Model-based Testing of Hybrid Systems

Thao Dang

December 6, 2010

Contents

Chapter 1

Model-based testing of hybrid systems

1.1 Introduction

Hybrid systems, that is, systems exhibiting both continuous and discrete dynamics, have proven to be a useful mathematical model for various physical phenomena and engineering systems. Due to the safety critical features of many such applications, much effort has been devoted to the development of automatic analysis methods and tools for hybrid systems, based on formal verification. Although these methods and tools have been successfully applied to a number of interesting case studies, their applicability is still limited to systems of small size because of the complexity of formal verification. It is thus clear that for systems of industrial size, one needs more light-weight methods. Testing is another validation approach, which can be used for much larger systems and is a standard tool in industry, although it can only reveal an error but does not permit proving its absence. A question of great interest is thus to bridge the gap between the verification and testing approaches, by defining a formal framework for testing of hybrid systems and developing methods and tools that help automate the testing process.

In this work we adopt a model-based testing approach. This approach allows the engineer to perform validation during the design, where detecting and correcting errors on a model are less expensive than on an implementation.

We first briefly review related work in model-based testing. The development of the first model-based testing frameworks was motivated by digital circuit testing and is based on Mealy machines [30]. More recently, frameworks based on other models, such as finite labeled transition systems, were proposed (see for example [39]). These models are of asynchronous nature and appropriate for the applications in communication protocols. Another important application area is software testing for which models, such as flow graphs, and coverage techniques have been used [20]. Recently, model-based testing has been extended to real-time systems. Timed automata have become a popular model for modelling and verifying real-time systems during the past decade, and a number of methods for testing real-time systems based on variants of this model or other similar models (such as timed Petri nets) have been proposed (e.g., see [26, 34, 8]). Although the practice of testing, especially in industry, is still empirical and ad-hoc, formal testing has become progressively accepted [17]. This is, on the one hand, due to the success of the formal techniques in a number of domains (such as model-checking of digital circuits) and, on the other hand, due to the development of commercial tools for automatic test generation. Among these tools, we can mention: Telelogic TestComposer (http://www.telelogic.com) for SDL models, Reactis Tester (http://www.reactive-systems.com) for Simulink® models (http://www.mathworks.com), Conformiq Test Generator (http://www.conformiq.com) for UML State-Chart models.

Concerning hybrid systems, model-based testing is still a new research domain. We defer a discussion on related work on hybrid systems testing to Section 1.14. A number of special characteristics of hybrid systems make their testing particularly challenging, in particular:

- *Combination of the complexity in both discrete and continuous aspects.* While continuous systems have been well studied in control theory and continuous mathematics, and discrete systems have been investigated in computer science, the interaction between continuous and discrete dynamics leads to fundamental problems (such as undecidability) which are not yet well understood or for which a general solution is often impossible.
- *Infiniteness of the state space of a hybrid system and of the input space.* In general, in order to test an open system, one first must feed an input signal to the system and then

check whether the behavior of the system induced by this input signal is as expected. When there is an infinite number of possible input signals, it is important to choose the ones that lead to interesting scenarios (with respect to the property/functionality to test).

To deal with these issues, we take an approach that draws on ideas from two domains: the algorithmic analysis methodology from computer science and methods from control theory. To model hybrid systems, we use hybrid automata [22]. This model, which can be roughly described as an extension of automata with continuous variables evolving according to differential equations, is a mathematical model largely used by computer scientists and control engineers to reason about problems related to hybrid systems. In addition, this model is expressive enough to describe complex hybrid phenomena arising in numerous applications, and its well-defined semantics permits accurate interpretation of testing results.

The main results we present in this chapter can be summarized as follows.

- **Formal framework for conformance testing of hybrid systems**. This framework uses the commonly-accepted hybrid automaton model and allows, on the one hand, to formally reason about the relation between a system under test and a specification, and on the other hand, to develop test generation algorithms.
- **Novel test coverage measure**. This is a challenging problem in testing. Intuitively, test coverage is a way to characterize the relation between the number and the type of tests to execute and the portion of the system's behavior effectively tested. The classical notions of coverage, introduced mainly for software testing (such as statement coverage, if-then-else branche coverage, path coverage) are unsuitable for the behaviors of a hybrid system defined as solutions of some differential equations. We thus propose a novel coverage measure, which on the one hand reflects our goal of testing safety and reachability properties and, on the other hand, can be efficiently computed. This measure is based on the equidistribution degree of a set of states over the state space and furthermore can be used to guide the test generation process.
- **Coverage-guided test generation**. We first propose a test generation algorithm that

is based on the RRT (Rapidly-exploring Random Tree) algorithm [28], a probabilistic motion planning technique in robotics. This RRT algorithm has been successful in finding feasible trajectories in motion planning. We then include in this algorithm a procedure for guiding the test generation process using the above mentioned coverage measure. Furthermore, we introduce a new notion of disparity between two point sets, in order to tackle "blocking" situations the test generation algorithm may enter. Indeed, in order to increase the coverage, the algorithm may try to explore the directions that are not reachable by the system's dynamics. We can detect such situations by comparing the distribution of the goal states and the visited states, using their disparity. If the disparity is large, it means that the visited states do not follow the goal states. This indicates that the goal states may not reachable and we should change the goal state sampling strategy.

- **Actuator and sensor imprecision**. Because of the limitations of practical actuators and sensors, the tester cannot realize exactly an input value specified as a real-valued vector as well as measure exactly the state of the system. We handle this using sensitivity analysis.

- **Tool development.** We have implemented a tool for conformance testing of hybrid systems, called **HTG**. The core of the tool is the implementation of the coverage-guided test case generation algorithm and the methods for estimating the coverage and disparity measures.

- **Applications**. Besides traditional applications of hybrid systems, we explore a new domain which is analog and mixed signal circuits. Indeed, hybrid systems provide a mathematical model appropriate for the modeling and analysis of these circuits. The choice of this application domain is motivated by the need in automatic tools to facilitate the design of these circuits which, for various reasons, is still lagging behind the digital circuit design. Besides hybrid automata described using a textual language, the tool **HTG** can accept as input electrical circuits specified using SPICE netlists. We have treated a number of case studies from control applications as well as from analog and mixed signal circuits. The experimental results obtained using the tool **HTG** show its applicability to systems with complex dynamics and its scalability to high dimensional systems (with up to 200 continuous variables).

Before presenting these results, we first describe our conformance testing framework and test coverage measure.

1.2 Model

Conformance testing provides a means to assess the correctness of an implementation with respect to a specification by performing experiments on the implementation and observing its responses. When the specification is described by a formal model, the international standard "Formal Methods in Conformance Testing" (FMCT) [38] provides a framework of conformance testing, which includes abstract concepts (such as conformance, test cases, test execution, test generation), and the requirements on these concepts. A testing approach that is based on a formal model is called a *model-based testing* approach. Depending on the type of formal models, various frameworks can be developed for conformance testing. In this work, following the standard FMCT, we are interested in developing a conformance testing framework for embedded systems, using the hybrid automaton model [4]. Intuitively, a hybrid automaton is an automaton augmented with continuous variables that evolve according to some differential equations.

Definition 1 (Hybrid automaton) *A* hybrid automaton *is a tuple* $\mathcal{A} = (\mathcal{X}, Q, E, F, \mathcal{I}, \mathcal{G}, \mathcal{R})$ *where*

- $\mathcal{X}$ *is the continuous state space and is a bounded subset of* $\mathbb{R}^n$*;*
- Q *is a (finite) set of locations (or discrete states);*
- $E \subseteq Q \times Q$ *is a set of discrete transitions;*
- $F = \{F_q \mid q \in Q\}$ *such that for each* $q \in Q$*,* $F_q = (f_q, U_q)$ *defines a differential equation:*

$$\dot{x}(t) = f_q(x(t), u(t))$$

where $x \in \mathcal{X}$ is the continuous state, $u(\cdot) \in \mathcal{U}_q$ is the input of the form $u : \mathbb{R}^+ \to U_q \subset \mathbb{R}^m$. The set $\mathcal{U}_q$ is the set admissible inputs and consists of piecewise continuous functions. We assume that all f_q are Lipschitz continuous[1]. *In location q, the evolution of the continuous variables is governed by $\dot{x}(t) = f_q(x(t), u(t))$.*

- $\mathcal{I} = \{\mathcal{I}_q \subseteq \mathcal{X} \mid q \in Q\}$ *is a set of staying conditions;*
- $\mathcal{G} = \{\mathcal{G}_e \mid e \in E\}$ *is a set of guards such that for each discrete transition $e = (q, q') \in E$, $\mathcal{G}_e \subseteq \mathcal{I}_q$;*
- $\mathcal{R} = \{\mathcal{R}_e \mid e \in E\}$ *is a set of reset maps. For each $e = (q, q') \in E$, $\mathcal{R}_e : \mathcal{G}_e \to 2^{\mathcal{I}_{q'}}$ defines how x may change when $\mathcal{A}$ switches from q to q';*
- *The initial state of the automaton is denoted by (q_{init}, x_{init}).*

A *hybrid state* is a pair (q, x) where $q \in Q$ and $x \in \mathcal{X}$. The hybrid state space is $\mathcal{S} = Q \times \mathcal{X}$. In the rest of the paper, for brevity, we often use "state" to refer to a hybrid state.

A state (q, x) of $\mathcal{A}$ can change in two ways as follows: (1) by a *continuous evolution*, the continuous state x evolves according to the dynamics f_q while the location q remains constant; (2) by a *discrete evolution*, x satisfies the guard condition of an outgoing transition, the system changes the location by taking this transition and possibly changing the values of x according to the associated reset map. More formally, continuous and discrete evolutions are defined as follows.

Given a real number $h > 0$ and an admissible input function $u(\cdot) \in \mathcal{U}_q$, $(q, x) \overset{u(\cdot),h}{\to} (q, x')$ is a *continuous evolution* at the location q from the hybrid state (q, x) to (q, x'), iff $x' = \xi_{x,u(\cdot)}(h)$ and for all $t \in [0, h] : \xi_{x,u(\cdot)}(t) \in \mathcal{I}_q$, where $\xi_{x,u(\cdot)}(t)$ is the solution of the differential equation at the location q with the initial condition x and under the input $u(\cdot)$. In other words, x' is reached from x under the input $u(\cdot)$ after exactly h time, and we say that $u(\cdot)$ is *admissible* starting at (q, x) for h time.

Given a transition $e = (q, q') \in E$, $(q, x) \overset{e}{\to} (q', x')$ is a discrete evolution iff $x \in \mathcal{G}_e$ and $x' \in \mathcal{R}_e(x)$. We say that (q', x') is reachable from (q, x) and the discrete transition e

[1]The function f_q is Lipschitz continuous if there exists a constant K such that $\forall x, y : ||f_q(x) - f_q(y)|| \leq K||x - y||$, where $|| \cdot ||$ is some norm of $\mathbb{R}^n$. This condition ensures the existence and uniqueness of solutions of the differential equations.

is admissible at (q, x). Unlike *continuous evolutions*, *discrete evolutions* are instantaneous, which means that they do not take time.

It is important to note that this model allows capturing *non-determinism* in both continuous and discrete dynamics. The non-determinism in continuous dynamics is caused by the uncertainty in the input function. For example, when the input is used to model some external disturbances or modelling errors, we do not know the exact input fucntion but only its range. The non-determinism in discrete dynamics is caused by the fact that at some states it is possible for the system to stay at the current location or to switch to another one. In addition, multiple transitions can be enabled at some states. This non-determinism is useful for describing disturbances from the environment and imprecision in modelling and implementation. We assume that the hybrid automata we consider are non-Zeno[2].

Figure 1.1 shows a hybrid automaton that has 3 locations q_1, q_2, q_3. From each location q_i, there is a discrete transition (q_i, q_j) such that $j \neq i$.

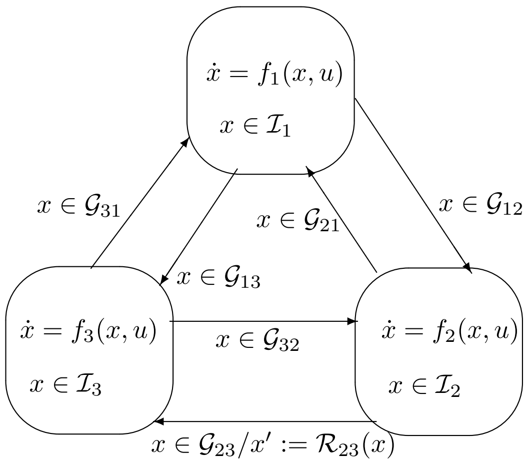


Figure 1.1: A hybrid automaton.

Figure 1.2 illustrates the evolution of this hybrid automaton. We denote by $\mathcal{G}_{ij}$ the guard of the transition (q_i,q_j). In Figure 1.2 only the guards $\mathcal{G}_{12}$ and $\mathcal{G}_{23}$ that are the shaded regions . All the locations have the same staying set which is the bounding box.

[2]A Zeno behavior can be described informally as the system making an infinite number of discrete transitions in a finite amount of time.

As we can see from the figure, starting from the initial state (q_1, y_0) the system can generate a set of infinite number of trajectories. For instance, we consider the following trajectories. Starting from (q_1, y_0), under two different continuous input functions, from the system generates two different trajectories which can reach the guards $\mathcal{G}_{12}$ and $\mathcal{G}_{13}$. The points y_1, y_2' and y_3' correspond to three different choices: (1) remaining at location q_1 and continuing with the dynamics of f_1; (2) taking the transition (q_1, q_2) at the point y_2 and evolving under the dynamics of f_2 to y_2'; (3) taking the transition (q_1, q_3) at the point y_2, jumping then to y_3 according to the reset $\mathcal{R}_{13}$ and from there evolving under the dynamics of f_3 to y_3'.

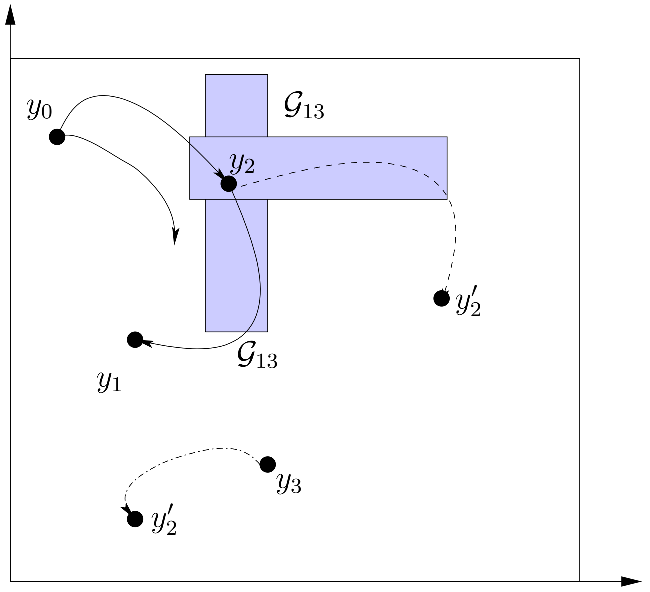


Figure 1.2: Hybrid automaton evolution.

1.3 Conformance testing

In the first part of this section, we define the notions of inputs and observations. In the second part, we define conformance relation, test cases, and test executions.

Our testing goal is to make statements about the conformance relation between the behaviors of an implementation or, more generally, a system under test (SUT) and a specification.

The specification is formal and modeled by a hybrid automaton. The conformance is defined as a relation $\preccurlyeq \subseteq \Xi \times \mathcal{H}$ where Ξ is a set of SUTs of interest, and $\mathcal{H}$ is a set of hybrid automata modeling the specifications of interest. The systems under test are physical systems, but it can be assumed that all the SUTs in Ξ can be described by a class of formal models, which is a set $\mathcal{H}_s$ of hybrid automata. It is important to note that we assume that a model for each SUT in Ξ exists[3] but do not assume that we know it. This assumption enables us to include the system under test in our formal framework and to express formally the conformance relation $\preccurlyeq$ between the models of the SUTs and the specifications, that is $\preccurlyeq \subseteq \mathcal{H}_s \times \mathcal{H}$. Note that here we use the same notation $\preccurlyeq$ for the relation between the physical SUT and the specification and the relation between the model of the SUT and the specification. A system under test $S_t \in \Xi$ is said to *conform* to a specification $\mathcal{A} \in \mathcal{H}$ if and only if the model $\mathcal{A}_s \in \mathcal{H}_s$ of the SUT is related to $\mathcal{A}$ by $\preccurlyeq$, that is, $\mathcal{A}_s \preccurlyeq \mathcal{A}$.

