Form Methods Syst Des (2009) 34: 183–213
DOI 10.1007/s10703-009-0066-0

Coverage-guided test generation for continuous and hybrid systems

Thao Dang · Tarik Nahhal

Published online: 7 February 2009

Abstract In this paper, we describe a formal framework for conformance testing of continuous and hybrid systems, using the international standard 'Formal Methods in Conformance Testing' FMCT. We propose a novel test coverage measure for these systems, which is defined using the star discrepancy notion. This coverage measure is used to quantify the validation 'completeness'. It is also used to guide input stimulus generation by identifying the portions of the system behaviors that are not adequately examined. We then propose a test generation method, which is based on a robotic motion planning algorithm and is guided by the coverage measure. This method was implemented in a prototype tool that can handle high dimensional systems (up to 100 dimensions).

Keywords Hybrid systems · Model-based testing · Conformance testing · Test coverage · Test generation

1 Introduction

Hybrid systems have been recognized as a high-level model appropriate for embedded systems, since this model can describe, within one framework, the logical part and the continuous part of an embedded system.[1] Due to the gap between the capacity of exhaustive formal verification methods and the complexity of embedded systems in practice, testing is still the most commonly used validation method in industry. Its success is probably due to the fact that testing suffers less from the 'state explosion' problem. Indeed, the engineer can choose the "degree of validation" by the number of tests. In addition, this approach can be applied to the real system itself and not only to its model. Generally, testing of a reactive system is carried out by controlling the inputs and checking whether its behavior is as expected. Since it is impossible to enumerate all the admissible external inputs to the hybrid

[1] The reader is refered to proceedings of the conferences HSCC—Hybrid Systems: Computation and Control for more information about the recent developments.

T. Dang (✉) · T. Nahhal
VERIMAG, 2 avenue de Vignate, 38610 Gières, France
e-mail: Thao.Dang@imag.fr

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. For discrete systems, specified using programming languages or hardware design languages, some syntactic coverage measures can be defined, like exercising every statement or transition, 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. In fact, the continuous dynamics of a hybrid system is succinctly described by a set of differential equations, and this description, in comparison with program codes, often provides much less intuition about the system's behaviors. 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 stimulus form a kind of dense network in the reachable state space without too many big unexplored 'holes'.

In this work we adopt a model-based testing approach. This approach allows the engineer to perform validation during the design, where detecting and correcting errors on a model are less expensive than on an implementation. The main contributions of the paper can be summarized as follows. We define a formal framework for conformance testing of continuous and hybrid systems, using the international standard for formal conformance testing FMCT [26]. We propose a test coverage measure for these systems, which is defined using the star discrepancy notion from statistics. This coverage measure is used to quantify the validation 'completeness'. It is also used to guide input stimulus generation by identifying the portions of the system behaviors that are not adequately examined. We propose an algorithm for generating tests from hybrid automata. This algorithm is based on the RRT (Rapidly-exploring Random Tree) algorithm [15] from robotic motion planning and guided by the coverage measure.

The rest of the paper is organized as follows. We first describe our conformance testing framework and our test coverage measure. We then present the test generation algorithm and show how to use the coverage measure to guide the test generation process. We also prove the completeness property of the algorithm. Finally, we describe an implementation of the algorithm and some experimental results. Before concluding, we discuss related work.

2 Model

Conformance testing provides a means to assess the correctness of an implementation with respect to a specification by performing experiments on the implementation and observing its responses. When the specification is described by a formal model, the international standard 'Formal Methods in Conformance Testing' (FMCT) [26] provides a framework of conformance testing, which includes abstract concepts (such as conformance, test cases, test execution, test generation), and the requirements on these concepts.

In this work, following the spirit of FMCT, we are interested in developing a conformance testing framework for continuous and hybrid systems, using the hybrid automaton model [1]. Note that a continuous system can be modeled as a hybrid automaton with only one discrete state. A hybrid automaton is an automaton augmented with continuous variables that evolve according to some differential equations.

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

– $\mathcal{X}$ is the continuous state space and is a bounded subset of $\mathbb{R}^n$.

- Q is a (finite) set of locations (or discrete states).
- $E \subseteq Q \times Q$ is a set of discrete transitions.
- $F = \{F_q \mid q \in Q\}$ such that for each $q \in Q$, $F_q = (f_q, U_q)$ defines a differential equation:

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

  where $u(\cdot) \in \mathcal{U}_q$ is an admissible input function of the form $u : \mathbb{R}^+ \to U_q \subset \mathbb{R}^m$. We assume that all f_q are Lipschitz continuous.[2] The admissible input functions $u(\cdot)$ are piecewise continuous.
- $\mathcal{I} = \{\mathcal{I}_q \subseteq \mathcal{X} \mid q \in Q\}$ is a set of staying conditions.
- $\mathcal{G} = \{\mathcal{G}_e \mid e \in E\}$ is a set of guards such that for each discrete transition $e = (q, q') \in E$, $\mathcal{G}_e \subseteq \mathcal{I}_q$.
- $\mathcal{R} = \{\mathcal{R}_e \mid e \in E\}$ is a set of reset maps. For each $e = (q, q') \in E$, $\mathcal{R}_e : \mathcal{G}_e \to 2^{\mathcal{I}_{q'}}$ defines how x may change when $\mathcal{A}$ switches from q to q'.
- The initial state of the automaton is denoted by (q_{init}, x_{init}).

All the guard sets $\mathcal{G}_e$ and staying sets $\mathcal{I}_q$ are assumed to be compact. A *hybrid state* is a pair (q, x) where $q \in Q$ and $x \in \mathcal{X}$. The hybrid state space is $\mathcal{S} = Q \times \mathcal{X}$. In the rest of the paper, for brevity, we often use "state" to refer to a hybrid state. In location q, the evolution of the continuous variables is governed by $\dot{x}(t) = f_q(x(t), u(t))$.

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

Definition 2 (Continuous evolution) Given a 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 evolution* at the location q from the hybrid state (q, x) to (q, x'), iff $x' = \xi_{x,u(\cdot)}(h)$ and for all $t \in [0, h] : \xi_{x,u(\cdot)}(t) \in \mathcal{I}_q$, where $\xi_{x,u(\cdot)}(t)$ is the solution of the differential equation at the location q with the initial condition x and under the input $u(\cdot)$.

In other words, x' is reached from x under the input $u(\cdot)$ after exactly h time, and we say that $u(\cdot)$ is *admissible* starting at (q, x) for h time.

