Test des Systèmes hybrides

Tarik Nahhal

UNIVERSITÉ JOSEPH FOURIER – GRENOBLE 1

T H È S E

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ JOSEPH FOURIER

Spécialité: INFORMATIQUE

Présentée et soutenue publiquement

par

Tarik NAHHAL

le 15 Octobre 2007

Test des Systèmes hybrides

Préparée au Laboratoire **Verimag**
au sein de l'**École Doctorale** ***Mathématiques, Sciences et Technologies de l'Information***

JURY

Thierry Jéron	Président
Emilio Frazzoli	Rapporteur
Thierry Jéron	Rapporteur
Steven M. LaValle	Examinateur
Thao Dang	Directrice de thèse
Oded Maler	Directeur de thèse

UNIVERSITY JOSEPH FOURIER – GRENOBLE 1

T H E S I S

To obtain the grade of

UJF DOCTOR

Speciality: COMPUTER SCIENCE

Presented and Defended in public

by

Tarik NAHHAL

on Octobre, 15th 2007

Model-Based Testing of Hybrid Systems

Prepared in the **Verimag** Laboratory
within the **École Doctorale** ***Mathématiques, Sciences et Technologies de l'Information***

JURY

Thierry Jéron	President
Emilio Frazzoli	Reviewer
Thierry Jéron	Reviewer
Steven M. LaValle	Examinator
Thao Dang	Director
Oded Maler	Director

Dedicace
(à mes parents)

Remerciements

Je tiens tout d'abord à remercier l'ensemble des membres du jury:

Thierry Jéron, *Directeur de Recherche INRIA, pour avoir eu la patience de lire soigneusement ce document, pour ces questions qui m'a fournit en retour et pour sa présidence du jury.*

Emilio Frazzoli, *Professeur Assistant à l'Institut Massachusetts, d'avoir accepté d'être rapporteur de cette thèse.*

Steven M. LaValle, *Professeur Assistant à l'Université Illinois, de m'avoir honoré en examinant ce travail.*

Oded Maler, *Directeur de recherche CNRS, pour son soutien pour ce travail.*

J'exprime ma plus sincère gratitude à Madame **Thao Dang**, *ma directrice de thèse, pour l'intérêt qu'elle a attribué à mon travail de recherche, pour ces critiques toujours constructives et pour les nombreuses corrections qu'elle a apportées à toute ma production écrite.*

Je tiens également à remercier l'ensemble des membres du laboratoire Verimag et spécialement,

Nicolas Halbwachs, *directeur de Verimag, pour m'avoir accueilli dans son laboratoire et de m'avoir donné cette oportinuté de travailler avec des chercheurs de grande qualité.*

Abstract

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. Much effort has been devoted to the development of automatic analysis methods for such systems based on formal verification. Nevertheless, the applicability of these methods is still limited to small size systems due to the complexity of exhaustive analysis. Testing is another validation approach, which can be used for much larger systems and is a standard tool in industry, despite its limitations compared to algorithmic and deductive verification. This thesis is concerned with model-based testing of hybrid systems.

We proposed a formal framework for conformance testing of hybrid systems, which is defined according to the international standard for formal conformance testing (FMCT). Besides the main concepts in the formal framework, we addressed the problem of defining test coverage measures. We proposed two novel coverage measures, which not only are useful as a criterion to evaluate testing quality but also can be used to guide the test generation process in order to produce test cases with good coverage. We then developed a number of coverage-guided test generation algorithms for hybrid systems. These algorithms are based on a combination of the ideas from robotic path planning, equidistribution theory, algorithmic geometry, and numerical simulation. The algorithms have been implemented in a test generation tool for hybrid systems, called **HTG**, which can handle high dimensional systems. The tool has been successfully applied to treat a number of benchmarks in control systems and analog and mixed signal circuits.

Résumé

Les systèmes hybrides, systèmes combinant à la fois une dynamique continue et discrète, s'avèrent être un modèle mathématique utile pour différents phénomènes physiques, technologiques, biologiques ou économiques. Beaucoup d'efforts ont été consacrés à l'élaboration de méthodes automatiques d'analyse pour de tels systèmes, basées sur la vérification formelle. Néanmoins, l'applicabilité de ces méthodes est encore limitée aux systèmes de petite taille en raison de la complexité de l'analyse exhaustive. Le test est une autre approche de validation, qui peut être employée pour des systèmes beaucoup plus grands. En dépit de ses limitations comparées à la vérification algorithmique et déductive, le test reste l'outil standard dans l'industrie.

Nous proposons dans cette thèse une méthodologie formelle pour le test de conformité des systèmes hybrides, qui est définie selon la norme internationale pour le test formel de conformité (FMCT). Ensuite, nous abordons le problème de la définition de mesures de couverture de test. Pour cela, deux mesures de couverture sont proposées, qui sont non seulement utiles comme critère pour évaluer la qualité de test mais peuvent être aussi employées pour guider la génération de test vers une meilleur couverture. Des algorithmes de génération de test guidés par les mesures de couverture sont proposés. Ces algorithmes sont basés sur une combinaison des algorithmes de planification de trajectoires dans la robotique, de la théorie d'équidistribution, de la géométrie algorithmique et de la simulation numérique. L'outil HTG (Hybrid test generation) pour la génération de cas de test pour les systèmes hybrides implémente ces algorithmes, et a été appliqué avec succès pour traiter plusieurs études de cas provenants de différentes domaines (circuits analogiques et mixtes, systèmes de commande, etc.).

Contents

List of Figures

List of Tables

Chapter 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. One typical example is a chemical batch plant where a computer is used to supervise complex sequences of chemical reactions, each of which is modeled as a continuous process. In addition to discontinuities introduced by the computer, most physical processes admit components (e.g. switches) and phenomena (e.g. collision) whose most useful models are discrete. Hybrid system models arise in many applications, such as chemical process control [69], avionics [93], robotics, automobiles [12, 53, 13], manufacturing [83], and most recently in molecular biology [9].

Due to the safety critical features of many such applications, formal analysis is a topic of particular interest. Recently, much effort has been devoted to the development of automatic analysis methods and tools for hybrid systems, based on formal verification. This can be seen in a large number of publications on the topic in the recent proceedings of HSCC *Hybrid Systems: Computation and Control*, a major international conference of the domain. The goal of *formal verification* is to prove that the (designed) system satisfies a property. Due to the complexity and scale of real-life applications, automatic analysis is very desirable. This is a motivation to adopt the algo-

rithmic approach which consists in building software tools that can analyze automatically all the behaviors of a given system. Results in algorithmic verification of hybrid systems resulted in a number of verification tools, such as **Coho**[100] developed at University of British Columbia, **Verdict** [61] developed at University of Dortmund, **CheckMate** [30] developed at University of Carnegie Mellon, **VeriShift** [24] developed at University of Berkeley, **d/dt** [15] developed at VERIMAG, MPT [63] developed at ETH Zurich, HJB toolbox [76] developed at Stanford University. 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 due to the complexity of exhaustive analysis, required by formal verification. It soon became 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, despite its limitations compared to algorithmic and deductive verification. Indeed, like simulation techniques, testing can only reveal an error but does not permit proving the correctness of the system. The success of the testing approaches in industry is perhaps due to the fact that they suffer less from the problem of state explosion. Indeed, the engineer can choose the degree of validation by changing the number of tests. In fact, the larger number of tests are executed, the larger portion of behaviors of the system is validated, and therefore the more confidence in the correctness of the system we gain. This is different from the results of the type 'yes' or 'no' provided by the formal verification methods. On the other hand, testing can be applied to real systems, and not only on their models.

Although testing has been well studied in the context of finite state machines, it has not been much investigated for continuous and hybrid systems. Therefore, a question of great interest is 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 particular test generation from specifications.

Before continuing, we discuss the relation of our work to the current research in testing. Testing is obviously an activity that occupies a major part in the development of industrial applications. The International Organization for Standardization defines the testing standards, for example the language TTCN [54]. In research, testing is also the principal theme of many international conferences and projects. Testing is a vast domain, the approach we adopt in this thesis can be classified as a *model-based approach*, which has been a very active research area recently. In particular, our work addresses the problem of *conformance testing* with a focus on test generation using formal methods. Restricting attention to conformance testing as opposed to testing in general means restricting consideration to functional testing techniques (also referred to as behavioral testing, black box or specification based testing). In functional testing, the systems under test is subject to tests derived from the specifications only; no awareness of the structure of the systems under test is assumed.

Let us now briefly review related work in model-based conformance testing. The development of the first model-based testing frameworks was motivated by digital circuit testing and is based on Mealy machines [68]. More recently, frameworks based on other models, such as finite labeled transition systems, were proposed (see for example [96]). These models are of asynchronous nature which are 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 [43]. Recently, model-based testing has been extended to real-time systems. Timed automata have become a popular model for modeling 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 [81, 88, 28, 81, 5, 21]). Although the current practice of testing, especially in industry, is still empirical and ad-hoc, this situation is changing, and formal testing has become progressively accepted. This is, on 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 Simulink Tester (http://www.reactive-systems.com) for Simulink models, Conformiq Test Generator (http://www.conformiq.com) for UML State-Chart models.

Concerning hybrid systems, model-based testing is still a new research domain. The paper [90] proposed a framework for generating test cases by simulating hybrid models specified using the language CHARON. A probabilistic test generation approach, similar to ours, was presented in [38]. We devote a discussion on these approaches to Chapter 2

In the following we discuss the problematics of hybrid systems testing and explain the main ideas of our approach.

1.1 Problematics and our approach

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 needs to 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 problematics, we take an approach that draws on ideas from both domains, more precisely, the algorithmic analysis methodology from computer science and methods from control theory. To model hybrid systems, we use hybrid automata [48]. 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.

Then, to address the hybrid systems testing problem, we propose an approach that can be summarized by the following elements:

- **Formal framework for conformance testing using the hybrid automaton model**. This testing framework is defined following the international standard "Formal Methods in Conformance Testing" (FMCT) [95]. It includes the definitions of conformance relation, test cases, test executions, etc.

- **Novel test coverage measures**. 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 branch coverage, path coverage) are unsuitable for the behaviors of a hybrid system defined as solutions of some differential equations. We thus propose two novel coverage measures, which on one hand reflect the testing objectives and, on the other hand, can be efficiently computed.

- **Test generation**. We propose a test generation algorithm which is based on the RRT (Rapidly-exploring Random Tree) algorithm [66, 67], a probabilistic motion planning technique in robotics. This RRT algorithm has been successful in finding feasible trajectories in motion

planning. The idea of applying RRTs to the verification of hybrid systems was previously explored, such as in [23, 59, 26]. Their relationship with our work will be discussed in Chapter 4.

- **Method for guiding the test generation process**, based on the above mentioned coverage measures.
- **Development of a prototype test generation tool.**
- **Application of the tool to cases studies**. In particular, besides traditional applications of hybrid systems, we would like to 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.

1.2 Thesis outline

In Chapter 2, we discuss *hybrid automata* [7], the modeling formalism we use for hybrid systems. We then introduce our formal framework for conformance testing, which includes the basic concepts (such as conformance relation, test cases, test executions). The chapter contains the theoretical background necessary for the subsequent developments.

Chapter 3 is concerned with the development of two test coverage measures. We are interested in test coverage measures that describe how well the states visited during a test execution represent the reachable set. The first coverage measure we propose is based on the star discrepancy notion, and the second is based on the δ-cover notion. We then present the methods to compute these coverage measures.

In Chapter 4, we develop an algorithm for generating test cases from hybrid automata. This algorithm is an extension of the RRT (Rapidly-exploring

Random Tree) algorithm [65] to hybrid systems. An important property of this algorithm, namely probabilistic completeness, is then discussed.

In Chapter 5, in order to rapidly achieve a good coverage quality, we develop a method for guiding the test generation algorithm using the coverage measures.

The goal of Chapter 6 is to show how to implement the abstract algorithms introduced in the previous chapters. This can be seen as a concrete realization of these algorithms. We also briefly present the prototype test generation tool that we call **HTG**. The tool provides automatic test generation from hybrid automata.

Chapter 7 is devoted to a number of case studies treated using the tool **HTG**. These case studies come from control applications and analog and mixed signal circuits. To illustrate the performance of our test generation algorithms, some linear systems (with up to 100 continuous variables), which are randomly generated, were treated. The experimental results shows the applicability of our approach to high dimensional systems.

Each of the above chapters also includes a discussion of related work. The concluding chapter summarizes the contributions of the thesis and suggests future research directions.

For the best understanding of this thesis, the reader is encouraged to follow chapter by chapter. In particular, Chapter 2 contains important definitions and notations which are used throughout the thesis.

Chapter 2

Modeling and conformance testing

Hybrid systems are recognized as being appropriate high-level mathematical models for describing physical systems and phenomena with mixed continuous and discrete dynamics. By combining discrete event systems and continuous systems (defined by differential equations), they provide a powerful modeling tool for a very wide variety of complex systems.

As an example of hybrid systems, consider a digital engine controller in a car, which has to interact with the physical processes in the engine as well as with the events generated by the driver. The second example, is the computer controlled systems where a computer (which is fundamentally a finite discrete state machine) is used to control a physical process (which often can be modeled as continuous-time system). The increasing integration of such controllers results in highly complex system involving both continuous and discrete event dynamics.

The structure of this chapter is as follows. The first part gives some mathematical preliminaries in dynamical systems, which are useful for describing the main components of hybrid systems. Next, we introduce hybrid automata, the model we use in this thesis. We also present the important reachability notions, necessary for the development of our testing framework

and briefly review other hybrid models in the literature. The second part of this chapter is devoted to conformance relation for hybrid automata and important elements of conformance testing.

2.1 Preliminaries

We introduce in this section the basic definitions needed for describing behaviors of dynamical systems. Intuitively, a dynamical system describes the evolution of a *state* over *time*. Based on the type of their state, dynamical systems can be classified into: continuous systems, discrete systems and hybrid systems.

Throughout this thesis we will use a time domain $\mathcal{T} = \mathbb{R}^+$.

Definition 1 (Time interval sequence)**.** *A time interval sequence is a sequence of intervals $\{I_0, I_1, ..., I_N\}$, finite or infinite (i.e. N can be ∞), such that*

- *For all $k < N$, $I_k = [\tau_k, \tau'_k]$ where $\tau_k, \tau'_k \in \mathcal{T}$.*
- *For all k, $\tau_k \leq \tau'_k = \tau_{k+1}$ (that is, the right endpoint τ'_k of the interval I_k coincides with the left endpoint τ_{i+1} of the interval I_{k+1}.*
- *If $N < \infty$, then either $I_N = [\tau_N, \tau'_N]$ or $I_N = [\tau_N, \tau'_N)$.*

As we will see later, a time interval sequence will be used to define the time horizon over which the state of a hybrid system evolves.

Definition 2 (Temporal behavior)**.** *A temporal behavior over a topological space S is a partial function $\beta : \mathcal{T} \rightarrow S$ whose domain of definition is an interval $[0, r)$ for some $r \in \mathcal{T} \cup \{\infty\}$.*

We call r the *metric length* of β and β is said *infinite* if $r = \infty$.

Definition 3 (Piecewise-continuous behavior)**.** *A temporal behavior β is piecewise-continuous if it admits a time interval sequence $\{I_0, I_1, ..., I_N\}$ such that for every $k \leq N$, β is continuous on the interval I_k.*

Figure 2.1 shows an example of piecewise-continuous behaviors. Note that a piecewise-constant behavior is a special case of piecewise-continuous behaviors where β is constant on every time interval I_k.

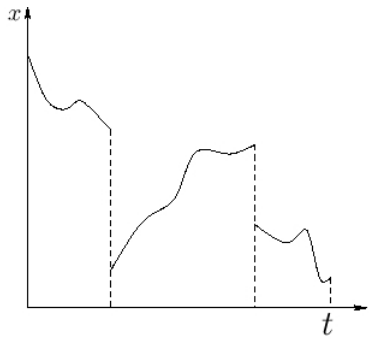

Figure 2.1: A piecewise-continuous behavior.

2.1.1 Discrete systems

A discrete system is a dynamical system where its state takes values in a countable or finite set $Q = \{q1, q2, ...\}$. We often use q to denote the state of such systems. For example, a light switch is a dynamical system the state of which takes on two values, $\{ON, OFF\}$. In the following, we focus on automata without input and output, a simple class of discrete systems which can be used to describe the discrete structure of a hybrid system.

Definition 4 (Automaton). *A finite automaton is $\mathcal{M} = (Q, \delta)$ where*

- *Q is a finite set of states.*
- *$\delta \subseteq \mathcal{Q} \times \mathcal{Q}$ is a transition relation which describes how the system may evolve.*

An example of finite automaton with two states is given in Figure 2.2, and the set of its states is $Q = \{q_1, q_2\}$.

Figure 2.2: A 2-state automaton.

Executions of an automaton. Given an initial state $q \in Q$, an *execution* of $\mathcal{M}$ is a sequence $\sigma : \mathbb{N} \to Q$ such that $\sigma(0) = q$ and for every $k > 0$, $\sigma(k+1) \in \delta(\sigma(k))$.

Notice that the executions of such automata can be *non-deterministic*, since the transition relation indicates a set of possible next states rather than a unique state. Since the definition of executions of an automaton does not involve real metric time, and thus the time lapses between the states in an execution are not specified. Given an execution σ, we can associate with it an infinite number of piecewise-constant behaviors of the form $\beta : \mathcal{T} \to Q$. Figure 2.3 illustrates a piecewise-constant behavior of the automaton in Figure 2.2.

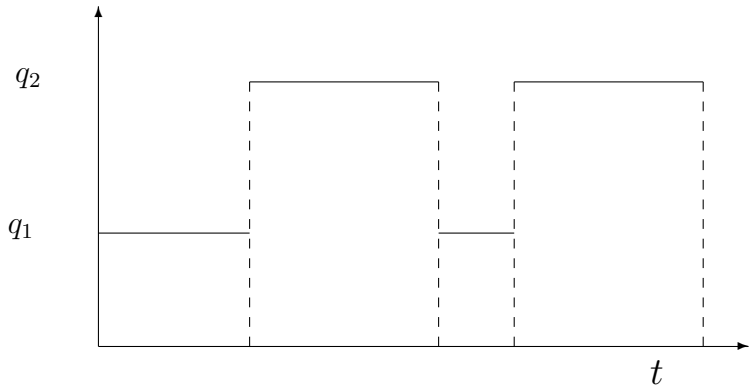

Figure 2.3: A piecewise-constant behavior of the automaton in Figure 2.2

2.1.2 Continuous systems

A continuous system is a dynamical system the state of which take values in $\mathbb{R}^n$ for some $n > 0$, and n is called the *dimension* of the system.

Definition 5 (Continuous system)**.**
A continuous system is $\mathcal{C} = (\mathcal{X}, f)$ where

- $\mathcal{X} = \mathbb{R}^n$ *is the state space.*
- $f : \mathcal{X} \to \mathbb{R}^n$ *is a continuous vector field.*

The behavior of the system is governed by the following differential equation:

$$\dot{x}(t) = f(x(t)) \tag{2.1}$$

where $x \in \mathcal{X}$ is the state of the system.

Trajectories of a continuous system. A trajectory of $\mathcal{C}$ starting from $x \in \mathcal{X}$ is a continuous behavior $\xi_x : \mathcal{T} \to \mathcal{X}$ such that ξ_x is the solution of (2.1) with initial condition $x(0) = x$. Given two states x, $x' \in \mathcal{X}$, we say that x' is *reachable* from x in time $t < \infty$ if $x' = \xi_x(t)$. We denote this by $x \xrightarrow{t} x'$.

We assume that f is globally Lipschitz in x, which guarantees the existence and uniqueness of solutions to (2.1) for every initial condition in $\mathcal{X}$ [52]. The continuous system of Definition 5 is *deterministic* in the sense that from a given state it generates a *unique* trajectory.

In addition to continuous systems without input, for testing purposes, we are interested in continuous systems with input.

Definition 6 (Continuous system with input)**.** *A continuous system with input is $\mathcal{C}_u = \{\mathcal{X}, U, f\}$ where $\mathcal{X} \subseteq \mathbb{R}^n$ is the state space of the system and $U \subset \mathbb{R}^m$ is the input set. The behavior of the system is described by the following differential equation:*

$$\dot{x}(t) = f(x(t), u(t)) \tag{2.2}$$

where $x \in \mathcal{X}$ is the state of the system and $u \in U$ is the input. We assume a set $\mathcal{U}$ of admissible input functions consisting of measurable functions of the form $u : \mathcal{T} \to U$.

Trajectories of a continuous system with input. A trajectory of $\mathcal{C}_u$ starting from a state $x \in \mathcal{X}$ under a given input u: $\mathcal{T} \to U$ is a continuous behavior $\xi_{x,u} : \mathcal{T} \to \mathcal{X}$ such that $\xi_{x,u}(t)$ is the solution of $\dot{x}(t) = f(x(t), u(t))$ with the initial condition $x(0) = x$.

We assume that the vector field f is globally Lipschitz in x and continuous in u. This assumption guarantees the existence and uniqueness of the solution of the differential equation (2.2) for a given input function $u \in \mathcal{U}$ [45, 52].

Notice that the behavior of a continuous system with input is *non-deterministic*. For a given initial condition x, each admissible input function u generates a different solution to (2.2). As a consequence, under all admissible input functions, the system produces a dense set of trajectories.

2.1.3 Hybrid systems

A hybrid system is a dynamical system where part of the state takes values in $\mathbb{R}^n$ while the other part takes values in a finite set. For example, a combination of an analog circuit and a switch is a hybrid system: one part of the state (namely the state of the analog circuit) is continuous, while the other part (namely the state of the switch) is discrete. In the following, we describe hybrid automata [11, 47, 55], the model that we use in this work. The motivation for choosing this model is that this model and a number of its variations have become the standard models, which are largely used by the researchers in the hybrid systems community, for both control design and verification purposes. Another reason is that this model is rich enough to describe a wide class of hybrid systems and, moreover, it provides a framework suitable for the conformance testing problem which we will tackle in the next chapters.

2.2 Hybrid automata

A hybrid automaton is an automaton augmented with continuous variables.

Definition 7 (Hybrid automaton)**.** *A* hybrid automaton *is a tuple* $\mathcal{A} = (\mathcal{X}, Q, F, \mathcal{I}, \mathcal{G}, \mathcal{R})$ *where*

- $\mathcal{X} \subseteq \mathbb{R}^n$ *is the continuous state space. We denote by* $V(\mathcal{A})$ *the set of continuous variables of* $\mathcal{A}$*.*
- Q *is a finite set of locations (or discrete states).*
- E *is a set of transitions.*
- $F = \{F_q \mid q \in Q\}$ *such that for each* $q \in Q$*,* $F_q = (f_q, U_q, W_q)$ *defines a differential equation:*

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

where $w \in W_q \subset \mathbb{R}^l$ *is the parameter vector, and* $u \in \mathcal{U}_q$ *is an admissible input function of the form* $u : \mathbb{R}^+ \to U_q \subset \mathbb{R}^m$*. During the evolution of the system, the parameter vector* w *is constant*[1]*.*

- $\mathcal{I} = \{\mathcal{I}_q \subseteq \mathbb{R}^n \mid q \in Q\}$ *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* $e \in E$*,* $\mathcal{G}_e \subseteq \mathcal{I}_q$*.*
- $\mathcal{R} = \{\mathcal{R}_e \mid e \in E\}$ *is a set of reset maps. For each* $e = (q, q') \in E$*,* $\mathcal{R}_e : \mathcal{G}_q \to 2^{\mathcal{I}_{q'}}$ *defines how* x *may change when* $\mathcal{A}$ *switches from* q *to* q'*.*

A *hybrid state* is a pair (q, x) where $q \in Q$ and $x \in \mathcal{X}$. The hybrid state space is $\mathcal{S} = Q \times \mathcal{X}$. The initial state of the automaton is denoted by (s_{init}, x_{init}). A state (q, x) can change in two ways:

[1] As we will see in Chapter 7 where we treat a number of case studies, the parameters are useful in many practical applications, such as electronic circuits.

- *By continuous evolution*: in location q the continuous evolution of x is governed by the differential equation $\dot{x}(t) = f_q(x(t), u(t), w)$.
- *By discrete evolution*: if there exists a transition $e = (q, q') \in E$ and $x \in \mathcal{G}_e$, then the transition e is enabled, the system can switch from location q to q' and the continuous variables is assigned to a new value $x' \in \mathcal{R}_e(x)$.

It is important to note that this model allows to capture non-determinism in both continuous and discrete dynamics. This non-determinism is useful for describing disturbances from the environment and imprecision in modelling and implementation.

We assume the existence and uniqueness of solutions of these differential equations in the model. Let $\xi_{x,u,w}(\cdot)$ be the solution of the differential equation at location q with the initial condition x, parameter w and under the input u.

Definition 8 (Continuous transition)**.** *Given a positive real number $h > 0$ and an admissible input function $u(\cdot) \in \mathcal{U}_q$, $(q, x) \overset{u(\cdot),h}{\rightarrow} (q, x')$ is a* continuous transition *at the location q of the* hybrid state *from (q, x) to (q, x'), if and only if $x' = \xi_{x,u,w}(h)$ and for all $t \in [0, h] : \xi_{x,u,w}(t) \in \mathcal{I}_q$.*

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. Figure 2.4 illustrates some continuous transitions, each of which results from applying a different input for h time.

Definition 9 (Discrete transition)**.** *Given a transition $e = (q, q') \in E$, $(q, x) \overset{e}{\rightarrow} (q', x')$ is a discrete transition 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 is admissible at (q, x). Unlike *continuous transitions*, *discrete transitions* are instantaneous.

It is customary to represent graphically a hybrid automaton by a directed graph whose vertices represent the locations and edges represent the transitions. We write the staying conditions and the differential equations inside

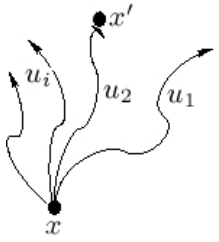

Figure 2.4: A set of continuous transitions starting from x.

the vertices. With the edges we associate the transition guards and the reset maps.

2.2.1 Example: a thermostat

We use the well-known thermostat example in the hybrid system literature [11] to illustrate the above notions. The thermostat consists of a heater and a thermometer which maintain the temperature of the room in some desired temperature range. The lower and upper thresholds of the thermostat system are set at x_m and x_M such that $x_m < x_M$. The heater is maintained on as long as the room temperature is below x_M, and it is turned off whenever the thermometer detects that the temperature reaches x_M. Similarly, the heater remains off if the temperature is above x_m and is switched on whenever the temperature falls to x_m. In practical situations, exact threshold detection is impossible due to sensor imprecision. Also, the reaction time of the on/off switch is usually non-zero. The effect of these inaccuracies is that we cannot guarantee switching exactly at the nominal values x_m and x_M. As we will see, this causes non-determinism in the discrete evolution of the temperature.

Formally we can model the thermostat as a hybrid automaton shown in Figure 2.5. The two operation modes of the thermostat are represented by two location 'on' and 'off'. The on/off switch is modeled by two discrete transitions between the locations. The continuous variable x models the

temperature, which evolves according to the following equations.

- If the thermostat is on, the evolution of the temperature is described by:
$$\dot{x} = f_1(x, u) = -x + 4 + u$$
- When the thermostat is off, the temperature evolves according to the following differential equation:
$$\dot{x} = f_2(x, u) = -x + u$$

The second source of non-determinism comes from the continuous dynamics. The input signal u of the thermostat models the fluctuations in the outside temperature which we cannot control. Figure 2.6 (left) shows this continuous non-determinism. Starting from the initial temperature x_0, the system can generate a "tube" of infinite number of possible trajectories, each of which corresponds to a different input signal u. To capture uncertainty of sensors, we define the first guard condition of the transition from 'on' to 'off' as an interval $[x_M - \epsilon, x_M + \epsilon]$ with $\epsilon > 0$. This means that when the temperature enters this interval, the thermostat can either turn the heater off immediately or keep it on for some time provided that $x \leq x_M + \epsilon$. Figure 2.6 (right) illustrates this kind of non-determinism. Likewise, we define the second guard condition of the transition from 'off' to 'on' as the interval $[x_m - \epsilon, x_m + \epsilon]$. Notice that in the thermostat model, the temperature does not change at the switching points, and the reset maps are thus the identity functions.

Finally we define the two staying conditions of the 'on' and 'off' locations as $x \leq x_M + \epsilon$ and $x \geq x_m - \epsilon$ respectively, meaning that the system can stay at a location while the corresponding staying conditions are satisfied.

2.2.2 Hybrid trajectories

We have defined the syntax of hybrid automata. Their semantics is expressed by the notion of hybrid trajectories that we discuss in the following.

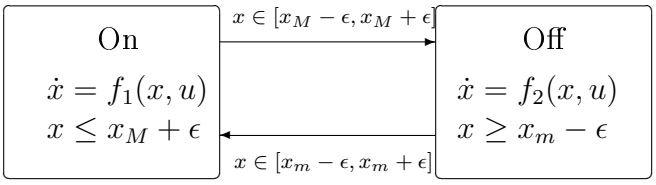

Figure 2.5: Model of the thermostat.

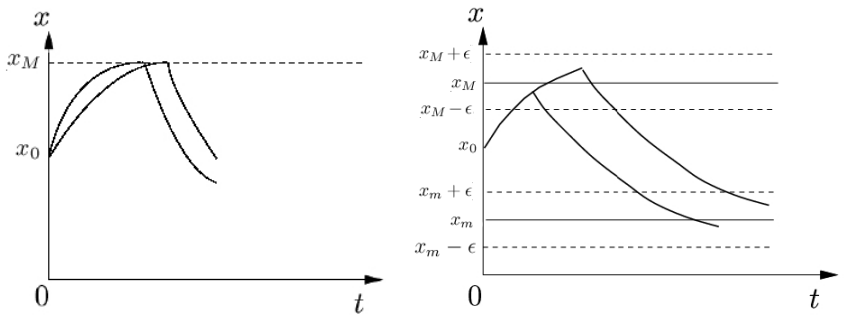

Figure 2.6: Two different behaviors of the temperature starting at x_0.

Throughout this section, we consider a hybrid automaton as in Definition 7, denoted by $\mathcal{A}$.

Definition 10 (Hybrid trajectory). *A hybrid trajectory of the hybrid automaton $\mathcal{A}$ starting from a state (q_0, x_0) is a triple $(\mathcal{I}, q, x)$ consisting of a time interval sequence $\mathcal{I} = \{I_0, I_1, \ldots, I_N\}$ and a piecewise-constant behavior $q_i : I_i \to Q$ and a piecewise-continuous behavior $x_i : I_i \to \mathbb{R}^n$, such that :*

- *Initially, $q_i(0) = q_0$ and $x_i(0) = x_0$*
- *Discrete evolution: for all i, $e = (q_i(\tau_i'), q_{i+1}(\tau_{i+1})) \in E$, $x_i(\tau_i') \in \mathcal{G}_e$ and $x_{i+1}(\tau_{i+1}) \in \mathcal{R}_e(x_i(\tau_i'))$*
- *Continuous evolution: for all i and for all $t \in I_i$, $q_i(t) = q_i(\tau_i) = q$, and $x_i(\cdot)$ is a solution of the differential equation at the location q with $x(t) \in \mathcal{I}_q$ for all $t \in I_i$.*

The number N of the time intervals is called the *logical length* of the trajectory. Figure 2.7 shows another example of hybrid automata. This example has 3 locations q_1, q_2, q_3. From each location q_i we have a discrete transition (q_i, q_j) such that $j \neq i$.