The system under test often operates within some environment. In our testing framework, a tester plays the role of the environment and it performs experiments on the SUT in order to study the conformance relation between the SUT and the specification. Such an experiment is called a *test*, and its specification is called a *test case*. A set of test cases is called a *test suite*, and the process of applying a test to a system under test is called a *test execution*. The tester works as follows. It emits the control inputs to the SUT and measures the observation sequences in order to produce a verdict $\nu \in \{P, F\}$ where P means "pass" (the observed behavior is allowed by the specification), F means "fail" (the observed behavior is not allowed by the specification). We continue by giving a detailed description of conformance relation. The problem of how to perform test executions and derive verdicts is discussed at the end of this section.

1.3.1 Conformance relation

Recall that the specification is modeled by a hybrid automaton $\mathcal{A}$ and the system under test SUT by another hybrid automaton $\mathcal{A}_s$. For brevity, when the context is clear, we often say

[3]Concerning embedded systems design, this assumption is not restrictive since, as mentioned in the introduction, hybrid systems can be used to model various physical systems with mixed continuous-discrete dynamics and are particularly appropriate for embedded systems.

"the system under test" to mean the automaton $\mathcal{A}_s$. To define the conformance relation, the notions of *inputs and observations* are necessary.

An input of the system which is controllable by the tester is called a *control input*; otherwise, it is called a *disturbance input*. We consider the following input actions. Control inputs are realized by actuators and observations by sensors. In practical systems, because of actuator and sensor imprecision, control inputs and observations are subject to errors. In the rest of this section, we assume that the tester can realize exact input values and observations are exactly measured. The testing problem with actuator and sensor imprecision is addressed in Section 1.11. It is worth noting that in this work we use the trace inclusion to define conformance relation. In the literature of conformance testing for discrete systems, more complex relations are considered, for example input-output conformance relation (see [39]).

Continuous input action. All the continuous inputs are assumed to be controllable by the tester. Since we want to implement the tester as a computer program, we are interested in piecewise-constant input functions; indeed, a computer cannot generate a function from reals to reals. Hence, a *continuous control action* $(\bar{u}_q, h)$, where $\bar{u}_q$ is the value of the input and h is the *duration*, specifies that the automaton continues with the continuous dynamics at the location q under the input $u(t) = \bar{u}_q$ for exactly h time. We say that $(\bar{u}_q, h)$ is *admissible at* (q, x) if the input function $u(t) = \bar{u}_q$ for all $t \in [0, h]$ is admissible starting at (q, x) for h time.

Discrete input actions. The discrete transitions are partitioned into *controllable* corresponding to discrete control actions and *uncontrollable* corresponding to discrete disturbance actions. The tester emits a discrete control action to specify whether the system should take a controllable transition (among the enabled ones) or continue with the same continuous dynamics. In the former case, it can also control the values assigned to the continuous variables by the associated reset map. Although non-determinism caused by the reset maps can be expressed in this framework (and considered in the proposed test generation algorithms), for simplicity of notation and clarity of explanation, we omit this type of non-determinism in the subsequent definitions. Hence, we denote a discrete control action by the corresponding transition, such as (q, q').

We use the following assumption about the inputs: continuous control actions are of higher priority than discrete actions. This means that after a continuous control action $(\bar{u}_q, h)$ is applied, no discrete transitions can occur during h time, that is until the end of that continuous control action. This assumption is not restrictive from a modeling point of view. Indeed, by considering all the possible values of h we can capture the cases where a discrete transition can occur before the termination of a continuous control action.

In this work, we are only interested in testing *non-blocking behaviors*, we thus need the notion of *admissible input sequences*. We write $(q, x) \xrightarrow{\iota} (q', x')$ to indicate that (q', x') is reached after applying the input action ι to the state (q, x).

Definition 2 (Admissible input sequence) *For a state (q, x), a sequence of input actions $\omega = \iota_0, \iota_1, \dots, \iota_k$ is admissible at (q, x) if*

- *ι_0 is admissible at (q, x), and*
- *for each $i = 1, \dots, k$, let (q_i, x_i) be the state such that $(q_{i-1}, x_{i-1}) \xrightarrow{\iota_{i-1}} (q_i, x_i)$, then ι_i is admissible at (q_i, x_i).*

The sequence $(q, x), (q_1, x_1), \dots, (q_k, x_k)$ is called the trace *starting at (q, x) under ω and is denoted by $\tau((q, x), \omega)$.*

We write $(q, x) \xrightarrow{\omega} (q', x')$ to indicate that (q', x') is reached from (q, x) after ω. We also say that (q', x') is forward reachable from (q, x) and (q, x) is backward reachable from (q', x'). In the rest of the paper, we simply say "reachable" to mean "forward reachable"; "backward reachable" is explicitly stated.

By the assumption about the inputs, uncontrollable discrete transitions cannot occur during a continuous control action. However, they can occur between control actions. Hence, the result of applying a control action is non-deterministic. To determine all possible traces that can be generated by applying a sequence of control actions, we must define an *admissible sequence of control actions*.

Given a state (q, x) and a control action c, let σ be a disturbance input sequence such that $c \oplus \sigma$, where $\oplus$ is the concatenation operator, is an admissible input sequence at (q, x). The sequence σ is called a disturbance input sequence *admissible after the control action c*. We denote by $\Lambda(c, (q, x))$ the set of all such disturbance input sequences.

To know whether a sequence of control actions is admissible, we must know which disturbance inputs are admissible after each control action. This means that we must know the successors after each control action. We first consider a sequence of two control actions $\omega_c = c_0\, c_1$. After accepting the control action c_0 and all the disturbance input sequences admissible after c_0, the set of all possible successors of (q, x) is:

$$\Upsilon(c_0, (q, x)) = \{(q', x') \mid \exists \sigma \in \Lambda(c_0, (q, x)) : (q, x) \overset{c_o \oplus \sigma}{\rightarrow} (q', x')\}.$$

It should be noted that if the first c_0 is admissible at (q, x) then $\Upsilon(c_0, (q, x))$ is not empty. We use the same notation Λ for the *set of all disturbance input sequences admissible after the control action sequence* $\omega_c = c_0\, c_1$:

$$\Lambda(\omega_c, (q, x)) = \bigcup_{(q',x') \in \Upsilon(c_0,(q,x))} \Lambda(c_1, (q', x')).$$

Therefore, we can now determine the set of all input sequences that can occur when we apply the control sequence $\omega_c = c_0\, c_1$. We denote this set by $\Sigma(\omega_c, (q, x))$, which can be defined as follows:

$$\begin{gathered}\Sigma(\omega_c, (q, x)) = \{c_0 \oplus \sigma_0 \oplus c_1 \oplus \sigma_1 \mid \sigma_0 \in \Lambda(c_0, (q, x)) \\ \wedge\ \exists (q', x') \in \Upsilon(c_0, (q, x)) :\ \sigma_1 \in \Lambda(c_1, (q', x'))\}.\end{gathered}$$

For a sequence ω_c of more than two control actions, the set $\Sigma(\omega_c, (q, x))$ can be defined similarly.

Definition 3 (Admissible control action sequence) *A control action sequence ω_c is admissible starting at (q, x) iff $\Sigma(\omega_c, (q, x))$ is not empty. The set of traces starting at (q, x) after an admissible control action sequence ω_c is $Tr((q, x), \omega_c) = \{\tau((q, x), \sigma) \mid \sigma \in \Sigma(\omega_c, (q, x))\}$.*

Intuitively, this means that an admissible control action sequence, when being applied to the automaton, does not cause it to be blocked. We denote by $S_{\mathcal{C}}(\mathcal{A})$ the *set of all admissible control action sequences* for the hybrid automaton $\mathcal{A}$ starting at the initial state (q_{init}, x_{init}).

Observations

We use the following assumptions about the *observability* of the hybrid automata $\mathcal{A}$ and $\mathcal{A}_s$:

- The locations of the hybrid automata $\mathcal{A}$ and $\mathcal{A}_s$ are observable.
- We assume a subset $V_o(\mathcal{A})$ and $V_o(\mathcal{A}_s)$ of observable continuous variables of $\mathcal{A}$ and $\mathcal{A}_s$ respectively. In addition, we assume that $V_o(\mathcal{A}) \subseteq V_o(\mathcal{A}_s)$, which means that an observable continuous variable of $\mathcal{A}$ is also an observable variable of $\mathcal{A}_s$.

Systems with more general partial observability are not considered in this chapter and is part of our current research. Since not all the continuous variables are observable, the following projection operator is necessary. The projection of a continuous state x of $\mathcal{A}$ on the observable variables $V_o(\mathcal{A})$ is denoted by $\pi(x, V_o(\mathcal{A}))$. The projection can be then defined for a trace as follows. The projection of a trace $\tau = (q_0, x_0), (q_1, x_1), (q_2, x_2) \ldots$ on $V_o(\mathcal{A})$ is

$$\pi(\tau, V_o(\mathcal{A})) = (q_0, \pi(x_0, V_o(\mathcal{A}))), (q_1, \pi(x_1, V_o(\mathcal{A}))), (q_2, \pi(x_2, V_o(\mathcal{A}))) \ldots .$$

A pair $(q, \pi(x, V_o(\mathcal{A}))$ where q is a location and x is the continuous state of the automation $\mathcal{A}$, is called an *observation*.

Definition 4 (Observation sequence) *Let ω be an admissible control action sequence starting at the initial state (q_{init}, x_{init}) of $\mathcal{A}$. The set of* observation sequences *associated with ω is $S_{\mathcal{O}}(\mathcal{A}, \omega) = \{\pi(\tau, V_o(\mathcal{A})) \mid \tau \in Tr((q_{init}, x_{init}), \omega)\}$.*

Before continuing, we remark that it is straightforward to extend the framework to observations described by (q, y) where q is a location, $y = g(x)$ and g is an output map.

Conformance relation

In the definition of the conformance relation between a system under test $\mathcal{A}_s$ and a specification $\mathcal{A}$, we assume that the set of all admissible control action sequences of $\mathcal{A}$ is a subset of that of $\mathcal{A}_s$, that is $S_{\mathcal{C}}(\mathcal{A}) \subseteq S_{\mathcal{C}}(\mathcal{A}_s)$. This assumption assures that the system under test can admit all the control action sequences that are admissible by the specification. Detecting the cases where the physical SUT does not admit some inputs that are allowed by the specification requires the ability to identify the states of the system from the observations. We do not consider this problem in this work.

Definition 5 (Conformance) *The system under test $\mathcal{A}_s$ is conform to the specification $\mathcal{A}$, denoted by $\mathcal{A} \preccurlyeq \mathcal{A}_s$, iff*

$$\forall \omega \in S_{\mathcal{C}}(\mathcal{A}) : \ \pi(S_{\mathcal{O}}(\mathcal{A}_s, \omega), V_o(\mathcal{A})) \subseteq S_{\mathcal{O}}(\mathcal{A}, \omega).$$

Intuitively, the system under test $\mathcal{A}_s$ is conform to the specification $\mathcal{A}$ if under every admissible control action sequence, the set of observation sequences of $\mathcal{A}_s$ is included in that of $\mathcal{A}$.

1.3.2 Test cases and test executions

In our framework, a *test case* is represented by a tree where each node is associated with an observation and each path from the root with an observation sequence. Each edge of the tree is associated with a control action. A physical *test execution* can be described as follows:

- The tester applies a test ζ to the system under test S_t.
- It measures and records a number of observations.
- The observations are measured at the end of *each* continuous control action and after *each* discrete (disturbance or control) action.

This conformance testing procedure is denoted by $exec(\zeta, S_t)$ which leads to an observation sequence, or a set of observation sequences if multiple runs of ζ are possible because of non-determinism. The above test execution process uses a number of implicit assumptions. First, observation measurements take zero time, and in addition, no measurement error is considered. Second, the tester is able to realize exactly the continuous input functions, which is often impossible in practice because of actuator imprecision, as mentioned earlier. Under these assumptions, one can only test the conformance of *a model of the system under test* to the specification in discrete time. These issues must be considered in order to address the actual testing of real systems and this is discussed in Section 1.11.

We focus on the case where each test execution involves a single run of a test case. The remaining question is how to interpret the observation sequences in order to produce a verdict. Let Ω denote the observation sequence domain. We thus define a verdict function: $\boldsymbol{v} : \Omega \rightarrow \{\mathbf{pass}, \mathbf{fail}\}$. Note that an observation sequence must cause a unique verdict. The observation sequences in Ω are grouped into two disjoint sets: the set O_p of observation sequences that cause a 'pass' verdict, the set O_f that cause a 'fail' verdict. Therefore, saying 'The system under test S_t passes the test ζ' formally means $\boldsymbol{v}(exec(\zeta, S_t)) = \mathbf{pass}$. This can then be extended to a test suite.

We now discuss some important requirements for a test suite. A test suite T_s is called *complete* if for a specification specified as a hybrid automaton $\mathcal{A}$:

$$S_t \preccurlyeq \mathcal{A} \Longleftrightarrow S_t \text{ passes } T_s \tag{1.1}$$

This means that a complete test suite can distinguish exactly between all conforming and non-conforming systems. In practice, it is generally impossible to fulfill this requirement, which often involves executing an infinite test suite. A weaker requirement is *soundness.* A test suite is sound if a system does not pass the test suite, then the system is non-conforming. We can see that this requirement is weaker than completeness, since it corresponds only to the left-to-right implication in (1.1).

After defining all the important concepts, it now remains to tackle the problem of generating test cases from a specification model. In particular, we want the test suites to satisfy the *soundness requirement*. A hybrid automaton may have an infinite number of infinite traces; however, the tester can only perform a finite number of test cases in finite time. Therefore, we must select a finite portion of the input space of the specification $\mathcal{A}$ and test the conformance of the system under test $\mathcal{A}_s$ with respect to this portion. The selection is done using a coverage criterion that we formally define in the next chapter. Hence, our testing problem is formulated as to automatically generate a set of test cases from the specification automaton to satisfy this coverage criterion.

1.4 Test coverage

Test coverage is a way to evaluate testing quality. More precisely, it is a way to relate the number of tests to carry out with the fraction of the system's behaviors effectively explored.

As mentioned in the introduction, a major problem with extending the "classic" testing approach to hybrid system is the infiniteness of the input signal space and of the state space. Indeed, in practice it is only possible to test the system with a finite number of input functions, for a bounded time horizon and, furthermore, the results are only in form of a finite number of finite sequences of points on trajectories of the system. In other words, a continuous/hybrid tester cannot produce in practice the output signals that are functions from reals to reals but only their approximation in discrete time. Given an analysis objective, such as to verify a safety property, the question that arises is thus how to choose appropriate input signals so as to fulfill the analysis objective as best as possible.

Since it is impossible to enumerate all the admissible external inputs to the hybrid system in question, much effort has been invested in defining and implementing notions of *coverage* that guarantee, to some extent, that the finite set of input stimuli against which the system is tested is sufficient for validating correctness. Test coverage criteria are indeed a way to evaluate the testing quality, or the degree of fulfilling the desired analysis objective. More precisely, it is a way to relate the number of simulations to carry out with the fraction of the system's behaviors effectively explored.

For discrete systems, specified using programming languages or hardware design languages, some syntactic coverage measures can be defined, such as statement coverage and if-then-else branch coverage, path coverage, *etc.* (see for example [20, 39]). In this work, we treat continuous and hybrid systems that operate in a metric space (typically $\mathbb{R}^n$) and where not much inspiration for the coverage issues derives from the syntax. On the other hand, the metric nature of the state space encourages more *semantic* notions of coverage, namely that all system trajectories generated by the input test patterns form a kind of dense network in the reachable state space without too many big unexplored 'holes'.