Definition 3 (Discrete evolution) Given a transition $e = (q, q') \in E$, $(q, x) \overset{e}{\rightarrow} (q', x')$ is a discrete evolution iff $x \in \mathcal{G}_e$ and $x' \in \mathcal{R}_e(x)$.

We say that (q', x') is reachable from (q, x) and the discrete transition e is admissible at (q, x). Unlike *continuous evolutions*, *discrete evolutions* are instantaneous, which means that they do not take time.

It is important to note that this model allows to capture *non-determinism* in both continuous and discrete dynamics. The non-determinism in continuous dynamics is caused be the

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

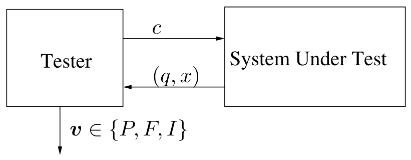
uncertainty in the input function. For example, when the input is used to model some external disturbances or modelling errors, we do not know the exact input function but only its range. The non-determinism in discrete dynamics by the fact that at some states it is possible for the system to stay at the current location or to switch to another one. In addition, multiple transitions can be enabled at some states. This non-determinism is useful for describing disturbances from the environment and imprecision in modelling and implementation. We assume that the hybrid automata we consider are non-Zeno.[3]

3 Conformance testing

In this section, we define the main concepts of our testing framework. Our testing goal is to make statements about the conformance relation between the behaviors of an implementation or, more generally, a system under test (SUT) and a specification. The specification is formal and is modeled by a hybrid automaton. The conformance will be defined as a relation $\preccurlyeq \subseteq \varXi \times HA$ where $\varXi$ is a set of SUTs of interest, and HA is a set of hybrid automata modeling the specifications of interest. The systems under test are physical systems, but it can be assumed that all the SUTs in $\varXi$ can be described by a class of formal models, which is a set HA_s of hybrid automata. It is important to note that we assume that a model for each SUT in $\varXi$ 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 $\preccurlyeq$ between the models of the SUTs and the specifications, that is $\preccurlyeq \subseteq HA_s \times HA$. Note that here we use the same notation $\preccurlyeq$ for the relation between the real SUT and the specification and the relation between the model of the SUT and the specification. A system under test $S_{ut} \in \varXi$ is said to *conform* to a specification $\mathcal{A} \in HA$ if and only if the model $\mathcal{A}_s \in HA_s$ of S_{ut} is related to $\mathcal{A}$ by $\preccurlyeq$, that is, $\mathcal{A}_s \preccurlyeq \mathcal{A}$.

The system under test often operates within some environment. In our testing framework, a tester plays the role of the environment and it performs experiments on the SUT in order to study the conformance relation between the SUT and the specification. Such an experiment is called a *test*, and its specification is called a *test case*. A set of test cases is called a *test suite*, and the process of applying a test to a system under test is called a *test execution*. The tester works as follows (see Fig. 1). It emits the control inputs to the SUT and measures the observation sequences in order to produce a verdict $v \in \{P, F\}$ where P means 'pass' (the observed behavior is allowed by the specification), F means 'fail' (the observed behavior is not allowed by the specification). We continue by giving a detailed description of conformance relation. The problem of how to perform test executions and derive verdicts is discussed at the end of this section.

Fig. 1 Test architecture



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

3.1 Conformance relation

Recall that the specification is modeled by a hybrid automaton $\mathcal{A}$ and the system under test SUT by another hybrid automaton $\mathcal{A}_s$. For brevity, when the context is clear, we often say 'the system under test' to mean the automaton $\mathcal{A}_s$. To define the conformance relation, we need the notions of *inputs and observations*.

3.1.1 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 input actions.

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

Discrete input actions The discrete transitions are partitioned into controllable corresponding to discrete control actions and uncontrollable corresponding to discrete disturbance actions. The tester emits a discrete control action to specify whether the system should take a controllable transition (among the enabled ones) or continue with the same continuous dynamics. In the former case, it can also control the values assigned to the continuous variables by the associated reset map. 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').

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

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

Definition 4 (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

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

The sequence $(q, x), (q_1, x_1), \ldots, (q_k, x_k)$ is called the *trace* starting at (q, x) under ω and is denoted by $\tau((q, x), \omega)$. The last state of $\tau((q, x), \omega)$ is denoted by $\text{last}(\tau((q, x), \omega))$.

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

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

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

To know whether a sequence of control actions is admissible, we 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 $\omega_c = c_0\, c_1$. After accepting the control action c_0 and all the disturbance input sequences admissible after c_0, the set of all possible successors of (q,x) is:

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

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

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

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

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

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

Definition 5 (Admissible control action sequence) A control action sequence ω_c is admissible starting at (q,x) iff $\Sigma(\omega_c,(q,x))$ is not empty. The set of traces starting at (q,x) after an admissible control action sequence ω_c is $\mathrm{Tr}((q,x),\omega_c) = \{\tau((q,x),\sigma) \mid \sigma \in \Sigma(\omega_c,(q,x))\}$.

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

3.1.2 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. In addition, we assume that $V_o(\mathcal{A}) \subseteq V_o(\mathcal{A}_s)$, which means that an observable continuous variable of $\mathcal{A}$ is also an observable variable of $\mathcal{A}_s$.

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 $\pi(x, V_o(\mathcal{A}))$. The projection can be then defined for a trace as follows. The projection of a trace $\tau = (q_0, x_0), (q_1, x_1), (q_2, x_2), \dots$ on $V_o(\mathcal{A})$ is

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

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

Definition 6 (Observation sequence) Let ω be an admissible control action sequence starting at the initial state (q_{init}, x_{init}) of $\mathcal{A}$. The set of *observation sequences* associated with ω is $S_{\mathcal{O}}(\mathcal{A}, \omega) = \{\pi(\tau, V_o(\mathcal{A})) \mid \tau \in \mathrm{Tr}((q_{init}, x_{init}), \omega)\}$.

3.1.3 Conformance relation

In the definition of the conformance relation between a system under test $\mathcal{A}_s$ and a specification $\mathcal{A}$, we assume that the set of all admissible control action sequences of $\mathcal{A}$ is a subset of that of $\mathcal{A}_s$, that is $S_{\mathcal{C}}(\mathcal{A}) \subseteq S_{\mathcal{C}}(\mathcal{A}_s)$. This assumption assures that the system under test can admit all the control action sequences that are admissible by the specification.

