of the benchmark hybrid automaton. The upper left box is the invariant set for the location $(1, 1)$. (a) The constant input vector in each location. (b) Sample trajectories for different initial conditions.

can control and it defines the input vector in each discrete location. Here, we consider the following input arrays:

$$C_1 = \begin{bmatrix} U & 2 & 4 \\ 4 & 3 & 4 \\ 2 & 2 & G \end{bmatrix} \quad C_2 = \begin{bmatrix} 2 & 3 & 6 \\ 3 & 3 & G \\ 2 & 2 & U \end{bmatrix} \quad C_3 = \begin{bmatrix} U & 2 & 4 \\ 2 & 2 & 4 \\ 1 & 1 & G \end{bmatrix}$$

where U denotes the unsafe set and G the goal set. The other user-input parameter is the set of initial conditions $\mathcal{X}_0 \times \mathcal{L}_0 \subseteq \mathcal{X} \times \mathcal{L}$. The requirement for $\mathcal{H}$ is that all of its trajectories starting in $\mathcal{X}_0 \times \mathcal{L}_0$ should avoid the unsafe set and eventually reach the goal set. Sample trajectories of the system appear in 1.(b).

Example 1 describes a typical verification problem for hybrid systems. The goal of exhaustive verification algorithms is to prove that there cannot exist a trajectory that falsifies the hybrid automaton assumptions, i.e. safety and reachability. In this paper, we try to solve a different problem in an attempt to overcome the theoretical and practical difficulties of exhaustive verification. Here, the target is not complete coverage of the set of initial conditions, but the computation of a (possibly) quick estimate of which part of the initial conditions is safe and/or unsafe for a bounded horizon using only a small number of tests. One of the most important aspects of such a testing methodology is that it should be completely transparent to the user with no (or very few) parameters to tune.

Problem 1 (Testing the benchmark example). Given the hybrid automaton $\mathcal{H}$ of Example 1 with a set of initial conditions $\mathcal{X}_0 \times \mathcal{L}_0$, a bounded horizon $T > 0$ and an unsafe `Unsafe` and/or `Goal` set, develop a strategy for picking test points in order to cover the set of initial conditions.

As mentioned in the previous section, we want to cover the whole set of initial conditions with finitely many test points. This requires the construction of robust neighborhoods around the test points. Each such neighborhood contains a set of points which initiate trajectories that have the same qualitative properties as the trajectory generated by the actual test point. By qualitative properties, we mean the sequence of locations that are visited and the safety property.

3 Robust testing for hybrid systems

In this section, we discuss the computation of robust neighborhoods of initial conditions. First, we are going to review the theory of bisimulation functions for dynamical systems [17]. The concept of bisimulation functions introduced in [17] is more general than what we are going to review here since we only consider systems without input.

Let $\phi : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ be a bisimulation function between a dynamical system

$$\Sigma : \tfrac{dx}{dt} = F(x),\ x \in \mathcal{X} \tag{1}$$

and itself. Such a function ϕ satisfies the following requirements:

$$\phi(x_1, x_2) \geq 0, \tag{2}$$

$$\tfrac{\partial \phi(x_1,x_2)}{\partial x_1} f(x_1) + \tfrac{\partial \phi(x_1,x_2)}{\partial x_2} f(x_2) \leq 0, \tag{3}$$

for every $x_1, x_2 \in \mathcal{X}$.

Denote the continuous flow of the dynamical system Σ as $\xi : \mathbb{R}_+ \times \mathcal{X} \to \mathcal{X}$, that is, $\xi(t, x_0)$ satisfies the differential equation

$$\tfrac{\partial \xi(t,x_0)}{\partial t} = f(\xi(t, x_0)),\ \xi(0, x_0) = x_0. \tag{4}$$

Note that the bisimulation function is nonincreasing with respect to the flow.