Figure 2.8 shows a projection on the 2-dimensional space $\mathcal{X} \subseteq \mathbb{R}^2$ of some hybrid trajectories of this hybrid automaton. We denote by $\mathcal{G}_{ij}$ the guard of the transition (q_i,q_j). The guards $\mathcal{G}_{12}$, $\mathcal{G}_{13}$, $\mathcal{G}_{23}$ and $\mathcal{G}_{31}$ are the shaded regions. All the locations have the same staying set which is the bounding box. The solid lines show the continuous evolution and the dashed lines show the discrete evolution.

As we can see from the figure, starting from the initial state (q_1, x_0) the system can generate a set of infinite number of hybrid trajectories. For instance, we consider the hybrid trajectory starting from (q_1, x_0). At (q_3, x_3), this trajectory has taken two consecutive discrete transitions: (q_1, q_2) and (q_2, q_3).

We also introduce the notion of *discrete paths* which will be used later in the definition of the distance between two hybrid states. For brevity, we often say 'path' instead of 'discrete path'.

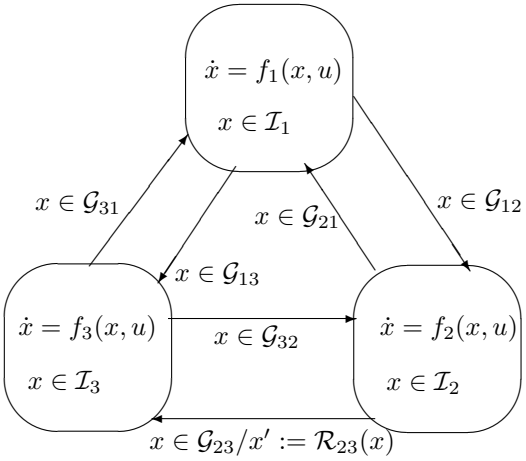

Figure 2.7: A 3-state hybrid automaton.

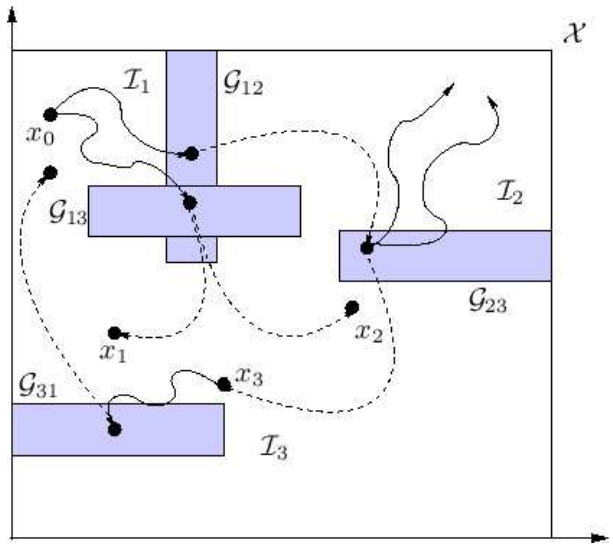

Figure 2.8: A set of hybrid trajectories (of the automaton in Figure 2.7) starting from (q_1, x_0).

Definition 11 (Path). *We define a discrete path as a sequence of consecutive discrete transitions* $\{(q_0, q_1), (q_1, q_2), \cdots, (q_{N-1}, q_N)\}$ *such that for each* $i \in \{0, 1, \ldots, N-1\}$, $(q_i, q_{i+1}) \in E$.

2.2.3 Reachability of hybrid automata

The reachable set from a given set S_0 of initial states by the hybrid automaton $\mathcal{A}$ can be defined as the set of all the states visited by the trajectories starting from states in S_0. For clarity, we first define a reachable state.

Definition 12 (Reachable state). *Let* (q, x) *be a hybrid state of the hybrid automaton* $\mathcal{A}$. *We say that the hybrid state* (q', x') *is reachable from* (q, x) *if there exists a hybrid trajectory* (τ, q, x) *where* $\mathcal{I} = \{I_0, I_1, \ldots, I_N\}$ *such that*

- $q_0 = q$ *and* $x_0(0) = x$, *and*
- $\exists i \in \{1, 2, \ldots, N\} : q_i = q' \ \wedge \ \exists t \in I_i : x_i(t) = x'$.

In other words, (q', x') is reachable from (q, x) if there exists a hybrid trajectory that starts at (q, x) and visits the state (q', x'). We denote this by $(q, x) \longrightarrow (q', x')$.

We now proceed to define a *reachable set*.

Definition 13 (Reachable set).

- *Given a state* (q, x), *the reachable set of the hybrid automaton* $\mathcal{A}$ *from* (q, x) *is*

$$Reach((q, x)) = \{(q', x') \in Q \times \mathcal{X} \mid (q, x) \longrightarrow (q', x')\}.$$

- *Given a set* $\mathcal{F}$ *of states, the reachable set of the hybrid automaton* $\mathcal{A}$ *from* (q, x) *is*

$$Reach(\mathcal{F}) = \{(q', x') \in Q \times \mathcal{X} \mid \exists x \in \mathcal{F} : (q, x) \longrightarrow (q', x')\}.$$

Blocking behaviors

The hybrid automaton model we are considering may have blocking trajectories (which are impossible in physical systems in practice) In this work, we

consider only blocking behaviors which are caused by discrete transitions. A trajectory is blocking if it reaches a hybrid state (q, x) from which it is impossible to continue neither by continuous dynamics (because the current trajectory is going out of the staying set) nor by discrete dynamics (because at the current state none of the transitions is enabled). This means that $x \notin \mathcal{I}_q$ and $x \notin \mathcal{G}_{(q,q')}$ for all $q, q' \in Q$.

To illustrate, again we use the thermostat example (see Figure 2.5) and consider the following situation. Suppose that there is a trajectory that reaches a hybrid state (on, x_s) where $x_s = x_M + \epsilon$, and from this state the thermostat is turned off. However, due to some setup error by a careless user, the threshold $x_m - \epsilon > x_M + \epsilon$, and the trajectory at (on, x_s) cannot continue further.

An automaton is *non-blocking* if from every (q, x) such that $x \in \mathcal{I}_q$ there are no blocking trajectories.

Zeno behaviors

A Zeno behavior is a piecewise-continuous behavior having an infinite logical length and a bounded metric length. In other words, such a behavior can make an infinite number of transitions in a bounded time interval. For example of the thermostat model, if the thresholds x_m and x_M are chosen such that the intervals $[x_m - \epsilon, x_m + \epsilon]$ and $[x_M - \epsilon, x_M + \epsilon]$ overlap, then from the points in the intersection of these intervals the thermostat can make an infinite number of switchings between the two modes in finite time.

An automaton is *non-Zeno* if it does not admit *Zeno* behaviors.

Properties of blocking and Zeno behaviors and conditions for their existence are important topics in hybrid systems reseach, and the article [37] and reference therein can be used for further reading.

We emphasize that in the rest of the thesis, we assume that *the hybrid automata that we treat are non-Zeno and non-blocking.*

2.2.4 Other hybrid models

Before continuing, we briefly review some other hybrid models.

A class of hybrid models, mostly developed by computer scientists, can be seen as an extension of traditional finite-state automata with progressively more complex continuous dynamics. Timed automata [10] can be viewed as a very restricted class of hybrid automata in which the derivative of all continuous variables is 1. Models with more complex continuous dynamics are phase-transition system [73], multirate timed automata [6], piecewise-constant derivative systems PCD [14], integration graphs [58], and rectangular hybrid automata [85] In these models, essentially, the guard and staying sets are polyhedra and the vector fields are constant in every location. A reason for focusing on these classes of hybrid automata is that the verification problem for them are solvable or semi-solvable, which allows to develop efficient tools, e.g. Kronos [102] and Uppaal [64] for timed automata, HyTech [49] and PHAVer [41] for linear hybrid automata[2] (hybrid automata where the continuous dynamics are described by constant derivative inclusions of the form $A\dot{x} \leq b$). In [6] it was shown that the reachability problem for general hybrid automata is undecidable, i.e. there is no general algorithm for any hybrid automaton.

For models with more complex continuous dynamics described by ordinary differential equations, exact analysis is not possible, and approximations have been used. The research along this line has led to a number of methods and tools, such as Coho [100], CheckMate [30], d/dt [15], VeriShift [24], HYSDEL [94], MPT [63], HJB toolbox [76].

On the other hand, other hybrid models were developed by control theorists in order to address their control design problems. Special cases of the hybrid automata considered here include switched systems [77], complementarity systems [97], mixed logic dynamic systems [46]. In addition, we can mention some model which are more general than the hybrid automata considered:

[2]Linear hybrid automata should not be confused with hybrid automata where continuous dynamics are described by linear differential equations.

impulse differential inclusions [16] (which allow more general differential inclusions), General Hybrid Dynamical Systems [25] (where the continuous state can take values in manifolds), and hybrid input/output automata [72], which, among other things, allow infinite dimensional continuous state.

2.3 Conformance testing

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 emerging international standard "Formal Methods in Conformance Testing" (FMCT) [95] provides a framework on how to perform conformance testing, which includes terminology, abstract concepts,such as conformance, test cases, test execution, test generation, and the requirements on these concepts. A testing approach which 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 spirit of model-based conformance testing, we are interested in developing a conformance testing framework for continuous and hybrid systems, using the hybrid automaton model.

The rest of this section is structured as follows. In the first part of this section, we define the main concepts of conformance testing and, in particular, 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 traces of an implementation or, more generally, a system under test (SUT) and a specification. The specification is formal and, as mentioned earlier, it is modeled by a hybrid automaton. The conformance will be defined as a relation $\approx \subseteq \Xi \times HA$ where Ξ is a set of systems under test of interest, and HA is a set of hybrid automata modeling the specifications of interest. Notice that the systems under test are physical systems, but it can

be assumed that all the systems under test in Ξ can be described by a class of formal model, which is a set HA_s of hybrid automata. It is important to note that we assume that a model for each system under test in Ξ exists 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 $\approx$ between the models of the systems under test and the specifications, that is $\approx \subseteq HA_s \times HA$. Note that here we use the same notation $\approx$ for the relation between the real SUT and the specification and that between the model of the SUT and the specification.

A system under test $S_{ut} \in \Xi$ is said to *conform* to a specification $\mathcal{A} \in HA$ if and only if the model $\mathcal{A}_s \in HA_s$ of SUT is related to $\mathcal{A}$ by $\approx$, that is, $\mathcal{A}_s \approx \mathcal{A}$.

The system under test SUT 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 (see Figure 2.9). It emits the control inputs to the system under test and measures the observation sequences in order to produce a verdict $\boldsymbol{v} \in \{P, F, I\}$ where P means 'pass' (the observed trace is allowed by the specification), F means 'fail' (the observed trace is not allowed by the specification), and I means 'inconclusive' (neither a 'pass' nor a 'fail' verdict can be assigned). We discuss the problem of how to perform test executions and derive verdicts in the end of this section. We continue by giving a detailed description of conformance relation.

As mentioned earlier, let the specification be modeled as a hybrid automaton $\mathcal{A}$ and the system under test SUT by another hybrid automaton $\mathcal{A}_s$. In the following, for brevity, when the context is clear, we often say 'the system under test' to mean the automaton $\mathcal{A}_s$ (which is a model of the real system under test). To define the conformance relation, we need the notions

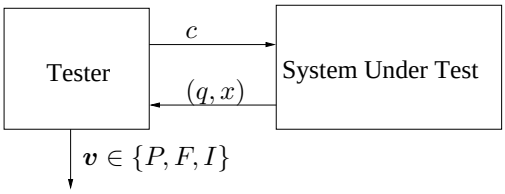

Figure 2.9: Test architecture.

of *inputs and observations*.

2.3.1 Inputs

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 inputs.

Continuous inputs

All the continuous inputs of the system are assumed to be controllable. Since we want to implement the tester as a computer program, we are interested in piecewise-constant continuous input functions (since 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 system continues with the continuous dynamics at 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 inputs

The discrete transitions are partitioned into controllable and uncontrollable discrete transitions. Those which are controllable correspond to discrete control actions, and the others 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 latter case, it can also control the values assigned to the continuous variables by the associated reset map. For simplicity of explanation, we will not consider non-determinism caused by the reset maps. Hence, we denote a discrete control action by the corresponding transition, such as (q, q').

In this work, we make the following two assumptions about the inputs:

- The discrete transitions are instantaneous, which means that their executions do not take time.

- Continuous control action 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, i.e. until the end of that continuous control action. This assumption is not restrictive, from a modeling point of view. As we shall see later, 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.

- When two or more discrete disturbance actions are enabled at the same time, the order of their executions is non-deterministic.

Definition 14 (Admissible input sequence)**.** *For a state (q, x), a sequence of input actions $\omega = \iota_0, \iota_1, \ldots, \iota_k$ is admissible at (q, x) if the following conditions are satisfied:*

- *ι_0 is admissible at (q, x),*

- *for each $i = 1, \ldots, k$, let (q_i, x_i) be the state such that $(q_{i-1}, x_{i-1}) \overset{\iota_{i-1}}{\rightarrow} (q_i, x_i)$, then ι_i is admissible at (q_i, x_i).*

The sequence $(q_0, x_0), \ldots, (q_k, x_k)$ is called the trace *starting at (q, x) under ω and is denoted by $\tau((q, x), \omega)$.*

Recall that $(q_{i-1}, x_{i-1}) \overset{\iota_{i-1}}{\rightarrow} (q_i, x_i)$ means that (q_i, x_i) is reached after applying the input ι_{i-1} to the state (q_{i-1}, x_{i-1}). We use the notation $(q, x) \overset{\omega}{\rightarrow} (q', x')$ to indicate that (q', x') is reachable from (q, x) after ω.

Example. We examine again the thermostat example, introduced in the previous section. We now modify the meaning of the input u in the differential equations as follows: u models a continuous control input rather than a disturbance from the environment. At some state $s = (q, x)$ a sequence of two input actions $\omega = \iota_0, \iota_1$ where $\iota_0 = (\bar{u}, h)$, $\iota_2 =$(**on**, **off**). Both of these inputs are control actions. We suppose that $q =$**on** and the temperature $x < x_M + \epsilon$. Thus, the input action ι_0 is admissible at (q, x) if during h time, under the input $u(t) = \bar{u}$, the temperature x does not reach $x_M + \epsilon$, that is the staying condition of the location ***on*** is still satisfied, then we say that the input action ι_0 is admissible at (q, x). After ι_0, the new state of the thermostat is $s' = (\mathbf{on}, x')$ with $x' > x$. At the state s',

- if $x' < x_M - \epsilon$, then the transition from the location **on** to **off** is not enabled, which means that the discrete control action $\iota_2 =$(**on**, **off**) is not admissible.

- if $x' \geq x_M - \epsilon$, then the control action $\iota_2 =$(**on**, **off**) is admissible at s' and the sequence ω is said to be admissible at $s = (q, x)$.

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. In the following, we define a set of all possible traces that can be generated by applying a sequence of control actions. Again, since we are only interested in non-blocking behaviors, we first 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 $\Sigma(c,(q,x))$ the set of all such disturbance input sequences.

We now extend this notion to a sequence of control actions. To do so, we need to know which disturbance inputs are admissible after each control action. This means that we need to know the successors after each control action. We first consider a sequence of two control actions $\omega_c = c_0\, c_1$.

When taking into account the occurrences of all admissible disturbance input sequences after the control action c_0, the set of all possible successors of (q,x) after applying c_0 is:

$$\Upsilon(c_0,(q,x)) = \{(q',x') \mid \exists \sigma \in \Sigma(c_0,(q,x)) : (q,x) \xrightarrow{\sigma} (q',x')\}.$$

We recall that $(q,x) \xrightarrow{\sigma} (q',x')$ indicates that (q',x') is reachable from (q,x) after the input sequence σ. 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 Σ for the *set of all disturbance input sequences that can occur after the control action sequence* $\omega_c = c_0\, c_1$:

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

Therefore, we can now determine the set of all input sequences (containing both control and disturbance actions) that can possibly occur when we apply the control sequence $\omega_c = c_0\, c_1$. We denote this set by $\Sigma(\omega_c,(q,x))$ and call it the *set of all admissible input sequences corresponding to the execution of* $\omega_c = c_0\, c_1$ starting at (q,x). This set can be defined as follows:

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

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

Definition 15 (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 *is:*

$$Tr((q,x),\omega) = \{\tau((q,x),\sigma) \mid \sigma \in \Sigma(\omega,(q,x))\}.$$

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 an initial state (q_{init}, x_{init}). In the definition of the conformance relation between a system under test $\mathcal{A}_s$ and a specification $\mathcal{A}$, we will assume that

- All the controllable inputs of the specification $\mathcal{A}$ are also the controllable inputs of the system under test $\mathcal{A}_s$.
- 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.

2.3.2 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.
- $V_o(\mathcal{A}) \subseteq V_o(\mathcal{A}_s)$, which means that an observable continuous variable of $\mathcal{A}$ is also an observable variable of $\mathcal{A}_s$.

[] Since not all the continuous variables are observable, we need the following projection operator. The projection of a continuous state x of $\mathcal{A}$ on the

observable variables $V_o(\mathcal{A})$ is denoted by $\pi(x, V_o(\mathcal{A}))$. We can also extend this operator to a set of states.

Definition 16 (Observation)**.** *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.

The projection can be then defined for a trace and a set of traces as follows. Given a trace: $\tau = (q_0, x_0), (q_1, x_1), (q_2, x_2) \ldots$, the projection of τ on $V_o(\mathcal{A})$ is

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

Definition 17 (Observation sequence)**.** *Let ω be an admissible control action sequence starting at an initial state (q_{init}, x_{init}) of $\mathcal{A}$. The set of* observation sequences *associated with ω is $S_{\mathcal{O}}(\mathcal{A}, \omega) = \{\pi(\tau, V_o(\mathcal{A})) \mid \tau \in Tr((q_{init}, x_{init}), \omega)\}$.*

The extension of the framework to a set of initial states will be discussed later.

2.3.3 Conformance relation

We are now ready to define the conformance relation between the system under test SUT and the specification.

Definition 18 (Conformance)**.** *The system under test $\mathcal{A}_s$ is conforms to the specification $\mathcal{A}$, denoted by $\mathcal{A} \approx \mathcal{A}_s$, iff*

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

Note that we have assumed earlier that $S_{\mathcal{C}}(\mathcal{A}) \subseteq S_{\mathcal{C}}(\mathcal{A}_s)$, that is a control action sequence which is admissible for $\mathcal{A}$ is also admissible for $\mathcal{A}_s$.

2.3.4 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. The tester performs experiments on $\mathcal{A}_s$ in order to study the relation between $\mathcal{A}$ and $\mathcal{A}_s$. A physical *test execution* can be described by the following points:

- The tester applies a test ζ to the system under test S_{ut}.
- 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 procedure is denoted by $exec(\zeta, S_{ut})$ which leads to an observation sequence, or a set of observation sequence if multiple runs of ζ are possible in case non-determinism is present). In the following, we focus on the case where each test execution involves a single run of a test case.

It is clear that the above test execution process uses a number of implicit assumptions:

- Observation measurements take zero time, and in addition, no measurement error is considered.
- The tester is able to realize exactly the continuous input functions (which is often not possible in practice due to actuator imprecision).

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: $\boldsymbol{v} : \Omega \rightarrow \{$ **pass**, **fail**, **inconclusive** $\}$. Note that an observation sequence must cause a unique verdict.

The observation sequences in Ω are grouped into three disjoint sets: the set O_p of observation sequences that cause a 'pass' verdict, the set O_f that cause a 'fail' verdict, and the set O_i that cause an 'inconclusive' verdict. Therefore, saying "The system under test S_{ut} passes the test ζ" formally means $\boldsymbol{v}(exec(\zeta, S_{ut}))$ =**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 given specification $\mathcal{A} \in HA$:

$$S_{ut} \approx \mathcal{A} \Longleftrightarrow S_{ut} \text{ passes } T_s \tag{2.3}$$

This means that a complete test suite can distinguish exactly between all conforming and non-conforming systems. In practice, it is generally impossible to fulfil this requirement, which often involves executing an infinite test suite. Another 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 (2.3).

After defining all the important concepts, it now remains to tackle the problem of generating test suites from a specification model. In particular, we want the test suites to satisfy the soundness requirement.

A hybrid automaton might have an infinite number of infinite traces; however, the tester can only perform a finite number of test cases in finite time. Therefore, we need to select a finite portion of the input space of $\mathcal{A}$ and test the conformance of $\mathcal{A}_s$ 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 hybrid automaton to satisfy this coverage criterion.

2.4 Related work

In Chapter 1, we have presented a review of the model-based conformance testing approaches for discrete and timed systems. Here we only discuss related works along this line for hybrid systems. The paper [90] proposed a framework for generating test cases by simulating hybrid models specified using the language CHARON [8]. 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 the paper 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 this work is to apply tests to models, as an alternative verification method.

In [59], 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 of the systems.

To our knowledge, there is no other work in developing a formal framework for conformance testing that follows the standards of FMCT (Formal Methods in Conformance Testing) as closely as the framework we propose.

Chapter 3

Test coverage

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 which 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 arising question is thus how to choose appropriate input signals so as 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. In this work, we treat continuous and hybrid systems that operate in a metric space (typically $\mathbb{R}^n$) and where there is not much inspiration coming from the syntax to the coverage issue. 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. In this work, with a focus on reachability and in particular safety properties, we are interested in defining a test coverage measure which describes how well the states visited by a test suite represent the reachable set. One way to do so is to look at how well the states are equidistributed over the reachable set. However, the reachable set is unknown, we can only consider the distribution of the visited states over the state space (which can be thought of as the potential reachable space).

We propose two new coverage measures. One is based on the star discrepancy notion from statistics, and the other on the δ-cover notion.

The chapter is organized in two parts, each of which is devoted to one of the two coverage measures. We also show how to compute each measure and discuss its properties. Before concluding, we discuss some related work. Most of the results presented in this chapter was published in [79].

3.1 Star discrepancy coverage

Geometric discrepancy is often called the theory of 'irregularities of distribution', which is typically measured with respect to some underlying geometric space.

Some arising questions are: In which way can n points be equidistributed

with respect to an underlying geometric space? How can we measure the uniformity of a given set of points?

When the underlying space is a box, this leads to the so-called star-discrepancy. The popularity of this measure is perhaps related to the well-known Koksma-Hlawka inequality used in the quasi-Monte Carlo techniques (see for example [19]). It shows that for multivariate integration, a low star discrepancy sequences are desirable as well as the algorithms providing such sequence.

3.1.1 Star discrepancy

In the following, we briefly review the star discrepancy notion and show why it is suitable for describing a test coverage measure for continuous and hybrid systems. We need to introduce first some notations.

Definition 19 (Sub-box). *Given a n-dimensional box* $\mathcal{B} = [l_1, L_1] \times \ldots \times [l_n, L_n]$*, we define a sub-box of* $\mathcal{B}$ *as a box of the following form* $J = \prod_{i=1}^{n} [l_i, \beta_i]$ *with* $\beta_i \in [l_i, L_i]$.

Figure 3.1 shows an example to illustrate the definition. We also use the notation J_x to indicate a sub-box with x as its top-right vertex. Let $\Gamma_{\mathcal{B}}$ be the set of all such sub-boxes of $\mathcal{B}$.

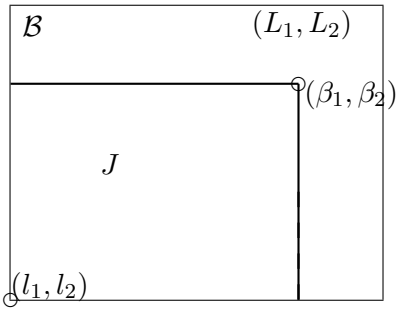

Figure 3.1: illustration of a sub-box J.

Definition 20 (Box partition). *We define a box partition of a box $\mathcal{B}$ 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 with each other. Each such box is called an* elementary box.

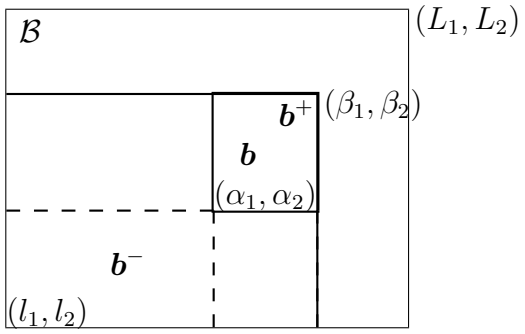

Figure 3.2: Illustration of the boxes $\boldsymbol{b}^-$ and $\boldsymbol{b}^+$.

Let Π be a box partition of the box $\mathcal{B}$. With every elementary box $\boldsymbol{b}^i = [\alpha_1, \beta_1] \times \ldots \times [\alpha_n, \beta_n]$ in Π, we associate a pair of sub-boxes $\boldsymbol{b}^+ = [l_1, \beta_1] \times \ldots \times [l_n, \beta_n]$ and $\boldsymbol{b}^- = [l_1, \alpha_1] \times \ldots \times [l_n, \alpha_n]$. Figure 3.2 gives an illustration of these boxes.

As mentioned earlier, the star discrepancy is a measure for the irregularity of set of points in a box. Let P be a set of k points within the box $\mathcal{B}$, which we call the bounding box, and J be a sub-box of $\mathcal{B}$.

The local discrepancy of the set P with respect to the sub-box J is defined as follows:

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

Definition 21 (Star discrepancy). *The star discrepancy of a point set P with respect to the box $\mathcal{B}$ is defined as:*

$$D^*(P, \mathcal{B}) = sup_{J \in \Gamma_{\mathcal{B}}} D(P, J). \tag{3.1}$$

It is not hard to prove the following property of the star discrepancy [92].

Proposition 1. *The star discrepancy of a point set P with respect to a box $\mathcal{B}$ satisfies*

$$0 < D^*(P, \mathcal{B}) \leq 1.$$

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

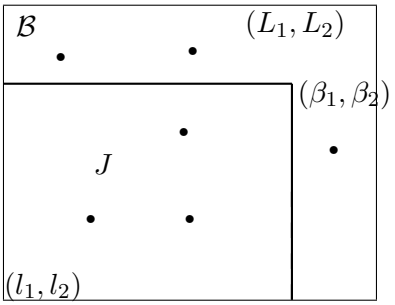

Figure 3.3: the star discrepancy notion.

We now illustrate the star discrepancy notion with a simple example shown in Figure 3.3. In this example, we have a set P of 6 points inside a two-dimensional box $\mathcal{B} = [l_1, L_1] \times [l_2, L_2]$. For any sub-box J with its bottom-left vertex at (l_1, l_2) and its bottom-right vertex inside $\mathcal{B}$ we can determine:

- the number $A(P, J)$ of points of the set P that lie inside the sub-box J. In the example of Figure 3.3, $A(P, J) = 3$.
- the volume $vol(J)$ of the rectangle J.

The local discrepancy associated with the sub-box J is the absolute difference $|A(P, J)/6 - vol(J)/vol(\mathcal{B})|$. The star discrepancy $D^*(P, \mathcal{B})$ is equal to the largest local discrepancy of all such sub-boxes J of $\mathcal{B}$. If the set P is well equidistributed over $\mathcal{B}$, then the quantities $\frac{A(P,J)}{6}$ and $\frac{vol(J)}{vol(\mathcal{B})}$ are not much different for any sub-box J of $\mathcal{B}$. Thus, the star discrepancy of a set of points P is a measure that tells how far the distribution of P is from this perfect situation.

Example. To shown an intuitive meaning of the star discrepancy, we use some sequences of 100 points inside the 2-dimensional unit cube with the bottom-left vertex at the origin. The first example is the Faure sequence [39], a well-known low-discrepancy sequence (see Figure 3.4).

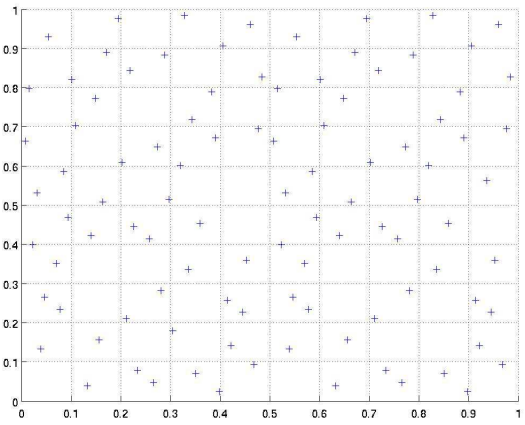

Figure 3.4: Faure sequence of 100 points. Its star discrepancy value is 0.048.

As we can observe from Figure 3.4, this set of points 'covers well' the cube, 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 [99], which is also a well-known low discrepancy sequence. Figure3.5 shows 100 points of this sequence within the same unit cube. The value of the star discrepancy of the Halton sequence is about 0.050, meaning that the Faure sequence is slightly more equidistributed than the Halton sequence.

The star discrepancy values of these two sequences are close and indeed visually, it is hard to see from the figures which one of these two sequences are 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 3.6) from which we can observe

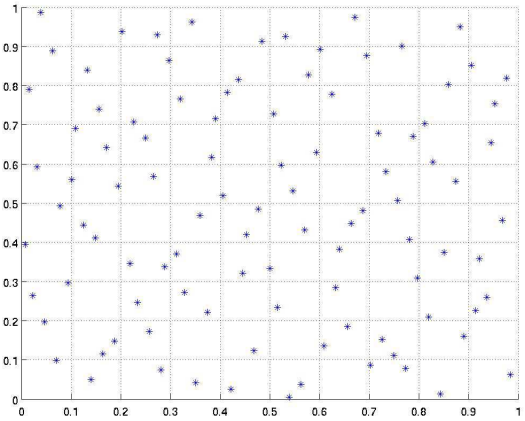

Figure 3.5: Halton sequence of 100 points. The star discrepancy value is 0.05.

that this sequence is less equidistributed over the box. This is confirmed by its star discrepancy value 0.1, which is higher than that of the Faure and Halton sequences.

Clearly, the star discrepancy is a meaningful measure that can characterize the uniformity quality of point set distributions. This makes the star discrepancy suitable to be a test coverage measures for continuous and hybrid systems.

3.1.2 Test coverage measure

As a test coverage measure, we use the star discrepancy to describe how well the states visited by a test suite are equidistributed over the state space of the system.

Since the hybrid state space can be seen as a set of continuous spaces, we define first the test coverage for each location and the hybrid test coverage can then be defined as the average of the coverages of all the locations.

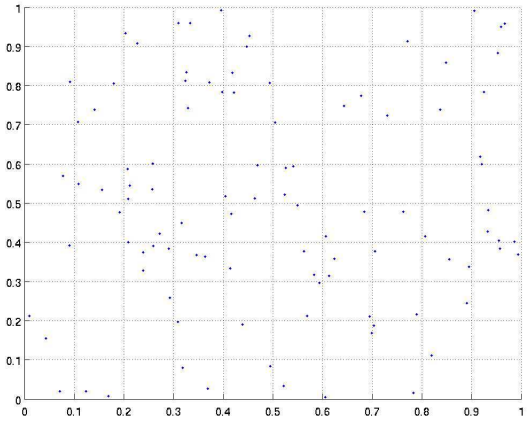

Figure 3.6: C pseudo-random sequence of 100 points. The star discrepancy value is 0.1.

We denote a set of states of a hybrid automaton $\mathcal{A}$ as:

$$\mathcal{P} = \{(q, x) \mid q \in Q \ \wedge \ x \in \mathcal{I}_q\} \tag{3.2}$$

where Q is the set of locations of $\mathcal{A}$ and $\mathcal{I}_q$ is the staying condition associated with the location q.