Definition 7 (Conformance) The system under test $\mathcal{A}_s$ is conform to the specification $\mathcal{A}$, denoted by $\mathcal{A} \preccurlyeq \mathcal{A}_s$, iff

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

Intuitively, the system under test $\mathcal{A}_s$ is conform to the specification $\mathcal{A}$ if under every admissible control action sequence, the set of observation sequences of $\mathcal{A}_s$ is included in that of $\mathcal{A}$. 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 $\mathcal{A}_s$. Detecting the cases where the physical SUT does not admit some inputs that are allowed by the specification requires the ability to identify the states of the system from the observations. We do not consider this problem in this work.

Note that we use the trace inclusion to define conformance relation. In the literature of conformance testing for discrete systems, more complex relations are considered, for example input-output conformance relation (see [25]).

3.2 Test cases and test executions

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

- The tester applies a test ζ to the system under test S_{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 $\mathrm{exec}(\zeta, S_{ut})$ which leads to an observation sequence, or a set of observation sequence if multiple runs of ζ are possible due to non-determinism. The above test execution process uses a number of implicit assumptions. First, observation measurements take zero time, and in addition, no measurement error is considered. Second, the tester is able to realize exactly the continuous input functions, which is often impossible in practice due to actuator imprecision. Under these assumptions, one can only test the conformance of *a model of the system under test* to the specification in discrete time. Considering these issues in order to address the actual testing of real systems under test is part of our future work.

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

We now discuss some important requirements for a test suite. A test suite T_s is called *complete* if for a given specification $\mathcal{A} \in HA$:

$$S_{ut} \preccurlyeq \mathcal{A} \iff S_{ut} \text{ passes } T_s. \tag{1}$$

This means that a complete test suite can distinguish exactly between all conforming and non-conforming systems. In practice, it is generally impossible to fulfill this requirement, which often involves executing an infinite test suite. A weaker requirement is *soundness*. A test suite is sound if a system does not pass the test suite, then the system is non-conforming. We can see that this requirement is weaker than completeness, since it corresponds only to the left-to-right implication in (1).

After defining all the important concepts, it now remains to tackle the problem of generating test cases from a specification model. In particular, we want the test suites to satisfy the *soundness requirement*. A hybrid automaton 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 the specification $\mathcal{A}$ and test the conformance of the system under test $\mathcal{A}_s$ with respect to this portion. The selection is done using a coverage criterion that we formally define in the next chapter. Hence, our testing problem is formulated as to automatically generate a set of test cases from the specification automaton to satisfy this coverage criterion.

4 Test coverage

Test coverage is a way to evaluate testing quality. More precisely, it is a way to relate the number of tests to carry out with the fraction of the system's behaviors effectively explored. As mentioned earlier, the classic coverage notions mainly used in software testing, such as statement coverage and branch coverage, path coverage (see for example [25, 29]), are not appropriate for the trajectories of continuous and hybrid systems defined by differential equations. However, geometric properties of the hybrid state space can be exploited to

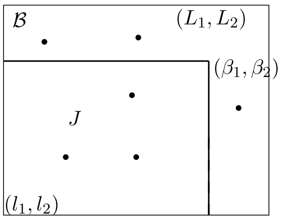


Fig. 2 Illustration of the star discrepancy notion

define a coverage measure which, on one hand, has a close relationship with the properties to verify and, on the other hand, can be efficiently computed or estimated. In this work, we are interested in *state coverage* and focus on a measure that describes how 'well' the visited states represent the reachable set of the system. This measure is defined using the *star discrepancy* notion in statistics, which characterises the uniformity of the distribution of a point set within a region. Note that the reachable sets of hybrid systems are often non-convex with complex geometric form, therefore considering only corner cases does not always cover the behaviors that are important for reachability properties, especially in high dimensions. Hence, for a fixed number of visited states (which reflects the computation cost to produce a test suite), we want the visited states to be equidistributed over the reachable set as much as possible, since this provides a good representation of all possible reachable states.

4.1 Star discrepancy

We first briefly recall the star discrepancy. The star discrepancy is an important notion in equidistribution theory as well as in quasi-Monte Carlo techniques (see for example [3]). Recently, it was also used in probabilistic motion planning to enhance the sampling uniformity [12].

Let P be a set of k points inside $\mathcal{B} = [l_1, L_1] \times \cdots \times [l_n, L_n]$. Let $\mathcal{J}$ be the set of all sub-boxes J of the form $J = \prod_{i=1}^{n}[l_i, \beta_i]$ with $\beta_i \in [l_i, L_i]$ (see Fig. 2). The local discrepancy of the point set P with respect to the sub-box J is defined as follows:

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

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

Definition 8 (Star discrepancy) The star discrepancy of a point set P with respect to the box $\mathcal{B}$ is defined as:

$$D^*(P, \mathcal{B}) = \sup_{J \in \mathcal{J}} D(P, J). \tag{2}$$

It is not hard to prove the following property of the star discrepancy [24].