Proposition 1 ([17]). *For any $x_1^0, x_2^0 \in \mathcal{X}$, the bisimulation function evaluated along the flows of the initial conditions x_1^0 and x_2^0 is nonincreasing, i.e. for any $t_2 \geq t_1 \geq 0$ it is $\phi(\xi(t_1, x_1^0), \xi(t_1, x_2^0)) \geq \phi(\xi(t_2, x_1^0), \xi(t_2, x_2^0))$.*

We denote the ε-neighborhood (or ε-ball) of $x \in \mathcal{X}$ with respect to a bisimulation function ϕ as $B_\phi(x, \varepsilon)$, i.e. $B_\phi(x, \varepsilon) = \{y \in \mathcal{X} \mid \phi(x, y) \leq \varepsilon\}$. The following corollary is a direct consequence of Proposition 1.

Corollary 1. *For any $x, y \in \mathcal{X}$, if $y \in B_\phi(x, \varepsilon)$ for some $\varepsilon > 0$, then for every $t \geq 0$ it is $\xi(t, y) \in B_\phi(\xi(t, x), \varepsilon)$.*

Thus, the ε-neighborhood defined by the bisimulation function ϕ is invariant with respect to the flow of the dynamical system. If we define the (directed) metric $d_\phi(x(\cdot), y(\cdot))$ between different state trajectories of the system Σ with respect to the bisimulation function ϕ as

$$d_\phi(x(\cdot), y(\cdot)) := \sup_{t \geq 0} \phi(x(t), y(t)),$$

then the corollary above is equivalent to $d_\phi(\xi(\cdot, x), \xi(\cdot, y)) \leq \phi(x, y)$ for any $x, y \in \mathcal{X}$. Hereunder, we shall assume that bisimulation functions are symmetric, that is, $\phi(x, y) = \phi(y, x)$. A bisimulation function that is symmetric and forms a metric on the space $\mathcal{X}$ is called a *contraction metric*. Such functions are used in contraction analysis in relation to the stability of a system [18, 19].

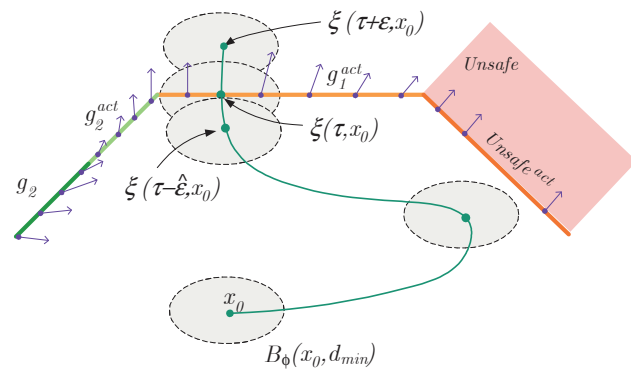

Fig. 2. An illustration for Definition 1 and Proposition 2 for $j = 1$. The line on the left is the guard g_2. Its part with lighter shade, g_2^{act} is the active part, where the vector field points outward. The guard g_1 is active everywhere, as the vector field there points outward. The boundary of the unsafe set in this picture is active since the vector field points into the unsafe set.

When the dynamics are affine,

$$F(x) = Ax + b \text{ for } x \in \mathbb{R}^n, A \in \mathbb{R}^{n\times n}, b \in \mathbb{R}^{n\times 1},$$

we can propose that the bisimulation function assumes the form

$$\phi(x_1, x_2) = (x_1 - x_2)^T M(x_1 - x_2),$$

where M is a positive semidefinite matrix. Thus, the bisimulation function defines a Euclidean metric in a (linearly) transformed space. It can be shown that such a bisimulation function is essentially a Lyapunov function, and it exists if and only if the system is stable [20].

In the following, we are going to construct robust testing neighborhoods using the level sets of a bisimulation function. For that, we need a few definitions.

Definition 1. *For any location $l \in \mathcal{L}$ we define the set of outgoing transitions from l as $Out(l) \subseteq E$. For any transition $e = (l, l', g, r) \in Out(l)$, we denote by g^{act} the active part of the guard g, which is the part $g^{act} \subset g$ of the guard that can be reached from inside $Inv(l)$, i.e. we exclude from g the points where the vector field $F(\cdot, l)$ points inward. Similarly, we define* $\texttt{Unsafe}^{act}$ *to be the portion of the boundary of the unsafe set that is reachable from the safe portion of the state space.*

See Fig. 2 for an illustration of the definition of the active guard and of the proposition below.