Two main challenges in defining a test coverage measure are the following. First, it should be meaningful to reflect testing quality with respect to a given analysis objective. Second, one must be able to compute this measure. The above mentioned classic coverage notions mainly used in software testing are not appropriate for the trajectories of continuous and hybrid systems defined by differential equations. However, geometric properties of the hybrid state space can be exploited to define a coverage measure which, on the one hand, has a close relationship with the properties to verify and, on the other hand, can be efficiently computed or estimated. In this work, we are interested in *state coverage* and focus on a measure that describes how 'well' the visited states represent the reachable set of the system. This measure is defined using the *star discrepancy* notion in statistics, which characterizes the uniformity of the distribution of a point set within a region. Note that the reachable sets of hybrid systems are often non-convex with complex geometric form, therefore considering only corner cases does not always cover the behaviors that are important for reachabilily properties, especially in high dimensions. Hence, for a fixed number of visited states (which reflects the computation cost to produce a test suite), we want the visited states to be equidistributed over the reachable set as much as possible, since this provides a good representation of all possible reachable states.

1.4.1 Star discrepancy

We first briefly recall the star discrepancy. The star discrepancy is an important notion in equidistribution theory as well as in quasi-Monte Carlo techniques (see for example [7]).

Recently, it was also used in probabilistic motion planning to enhance the sampling uniformity [29].

Let P be a set of k points inside $\mathcal{B} = [l_1, L_1] \times \ldots \times [l_n, L_n] \subset \mathbb{R}^n$. Let $\mathcal{J}$ be the set of all sub-boxes J of the form $J = \prod_{i=1}^{n}[l_i, \beta_i]$ with $\beta_i \in [l_i, L_i]$ (see Figure 1.3). The local discrepancy of the point set P with respect to the sub-box J is defined as follows:

$$D(P, J) = \left|\frac{A(P, J)}{k} - \frac{vol(J)}{vol(\mathcal{B})}\right|$$

where $A(P, J)$ is the number of points of P that are inside J, and $vol(J)$ is the volume of the box J.

Definition 6 (Star discrepancy) *The star discrepancy of a point set P with respect to the box $\mathcal{B}$ is defined as:*

$$D^*(P, \mathcal{B}) = sup_{J \in \mathcal{J}} D(P, J). \tag{1.2}$$

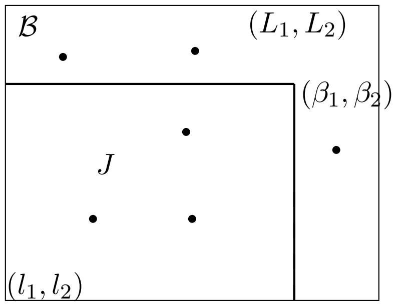


Figure 1.3: Illustration of the star discrepancy notion.

It is not hard to prove the following property of the star discrepancy [37].

Proposition 1 *The star discrepancy of a point set P with respect to a box $\mathcal{B}$ satisfies* $0 < D^*(P, \mathcal{B}) \leq 1$.

Intuitively, the star discrepancy is a measure for the irregularity of a set of points. A large value $D^*(P, \mathcal{B})$ means that the points in P are not much equidistributed over $\mathcal{B}$. When the region is a box, the star discrepancy measures how badly the point set estimates the volume of the box.

Example. To show an intuitive meaning of the star discrepancy, we use some sequences of 100 points inside a 2-dimensional unit box. The first example is the Faure sequence [16], a well-known low-discrepancy sequence (see Figure 1.4). As we can observe from the figure, this set of points 'covers well' the box, in the sense that the points are well-equidistributed over the box. Its star discrepancy value is 0.048. The second example is the Halton sequence [41]

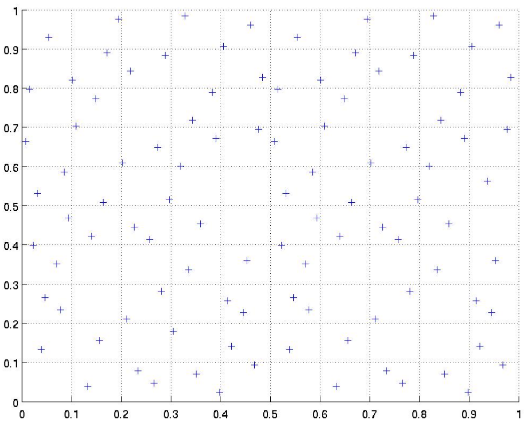


Figure 1.4: Faure sequence of 100 points. Its star discrepancy value is 0.048.

shown in Figure 1.5, which is also a well-known low discrepancy sequence. The value of the star discrepancy of the Halton sequence is about 0.050, indicating that the Faure sequence is more equidistributed than the Halton sequence. The star discrepancy values of these two sequences are however close, and indeed visually it is difficult to see from the figures which one is better equidistributed. We now give another example which is a sequence of 100 points generated by a pseudo-random function provided by the C library system. This sequence is shown in Figure 1.6, from which we can observe that this sequence is not well-equidistributed over the box. This is confirmed by its star discrepancy value 0.1. The star discrepancy is thus a meaningful measure that can characterize the uniformity quality of a point set distribution.

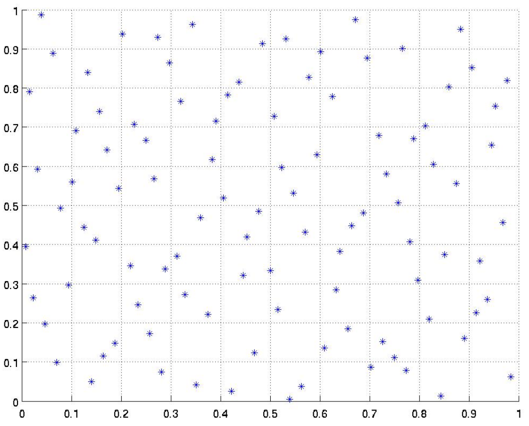


Figure 1.5: Halton sequence of 100 points. The star discrepancy value is 0.05.

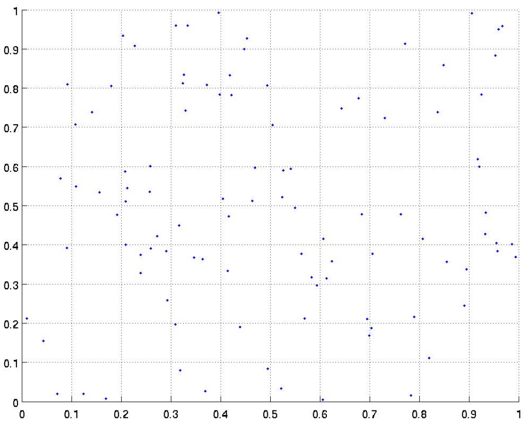


Figure 1.6: A sequence of 100 points generated by a pseudo-random function in the C library. Its star discrepancy value is 0.1.

1.4.2 Coverage estimation

To evaluate the coverage of a set of states, we must compute the star discrepancy of a point set, which is not an easy problem (see for example [14]). Many theoretical results for one-dimensional point sets are not generalizable to higher dimensions, and among the fastest algorithms, the one proposed in [14] has time complexity $\mathcal{O}(k^{1+n/2})$ where n is the dimension. In this work, we do not try to compute the star discrepancy but approximate it by estimating a lower and upper bound. These bounds as well as the information obtained from their estimation are then used to decide which parts of the state space have been 'well explored' and which parts must be explored more. This estimation is carried out using a method published in [37]. Let us briefly describe this method for computing the star discrepancy $D^*(P, \mathcal{B})$ of a point set P w.r.t. a box $\mathcal{B}$. Although in [37] the box $\mathcal{B}$ is $[0,1]^n$, we extended it to the case where $\mathcal{B}$ can be any full-dimensional box. Intuitively, the main idea of this estimation method is to consider a finite box partition of the box $\mathcal{B}$, instead of considering an infinite number of all sub-boxes as in the definition of the star discrepancy. Let $\mathcal{B} = [l_1, L_1] \times \ldots \times [l_n, L_n]$. In what follows, we often call this box $\mathcal{B}$ the *bounding box.* We define a box partition of $\mathcal{B}$ as a set of boxes $\Pi = \{\boldsymbol{b}^1, \ldots, \boldsymbol{b}^m\}$ such that $\cup_{i=1}^m \boldsymbol{b}^i = \mathcal{B}$ and the interiors of the boxes $\boldsymbol{b}^i$ do not intersect. Each such box is called an *elementary box.* Given a box $\boldsymbol{b} = [\alpha_1, \beta_1] \times \ldots \times [\alpha_n, \beta_n] \in \Pi$, we define $\boldsymbol{b}^+ = [l_1, \beta_1] \times \ldots \times [l_n, \beta_n]$ and $\boldsymbol{b}^- = [l_1, \alpha_1] \times \ldots \times [l_n, \alpha_n]$ (see Figure 1.7 for an illustration).

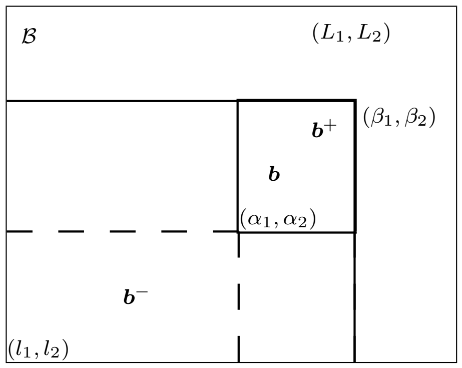


Figure 1.7: Illustration of the boxes $\boldsymbol{b}^-$ and $\boldsymbol{b}^+$.

For any finite box partition Π of $\mathcal{B}$, the star discrepancy $D^*(P, \mathcal{B})$ of the point set P with respect to $\mathcal{B}$ satisfies: $C(P, \Pi) \leq D^*(P, \mathcal{B}) \leq B(P, \Pi)$ where the upper and lower bounds are:

$$\begin{aligned} B(P,\Pi) &= \max_{\boldsymbol{b}\in\Pi} \max\{\frac{A(P,\boldsymbol{b}^+)}{k} - \frac{vol(\boldsymbol{b}^-)}{vol(\mathcal{B})}, \frac{vol(\boldsymbol{b}^+)}{vol(\mathcal{B})} - \frac{A(P,\boldsymbol{b}^-)}{k}\} \quad (1.3) \\ C(P,\Pi) &= \max_{\boldsymbol{b}\in\Pi} \max\{|\frac{A(P,\boldsymbol{b}^-)}{k} - \frac{vol(\boldsymbol{b}^-)}{vol(\mathcal{B})}|, |\frac{A(P,\boldsymbol{b}^+)}{k} - \frac{vol(\boldsymbol{b}^+)}{vol(\mathcal{B})}|\} \quad (1.4) \end{aligned}$$

The imprecision of this approximation is the difference between the upper and lower bounds, which can be bounded by $B(P,\Pi) - C(P,\Pi) \leq W(\Pi)$ where

$$W(\Pi) = \max_{\boldsymbol{b}\in\Pi}(vol(\boldsymbol{b}^+) - vol(\boldsymbol{b}^-))/vol(\mathcal{B}) \quad (1.5)$$

Thus, one must find a partition Π such that this difference is small.

1.4.3 Hybrid systems test coverage

Since a hybrid system can only evolve within the staying sets of the locations, we are interested in the coverage with respect to these sets. We assume that all the staying sets are boxes, since the star discrepancy is defined for points inside a box. If a staying set $\mathcal{I}_q$ is not a box, we can take the smallest oriented (or non-axis-aligned) box that encloses it and apply the star discrepancy definition in (1.2) to that box after an appropriate coordinate transformation. Another possible solution, which is part of our current work and reported in this chapter, is to consider the discrepancy notion for sets with more general geometric forms [7].

Definition 7 (Test coverage) *Let $\mathcal{P} = \{(q, P_q) \mid q \in Q \ \wedge \ P_q \subset \mathcal{I}_q\}$ be the set of states. The coverage of $\mathcal{P}$ is defined as:*

$$Cov(\mathcal{P}) = \frac{1}{||Q||}\sum_{q\in Q} 1 - D^*(P_q, \mathcal{I}_q)$$

where $||Q||$ is the number of locations in Q.

We can see that a large value of $Cov(\mathcal{P})$ indicates a good space-covering quality. If $\mathcal{P}$ is the set of states visited by a test suite, our objective is to maximize $Cov(\mathcal{P})$.

1.5 Test generation

Our test generation is based on a randomized exploration of the reachable state space of the system. It is inspired by the Rapidly-exploring Random Tree (RRT) algorithm, which is a successful motion planning technique for finding feasible trajectories of robots in an environment with obstacles (see [27] for a survey). More precisely, we extend the RRT algorithm to hybrid systems. Furthermore, we combine it with a guiding tool in order to achieve a good coverage of the system's behaviors we want to test. To this end, we use the coverage measure defined in the previous section. Before continuing, we mention that related work on test generation for hybrid models is described in Section 1.14.

In this section, we describe the extension of the RRT algorithm to hybrid system, which we call the hRRT algorithm. The combination of the hRRT algorithm with the guiding tool is explained in the next section.

The algorithm stores the visited states in a tree, the root of which corresponds to the initial state. The construction of the tree is summarized in Algorithm 1.

Algorithm 1 Test generation algorithm hRRT

```
$k = 1$
$\mathcal{T}^k.init(s_{init})$                                   ▷ $s_{init}$: initial state
repeat
    $s_{goal} = \textsc{Sampling}(\mathcal{S})$                    ▷ $\mathcal{S}$: hybrid state space
    $s^k_{near} = \textsc{Neighbor}(\mathcal{T}^k, s^k_{goal})$
    $(s^k_{new}, u^k_{q_{near}}) = \textsc{ContinuousSucc}(s^k_{near}, h)$       ▷ $h$: time step
    $\textsc{DiscreteSucc}(\mathcal{T}^k, s^k_{new})$
    $k{+}{+}$
until $k \geq k_{max}$
```

The tree constructed at the k^{th} iteration is denoted by $\mathcal{T}^k$. The function SAMPLING samples a hybrid state $s^k_{goal} = (q^k_{goal}, x^k_{goal})$ to indicate the direction towards which the tree is expected to evolve. Then, a starting state $s^k_{near} = (q^k_{near}, x^k_{near})$ is determined as a neighbor of s^k_{goal} using a hybrid distance which is described later. Expanding the tree from s^k_{near} towards s^k_{goal} is done as follows:

- The function ContinuousSucc tries to find the input $u^k_{q_{near}}$ such that, after one time step h, the current continuous dynamics at q^k_{near} takes the system from s^k_{near} towards s_{goal}, and this results in a new continuous state x^k_{new}. A new edge from s_{near} to $s^k_{new} = (q^k_{near}, x^k_{new})$, labeled with the associated input $u^k_{q_{near}}$, is then added to the tree. To find s^k_{new}, when the set U is not finite it can be sampled, or one can solve a local optimal control problem.

- Then, from s^k_{new}, the function DiscreteSucc computes its successors by all possible discrete transitions and add them in the tree. A discrete successor by a transition is computed by testing whether s^k_{new} satisfies its guard and if so applying the associated reset function to s^k_{new}.

The algorithm terminates after some maximal number of iterations. Another possible termination criterion is that a satisfactory coverage value is reached. In the classic RRT algorithms, which work in a continuous setting, only x_{goal} must be sampled, and a commonly used sampling distribution of x_{goal} is uniform over $\mathcal{X}$. In addition, the point x_{near} is defined as a nearest neighbor of x_{goal} in some usual distance, such as the Euclidian distance. In our hRRT algorithm, the goal state sampling is not uniform and the function Sampling plays the role of guiding the exploration via a biased sampling of x_{goal}.

Hybrid distance and computation of neighbors

Defining a metric for the hybrid state space is difficult because of the discrete component of a hybrid state. We propose an approximate distance between two hybrid states, which is used in the function Neighbor of the hRRT algorithm.

Given two hybrid states $s = (q, x)$ and $s' = (q', x')$, if they have the same discrete component, that is, $q = q'$, we can use some usual metric in $\mathbb{R}^n$, such as the Euclidian metric. When $q \neq q'$, it is natural to use the average length of the trajectories from one to another. Figure 1.8 illustrates this definition. We consider a trajectory corresponding to a sequence of transitions $\gamma = e_1, e_2$ where $e_1 = (q, q_1)$ and $e_2 = (q_1, q')$. We call γ a discrete path from $s = (q, x)$ to $s' = (q', x')$.