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

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

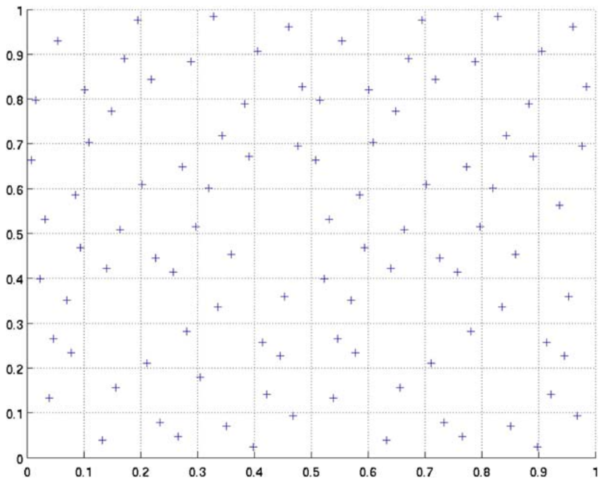


Fig. 3 Faure sequence of 100 points. Its star discrepancy value is 0.048

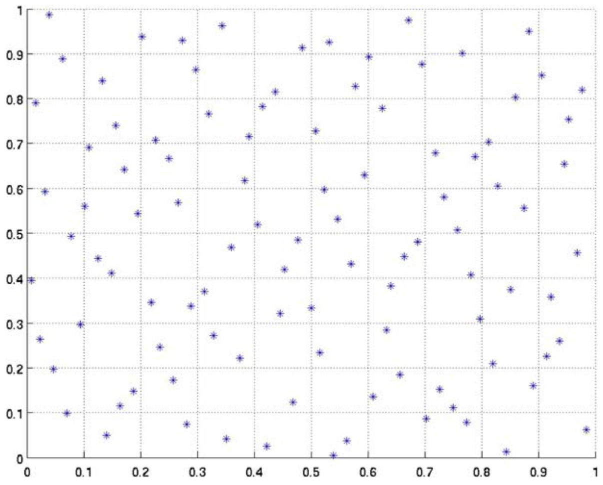


Fig. 4 Halton sequence of 100 points. The star discrepancy value is 0.05

Example To show an intuitive meaning of the star discrepancy, we use some sequences of 100 points inside a 2-dimensional unit box. The first example is the Faure sequence [13], a well-known low-discrepancy sequence (see Fig. 3). As we can observe from the figure, this set of points 'covers well' the box, in the sense that the points are well-equidistributed over the box. Its star discrepancy value is 0.048. The second example is the Halton sequence [28] shown in Fig. 4, which is also a well-known low discrepancy sequence. The value of the star discrepancy of the Halton sequence is about 0.050, indicating that the Faure sequence is more equidistributed than the Halton sequence. The star discrepancy values of these two sequences are however close, and indeed visually it is hard to see from the figures which one is better equidistributed. We now give another example which is a sequence of 100 points generated by a pseudo-random function provided by the C library system. This sequence is shown in Fig. 5, from which we can observe that this sequence is not well-equidistributed over the box. This is confirmed by its star discrepancy value 0.1. The star discrepancy is thus a meaningful measure that can characterize the uniformity quality of a point set distribution.

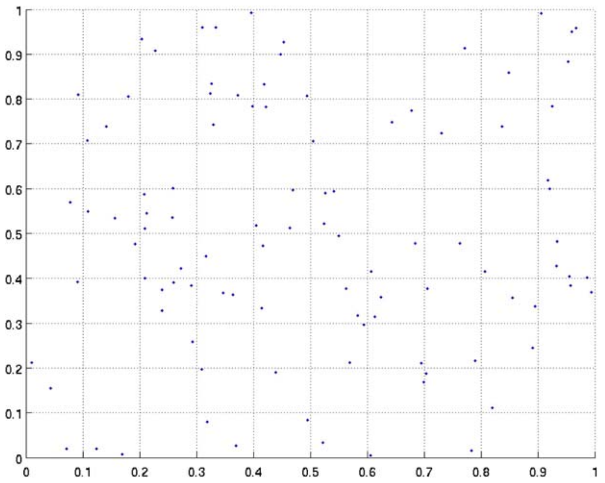


Fig. 5 A sequence of 100 points generated by a pseudo-random function in the C library. Its star discrepancy value is 0.1

This makes the star discrepancy suitable to be a test coverage measure for continuous and hybrid systems.

4.2 Coverage estimation

To evaluate the coverage of a set of states, we need to compute the star discrepancy of a point set, which is not an easy problem (see for example [7]). Many theoretical results for one-dimensional point sets are not generalizable to higher dimensions, and among the fastest algorithms, the one proposed in [7] has time complexity $\mathcal{O}(k^{1+d/2})$. In this work, we do not try to compute the star discrepancy but approximate it by estimating a lower and upper bound. These bounds as well as the information obtained from their estimation are then used to decide which parts of the state space have been 'well explored' and which parts need to be explored more. This estimation is done using a method published in [24]. Let us briefly describe this method for computing the star discrepancy $D^*(P, \mathcal{B})$ of a point set P w.r.t. a box $\mathcal{B}$. Although in [24] the box $\mathcal{B}$ is $[0, 1]^n$, we extended it to the case where $\mathcal{B}$ can be any full-dimensional box.

Intuitively, the main idea of this estimation method is to consider a finite box partition of the box $\mathcal{B}$, instead of considering an infinite number of all sub-boxes as in the definition of the star discrepancy. Let $\mathcal{B} = [l_1, L_1] \times \cdots \times [l_n, L_n]$. In what follows, we often call this box $\mathcal{B}$ the *bounding box*. We define a box partition of $\mathcal{B}$ as a set of boxes $\Pi = \{\boldsymbol{b}^1, \ldots, \boldsymbol{b}^m\}$ such that $\bigcup_{i=1}^m \boldsymbol{b}^i = \mathcal{B}$ and the interiors of the boxes $\boldsymbol{b}^i$ do not intersect. Each such box is called an *elementary box*. Given a box $\boldsymbol{b} = [\alpha_1, \beta_1] \times \cdots \times [\alpha_n, \beta_n] \in \Pi$, we define $\boldsymbol{b}^+ = [l_1, \beta_1] \times \cdots \times [l_n, \beta_n]$ and $\boldsymbol{b}^- = [l_1, \alpha_1] \times \cdots \times [l_n, \alpha_n]$ (see Fig. 6 for an illustration).

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

$$B(P, \Pi) = \max_{\boldsymbol{b} \in \Pi} \max \left\{ \frac{A(P, \boldsymbol{b}^+)}{k} - \frac{\mathrm{vol}(\boldsymbol{b}^-)}{\mathrm{vol}(\mathcal{B})}, \frac{\mathrm{vol}(\boldsymbol{b}^+)}{\mathrm{vol}(\mathcal{B})} - \frac{A(P, \boldsymbol{b}^-)}{k} \right\}, \tag{3}$$

$$C(P, \Pi) = \max_{\boldsymbol{b} \in \Pi} \max \left\{ \left| \frac{A(P, \boldsymbol{b}^-)}{k} - \frac{\mathrm{vol}(\boldsymbol{b}^-)}{\mathrm{vol}(\mathcal{B})} \right|, \left| \frac{A(P, \boldsymbol{b}^+)}{k} - \frac{\mathrm{vol}(\boldsymbol{b}^+)}{\mathrm{vol}(\mathcal{B})} \right| \right\}. \tag{4}$$

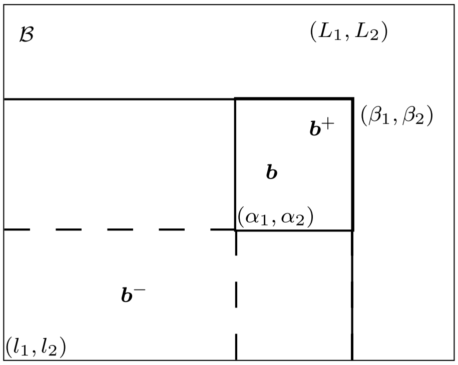


Fig. 6 Illustration of the boxes $\boldsymbol{b}^-$ and $\boldsymbol{b}^+$

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

$$W(\Pi) = \max_{\boldsymbol{b} \in \Pi} (\mathrm{vol}(\boldsymbol{b}^+) - \mathrm{vol}(\boldsymbol{b}^-))/\mathrm{vol}(\mathcal{B}). \tag{5}$$

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

4.3 Hybrid systems test coverage

Since a hybrid system can only evolve within the staying sets of the locations, we are interested in the coverage with respect to these sets. For simplicity we assume that all the staying sets are boxes.