We assume that for every location q, the staying set $\mathcal{I}_q$ is a box that we often call the *bounding box* of the location q. If a set $\mathcal{I}_q$ is not a box, we can take the smallest oriented box that encloses it. Recall that the i^{th} coordinate of the point x is denoted by x_i.

Definition 22 (Coverage at a location)**.** *For a given set $\mathcal{P}$ of states visited by a test suite $\mathcal{TS}$, the coverage of the test suite $\mathcal{TS}$ at location q is defined as:*

$$Cov(\mathcal{P}, q) = 1 - D^*(\mathcal{P}_q, \mathcal{I}_q) \tag{3.3}$$

A large value of $Cov(\mathcal{P}, q)$ indicate a 'good' equidistribution of the set $\mathcal{P}_q$ over the box space $\mathcal{I}_q$.

Definition 23 (Hybrid test coverage)**.** *The hybrid test coverage of the set $\mathcal{P}$ of states visited by a test suite $\mathcal{TS}$ is defined as:*

$$Cov(\mathcal{P}) = \frac{1}{||Q||} \sum_{q \in Q} Cov(\mathcal{P}, q) \tag{3.4}$$

where $||Q||$ is the number of locations of the hybrid automaton $\mathcal{A}$.

To compute this coverage measure, we need to compute the star discrepancy in each location. The exact computation of the star discrepancy is not easy (see for example [80, 33]). Many theoretical results for one-dimensional point sets are not generalizable to higher dimensions, and among the fastest algorithms we can mention the one proposed in [33] of time complexity $\mathcal{O}(k^{1+d/2})$. In this work, in order to be able to handle high dimensional systems, we do not try to compute the star discrepancy but approximate it by estimating a lower and upper bound. These bounds are then used to decide whether the box $\boldsymbol{b}$ has been 'well explored' or it needs to be explored more. This estimation is based on the method published by Eric Thiémard [91, 92], which we call the *Thiemard* method.

3.1.3 Star discrepancy estimation

In the following we present the *Thiemard* method and its main properties. Although in these results the box $\mathcal{B}$ is $[0,1]^n$, we have extended to the general case where $\mathcal{B}$ can be any full-dimensional box.

Thiemard method

Given a bounding box $\mathcal{B}$, let Π be a box partition of $\mathcal{B}$ and P a set of points inside $\mathcal{B}$.

For each box $\boldsymbol{b} \in \Pi$ we define two quantities:

$$\mu_c(\boldsymbol{b}) = \frac{A(P, \boldsymbol{b}^+)}{k} - \frac{vol(\boldsymbol{b}^-)}{vol(\mathcal{B})}, \tag{3.5}$$

$$\mu_o(\boldsymbol{b}) = \frac{vol(\boldsymbol{b}^+)}{vol(\mathcal{B})} - \frac{A(P, \boldsymbol{b}^-)}{k} \tag{3.6}$$

We then denote by μ_m the larger value of μ_c and μ_o:

$$\mu_m(\boldsymbol{b}) = \max\{\mu_c(\boldsymbol{b}), \mu_o(\boldsymbol{b})\} \tag{3.7}$$

We also denote:

$$c(\boldsymbol{b}) = \max\{|\frac{A(P, \boldsymbol{b}^-)}{k} - \frac{vol(\boldsymbol{b}^-)}{vol(\mathcal{B})}|, \ |\frac{A(P, \boldsymbol{b}^+)}{k} - \frac{vol(\boldsymbol{b}^+)}{vol(\mathcal{B})}|\} \tag{3.8}$$

The following theorem [92] shows a lower bound and an upper bound of the star discrepancy.

Theorem 1 (Bounds on the star discrepancy [92]). *Given a box partition Π of the box $\mathcal{B}$, 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 $B(P, \Pi)$ and $C(P, \Pi)$ are:*

$$B(P, \Pi) = \max_{\boldsymbol{b} \in \Pi} \mu_m(\boldsymbol{b}) \tag{3.9}$$

$$C(P, \Pi) = \max_{\boldsymbol{b} \in \Pi} c(\boldsymbol{b}) \tag{3.10}$$

Proof. We observe that for a given $\boldsymbol{b} \in \Pi$ and a point $x \in \boldsymbol{b}$ we have

$$D(P, J_x) \leq max\{\frac{A(P, b^+)}{k} - \frac{vol(b^-)}{vol(\mathcal{B})}, \frac{vol(b^+)}{vol(\mathcal{B})} - \frac{A(P, b^-)}{k}\}$$

Thus, we obtain

$$D^*(P, \mathcal{B}) \leq \max_{\boldsymbol{b} \in \Pi}\{\mu_m(\boldsymbol{b})\}.$$

Finally, it is easy to see that $C(P, \Pi) = \max_{\boldsymbol{b} \in \Pi}\{c(\boldsymbol{b})\}$ is a lower bound of the star discrepancy. □

$$D(P, J_x) \leq Max_{\boldsymbol{b} \in \Pi} \ \ max\{\frac{\#Pts(P, b^+)}{k} - \frac{vol(b^-)}{vol(\mathcal{B})}, \frac{vol(b^+)}{vol(\mathcal{B})} - \frac{\#Pts(P, b^-)}{k}\}$$

3.1.4 Estimation error

In the next section we discuss the imprecision of this approximation, which is defined as the difference between the upper and lower bounds. Indeed, a bound of this difference, which depends on the box partition Π, can be derived.

Definition 24 (Weight of a box)**.** *We define the weight $W(\boldsymbol{b})$ of an elementary box $\boldsymbol{b} = [\alpha_1, \beta_1] \times \ldots \times [\alpha_n, \beta_n] \in \Pi$ as the difference in volume of the two sub-boxes $\boldsymbol{b}^+ = [l_1, \beta_1] \times \ldots \times [l_n, \beta_n]$ and $\boldsymbol{b}^- = [l_1, \alpha_1] \times \ldots \times [l_n, \alpha_n]$ divided by the total volume of $\mathcal{B}$*

$$W(\boldsymbol{b}) = \frac{vol(\boldsymbol{b}^+) - vol(\boldsymbol{b}^-)}{vol(\mathcal{B})} \tag{3.11}$$

and the weight $W(\Pi)$ of the partition Π is

$$W(\Pi) = \max_{\boldsymbol{b} \in \Pi} W(\boldsymbol{b}) \tag{3.12}$$

$\leq max_{\boldsymbol{b} \in \Pi} Weight(\boldsymbol{b})$ where $Weight(\boldsymbol{b}) = \frac{vol(\boldsymbol{b}^+) - vol(\boldsymbol{b}^-)}{vol(\mathcal{B})}$

In [92] the author proved that the interval of the star discrepancy bounds cannot be larger than $W(\Pi)$.

Theorem 2 (Error bound [92])**.** *Let $B(P, \Pi)$ and $C(P, \Pi)$ be the upper and lower bounds in (3.9) and (3.10). Then,*

$$B(P, \Pi) - C(P, \Pi) \leq W(\Pi) \tag{3.13}$$

Thus, to achieve a good estimation, one needs to find a partition Π such that the weight $W(\Pi)$ is small. Since the bound depends only on the partition Π, so the $W(\Pi)$ value can be a user-defined parameter to specify an error tolerance.

This theorem expresses the worst estimation error of the star discrepancy using *Thiemard* method. Since $W(\Pi)$ is independent of the point

set, one may choose a small tolerance error δ and construct a partition Π with $W(\Pi) \leq \delta$ to guarantee an interval of width at most δ containing the exact value of the star discrepancy.

The algorithm [92] for constructing a low-weight box partition generates a number of boxes exponential in the dimension. This makes the method very time and memory consuming. Indeed, as we will show later, in our coverage-guided test generation algorithm, we do not try to estimate the star discrepancy precisely in each iteration, but to use the information from an incremental estimation of the coverage in order to guide the exploration.

In the following we show an interesting property of the estimation. This property involves the local error at each elementary box. This property will be used in the method of guiding the test generation based on the coverage.

Lemma 2 (Local error)**.** *For any box* $\boldsymbol{b}$ *in the finite partition* Π *of the bounding box* $\mathcal{B}$*, we have*

$$\mu_m(\boldsymbol{b}) - c(\boldsymbol{b}) \leq W(\boldsymbol{b}) \tag{3.14}$$

Proof. First we observe that for each $\boldsymbol{b} \in \Pi$, we have

$$\mu_o(\boldsymbol{b}) = W(\boldsymbol{b}) + \frac{vol(\boldsymbol{b}^-)}{vol(\mathcal{B})} - \frac{A(P, \boldsymbol{b}^-)}{k} \tag{3.15}$$

Combining the above with the definition of the local discrepancy $D(P, \boldsymbol{b}^-)$ of $\boldsymbol{b}^-$ in (3.30) and after the straightforward calculations we obtain

$$\mu_o(\boldsymbol{b}) \leq W(\boldsymbol{b}) + D(P, \boldsymbol{b}^-) \tag{3.16}$$

On the other hand, we have:

$$\mu_c(\boldsymbol{b}) = W(\boldsymbol{b}) - \frac{vol(\boldsymbol{b}^+)}{vol(\mathcal{B})} + \frac{A(P, \boldsymbol{b}^+)}{k}$$

Likewise, combining the above with the definition of the local discrepancy $D(P, \boldsymbol{b}^+)$ of $\boldsymbol{b}^+$ in (3.30), we obtain

$$\mu_c(\boldsymbol{b}) \leq W(\boldsymbol{b}) + D(P, \boldsymbol{b}^+) \tag{3.17}$$

Since the lower bound of the local discrepancy w.r.t. the box $\boldsymbol{b}$

$$c(\boldsymbol{b}) = \max\{D(P, \boldsymbol{b}^+), D(P, \boldsymbol{b}^-)\} \tag{3.18}$$

Thus we obtain

$$\mu_m(\boldsymbol{b}) - c(\boldsymbol{b}) \leq W(\boldsymbol{b}) \tag{3.19}$$

□

3.1.5 Coverage estimation

Given a location $q \in Q$, we can approximate the coverage at q of the set $\mathcal{P}$ of states as follows:

$$Cov(\mathcal{P}, q) = 1 - \frac{B(\mathcal{P}_q, \Pi_q) + C(\mathcal{P}_q, \Pi_q)}{2} \tag{3.20}$$

where $B(\mathcal{P}_q, \Pi_q)$ and $C(\mathcal{P}_q, \Pi_q))$ are respectively the values of the upper and lower bounds of the star discrepancy of the set $\mathcal{P}_q$ w.r.t. the staying set $\mathcal{I}_q$.

We impose the following condition on the partitions of the locations. For each pair of locations $q, p \in Q$ such that $q \neq p$, we have :

$$W(\Pi_q) = W(\Pi_p) \tag{3.21}$$

Without this condition, the coverage estimation at two different locations might give misleading information about which location is better explored because their star discrepancy values are not estimated with the same precision. Since we want to use the coverage measure to guide the test case generation, a large difference between the partition weights might wrongly deviate the exploration.

3.2 Coverage measure using δ-cover

In this section we introduce another coverage measure, which is defined using a comparison with a reference point set that forms a δ-cover over the

region of interest. In order to explain the motivation of introducing this alternative coverage measure, we first illustrate the difference between the star discrepancy and δ-cover via some concrete examples.

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 3.7).

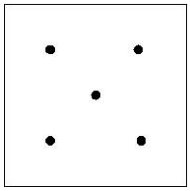

Figure 3.7: The point set P.

The estimation of the value of the star discrepancy of the set P w.r.t. $\mathcal{B}$, using the *Thiemard* method, 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 consider a non-empty set Q of points inside $\mathcal{B}$ that does not contain any points of P. The estimation results show that the star discrepancy $D^*(P \cup Q, \mathcal{B})$ of the union $P \cup Q$ might be larger or smaller that that of P.

For example, for the following set Q, shown in Figure 3.8,

$$Q = \{(0.06, 0.06), (0.12, 0.12), (0.06, 0.12), (0.12, 0.06)$$

the value of $D^*(P \cup Q, \mathcal{B}) \simeq 0.493$. Therefore, adding this set Q in P increases the star discrepancy of P, 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. Furthermore, geometrically speaking, the set $P \cup Q$ "covers more space" than the set P. This means that the star discrepancy allows to compare the space coverage quality between the point sets of the same cardinality; it however may give misleading

comparison between the sets of different cardinalities. In the following, we introduce another test coverage measure, which indeed does not suffer from this problem. It is defined using the δ-cover notion that we explain in the following.

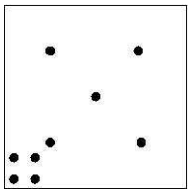

Figure 3.8: The set $P \cup Q$.

3.2.1 δ-cover notion

We first briefly review the notion of δ-covers. For a more detailed description of this notion, the reader is referred to [74, 34, 51]. We remark that, like the star discrepancy, the δ-cover notion was also introduced in the study of the worst case error of multivariate integration [35, 50, 74].

Again, we consider a box $\mathcal{B} \subseteq \mathbb{R}^n$, which we call the *bounding box*, and a real number $\delta > 0$. The bottom-left vertex of $\mathcal{B}$ is denoted by $bot(\mathcal{B})$. Given two points $x, y \in \mathbb{R}^n$, we define a partial order between them as follows: $x \preceq z$ iff for all $i \in \{1, .., n\} : \ x_i \leq z_i$

Definition 25 (δ-covering box)**.** *A pair (x, z) of points in $\mathcal{B}$ forms a* **δ-covering pair** *if the following conditions are satisfied:*

1. $x \preceq z$
2. $\dfrac{vol(J_z) - vol(J_x)}{vol(\mathcal{B})} \leq \delta.$

The box with x and z as its bottom-left and top-right vertices, denoted by $\square(x, z)$, is called a **δ-covering box**.

We recall that in the above definition, J_x is a box that has the same bottom-left vertex as $\mathcal{B}$, and x is its top-right vertex; $vol(J_x)$ is the volume of J_x.

Definition 26 (δ-cover). *A finite set P of points inside $\mathcal{B}$ is called a* **δ-cover** *of $\mathcal{B}$ if for every $y \in \mathcal{B}$ there exist $x, z \in P \cup bot(\mathcal{B})$ such that $y \in \square(x, z)$ and $\square(x, z)$ is a δ-covering box. The number of points in P is called the cardinality of the cover.*

We recall that $\square(x, z)$ denotes the box with x as its bottom left vertex and z its top right vertex.

Intuitively, for any point y in the box $\mathcal{B}$, we can find a δ-covering pair in P such that the corresponding covering box contains y. It is easy to see that if a point set P is a δ-cover of a box $\mathcal{B}$, then the set resulting from adding points in P is also a δ-cover of $\mathcal{B}$.

Example. Let m be a strictly positive natural number. We consider a regular grid of size $1/m$ over the box $\mathcal{B} = [0, 1]^n$. The resulting set of all grid points $\{1/m, 2/m, \cdots, 1\}^n$ is a δ-cover of $\mathcal{B}$, where $m = \lceil n/\delta \rceil$. The cardinality of this set of point is m^n. It is important to note that this δ-cover is not minimal in the sense that there are other δ-coverswith smaller cardinalities.

This δ-cover can be used as a reference point set to measure the coverage of the point sets generated by the test suites. In the following we introduce a notion to characterize how much the distributions of two point sets, which we call the disparity between two point sets.

3.2.2 Disparity between two point sets

The notion of disparity between two point sets that we develop here is inspired by the star discrepancy definition. Indeed, by definition, the star discrepancy of a set P w.r.t. the box $\mathcal{B}$ can be seen as a comparison between P and an 'ideal' infinite set of points distributed all over $\mathcal{B}$.

Given two sets P and Q of points inside $\mathcal{B}$, the *disparity* between P and Q is defined as follows. Let J be a sub-box of $\mathcal{B}$, we define the local disparity between P and Q with respect to the sub-box J as:

$$\gamma(P, Q, J) = |\frac{A(P, J)}{||P||} - \frac{A(Q, J)}{||Q||}| \tag{3.22}$$

where $A(P, J)$ is the number of points of P inside J.

Definition 27 (Disparity)**.** *The disparity between P and Q with respect to the bounding box $\mathcal{B}$ is defined as:*

$$\gamma^*(P, Q, \mathcal{B}) = sup_{J \in \Gamma_{\mathcal{B}}} \gamma(P, Q, J) \tag{3.23}$$

The following result is a direct consequence of the above definition.

Proposition 3. *The disparity between P and Q with respect to the bounding box $\mathcal{B}$ satisfies*

$$0 < \gamma^*(P, Q, \mathcal{B}) \leq 1.$$

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'.

The exact computation of the disparity is as hard as the exact computation of the star discrepancy, which is due to the infinite number of the sub-boxes. In the section 3.2.4 we propose an estimation by the lower and upper bound for this new measure.

Example. In order to illustrate our notion of disparity , let consider again the three sequences of points. As mentioned earlier, the star discrepancy values of the Faure and the Halton sequences are close to each other. In Figure 3.9 we show the two sequences in the same frame to show illustrate their disparity. The disparity between the two sets of this example is 0.06, indicating that the Halton and the Faure sequences have a similar distribution.

Now, we compare the Faure sequence with the random sequence generated by the C library. Figure 3.10 displays the two sequences, each of which has

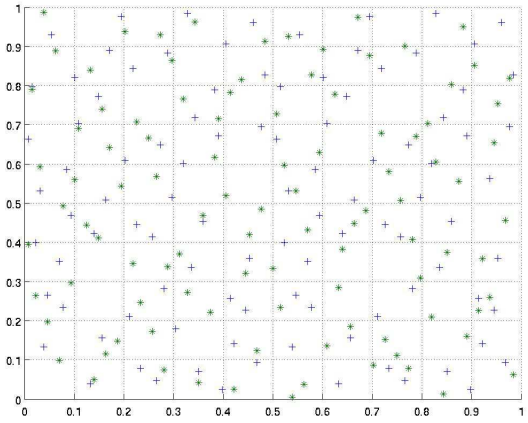

Figure 3.9: Disparity between the Faure and the Halton sequences is 0.06. The points in the Faure sequence are drawn using the + sign and those in the Halton sequence using the $*$ sign.

100 points. We have seen that 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 shown in Figure 3.11 a comparison between the Faure sequence and a set of 100 points concentratred in some small rectangle of the bounding box. We call the latter P_b. The disparity between them (which is 0.54) is very large.

This leads us to the idea of using the disparity to express the coverage of a point set by comparing it with a reference point set which is some δ-cover.

3.2.3 δ-coverage measure

We now combine the above δ-cover and the disparity notions to define a new coverage measure, which we call δ-coverage . Again, we consider a bounding

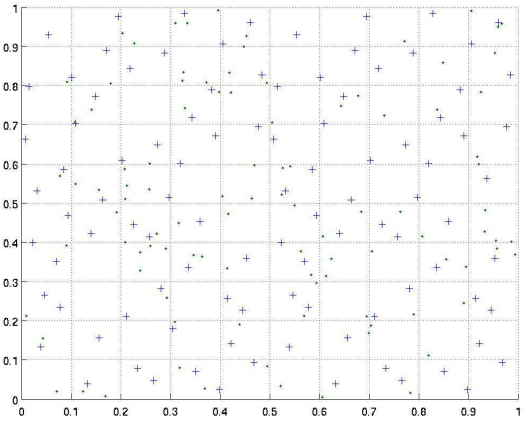

Figure 3.10: Disparity between the Faure and C pseudo-random sequences is 0.12. The points in the Faure sequence are drawn using the + sign and those in the C pseudo-random sequence using the $*$ sign.

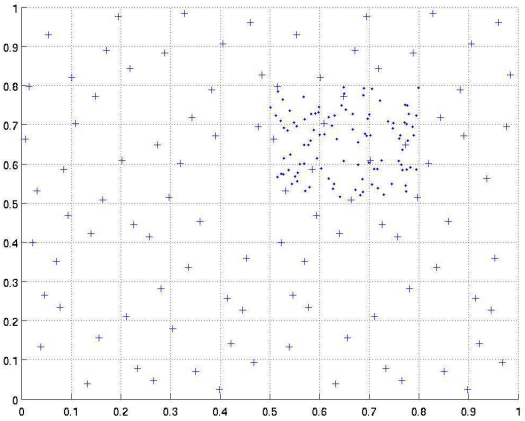

Figure 3.11: Disparity between Faure and a set P_c. The points in the Faure sequence are drawn using the + sign.

box $\mathcal{B}$.

Definition 28 ((d,δ)-cover)**.** *Let $\delta > 0$ and $d \geq 0$ are two positive real numbers. We say a set P of k points inside $\mathcal{B}$ is a (d,δ)-cover of $\mathcal{B}$ if there exists a set Q of points in $\mathcal{B}$ such that:*

- *Q is a δ-cover of $\mathcal{B}$, and*
- *$\gamma^*(P,Q,\mathcal{B}) = d$.*

To define the δ-coverage of a point set P, we first define a set Q, which is a δ-cover of $\mathcal{B}$ with a reasonable value of δ. We call Q the *reference set.* Then, we compare P with Q by their disparity.

Definition 29 (δ-coverage)**.** *Given a set P of k points inside $\mathcal{B}$ and a reference δ-cover P_r in $\mathcal{B}$, let d be the disparity between the set P and P_r, that is $\gamma^*(P,P_r,\mathcal{B}) = d$. Then, we define the δ-coverage of P as $d+\delta$.*

It is important to note that in this definition, we first fix some δ-cover P_r with δ reflecting a desired coverage and then measure how much the point set P under study differs from P_r.

Coverage measure as a termination criterion

As we have seen earlier, the star discrepancy allows to compare the equidistribution quality of the point sets with the same cardinality. Nevertheless, when the point sets have different cardinalities, this measure does not reflect the coverage we want to express. Indeed, adding new states may increase the star discrepancy, and thus the star discrepancy does not indicate how close the current coverage value is to a desired one. Therefore, we do not use the star discrepancy coverage measure to decide when the test generation algorithm can terminate. To this end, the δ-coverage is suitable, since it is monotonic in the cardinality of the point sets. Therefore, we can fix a desired δ-coverage value and stop the algorithm whenever the coverage of the generated states reaches this value.

Example. To illustrate this notion of coverage, we consider first a reference set P_r of 287 points within a bounding box defined by the 2-dimensional unit cube $\mathcal{B}$ with the bottom-left vertex at the origin. This reference set (see Figure 3.12) is a δ-cover of $\mathcal{B}$ with $\delta = 0.12$.

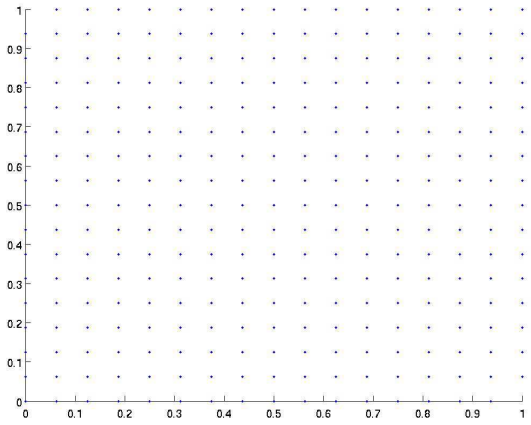

Figure 3.12: A δ-cover P_r of the unit cube with $\delta = 0.12$.

Now we illustrate the δ-coverage of two sets of points P_1 and P_2 using the reference set P_r. The first set P_1 is constructed from 287 points randomly chosen within the bounding space $\mathcal{B}$. Figure 3.13 shows this set together with the reference set P_r. We note that the two sets have the same number of points and the disparity between them is 0.13. Thus combining the values of δ and the disparity value, the value of δ-coverage of P_1 is $0.12 + 0.13 = 0.25$.

Next, we consider a second set P_2 of 287 points randomly sampled within the sub space S of $\mathcal{B}$, defined as flows: $S = \{x \in \mathcal{B} \mid x_1 \geq x_2\}$. Figure 3.14 shows this set together with the reference set P_r. The disparity between P_2 and P_r is 0.34, and the value of δ-coverage of P_2 is $0.12 + 0.34 = 0.46$. Using the δ-coverage values we can say that the first set P_1 has a better coverage of the box $\mathcal{B}$ than the second set P_2.

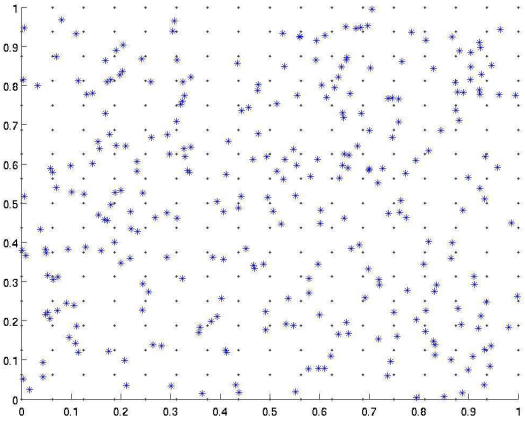

Figure 3.13: The set P_1 (drawn using $*$) with the δ-cover reference P_r. The δ-coverage of P_1 is 0.25.

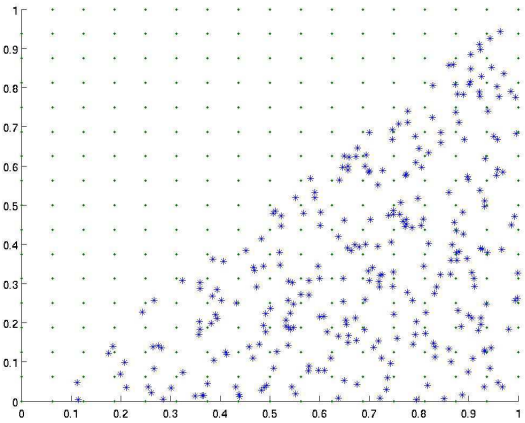

Figure 3.14: The set P_2 (drawn using $*$) with the δ-cover reference P_r. The δ-coverage of P_2 is 0.46.

3.2.4 Disparity estimation

We proceed to present a method for estimating the disparity . This method is similar to the method for estimating the star discrepancy. 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

$$\mu'_m(\boldsymbol{b}) = \max\{\mu'_c(\boldsymbol{b}), \mu'_o(\boldsymbol{b})\} \tag{3.24}$$

where $\mu'_c(\boldsymbol{b}) = \dfrac{A(P,\boldsymbol{b}^+)}{||P||} - \dfrac{A(Q,\boldsymbol{b}^-)}{||Q||}$, $\mu'_o(\boldsymbol{b}) = \dfrac{A(Q,\boldsymbol{b}^+)}{||Q||} - \dfrac{A(P,\boldsymbol{b}^-)}{||P||}$ and

$$c'(\boldsymbol{b}) = \max\{|\frac{A(P,\boldsymbol{b}^-)}{||P||} - \frac{A(Q,\boldsymbol{b}^-)}{||Q||}|,\ |\frac{A(P,\boldsymbol{b}^+)}{||P||} - \frac{A(Q,\boldsymbol{b}^+)}{||Q||}|\} \tag{3.25}$$

Theorem 3. [Upper and lower bounds] *An upper bound $B'(P,Q,\Pi)$ and a lower bound $C'(P,Q,\Pi)$ of the disparity between P and Q is:*

$$B'(P,Q,\Pi) = \max_{\boldsymbol{b}\in\Pi}\{\mu'_m(\boldsymbol{b})\} \tag{3.26}$$

$$C'(P,Q,\Pi) = \max_{\boldsymbol{b}\in\Pi}\{c'(\boldsymbol{b})\} \tag{3.27}$$

Proof. Let Π be a box partition of $\mathcal{B}$. We observe that for a given $\boldsymbol{b} \in \Pi$ and a point $x \in \boldsymbol{b}$ we have

$$D(P,Q,J_x) \leq \max\{\frac{A(P,\boldsymbol{b}^+)}{||P||} - \frac{A(Q,\boldsymbol{b}^-)}{||Q||}, \frac{A(Q,\boldsymbol{b}^+)}{||Q||} - \frac{A(P,\boldsymbol{b}^-)}{||P||}\}$$

Thus, we obtain

$$D^*(P,Q,\mathcal{B}) \leq \max_{\boldsymbol{b}\in\Pi}\{\mu'_m(\boldsymbol{b})\}$$

Similarly, we can prove that $C'(P,Q,\Pi) = \max_{\boldsymbol{b}\in\Pi}\{c'(\boldsymbol{b})\}$ is a lower bound of the disparity . □

3.2.5 Estimation error

We now give a bound on the error in the above estimation. We define the $\mathcal{W}$-zone, denoted by $\mathcal{W}(\boldsymbol{b})$, of an elementary box $\boldsymbol{b} = [\alpha_1,\beta_1]\times\ldots\times[\alpha_n,\beta_n] \in \Pi$, 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 $\boldsymbol{b}^- = [l_1, \alpha_1] \times \ldots \times [l_n, \alpha_n]$. The following lemma shows a local error bound at each elementary box in

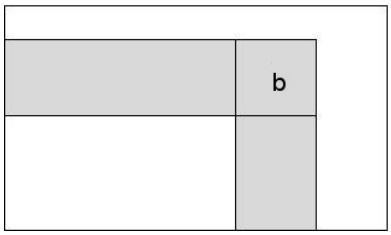

Figure 3.15: Illustration of the $\mathcal{W}$-zone of the box $\boldsymbol{b}$.

Π.

Lemma 4. [Box estimation error] *For any box $\boldsymbol{b}$ in the finite partition Π of the bounding box $\mathcal{B}$, we have:*

$$\mu'_m(\boldsymbol{b}) - c'(\boldsymbol{b}) \leq \max\{\frac{A(P, \mathcal{W}(\boldsymbol{b}))}{||P||}, \frac{A(Q, \mathcal{W}(\boldsymbol{b}))}{||Q||}\} \tag{3.28}$$

where $A(P, \mathcal{W}(\boldsymbol{b}))$ is the number of points of P within the zone $\mathcal{W}(\boldsymbol{b})$.

Proof. First we observe that for each $\boldsymbol{b} \in \Pi$, we can rewrite

$$\mu'_c(\boldsymbol{b}) = \frac{A(P, \boldsymbol{b}^-)}{||P||} - \frac{A(Q, \boldsymbol{b}^-)}{||Q||} + \frac{A(P, \mathcal{W}(\boldsymbol{b}))}{||P||} \tag{3.29}$$

and

$$\mu'_o(\boldsymbol{b}) = \frac{A(Q, \boldsymbol{b}^-)}{||Q||} - \frac{A(P, \boldsymbol{b}^-)}{||P||} + \frac{A(Q, \mathcal{W}(\boldsymbol{b}))}{||Q||} \tag{3.30}$$

On the other hand, we have

$$\max\{\frac{A(Q, \boldsymbol{b}^-)}{||Q||} - \frac{A(P, \boldsymbol{b}^-)}{||P||}, \frac{A(P, \boldsymbol{b}^-)}{||P||} - \frac{A(Q, \boldsymbol{b}^-)}{||Q||}\} \leq D(P, Q, \boldsymbol{b}^-) \tag{3.31}$$

Thus we obtain

$$\mu'_m(\boldsymbol{b}) - c'(\boldsymbol{b}) \leq \max\{\frac{A(P, \mathcal{W}(\boldsymbol{b}))}{||P||}, \frac{A(Q, \mathcal{W}(\boldsymbol{b}))}{||Q||}\} \tag{3.32}$$

□

Using Lemma 4 we obtain the following bound on the error of the estimation, defined as the difference between the upper and lower bounds.