Proposition 2. *Let $x_0 \in Inv(l)$ for some location $l \in \mathcal{L}$, and assume that the state trajectory $\xi(t, x_0)$ lies entirely in $Inv(l)\backslash\texttt{Unsafe}$ for $t \leq \tau$. Suppose that*

$Out(l) = \{e_1, e_2, \cdots, e_n\}$ and that g_i is the guard of e_i, $i = 1, \ldots, n$. Let τ be the time when the state trajectory hits a guard g_j, which is the guard of the transition e_j for some $j \in \{1, \ldots, n\}$. Suppose that we have a bisimulation function ϕ for the continuous dynamics in location l. We also assume that there is a positive time lag $\varepsilon > 0$ such that $\xi(\tau + \varepsilon, x_0) \notin Inv(l)$. We define

$$\begin{aligned} d_{out} &:= \inf_{y \in g_j} \phi(\xi(\tau + \varepsilon, x_0), y), \\ d_i &:= \inf_{0 \le t \le \tau + \varepsilon} \inf_{y \in g_i^{act}} \phi(\xi(t, x_0), y),\ i \in \{1, \ldots, n\}/\{j\}, \\ d_{unsafe} &:= \inf_{0 \le t \le \tau + \varepsilon} \inf_{y \in Inv(l) \cap \mathtt{Unsafe}^{act}} \phi(\xi(t, x_0), y), \\ d_{\min} &:= \min\{d_{out}, d_{unsafe}, d_1, \ldots, d_{j-1}, d_{j+1}, \ldots, d_n\}, \\ \hat{\varepsilon} &:= \inf\left\{\delta > 0 \mid B_\phi(\xi(\tau - \delta, x_0), d_{\min}) \subset \underline{Inv}(l)\right\}. \end{aligned}$$

The following statement holds. For any $x_0' \in B_\phi(x_0, d_{\min}) \cap Inv(l)$, the state trajectory $\xi(t, x_0')$ exits $Inv(l)$ through transition e_j at time $t \in [\tau - \hat{\varepsilon}, \tau + \varepsilon]$ and is safe at least until it exits location l.

Proof. See Fig. 2 for an illustration. By construction of $d_{\min}$, we can infer that for any $t \in [0, \tau + \varepsilon]$ and $i \in \{1, \ldots, n\}/\{j\}$, $B_\phi(\xi(t, x_0), d_{\min}) \cap g_i^{act} = \emptyset$, and $B_\phi(\xi(t, x_0), d_{\min}) \cap \mathtt{Unsafe}^{act} \cap Inv(l) = \emptyset$.

We then invoke Corollary 1 and infer that any state trajectory originating in $B_\phi(x_0, d_{\min})$ will not be unsafe nor touch the active guards g_i^{act}, $i \in \{1, \ldots, n\}/\{j\}$, within the time interval $[0, \tau + \varepsilon]$. We also know that the neighborhood $B_\phi(\xi(\tau + \varepsilon, x_0), d_{\min})$ lies entirely outside of $Inv(l)$, beyond g_j. This implies that any trajectory starting in $B_\phi(x_0, d_{\min}) \cap Inv(l)$ crosses g_j before $t = \tau + \varepsilon$. Finally, since the neighborhood $B_\phi(\xi(t, x_0), d_{\min})$ does not touch any active guard, for $t \in [0, \tau - \hat{\varepsilon})$, we also know that the trajectories will not touch any active guard before time $t = \tau - \hat{\varepsilon}$.

Proposition 2 provides us with a way to compute a neighborhood around the initial state x_0, which consists of initial states that have the same qualitative behavior as x_0. Namely, they lead to a trajectory that exits location l by taking the same transition and which is safe at least until it performs that transition. In addition to that, we obtain a timing guarantee in the form of a time interval where the transition is guaranteed to occur if the initial state belongs to the computed neighborhood. The next step is to design an algorithm that uses Proposition 2 repetitively in order to deal with trajectories that take multiple transitions.