Definition 9 (Test coverage) Let $\mathcal{P} = \{(q, P_q) \mid q \in Q \ \wedge \ P_q \subset \mathcal{I}_q\}$ be the set of states. The coverage of $\mathcal{P}$ is defined as:

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

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

If a staying set $\mathcal{I}_q$ is not a box, we can take the smallest oriented box that encloses it and apply the star discrepancy definition in (2) to that box after an appropriate coordinate transformation. We can see that a large value of $\mathrm{Cov}(\mathcal{P})$ indicates a good space-covering quality. If $\mathcal{P}$ is the set of states visited by a test suite, our objective is to maximize $\mathrm{Cov}(\mathcal{P})$.

5 Test generation

Our test generation is based on a randomized exploration of the reachable state space of the system. It is inspired by the Rapidly-exploring Random Tree (RRT) algorithm, which is a successful motion planning technique for finding feasible trajectories of robots in an environment with obstacles (see [15] for a survey). More precisely, we extend the RRT algorithm to hybrid systems. Furthermore, we combine it with a guiding tool in order to achieve a good coverage of the system's behaviors we want to test. To this end, we use the coverage measure defined in the previous section.

Algorithm 1 Test generation algorithm hRRT

```
$k = 1$
$\mathcal{T}^k.init(s_{init})$                                   ▷ $s_{init}$: initial state
**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** $k \geq k_{max}$
```

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

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

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

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

The algorithm terminates after some maximal number of iterations. Another possible termination criterion is that a satisfactory coverage value is reached. In the classic RRT algorithms, which work in a continuous setting, only x_{goal} needs to be sampled, and a commonly used sampling distribution of x_{goal} is uniform over $\mathcal{X}$. In addition, the point x_{near} is defined as a nearest neighbor of x_{goal} in some usual distance, such as the Euclidean distance. In our hRRT algorithm, the goal state sampling is not uniform and the function SAMPLING plays the role of guiding the exploration via a biased sampling of x_{goal}. This will be discussed in detail later, and in the following we show how to compute the other functions of the algorithm.

5.1 Hybrid distance and computation of neighbors

As mentioned earlier, in most versions of the RRT algorithm, where the state space is a subset of $\mathbb{R}^n$, the query of nearest neighbors often uses the Euclidean 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 the hRRT algorithm.

Given two hybrid states $s = (q, x)$ and $s' = (q', x')$, if they have the same discrete component, that is, $q = q'$, we can use some usual metric in $\mathbb{R}^n$, such as the Euclidean 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 useful definitions and notations.

Given two sets A and B in $\mathbb{R}^n$, we can define the *average distance between A and B*, denoted by $\overline{d}(A, B)$, as a distance (such as the Euclidean 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 with a set of vertices $V = \{v_1, \dots, v_J\}$, then the centroid of A is $v_c = \frac{1}{J}\sum_{j=1}^{J} v_j$.

Definition 10 (Average length of a path) Let $\gamma = (q_1, q_2), \dots, (q_{m-1}, q_m)$ be a discrete path in the hybrid automaton $\mathcal{A}$, the average length of γ, denoted by $\mathrm{len}(\gamma)$, is

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

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 11 (Average length of trajectories) Let $\gamma = (q_1, q_2), \dots, (q_{m-1}, q_m)$ be a discrete path from location $q_1 = q$ to $q_m = q'$ 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:

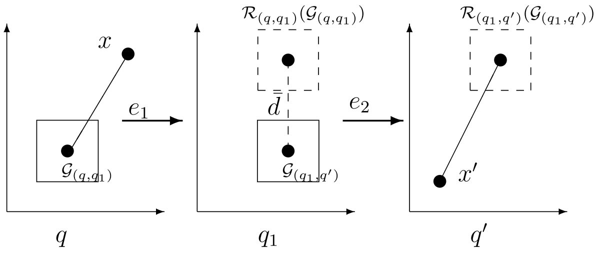
$$\mathrm{len}_\gamma(s, s') = \overline{d}(x, fG(\gamma)) + \mathrm{len}(\gamma) + \overline{d}(x', lR(\gamma))$$

where $fG(\gamma) = \mathcal{G}_{(q_1,q_2)}$ is the first guard of γ, and $lR(\gamma) = \mathcal{R}_{(q_{m-1},q_m)}(\mathcal{G}_{(q_{m-1},q_m)})$ is the set resulting from applying the reset map of the last transition to its guard set.

Example Figure 7 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.

Fig. 7 Illustration of average length of trajectory


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

– 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 Fig. 7 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'.

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

Definition 12 (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 norm in $\mathbb{R}^n$.
- If $q \neq q'$, there are two cases:
  - If $\Gamma(q, q') \neq \emptyset$, then $d_H(s, s') = \min_{\gamma \in \Gamma(q,q')} \mathrm{len}_\gamma(s, s')$. The path γ that minimizes $\mathrm{len}_\gamma(s, s')$ 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 in $\mathbb{R}^n$. In this work, we will use the Euclidean distance and the notation $\|\cdot\|$ denotes this distance.

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

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

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

5.2 Computing continuous and discrete successors

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

When the states s_{near} and s_{goal} are at different locations, let γ be the shortest path from s_{near} to s_{goal}. It is natural to make the system follow this path. Therefore, we want to steer the system from x_{near} towards the first guard of γ. In both of the two cases, one 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 sample a finite number of inputs and pick from this set a best input, that makes the system approach the boundary of the guard of γ as much as possible. In addition, we can prove that by appropriately sampling the input set, the completeness property of our algorithm is preserved (see Sect. 7). It is important to emphasize that the function CONTINUOUSSUCC needs to assure that the trajectory segment from x_{near} stays in the staying set of the current location.