Theorem 4 (Error bounds)**.** *Let $B'(P,Q,\Pi)$ and $C'(P,Q,\Pi)$ be the upper and lower bounds of the disparity between P and Q. Then,*

$$B'(P,Q,\Pi) - C'(P,Q,\Pi) \leq \max_{\boldsymbol{b}\in\Pi} \max\{\frac{A(P,\mathcal{W}(\boldsymbol{b}))}{||P||}, \frac{A(Q,\mathcal{W}(\boldsymbol{b}))}{||Q||}\}. \quad (3.33)$$

3.3 Summary and related work

In this chapter we have defined two test coverage measures for our testing framework. Both of them are based on the equidistribution degree of a set of states over the state space. The first measure is defined using the star discrepancy notion and the second using the δ-cover notion.

Concerning related work in test coverage for continuous and hybrid systems, we mention the work published in [38]. In this paper, the authors proposed a coverage measure based on a discretized version of dispersion, since the dispersion 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. 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 state 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 in [38] this coverage measure is used only as a termination criterion. In our work we use the coverage measures as a termination criterion as well as to guide the exploration.

Chapter 4

Test generation

In this chapter we develop an automatic test generation algorithm for hybrid systems from specifications described by hybrid automata.

Finding the test cases that corresponds to the trajectories violating the conformance relation in our testing problem can be seen as path planning problem in robotic, where the goal is the find a feasible trajectories in some environment that take robot from an initial point to a goal point. In this work, we propose a test case generator algorithm based on the well-known algorithm RRT (Rapidly-exploring Random Trees) [65], a probabilistic path and motion planning technique with a good space-covering property.

This chapter will be organized as follows. The first part will be devoted to RRT algorithm. In the second part we extend the RRT algorithm to treat hybrid systems. Most of the results presented in this chapter were published in [32, 78, 79].

4.1 Rapidly-Exploring Random Trees (RRTs)

In path and motion planning, Rapidly-exploring Random Trees (RRTs) are a successful technique to find trajectories connecting a given set of points in an environment with obstacles. In this section we recall the main con-

cepts of the classical RRT algorithm. By 'classical RRT algorithm', we mean the basic algorithm without problem-specific optimization. More concretely, we describe an abstract algorithm that summarizes the essential ideas of the RRT algorithm for a continuous system in a bounded metric space. We often say 'the RRT algorithm' to refer to this abstract algorithm. For a thorough description of RRTs and their applications in various domains, the reader is referred to a survey [65] and numerous articles in the RRT literature.

Before presenting the abstract RRT algorithm, we formally state the problem it addresses.

4.1.1 Problem formulation

A variety of problems in path and motion planning can be solved by the RRT algorithms. In the following we focus only on a class of problems which are close to the our test generation problem. The setting of these problems is defined in terms of a dynamical system with the following elements:

1. A topological space $\mathcal{X}$ which is called the *state space*.

2. A set of differential constraints of the form:

$$f(t, x(t), \dot{x}(t), u(t)) = 0 \tag{4.1}$$

   where x denotes the state of the system and u the input.

3. A set $\mathcal{U}$ of admissible input functions of the form $\mathbb{R}^+ \to U$ where U is the set of input values.

4. A function $C : \mathcal{X} \to \{true, false\}$ determines for each state in $\mathcal{X}$ whether it satisfies some algebraic constraints. This function thus corresponds to a subset of the state space $\mathcal{X}$. We call this subset the *free space* and denote it by $\mathcal{X}_{free}$.

5. A metric $\rho : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$ which defines the distance between two states in $\mathcal{X}$.

6. An initial state x_{init} and a goal state x_{goal}.

Problem 1. *Find a feasible trajectory connecting the initial state x_{init} to the goal state x_{goal}.*

In other words, we want to compute a trajectory $\phi(t)$ which is a solution of the equation (4.1) with the initial condition $\phi(0) = x_{init}$ such that

- $\exists T > 0 : \phi(T) = x_{goal}$ and,
- the trajectory segment in the time interval $[0, T]$ satisfies the algebraic constraints, that is, $\forall \tau \in [0, T] : \phi(\tau) \in \mathcal{X}_{free}$.

Computing such a trajectory $\phi(t)$ also means finding an input function that generates this trajectory. It is also possible to specify a set of initial and goal states, and the problem can thus be stated as to construct a network of feasible trajectories.

Variants of the above formulation can be derived to capture a number of well-known problems in path and motion planning. For example, most basic *path planning* problems can be seen as a search in the state space $\mathcal{X}$ of a path connecting the initial state to the goal state. In these problems, a state is often called a configuration consisting of the position and orientation of a number of objects in a 2D or $3D$ environment with obstacles. The free space is thus the part of the state space where these objects do not collide with the obstacles. In these problems, the algebraic constraints arising from the obstacles are often complex, and an explicit description of the free space may not be available. In a basic *holonomic path planning* problem, the differential constraints are given as a set of inputs which are the constraints on the velocities of the objects of the form $||\dot{x}|| \leq c$. In a *nonholonomic path planning* problem, there are additionally integrable constraints on the velocities which can be described as in the equation (4.1). Note that from these constraints (often called the equation of motion), we can derive a *successor function* specifying the state resulting from applying an input u to a state x over a time interval $[0, h]$. This successor function can also be provided as an

input of the problem in form of an incremental simulator that implements some numerical integration scheme:

$$x' = \sigma(f, x, u, h). \tag{4.2}$$

In a *kinodynamic path planning* problem, the differential constraints involves both the velocities and the accelerations; therefore, the accelerations are part of the state vector.

4.1.2 Abstract algorithm

Essentially, the RRT algorithm constructs a tree $\mathcal{T}$, often called an RRT tree, the vertices of which are states in $\mathcal{X}_{free}$. The root of the tree corresponds to the initial state x_{init}. Each directed edge of the tree $\mathcal{T}$ is labeled with an input selected from a set $\mathcal{U}$ of admissible input functions. Hence, an edge labeled with u that connects from the vertex x to the vertex x' means that the state x' is reached from x by applying the input u over a duration of h time, called a *time step*. We assume that an incremental simulator, as in (4.2), is provided.

The construction of the tree $\mathcal{T}$ is shown in Algorithm 1 where K is the user-defined maximal number of iterations. We use the notation $\mathcal{T}^k$ to refer to the tree constructed at the k^{th} iteration and $Vertices(\mathcal{T}^k)$ denotes the set of vertices of the RRT tree $\mathcal{T}^k$. By abuse of notation, we often use the notation of a vertex to refer to the state at that vertex. The root of the tree is initialized with the initial state x_{init}. Each iteration of the algorithm consists of the following steps:

1. The function RANDOM_STATE samples a *goal state* x_{goal} from the free space $\mathcal{X}_{free}$. We call it a goal state because it indicates the direction towards which the tree is expected to evolve.

2. Then, in the function NEAREST_NEIGHBOR, a *neighbor state* is determined as a state x_{near} in the tree that is closest to the goal state x_{goal}, according to the pre-defined metric ρ. This neighbor state is used as the starting state for the next expansion of the tree.

Algorithm 1 The abstract RRT algorithm

procedure RRT_TREE_GENERATION(x_{init}, x_{goal}, K)
  $\mathcal{T}^0.init(x_{init})$
  $k = 1$
  repeat
    $x_{goal} =$ RANDOM_STATE($\mathcal{X}_{free}$);
    $x_{near} =$ NEAREST_NEIGHBOR($\mathcal{T}^k, x_{goal}$);
    $(u, x_{new}) =$ NEW_STATE(x_{near}, x_{goal}, h);
    $\mathcal{T}^k$.ADD_VERTEX(x_{new});
    $\mathcal{T}^k$.ADD_EDGE(x_{near}, x_{new}, u);
    $k++$;
  until $(k \geq k_{max}) \ \vee \ x_{goal} \in Vertices(\mathcal{T}^k)$
end procedure

3. The function NEW_STATE creates a trajectory from x_{near} towards the goal state x_{goal} by applying an admissible input function u for some time h. This input function can be determined such that the resulting successor state x_{new} is as close to x_{goal} as possible, or it can be chosen at random. It is important to emphasize that the procedure NEW_STATE needs to assure that the trajectory segment from x_{near} to x_{new} stays in the free space.

4. Finally, a new vertex corresponding to x_{new} is added in the tree $\mathcal{T}$ with an edge from x_{near} to x_{new} labeled with u.

Figure 4.1 illustrates one iteration of the algorithm. The algorithm terminates after k_{max} iterations or until the goal state is reached.

Note that in most RRT algorithms, the sampling distribution of x_{goal} is *uniform* over the free space $\mathcal{X}_{free}$, and the metric ρ is the *Euclidian distance*. Different implementations of the functions in the abstract algorithm and different choices of the metric and of the successor functions in the problem formulation result in different versions of the RRT algorithm.

In the above algorithm we assume a fixed time step h in the computation

of the new states. It is worth emphasizing that this time step is not the internal time step used by numerical integrators. In addition, detection of discrete events is also done during the integration.

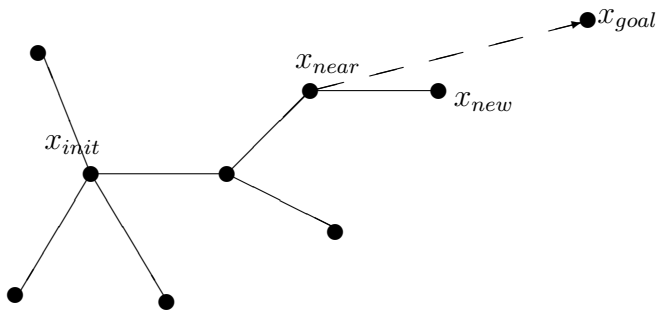

Figure 4.1: The RRT iteration.

Example. We illustrate the algorithm with a two-dimensional example where the state space is the square $\mathcal{X} = [0, 100] \times [0, 100]$. In this example, there are no obstacles and hence $\mathcal{X} = \mathcal{X}_{free}$. The differential constraints are described by: $f(t, x) = u(t)$ where $\forall t \geq 0 : u(t) \in U = \{u \in \mathbb{R}^2 \mid ||u|| \leq 1\}$. The initial state is $x_{init} = (50, 50)$ and the time step $h = 1$. Figure 5.1 shows that the RRT tree rapidly explores all directions of the state space $\mathcal{X}$.

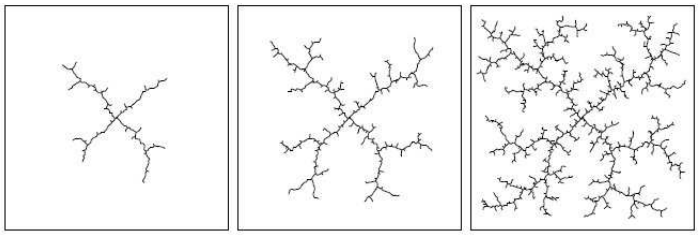

Figure 4.2: Exploration by the RRT algorithm [66].

4.1.3 Properties

In this section, we present some important properties of the RRT algorithm. These properties indeed make this exploration technique very suitable for our test generation problem.

Probabilistic completeness

Probabilistic completeness is an important property of the RRT algorithm, which is stated as follows.

Theorem 5. *If a feasible trajectory from the initial state x_{init} to the goal state x_{goal} exists, then the probability that the RRT algorithm finds it tends to* 1 *as the number k of iterations tends to infinity.*

The proof of the theorem can be found in [62, 65]. Although the interest of this theorem is mainly theoretical, since it is impossible in practice to perform an infinite number of iterations, this result is a way to explain the good space-covering property of the RRT algorithm, which we discuss in the following.

Space-covering property

Although the main idea of the RRT algorithm is simple, the algorithm allows to explore the state space efficiently. This technique has been successfully used in solving practical path planning problems. It is however hard to characterize the convergence rate of the algorithm for general cases. In the following we give an intuitive explanation of the good space-covering property of the algorithm. Some theoretical results on the convergence of the algorithm for a number of special cases were published in [29].

Before continuing, we briefly recall the *Voronoi diagram* notion, since it will be used in our explanation. The Voronoi diagram of a set V of points in $\mathbb{R}^n$ is the partition of $\mathbb{R}^n$ into k polyhedral regions $vocell(v)$ with $v \in V$. Each region $vocell(v)$, 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, or more precisely,

$$vocell(v) = \{x \in \mathbb{R}^n \mid \forall w \in V \setminus \{v\} : \ d(x, v) \leq d(x, w)\}$$

where d is the Euclidean distance function. For more information on Voronoi diagrams, the reader is referred to [40].

Let $V = Vertices(\mathcal{T}^k)$ be the set of k vertices of the RRT tree $\mathcal{T}^k$ constructed in the k^{th} iteration (which means that a new vertex is added to the tree in each iteration). We consider the Voronoi diagram of the set V.

Since the goal states are uniformly sampled over the free space $\mathcal{X}_{free}$, the probability that x_{goal} is inside the Voronoi cell of a vertex v depends on the volume of the intersection of $vocell(v)$ and $\mathcal{X}_{free}$. Note that the tree can evolve only in the free space $\mathcal{X}_{free}$. In addition, by the definition of Voronoi diagrams, if $x_{goal} \in vocell(v)$ then v is a closest neighbor of x_{goal}. Therefore, the larger the volume of the set $vocell(v) \cap \mathcal{X}_{free}$ is, the greater the probability that v is selected as the starting point for expansion is. Note that during the first iterations of the algorithm, the region covered by the tree is still small, and the vertices on the 'boundary' of the tree have larger Voronoi cells. Therefore, the exploration is biased towards the large unvisited regions surrounding the tree. Figure 4.4 shows the evolution of the Voronoi diagrams of a tree incrementally constructed by the algorithm.

To give an intuitive explanation why the RRT algorithm is an efficient search method, we compare it with a simple randomized algorithm which incrementally constructs a tree as follows. In each iteration, it chooses at random a vertex from the current tree and an input from the set of admissible inputs. It then creates a new vertex using the successor function. Figure 4.3 shows the experimental result obtained by this algorithm, compared to the result for the same example obtained by the RRT algorithm. We can see from the figure that the exploration of the RRT algorithm is spread over the state space while the exploration of this simple algorithm is concentrated within a small part of the state space around the initial point. This can be explained as follows. If the vertex for expansion is *uniformly* chosen from the

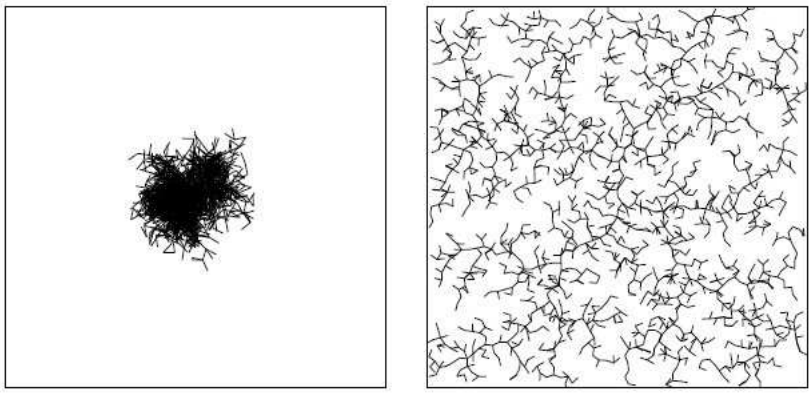

Figure 4.3: The result obtained by the simple randomized algorithm (left) and the result obtained by the RRT algorithm (right). Each tree contains 2000 vertices [67].

set of existing vertices, then the vertices in the already well-explored regions have the same probability of being chosen as the vertices in the unexplored regions. In the RRT algorithm, the exploration, in contrast, is biased towards the large regions which are not yet visited.

4.2 Test generation algorithm

In this section we propose a test case generation algorithm, which we call **hRRT**. This algorithm is based on an extension of the RRT algorithm to hybrid systems. We defer a discussion on some previous work along this line to the end of this chapter. Let $\mathcal{A}$ and $\mathcal{A}_s$ be two hybrid automata respectively modeling the specification and the system under test SUT. Essentially, our test generation algorithm consists of the following two steps:

- From the specification automaton $\mathcal{A}$, generate an exploration tree using an extension of the RRT algorithm.
- Determine the verdicts for the executions in the exploration tree, and

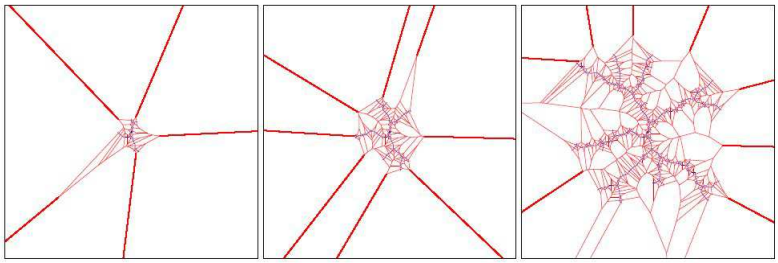

Figure 4.4: The evolution of the Voronoi diagram of a tree constructed by the RRT algorithm [67].

extract a set of test cases interesting with respect to the property to verify.

In the following we describe these two steps. Since the test generation algorithm operates on the specification automaton, for brevity we often simply say 'the hybrid automaton' to refer to the specification automaton $\mathcal{A}$.

4.2.1 Exploration tree construction

Extending the RRT algorithm to hybrid systems requires the ability to compute the following basic functions of the algorithm for these systems.

- RANDOM_STATE: Since the state space is hybrid, sampling a state requires not only sampling a continuous state but also a discrete state. In addition, as we will see later, instead of uniform sampling, we want to guide the sampling process in order to achieve a good coverage of the behaviors of interest.
- NEAREST_NEIGHBOR: To determine a nearest neighbor of a state, we need to define a distance between two hybrid states. In a continuous setting where the state space is a subset of $\mathbb{R}^n$, many distance metrics exist and can be used in the RRT algorithms. Nevertheless, in a hybrid setting defining a meaningful hybrid distance is a difficult problem.

In the following we propose a hybrid distance, which is not a metric but proved to be appropriate for our purposes of developing guiding strategies.

- NEW_STATE: The successor function for a hybrid system should compute not only successors by continuous evolution but also successors by discrete evolution.

Like the RRT algorithm, our test generation algorithm stores the visited states in a tree $\mathcal{T}$ the root of which corresponds to the initial state, and each edge of $\mathcal{T}$ is labeled with a control action.

The construction of an exploration tree from the specification automaton $\mathcal{A}$ is summarized in Algorithm 2. We change the name of the basic functions of the RRT algorithm to emphasize that in our algorithm there are important modifications in these functions.

Algorithm 2 Test generation algorithm **hRRT**

$\mathcal{T}^k.init(s_{init})$ ▷ s_{init}: *initial state*
$k = 1$
repeat
  $s_{goal} = \text{SAMPLING}(\mathcal{S})$ ▷ $\mathcal{S}$: *hybrid state space*
  $s^k_{near} = \text{NEIGHBOR}(\mathcal{T}^k, s^k_{goal})$
  $(s^k_{new}, u^k_{q_{near}}) = \text{CONTINUOUSSUCC}(s^k_{near}, h)$ ▷ h: *time step*
  $\text{DISCRETESUCC}(\mathcal{T}^k, s^k_{new})$
  $k{+}{+}$
until $k \geq k_{max}$

As in the function RANDOM_STATE, in the k^{th} iteration 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 $s^k_{near} = (q^k_{near}, x^k_{near})$ for expansion is determined as a neighbor of s^k_{goal}. The definition of the distance between two hybrid states will be given later.

Expanding the tree from s_{near} towards s_{goal} is done as follows:

- Next, 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 is a new continuous state x^k_{new}. A new edge from s_{near} to $s^k_{new} = (q^k_{new}, x^k_{new})$, 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.

The algorithm terminates after some maximal number of iterations. Another possible termination criterion is that a satisfactory coverage value is reached. As mentioned in Chapter 3, the δ-coverage is used for this purpose. The function Sampling plays the role of guiding the exploration and is discussed in detail in the next chapter. In the following, we show how to compute the functions in the algorithm.

Hybrid distance and computation of neighbors

In most versions of RRT algorithms, where the state space is a subset of $\mathbb{R}^n$, the query of nearest neighbors often uses the Euclidian distance. Defining a metric for the hybrid state space is difficult, due to the discrete component of a hybrid state. In this section we propose an approximate distance between two hybrid states, which will be used in the function Neighbor of Algorithm 2.

We recall that the topological space we are concerned with is a subset of $Q \times \mathbb{R}^n$.

Definition 30 (Metric)**.** *A metric is defined as a function $\rho : X \times X \to \mathbb{R}$ where X is a topological space such that for any $x_1, x_2, x_3 \in X$:*

- $\rho(x_1, x_2) \geqslant 0$ *(nonnegativity)*
- $\rho(x_1, x_2) = 0$ *iff* $x_1 = x_2$ *(reflexivity)*

- $\rho(x_1, x_2) = \rho(x_2, x_1)$ *(symmetry)*
- $\rho(x_1, x_2) \leqslant \rho(x_1, x_3) + \rho(x_3, x_2)$ *(triangle inequality)*

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. Note that this way, the hybrid distance we define is not symmetric. We present first some definitions and notation necessary for describing our hybrid distance.

Definition 31. *Given two sets A and B in $\mathbb{R}^n$, we define the average distance between A and B, denoted by $\overline{d}(A, B)$, as a distance between their geometric centroids.*

If a set A models a physical object with uniform density, then the geometric centroid of a set A coincides with its center of mass. If the set A is a bounded convex polyhedron, then the centroid of A can be defined as follows. Let $V = \{v_1, \ldots, v_J\}$ be the set of vertices of A. Then the centroid of A is:

$$c = \frac{1}{J} \sum_{j=1}^{J} v_j.$$

Definition 32 (Average length of a path)**.** *Given a path*

$$\gamma = (q_1, q_2), (q_2, q_3), \ldots, (q_{m-1}, q_m))$$

in the hybrid automaton $\mathcal{A}$, we define the average length of γ, denoted by $len(\gamma)$ as follows:

$$len(\gamma) = \sum_{i=1}^{m-1} \overline{d}(\mathcal{R}_{(q_i, q_{i+1})}(\mathcal{G}_{(q_i, q_{i+1})}), \mathcal{G}_{(q_{i+1}, q_{i+2})})$$

where $\overline{d}$ is the average distance between two sets. We recall that $\mathcal{R}_{(q_i, q_{i+1})}$ and $\mathcal{G}_{(q_i, q_{i+1})}$ are respectively the reset function and the guard associated with the transition from q_i to q_{i+1}.

Definition 33 (Average length of trajectories)**.** *Let γ be a discrete path from location q to q' in the automaton $\mathcal{A}$. Let $s = (q, x)$ and $s = (q', x')$ be two hybrid states. Then, we define the average length of trajectories from $s = (q, x)$ to $s = (q', x')$ following the path γ as:*

$$len_\gamma(s, s') = \overline{d}(x, fG(\gamma)) + len(\gamma) + \overline{d}(x', lR(\gamma))$$

where $fG(\gamma) = \mathcal{G}_{(q,q_1)}$ denotes the first guard of γ, and $lR(\gamma) = \mathcal{R}_{(q_k,q')}(\mathcal{G}_{(q_k,q')})$ denotes the set resulting from applying the reset map of the last transition to its guard set.

Example. Figure 4.5 illustrates the above definitions. We consider a path $\gamma = e_1, e_2$ where $e_1 = (q, q_1)$ and $e_2 = (q_1, q')$.

- The average length of the 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 length of trajectories from $s = (q, x)$ to $s = (q', x')$ following the path γ is the sum of three distances (shown in Figure 4.5 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'.

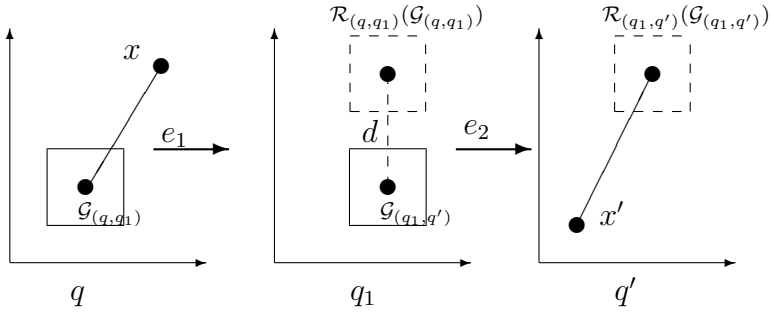

Figure 4.5: Illustration of average length of trajectory.

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 34 (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 some metric for $\mathbb{R}^n$.*
- *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, we can use any metric for $\mathbb{R}^n$. In this work, we will use the Euclidian distance and the notation $||\cdot||$ denotes this distance.

Then, in the k^{th} iteration of Algorithm 2, the function NEIGHBOR can be computed using this hybrid distance as follows:

$$s_{near} = argmin_{s \in V^k} d_H(s, s_{goal})$$

where V^k is the set of all the states stored at the vertices of the tree $\mathcal{T}^k$.

Computing continuous and discrete successors

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

When the states s_{near} and s_{goal} are at different locations, let γ be the path from q_{near} to q_{goal} with the shortest average length. 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 needs to 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 can sample a finite number of inputs and pick from this set an optimal input. In addition, we can prove that by appropriately sampling the input set, the completeness property of our algorithm is preserved.

The computation of discrete successors in DiscreteSucc, which involves testing a guard condition and applying a reset map, is rather straightforward. Notice that it is important to consider all the discrete successors by disturbance transitions since they are not controllable by the tester.

4.2.2 Test cases and verdicts

The tree constructed by Algorithm 2 can be used to extract a set of test cases. In addition, when applying such test cases to the system under test, the tree can be used to compare the observations from the real systems and the expected observations in the tree. This allows a decision whether the system satisfies the conformance relation.

4.3 Reachability completeness

As mentioned earlier, the probabilistic completeness is an important property of the RRT algorithm. An arising question is whether our test generation algorithm, built upon the RRT algorithm, preserves this property of our test generation algorithm, which we prove in this section.

We first remark that in the path and motion planning context, the proofs of the completeness of the RRT algorithms often assume that the whole free configuration space is 'controllable' in the sense that it is possible to reach

any point in $\mathcal{X}_{free}$ from the initial state x_{init} (see for example [62]). In the hybrid state space $\mathcal{S} = Q \times \mathcal{X}$, generally, not all the points in $\mathcal{S}$ are reachable from s_{init}. Indeed, if this were true, the verification problem would be solved. But we can still prove the completeness with respect to the computation of the reachable set. We call this property the *reachability completeness*. The proof of this result follows the idea of the proof in [29]. However, the lack of the above-mentioned controllability assumption makes the proof more complicated. We first introduce some definitions and intermediate results. We use the notation Pr for probabilities.

Definition 35 (Full coverage sampling condition)**.** *For any set $Y \subseteq \mathcal{S}$ such that the volume $vol(Y)$ of Y is positive, if the probability that $s^k_{goal} \in Y$ is strictly positive, then we say that the sampling process satisfies the* full coverage sampling condition.

It is easy to see that a simple uniform sampling method (that is, uniformly sampling a location q from the set Q of all locations and then uniformly sampling a continuous state in the staying set of the location q) satisfies this condition. As we will show later, the full coverage sampling condition is sufficient to guarantee the completeness. In the remainder of the section, we assume that the function SAMPLING in our test generation algorithm satisfies this property, and additionally, the reachable set has a positive volume.

Definition 36 (Neighborhood)**.** *Given a state $s = (q, x) \in \mathcal{S}$ and $\varepsilon > 0$, we define the ε-neighborhood of s as*

$$\mathcal{N}(s, \varepsilon) = \{s' \in \mathcal{S} \mid d_H(s', s) \leq \varepsilon\} \tag{4.3}$$

For a set V of states in $\mathcal{S}$, we denote

$$\mathcal{N}(V, \varepsilon) = \bigcup_{s \in V} \mathcal{N}(s, \varepsilon)$$

and call the set $\mathcal{N}(V, \varepsilon)$ **the neighborhood** *of V.*

Lemma 5. *Let R be a set of states reachable in a finite time such that $vol(R) > 0$. Then, there exists a time step h and a finite k such that*

$$\exists v \in V^k : Pr[v \in R] > 0. \tag{4.4}$$

where V^k is the set of the vertices of $\mathcal{T}^k$ at iteration k.

Proof. Let τ_{max} be the smallest time such that at τ_{max} all the trajectories from the initial state s_{init} have already visited R. Let τ_{min} be the the smallest time such that at any time $t < \tau_{min}$ none of the trajectories from s_{init} has reached R. We choose a time step h such that there exists a integer $j > 0$ such that $jh \in [\tau_{min}, \tau_{max}]$.

Then, let $K_R = \lfloor \frac{\tau_{max}}{h} \rfloor$. For $k > 0$, the time interval $[(k-1)h, kh]$ is denoted by I_k. We denote by R^k the reachable set from s_{init} such that

$$R^k = \{s \in \mathcal{S} \mid \exists \omega \in \Omega : s \in \tau_{I_k}(s_{init}, \omega) \ \wedge \ \tau(s_{init}, \omega) \cap R \neq \emptyset\}. \quad (4.5)$$

In the above definition, Ω is the set of all admissible control sequences and $\tau_{I_k}(s_{init}, \omega)$ is a segment the trajectory $\tau_{I_k}(s_{init}, \omega)$ in the time interval I_k. Intuitively, R^k contains all the states reachable during the time interval I_k by the trajectories from s_{init} that can reach R.

We assume that for all $k > 0$, $vol(R^k) > 0$. We first prove that for all $k > 0$

$$Pr[x^k_{new} \in R^k] > 0. \quad (4.6)$$

We prove this by induction. At the first iteration $k = 1$, it is always true that $s^1_{near} = s_{init}$ (because the tree contains only one vertex s_{init}). Since R^1 is reachable during the first time interval, then $Pr[s^1_{new} \in R^1] > 0$ (for example, by choosing appropriately an input sequence as shown in (4.5)); therefore, the inequality (4.6) is true for $k = 1$.

We now need to prove that (4.6) is also true for $k+1$, assuming that it is true for k. Due to the *full coverage sampling condition* (i.e. any state in $\mathcal{S}$ has a non-null probability of being sampled to be a goal state) and $vol(R^k) > 0$, we can assure that the probability $Pr[s^{k+1}_{near} \in R^k] > 0$. If $s^{k+1}_{near} \in R^k$ and $k \leq K_R$, it then follows from (4.5) that $Pr[s^{k+1}_{new} \in R^{k+1}] > 0$. Using the formula for conditional probability, we have that (4.6) is true for $k+1$.

Then, by the definition (4.5), we can find $k \leq K_R$ such that all trajectories from $s^k_{new} \in R^k$ reach R after at most h time. If so, this implies that

$Pr[s_{new}^{k+1} \in R] > 0$. Furthemore, assuming that all s_{new}^{k} are added to the tree, the statement of the lemma is thus proved. □

Remark 6. *The following conditions are sufficient for the validity of the proof:*

- *(C1) There is a non-null probability that each state in V^k is selected to be s_{near}^{k}.*
- *(C2) If R is a set of reachable states with positive volume, then for all $k > 0$ $Pr[s_{new}^{k} \in R] > 0$. Intuitively, this assumption means that there is a non-null probability that 'each reachable direction' is selected.*

Indeed, the full coverage sampling condition and the assumption that $\forall k > 0 : volV^k > 0$ used in the proof imply the above conditions.

We proceed with the main result concerning the preservation of the completeness of our test generation algorithm.