Given a hybrid automaton $\mathcal{H} = (\mathcal{X}, \mathcal{L}, E, Inv, F)$. We denote the continuous flow at every location $l \in L$ as $\xi_l(\cdot, \cdot)$, and we assume that we have a bisimulation function for the dynamics in location $l \in L$, which is $\phi_l(\cdot, \cdot)$. A testing trajectory is a sequence $(x_i, l_i, e_i, \tau_i)_{i=0,\cdots,N}$ such that:

- $l_i \in L, x_i \in Inv(l_i), e_i \in Out(l_i), \tau_i > 0$, for every $i \in \{0, 1, \ldots, N\}$,
- If we define $e_i = (l_i, l_{i+1}, g_i, r_i)$, then $\xi_{l_i}(\tau_i, x_i) \in g_i$, $x_{i+1} = r_i(\xi_{l_i}(\tau_i, x_i))$, $\xi_{l_i}(t, x_i) \in \underline{Inv}(l_i)$ for all $t \in [0, \tau_i)$, for every $i \in \{0, 1, \ldots, N-1\}$,

We define $T := \sum_{i=0}^{N-1} \tau_i$, which is the time where the trajectory enters the final state. The length of the test is $T + \tau_N$. Given a testing trajectory, the algorithm for constructing a robust tube around a nominal trajectory is given as follows.

Algorithm 1 *The following are the steps:*

1. *Define the avoided set as the union of the unsafe set and active parts of all the outgoing guards from l_N, i.e.*[4]

$$D_N := \texttt{Unsafe}^{act} \cup_{g \in Out(l_N)} g^{act}. \tag{5}$$

2. *Compute (or obtain a lower bound on)*

$$d^N_{\min} := \inf_{t \leq \tau_N} \inf_{y \in D_N} \phi_{l_N}(\xi_{l_N}(t, x_N), y). \tag{6}$$

3. *Define the allowed guard*

$$A_{N-1} := r^{-1}_{N-1}(r_{N-1}(g_{N-1}) \cap B_{\phi_{l_N}}(x_N, d^N_{\min})). \tag{7}$$

This is the set of states on the guard of the transition between l_{N-1} and l_N that is reset into the $d^N_{\min}-$ neighborhood of x_N (with respect to the bisimulation function ϕ_{l_N}).

4. *Define the avoided set*

$$D_{N-1} := (\texttt{Unsafe}^{act} \cup_{g \in Out(l_{N-1})} g^{act}) \backslash A_{N-1}. \tag{8}$$

5. *Pick a time lag $\varepsilon_{N-1} > 0$ such that*

$$\xi_{l_{N-1}}(\tau_{N-1} + \varepsilon_{N-1}, x_{N-1}) \notin Inv(l_{N-1}).$$

We present an algorithm for picking a good time lag later in this paper.

6. *Compute (or obtain a lower bound on)*

$$d^{N-1}_{\min} := \min \Bigg(\inf_{y \in g_{N-1}} \phi_{l_{N-1}}(\xi_{l_{N-1}}(\tau_{N-1} + \varepsilon_{N-1}, x_{N-1}), y), \\ \inf_{t \leq \tau_{N-1} + \varepsilon_{N-1}} \inf_{y \in D_{N-1}} \phi_{l_{N-1}}(\xi_{l_{N-1}}(t, x_{N-1}), y) \Bigg).$$

7. *Define*

$$\hat{\varepsilon}_{N-1} := \inf \left\{ \delta > 0 \mid B_{\phi_{N-1}}(\xi_{l_{N-1}}(\tau_{N-1} - \delta, x_{N-1}), d^{N-1}_{\min}) \subset \underline{Inv}(l_{N-1}) \right\}.$$

8. *Repeat steps 3 - 7 to obtain A_i, D_i, ε_i, $d^i_{\min}, \hat{\varepsilon}_i$, $i = 0, 1, \ldots, N-2$.*

[4] Notice that for simplicity, we abuse the notation and associate the transition with its guard.

A property of the result of this iteration is presented in the following theorem, whose proof can essentially be constructed by repeated application of Proposition 2.