- The average length of the discrete path γ is simply the distance between the image of the first guard $\mathcal{G}_{(q,q_1)}$ by the first reset function $\mathcal{R}_{(q,q_1)}$ and the second guard $\mathcal{G}_{(q_1,q')}$. This distance is shown in the middle figure. The average distance between two sets is defined as the Euclidean distance between their geometric centroids.
- The average length of trajectories from $s = (q, x)$ to $s = (q', x')$ following the discrete path γ is the sum of three distances (shown in Figure 1.8 from left to right): the distance between x and the first guard $\mathcal{G}_{(q,q_1)}$, the average length of the path, and the distance between $\mathcal{R}_{(q_1,q')}(\mathcal{G}_{(q_1,q')})$ and x'.

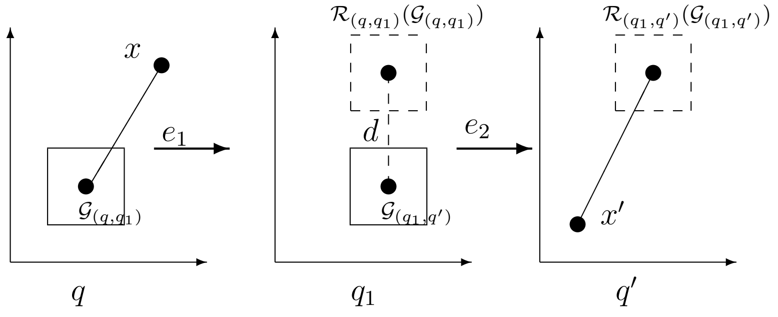

x
e_1
$\mathcal{G}_{(q,q_1)}$
q
$\mathcal{R}_{(q,q_1)}(\mathcal{G}_{(q,q_1)})$
d
e_2
$\mathcal{G}_{(q_1,q')}$
q_1
$\mathcal{R}_{(q_1,q')}(\mathcal{G}_{(q_1,q')})$
x'
q'

Figure 1.8: Illustration of the average length of a trajectory.

Now we are ready to define the *hybrid distance* from s to s'. We denote by $\Gamma(q, q')$ the set of all discrete paths from q to q' in the hybrid automaton $\mathcal{A}$.

Definition 8 (Hybrid distance) *Given two hybrid states $s = (q, x)$ and $s' = (q', x')$, the* hybrid distance *from s to s', denoted by $d_H(s, s')$, is defined as follows :*

- *If $q = q'$, then $d_H(s, s') = ||x - x'||$ where $|| \cdot ||$ is the Euclidean distance in $\mathbb{R}^n$.*
- *If $q \neq q'$, there are two cases:*
  - *If $\Gamma(q, q') \neq \emptyset$, then $d_H(s, s') = \min_{\gamma \in \Gamma(q,q')} len_\gamma(s, s')$. The path γ that minimizes $len_\gamma(s, s')$ is called the* shortest path *from s to s'.*

– *Otherwise,* $d_H(s, s') = \infty$.

It is easy to see that the hybrid distance d_H is only a pseudo metric since it does not satisfy the symmetry requirement. Indeed, the underlying discrete structure of a hybrid automaton is a directed graph. In the above definition, instead of the Euclidian distance we can use any metric in $\mathbb{R}^n$.

Then, in each iteration of hRRT, the function NEIGHBOR can be computed as follows. A neighbor of the goal state s_{goal} is:

$$s_{near} \in arg\, min_{s \in V} d_H(s, s_{goal})$$

where V is the set of all the states stored at the vertices of the tree.

Computing continuous and discrete successors

We first describe the function CONTINUOUSSUCC. If the states s_{near} and s_{goal} have the same location component, we want to expand the tree from x_{near} towards x_{goal} as closely as possible, using the continuous dynamics at that location.

When the states s_{near} and s_{goal} are at different locations, let γ be the shortest path from s_{near} to s_{goal}. It is natural to make the system follow this path. Therefore, we want to steer the system from x_{near} towards the first guard of γ. In both of the two cases, one must solve an optimal control problem with the objective of minimizing the distance to some target point. This problem is difficult especially for systems with non-linear continuous dynamics. Thus, we can trade some optimality for computational efficiency. When the input set U is not finite, we sample a finite number of inputs and pick from this set a best input, that makes the system approach the boundary of the guard of γ as much as possible. In addition, we can prove that by appropriately sampling the input set, the completeness property of our algorithm is preserved [13]). It is important to emphasize that the function CONTINUOUSSUCC must assure that the trajectory segment from x_{near} remains in the staying set of the current location.

The computation of discrete successors in DiscreteSucc, which involves testing a guard condition and applying a reset map, is rather straightforward.

Test cases and verdicts

The tree constructed by the hRRT algorithm can be used to extract a test suite. In addition, when applying such test cases to the system under test, the tree can be used to compare the observations from the physical systems and the expected observations in the tree, which allows a decision whether the system satisfies the conformance relation.

1.6 Coverage-guided test generation

In this section we propose a tool for guiding the test generation algorithm. This tool is based on the coverage measure defined using the star discrepancy. The goal of the guiding tool is to use the sampling process to bias the evolution of the tree towards the interesting region of the state space, in order to rapidly achieve a good coverage quality. In each iteration, we use the information of the current coverage to improve it. Indeed, the coverage estimation provides not only an approximate value of the current coverage, but also the information about which regions need to be explored more.

Sampling a goal state $s_{goal} = (q_{goal}, x_{goal})$ in the hybrid state space $\mathcal{S}$ consists of two steps:

1. Sample a goal location q_{goal} from the set Q of all the locations, according to some probability distribution.
2. Sample a continuous goal state x_{goal} inside the staying set $\mathcal{I}_{q_{goal}}$ of the location q_{goal}.

Location sampling

Recall that we want to achieve a good testing coverage quality, which is equivalent to a small value of the star discrepancy of the points visited at each location. More concretely, in each iteration, we want to bias the goal state sampling distribution according to the current coverage of the visited states. To do so, we first sample a location and then a continuous state. Let $\mathcal{P} = \{(q, P_q) \mid q \in Q \wedge P_q \subset \mathcal{I}_q\}$ be the current set of visited states. The location sampling distribution depends on the current continuous state coverage of each location:

$$Pr[q_{goal} = q] = \frac{D^*(P_q, \mathcal{I}_q)}{\sum_{q' \in Q} D^*(P_{q'}, \mathcal{I}_{q'})}$$

where the notation Pr is used for probabilities. Intuitively, we assign a higher sampling probability to the locations with lower continuous state coverage. As we have shown earlier, the star discrepancy is approximated by a lower bound and an upper bound. We thus compute the above probability $Pr[q_{goal} = q]$ using these bounds and then taking the mean of the results.

Continuous state sampling

We now show how to sample x_{goal}, assuming that we have already sampled a location $q_{goal} = q$. In the remainder of the paper, to give geometric intuitions, we often call a continuous state a point. In addition, since all the staying sets are assumed to be boxes, we denote the staying set $\mathcal{I}_q$ by the box $\mathcal{B}$ and denote the current set of visited points at the location q simply by P instead of P_q. Let k be the number of points in P. Let Π be a finite box partition of $\mathcal{B}$ that is used to estimate the star discrepancy of P. The sampling process consists of two steps. In the first step, we sample an elementary box $\boldsymbol{b}_{goal}$ from the set Π; in the second step we sample a point x_{goal} in $\boldsymbol{b}_{goal}$ uniformly.

The elementary box sampling distribution in the first step is biased in order to optimize the coverage. Guiding is thus done via the goal box sampling process. Let Π be the box partition used in the coverage estimation, and we denote by P the current set of visited states. The objective is to define a probability distribution over the set of elementary boxes of Π. This probability distribution is defined at each iteration of the test generation algorithm. Essentially, we favor the selection of a box if adding a new state in this box allows to improve the coverage of the visited states. This is captured by a potential influence function, which assigns to each elementary box $\boldsymbol{b}$ in the partition a real number that reflects the change in the coverage if a new state is added in $\boldsymbol{b}$. The current coverage is given in form of a lower and an upper bound. In order to improve the coverage, we aim at reducing both of the bounds. More details on the method can be found in [13].

Let us summarize the developments so far. We have shown how to sample a goal hybrid state. This sampling method is not uniform but biased in order to achieve a good coverage of the visited states. From now on, the algorithm hRRT in which the function Sampling uses this coverage-guided method is called the gRRT algorithm, which means 'guided hRRT'.

We can prove that the gRRT algorithm preserves the *probabilistic completeness* of RRT [13]. Roughly speaking, the probabilistic completeness property [25] states that if the trace we search for is feasible, then the probability that the algorithm finds it approaches 1 as the number k of iterations approaches infinity.

To demonstrate the performance of gRRT, we use two illustrative examples. For brevity, we call the classical RRT algorithm using uniform sampling and the Euclidian metric hRRT. The reason we choose these examples is that they differ in the reachability property. In the first example, the system is "controllable" in the sense that the entire state space is reachable from the initial states (by using appropriate inputs), but in the second example the reachable set is only a small part of the state space. These examples will also be used to validate the efficiency of the new guiding method that we propose.

Example 1 *This is a two-dimensional continuous system where the state space $\mathcal{X}$ is a box $\mathcal{B} = [-3, -3] \times [3, 3]$. The continuous dynamics is $f(x, t) = u(t)$ where the input set is $U = \{u \in \mathbb{R}^2 \mid ||u|| \leq 0.2\}$.*

We use 100 input values resulting from a discretization of the set U. The initial state is $(-2.9, -2.9)$. The time step is 0.002. Figure 1.9 shows the result obtained using gRRT, and Figure 1.10 shows the evolution after each iteration of the coverage of the states generated by gRRT (solid curve) and by hRRT (dashed curve), which indicates that gRRT achieved a better coverage quality especially in convergence rate.

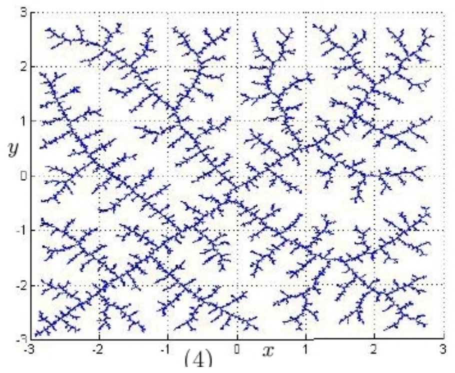


Figure 1.9: The gRRT exploration result for Example 1.

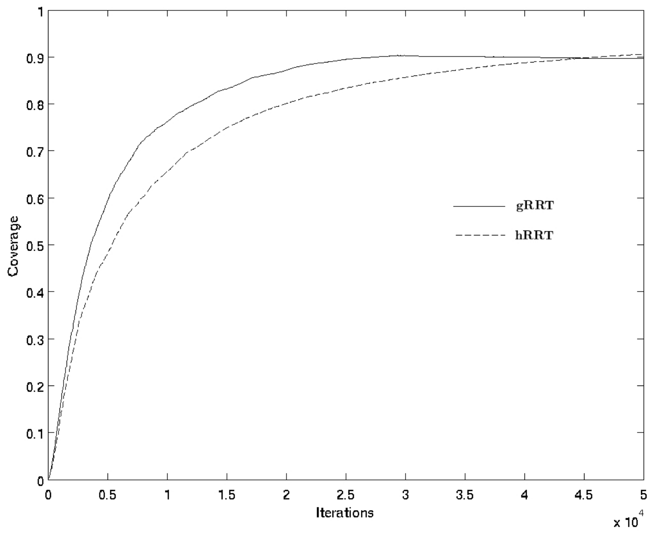


Figure 1.10: The test coverage evolution using hRRT and gRRT.

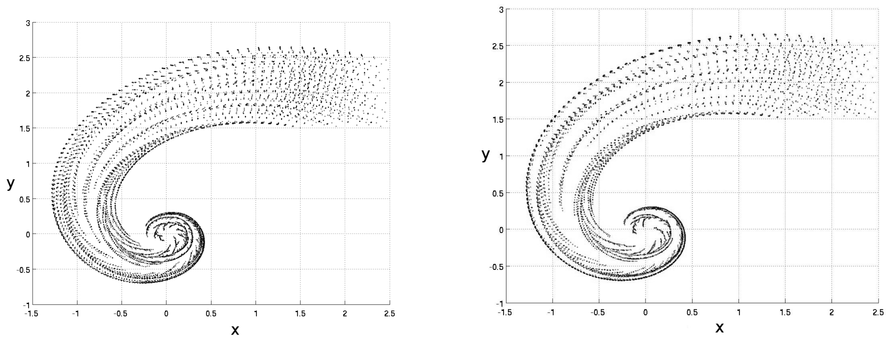


Figure 1.11: Results obtained using the guided sampling method (left) and using the uniform sampling method (right).

Example 2 *This example is a linear system with a stable focus at the origin. Its dynamics is*

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} -1 & -1.9 \\ 1.9 & -1 \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$$

We let the dynamics be slightly perturbed by an additive input u. The state space is the box $\mathcal{B} = [-3,-3] \times [3,3]$. The input set $U = \{u \in \mathbb{R}^2 \mid ||u|| \leq 0.2\}$. Figure 1.11 shows the results obtained after 50000 *iterations. We can see that again the guided sampling method achieved a better coverage result for the same number of points since the points visited by the guided sampling method are more equidistributed over the state space.*

1.7 Controllability issue

From different experiments with Example 2, we observe that the coverage performance of gRRT is not satisfying when the reachable space is only a small part of the entire state space. To illustrate this, we increase the state space from $\mathcal{B} = [-3,-3] \times [3,3]$ to $\mathcal{B}' = [-5,-5] \times [5,5]$. For the larger state space, the coverage quality is poorer (see Figure 1.12). This can be explained as follows. There are boxes, such as those near the bottom right vertex of the bounding box, which have a high potential of reducing the bounds of the star discrepancy. Thus, the sampler frequently selects these boxes. However, these boxes are not reachable from the initial states, and all attempts to reach them do not expand the tree beyond the boundary of the reachable set. This results in a large number of points

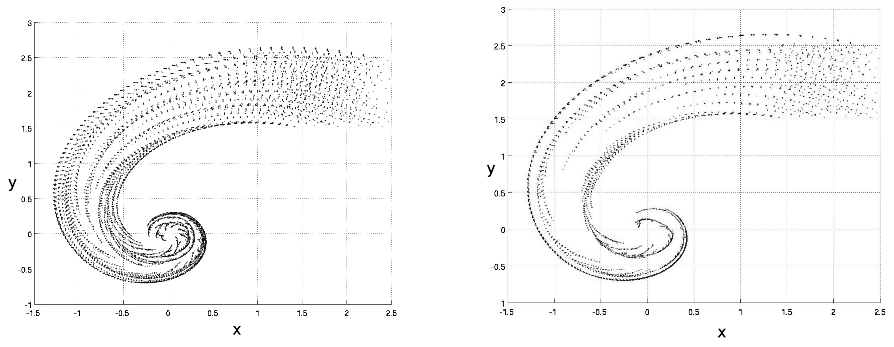


Figure 1.12: Results for the state spaces $\mathcal{B}$ (left) and $\mathcal{B}'$ (right).

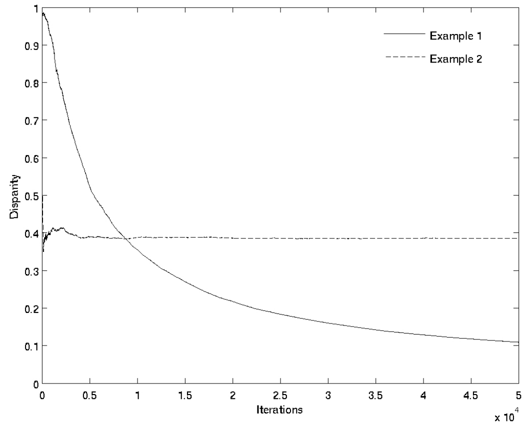
concentrated near this part of the boundary, while other parts of the reachable set are not well explored.