Theorem 6 (Reachability completeness)**.** *Let V^k be the set of states stored at the vertices of the tree $\mathcal{T}^k$. Given $\varepsilon > 0$ and a* **reachable** *state $s = (q, x)$, the probability that there exists a state $s' \in V^k$ such that s' is in the ε-neighborhood of s approaches 1 when k approaches infinity, that is*

$$lim_{k\to\infty} Pr[\exists s' \in V^k : s' \in \mathcal{N}(s, \varepsilon)] = 1 \tag{4.7}$$

Proof. We first remark that at each location, the reachable set is connected; therefore, for any $\varepsilon > 0$ the set

$$B_{reach}(s) = Reach \cap \mathcal{N}(s, \varepsilon) \tag{4.8}$$

(where *Reach* is the reachable set from s_{init}) has a positive volume. Hence, using the full coverage sampling property, the probability $Pr[s_{goal}^{k} \in B_r(s)] > 0$ for all $k > 0$.

We define the distance from $s = (q, x)$ to V^k as

$$d^k(s) = min_{s' \in V^k} d_H(s', s).$$

Initially, the tree contains only the initial state, that is, $V^0 = \{s_{init}\}$; hence $d^0(s) = d_H(s_{init}, s)$. Note that $d_H(s_{init}, s) < \infty$ since s is reachable from s_{init}.

If at iteration k, the tree already contains a vertex inside $B_{reach}(s)$, then (4.7) is proved.

It remains to prove (4.7) for the case where all the states in V^k are not in $B_{reach}(s)$. We have seen that $Pr[s^k_{goal} \in B_{reach}(s)] > 0$, and we suppose that $s^k_{goal} \in B_{reach}(s)$. Because the whole set $B_{reach}(s)$ is reachable, by Lemma 5, there exists a finite $k' > k$ such that

$$\exists s' \in V^{k'} : Pr[s' \in B_{reach}(s)] > 0.$$

Note that the fact $V^{k'}$ contains a state in $B_{reach}(s)$ implies that $d^{k'}(s) < d^k(s)$, since we are considering the case where none of the states in V^k is in $B_{reach}(s)$. From k In addition, $d^k(s)$ is non-increasing with respect to k; therefore the expected value of the distance to s at iteration k' must be smaller than that at iteration k, that is $E(d^{k'}(s)) < E(d^k(s))$. Therefore, $lim_{k\to\infty} Pr[d^k(s) < \varepsilon] = 1$, which means that

$$lim_{k\to\infty} Pr[\exists s' \in V^k : s' \in \mathcal{N}(s, \varepsilon)] = 1.$$

□

Remark. The validity of the proof of the reachability completeness requires the conditions (C1) and (C2). These assumptions guarantee that for any reachable state s there is a non-null probability that the new vertex s^{k+1}_{new} reduces the distance from s to the tree. In fact, the selection of s^k_{goal} controls the growth of the tree by determining both the starting state s^k_{near} and the direction of the expansion in each iteration. Consequently, to preserve the completeness it suffices to guarantee the satisfaction of the assumptions (C1) and (C2). The following lemma shows a sufficient condition for (C2) to be verified.

Lemma 7. *If the control set U is finite and for each $u \in U$ $Pr[u^k = u] > 0$, then the condition (C2) is satisfied.*

In the classic RRT algorithms, the initial state for each iteration is a nearest neighbor of the goal point, and the new vertex is then computed by solving an optimal control problem (whose objective is to minimize the distance to the current goal point). These two problems are difficult, especially for non-linear systems in high dimensions.

It is important to note that even in a continuous setting, finding an exact nearest neighbor is expensive, especially in high dimensions. We can exploit the above remark to derive a variant of the RRT algorithm which has lower complexity. Indeed, to determine the initial points we can use *approximate nearest neighbors*, provided that the two assumptions are satisfied.

4.4 Related work and discussion

In this chapter we have defined a test case generation algorithm for hybrid systems. This algorithm is build upon the well-known RRT algorithm for path and motion planning.

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 [38, 23, 59, 26, 84] and references therein). In this section, 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 [59]. The difference is that we use the centroids of the guard sets to define the distance between these sets, while the author of [59] uses the minimal clearance distance between these sets, which is harder 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 account the system's dynamics. It is based on the spatial positions of the states. In [59] 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 from one state to another, using the information on the derivatives of the variables. Another distance proposed in [59] 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 [38, 59] 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 [23], 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 [84] 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 a way to define a hybrid distance as well as a guiding heuristics.

In the next chapter, we tackle the problem of guiding the test generation process, in order to quickly achieve a good coverage quality. In other words, this can be seen as a way to speed up the convergence of the algorithm, in terms of coverage. We will do so by guiding the goal states sampling instead of using the uniform sampling.

Chapter 5

Coverage-guided generation

In this chapter we propose a tool for guiding the test generation algorithm **hRRT**presented in the previous chapter. This tool is based on the coverage measure defined using the star discrepancy. The goal of guiding tool is the guide the sampling process to bias the evolution of the tree towards the interesting region of the state space, so that to rapidly achieve a good coverage quality.

We propose two strategies for guiding **hRRT**, which results in two new algorithms. The first one is called **gRRT**, and the second one called **agRRT** which is an adaptive version of **gRRT**. Most of the results presented in this chapter were published in [78, 79].

5.1 Goal state sampling

In the following, without loss of generality, we can assume that the free space is the whole state space. Sampling a goal state $s_{goal} = (q_{goal}, x_{goal})$ in the hybrid state space $\mathcal{S}$ consists of the following two steps:

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

location q_{goal} since when the automaton is at a location, its continuous state can only evolve within the staying set of the location.

In our coverage-guided exploration we want to bias the goal state sampling distribution according to the current coverage of the visited states. In each iteration, if a location is not yet well explored, that is its coverage is low, we give it a higher probability to be selected.

Let $\mathcal{P} = \{(q, P_q) \mid q \in Q \wedge P_q \subset \mathcal{I}_q\}$ be the current set of visited states. We sample the goal location according to the following distribution:

$$Pr[q_{goal} = q] = \frac{1 - Cov(\mathcal{P}, q)}{||Q|| - \sum_{q' \in Q} Cov(\mathcal{P}, q')} \tag{5.1}$$

where $Cov(\mathcal{P}, q)$ is the current test coverage at the location q and $||Q||$ is the number of locations.

We proceed to show how we bias the sampling of the goal continuous state x_{goal}. Suppose that we have sampled a discrete location $q_{goal} = q$. To give geometric intuitions, we often call a continuous state a point. In addition, we assume that all the staying sets are boxes, and we thus denote the staying set $\mathcal{I}_q$ by the box $\mathcal{B}$.

Before describing in detail our continuous state sampling method, we discuss the difficult of our problem and explain the motivation of our solution.

5.2 Motivating examples

Before describing in detail our continuous state sampling method, we discuss the difficult of our problem and explain the motivation of our solution. For illustration, we use two examples. The first one is the one shown in the previous chapter for which the classical **hRRT** algorithm performs well, and the second is an example for which the **hRRT** algorithm with uniform sampling does not perform well.

The reason we choose these examples is that they differ in the reachability property. In the first example, the system are 'controllable' in the sense

that the whole state space is reachable from the initial state, but in the second example the reachable set is only a small region in the state space. A similar challenging example in motion planning was introduced in [101] where **hRRT** with uniform sampling fail to quickly find the solution.

Indeed, these two examples will be used through this chapter to validate the efficiency of the coverage-guided sampling method that we propose.

Example 1

We first recall the example.

Example 1. *This example 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 (or the differential constraint) is $f(x, t) = u(t)$ where the input set is $U = \{u \in \mathbb{R}^2 / ||u|| \leq 0.2\}$.*

- $\mathcal{B} = [-3, -3] \times [3, 3]$
- $\dot{x} = u$ where $u \in U = \{u \in \mathbb{R}^2 \ \mid \ ||u|| \leq 0.2\}$

Indeed, when treating this example with the **hRRT** algorithm 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. To evaluate the performance of the algorithm in terms of coverage, we use the coverage measure defined using the star discrepancy. It is important to emphasize that the **hRRT** algorithm used here is the classical one that uses uniform sampling and the Euclidian metric. The uniform sampling is implemented using the C++ pseudo-random function.

For the star discrepancy estimation we use a static partition Π such that $W(\Pi) = 0.03$.

Figure 5.1 illustrates how fast the **hRRT** algorithm covers the global state space. Figure 5.2 shows the coverage after each iteration k of two sets of states: the set P^k of the visited states of the **hRRT** tree (solid curve in

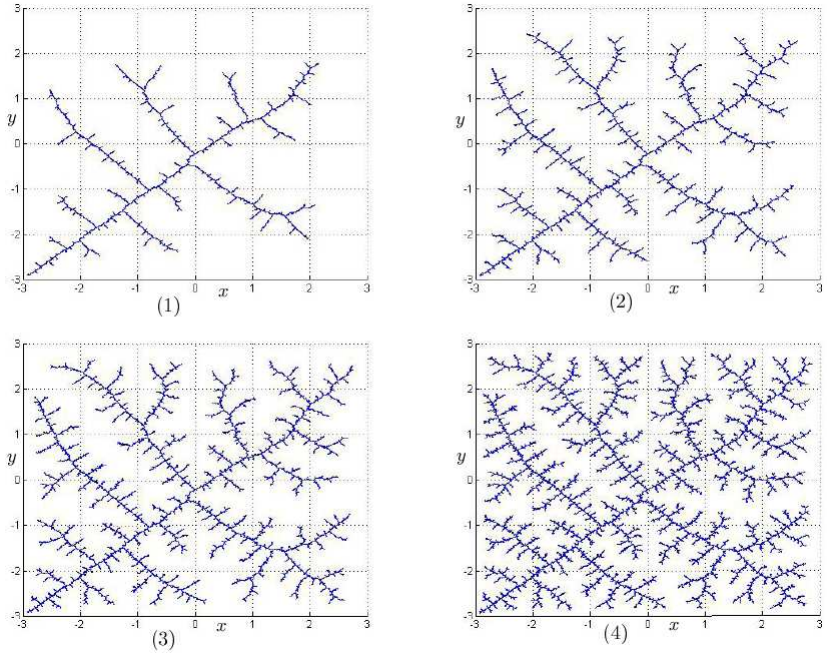

Figure 5.1: The **hRRT** generation expansion.

Figure 5.2) and the set G^k of sampled goal states(dashed curve in Figure 5.2). Note that these sets have the same number of points. The figure also show the disparity $\gamma^*(P^k, G^k, \mathcal{B})$ between them.

From the figure, we can see that the coverage of the visited points P^k catches up with the coverage of the goal points G^k until a saturation. The disparity between the two sets is also shown in the figure.

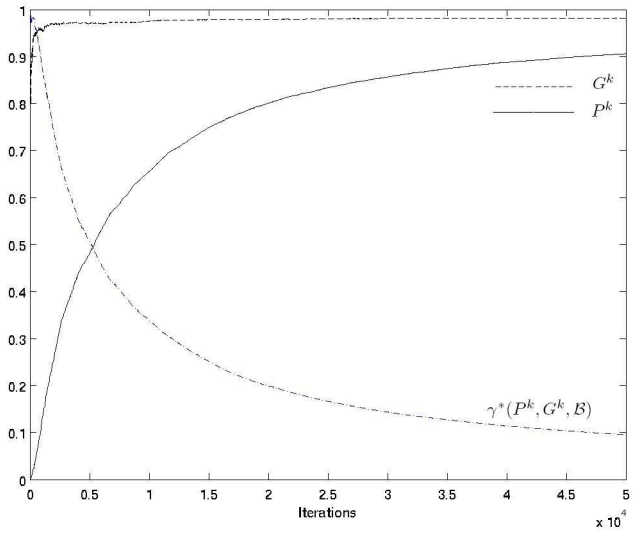

Figure 5.2: Coverage and disparity evolution of P^k and G^k .

Example 2

Example 2. *This example is a linear system in* $\mathbb{R}^2$ *with a stable focus at the origin. Its dynamics is described by:*

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

We let the dynamics be slightly perturbed by an additive input u*. The state space is the bounding box* $\mathcal{B} = [-3, -3] \times [3, 3]$*. The input set* $U = \{u \in \mathbb{R}^2 \mid ||u|| \leq 0.2\}$.

The phase portrait of the dynamics is shown in Figure 5.3. Again, we

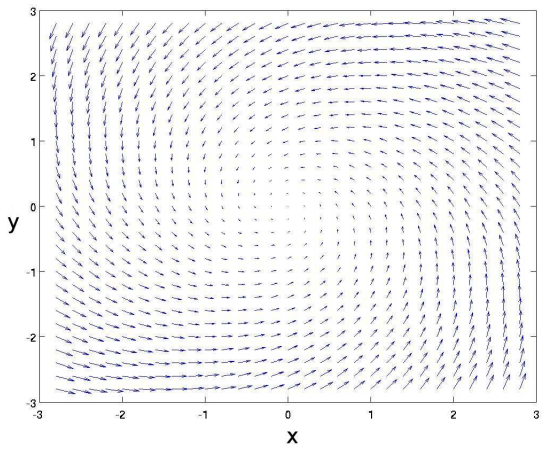

Figure 5.3: The vector field and the set of initial points.

sample from this set 100 values to run with the classical **hRRT** algorithm. In this example, we consider a set of initial states defined by the box $[2, 2] \times [2.5, 2.5]$. We start the **hRRT** algorithm with a set of 100 initial points uniformly sampled within the box. The time step is 0.002. For the coverage estimation we use a static partition Π with $W(\Pi) = 0.03$. Figure 5.4 shows the results obtained after 50000 iterations.

We can see that the disparity between the set of goal states and the set of visited states is large. This can be explained as follows. As mentioned earlier, due to the uniform sampling of goal states, the RRT algorithm is biased by the Voronoi diagram of the vertices of the tree. If the actual reachable set is only a small fraction of the state space, the uniform sampling over the whole state space leads to a strong bias in selection of the points on the boundary of the tree, and the interior of the reachable set can only be explored after a large number of iterations. Indeed, if the reachable was known, sampling within the reachable set would produce better coverage results. In the next sections, we describe two methods for guiding the sampling distribution towards optimizing the coverage, using the coverage information of the current

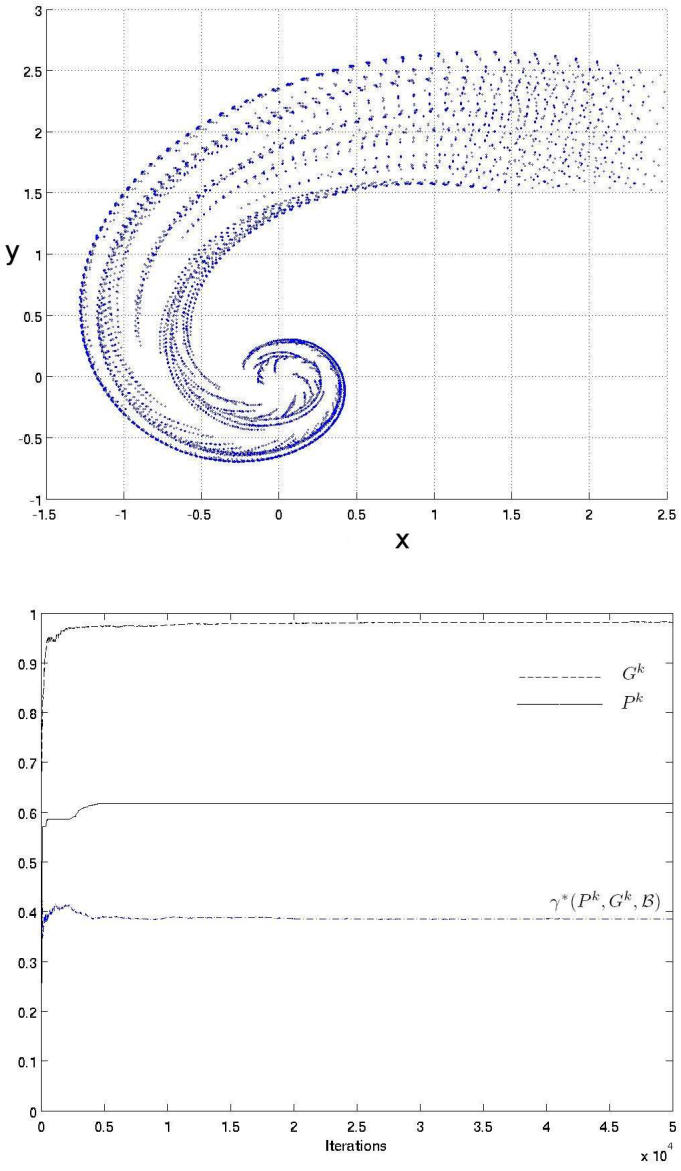

Figure 5.4: Result obtained using **hRRT** algorithm.

sets of visited states and goal states.

5.3 Coverage-guided sampling

The main idea of our coverage-guided exploration is to guide the goal state sampling in order to achieve a good coverage of the visited states. 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.

We recall that the coverage estimation is done using a box partition Π of the state space $\mathcal{B}$. We have assumed that all the staying sets are boxes. In what follows, we often call this box $\mathcal{B}$ the *bounding box*. Our method of sampling of a continuous goal state consists of two steps. In the first step, we sample an elementary box $\boldsymbol{b}_{goal}$ from the partition Π. In the second step, we *uniformly* sample a point x_{goal} in $\boldsymbol{b}_{goal}$, as shown in Algorithm 3. Guiding is

Algorithm 3 Continuous goal state sampling

procedure SAMPLING( )
  b_{goal} = BOXSAMPLING(Π)
  x_{goal} = UNIFORMSAMPLING(b_{goal})
return x_{goal}
end procedure

thus done in the goal box sampling process. The next section is devoted to this problem.

5.3.1 Goal box sampling

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 the lower and the upper bounds.

5.3.2 Reducing the lower bound

We associate with each box $\boldsymbol{b} \subseteq \Pi$ a number $A^*(\boldsymbol{b})$ such that

$$\frac{vol(J)}{vol(\mathcal{B})} = \frac{A^*(P, J)}{k}$$

where *vol* denotes the volume of a set. Intuitively, $A^*(\boldsymbol{b})$ represents the required number of points in the box $\boldsymbol{b}$ so that the ratio between the number of points in $\boldsymbol{b}$ and the total number of points is exactly the ratio between the volume of $\boldsymbol{b}$ and that of the bounding box.

Let $A(P, \boldsymbol{b})$ be the number of points of P which are inside $\boldsymbol{b}$. We denote

$$\Delta_A(J) = A(P, J) - A^*(P, J). \tag{5.2}$$

The sign of $\Delta_A(\boldsymbol{b})$ reflects a 'lack' or an 'excess' of points in the box $\boldsymbol{b}$, and its absolute value indicates how significant the lack or the excess is,

Now for a given elementary box $\boldsymbol{b} \in \Pi$, we can rewrite the local lower bound of the star discrepancy $D^*(P, \mathcal{B})$ of the point set P as

$$c(\boldsymbol{b}) = \frac{1}{k} \max\{|\Delta_A(\boldsymbol{b}^+)|, |\Delta_A(\boldsymbol{b}^-)|\}.$$

Hence, by Theorem 3 (in Chapter 3), the lower bound of the star discrepancy $D^*(P, \mathcal{B})$ becomes

$$C(P, \Pi) = \max_{\boldsymbol{b} \in \Pi} \{c(\boldsymbol{b})\}.$$

Our strategy to reduce the lower bound $C(P, \Pi)$ is based on the impact of adding a new point in each box $\boldsymbol{b}$ on $|\Delta_A(\boldsymbol{b}^+)|$ and $|\Delta_A(\boldsymbol{b}^-)|$ and thus on $C(P, \Pi)$.

We observe that adding a point in $\boldsymbol{b}$ reduces $|\Delta_A(\boldsymbol{b}^+)|$ if $\Delta_A(\boldsymbol{b}^+) < 0$ and increases $|\Delta_A(\boldsymbol{b}^+)|$ otherwise. However, doing so does not affect $\Delta_A(\boldsymbol{b}^-)$ (see Figure 5.5). Thus, we define a function reflecting the potential influence on the lower bound as follows:

$$\xi(\boldsymbol{b}) = \frac{1 - \Delta_A(\boldsymbol{b}^+)/k}{1 - \Delta_A(\boldsymbol{b}^-)/k}, \tag{5.3}$$

and we favor the selection of $\boldsymbol{b}$ if the value $\xi(\boldsymbol{b})$ is large. Note that for any box $\boldsymbol{b}$ inside $\mathcal{B}$, we have $1 - \Delta_A(\boldsymbol{b})/k > 0$.

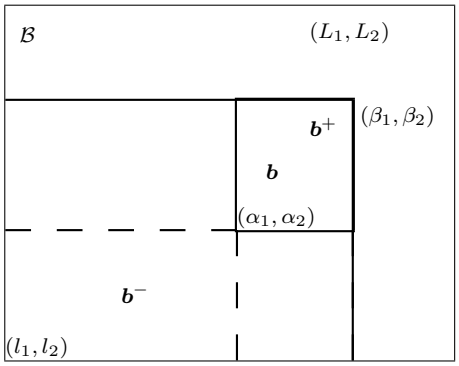

Figure 5.5: Illustration of the boxes $\boldsymbol{b}^-$ and $\boldsymbol{b}^+$.

The interpretation of the function ξ is as follows. If $\Delta_A(\boldsymbol{b}^+)$ is negative and its absolute value is large, the 'lack' of points in $\boldsymbol{b}^+$ is significant. In this case, $\xi(\boldsymbol{b})$ is large, meaning that the selection of $\boldsymbol{b}$ is favored. On the other hand, if $\Delta_A(\boldsymbol{b}^-)$ is negative and its absolute value is large, then $\xi(\boldsymbol{b})$ is small, because it is preferable not to select $\boldsymbol{b}$ in order to increase the chance of adding new points in $\boldsymbol{b}^-$.

5.3.3 Reducing the upper bound

We can rewrite the definition of the upper bound given by (3.9) in Chapter 3 as follows:

$$B(P, \Pi) = \frac{1}{k} \max_{\boldsymbol{b} \in \Pi} \mu_m(\boldsymbol{b}) \tag{5.4}$$

where $\mu_m(\boldsymbol{b}) = \max\{\mu_c(\boldsymbol{b}), \mu_o(\boldsymbol{b})\}$. Using (5.2), we can write

$$\mu_c(\boldsymbol{b}) = A(P, \boldsymbol{b}^+) - A^*(\boldsymbol{b}^-)$$

and

$$\mu_o(\boldsymbol{b}) = A^*(\boldsymbol{b}^+) - A(P, \boldsymbol{b}^-).$$

Since the value of μ_m is determined by comparing μ_c with μ_o. After straightforward calculations, the inequality $\mu_c(\boldsymbol{b}) - \mu_o(\boldsymbol{b}) \leq 0$ is equivalent to

$$\mu_c(\boldsymbol{b}) - \mu_o(\boldsymbol{b}) = \Delta_A(P, \boldsymbol{b}^+) + \Delta_A(P, \boldsymbol{b}^-) \leq 0.$$

Therefore,

$$\mu_m(\boldsymbol{b}) = \begin{cases} \mu_o(\boldsymbol{b}) & \text{if } \Delta_A(\boldsymbol{b}^+) + \Delta_A(\boldsymbol{b}^-) \leq 0, \\ \mu_c(\boldsymbol{b}) & \text{otherwise.} \end{cases} \tag{5.5}$$

Again, we observe that adding a point in $\boldsymbol{b}$ increases $\mu_c(\boldsymbol{b})$, but this does not affect $\mu_o(\boldsymbol{b})$. To reduce $\mu_o(\boldsymbol{b})$ we need to add points in $\boldsymbol{b}^-$. Hence, if $\boldsymbol{b}$ is a box in Π that maximizes μ_c in (5.4), it is preferable not to add more points in $\boldsymbol{b}$ but in the boxes where the values of $\frac{1}{k}\mu_m$ are much lower than the current value of $B(P, \Pi)$, in particular those inside $\boldsymbol{b}^-$.

Using the same reasoning for each box $\boldsymbol{b}$ locally, the smaller $|\Delta_A(P, \boldsymbol{b}^+) + \Delta_A(P, \boldsymbol{b}^-)|$ is, the smaller sampling probability we give to $\boldsymbol{b}$. Indeed, as mentionned earlier, if $\mu_m(\boldsymbol{b}) = \mu_c(\boldsymbol{b})$, increasing $\mu_c(\boldsymbol{b})$ directly increases $\mu_m(\boldsymbol{b})$. On the other hand, if $\mu_m(\boldsymbol{b}) = \mu_o(\boldsymbol{b})$, increasing $\mu_c(\boldsymbol{b})$ may make it greater than $\mu_o(\boldsymbol{b})$ and thus increase $\mu_m(\boldsymbol{b})$, because small $|\Delta_A(P, \boldsymbol{b}^+) + \Delta_A(P, \boldsymbol{b}^-)|$ implies that $\mu_c(\boldsymbol{b})$ is close to $\mu_o(\boldsymbol{b})$.

We define two functions reflecting the global and local potential influences on the upper bound:

$$\beta_g(\boldsymbol{b}) = B(P, \Pi) - \frac{\mu_m(\boldsymbol{b})}{k}$$

and

$$\beta_l(\boldsymbol{b}) = \beta_g(\boldsymbol{b}) \, \frac{|\Delta_A(P, \boldsymbol{b}^+) + \Delta_A(P, \boldsymbol{b}^-)|}{k}.$$

We can verify that $\beta_g(\boldsymbol{b})$ and $\beta_l(\boldsymbol{b})$ are always positive.

Finally, we combine these functions with ξ in (5.3) (which describes the potential influence on the lower bound) to obtain a potential influence function on both of the bounds:

$$\nu(\boldsymbol{b}) = \kappa_\xi \xi(\boldsymbol{b}) + \kappa_g \beta_g(\boldsymbol{b}) + \kappa_l \beta_l(\boldsymbol{b})$$

where κ_ξ, κ_g, and κ_l are non-negative weights that can be user-defined parameters.

Box probability distribution. We are now ready to define a probability distribution for the boxes in the box partition Π. Let P^k be a set of visited states after the k^{th} iteration, we define the probability of selecting $\boldsymbol{b} \in \Pi$ as follows:

$$Pr[\boldsymbol{b}_{goal} = \boldsymbol{b}] = \frac{\nu(\boldsymbol{b})}{\sum_{\boldsymbol{b} \in \Pi} \nu(\boldsymbol{b})}.$$

Let us summarized 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, Algorithm 2 in which the function SAMPLING uses this coverage-guided method is called the **gRRT** algorithm (which means "guided **hRRT**").

To demonstrate the performance of the **gRRT** algorithm, in the following we present the results obtained using **gRRT** and **hRRT** for the two examples, introduced in the beginning of the chapter.

Example 1. In Figure 5.6 where one can see the evolution of the coverage of the states generated by **gRRT** and **hRRT**.

The dashed curve describes the coverage obtained using the **hRRT**-based test generation algorithm with uniform sampling, and the solid one represents the result obtained by the algorithm **gRRT**. From the figure we can see a better coverage result especially in convergence rate, compared to the algorithm with uniform sampling.

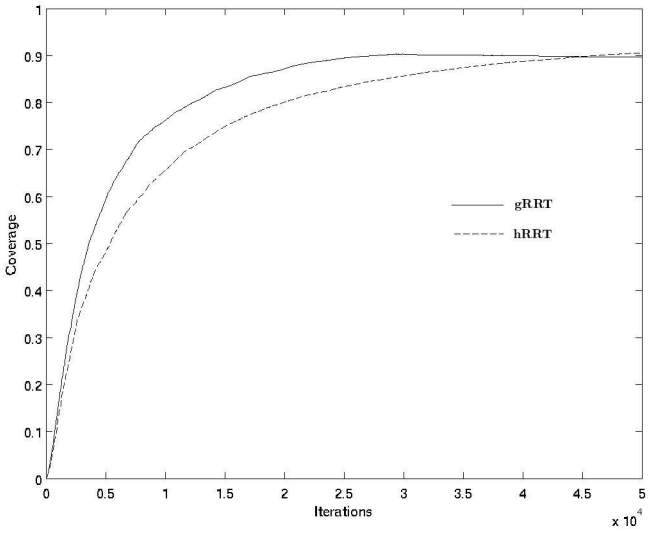

Figure 5.6: Test coverage evolution using **hRRT** and **gRRT**.

Example 2. Figure 5.7 shows the results obtained by **hRRT** and **gRRT** after 50000 iterations. Again, the **gRRT** algorithm has a better coverage result.

Controllability issue

We observe from the experiments with Example 2 that the coverage performance of **gRRT** is not satisfying when the reachable space is only a small part of the whole state space. To illustrate this, we increase the state space of the system from $\mathcal{B} = [-3, -3] \times [3, 3]$ to $\mathcal{B}' = [-5, -5] \times [5, 5]$. For the larger state space, we observe a poorer coverage quality of the result(see Figure 5.8). This can be explained as follows. There are boxes, such as those near the lower corner on the right of the bounding box, which have a high potential of reducing the bounds of the star discrepancy. Thus, the sampler frequently selects the box b. However, this box is not reachable and all attempts to reach this box do not expand the tree beyond the boundary

of the reachable set. This results in a large number of points concentrated near this part of the boundary, while other parts of the reachable set are not well explored.

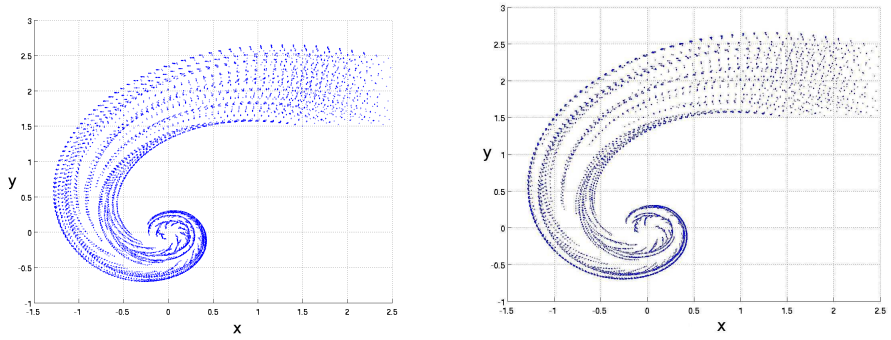

Figure 5.7: Result obtained using **gRRT** (left) and **hRRT** (right).

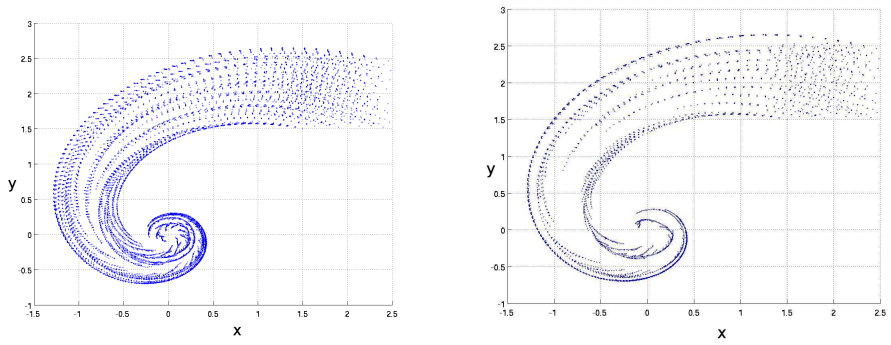

Figure 5.8: Result for the state spaces $\mathcal{B}$ (left) and $\mathcal{B}'$ (right).