Theorem 2. *Given a testing trajectory of a hybrid system* $(x_i, l_i, e_i, \tau_i)_{i=0,\cdots,N}$, *let* $d^0_{\min}$, ε_i, $\hat{\varepsilon}_i$, $i = 0, 1, \ldots, N-1$ *be obtained from the iteration in Algorithm 1. Define*

$$\varepsilon := \sum_{i=0}^{N-1} \varepsilon_i, \ \hat{\varepsilon} := \sum_{i=0}^{N-1} \hat{\varepsilon}_i.$$

Any testing trajectory that starts in $B_{l_0}(x_0, d^0_{\min})$ *has the following properties.*
(i) It follows the same sequence of locations, $(l_i)_{i=0,\ldots,N}$ *and it enters the final location* l_N *at* $t \in [T - \hat{\varepsilon}, T + \varepsilon]$,
(ii) The trajectory is safe at least until τ_N *time unit after it enters* l_N.

An essential part of Algorithm 1 is the generation of the time lags ε_i (see Step 5). First of all, notice that a small ε_i is more desirable than a larger one. This is because ε_i is a measure in the slackness in the timing when the trajectories in the tube hit the desired guard (see Theorem 2). The idea is to construct ε_i as small as possible, but large enough so that by introducing this time lag, we are sure that all the trajectories in the constructed tube hit the desired guard within the time interval $[\tau_i, \tau_i + \varepsilon_i]$. In order to do this, we can replace Steps 5 and 6 in Algorithm 1 with the following steps.

Step 5'. Compute

$$\hat{d}^{N-1}_{\min} := \inf_{t \le \tau_{N-1}} \inf_{y \in D_{N-1}} \phi_{l_{N-1}}(\xi_{l_{N-1}}(t, x_{N-1}), y).$$

Step 5". Compute

$$\varepsilon_{N-1} = \min\Big(\inf\Big\{e \mid \inf_{y \in g_{N-1}} \phi_{l_{N-1}}(\xi_{l_{N-1}}(\tau_{N-1} + e, x_{N-1}), y) \ge \hat{d}^{N-1}_{\min}, \\ \xi_{l_{N-1}}(\tau_{N-1} + e, x_{N-1}) \notin Inv(l_{N-1})\Big\}, \varepsilon_{\max}\Big).$$

Step 6'. If $\varepsilon_{N-1} < \varepsilon_{\max}$ then $d^{N-1}_{\min} = \hat{d}^{N-1}_{\min}$, otherwise

$$d^{N-1}_{\min} = \sup_{0 \le e \le \varepsilon_{\max}} \inf_{y \in g_{N-1}} \phi_{l_{N-1}}(\xi_{l_{N-1}}(\tau_{N-1} + e, x_{N-1}), y).$$

In Step 5' we compute the largest level set that fits within the allowed set. In Step 5", we want to find the minimum time lag such that the computed level set lies entirely beyond the desired guard (and hence outside of the invariant set $Inv(l_{N-1})$). See Fig. 2 for an illustration. Because such time lag might not exist, or is too large, we can establish a maximum allowed value for the time lag, $\varepsilon_{\max}$. If such time lag is found and is smaller than $\varepsilon_{\max}$, then this value is used. If it is not found, then we compute the largest level set that can be fit outside of the invariant set. This is done in Step 6'.

4 Test generation and coverage strategies

In the benchmark problem that we are working on, our goal is to cover the given set of initial states with robust neighborhoods. In the previous section, we have presented an algorithm for computing the robust neighborhood around a given initial state. What needs to be done next is to select subsequent initial states from the given set, so as to (eventually) cover the whole set and/or to provide a quantitative measure of coverage based on the executed tests. The strategy for selecting the test points is called the *test generation*.

An important issue in test generation is the notion of coverage, which qualitatively characterizes the number and the type of tests generated. There are a number of coverage criteria based on the test requirement, which can be categorized into two classes: initial state coverage and structural coverage. The first type of coverage criteria is concerned with covering the set of initial states and characterizing each test case that has been generated. The second class of coverage criteria is concerned with analyzing the structural coverage of a test trajectory, such as location coverage and transition coverage. This notion of coverage can capture more aspects of the execution than just coverage of the initial states. The main challenge here is how to generate tests so as to meet particular coverage criteria. In this paper, we are only concerned with the coverage of the set of initial states and leave the prospect of using our framework to analyze structural coverage as future work.

There are a number of strategies for initial state coverage:

Randomized Strategy: The first strategy for covering the set of initial states is to pick points randomly. Consequently, it is hardly possible to guarantee efficient coverage. However, a randomized strategy might be an attractive option because of its simplicity.