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

5.3 Test cases and verdicts

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

6 Coverage-guided test generation

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

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

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

6.1 Location sampling

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

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

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

6.2 Continuous state sampling

We now show how to sample x_{goal}, assuming that we have already sampled a location $q_{goal} = q$. In the remainder of the paper, to give geometric intuitions, we often call a continuous state a point. In addition, since all the staying sets are assumed to be boxes, we denote the staying set $\mathcal{I}_q$ by the box $\mathcal{B}$ and denote the current set of visited points at the location q simply by P instead of P_q. Let k be the number of points in P. Let Π be a finite box partition of $\mathcal{B}$ that is used to estimate the star discrepancy of P. The sampling process consists of two steps. In the first step, we sample an elementary box $\boldsymbol{b}_{goal}$ from the set Π; in the second step we sample a point x_{goal} in $\boldsymbol{b}_{goal}$ uniformly, as shown in Algorithm 2. The elementary box sampling distribution in the first step is biased in order to optimize the coverage. Guiding is thus done via the goal box sampling process. The next section is devoted to this problem.

Algorithm 2 Continuous goal state sampling

```
procedure SAMPLING( )
    $\boldsymbol{b}_{goal} = \text{BOXSAMPLING}(\Pi)$
    $x_{goal} = \text{UNIFORMSAMPLING}(\boldsymbol{b}_{goal})$
return $x_{goal}$
end procedure
```

6.3 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 of the bounds.

6.3.1 Reducing the lower bound

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

$$\frac{\mathrm{vol}(\boldsymbol{b})}{\mathrm{vol}(\mathcal{B})} = \frac{A^*(\boldsymbol{b})}{k}$$

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

$$\Delta_A(\boldsymbol{b}) = A(P,\boldsymbol{b}) - A^*(\boldsymbol{b}). \tag{6}$$

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, using the formula (4), 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 Fig. 8). Thus, we define

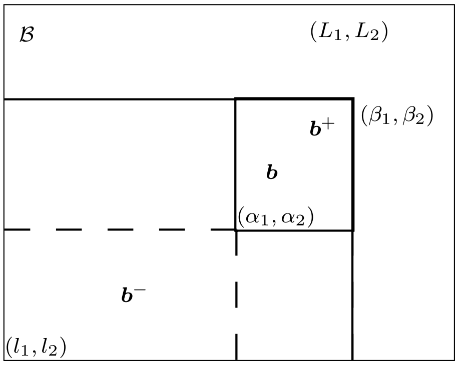


Fig. 8 Illustration of the boxes $\boldsymbol{b}^-$ and $\boldsymbol{b}^+$

a function reflecting the potential influence on the lower bound as follows:

$$\xi(\boldsymbol{b}) = \frac{1 - \Delta_A(\boldsymbol{b}^+)/k}{1 - \Delta_A(\boldsymbol{b}^-)/k}, \tag{7}$$

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 $1 - \Delta_A(\boldsymbol{b})/k > 0$.

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 $\boldsymbol{b}^-$.

6.3.2 *Reducing the upper bound*

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

$$B(P, \Pi) = \frac{1}{k} \max_{\boldsymbol{b} \in \Pi} \Delta_A^m(\boldsymbol{b}) \tag{8}$$

where $\Delta_A^m(\boldsymbol{b}) = \max\{\Delta_A^c(\boldsymbol{b}), \Delta_A^o(\boldsymbol{b})\}$. Using (6), we can write

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

and

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

Since the value of Δ_A^m is determined by comparing Δ_A^c with Δ_A^o. After straightforward calculations, the inequality $\Delta_A^c(\boldsymbol{b}) - \Delta_A^o(\boldsymbol{b}) \leq 0$ is equivalent to

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

Therefore,

$$\Delta_A^m(\boldsymbol{b}) = \begin{cases} \Delta_A^o(\boldsymbol{b}) & \text{if } \Delta_A(\boldsymbol{b}^+) + \Delta_A(\boldsymbol{b}^-) \leq 0, \\ \Delta_A^c(\boldsymbol{b}) & \text{otherwise.} \end{cases} \tag{9}$$

Again, we observe that adding a point in $\boldsymbol{b}$ increases $\Delta^c_A(\boldsymbol{b})$, but this does not affect $\Delta^o_A(\boldsymbol{b})$. To reduce $\Delta^o_A(\boldsymbol{b})$ we need to add points in $\boldsymbol{b}^-$. Hence, if $\boldsymbol{b}$ is a box in Π that maximizes Δ^c_A in (8), it is preferable not to add more points in $\boldsymbol{b}$ but in the other elementary boxes where the values of $\frac{1}{k}\Delta^m_A$ 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 mentioned earlier, if $\Delta^m_A(\boldsymbol{b}) = \Delta^c_A(\boldsymbol{b})$, increasing $\Delta^c_A(\boldsymbol{b})$ directly increases $\Delta^m_A(\boldsymbol{b})$. On the other hand, if $\Delta^m_A(\boldsymbol{b}) = \Delta^o_A(\boldsymbol{b})$, increasing $\Delta^c_A(\boldsymbol{b})$ may make it greater than $\Delta^o_A(\boldsymbol{b})$ and thus increase $\Delta^m_A(\boldsymbol{b})$, because small $|\Delta_A(P,\boldsymbol{b}^+) + \Delta_A(P,\boldsymbol{b}^-)|$ implies that $\Delta^c_A(\boldsymbol{b})$ is close to $\Delta^o_A(\boldsymbol{b})$.

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

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

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

7 Reachability completeness

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

Theorem 1 *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 this result can be found in [14, 15]. 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.

An arising question is whether our test generation algorithm, built upon the RRT algorithm, preserves this property. This is indeed true, 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 the free configuration space from the initial point (see for example [14]). In the hybrid state space $\mathcal{S} = Q \times \mathcal{X}$, generally not all the points are reachable from the initial state 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 [5]. However, the lack of the above-mentioned controllability assumption makes the proof more complicated.

We first introduce some assumptions about the hybrid systems we consider, under which the completeness property of our algorithms is proven.

7.1 Sufficient conditions for reachability completeness

We first observe from the completeness proof for continuous systems that the following conditions are sufficient for its validity: (C1) there is a non-null probability that each state in V^k is selected to be s^k_{near}, and (C2) there is a non-null probability that 'each reachable direction' is selected.