It is important to emphasize that this problem is not specific to **gRRT**. The **hRRT** algorithm and, more generally, any algorithm that does not take into account the controllability 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. 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 favor a *refinement*, that is an exploration in the interior of the already visited regions.

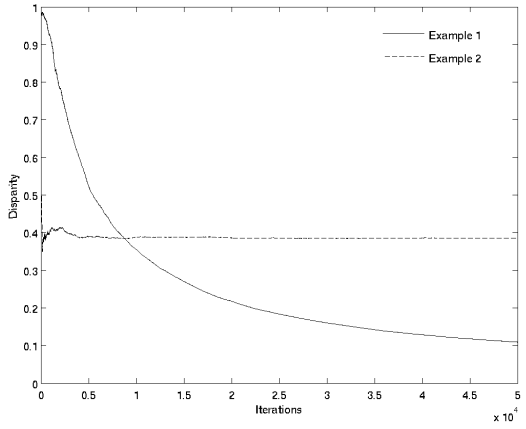

Figure 5.9: The evolution of the disparity between the set P^k of visited states and the set G^k of goal states

Figure 5.9 shows the evolution of the disparity between the set P^k of visited states and the set G^k of goal states for the two examples. We observe that for the controlable system in Example 1 the visited states follow the goal states, and thus the disparity decreases 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 to which the tree cannot be expanded further.

Figure 5.10 provides an intuitive explanation of this phenomenon. It shows the Voronoi diagram of the set of visited states. In this example, the 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. Hence, if the goal states are uniformly sampled within 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.

In RRT literature we say that RRT is Voronoi-biased [70] since at each iteration it tends to grow from the node with the largest Voronoi area.

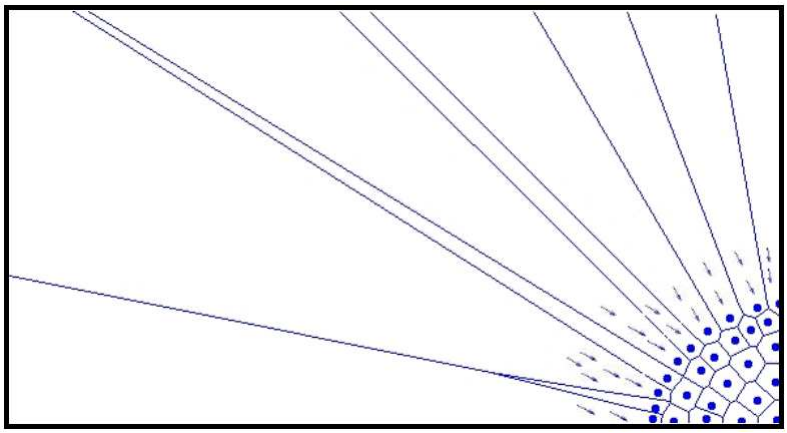

Figure 5.10: Illustration of the 'controllability' problem.

To tackle this problem, we propose to combine the coverage-guided sampling method of **gRRT** with a disparity-based sampling method, in order to better adapt to the controllability of the system. This is the topic of the next section.

5.4 Combining with disparity-guided sampling

The essential idea of our disparity-based sampling method is to detect when the controllability 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 visited states is to reduce the sampling space. Indeed, instead of sampling uniformly all over the state space, we can make a bounding box of the set P^k and give

more probability of sampling inside this box than outside it. In addition, another method is to guide the sampling using the disparity information. In the following we briefly describe this method.

As we have seen in Chapter 3, the disparity between two point sets can be estimated using a method similar to the one for estimating the star discrepancy. The objective now is to reduce the disparity between the set G^k of goal states and the set P^k of visited states.

To guide the generation to favor a refinement, we use the disparity measure and, 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 and is thus omitted here.

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 until 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 to manually decide when to switch. We call the resulting algorithm **agRRT**. The letter 'a' in this acronym stands for 'adaptive'.

Example 1. Figure 5.11 shows the result obtained using **agRRT** for Example 1.

The solid curve represents the coverage of P^k and the dashed one the coverage of G^k. The Dash-dot curve represents the disparity between G^k

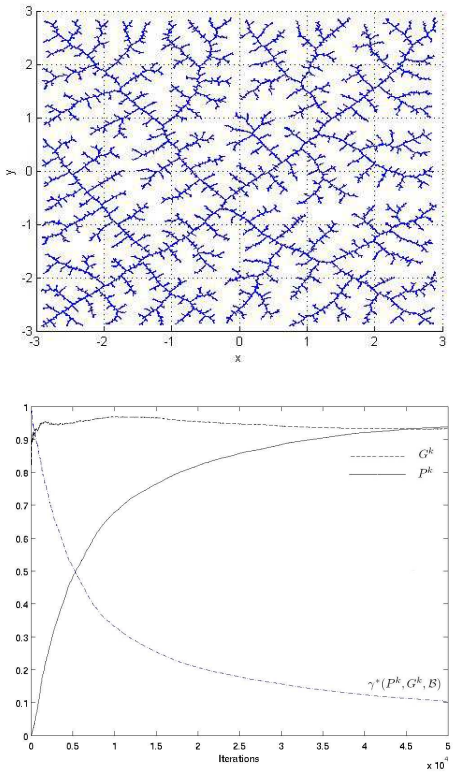

Figure 5.11: Test coverage of the result obtained using **agRRT** for Example 1.

and P^k. Figure 5.12 shows that the final result obtained using **agRRT** has a better coverage than that obtained using **gRRT**.

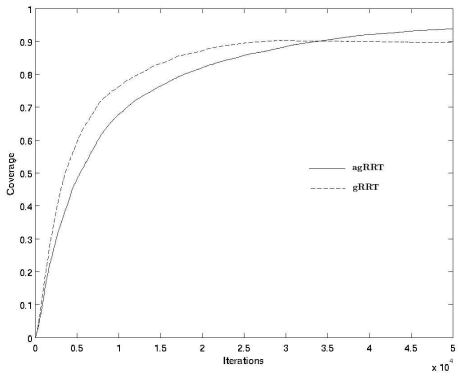

Figure 5.12: Test coverage results obtained using **gRRT** and **agRRT** for Example 1.

Example 2. The result obtained using **agRRT** For Example 2 is shown in Figure 5.13. 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 thus has a higher density of sampled goal states than the region outside the reachable set, as one can see in the figure.

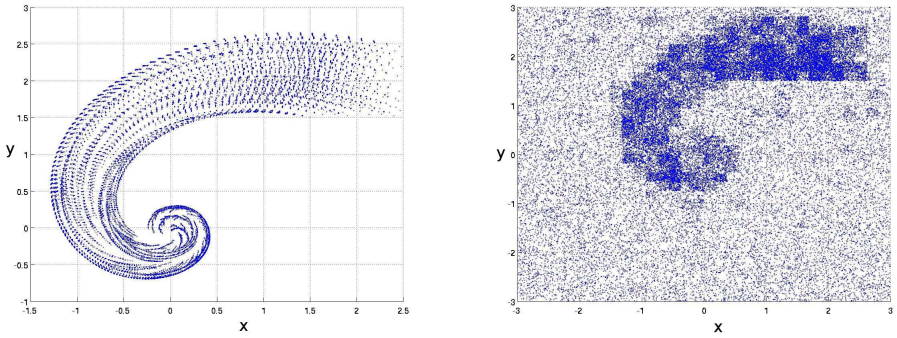

Figure 5.13: Result after $k = 50000$ iterations, obtained using **agRRT** (left: the set of visited states P^k, right: the set of goal states G^k).

5.5 Summary and related work

In this chapter we have proposed a tool for guiding our RRT-based test generation algorithm, described in the previous chapter. The guiding tool uses the star discrepancy coverage measure to guide the generation, by biasing the goal state sampling. This results in the **gRRT** algorithm. In order to account for the controllability of the system (which may prevents the algorithm to explore well the interior of the reachable space), we proposed to combine **gRRT** with a method that uses the information about the disparity between the goal states and the visited states in order to steer the exploration towards the area where the controllability of the system allows to better improve the global coverage of the result. This combination results in the **agRRT** algorithm.

We have also provided some examples to show the efficiency of this guiding tool, in terms of coverage improvement. More experimental results are presented in Chapter 7 where our coverage-guided test generation tool is applied to a number of benchmarks in control applications and in analog and mixed-signal circuits.

Concerning related work along this line, sampling the configuration space

has been one of the fundamental issues in probabilistic motion planning.

Finally, our idea of guiding the simulation via the sampling process has some similarity with the sampling domain control [101]. As mentionned 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 need to reflect the geometric and differential constraints of the system, and more generally, the controllability of the system. In [71], another method for biasing the exploration was proposed. The main idea of this method 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. This method tries 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 [38] 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 points in order to improve it. Additionally, we combine this with controllability information (obtained from the disparity estimation) to obtain a more efficient guiding strategy.

Chapter 6

Implementation

In this chapter we describe an implementation of the algorithms developed in the preceding chapters. More concretely, we show how to implement the main functions in these algorithms.

6.1 Data structure

In addition to the tree that is used to store the explored executions, to facilitate the computation of the required geometric operations, such as finding a neighbor, we store the visited states using the following data structure.

The data structure has two levels: the first level corresponding to the discrete automaton of the hybrid system under study, and the second corresponds to the continuous state spaces of the locations (see Figure 6.1). The automaton level stores the discrete locations as nodes connected by edges corresponding to the transitions. Each node of the automaton is associated with the staying set of the location. The guard sets and the reset maps are associated with the corresponding transitions.

Given a node q of the automaton, the continuous states are stored using a data structure similar to a *k-d tree* [22]. We call this data structure a-tree. This is a space-partitioning data structure for organizing points in a

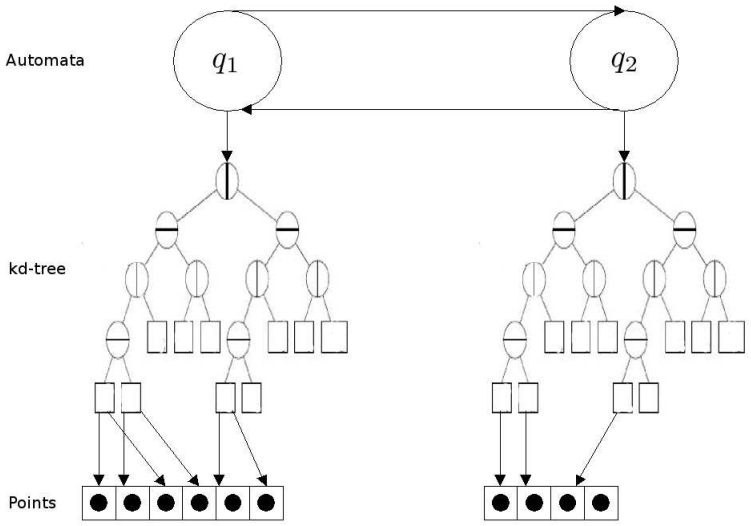

Figure 6.1: Data structure.

k-dimensional space. Each node of the tree has exactly two children. Each internal node is associated with the information about a partitioning plane: its axis i and position c, and the partitioning plane is thus $x_i = c$ (where x_i is the i^{th} coordinate of x). Note that we use only splitting planes that are perpendicular to one of the coordinate system axes. Indeed, this is also appropriate to encode the box partition used for the estimation of the coverage. The additional information associated with a leaf is a set of visited points. Each node thus corresponds to a box resulting from a hierarchical box-partition of the state space. The box of the root of the tree is the staying set $\mathcal{I}_q$ of the location which is a the bounding box $\mathcal{B}$. Each leaf of the tree corresponding to an elementary box, and the set of all the boxes at the leaves define a box partition of $\mathcal{I}_q$.

The tree and the partition of a 2-dimensional example is shown in Figure 6.4, where the axes of the partitioning planes are specified by the horizontal and vertical bars inside the nodes.

6.1.1 Construction of the a-tree

Generally, a standard kd-tree is built for a fixed set of points. Figure 6.2 shows a standard kd-tree built from a set of 10 points.

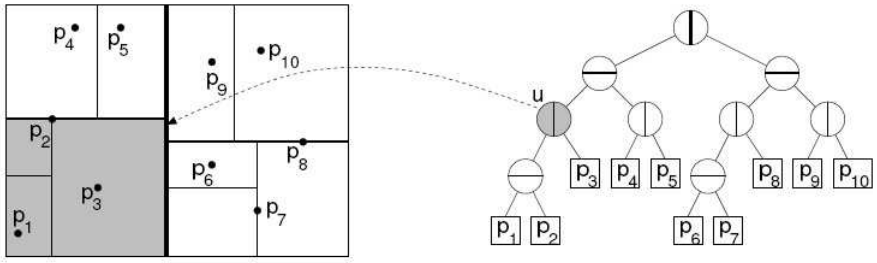

Figure 6.2: A kd-tree example.

Unlike the standard kd-trees, the construction of our a-tree is incremental because of the evolution of the point sets we need to handle. The tree is updated after a new state is generated.

We start with a rough box partition of the state space and refine it by splitting the elementary boxes, according to the density of points in the boxes and the desired precision of coverage estimation as well as the performance of the nearest neighbor search. The splitting operation is specified by an axis perpendicular to the splitting hyperplane and a point defining the position of the hyperplane. In the following we discuss some the splitting rules implemented in our tool.

Splitting rules. Let $\boldsymbol{b}$ denote the box at a leaf that we want to split and $S = \{p^1, p^2, \ldots, p^N\}$ denote the current set of N points in $\boldsymbol{b}$. The *aspect ratio* of a box is defined as the ratio between its longest and shortest side lengths. We define the *spread* l of S along a dimension i, denoted by $l(i, S)$ as the difference between the largest and smallest coordinate in this dimension, that is,

$$l(i, S) = \max_{j \in \{1,\ldots,N\}} p^j[i] - \min_{j \in \{1,\ldots,N\}} p^j[i].$$

where $p[i]$ is the i^{th} coordinate of point p.

We use the following two splitting rules:

- **Median rule**: This is a standard kd-tree splitting rule. The splitting dimension is the dimension of the maximum spread of S. The splitting point is the median of the coordinates of S along this dimension. A median partition of the points in S is then performed. This rule guarantees that all the boxes in the resulting partition have at least one point and the inconvenient is the high weight of the partition.

- **Midpoint rule**: This rule defines a splitting hyperplane orthogonal to the longest side of the box through its midpoint. This simple rule guarantees that all the resulting elementary boxes have a low aspect ratio and thus a low weight partition. This is important for the precision of the coverage estimation. A drawback of this rule is the existence of empty boxes. Figure 6.3 shows the result of a dynamic construction using this splitting rule.

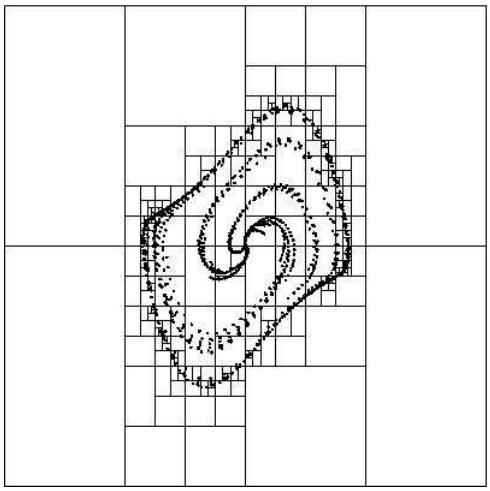

Figure 6.3: A dynamic a-tree construction using midpoint splitting rule. The result obtained for the Van der Pol system.

...

6.1.2 Adding a new point

We denote by $\mathcal{T}_q$ the tree used to store the states visited at location q, and by Π_q the box partition of the staying set $\mathcal{I}_q$. Adding a new point x in $\mathcal{T}_q$ requires finding the elementary box in Π_q that contains a given point x. This can be done using Algorithm 4. Note that the information associated with a node s consists of a partitioning axis $k = s.axis()$ and a partitioning position $d = s.pos()$, which define a *partitioning plane* $x[k] = d$.

Algorithm 4 Compute the box that contains x

procedure CONTAININGBOX((q, x))
  $s = root(\mathcal{T}_q)$, $H = \emptyset$
  while (!ISLEAF($\mathcal{T}_q, s$)) **do**
    $k = s.axis()$, $d = s.pos()$ ▷ k and d *are the axis and the position of the splitting hyperplane.*
    if ($x[k] \geq d$) **then**
      $s = s.$RIGHTCHILD(), $\sigma = -1$
    else
      $s = s.$LEFTCHILD(), $\sigma = 1$
    end if
    $H = H \cup \{\mathcal{H}(k, d, \sigma)\}$
  end while
  $\boldsymbol{b} = constructBox(H)$, $V = s.ptset()$
return (s, $\boldsymbol{b}$, V)
  s is the leaf that contains the point x
  $\boldsymbol{b}$ is the box at s
  V is the point set inside $\boldsymbol{b}$
end procedure

Algorithm 4 traverses the tree from its root to a leaf whose box contains x; this box is thus called the *containing box* of x. This procedure also collects all the half-spaces defining the containing box $\boldsymbol{b}$ and the point set at the leaf. In the algorithm, $\mathcal{H}(k, d, \sigma)$ denotes the half-space defined as $\{x \mid \sigma x[k] \leq \sigma d\}$ where k and d are the axis and the position of the splitting hyperplane

associated with the node s of the tree.

6.1.3 Computing the box of a leaf

To reduce memory usage, we do not store the coordinates of the elementary boxes of the leaves but compute them only when necessary. Algorithm 5 shows how to do this.

Algorithm 5 Computing the box of a leaf v_0

procedure GETBOX(v_0)
  $l = \text{BOTTOMVERTEX}(\mathcal{I}_q)$
  $L = \text{TOPVERTEX}(\mathcal{I}_q)$
  $\mathcal{D}^+ = \{1, 2, \ldots, n\}$ ▷ n: *is the dimension of system*
  $\mathcal{D}^- = \{1, 2, \ldots, n\}$
  $v = v_0$, $p = parent(v_0)$
  while (!ISROOT($\mathcal{T}, v$)) and (!ISEMPTY($\mathcal{D}^- \cup \mathcal{D}^+$)) **do**
    $k = p.axis()$, $d = p.pos()$
    if ($s == p$.RIGHTCHILD()) **then**
      $\mathcal{D}^- = \mathcal{D}^- \setminus \{k\}$ and $l[k] = d$
    else
      $\mathcal{D}^+ = \mathcal{D}^+ \setminus \{k\}$ and $L[k] = d$
    end if
    $v = p$, $p = parent(v)$
  end while
return (l, L) ▷ l and L: *bottom left and top right vertices*
end procedure

It should be note that Algorithm 5 can be used to determine the box associated with an internal node of the tree, by letting the input v_0 be this node. In this algorithm, we traverse the tree from the leaf upwards until we collect the constraints on all the dimensions (to define a bounded box).

6.2 Main functions

For clarity, we first recall the test generation algorithm and then present the implementation of the functions in this algorithm.

Algorithm 6 Test generation algorithm

$\mathcal{T}.init(s_{init})$, $j = 1$ ▷ s_{init}: *initial state*
repeat
  UPDATECOVERAGE($\mathcal{T}$)
  $s_{goal} =$ SAMPLING($\mathcal{S}$) ▷ $\mathcal{S}$: *hybrid state space*
  $s_{near} =$ NEIGHBOR($\mathcal{T}, s_{goal}$)
  $(s_{new}, u_{q_{near}}) =$ CONTINUOUSSTEP(s_{near}, h) ▷ h: *time step*
  DISCRETESTEPS($\mathcal{T}, s_{new}$), $j++$
until $j \geq J_{max}$

6.2.1 Updating the coverage estimation

For each location $q \in \mathcal{Q}$, this function estimates the coverage of the visited states at location q. This is done using the method described in Chapter 3. The estimation involves computing a lower and upper bound of the star discrepancy. To do so, for each elementary box in the partition Π_q of the location q, we compute the number of points in the associated boxes $\boldsymbol{b}^+$ and $\boldsymbol{b}^-$. The estimation of the coverage can also be done in an incremental manner.

When a new point x is added in the tree, the estimation of the star discrepancy needs to be updated. More concretely, we need to find all the elementary boxes $\boldsymbol{b}$ such that the new point has increased the number of points in the corresponding $\boldsymbol{b}^-$ and $\boldsymbol{b}^+$. These boxes are indeed those which intersect with the box $B_x = [x_1, L_1] \times \ldots \times [x_n, L_n]$. In addition, if $\boldsymbol{b}$ is a subset of B_x, the numbers of points in both the boxes $\boldsymbol{b}^+$ and $\boldsymbol{b}^-$ need to be incremented; if $\boldsymbol{b}$ intersects with B_x but is not entirely inside B_x, only the number of points in $\boldsymbol{b}^+$ needs to be incremented.

Searching for all the elementary boxes that are affected by x can be done

by traversing the tree from the root and visiting all the nodes the boxes of which intersect with B_x. In the example of Figure 6.4, the box B_x is the dark rectangle, and the nodes of the trees visited in this search are drawn as dark circles. This search is summarized in Algorithm 7. Indeed, we use the idea of range search algorithms to perform this frequently used operation. We start by initializing a queue S with the root of the tree $\mathcal{T}_q$. This queue is used to maintain all the nodes whose the associated box intersects with the range box B_x. From the root and at each internal node of the tree we check this intersection between the range box and there associated childes. If the box of the left child intersects with B_x, we insert the two child node at the end of the queue S, meaning that the box at the right child also intersects. If only the box of right child intersects with B_x, we insert only the right child node at the end S. The algorithm repeats this until all the nodes in the queue S are leaves.

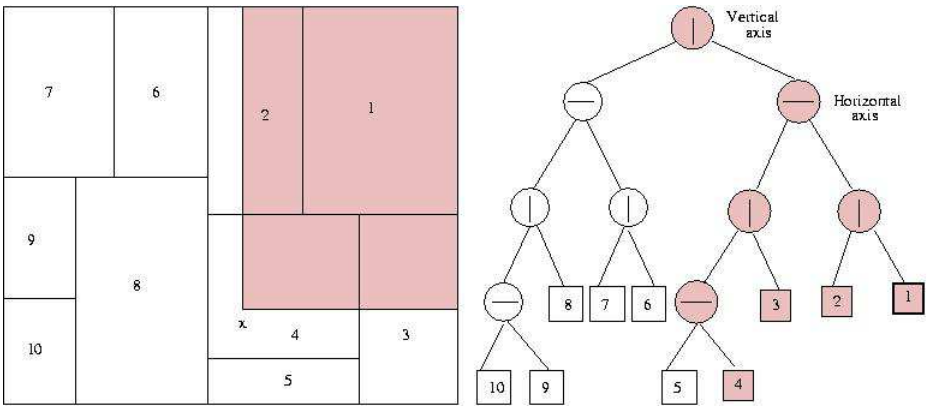

Figure 6.4: Illustration of the update of the star discrepancy estimation.

6.2.2 Sampling

First, a location q_{goal} is sampled using the distribution defined in Chapter 5. This distribution depends on the current coverages of the locations. Then, a goal point x_{goal} within the staying condition $\mathcal{I}_{q_{goal}}$ of q_{goal}, by sampling an elementary box from the partition and then a goal point in the box.

Algorithm 7 Range search

```
procedure RangeSearch((q, x))
    S = {root(T_q)}
    while (S contains a non-leaf node) do
        s = POP(S)
        if (!isLeaf(T_q, s)) then
            k = s.axis(), d = s.pos()
            if (x[k] ≥ d) then    ▷ Only the box of the right child intersects with B_x
                S.append(s.rightChild())
            else    ▷ Both of the boxes at the right nodes intersect with B_x
                S.append(s.leftChild())
                S.append(s.rightChild())
            end if
        end if
    end while
    for all b ∈ boxes(S) do
        if x ∈ b^- then
            (counter^+, counter^-)++
        else
            (counter^+)++
        end if
    end for
end procedure
```

Note that sampling an elementary box is equivalent to sampling a leaf. The sampling process is summarized in Algorithm 8. The computation of the functions DiscreteSampling, BiasedDistribution and BoxSampling are detailed in Chapter 5.

Algorithm 8 Sampling method

procedure Sampling( )
  q_{goal} = DiscreteSampling($\mathcal{Q}$)
  BiasedDistribution($\Pi_{q_{goal}}$) ▷ $\Pi_{q_{goal}}$: *box partition of* q_{goal}
  s = BoxSampling($\Pi_{q_{goal}}$) ▷ *Sample a leaf*
  $\boldsymbol{b}$ = Getbox(s)
  x_{goal} = UniformSampling($\boldsymbol{b}$)
return $s_{goal} = (q_{goal}, x_{goal})$
end procedure

6.2.3 Neighbor search

In this section we show how to find a neighbor $s_{near} = (q_{near}, x_{near})$ of a goal state $s_{goal} = (q_{goal}, x_{goal})$ using the hybrid distance $d_H(\cdot, \cdot)$ defined in Chapter 4.

Algorithm 9 takes as input a goal state $s_{goal} = (q_{goal}, x_{goal})$ and also the elementary box that contains x_{goal} (which are produced by the function Sampling (see Algorithm 8).

We first recall some notations related to the hybrid distance. For a given discrete path $\gamma = q, q_1, \ldots, q_k, q'$ between two location q and q', $fG(\gamma)$ is the guard of the first transition of γ, and $lR(\gamma) = \mathcal{R}_{(q_k,q')}(\mathcal{G}_{(q_k,q')})$ is the set resulting from applying the reset map $\mathcal{R}_{(q_k,q')}$ of the guard of the last transition. Since the centroids of these two sets are used as the starting and ending points in the definition of the average path length, we denote them by $\gamma.Start$ and $\gamma.End$.

In the algorithm, the function IsEmpty($\boldsymbol{b}$) returns TRUE if the box $\boldsymbol{b}$ is not yet visited, and it returns FALSE otherwise. The function IsEmpty(q)

Algorithm 9 NEIGHBOR method

procedure NEIGHBOR($(q_{goal}, x_{goal}, \boldsymbol{b}_{goal})$)
  $d = \infty$ ▷ *Initialize the distance to be infinity*
  if (!ISEMPTY(q_{goal})) **then** ▷ *If q_{goal} is already visited then find a continuous neighbor*
    x_{near}=**CONTINUOUSNEIGHBOR**(q_{goal}, x_{goal})
    $d = ||x_{near} - x_{goal}||$, $q_{near} = q_{goal}$, $x_{dir} = x_{goal}$
  end if ▷ *For all other locations that are already visited*
  for all $q \in \mathcal{Q} \setminus \{q_{goal}\}$ **do**
    if (!ISEMPTY(q)) **then**
      Γ=**GETALLPATHS**(q, q_{goal})
      for $\gamma \in \Gamma$ **do**
        x_γ=**CONTINUOUSNEIGHBOR**($q, \gamma.Start$)
        $d_\gamma = len(\gamma)||x - \gamma.Start|| + len(\gamma) + ||\gamma.End - x_{goal}||$
        if ($d_\gamma < d_m$) **then**
          $x_{near} = x$, $d = d_\gamma$,$q_{near} = q$, $c = \gamma.Start$
        end if
      end for
    end if
  end for
  if (d_m==∞) **then return** ($\emptyset$)
  elsereturn (q_{near}, x_{near}, c)
  end if
end procedure

where q is a location works similarly. The algorithm consists of two steps. In the first step, we try to compute a continuous neighbor, that is a neighbor that is at the same location as the goal state. In the second step, we try to compute a hybrid neighbor using the hybrid distance. We then select one that is has a shorter distance to the goal state.

- **Step** 1: If the goal location q_{goal} is already visited, then we search for a continuous neighbor in the staying set of the location, using the function **ContinuousNeighbor** which is discussed in the next section. This continuous neighbor has the distance d to x_{goal}.

- **Step** 2: To find a hybrid neighbor, let Q_v be the set of all the locations which are already visited. We consider the set Γ of all the paths from a location in Q_v to q_{goal}. Then, for each path $\gamma \in \Gamma$, where q is its starting location and c is centroid of its first guard, we determine from the tree $\mathcal{T}_q$ a neighbor x_γ of c, using the function **ContinuousNeighbor**. We also compute the average length d_γ of the trajectories from (q, x) to (q_{goal}, x_{goal}) with respect to the path γ, using Definition 34. After this computation, we have a set of neighbors (q, x_γ), each of which corresponds to a path $\gamma \in \Gamma$. Finally, we choose among these neighbors (q, x_γ) the one that has the shortest length d_γ to be x_{near}. We also compare d_γ with the distance d of the continuous neighbor, computed in the first step.

If a hybrid neighbor (q_{near}, x_{near}) is chosen via the path γ, the algorithm also returns the centroid c of the first guard of γ. Indeed, to approach the goal state from this hybrid neighbor, the system needs to approach this guard first.

Set of neighbor elements (q_{near}, x_{near}) and a direction x_{dir} where the near state have to explore. In the continuous setting of our test case generation this direction is simply the goal state x_{goal} but in the hybrid state space this direction depend on the path that minimize our hybrid distance betwen s_{goal} and s_{near}.

6.2.4 Continuous neighbor search

In this section we describe the function ContinuousNeighbor in Algorithm 9.

In most versions of the RRT algorithm, the initial point for each iteration is a nearest neighbor of the goal point. Due to the high complexity of the computation of an exact nearest neighbor, we resort to computing a neighbor which may not be a nearest one but close to the x_{goal}. We call this *an approximate nearest neighbor*. The computation of such an approximate nearest neighbor can be done as follows:

- First, we find the elementary box $\boldsymbol{b}$ with at least one visited point that is closest to x_{goal}. Note that the goal box $\boldsymbol{b}_{goal}$ that contains x_{goal} may not contain any visited points.

- Then, we find a point in the box $\boldsymbol{b}$ which is the closest to x_{goal}. It is easy to see that $\boldsymbol{b}$ does not necessarily contain a nearest neighbor of x_{goal}, which may indeed be in a neighboring box.

The remaining question now is to find a nearest box. Before continuing with this, we remark that, besides the complexity reason, we use this approximation since the sampling distribution reflects the boxes we want to explore, that is, the sampled goal box $\boldsymbol{b}_{goal}$ indicates the region we want to explore. In addition, as we will show later, this neighbor approximation preserves the completeness.

Computing a nearest box We define the distance between a point x and a box $\boldsymbol{b}$ as follows. The box $\boldsymbol{b}$ can be written as the product of n intervals of the form $I_i = [l_i, L_i]$.

We define the offset from x to $\boldsymbol{b}$ as a n-dimensional vector ω_b where the i^{th} component is defined as follows:

$$\omega_{\boldsymbol{b}}[i] = min_{y \in I_i} |x[i] - y[i]|$$

where the notation $x[i]$ is used to denote the i^{th} coordinate of the point x. It is easy to see that $\omega_b[i]$ is negative if $q[i]$ is on the left of I_i, zero if it is inside I_i, and positive if it is on the right of I_i.

Then, we define the distance from x to $\boldsymbol{b}$, denoted by $d(x, \boldsymbol{b})$ as the sums of the squares of these offsets.

$$d(x, \boldsymbol{b}) = \left(\sum_{i=1}^{i=d} |\omega_{\boldsymbol{b}}[i]|^2 \right)^{1/2}.$$

In the above we use the Euclidian norm, but any Minknowski norm can also be used.

Figure 6.5 illustres this notion. In this figure, the $d(y, \boldsymbol{b}) = 0$ and $d(x, \boldsymbol{b})$ is the Euclidean distance between x and the bottom left vertex of the box $\boldsymbol{b}$.