It is important to emphasize that this problem is not specific to gRRT. The RRT algorithm using the uniform sampling method and, more generally, any algorithm that does not take into account the differential constraints of the system, may suffer from this phenomenon. This phenomenon can however be captured by the evolution of the disparity between the set of goal states and the set of visited states. This notion is formally defined in the next section. Roughly speaking, it describes how different their distributions are. When the disparity does not decrease after a certain number of iterations, this often indicates that the system cannot approach the goal states, and it is better not to favor an expansion towards the exterior but a *refinement*, that is an exploration in the interior of the already visited regions.

Figure 1.13 shows the evolution of the disparity between the set P^k of visited states at the k^{th} iteration and the set G^k of goal states for the two examples. We observe that for the system of Example 1 which can reach any state in the state space (by choosing appropriate admissible inputs), the visited states follow the goal states, and thus the disparity gets stabilized over time. However, in Example 2, where the system cannot reach everywhere, the disparity does not decrease for a long period of time, during which most of the goal states indicate a direction towards which the tree cannot be expanded further.

Figure 1.13 shows the Voronoi diagram[4] of a set of visited states. In this example, the

[4]The Voronoi diagram of a set V of points in $\mathbb{R}^n$ is the partition of $\mathbb{R}^n$ into k polyhedral regions. Each region corresponds



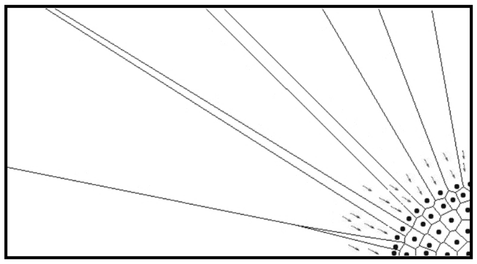


Figure 1.13: Left: The evolution of the disparity between the set P^k of visited states and the set G^k of goal states. Right: Illustration of the controllability issue.

boundary of the reachable set can be seen as an "obstacle" that prevents the system from crossing it. Note that the Voronoi cells of the states on the boundary are large (because they are near the large unvisited part of the state space). Hence, if the goal states are uniformly sampled over the whole state space, these large Voronoi cells have higher probabilities of containing the goal states, and thus the exploration is "stuck" near the boundary, while the interior of the reachable set is not well explored.

To tackle this problem, we introduce the notion of disparity to describe the 'difference' in the distributions of two sets of points. The controllability problem can be detected by a large value of the disparity between the goal states and the visited states. We can thus combine gRRT with a disparity based sampling method, in order to better adapt to the dynamics of the system. This is the topic of the next section.

1.8 Disparity

The notion of disparity between two point sets that we develop here is inspired by the star discrepancy. Indeed, by definition, the star discrepancy of a set P w.r.t. the box $\mathcal{B}$ can be

to a point $v \in V$, called the Voronoi cell of v, is defined as the set of points in $\mathbb{R}^n$ which are closer to v than to any other points in V.

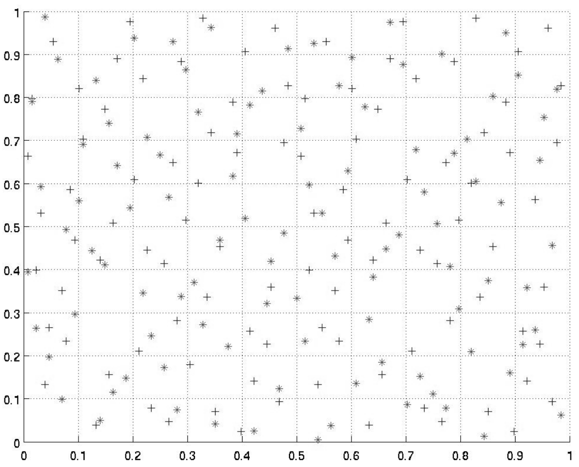


Figure 1.14: The disparity between the Faure sequence (drawn using the + signs) and the Halton sequence (drawn using the $*$ signs) is 0.06.

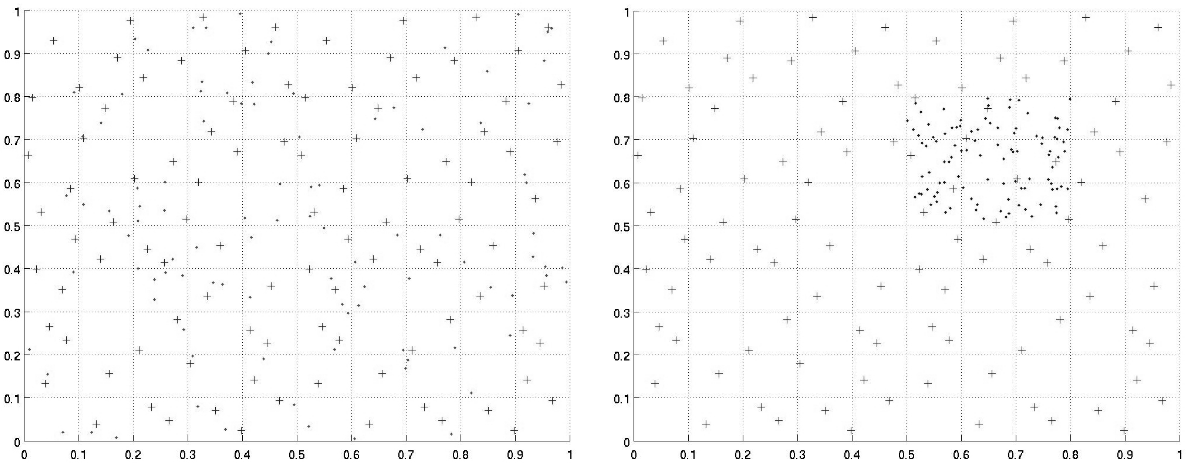


Figure 1.15: Left: The disparity between the Faure sequence (+ signs) and a C pseudo-random sequence ($*$ signs) is 0.12. Right: The disparity between the Faure sequence (+ signs) and another C pseudo-random sequence ($*$ signs) is 0.54.

seen as a comparison between P and an 'ideal' infinite set of points distributed all over $\mathcal{B}$.

Let P and Q be two sets of points inside $\mathcal{B}$. Let J be a sub-box of $\mathcal{B}$ which has the same bottom-left vertex as $\mathcal{B}$ and the top-right vertex of which is a point inside $\mathcal{B}$. Let Γ be the set of all such sub-boxes. We define the local disparity between P and Q with respect to the sub-box J as: $\gamma(P, Q, J) = |\frac{A(P,J)}{||P||} - \frac{A(Q,J)}{||Q||}|$ where $A(P, J)$ is the number of points of P inside J and $||P||$ is the total number of points of P.

Definition 9 (Disparity) *The disparity between P and Q with respect to the bounding box $\mathcal{B}$ is defined as: $\gamma^*(P, Q, \mathcal{B}) = sup_{J\in\Gamma}\gamma(P, Q, J)$.*

The disparity satisfies $0 < \gamma^*(P, Q, \mathcal{B}) \leq 1$. This property is a direct consequence of the above definition. A small value $\gamma^*(P, Q, \mathcal{B})$ means that the distributions of the sets P and Q over the box $\mathcal{B}$ are 'similar'.

To illustrate our notion of disparity, we consider two well-known sequences of points: the Faure sequence [16] and the Halton sequence [41], which are shown in Figure 1.14. Their disparity is 0.06, indicating that they have similar distributions. We then compare the Faure sequence with a sequence generated by the C library. Figure 1.15 displays these two sequences, each of which has 100 points. The star discrepancy coverage of the Faure sequence is much better than that of the C sequence, and in fact the disparity between them (which is 0.12) is twice larger than that between the Faure and Halton sequences. The last example is used to compare the Faure sequence and a set of 100 points concentratred in some small rectangle inside the bounding box. Their disparity is 0.54.

Disparity estimation

The exact computation of the disparity is as difficult as the exact computation of the star discrepancy, which is because of the infinite number of the sub-boxes. We propose a method for estimating a lower and an upper bound for this new measure. Let Π be a box partition of $\mathcal{B}$. Let P, Q be two sets of points inside $\mathcal{B}$. For each elementary box $\boldsymbol{b} \in \Pi$ we denote

$\mu_m(\boldsymbol{b}) = \max\{\mu_c(\boldsymbol{b}), \mu_o(\boldsymbol{b})\}$ where $\mu_c(\boldsymbol{b}) = \dfrac{A(P, \boldsymbol{b}^+)}{||P||} - \dfrac{A(Q, \boldsymbol{b}^-)}{||Q||}$, $\mu_o(\boldsymbol{b}) = \dfrac{A(Q, \boldsymbol{b}^+)}{||Q||} - \dfrac{A(P, \boldsymbol{b}^-)}{||P||}$. We also denote

$$c(\boldsymbol{b}) = \max\{|\frac{A(P, \boldsymbol{b}^-)}{||P||} - \frac{A(Q, \boldsymbol{b}^-)}{||Q||}|, |\frac{A(P, \boldsymbol{b}^+)}{||P||} - \frac{A(Q, \boldsymbol{b}^+)}{||Q||}|\}.$$

Theorem 1 (Upper and lower bounds) *An upper bound $B_d(P, Q, \Pi)$ and a lower bound $C_d(P, Q, \Pi)$ of the disparity between P and Q are: $B_d(P, Q, \Pi) = \max_{\boldsymbol{b}\in\Pi}\{\mu_m(\boldsymbol{b})\}$ and $C_d(P, Q, \Pi) = \max_{\boldsymbol{b}\in\Pi}\{c(\boldsymbol{b})\}$.*

The proof of the theorem can be found in [12].

Estimation error

We now give a bound on the estimation error. For each elementary box $\boldsymbol{b} = [\alpha_1, \beta_1] \times \ldots \times [\alpha_n, \beta_n] \in \Pi$, we define the $\mathcal{W}$-zone, denoted by $\mathcal{W}(\boldsymbol{b})$, as follows: $\mathcal{W}(\boldsymbol{b}) = \boldsymbol{b}^+ \setminus \boldsymbol{b}^-$. We recall that $\boldsymbol{b}^+ = [l_1, \beta_1] \times \ldots \times [l_n, \beta_n]$ and $\boldsymbol{b}^- = [l_1, \alpha_1] \times \ldots \times [l_n, \alpha_n]$.

We can prove the following bound on the error of the estimation, defined as the difference between the upper and lower bounds [12].

Theorem 2 (Error bounds) *Let $B_d(P, Q, \Pi)$ and $C_d(P, Q, \Pi)$ be the upper and lower bounds of the disparity between P and Q. Then,*

$$B_d(P, Q, \Pi) - C_d(P, Q, \Pi) \leq \max_{\boldsymbol{b}\in\Pi} \max\{\frac{A(P, \mathcal{W}(\boldsymbol{b}))}{||P||}, \frac{A(Q, \mathcal{W}(\boldsymbol{b}))}{||Q||}\}.$$

The above error bounds can be used to dynamically refine the partition.

1.9 Disparity-guided sampling

The essential idea of our disparity based sampling method is to detect when the dynamics of the system does not allow the tree to expand towards the goal states and then to avoid such situations by favoring a refinement, that is an exploration near the already visited states.

A simple way to bias the sampling towards the set P^k of already visited states is to reduce the sampling space. Indeed, we can make a bounding box of the set P^k and give more probability of sampling inside this box than outside it. Alternatively, we can guide the sampling using the disparity information as follows. The objective now is to reduce the disparity between the set G^k of goal states and the set P^k of visited states. Like the guiding method using the star discrepancy, we define for each elementary box $\boldsymbol{b}$ of the partition a function $\eta(\boldsymbol{b})$ reflecting the potential for reduction of the lower and upper bounds of the disparity between P^k and G^k. This means that we favor the selection of the boxes that make the distribution of goal states G^k approach that of the visited states P^k. Choosing such boxes can improve the quality of refinement. The formulation of the potential influence function for the disparity-based sampling method is similar to that for the coverage guided sampling.

A combination of the coverage guided and the disparity guided sampling methods is done as follows. We fix a time window T_s and a threshold ϵ. When using the coverage guide method, if the algorithm detects that the disparity between the G^k and P^k does not decrease by ϵ after T_s time, it switches to the disparity guided method till the disparity is reduced more significantly and switches back to the coverage guide method. Note that a non-decreasing evolution of the disparity is an indication of the inability of the system to approach the goal states. In an interactive exploration mode, it is possible to let the user manually decide when to switch. As mentioned earlier, we call the resulting algorithm agRRT (the letter 'a' in this acronym stands for "adaptive").

Examples. We use the examples in the previous section to demonstrate the coverage improvement of agRRT. Figure 1.16 shows that the final result for Example 1 obtained using agRRT has a better coverage than that obtained using gRRT. The solid curve represents the coverage of the set P^k of visited states and the dashed one the coverage of the set G^k of goal states. The dash-dot curve represents the disparity between G^k and P^k.

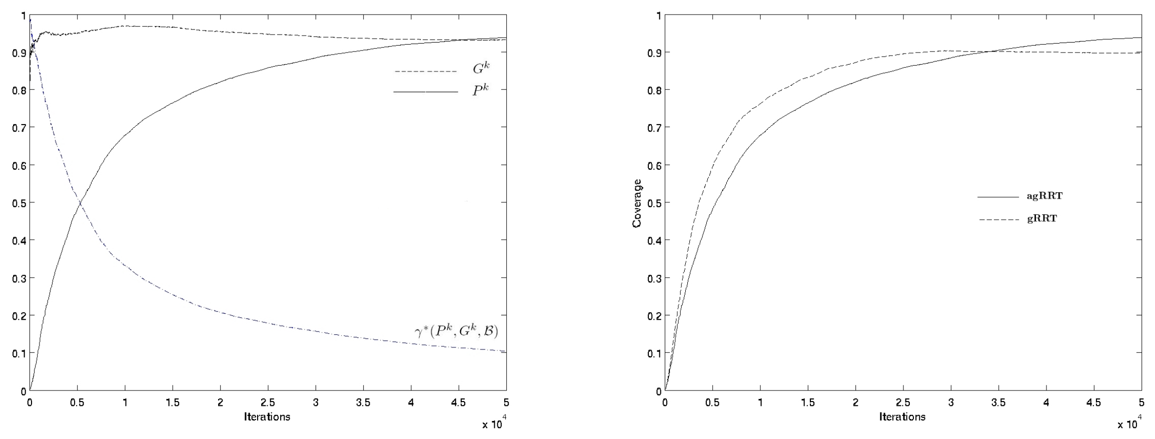


Figure 1.16: Left: Test coverage of the result obtained using agRRT for Example 1. Right: Comparing gRRT and agRRT.

The result obtained using agRRT for Example 2 is shown in Figure 1.17, which also indicates an improvement in coverage quality. The figure on the right shows the set of generated goal states. The states are drawn in dark color. In this example, we can observe the adaptivity of the combination of gRRT and agRRT. Indeed, in the beginning, the gRRT algorithm was used to rapidly expand the tree. After some time, the goal states sampled from the outside of the exact reachable space do not improve the coverage, since they only create more states near the boundary of the reachable set. In this case, the disparity between P^k and G^k does not decrease, and the agRRT is thus used to enable an exploration in the interior of the reachable set. The interior of the reachable set thus has a higher density of sampled goal states than the outside, as one can see in the figure.

1.10 Termination criterion using disparity

We first explain why it is not suitable to use the star discrepancy coverage to decide the termination of the test generation algorithm. Indeed, the star discrepancy allows to com-

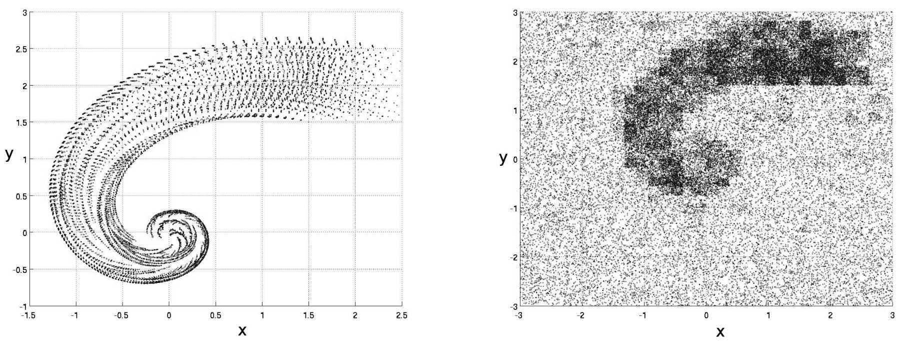


Figure 1.17: Result after $k = 50000$ iterations, obtained using agRRT (left: the set of visited states P^k, right: the set of goal states G^k).