Greedy Strategy: Under this strategy, we first pick a point and run the testing algorithm with it. Then, we subtract the computed robust ball around the initial point from the set of initial states and pick the center of the maximum ball that can be fitted into the remaining space as the next test point.

Tessellation-based Strategy: Picking points at random may not ensure uniform coverage. One possible strategy to ensure uniform coverage is to use tessellation of the initial state space based on an appropriate metric. This strategy does not scale well as the dimension of the state space increases.

Minimal Dispersal-based Strategy: Picking points so as to minimize the *dispersion* [21] of the points in the set of initial states. This strategy involves the generation of weighted Voronoi diagrams in the set. The goal is to pick the points incrementally so as to maximize the radius of a non-overlapping ball that can be inserted in the set. We use this method to analyze the benchmark problem (see the following section).

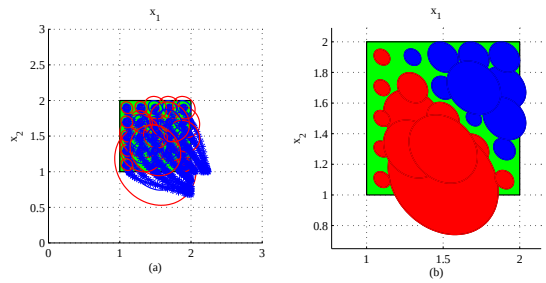

Fig. 3. Result after 25 simulations for the problem instance of Example 2.

5 Numerical Results and Discussion

In this section, we present some numerical results using our prototype MATLAB implementation of the robust testing algorithm. The experimental results will helps us discuss the strengths and weaknesses of our approach.

One of the advantages of using robust testing methodologies is that we can obtain an estimate of the degree of coverage of initial conditions that we have achieved. Theoretically, this can be done by computing the volume of the intersection of the robustness ellipsoid $\mathcal{E}$ with the polytope that defines the set of initial conditions $\mathcal{X}_0$. Nonetheless, this is not feasible computation-wise when the robustness ellipsoid $\mathcal{E}$ is not contained inside the set of initial conditions. Therefore, we compute the maximum ellipsoid that fits inside the intersection of $\mathcal{E}$ and $\mathcal{X}_0$. This can lead to a significant under-approximation of the actual covered space (see Example 3).

The following testing problem provides some inside on the principles behind our testing algorithm. The planar choice of initial conditions help us visualize the coverage of initial conditions, since the same is not possible when testing a 4D set of initial conditions.

Example 2. The first case that we consider is testing the navigation benchmark for the input array C_1 and the set of initial conditions $\mathcal{X}_0 = [1,2] \times [1,2] \times \{-0.2\} \times \{0\}$ with $\mathcal{L}_0 = \{(2,2)\}$ (*light gray* region in Fig. 3). Here instead of using the Minimal Dispersal-based Strategy, we create a grid of 25 points which serve as initial conditions for each simulation. The resulting simulations appear in Fig. 3.(a). The ellipsoids centered at the initial conditions denote the projections of the 4D ellipsoids on the position plane $x_1 - x_2$. In Fig. 3.(b), we present the covered space of initial conditions after 25 simulations. Here, the ellipsoids are the intersection of the corresponding 4D ellipsoids with the position plane. The *gray* and *black* ellipsoids denote covered initial conditions whose corresponding trajectories followed different discrete paths. Note that there exists a clear partition of $\mathcal{X}_0$ into two subsets of initial conditions that initiate trajectories that traverse different discrete paths. In this case, our proposed under-approximation algorithm for coverage computed 48% of covered initial conditions.

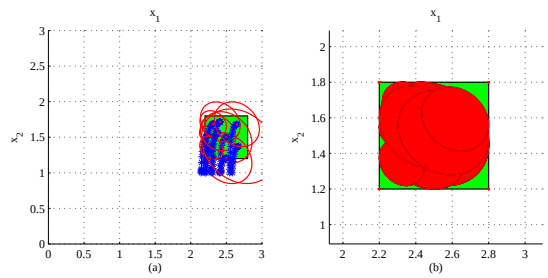

Fig. 4. Result after 9 simulations for the problem instance of Example 3.

The next example indicates that when the set of initial conditions is thin and the system is robust with respect to the specifications (unsafe and/or goal set), the testing problem becomes easier.