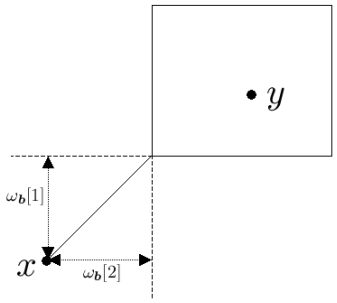

Figure 6.5: Illustration of the distance from a point to a box.

The algorithm works as follows. The algorithm maintains a priority queue S. Each element of S consists of a node v of the tree and a priority defined as the inverse of the distance from x to the box at the node v, that is $1/d(x, \boldsymbol{b})$. We start with the root of the tree. In each iteration, using the function ComputeDistance, we compute the distance between the query point x and the boxes (denoted $\square(\cdot)$ in Algorithm 10) at each child of the current v and then insert the child in the queue S if it contains visited points. The algorithm iterates over all the elements of the priority queue S until it reaches a leaf node, and the box at this leaf is the nearest box of the query point x (which contains at least one point).

In addition, the computation of the distance from a point to a box can be

Algorithm 10 Compute an approximate neighbor of x_{goal}

```
procedure ContinuousNeighbor(q, x)
    v=T_q.root, d = 0
    repeat
        if (!Empty(v.leftChild()) then
            d_l=ComputeDistance(d, x, □(v.leftChild))
            S.insert(v.leftChild(), d_l)
        end if
        if (!Empty(v.righChild()) then
            d_r=ComputeDistance(d, x, □(v.rightChild)
            S.insert(v.leftChild(), d_r)
        end if
        (v, d)=S.popHighPriority()
    until (!isLeaf(v))
end procedure
```

done in an incremental way as follows.

Incremental distance computation. When the algorithm descends the tree from a node v to its children v_l and v_r, it is possible to compute incrementally the distance from x to the two children boxes $\boldsymbol{b}_l$ and $\boldsymbol{b}_r$. Let k denote the splitting dimension and let p denote the splitting value. We can assume that $\boldsymbol{b}_l$ is closer to x than $\boldsymbol{b}_r$ is. The other case is handled similarly. (see Figure 6.6)

Because $\boldsymbol{b}_l$ is the closer box to x, its distance and offsets from x are the same as the box $\boldsymbol{b}_v$ at the parent node v. For $\boldsymbol{b}_r$, we observe that for each dimension $i \neq k$, $\omega_{\boldsymbol{b}_r}[i] = \omega_{\boldsymbol{b}_v}[i]$, since these coordinates are not affected by the current splitting. Since this is the farther of the two children, it follows that along the splitting dimension, the offset between x and $\boldsymbol{b}_r$ is the distance from x to the splitting value, that is, $\omega_{\boldsymbol{b}_r}[k] = x[k] - p$.

To compute the distance between x and $\boldsymbol{b}_r$, we simply subtract the square of the existing offset $\omega_v[k]$ and add the square of the new offset $\omega_{\boldsymbol{b}_r}[k]$

$$d(x, \boldsymbol{b}_r) = (d(x, \boldsymbol{b}_v)^2 - \omega_{\boldsymbol{b}_v}[k]^2 + \omega_{\boldsymbol{b}_r}[k]^2)^{1/2}.$$

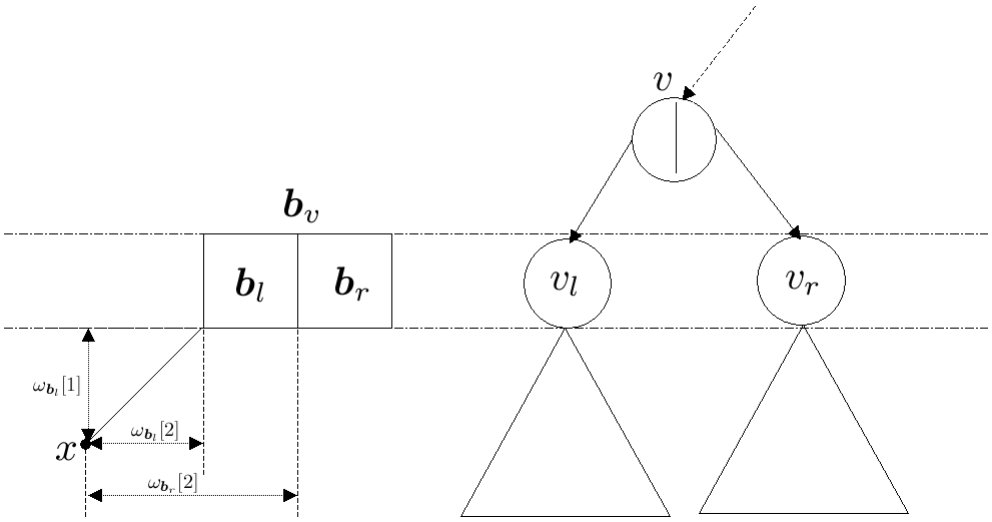

Figure 6.6: Incremental distance calculation technique

6.3 The hybrid test generation tool: HTG

We have implemented the above algorithm in a tool, which we call **HTG**. The implementation has a modular architecture (see Figure 6.7). The solid boxes represent the algorithms that we implemented, and the other boxes are the numerical integration tools **RADAU** [2] and **SICONOS** [4] that we use.

The data module contains the implementations of all data structures and the functions in the test generation algorithm.

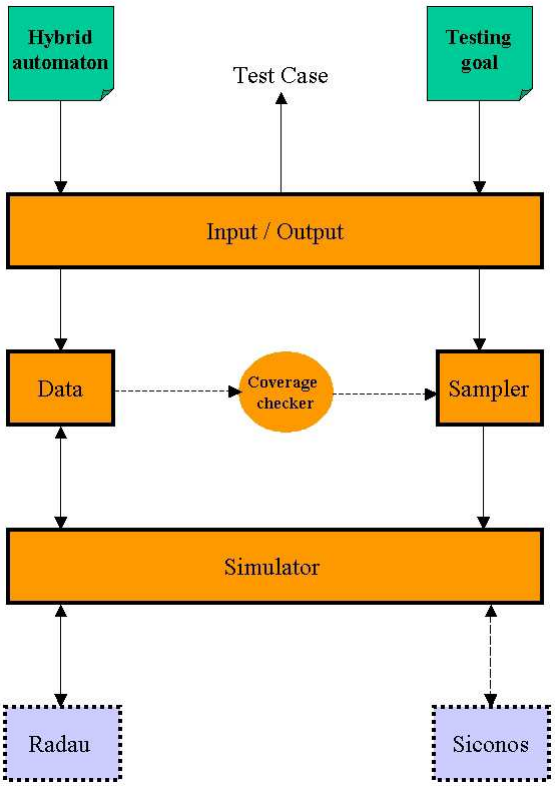

Figure 6.7: The modules of the tool.

Chapter 7

Case studies

In this chapter we treat a number of case studies to demonstrate the performance of our test generation algorithm. These experimental results were obtained using the prototype tool, called **HTG**. All the results reported in this chapter were obtained by running the tool on a 1.4 GHz Pentium III using a GNU/Linux system.

We are concerned with demonstrating, on one hand, the performance of our approach in terms of time and coverage efficiency and, on the other hand, the applicability of our testing approach to the domain of analog and mixed signal circuits. Indeed hybrid systems provide a suitable formal model for describing the behaviors of such circuits.

7.1 Linear systems

To demonstrate the time efficiency of **HTG**, we use a set of examples of linear systems in various dimensions

$$\dot{x} = Ax + u$$

In this experiment, we did not exploit the linearity of the dynamics and the tested systems were randomly generated: the matrix A is in Jordan canonical form, each diagonal value of which is randomly chosen from $[-3, 3]$

dim n	Lower bound		Upper bound	
	gRRT	RRT	**gRRT**	RRT
3	0.451	0.546	0.457	0.555
5	0.462	0.650	0.531	0.742
10	0.540	0.780	0.696	0.904

Table 7.1: Discrepancy results obtained for some linear systems using **gRRT** and **RRT**.

dim n	Time (min)
5	1
10	3.5
20	7.3
50	24
100	71

Table 7.2: Computation time of **gRRT** for some linear systems.

and the input set U contains 100 values randomly chosen from $[-0.5, 0.5]^n$. We fix a maximal number $K_{max} = 50000$ of visited states. In terms of coverage, the star discrepancy of the results obtained by gRRT and the classic RRT algorithm are shown in Table 7.1, which indicates that our algorithm achieved a better coverage quality. These discrepancy values were computed for the final set of visited states, using a partition optimal w.r.t. to the imprecision bound in (3.33). Note that in each iteration of our test generation algorithm we do not compute such a partition because it is very expensive. This is also the reason why we could not compare the coverage for higher dimensional systems, since a precise coverage estimation using optimal partitions is too expensive for such systems.

Table 7.2 shows the time efficiency of **HTG** for linear systems of dimensions up to 100.

7.2 Aircraft collision

This case study is one of the well-known benchmarks in the hybrid systems literature [75]. In this paper, 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 7.1, 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).

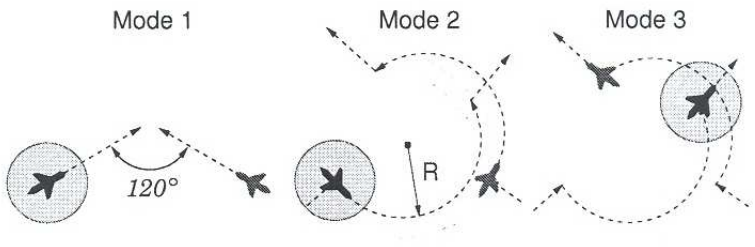

Figure 7.1: Aircraft behavior in the three modes [75].

Then, as soon as two aircraft are within the distance R (km) 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 7.2. The guard transition between mode 1 and mode 2 is given by $D(i, j) < R$ which means that the aircraft i is at R (km) distance from the aircraft j. The dynamics of each aircraft is as follows: For an ideal situation without external disturbances, we have:

$$\begin{aligned}
\dot{x}_i &= vcos(\theta_i) + d_1 sin(\theta_i) + d_2 cos(\theta_i), \\
\dot{y}_i &= vsin(\theta_i) - d_1 cos(\theta_i) + d_2 sin(\theta_i) \\
\dot{\theta}_i &= \omega \\
\dot{\theta}_i &= 0
\end{aligned}$$

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

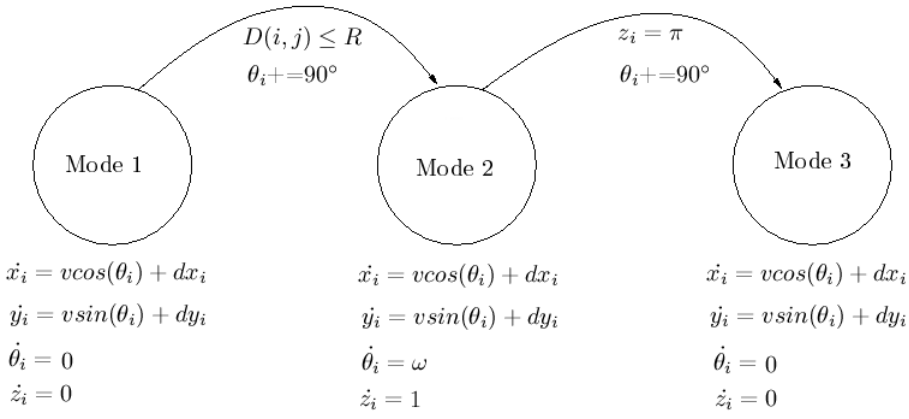

Figure 7.2: System dynamics for the three modes.

Results. Thus 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 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 7.3. 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 minutes and a collision was also found. In Figure 7.3 we show the projected positions of the eight aircraft on a 2-dimensional space.

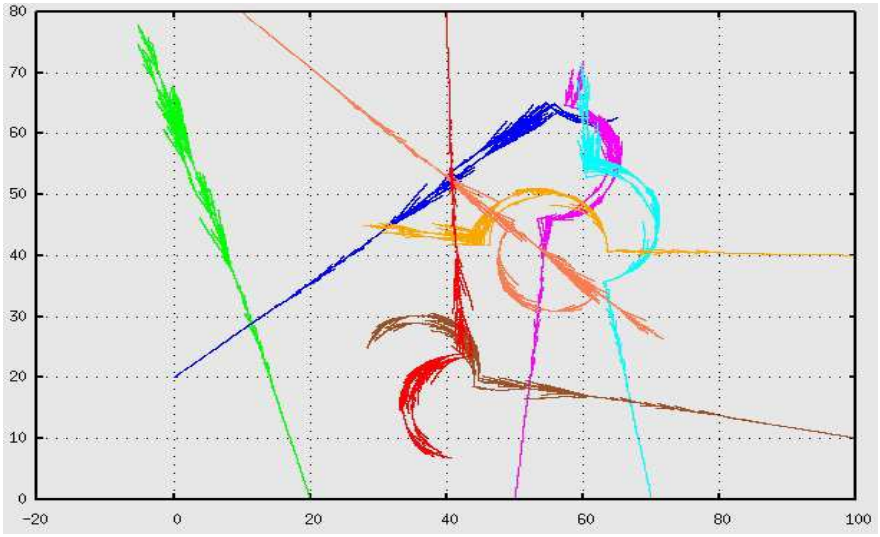

Figure 7.3: Eight-aircraft collision avoidance (50000 visited states, computation time: 10 min.

7.3 Analog and mixed signal circuits

The increasing need for analog and mixed-signal circuits has motivated the development in design and test tools for these circuits. Analog and mixed signal testing is considered to be a very difficult task. Even when the area of the analog part in a mixed-signal circuit is small, the cost of its testing covers a significant proportion of the global manufacturing cost.

In comparison with the digital counterpart, the specific difficulties of analog and mixed-signal testing are the following. While digital testing can use simple fault models (such as stuck-at-faults), fault models in analog designs are often complex and reflect the process-related disturbances, such as parameter deviations or size changes which have an infinite domain of possible values. In addition, the performance measures (such as transfer curves, frequency characteristics) are more complex than patterns of 1's and 0's at the outputs in a digital circuit.

In this work, we focus on the problem of automatic test generation, which involves computing a set of input patterns that permit detecting a given fault (for example, by comparing the differences in the outputs with some predetermined tolerance thresholds). Test generation has been considered for analog circuits such as in [44, 87, 86], for mixed-signal circuits such as in [17, 57, 20], using a variety of techniques, such as static test generation [98], sensitivity computation [44], Monte-Carlo simulation [87], and optimization [27].

Concerning coverage, in circuit testing, fault coverage is an important concern. A fault is said to be detected by a test input pattern if, when applying the input pattern to the circuit, different output patterns can be observed, for the reference (non-faulty) circuit and the faulty circuit. Generally, faults can be categorized into catastrophic and parametric faults. Examples of catastrophic faults include a change in the circuit topology, a global deviation of the circuit behavior. Parametric faults refer to small changes in the parameters that do not affect the circuit functionality. For example, a band-pass filter which has a frequency response with the correct shape but it passes a larger range of frequencies. It is often assumed that beyond a deviation of 10% is considered to have caused faults.

In the following, we demonstrate the applications of our test generation method for analog and mixed-signal circuits using hybrid system models. Hybrid systems, combining discrete event systems and continuous systems, can naturally describe the behaviors of these circuits. Formal verification of these circuits using these models has been investigated in [31, 41]. The

advantage of using hybrid system models is that the test generation is performed within a unified framework without separation between the digital and analog parts. In addition, the test generation process is guided by a coverage measure which is suitable for circuit applications. We now show the experimental results for three benchmark circuits:

- A tunnel diode circuit
- An amplifier transistor
- A voltage controlled oscillator (VCO)
- A Delta-Sigma modulator, a very popular circuit for analog to digital conversion. This is a mixed signal circuit.

7.3.1 Tunnel diode circuit

This circuit is taken from [1] and its diagram is shown in Figure 7.4.

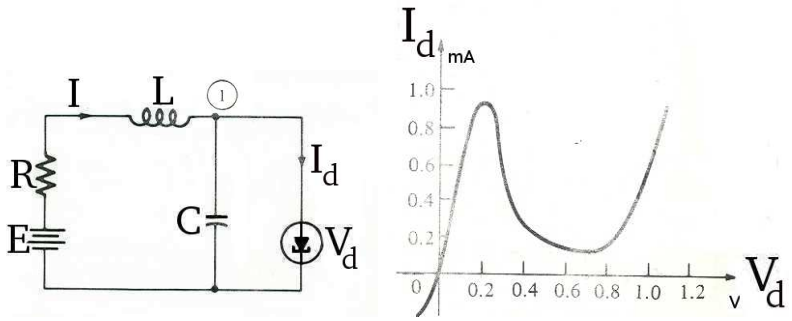

Figure 7.4: The tunnel diode circuit (left) and the Tunnel diode characteristic (right).

Modeling. The state variables are $(x_1, x_2) = (I, V_d)$ where I is the current through the inductor and V_d is the voltage across the diode (see Figure 7.4). Applying the Kirchhoff's law, and after some mathematical calculations,

we obtain the flowing differential equations describing the behavior of the circuit:

$$\begin{aligned} \dot{x}_1 &= \tfrac{1}{C}(-\iota(x_2) + x_1) \\ \dot{x}_2 &= \tfrac{1}{L}(E - Rx_1 - x_2) \end{aligned}$$

The circuit has the following circuit parameters: $C = 2pF$, $L = 5nH$, $E = 1.2V$, $R = 1.5k\Omega$. The tunnel diode characteristic, shown in Figure 7.4, is described by the following equation

$$I_d = \iota(V_d) = 17.76V_d - 103.79V_d^2 + 229.62V_d^3 - 226.31V_d^4 + 83.72V_d^5.$$

The phase portrait of the dynamics is shown in Figure 7.5

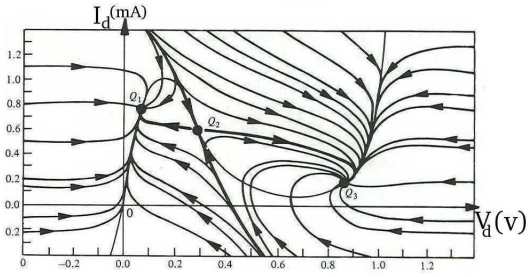

Figure 7.5: Tunnel diode circuit phase portrait

The equilibrium states Q_1 and Q_3 in Figure 7.5 are said to be asymptotically stable because all trajectories originating from points in a small neighborhood of Q_1 or Q_3 tend to Q_1 or Q_3 as $t \to \infty$. In contrast, the equilibrium state Q_2 is said to be unstable because there exist points arbitrarily close to Q_2 whose trajectories diverge from Q_2 as $t \to \infty$.

In [1], the computation of the of the equilibrium states gives the following results (see Table 7.3).

This tunnel diode circuit has been used as a computer memory, where the equilibrium state Q_1 is associated with the binary state "0" and the equilibrium state Q_3 is associated with the binary state "1". We observe

State	$V_d(V)$	$I_d(mA)$
Q_1	0.06	0.76
Q_2	0.29	0.60
Q_3	0.88	0.20

Table 7.3: Equilibrium state values.

from the phase portrait in Figure 7.5 that triggering from Q_1 to Q_3, or vice versa, means applying a small triggering signal of sufficient duration which allows the trajectory to move over to the other side of the separation curve.

We study the behavior of the circuit under two types of variations:

- the variation on the diode characteristic modeled by $I_d = \iota(V_d) + \Delta_\iota$, and
- the source voltage variation Δ_E.

In our testing framework, these variations (Δ_ι, Δ_E) can be considered as control inputs that the tester can manipulate.

Results. We used the **HTG** tool to generate test cases for the circuit. The state space is $\mathcal{B} = [-0.2, -0.2] \times [1.2, 1.2]$. The range of variation is $[-0.12, -0.12] \times [0.12, 0.12]$. The initial points are in a set of 100 points randomly sampled in $[0.20, 0.59] \times [0.3, 0.61]$, a region near the unstable equilibrium state.

Figure 7.6 shows the set of observations after 1 mn. We can see that, with these disturbances, all the generated traces are still consistent with the phase portrait.

7.3.2 Transistor amplifier

In this section we treat the transistor amplifier benchmark, taken from [36]. Its diagram is shown in Figure 7.8. The circuit equations are a system of

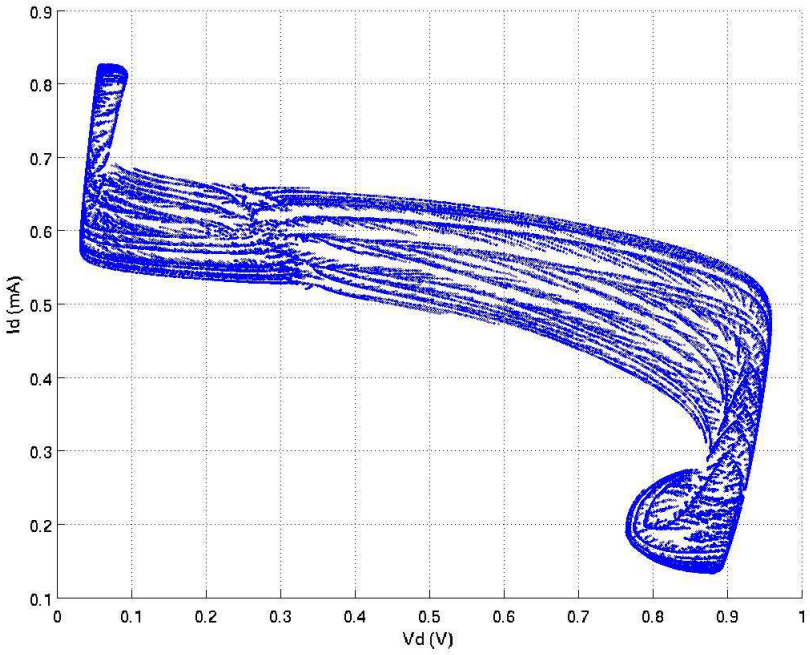

Figure 7.6: Test generation results for the tunnel diode circuit

differential-algebraic equations DAEs of index 1 with 8 continuous variables.

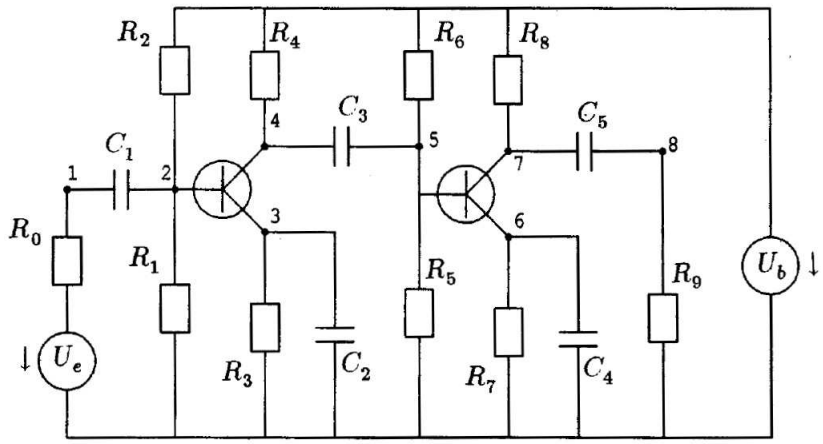

Figure 7.7: Transistor amplifier circuit [36].

Modeling. In this circuit, U_e is the input signal and U_8 is the amplified output voltage. The circuit contains two transistors of the form depicted in Figure 7.8.

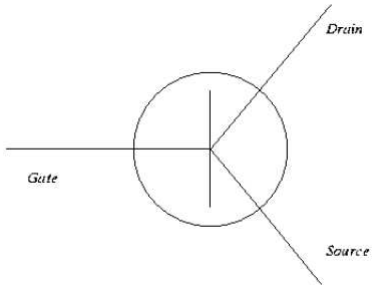

Figure 7.8: Schematic representation of a transistor.

As a simple model for the behavior of the transistors, we assume that the currents through the gate, drain and source, denoted by I_G, I_D and I_S, are given by:

$$I_G = (1 - \alpha)g(U_G - U_S),$$

$$I_D = \alpha.g(U_G - U_S),$$

$$I_S = g(U_G - U_S)$$

where U_G and U_S denote the voltage at the gate and source respectively, and $\alpha = 0.99$. The function g is given by:

$$g(x) = \beta.(e^{\frac{x}{U_F}} - 1)$$

where $\beta = 10^{-6}$ and $U_F = 0.026$.

To formulate the circuit equations, Kirchhoff's law is used in each numbered node. All currents passing through the circuit components can be expressed in terms of the voltages $U_1, \cdots, U_8$. Consider for instance node 1. The current I_{C_1} passing through capacitor C_1 is given by

$$I_{C_1} = \frac{d}{dt}(C_1(U_2 - U_1))$$

and the current I_{R_0} passing through the resistor R_0 by

$$I_{R_0} = \frac{U_e - U_1}{R_0}$$

Here, the currents are directed towards node 1 if the current is positive. We pose

$$U_i = y_i, \forall i \in \{1, \cdots, 8\}$$

and with the similar derivation for the other nodes gives the system:

$$M\frac{dy}{dt} = f(t, y) \tag{7.1}$$

where the matrix M is given by:

$$\begin{pmatrix} -C_1 & C_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ C_1 & -C_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -C_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -C_3 & C_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & C_3 & -C_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -C_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -C_5 & C_5 \\ 0 & 0 & 0 & 0 & 0 & 0 & C_5 & -C_5 \end{pmatrix}.$$

and the function f is given by:

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

A consistent initial state at $t = 0$ is

$$y(0) = \begin{pmatrix} 0 \\ U_b/(\frac{R_2}{R_1} + 1) \\ U_b/(\frac{R_2}{R_1} + 1) \\ U_b \\ U_b/(\frac{R_6}{R_5} + 1) \\ U_b/(\frac{R_6}{R_5} + 1) \\ U_b \\ 0 \end{pmatrix}, y'(0) = \begin{pmatrix} 51.338775 \\ 51.338775 \\ -U_b/((\frac{R_2}{R_1} + 1)(C_2.R_3)) \\ -24.9757667 \\ -24.9757667 \\ U_b/((\frac{R_6}{R_5} + 1)(C_4.R_7)) \\ -10.00564453 \\ -10.00564453 \end{pmatrix},$$

Numerical solution. To solve the differential algebraic system (7.1), we use the numerical solver RADAU5. The integration method implemented in RADAU5 is an implicit Runge-Kutta method (RADAU IIA) of order 5 (see [2] for more detail about the method).

Figure 7.9 shows the numerical solution of the amplified voltage output $y_8 = U_8$ (at node 8) obtained using RADAU5, under the input signal

$$U_e(t) = 0.1sin(200\pi t)$$

The technical parameters are given in Table7.4:

Testing. We are interested in studying the influence of circuit parameter uncertainty on the transient properties, such as overshoot, stabilization time. The uncertainty we consider is indeed a perturbation in the function

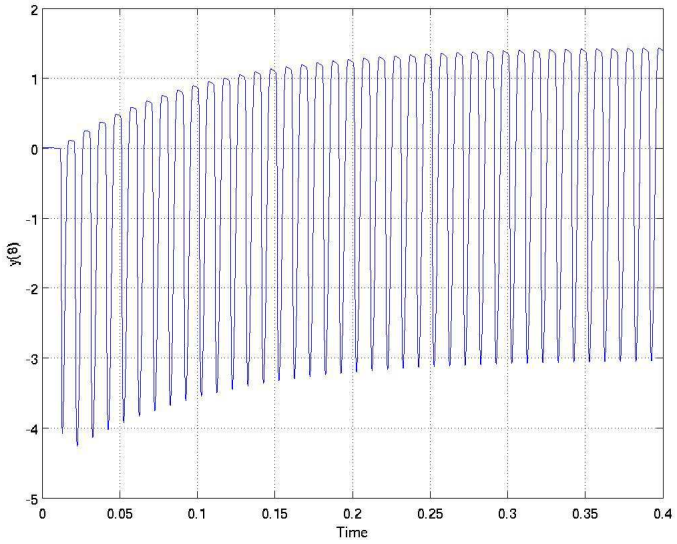

Figure 7.9: The output signal voltage

Parameter	Value
R_0	1000
R_k	9000 $(1 \leq k \leq 9)$
C_k	$k.10^{-6}$ $(1 \leq k \leq 5)$
U_b	6
U_F	0.026
α	0.99
β	10^{-6}

Table 7.4: Technical parameters.

describing 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 $\epsilon \in [\epsilon_{min}, \epsilon_{max}] = [-5e-5, 5e-5]$. For testing purposes, ϵ is considered as an input controllable by the tester.

Results. We used the **HTG** tool to generate test cases, which indicate the presence of traces with overshoots. The acceptable interval of U_8 in the non-perturbed circuit is $[-3.01, 1.42]$ (see Figure 7.9). The result is presented in Figures 7.10 and 7.11. In this figures we show the generated observation sequences projected on the variable U_8 over time.

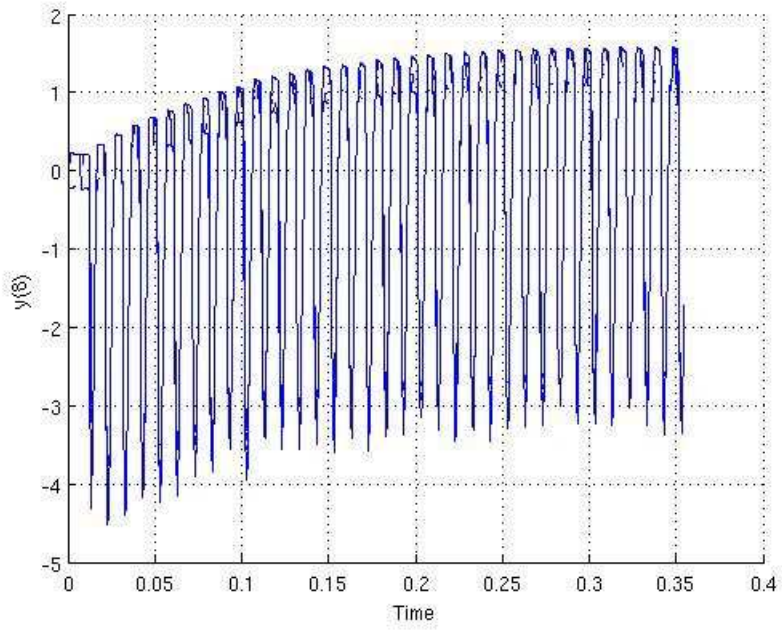

Figure 7.10: Test generation result for the transistor amplifier.

This result was obtained after 50000 iterations with 10 discretized control input values. The computation time was about 3 mn.

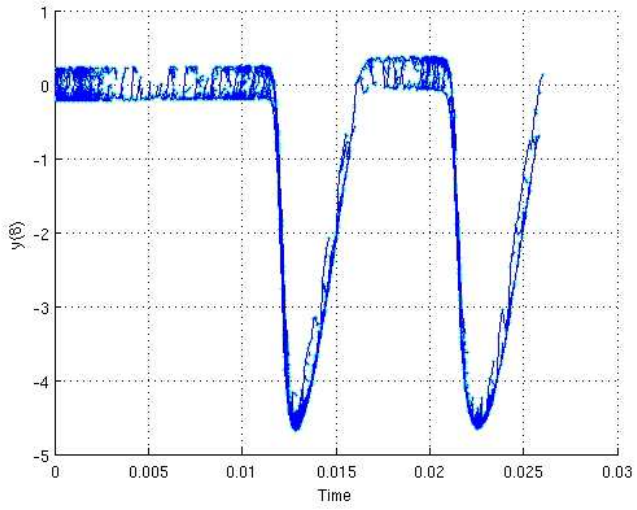

Figure 7.11: Test generation result for the transistor amplifier, zoom in the first 0.03s.