The satisfaction of these conditions guarantees that for any reachable state s there is a non-null probability that the new state s^{k+1}_{new} reduces the distance from the tree to s. 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. In addition, for a continuous system, that is a hybrid automaton with $Q = \{q\}$, if the control set U_q is finite and for each $u \in U_q$ the probability that the input value u is selected in each iteration k is positive, then the condition (C2) is satisfied. In addition, the reachable set needs to satisfy some neighborhood property which we will detail later in Assumption 1.

We now derive similar conditions for hybrid automata. A set of hybrid states $Y = \{(q, Y_q) \mid q \in Q \ \wedge \ Y_q \subseteq \mathcal{X}\}$ is said to have positive volume (or measure) if for *all* $q \in Q$ the volume (or measure) of Y_q is positive. We denote the volume of Y by $\mathrm{vol}(Y)$.

Definition 13 (Full coverage sampling condition) For every $k > 0$ and every set $Y \subseteq \mathcal{S}$ with positive volume, if the probability that in each iteration $s^k_{goal} \in Y$ is strictly positive, that is,

$$\forall k > 0\ \forall Y \subseteq \mathcal{S} \quad \Pr[s^k_{goal} \in Y] > 0,$$

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.

We can prove that for hybrid automata the full coverage sampling condition guarantees that the condition (C1) is satisfied. To guarantee the condition (C2) for such systems, in addition to the above-mentioned condition that each admissible continuous input value u has a non-null probability of being selected in each iteration, we need further conditions to guarantee that every reachable discrete transition has a non-null probability of being visited by the algorithm. This is formally described in the following.

We denote by R^j the reachable set defined recursively as follows:

$$R^j = \{s' \in \mathrm{Tr}(s, \omega) \mid \omega \in \tilde{S}_C \wedge s \in R^{j-1}\}; \quad j = 1, 2, \ldots, \tag{10}$$

with $R^0 = \{s_{init}\}$. In this definition, $\tilde{S}_C$ is the set of all admissible control sequences that either contain only continuous control actions, or start with a discrete control action which is followed by only continuous control actions. Recall that $\mathrm{Tr}(s, \omega)$ is the set of traces from s under the control action sequence ω. Intuitively, R^0 contains only the initial state, and R^j contains all the states reachable from R^{j-1} by one discrete transition (if possible) and then let time pass. We denote by R^j_e the set of all the states in R^j at which the transition e is enabled.

To state the completeness result, we need to introduce some notions. For a reachable state $s \in \mathcal{S}$, we define the set $\mathrm{Reach}_b(s)$ of reachable states leading to s as follows:

$$\mathrm{Reach}_b(s) = \{s' \in \mathcal{S} \mid \exists \omega \in S_C(\mathcal{A}) : s' \in \tau((q, x), \omega) \wedge s = \mathrm{last}(\tau((q, x), \omega))\}. \tag{11}$$

Given a state $s = (q, x) \in \mathcal{S}$ and a real number $\epsilon > 0$, we denote by $\mathrm{Ball}(s, \epsilon) = \{(q, x) \mid x \in \mathrm{Ball}_c(x, \epsilon)\}$ where $\mathrm{Ball}_c(x, \epsilon) \subseteq \mathbb{R}^n$ is the ball centered at x with radius ϵ.

Given a set $Y \subseteq \mathcal{S}$, if $\forall s \in Y\ \forall \epsilon > 0 : \mathrm{vol}(Y \cap \mathrm{Ball}(s, \epsilon)) > 0$, we say that Y satisfies the *positive volume neighborhood property*.

Definition 14 (Neighborhood) Given a state $s = (q, x) \in \mathcal{S}$ and a real number $\varepsilon > 0$, we define the ε-neighborhood of s as $\mathcal{N}(s, \varepsilon) = \{s' \in \mathcal{S} \mid d_H(s', s) \leq \varepsilon\}$.

From now on, we restrict our attention to a class of hybrid automata and prove the completeness property for such systems under the following assumptions.[4]

Assumption 1

- (H1) The function SAMPLING Algorithm 1 satisfies the full coverage sampling property.
- (H2) For all $q \in Q$, the set U_q is finite. In addition, for all $q \in Q$, for all $u \in U_q$, for all $k > 0$ $\Pr[u^k = u] > 0$ (where u^k is the value of the continuous input function that is applied at the kth iteration).
- (H3) For every reachable state s, the set $\mathrm{Reach}_b(s)$ of reachable states leading to s (defined in (11)) satisfies the *positive volume neighborhood property*.
- (H4) For all $j > 0$ and for all $e \in E$, if $R^j_e \neq \emptyset$ then R^j_e satisfies the *positive volume neighborhood property*.

As we will see later, the first three assumptions (H1), (H2), and (H3) guarantee the reachability completeness for the continuous dynamics of the hybrid automaton $\mathcal{A}$. The assumption (H4) guarantees the completeness for its discrete transitions, that is a transition e, if reachable, always has a chance to be visited by the algorithm.

7.2 Reachability completeness result

We proceed with the main result concerning the completeness property preservation of our test generation algorithm for the hybrid automata which satisfy Assumption 1.

[4]We use the letter 'H' to emphasize that these conditions are for hybrid automata.

Theorem 2 (Reachability completeness) *Let V^k be the set of states stored at the vertices of the tree $\mathcal{T}^k$ at the kth iteration. 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*

$$\lim_{k\to\infty} \Pr[\exists s' \in V^k : s' \in \mathcal{N}(s, \varepsilon)] = 1. \qquad (12)$$

The proof of this theorem is presented in Appendix.

In the classic RRT algorithms for continuous systems, in each iteration the starting point for expansion is a nearest neighbor of the goal point. Finding an exact nearest neighbor is expensive, especially in high dimensions. We can derive a variant of the RRT algorithm which has lower complexity. Indeed, to determine the starting states we can use *approximate nearest neighbors*, provided that the condition (H1) is satisfied.

8 Implementation and experimental results

The current implementation of the above described algorithms works for hybrid automata where continuous dynamics can be non-linear. The staying sets of the locations are boxes. The transition guard set are convex-polyhedra. The reset maps can be non-linear.