Example 3. Consider again C_1, but now with the following set of initial conditions $\mathcal{X}_0 = [2.2, 2.8] \times [1.2, 1.8] \times \{-0.2\} \times \{0\}$ with $\mathcal{L}_0 = \{(2, 3)\}$. This set of initial conditions has been verified to be safe with respect to the unsafe set in [9]. Using the testing algorithm we can cover the set of initial conditions with only 9 simulations (Fig. 4.(a)). In Fig. 4.(a) the ellipsoids represent the intersection of the corresponding 4D ellipsoids with the position plane, while in Fig. 4.(b) we present the under-approximation of the aforementioned ellipsoids with ellipsoids that fit inside $\mathcal{X}_0$. Numerically, we compute a coverage estimate of 72%.

The previous example also shows that even though by visual inspection we can verify that we have tested all the set of initial conditions, numerically we do not have an accurate way to confirm that. Next, we show the main strength of the testing framework, i.e. easy detection of robustly unsafe systems.

Example 4. Consider the input array C_2 with initial conditions $\mathcal{X}_0 = [0, 1] \times [2, 3] \times [-1, 1] \times [-1, 1]$ and $\mathcal{L}_0 = \{(1, 1)\}$. This was proven to be unsafe with just 10 simulations (see Fig. 5). Notice the complicated hybrid dynamics.

Finally, we apply our framework to a more demanding example.

Example 5. Here, we use input array C_3 with initial conditions $\mathcal{X}_0 = [0, 1] \times [0, 1] \times [-0.1, 0.5] \times [-0.05, 0.25]$ and $\mathcal{L}_0 = \{(3, 1)\}$. This example was proven to be safe in [22] using the verification toolbox PHAVer [7]. Our testing algorithm was able to cover 7% of the initial conditions after 300 simulations.

On-going research is focused on obtaining better estimates of the covered set of initial conditions. Finally, one of the main advantages of our robust testing framework is that it can be effectively parallelized by simply assigning a different test trajectory to each CPU.

6 Concluding remarks

In this paper, we presented an algorithm for test generation for hybrid systems. The algorithm is based on a computational method for robust testing. We implemented the algorithm to verify a navigation benchmark problem [9]. One

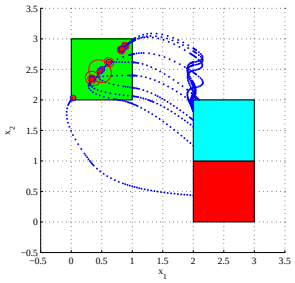

Fig. 5. The testing framework can potentially detect the unsafety of the system with just one test. Legend: *upper left square* - initial conditions, *lower right square* - unsafe set, *upper right square* - goal set.

advantage of our algorithm, compared to some other tools, is that we do not need to tune any parameter beforehand.

As future research agenda, we identify a number of potential directions. For example, we are going to develop a framework for robust testing of linear temporal logic properties [23], and develop a probabilistic notion of robust testing by using the idea of stochastic bisimulation function [24]. The algorithm that we presented in this paper is also able to provide a timing guarantee for the occurrence of the transitions. Although this feature is not exploited in the example that we presented in this paper, it can potentially be applied in automatic translation of hybrid automata into timed automata. Such a translation is useful for example in verification and observer design for hybrid systems [25].

Acknowledgments. This research is partially supported by the NSF PECASE 0132716, NSF EHS 0311123, NSF ITR 0121431, ARO MURI Grant DAAD 19-02-01-0383, NSF CNS 0410662, NSF CNS 0509327, NSF CNS 0509143, and the ARO W911NF-05-1-0182 research grants.