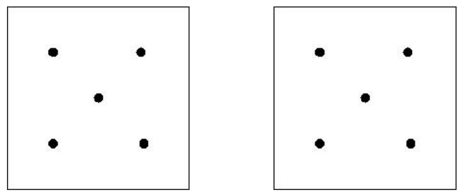


Figure 1.18: The point set P (left) and the point set $P \cup Q$ (right).

pare the spatial coverage quality between the point sets of the same cardinality; it however may give misleading comparison between the sets of different cardinalities. To illustrate this, we consider a set of two-dimensional points: $P = \{(0.25, 0.25), (0.5, 0.5), (0.75, 0.25), (0.25, 0.75), (0.75, 0.75)\}$ inside the bounding box $\mathcal{B} = [0, 1] \times [0, 1]$, shown in Figure 1.18-(left). The star discrepancy estimation gives: $D^*(P, \mathcal{B}) \simeq 0.347$. An arising question is how the star discrepancy changes when we add more points in P. We now consider another non-empty set Q of points inside $\mathcal{B}$ that does not contain any points of P. Different experiments show that the star discrepancy of the union $P \cup Q$ may be larger or smaller than that of P. For example, for the following set $Q = \{(0.06, 0.06), (0.12, 0.12), (0.06, 0.12), (0.12, 0.06)$ the value of $D^*(P \cup Q, \mathcal{B}) \simeq 0.493$ (see Figure 1.18-right). In other words, adding this set Q in P increases the star discrepancy, which can be easily understood because the set Q is not well equidistributed over the box $\mathcal{B}$. However, from a verification point of view, the union $P \cup Q$ provides more information about the correctness of the system than the set P. In addition, geometrically speaking, the set $P \cup Q$ "covers more space" than the set P.

Therefore, we do not use the star discrepancy coverage measure to decide when the test generation algorithm can terminate. Instead, to detect when the coverage reaches a "saturation", we use the disparity between two consecutive sets of visited states. If its value remains below some predefined threshold Δ_γ after a predefined number K of iterations, we can stop the algorithm because the distribution of the sets of visited states are not much changed and the coverage is not significantly improved. This criterion can be used together with a maximal number of iterations.

1.11 Actuator and sensor imprecision

Because of the limitations of practical actuators and sensors, the tester cannot realize exactly an input value specified as a real-valued vector as well as measure exactly the state of the system. We first explain how actuator imprecision influences the testing process.

Actuator imprecision. We consider the following continuous dynamics of a location: $\dot{x}(t) = f(x(t), u(t))$. Given an initial state x and an input value u, let $\xi_{x,u}$ denote the unique trajectory during the time interval $[0, h]$.

Because of actuator imprecision, when the tester emits an input u to the system under test, indeed some input $v = u + \delta_u$ with $|\delta_u| \leq \epsilon_u$ is applied. We call ϵ_u the actuator imprecision bound. After one step, this uncertainty causes the new state $y = \xi_{x,v}(h)$ to deviate from the exact state $y' = \xi_{x,u}(h)$. In the next step, this deviation can be considered as an uncertainty in the initial condition, namely $(y' - y)$, which causes further deviation.

Therefore, only when the observation measured by the tester lies outside some admissible deviation neighborhood, the conformance property is considered violated. Therefore, we must compute the admissible deviation neighborhoods.

Let $M_x = \dfrac{\partial \xi_{x,u}}{\partial x}$ and $M_u = \dfrac{\partial \xi_{x,u}}{\partial u}$ denote the partial derivatives of $\xi_{x,u}$ with respect to the initial condition x and to the input u. They are called the *sensitivity matrices*. Given a neighboring initial state y and a neighboring input value v, we can get an estimation of $\xi_{y,v}$ by dropping higher order terms in the Taylor expansion of $\xi_{x,u}(t)$ seen as a function of

x and of u:

$$\hat{\xi}_{y,v}(h) = \xi_{x,u}(h) + M_x(h)(y-x) + M_u(h)(v-u) \tag{1.6}$$

The deviation is thus $\delta = |M_x(h)(y-x) + M_u(h)\epsilon_u|$. The deviation neighborhood is defined as a ball centered at $\xi_{x,u}(h)$ with radius δ. To compute the sensitivity matrices, we solve the following differential equations, which result from taking the derivative of the solution:

$$\begin{aligned} \dot{x} &= f(x,u), && (1.7)\\ \dot{M}_x &= \frac{\partial f(x,u)}{\partial x} M_x, && (1.8)\\ \dot{M}_u &= \frac{\partial f(x,u)}{\partial x} M_u + \frac{\partial f(x,u)}{\partial u} && (1.9) \end{aligned}$$

with the initial condition $\xi_x(0) = x$ and $M_x(0) = \mathbf{I}_n$ (the identity matrix) and $M_u(0) = \mathbf{0}_n$ (the zero matrix). These are $n+m+1$ ordinary differential equations of order n. Note that the cost can be made less than that of the resolution of $n+m+1$ different systems since the Jacobian $\dfrac{\partial f(x,u)}{\partial x}$ can be extensively reused in the computation.

Therefore, in the RRT tree construction, when we compute a new state from a given state x and input u, we additionally compute the corresponding sensitivity matrix $M_x(h)$ and $M_u(h)$ and associate these matrices to the new node created to store the new state. We also propagate the neighborhoods in order to detect possible executions of uncontrollable discrete transitions. If the current state does not enable a uncontrollable transition but the associated neighborhood intersects with its guard, we compute the image of the intersection by the reset map and take its centroid to define a new visited state. The diameter of the image is the radius of the associated neighborhood.

Verdict. During the test, let $\tilde{s}^i = (\tilde{q}^i, \tilde{x}^i)$ be the observation at the i^{th} step and $s^i = (q^i, x^i)$ be the corresponding expected state in the tree. Let M_x^i and M_u^i be the sensitivity matrices associated with the node (q^i, x^i). The system under test is assumed to satisfy the conformance property at the initial state. The verdict is then decided as follows. We suppose that the conformance property is not violated until the i^{th} step. There are two cases:

1. (C1) $s^{(i+1)}$ is reached by the continuous dynamics. Then, if $|\tilde{x}^{(i+1)} - x^{(i+1)}| > M_x^i|\tilde{x}^i -$

$x^i| + M_u^i \epsilon_u$, we conclude that the conformance property is violated and stop the test. Otherwise, we continue the test.

2. (C2) $s^{(i+1)}$ is reached by a discrete transition. Then, if $\tilde{q}^i = q^i$ and $|\tilde{x}^{(i+1)} - x^{(i+1)}| > \delta^{(i+1)}$ where $\delta^{(i+1)}$ is the deviation radius, we conclude that the conformance property is violated and stop the test. Otherwise, we continue the test.

Sensor imprecision. Let the sensor imprecision be modeled by an upper bound ϵ_x on the distance between the actual continuous state and the observation measured by the tester. The distance between the observation and the expected states is now tested againts a bound larger by ϵ_x, that is $|\tilde{x}^{(i+1)} - x^{(i+1)}| > M_x^i(\tilde{x}^i - x^i) + M_u^i \epsilon_u + \epsilon_x$ for the above (C1) and $|\tilde{x}^{(i+1)} - x^{(i+1)}| > \delta^{(i+1)} + \epsilon_x$ for (C2).

1.12 Tool HTG

We have implemented the above algorithms in a test generation tool, called HTG. Its implementation uses a data structure similar to a k-d tree, which facilitates the required operations, such as updating the tree, finding neighbors, updating the star discrepancy and disparity. Indeed, using this data structure, we can efficiently encode a box partition for storing and accessing the visited states as well as for the star discrepancy and disparity estimations. In the following, we briefly describe some important functions. A more detailed description of the implementation can be found in [12]

1.12.1 Numerical simulation

In most classic versions of hybrid automata, continuous dynamics are defined using ordinary differential equations (ODEs). To analyze analog and mixed-signal circuits, the behavior of which are described using differential algebraic equations (DAEs), we adapt the model to capture this particularity and use the well-known RADAU algorithm for solving DAEs [21]. In addition, with view to applications in analog and mixed-signal circuits, an efficient and reliable simulation method is key. The state-of-the-art SPICE simulator is prone to convergence problems when dealing with circuit components with stiff characteristics. We also integrated in our test generation tool a connection to the platform SICONOS, developed at INRIA Rhônes-Alpes, that contains a number of simulation algorithms based on *the non-smooth approach* [2], which have proved to be efficient for systems with stiff dynamics.

1.12.2 SPICE netlists as input systems.

An important new feature of the tool is its ability to deal with circuits describes using SPICE, in addition to textual descriptions of a hybrid automata. This facilitates the application of the tool to practical circuits. Note that our test generation method works on the differential equations of a hybrid automaton, which is an appropriate mathematical model to describe circuit dynamics. However, the circuit equations must be generated from a SPICE netlist. Common SPICE parsers do not directly provide a succinct form of circuit equations but generate complex equations for the numerical resolution of each simulation step. Furthermore, using SPICE descriptions we cannot specify uncertain inputs. The solution we propose to address this can be described as follows:

- The inputs (controllable by tester) can be source currents or voltages in SPICE. Their uncertainty is described in an auxiliary file.
- We use the tool ACEF [1] to generate the numerical simulation equations in each step. The values of the conptrollable inputs, computed by the test generation algorithm, are communicated to ACEF [1] before the generation in each step. The generation can be optimized by only updating some terms involving the modified input in the equations of the previous steps.

The test cases generated by the tool can be visualized by different standard viewers such as **MATLAB**$^{®}$ and **gnu** plots.

The tool can also be used as a systematic simulator to verify properties of a model. For this problem, the tester can manipulate not only control inputs but also disturbance inputs.

Systematic simulation can be seen as a good compromise between exhaustive verification and ad-hoc simulation.

1.13 Applications

The tool has been tested on various examples, which proved its scalability to high dimensional systems (up to a few hundred continuous variables) [13]. In addition, we have also successfully applied the tool to a number of case studies in control systems and in analog and mixed signal circuits. In the following, to provide an overview of the applicability of the tool, we briefly report some of these applications.

1.13.1 Aircraft collision avoidance system

To illustrate the application of our algorithm to hybrid systems, we use the aircraft collision avoidance problem [32], which is a well-known benchmark in the hybrid systems literature. In [32], the authors treated the problem of collision avoidance of two aircraft. To show the scalability of our approach we consider the same model with N aircraft.

As shown in Figure 1.19, all the aircraft are at a fixed altitude. Each aircraft i has three states (x_i, y_i, θ_i) where x_i and y_i describe the position and θ_i is the relative heading of the aircraft. Each aircraft begins in straight flight at a fixed relative heading (mode 1).

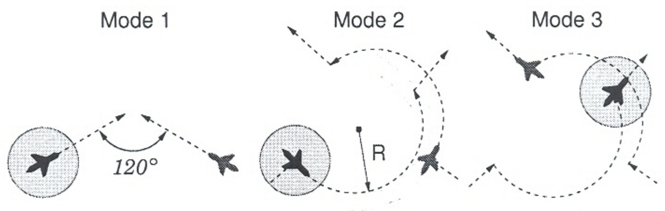


Figure 1.19: Aircraft behavior in the three modes [32].

As soon as two aircraft are within the distance R between each other, they enter mode 2. In this mode each aircraft makes an instantaneous heading change of 90 degrees, and begins a circular flight for π time units. After that, they switch to mode 3 and make another instantaneous heading change of 90 degrees and resume their original headings from mode 1.

The dynamics of the system are shown in Figure 1.20. The guard transition between mode 1 and mode 2 is given by $Dij < R$, which means that the aircraft i is at R distance from the aircraft j. The dynamics of each aircraft is as follows:

$$\begin{aligned}\dot{x}_i &= vcos(\theta_i) + dx_i, \\ \dot{y}_i &= vsin(\theta_i) + dy_i \\ \dot{\theta}_i &= \omega\end{aligned}$$

The continuous inputs are dx_i and dx_i describing the external disturbances on the aircraft (such as wind):

$$\begin{aligned}dx_i &= d_1 sin(\theta_i) + d_2 cos(\theta_i), \\ dy_i &= -d_1 cos(\theta_i) + d_2 sin(\theta_i),\end{aligned}$$

and $-\delta \leq d_1, d_2 \leq \delta$.

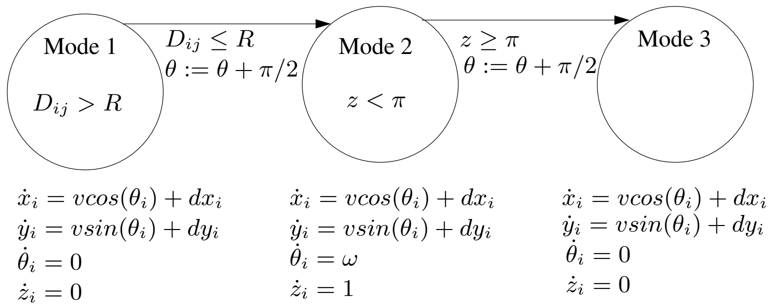

Mode 1
$D_{ij} > R$
$D_{ij} \leq R$
$\theta := \theta + \pi/2$
Mode 2
$z < \pi$
$z \geq \pi$
$\theta := \theta + \pi/2$
Mode 3
$\dot{x}_i = vcos(\theta_i) + dx_i$
$\dot{y}_i = vsin(\theta_i) + dy_i$
$\dot{\theta}_i = 0$
$\dot{z}_i = 0$
$\dot{x}_i = vcos(\theta_i) + dx_i$
$\dot{y}_i = vsin(\theta_i) + dy_i$
$\dot{\theta}_i = \omega$
$\dot{z}_i = 1$
$\dot{x}_i = vcos(\theta_i) + dx_i$
$\dot{y}_i = vsin(\theta_i) + dy_i$
$\dot{\theta}_i = 0$
$\dot{z}_i = 0$

Figure 1.20: System dynamics for the three modes.

Results. For N aircraft, the system has $3N + 1$ continuous variables (one for modeling a clock). For the case of $N = 2$ aircraft, when the collision distance is 5 (that is a collision occurs if the distance between the aircraft is not greater than 5), no collision was detected after visiting 10000 visited states, and the computation time was 0.9 min. The result for

$N = 8$ aircraft with the disturbance bound $\delta = 0.06$ is shown in Figure 1.21, where we show the projected positions of the eight aircraft on a 2-dimensional space. For this example, the computation time for 50000 visited states was 10 min and a collision was found. For a similar example with $N = 10$ aircraft, the computation time was 14 min and a collision was also found.

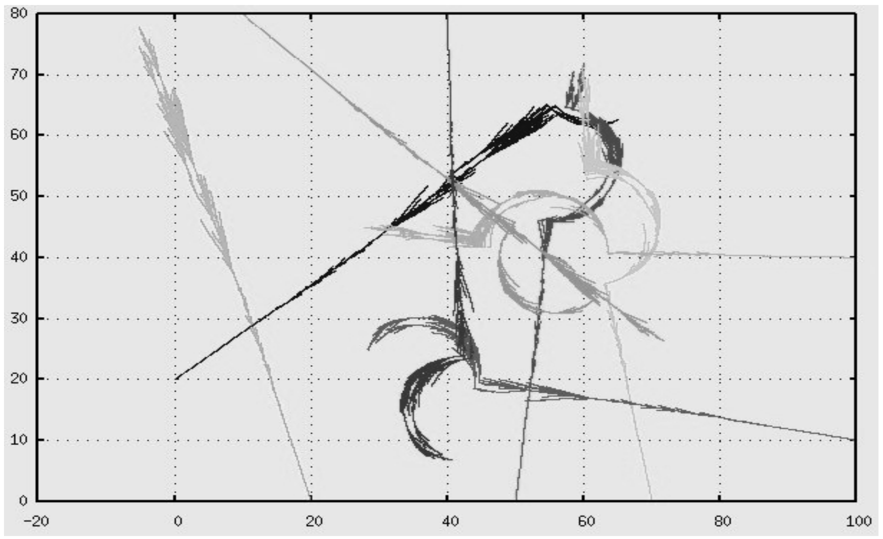


Figure 1.21: Eight-aircraft collision avoidance (50000 visited states, computation time: 10 min.