7.3.3 Voltage controlled oscillator

We examined in this section the voltage controlled oscillator (VCO). This circuit is taken from [42], shown in Figure 7.12. The behavior of this circuit can be represented by a system of differential-algebraic equations DAEs with 55 continuous variables. The input is modeled by an ideal voltage controlled current which is mirrored by $TN1$, $TP1$, $TN2$, $TP2$, $TN5$ charging the capacitance C_2. Assuming the output voltage v_{C_1} to be at V_{DD}, C_2 is charged up linearly by the input current through the switch $TN3$, $TP3$ controlled by the inverter ($TN4$, $TP4$). As v_{C_2} exceeds the positive threshold voltage of the Schmitt trigger, determined by the resistors R_1 and R_2, v_{C_1} changes to V_{SS}. Consequently, C_2 is discharged until the initial status is reached leading to an oscillation. Figure 7.14 shows the oscillation projected on variables v_{C_1} and v_{C_2}.

Testing. We are interested the oscillating frequency of the variables v_{C_1} and v_{C_2}. Indeed, this frequency is a linear function of the input voltage. We

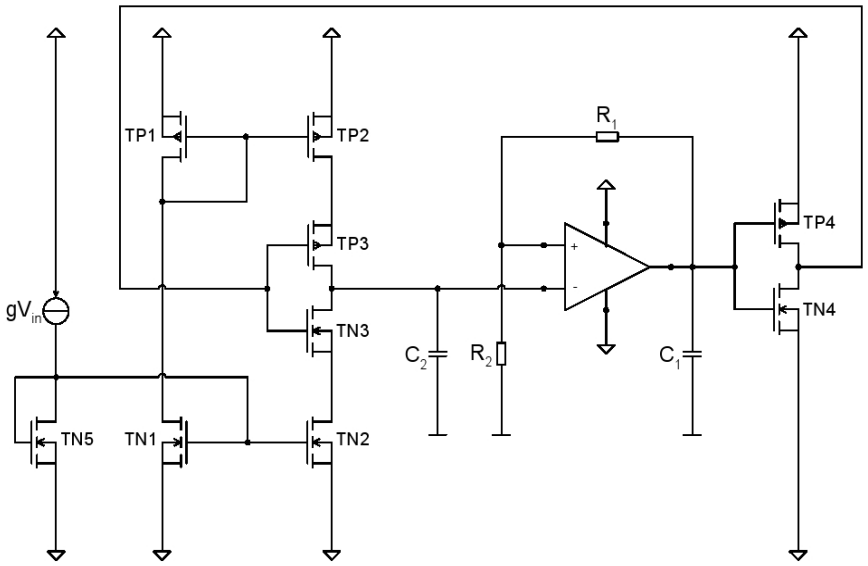

Figure 7.12: Voltage controlled oscillator (VCO) circuit.

study the influence of a time-variant perturbation in C_2 on the frequency. This perturbation is modeled as an input signal. In this example we show that, in addition to conformance relation, using this framework, we can test a property of the input/output relation. More precisely, for a given input sequence, we want compare the observation sequences of the specification $\mathcal{A}$ and those of the system under test $\mathcal{A}_s$, with respect to a property. As an example, the oscillating period $T \pm \delta$ of a variable $x_1 = v_{C_2}$ can be expressed using a simple automaton with one clock y in Figure 7.13 (similar to a monitor automaton in [41]). The question is to know if given an oscillating trace in $\mathcal{A}$, its corresponding trace in $\mathcal{A}_s$ is 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 we are additionally interested in a property which states that all traces of the system under test oscillate with the period $T \pm \delta$. 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 observation sequences that cause a 'pass' verdict is empty, and therefore the remaining observation sequences (that do not cause a 'fail' verdict) causes a 'inconclusive' verdict.

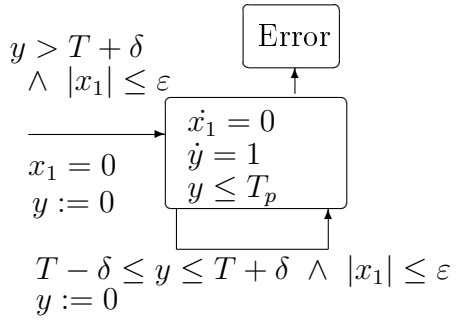

Figure 7.13: Automaton for an oscillation specification.

Results. We consider a constant input voltage $u_{in} = 1.7$. 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 [42]. Figure 7.15 shows the observation sequence projected on v_{C_1} and v_{C_2}. We note that the number of generated states was 30000 with 5 sampled input values and the computation time was 14mn. In this experiment, the coverage measure was defined on the projection of the state space on v_{C_1} and v_{C_2}, and not on the full-dimensional state space.

7.4 Delta-Sigma circuit

In this section we treat a mixed-signal circuit, which is a Delta-Sigma modulator [3], This circuit is a very popular technique to perform analog to digital conversion. The principles of Delta-Sigma modulation [18] can be described by the following points:

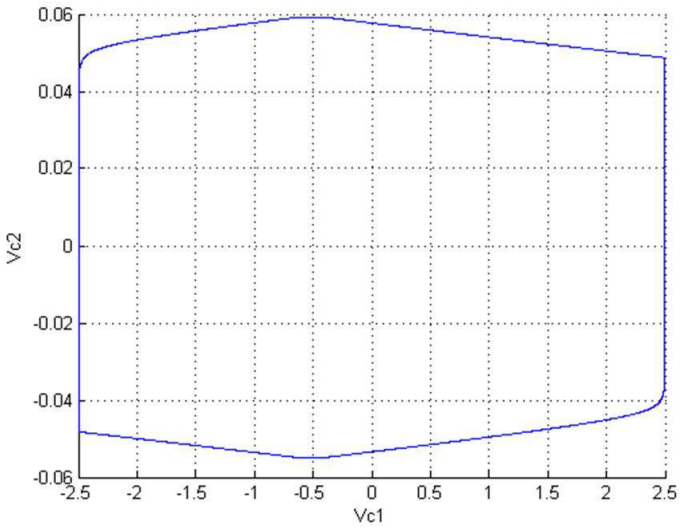

Figure 7.14: Variables v_{C_1} and v_{C_2} projection (without perturbation).

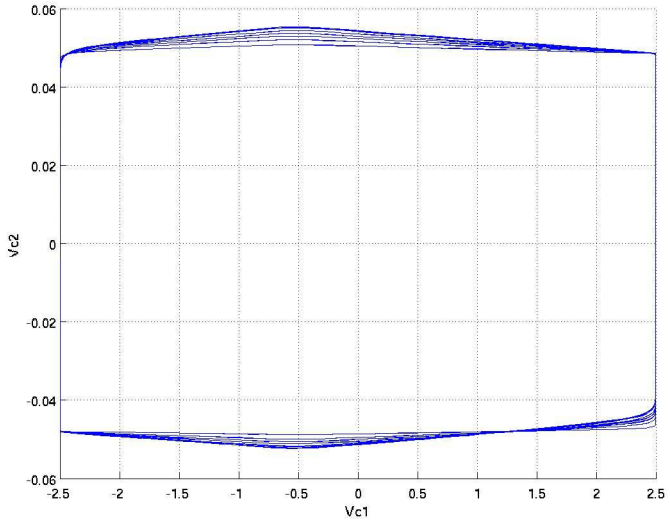

Figure 7.15: Variables v_{C_1} and v_{C_2} projection.

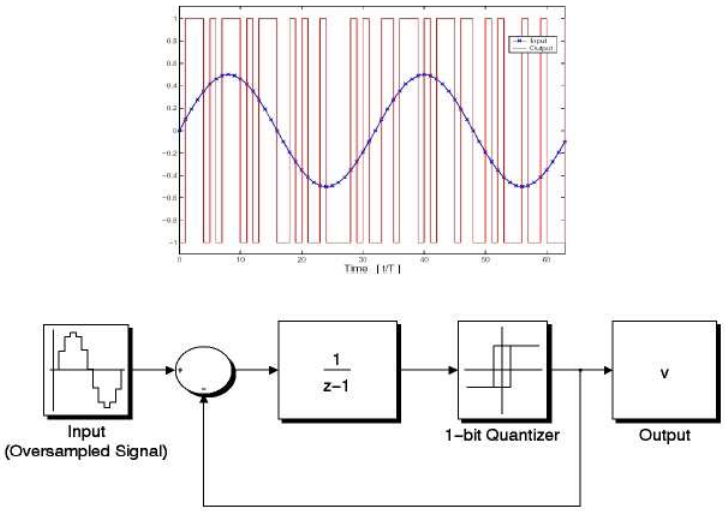

Figure 7.16: A first order Delta-Sigma modulator and an example of an input-output plot.

- Anti-aliasing: used to ensure that the signal band width lies within a given range $[-f_b, f_b]$,
- Oversampling or sampling at a frequency greater than the Nyquist rate $2 \times f_b$,
- Noise shaping so that the quantization error is 'pushed' toward high frequencies outside the bandwidth of interest,
- Quantization, typically on few bits.

Figure 7.16 shows how the Delta-Sigma modulation works. 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 which is the difference between the input and the output of the quantizer is fed back with negative gain and 'accumulated' in the integrator $\frac{1}{z-1}$. Then, when the accumulated error

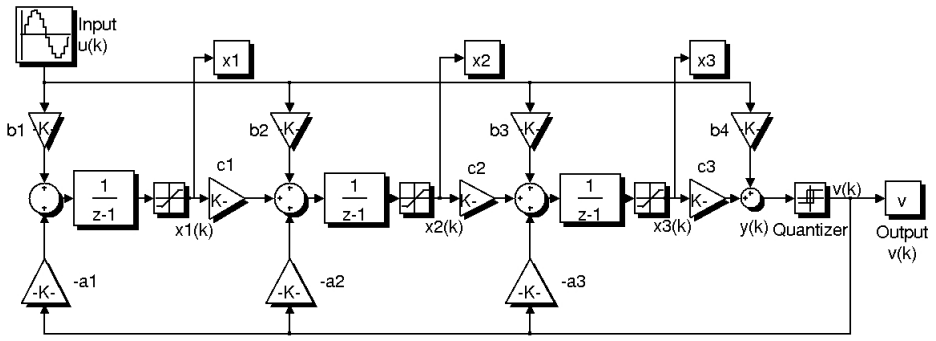

Figure 7.17: Model of a third-order modulator: Saturation blocks model saturation of the integrator.

reaches a certain threshold, the quantizer switches the value of the output to -1 for some time, which reduces the mean of the quantization error.

A third-order Delta-Sigma modulator is modeled as a hybrid automaton, shown in Figure 7.18. It is called third-order since it uses a third order filter to process noise.

Higher-order modulators achieve better performance but induce stability problems. A modulator is said to be stable if under a bounded input, the states of its integrators are bounded. Stability analysis for such circuits is still a challenging problem [3], due to the presence of two sources of non-linearities: saturation and quantization. The discrete-time dynamics of the system is as follows:

$$x(k+1) = Ax(k) + bu(k) - sign(y(k))a, \tag{7.2}$$

$$y(k) = c_3 x_3(k) + b_4 u(k), \tag{7.3}$$

where matrix A, vectors a and b are constants depending on the various gains of the model, $x(k) \in \mathbb{R}^3$ represents 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

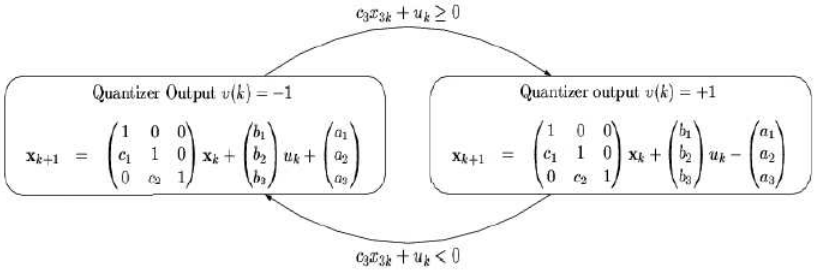

Figure 7.18: A hybrid automaton model of the Delta-Sigma modulator.

system dynamics is affine continuous. A modulator is stable if under a bounded input, the states of its integrators are bounded.

Results. We are intersted in testing the the stability property of the modulator. 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 7.19 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 $(\sup x_1(k))_k$ 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 [31] using an optimization-based method.

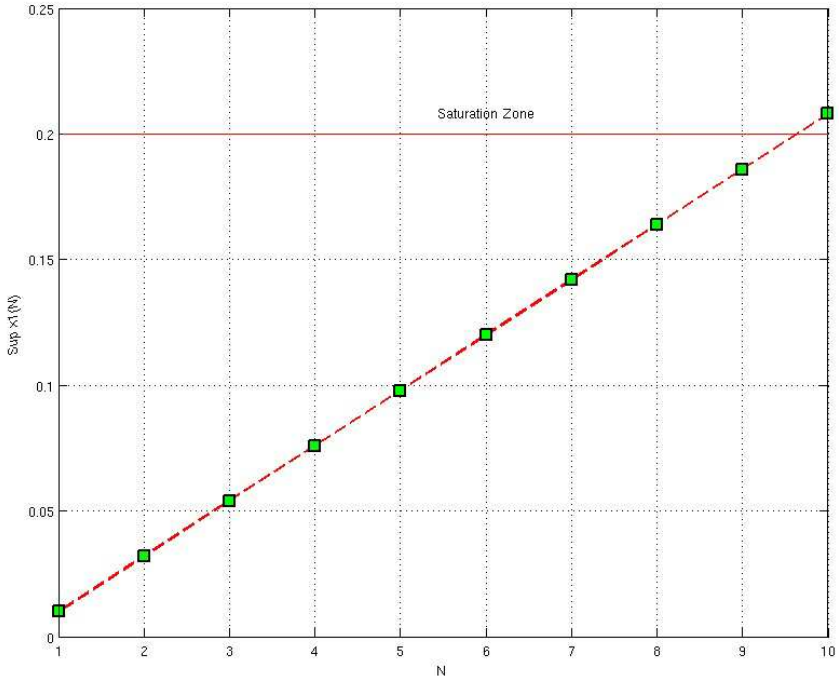

Figure 7.19: Test generation result for the Delta-Sigma circuit.

Chapter 8

Conclusion

8.1 Contributions

As we mentioned in the introduction of the thesis, model-based conformance testing has not been much developed for hybrid systems. This thesis is concerned with this problem and its main contributions can be summarized as follows.

In this thesis, we proposed a formal framework for conformance testing of hybrid systems. This framework uses the commonly-accepted hybrid automaton model. Furthermore, we developed a test generation algorithm, which exploits the ideas from robotic path planning. Another

We have adapted the hybrid automata as formal model for describing the system under test, believing that still the suitable language for modeling any complex system combining continuous and discrete parts.

The main contributions of the thesis can be summarized as follows:

Conformance testing framework and test generation.

- We proposed a formal framework for conformance testing of hybrid systems. This framework uses the commonly-accepted hybrid automaton model. The framework is defined according to the international

standard for formal conformance testing [95]. This framework allows, on one hand, to formally reason about the conformance relation between a system under test and a specification, and on the other hand, to develop test generation algorithms.

- Besides the main concepts in the framework of conformance testing, we addressed the problem of defining test coverage measures. We proposed two novel coverage measures, which not only are useful as a criterion to evaluate testing quality but also can be used to guide the test generation process in order to produce test cases with good coverage

- We developed a number of coverage-guided test generation algorithms for hybrid systems. These algorithms are based on a combination of the ideas from robotic path planning, equidistribution theory, algorithmic geometry, and numerical simulation.

Tool development and applications. We have implemented a tool for conformance testing of hybrid systems, called **HTG**. The input of our tool is a hybrid automaton described using a textual language input. The core of the tool is the implementation of the coverage-guided test case generation algorithm and the methods for estimating coverage measures. 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

8.2 Future work

The theoretical and practical results obtained in this thesis open a number of promising directions for future research:

Testing.

- First, we are interested in enriching our framework to capture partial observability, sensor and actuator imprecision.
- Convergence rate of the exploration in the test generation algorithm is another interesting theoretical problem to tackle. This problem is particular hard especially in the verification context where the system is subject to uncontrollable inputs.

Application to analog and mixed signal circuits. We intend to apply the results of this research to validation of analog and mixed-signal circuits, a domain where testing is a widely used technique. The current version of the tool is not yet as a general purpose as we would like. A number of issues remain to resolve in order to increase the applicability of the tool towards industrial systems. First of all, an efficient and reliable simulation method is key. The state-of-the-art SPICE simulator is prone to convergence problems when the dynamics of the circuit has fast variations caused by components with stiff characteristics. Currently, we are working in collaboration with researchers at INRIA Rhône-Alpes, and a topic of our undergoing research is to integrate in our test generation tool their simulation algorithms based on *the non-smooth approach* [4]. The applications of these simulation algorithms to mechanical systems have already been proved successful, and they are now being adapted to electrical systems. On the other hand, we are also interested in a tool for automatic generation of hybrid automata from commonly-used circuit descriptions, such as SPICE netlists or MATLAB/Simulink.

Application to related problems. The test generation techniques developed in this thesis can also be applied to two related problems, which are of particular interest:

- Systematic simulation [56]. This involves verifying that a given model satisfies a property by performing a number of simulations guided by some criteria, such as optimality, coverage. Systematic simulation can be thought of as a special case of testing, where not only the inputs of the systems but also the external disturbances are controllable by

the simulation algorithms. This approach can be used as a good compromise between exhaustive verification (which suffers from the state explosion problem and ad-hoc simulation which does not provide coverage guarantee.

- Run-time verification (or monitoring) [89, 60, 103, 82]. This involves observing the behavior of a real system during its execution and check whether the behavior satisfies a given property. Monitoring is also a special case of testing, where the tester is passive, that is it can only observe the output of the system but cannot control the inputs.

We thus intend to include new functionalities in our test generation tool in order to address the above problems.

Bibliography

[1] *L. O. Chua, C. A. Desoer, and E. S. Kuh. Linear and nonlinear circuits*, chapter 7. McGraw-Hill Book Company, 1987.

[2] *Ernst Hairer and Gerhard Wanner. Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems.*, chapter IV.8. Springer Series in Computational Mathematics 14, 1991.

[3] *B. Pérez-Verdú and F. Medeiro and A. Rodríguez-Vázquez. Top-Down Design of High-Performance Sigma-Delta Modulators*, chapter 2. Kluwer Academic Publishers, 2001.

[4] 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, Vienna, 2006 ISBN 3-901608-30-3.

[5] Thierry Jéron Ahmed Khoumsi and Hervé Marchand. Test cases generation for nondeterministic real-time systems. In *Formal Approaches to Software Testing*, 2004.

[6] 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*, 138:3–34, 1995.

[7] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, LNCS 736, pages 209–229. Springer-Verlag, 1993.

[8] 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.

[9] Rajeev Alur, Calin Belta, Vijay Kumar, Max Mintz, George J. Pappas, Harvey Rubin, and Jonathan Schug. Biocomputation: modeling and analyzing biomolecular networks. *Comput. Sci. Eng.*, 4(1):20–31, 2002.

[10] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[11] Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. In *IEEE Real-Time Systems Symposium*, pages 2–11, 1993.

[12] Marco Antoniotti and Aleks Göllü. Shift and smart-ahs: a language for hybrid system engineering modeling and simulation. In *DSL'97: Proceedings of the Conference on Domain-Specific Languages on Conference on Domain-Specific Languages (DSL), 1997*, pages 14–14, Berkeley, CA, USA, 1997. USENIX Association.

[13] Array. Automotive engine control and hybrid systems: Challenges and opportunities. *Proceedings of the IEEE*, 88(7):888–912, July 2000.

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

[15] Euene Asarin, Thao Dang, and Oded Maler. The d/dt tool for verification of hybrid system, 2002.

[16] J.-P. Aubin, J. Lygeros, M. Quincampoix, S. Sastry, and N. Seube. Impulse differential inclusions: a viability approach to hybrid systems. *IEEE Transactions on Automatic Control*, 47:2–20, January 2002.

[17] B. Ayari, N. Ben Hamida, and B. Kaminska. Automatic test vector generation for mixed-signal circuits. In *EDTC '95: Proceedings of the 1995 European conference on Design and Test*, page 458, Washington, DC, USA, 1995. IEEE Computer Society.

[18] 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.

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

[20] N. Ben-Hamida, K. Saab, D. Marche, and B. Kaminska. A perturbation based fault modeling and simulation for mixed-signal circuits. In *ATS '97: Proceedings of the 6th Asian Test Symposium*, page 182, Washington, DC, USA, 1997. IEEE Computer Society.

[21] 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.

[22] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[23] Amit Bhatia and Emilio Frazzoli. Incremental search methods for reachability analysis of continuous and hybrid systems. In *HSCC*, pages 142–156, 2004.

[24] Oleg Botchkarev and Stavros Tripakis. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In *HSCC '00: Proceedings of the Third International Workshop on Hybrid Systems: Computation and Control*, pages 73–88, London, UK, 2000. Springer-Verlag.

[25] M. Branicky, V. Borkar, and S. Mitter. A unified framework for hybrid control: model and optimal control theory, 1998.

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

[27] Bernhard Burdiek. Generation of optimum test stimuli for nonlinear analog circuits using nonlinear programming and time-domain sensitivities.

[28] Rachel Cardell-Oliver. Conformance test experiments for distributed real-time systems. In *ISSTA '02: Proceedings of the 2002 ACM SIGSOFT international symposium on Software testing and analysis*, pages 159–163, New York, NY, USA, 2002. ACM Press.

[29] P. Cheng and S. LaValle. Resolution complete rapidly-exploring random trees, 2002.

[30] Alongkrit Chutinan and Bruce H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. *Lecture Notes in Computer Science*, 1569:76–??, 1999.

[31] Thao Dang, Alexandre Donzé, and Oded Maler. Verification of analog and mixed-signal circuits using hybrid system techniques. In *FMCAD*, pages 21–36, 2004.

[32] Thao Dang and Tarik Nahhal. Randomized simulation of hybrid systems for circuit validation. In *Forum on Specification and Design Languages*, pages 9–14, 2006.

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

[34] Benjamin Doerr, Michael Gnewuch, and Anand Srivastav. Bounds and constructions for the star-discrepancy via δ-covers. *J. Complex.*, 21(5):691–709, 2005.

[35] Michael Drmota and Robert F. Tichy. *Sequences, discrepancies and applications.* Lecture Notes in Mathematics. 1651. Berlin: Springer. xiii, 503 p. DM 126.00; öS 919.80; sFr. 111.00 , 1997.

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

[37] M. Egerstedt, K. Johansson, J. Lygeros, and S. Sastry. Behavior based robotics using regularized hybrid automata, 1999.

[38] 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.

[39] Henri Faure. Discrepance de suites associees à un système de numeration., 1978.

[40] S Fortune. A sweepline algorithm for voronoi diagrams. In *SCG '86: Proceedings of the second annual symposium on Computational geometry*, pages 313–322, New York, NY, USA, 1986. ACM Press.

[41] Goran Frehse, Bruce H. Krogh, Rob A. Rutenbar, and Oded Maler. Time domain verification of oscillator circuit properties. *Electr. Notes Theor. Comput. Sci.*, 153(3):9–22, 2006.

[42] 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.

[43] 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.

[44] Naim Ben Hamida, Khaled Saab, David Marche, Bozena Kaminska, and Guy Quesnel. Limsoft: Automated tool for design and test integration of analog circuits. *itc*, 00:571, 1996.

[45] P. Hartman. *Ordinary Differential Equations*. Wiley, 1964.

[46] W. Heemels, B. De Schutter, and A. Bemporad. Equivalence of hybrid dynamical models, 2001.

[47] T. A. Henzinger. The theory of hybrid automata. In *LICS '96: Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, page 278, Washington, DC, USA, 1996. IEEE Computer Society.

[48] 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.

[49] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.

[50] F. Hickernell, I. Sloan, and G. Wasilkowski. On tractability of weighted integration over bounded and unbounded regions in r, 2003.

[51] Aicke Hinrichs. Covering numbers, vapnik-červonenkis classes and bounds for the star-discrepancy. *J. Complex.*, 20(4):477–483, 2004.

[52] M.W. Hirsch and S. Smale. *Differential Equations, Dynamical Systems and Linear Algebra*. Academic Press, 1974.

[53] R. Horowitz and P. Varaiya. Control design of an automated highway system, 2000.

[54] ISO/IEC.Transfer and Management for OSI. Framework: Formal methods in conformance testing. 1996.

[55] K. Johansson, J. Lygeros, S. Simic, and J. Zhang. Dynamical properties of hybrid automata, 2000.

[56] Jim Kapinski, Bruce H. Krogh, Oded Maler, and Olaf Stursberg. On systematic simulation of open continuous systems. In *HSCC*, pages 283–297, 2003.

[57] H. Kerkhoff, R. Tangelder, H. Speek, and N. Engin. Mismatch: A basis for semi-automatic functional mixed-signal test-pattern generation. In *IEEE Int. Conf. on Electronics, Circuits, and Systems*, 1996.

[58] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration graphs, a class of decidable hybrid systems. In P.J. Antsaklis, W. Kohn, M. Lemmon, A. Nerode, and S. Sastry, editors, *Proc. of Workshop on Theory of Hybrid Systems*, LNCS 736, pages 179–208. Springer-Verlag, 1992.

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

[60] Moonjoo Kim, Insup Lee, Usa Sammapun, Jangwoo Shin, and Oleg Sokolsky. Monitoring, checking, and steering of real-time systems.

[61] Stefan Kowalewski and Heinz Treseler. Verdict - a tool for model-based verification of real-time logic process controllers. In *WPDRTS '97: Proceedings of the 1997 Joint Workshop on Parallel and Distributed Real-Time Systems (WPDRTS / OORTS '97)*, page 217, Washington, DC, USA, 1997. IEEE Computer Society.

[62] 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.

[63] M. Kvasnica, P. Grieder, and M. Baotić. Multi-Parametric Toolbox (MPT), 2004.

[64] K. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Software Tools for Technology Transfert*, 1(1), 1997.

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

[66] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University, Oct. 1998.

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

[68] 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.

[69] Bengt Lennartson, Michael Tittus, Bo Egardt, and Stefan Pettersson. Hybrid systems in process control.

[70] S. Lindemann and S. LaValle. Current issues in sampling-based motion planning, 2004.

[71] 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.

[72] Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. Hybrid I/O automata. In *82*, page 16. Centrum voor Wiskunde en Informatica (CWI), ISSN 0169-118X, 31 1995.

[73] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, volume 600, pages 447–484, Mook, The Netherlands, 3–7 June 1991. Springer-Verlag.

[74] H. N. Mhaskar. On the tractability of multivariate integration and approximation by neural networks. *J. Complex.*, 20(4):561–590, 2004.

[75] Ian Mitchell and Claire Tomlin. Level set methods for computation in hybrid systems. In *HSCC '00: Proceedings of the Third International Workshop on Hybrid Systems: Computation and Control*, pages 310–323, London, UK, 2000. Springer-Verlag.

[76] Ian M. Mitchell and Jeremy A. Templeton. A toolbox of hamilton-jacobi solvers for analysis of nondeterministic continuous and hybrid systems. In *Hybrid Systems Computation and Control*, pages 480–494. Springer-Verlag, 2005.

[77] A. Stephen Morse. *Control Using Logic-Based Switching*, chapter 69-114. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[78] Tarik Nahhal and Thao Dang. Guided randomized simulation. In W. Damm and H. Hermanns, editors, *International Conference on Hybrid Systems: Computation and Control (HSCC)*, volume 4416, pages 731–735, Berlin, Germany, 2007. Lecture Notes in Computer Science, Springer-Verlag Heidelberg.

[79] Tarik Nahhal and Thao Dang. Test coverage for continuous and hybrid systems. In W. Damm and H. Hermanns, editors, *International Conference on Computer Aided Verification (CAV)*, volume 4590, pages 468–481, Berlin, Germany, 2007. Lecture Notes in Computer Science, Springer-Verlag Heidelberg.

[80] EH. Niederreiter. Discrepancy and convex programming. *Ann. Mat. Pura Appl*, 93:89–97, 1972.

[81] Brian Nielsen and Arne Skou. Automated test generation from timed automata. In *TACAS 2001: Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 343–357, London, UK, 2001. Springer-Verlag.

[82] D. Nickovic O. Maler. Monitoring temporal properties of continuous signals. In *FORMATS/FTRTFT'04,*, LNCS 3253, pages 152–166. Springer, 2004.

[83] David L. Pepyne and Christos G. Cassandras. Modeling, analysis, and optimal control of a class of hybridsystems. *Discrete Event Dynamic Systems*, 8(2):175–201, 1998.

[84] Erion Plaku, Lydia E. Kavraki, and Moshe Y. 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.

[85] A. Puri and P. Varaiya. Decidability of hybrid systems with rectangular differential inclusions. In *Computer Aided Verification, CAV'94*, LNCS 816, pages 54–104. Springer-Verlag, 1994.

[86] C.-J. Richard Shi and Michael W. Tian. Automatic test generation for linear analog circuits under parameter variations. In *ASP-DAC*, pages 501–506, 1998.

[87] S. J. Spinks, C. D. Chalk, I. M. Bell, and M. Zwolinski. Generation and verification of tests for analog circuits subject to process parameter deviations. *J. Electron. Test.*, 20(1):11–23, 2004.

[88] Jan Springintveld, Frits Vaandrager, and Pedro R. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 254(1–2):225–257, 2001.

[89] L. Tan, J. Kim, and I. Lee. Testing and monitoring model-based generated program, 2003.

[90] 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.

[91] E. Thiémard. Computing Bounds for the Star Discrepancy. *Computing*, (65):169–186, 2000. PRO 2000.09.

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

[93] C. Tomlin, G. Pappas, and S. Sastry. Conflict resolution for air traffic management : A study in muti-agent hybrid systems, 1998.

[94] F. D. Torrisi and A. Bemporad. Hysdel-a tool for generating computational hybrid models for analysis and synthesis problems. *Control Systems Technology, IEEE Transactions on*, 12(2):235–249, 2004.

[95] 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, The Netherlands, 1994. North-Holland Publishing Co.

[96] 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.

[97] A. van der Schaft and J. Schumacher. Complementarity modeling of hybrid systems, 1998.

[98] W. Verhaegen, G. Van der Plas, and G. Gielen. Automated test pattern generation for analog integrated circuits. In *VTS '97: Proceedings of the 15th IEEE VLSI Test Symposium (VTS'97)*, page 296, Washington, DC, USA, 1997. IEEE Computer Society.

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

[100] Chao Yan. *Coho: A Verification Tool for Circuit Verification by Reachability Analysis.* PhD thesis, University of British Columbia, 2003.

[101] A. Yershova, L. Jaillet, T. eon, and S. LaValle. Dynamic-domain rrts: Efficient exploration by controlling the sampling domain, 2005.

[102] S. Yovine. Kronos: A verification tool for real-time systems. *Software Tools for Technology Transfer*, 1(1):123–133, 1997.

[103] Mohamed H. Zaki, Sofiène Tahar, and Guy Bois. A practical approach for monitoring analog circuits. In *GLSVLSI '06: Proceedings of the 16th ACM Great Lakes symposium on VLSI*, pages 330–335, New York, NY, USA, 2006. ACM Press.