References

1. Dang, T., Maler, O.: Reachability analysis via face lifting. In: Hybrid Systems: Computation and Control. Volume 1386 of LNCS., Springer Verlag (1998) 96–109
2. Kurzhanski, A.B., Varaiya, P.: Ellipsoidal technique for reachability analysis. In: Hybrid Systems: Computation and Control. Volume 1790 of LNCS., Springer Verlag (2000) 202–214
3. Mitchell, I., Tomlin, C.J.: Level set methods in for computation in hybrid systems. In: Hybrid Systems: Computation and Control. Volume 1790 of LNCS., Springer Verlag (2000) 310–323
4. Alur, R., Dang, T., Ivancic, F.: Reachability analysis of hybrid systems via predicate abstraction. In: Hybrid Systems: Computation and Control. Volume 2289 of LNCS., Springer Verlag (2002) 35–48
5. Han, Z., Krogh, B.H.: Reachability analysis of hybrid control systems using reduced-order. In: Proc. American Control Conference. (2004) 1183–1189
6. Prajna, S., Jadbabaie, A.: Safety verification of hybrid systems using barrier certificates. In: Hybrid Systems: Computation and Control. Volume 2993 of LNCS., Springer Verlag (2004) 477–492

7. Frehse, G.: PHAVer: Algorithmic verification of hybrid systems past HyTech. In: Hybrid Systems: Computation and Control. Volume 3414 of LNCS., Springer Verlag (2005) 258–273
8. Girard, A.: Reachability of uncertain linear systems using zonotopes. In: Hybrid Systems: Computation and Control. Volume 3414 of LNCS., Springer Verlag (2005) 291–305
9. Fehnker, A., Ivancic, F.: Benchmarks for hybrid systems verification. In: Hybrid Systems: Computation and Control. Volume 2993 of LNCS., Springer Verlag (2004) 326–341
10. Esposito, J.M.: Randomized test case generation for hybrid systems: metric selection. In: Proc. 36th Southeastern Symposium of System Theory. (2004)
11. Branicky, M.S., Curtiss, M.M., Levine, J., Morgan, S.: RRTs for nonlinear, discrete, and hybrid planning and control. In: Proc. IEEE Conf. Decision and Control, Hawaii, USA (2003)
12. van Osch, M.: Automated model-based testing of χ simulation models with TorX. In: Quality of Software Architectures and Software Quality. Volume 3712 of LNCS., Springer Verlag (2005) 227–241
13. Esposito, J.M.: Automated test trajectory for hybrid systems. In: Proc. 35th Southeastern Symposium of System Theory. (2003)
14. Tan, L., Kim, J., Lee, I.: Testing and monitoring model-based generated program. Electronic Notes in Theoretical Computer Science **89**(2) (2003) http://www.elsevier.nl/locate/lntcs/volume89.html.
15. Hong, H.S., Lee, I., Sokolsky, O., Ural, H.: A temporal logic based theory of test coverage and generation. In: TACAS '02: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, London, UK, Springer-Verlag (2002) 327–341
16. Alur *et. al.*, R.: The algorithmic analysis of hybrid systems. Theoretical Computer Science **138** (1995) 3–34
17. Girard, A., Pappas, G.J.: Approximation metrics for discrete and continuous systems. accepted for publication on *IEEE Trans. Automatic Control* (March 2005)
18. Lohmiller, W., Slotine, J.J.E.: On contraction analysis for nonlinear systems. Automatica **34**(6) (1998) 683–696
19. Aylward, E., Parillo, P.A., Slotine, J.J.E.: Algorithmic search for contraction metrics via sos programming. In: Proc. American Control Conference, Minneapolis, USA (2006)
20. Brogan, W.L.: Modern control theory. Prentice Hall International, New Jersey (1991)
21. LaValle, S.M.: Planning algorithms. Cambridge University Press (2006)
22. Makhlouf, I.B., Kowalewski, S.: An evaluation of two recent reachability analysis tools for hybrid systems. In: Proc. IFAC Conf. Analysis and Design of Hybrid Systems, Alghero, Italy, IFAC (2006) 377–382
23. Fainekos, G.E., Girard, A., Pappas, G.J.: Temporal logic verification using simulation. In: Proceedings of FORMATS. Volume 4202 of LNCS., Springer (2006) 171–186
24. Julius, A.A.: Approximate abstraction of stochastic hybrid automata. In: Hybrid Systems: Computation and Control. Volume 3927 of LNCS., Springer Verlag (2006) 318–332
25. D'Innocenzo, A., Di Benedetto, M.D., Di Gennaro, S.: Observability of hybrid automata by abstraction. In: Hybrid Systems: Computation and Control. Volume 3927 of LNCS., Springer Verlag (2006) 169–183