1.13.2 Robotic vehicle benchmark

Another example is adapted from the robotic navigation system benchmark [35]. We consider a car with the following continuous dynamics with 5 variables: $\dot{x} = vcos(\theta)$, $\dot{y} = vsin(\theta)$, $\dot{\theta} = vtan(\phi)/L$, $\dot{v} = u_0$, $\dot{\phi} = u_1$ where x, y, θ describe the position and heading of the car, v is its speed and ϕ is its steering angle. The car can be in one of three car modes (smooth car, smooth unicycle, smooth differential drive). In this work, we consider only the smooth car mode.

The inputs of the system are u_0 and u_1 which are respectively the acceleration and steering control. The system uses a hybrid control law with 3 driver modes. In the first driver mode, called *RandomDriver*, the control inputs are selected uniformly at random between their lower and upper bounds. In the second driver mode, called *StudentDrive*, when the speed is low, u_0 is randomly chosen as in first mode; otherwise, the strategy is to reduce the speed. In the third driver mode, called *HighwayDrive*, the strategy is to reduce the speed when it is high and increase it when it is low. A detailed description of this control law can be found in [35].

Rather than to analyze a realistic navigation system model, we use this example to test the efficiency of our algorithms on a hybrid system with a larger number of locations. To this end, we created from this system two models. The terrain is partitioned into K rectangles using a regular grid $\mathcal{G} = \{0, \ldots, K_x - 1\} \times \{0, \ldots, K_y - 1\}$. Each rectangle is associated with a driver mode. The first model is a hybrid automaton with $K_x K_y$ locations and the system can only switch between the locations corresponding to adjacent rectangles.

In the second model, we allow more complicated switching behavior by letting the system jump between some rectangles that are not necessarily adjacent. The rectangle corresponding to the grid point $(i, j) \in \mathcal{G}$ is $R_{ij} = [il_x, jl_y] \times [(i+1)l_x, (j+1)l_y]$ where l_x and l_y are the sizes of the grid in the x and y coordinates. The absolute index of R_{ij} is an integer defined as follows: $\iota(R_{ij}) = iK_y + j$. From the rectangle R_{ij} with even absolute index, we allow a transition to R_{mn} such that $\iota(R_{mn}) = (\iota(R_{ij}) + J) mod(K_x K_y)$ (where $J > 0$ and *mod* is the modulo division). The guard set at R_{ij} is the right-most band of width ϵ_g, that is $[(i+1)l_x - \epsilon_g, jl_y] \times [(i+1)l_x, (j+1)l_y]$. After switching to R_{mn}, the car position (x, y) is reset to a random point inside the square of size ϵ_r defined as $[ml_x, (n+1)l_y - \epsilon_r] \times [ml_x + \epsilon_r, (n+1)l_y]$.

We compared the results obtained for the two models using the gRRT algorithm and the hRRT algorithm. In this experimentation the hRRT algorithm uses a uniform sampling (both over the discrete and continuous state space). Since we want to focus on the performance of the guiding tool, the two algorithms use the same hybrid distance definition and implementation. The parameters used in this experimentation are: $l_x = l_y = 20$, $l_x = l_y = 20$, the car position $(x, y) \in [-100, 100] \times [-100, 100]$, $\epsilon_g = \epsilon_r = 6$, $J = 6$. The number of locations in the hybrid automata is 100. For the first model without jumps, in terms of coverage efficiency, the algorithms are comparable. For the model with jumps, gRRT systematically produced better coverage results. However, gRRT is not always better than hRRT in terms of the number of covered locations. This is because of our coverage definition using the average of the continuous-state coverages of all the locations.

In terms of time efficiency, we now report the computation time of gRRT for the experimentations with various maximal visited states. For the first model, the computation times of gRRT are: 4.7s for 10000 states in the tree, 1min 26s for 50000 states, 6min 7s for 100000 states. For the second model, the computation times of gRRT are: 4.2s for 10000 states in the tree, 2min 5s for 50000 states, 4min 40s for 100000 states, and 20min 22s for 150000 states.

1.13.3 Analog and mixed-signal circuits

Using the above results, we implemented a test generation tool and tested it on a number of control applications, which proved its scalability to high dimensional systems [33]. In this implementation, all the sets encountered in the hybrid automaton definition are convex polyhedra. For circuit applications, we use the well-known RADAU algorithm for solving differential algebraic equations (DAE) [21]. We recall that solving high index[5] and stiff DAEs is computationally expensive, and in order to evaluate the efficiency of the test generation algorithm, we have chosen two practical circuits with DAEs of this type. The three circuits we treated are: a transistor amplifier, a voltage controlled oscillator, and a Delta-Sigma modulator circuit.

Transistor amplifier. The first example is a transistor amplifier model [21], shown in Figure 1.22, where the variable y_i is the voltage at node i; U_e is the input signal and $y_8 = U_8$ is the output voltage. The circuit equations are a system of DAEs of index 1 with 8 continuous

[5]The differential index of a DAE is a measure of the singularity of the DAE. It characterizes the difficulty in numerically solving the equation.

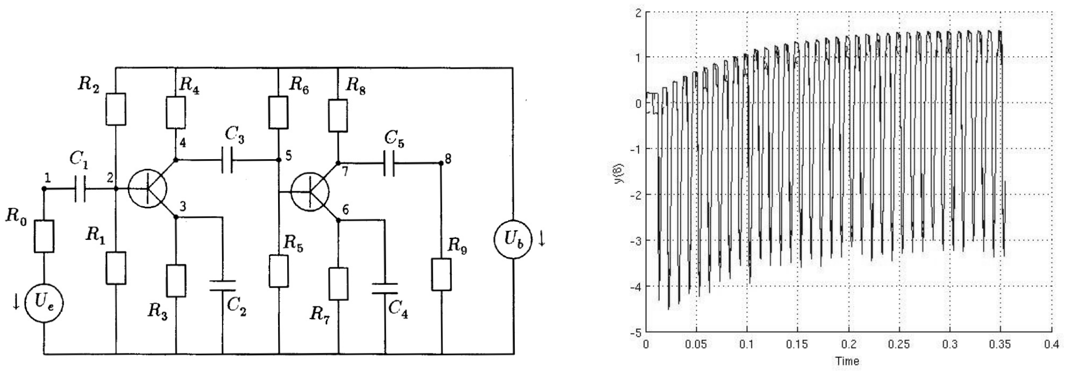


Figure 1.22: Test generation result for the transitor amplifier.

variables: $M\dot{y} = f(y, u)$. The function f is given by:

$$\begin{pmatrix} -U_e/R_0 + y_1/R_0 \\ -U_b/R_2 + y_2(1/R_1 + 1/R_2) - (\alpha - 1)g(y_2 - y_3) \\ -g(y_2 - y_3) + y_3/R_3 \\ -U_b/R_4 + y_4/R_4 + \alpha g(y_2 - y_3) \\ -U_b/R_6 + y_5(1/R_5 + 1/R - 6) - (\alpha - 1)g(y_5 - y_6) \\ -g(y_5 - y_6) + y_6/R_7 \\ -U_b/R_8 + y_7/R_8 + \alpha g(y_5 - y_6) \\ y_8/R_9 \end{pmatrix}.$$

The circuit parameters are: $U_b = 6$; $U_F = 0.026$; $R_0 = 1000$; $R_k = 9000$, $k = 1, \ldots, 9$; $C_k = k10^{-6}$; $\alpha = 0.99$; $\beta = 10^{-6}$. The initial state $y_{init} = (0, U_b/(R_2/R_1 + 1), U_b/(R_2/R_1 + 1), U_b, U_b/(R_6/R_5 + 1), U_b/(R_6/R_5 + 1), U_b, 0)$. To study the influence of circuit parameter uncertainty, we consider a perturbation in the relation between the current through the source of the two transistors and the voltages at the gate and source $I_S = g(U_G - U_S) = \beta(e^{\frac{U_G - U_S}{U_F}} - 1) + \epsilon$, with $\epsilon \in [\epsilon_{min}, \epsilon_{max}] = [-5e-5, 5e-5]$. The input signal $U_e(t) = 0.1sin(200\pi t)$. The acceptable interval of U_8 in the non-perturbed circuit is $[-3.01, 1.42]$. Once the initial transient period has settled down, the test case indicates the presence of traces with overshoots after 18222 iterations (corresponding to 1.1mn of computation time). The total computation time for generating 50000 states was 3 minutes. Figure 1.22 shows the generated states projected on U_8 over the first 0.03 seconds.

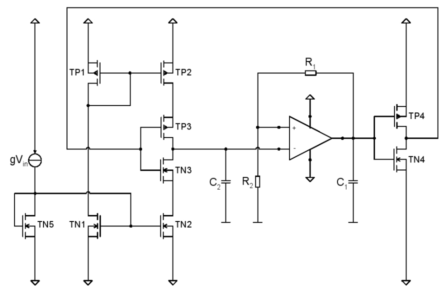


Figure 1.23: Voltage controlled oscillator (VCO) circuit.

Voltage controlled oscillator. The second circuit we examined is a voltage controlled oscillator (VCO) circuit [19], described by a system of DAEs with 55 continuous variables. In this circuit, the oscillating frequency of the variables v_{C_1} and v_{C_2} is a linear function of the input voltage u_{in}. We study the influence of a time-variant perturbation in C_2, modeled as an input signal, on this frequency. In this example we show that, in addition to conformance relation, using this framework, we can test a property of the input/output relation. The oscillating period $T \pm \delta$ of v_{C_2} can be expressed using a simple automaton with one clock y in Figure 1.24. The question is to know if given an oscillating trace in $\mathcal{A}$, its corresponding trace in $\mathcal{A}_s$ also oscillates with the same period. This additional automaton can be used to determine test verdicts for the traces in the computed test cases. If an observation sequence corresponds to a path entering the "Error" location, then it causes a "fail" verdict. Since we cannot use finite traces to prove a safety property, the set of obsevation sequences that cause a "pass" verdict is empty, and therefore the remaining obsevation sequences (that do not cause a "fail" verdict) cause an "inconclusive" verdict. We consider a constant input voltage $u_{in} = 1.7$. The coverage measure was defined on the projection of the state space on v_{C_1} and v_{C_2}. The generated test case shows that *after the transient time*, under a time-variant deviation of C_2 which ranges within $\pm 10\%$ of the value of $C_2 = 0.1e-4$, the variables v_{C_1} and v_{C_2} oscillate with the period $T \in [1.25, 1.258]s$ (with $\varepsilon = 2.8e-4$). This result is consistent with the result presented in [19]. The number of generated states was 30000 and the computation time was 14 minutes. Figure 1.24 shows the explored traces of v_{C_2} over time.

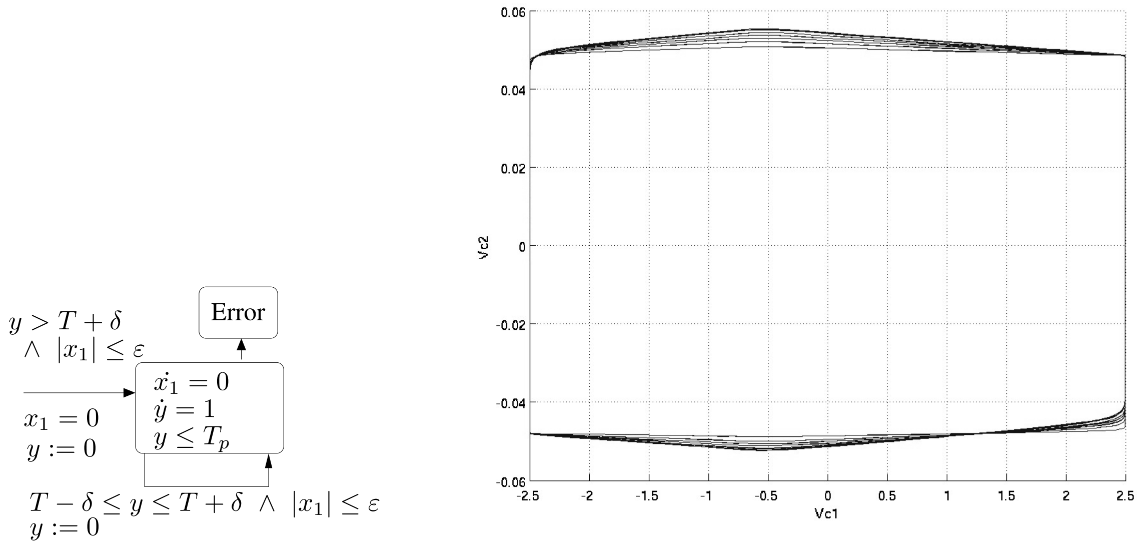

Error
$y > T + \delta$
$\wedge\ |x_1| \leq \varepsilon$
$\dot{x_1} = 0$
$\dot{y} = 1$
$y \leq T_p$
$x_1 = 0$
$y := 0$
$T - \delta \leq y \leq T + \delta\ \wedge\ |x_1| \leq \varepsilon$
$y := 0$
Vc2
Vc1

Figure 1.24: Left: Automaton for an oscillation specification. Right: Variable v_{C_2} over time. The number of generated states was 30000 and the computation time was 14mn.

Delta-Sigma circuit. The third example is a third-order Delta-Sigma modulator [6], which is a mixed-signal circuit shown in Figure 1.25. When the input sinusoid is positive and its value is less than 1, the output takes the +1 value more often and the quantization error is fed back with negative gain and accumulated in the integrator $\frac{1}{z-1}$. Then, when the accumulated error reaches a certain threshold, the quantizer switches the value of the output to -1 to reduce the mean of the quantization error. A third-order Delta-Sigma modulator is modeled as a hybrid automaton, shown in Figure 1.26. The discrete-time dynamics of the system is as follows: $x(k+1) = Ax(k) + bu(k) - sign(y(k))a$, $y(k) = c_3x_3(k) + b_4u(k)$ where $x(k) \in \mathbb{R}^3$ is the integrator states, $u(k) \in \mathbb{R}$ is the input, $y(k) \in \mathbb{R}$ is the input of the quantizer. Thus, its output is $v(k) = sign(y(k))$, and one can see that whenever v remains constant, the system dynamics is affine continuous. A modulator is stable if under a bounded input, the states of its integrators are bounded.

The test generation algorithm was performed for the initial state $x(0) \in [-0.01, 0.01]^3$ and the input values $u(k) \in [-0.5, 0.5]$. After exploring only 57 states, saturation was already detected. The computation time was less than 1 second. Figure 1.27 shows the values of $(\sup x_1(k))_k$ as a function of the number k of time steps. We can see that the sequence

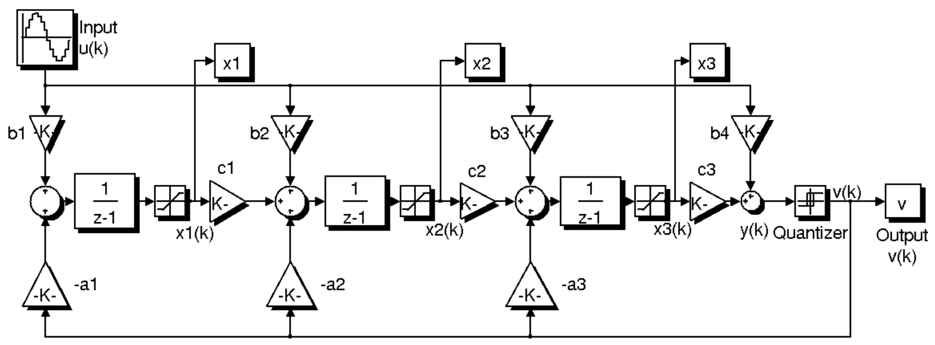


Figure 1.25: A third-order modulator: Saturation blocks model saturation of the integrators.

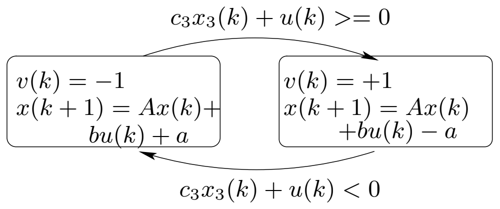


Figure 1.26: A hybrid automaton modeling the modulator.

$(\sup x_1(k))_k$ leaves the safe interval $[-x_1^{sat}, x_1^{sat}] = [-0.2, 0.2]$, which indicates the instability of the circuit. This instability for a fixed finite horizon was also detected in [11] using an optimization-based method.

1.14 Conclusion and related work

The main results presented in this chapter can be summarized as follows. We proposed a formal framework for conformance testing of hybrid systems. This framework uses the commonly-accepted hybrid automaton model. We also proposed a novel coverage measure, which not only is useful as a criterion to evaluate testing quality but also can be used to guide the test generation process. We then developed a number of coverage-guided test generation algorithms for hybrid systems. These algorithms are based on a combination of ideas from robotic path planning, equidistribution theory, algorithmic geometry, and numerical simulation. Based on these results, we implemented a tool for conformance testing of hybrid systems, called HTG, which allowed us to treat a number of case studies from control applications as well as from analog and mixed signal circuits. The experimental

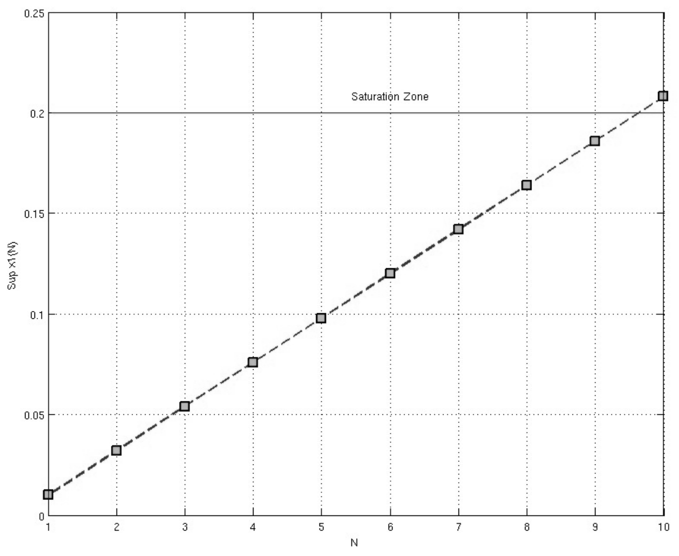


Figure 1.27: Test generation result for the Delta-Sigma circuit. The computation time was less than 1s.

results obtained using the tool HTG show its applicability to systems with complex dynamics and its scalability to high dimensional systems.

The remainder of this section is devoted to related work on hybrid systems testing. Concerning hybrid systems testing, the work [40] developed a theoretical framework for testing hybrid input-output conformance of hybrid transition system models. On an abstract level, this conformance relation is very close to the one we developed in this work. However, no concrete test generation algorithms and implementation were proposed in [40]. Concerning other models, a conformance testing framework was developed for qualitative action systems where the continuous dynamics are described using qualitative differential equations [3].

The paper [36] proposed a framework for generating test cases from simulation of hybrid models specified using the language CHARON [5]. In this work, the test cases are generated by restricting the behaviors of an environment automaton to yield a deterministic testing automaton. A test suite can thus be defined as a finite set of executions of the environment automaton. It is mentioned in [5] that to achieve a desired coverage, non-determinism in the environment automaton is resolved during the test generation using some randomized algorithm. However, this coverage as well as the randomized algorithm were not described in detail. Besides testing a real system, another goal of [5] is to apply tests to models, as an alternative validation method. In [24], the testing problem is formulated as to find a piecewise constant input that steers the system towards some set, which represents a set of bad states.

The RRT algorithm has been used to solve a variety of reachability-related problems such as hybrid systems planning, control, verification, and testing (see for example [15, 9, 24, 10, 35] and references therein). Here we only discuss a comparison of our approach with some existing RRT-based approaches for the validation of continuous and hybrid systems. Concerning the problem of defining a hybrid distance, our hybrid distance is close to that proposed in [24]. The difference is that we use the centroids of the guard sets to define the distance between these sets, while the author of [24] uses the minimal clearance distance between these sets, which is more difficult to compute. To overcome this difficulty, the author proposed to approximate this clearance distance by the diameter of the state space. An advantage of our hybrid distance is that it captures better the average cases, allowing not to always favor the extreme cases. Note also that our hybrid distance d_H does not take into account the system dynamics. It is based on the spatial positions of the states. In [24] the author proposed a time-based metric for two hybrid states, which can be seen as an approximation of the minimal time required to reach one state from another, using the information on the derivatives of the variables. Another distance proposed in [24] is called specification-based. This distance is typically defined with respect to some target set specifying some reachability property. It can be however observed that for many systems, this "direct" distance may mislead the exploration due to the controllability of the system. In [15, 24] and in our hRRT algorithm, the problem of optimally steering the system towards the goal states was not addressed. In other words, the evolution of the tree is mainly determined by the selection of nearest neighbors. In [9], the problem of computing optimal successors was considered more carefully, and approximate solutions for linear dynamics as well as for some particular cases of non-linear dynamics were proposed. The authors of [35] proposed a search on a combination of the discrete structure and the coarse-grained decomposition of the continuous state space into regions, in order to determine search directions. This can be thought of as an implicit way of defining a hybrid distance as well as a guiding heuristics.

Concerning test coverage for continuous and hybrid systems, in [15] the authors proposed a coverage measure based on a discretized version of dispersion, since the dispersion in general is very expensive to compute. Roughly speaking, the dispersion of a point set with respect to various classes of range spaces, such as balls, is the area of the largest empty range. This measure is defined over a set of grid points with a fixed size δ. For a given test, the spacing s_g of a grid point g is the distance from g to the nearest visited state by the test if it is smaller than δ, and $s_g = \delta$ otherwise. Let S be the sum of the spacings of all the grid points. This means that the value of S is the largest when the set of visited states is empty. Then, the coverage measure is defined in terms of how much the vertices of the tree reduce the value of S. It is important to note that while in our work, the coverage measure is used to guide the simulation, in [15] it is used as a termination criterion. The paper [23] addresses the problem of robust testing by quantifying the robustness of some properties under parameter perturbations. The work in [23] also considers the problem of how to generate test cases with a number of initial state coverage strategies. In addition, the work [18] is similar to ours in the idea of exploiting the existence of metrics on the system state space, which is natural for continuous and hybrid systems. Indeed, by using a concept of approximate bisimulation metrics, one can infer all the possible behaviors of the system in a neighborhood of one trajectory. Hence, by a finite number of simulations it is possible to decide whether the system is correct under all possible disturbances. However, this approach is applicable only for stable systems and the bisimulation metrics can be effectively computed only for restrictive classes of systems, such as systems with linear continuous dynamics.

Concerning guided exploration, sampling the configuration space has been one of the fundamental issues in probabilistic motion planning. Our idea of guiding the test generation via the sampling process has some similarity with the sampling domain control [42]. As mentioned earlier, the RRT exploration is biased by the Voronoi diagram of the vertices of the tree. If there are obstacles around such vertices, the expansion from them is limited and choosing them frequently can slow down the exploration. In the dynamic-domain RRT algorithm, the domains over which the goal points are sampled must reflect the geometric and differential constraints of the system, and more generally, the controllability of the system. In [31], another method for biasing the exploration was proposed and its main idea is to reduce the dispersion in an incremental manner. This idea is thus very close to the idea of our guiding method in spirit; however, their concrete realizations are different. The method in [31] attempts to lower the dispersion by using K samples in each iteration (instead of a single sample) and then select from them a best sample by taking into account the feasibility of growing the tree towards it. Finally, we mention that a similar idea was used in [15] where the number of successful iterations is used to define an adaptive biased sampling. To sum up, the novelty in our guiding method is that we use the information about the current coverage of the visited states in order to improve the coverage quality. Additionally, we combine this with controllability information (obtained from the disparity estimation) to obtain a more efficient guiding strategy.

A number of directions for future research can be identified. First, we are interested in defining a measure for trace coverage. Partial observability also must be considered. Convergence rate of the exploration in the test generation algorithm is another interesting theoretical problem to tackle. This problem is particular difficult especially in the verification context where the system is subject to uncontrollable inputs.

Bibliography

[1] V. Acary, O. Bonnefon, and P. Denoyelle. Automatic circuit equation formulation for nonsmooth electrical circuits. Technical report, BIPOP-INRIA, ANR VAL-AMS Report, Jan 2008.

[2] V. Acary and F. Pérignon. SICONOS: A software platform for modeling, simulation, analysis and control of non smooth dynamical system. In *Proceedings of MATHMOD 2006, 5th Vienna Symposium on Mathematical Modelling*, Vienna, 2006. ARGESIM Verlag.

[3] B. K. Aichernig, H. Brandl, and W. Krenn. Qualitative action systems. In *Formal Methods and Software Engineering, 11th International Conference on Formal Engineering Methods, ICFEM 2009*, volume 5885 of *Lecture Notes in Computer Science*, pages 206–225. Springer, 2009.

[4] 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.

[5] R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivan, C. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical modeling and analysis of embedded systems, 2002.

[6] P. M. Aziz, H. V. Sorensen, and J. van der Spiegel. An overview of Sigma-Delta converters. *Signal Processing Magazine, IEEE*, 13(1):61–84, 1996.

[7] J. Beck and W. W. L. Chen. Irregularities of distribution. In *Acta Arithmetica.* Cambridge University Press, 1997.

[8] Saddek Bensalem, Marius Bozga, Moez Krichen, and Stavros Tripakis. Testing conformance of real-time applications by automatic generation of observers. *Electr. Notes Theor. Comput. Sci.*, 113:23–43, 2005.

[9] A. Bhatia and E. Frazzoli. Incremental search methods for reachability analysis of continuous and hybrid systems. In *Hybrid Systems: Computation and Control*, volume 2993 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2004.

[10] M. Branicky, M. Curtiss, J. Levine, and S. Morgan. Sampling-based reachability algorithms for control and verification of complex systems. In *Thirteenth Yale Workshop on Adaptive and Learning Systems*, 2005.

[11] T. Dang and A. Donzé and O. Maler. Verification of analog and mixed-signal circuits using hybrid systems techniques. In Alan J. Hu and Andrew K. Martin, editors, *FMCAD'04 - Formal Methods for Computer Aided Design*, LNCS 3312, pages 21–36. Springer-Verlag, 2004.

[12] T. Dang and T. Nahhal. Model-based testing of hybrid systems. Technical report, Verimag, IMAG, Nov 2007.

[13] Thao Dang and Tarik Nahhal. Coverage-guided test generation for continuous and hybrid systems. *Formal Methods in System Design*, 34(2):183–213, 2009.

[14] David Dobkin and David Eppstein. Computing the discrepancy. In *Proceedings of the ninth annual symposium on Computational geometry SCG'93*, pages 47–52. ACM Press, 1993.

[15] J. Esposito, J. W. Kim, and V. Kumar. Adaptive RRTs for validating hybrid robotic control systems. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, Zeist, The Netherlands, July 2004.

[16] Henri Faure. Discrépance de suites associées à un système de numération (en dimension s). *Acta Arithm.*, 41:337–351, 1982.

[17] Marie-Claude Gaudel. Testing can be formal, too. In *6th International Joint Conference CAAP/FASE on Theory and Practice of Software Development*, pages 82–96. Springer-Verlag, 1995.

[18] A. Girard and G. J. Pappas. Verification using simulation. In *Hybrid Systems: Computation and Control HSCC'06*, LNCS 3927, pages 272–286. Springer, 2006.

[19] Darius Grabowski, Daniel Platte, Lars Hedrich, and Erich Barke. Time constrained verification of analog circuits using model-checking algorithms. *Electr. Notes Theor. Comput. Sci.*, 153(3):37–52, 2006.

[20] P. A. V. Hall H. Zhu and J. H. R. May. Software unit test coverage and adequacy. *ACM Computing Surveys (CSUR)*, 29(4):366–427, December 1997.

[21] E. Hairer, C. Lubich, and M. Roche. The numerical solution of differential-algebraic systems by Runge Kutta methods. In *Lecture Notes in Mathematics 1409*. Springer-Verlag, 1989.

[22] T.A. Henzinger. Hybrid automata with finite bisimulations. In F. Vaandrager and J. van Schuppen, editors, *Proc. ICALP'95*, LNCS 944, pages 324–335. Springer-Verlag, 1995.

[23] A. Agung Julius, Georgios E. Fainekos, Madhukar Anand, Insup Lee, and George J. Pappas. Robust test generation and coverage for hybrid systems. In *Hybrid Systems: Computation and Control*, volume 4416 of *Lecture Notes in Computer Science*, pages 329–342. Springer, 2007.

[24] J. Kim, J. Esposito, and V. Kumar. Sampling-based algorithm for testing and validating robot controllers. *Int. J. Rob. Res.*, 25(12):1257–1272, 2006.

[25] J. Kuffner and S. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA'2000), San Francisco, CA*, April 2000.

[26] Kim G. Larsen, Marius Mikucionis, and Brian Nielsen. Online testing of real-time systems using UPPAAL: Status and future work. In E. Brinksma, W. Grieskamp, J. Tretmans, and E. Weyuker, editors, *Perspectives of Model-Based Testing*, volume 04371 of *Dagstuhl Seminar Proceedings*, sept 2004.

[27] S. LaValle and J. Kuffner. Rapidly-exploring random trees: Progress and prospects, 2000. In Workshop on the Algorithmic Foundations of Robotics.

[28] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, May 2001.

[29] S.M. LaValle, M.S. Branicky, and S.R. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *Intl. Journal of Robotics Research*, 23(7-8):673–692, August 2004.

[30] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - A survey. In *Proceedings of the IEEE*, volume 84, pages 1090–1126, 1996.

[31] S. R. Lindemann and S. M. LaValle. Incrementally reducing dispersion by increasing Voronoi bias in RRTs. In *Proceedings IEEE International Conference on Robotics and Automation*, 2004.

[32] I. Mitchell and C. Tomlin. Level set method for computation in hybrid systems. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, LNCS 1790, pages 311–323. Springer-Verlag, 2000.

[33] T. Nahhal and T. Dang. Test coverage for continuous and hybrid systems. In *Computer Aided Verification CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 454–468. Springer, 2007.

[34] Alexandre Petrenko and Andreas Ulrich, editors. *Test Cases Generation for Nondeterministic Real-Time Systems*, volume 2931 of *LNCS*. Springer, 2003.

[35] E. Plaku, L. Kavraki, and M. Vardi. Hybrid systems: From verification to falsification. In W. Damm and H. Hermanns, editors, *International Conference on Computer Aided Verification (CAV)*, volume 4590, pages 468–481. Lecture Notes in Computer Science, Springer-Verlag Heidelberg, Berlin, Germany, 2007.

[36] L. Tan, J. Kim, O. Sokolsky, and I. Lee. Model-based testing and monitoring for hybrid embedded systems. In *Proceedings of IEEE Internation Conference on Information Reuse and Integration (IRI'04)*, 2004.

[37] Eric Thiémard. An algorithm to compute bounds for the star discrepancy. *J. Complexity*, 17(4):850–880, 2001.

[38] Jan Tretmans. A formal approach to conformance testing. In *Proceedings of the IFIP TC6/WG6.1 Sixth International Workshop on Protocol Test systems VI*, pages 257–276, Amsterdam, The Netherlands, 1994. North-Holland Publishing Co.

[39] Jan Tretmans. Testing concurrent systems: A formal approach. In *CONCUR '99: Proceedings of the 10th International Conference on Concurrency Theory*, pages 46–65, London, UK, 1999. Springer-Verlag.

[40] M. van Osch. Hybrid input-output conformance and test generation. In K. Havelund, M. Nunez, G. Rosu, and B. Wolff, editors, *Proceedings of FATES/RV 2006*, Lecture Notes in Computer Science 4262, pages 70–84. Springer, 2006.

[41] X. Wang and F. Hickernell. Randomized halton sequences, 2000.

[42] A. Yershova, L. Jaillet, T. Simeon, and S. LaValle. Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. In *Proc. of IEEE Int. Conf. Robotics and Automation*, 2005.