8.1 Implementation

In addition to the tree that is used to store the explored executions, to facilitate the computation of geometric operations, such as finding a neighbor, we store the points reachable by the dynamics at each location using a data structure similar to a k-d tree [19]. 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 ith coordinate of x). The additional information associated with a leaf is a set of visited points. Each node thus corresponds to an elementary box resulting from a hierarchical box-partition of the state space. The box of the root of the tree is $\mathcal{B}$. The tree and the partition of a 2-dimensional example is shown in Fig. 9, where the axes of the partitioning planes are specified by the horizontal and vertical bars inside the nodes. In the following, we briefly describe the main operations in the test generation algorithms. A detailed description of the implementation can be found in [21].

Approximate neighbors Since the computation of exact nearest neighbors is expensive (even in a continuous setting), we approximate a neighbor of x as follows: find the elementary box $\boldsymbol{b}$ which contains at least one visited point and, in addition, is closest to x (note that some elementary boxes may not contain any visited points). Then, we find a point in $\boldsymbol{b}$ which is closest to x. It is easy to see that $\boldsymbol{b}$ does not necessarily contain a nearest neighbor of x. We use this approximation because, on one hand the sampling distribution reflects the boxes we want to explore, and on the other hand, it has lower complexity. In addition, this approximation preserves the completeness.

Update the discrepancy estimation After adding a new point x, we need to update the estimation of the star discrepancy. 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 boxes $\boldsymbol{b}^-$ and $\boldsymbol{b}^+$. These boxes are indeed those which intersect with the box $B_x = [x_1, L_1] \times \cdots \times$

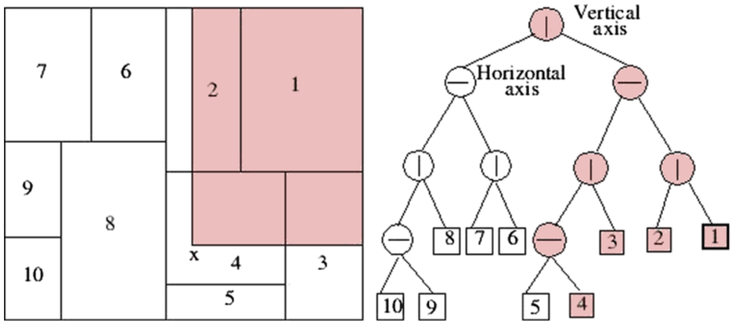


Fig. 9 Illustration of the update of the star discrepancy estimation

$[x_n, L_n]$. In addition, if $\boldsymbol{b}$ is a subset of B_x, the numbers of points in both $\boldsymbol{b}^+$ 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 Fig. 9, the box B_x is the dark rectangle, and the nodes of the trees visited in this search are drawn as dark circles.

Box splitting When the difference between the lower and upper bounds in the star discrepancy estimation is large, some boxes need to be split as indicated by (5). Additionally, splitting is also needed for efficiency of the neighbor computation.

8.2 Experimental results

We implemented the test generation algorithm using C++ in a prototype tool, and the results reported here were obtained by running the tool on a 1.4 GHz Pentium III.

8.2.1 Linear systems

First, to demonstrate the time efficiency of gRRT, 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 $[-3, 3]$ 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 values of the states generated by gRRT and the classic RRT algorithm are shown in Table 1, which indicates that our gRRT algorithm achieved a better coverage quality. These discrepancy values were computed for the final sets of visited states, using a partition optimal w.r.t. to the imprecision bound in (5). 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 2 shows the time efficiency of gRRT for linear systems of dimensions up to 100. The results obtained on a 2-dimensional system are visualized in Fig. 10.

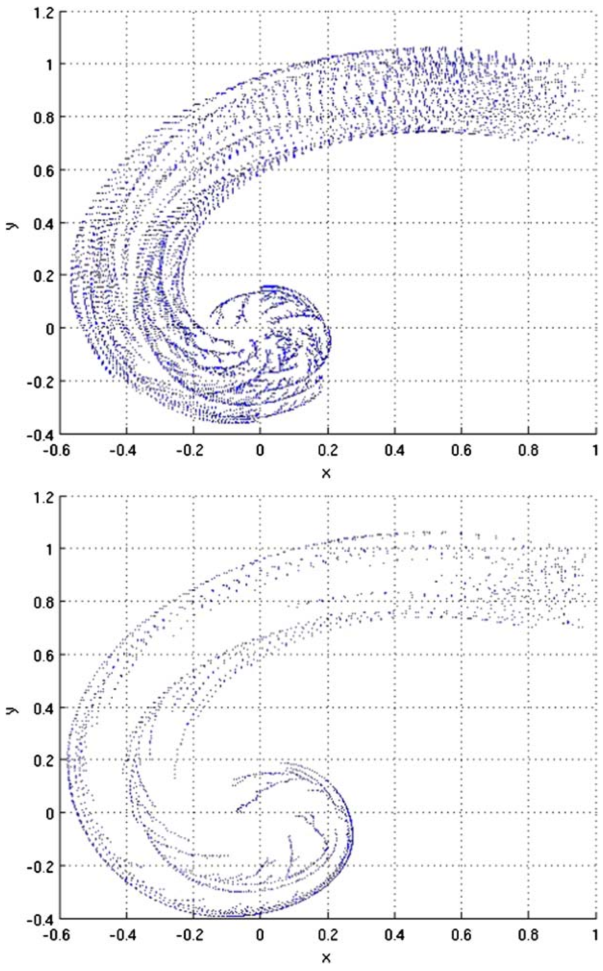


Fig. 10 Results (with the same number of visited states) obtained using the gRRT algorithm (*top*) and the RRT algorithm (*bottom*)

Table 1 Discrepancy results obtained for some linear systems using gRRT and RRT

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

8.2.2 Aircraft collision avoidance system

To illustrate the application of our algorithm to hybrid systems, we use the aircraft collision avoidance problem [18], which is a well-known benchmark in the hybrid systems literature. In this paper, the authors treated the problem of collision avoidance of two aircrafts. To show the scalability of our approach we consider the same model with N aircrafts.

Table 2 Computation time of gRRT for some linear systems

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

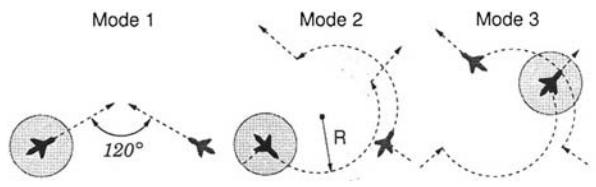
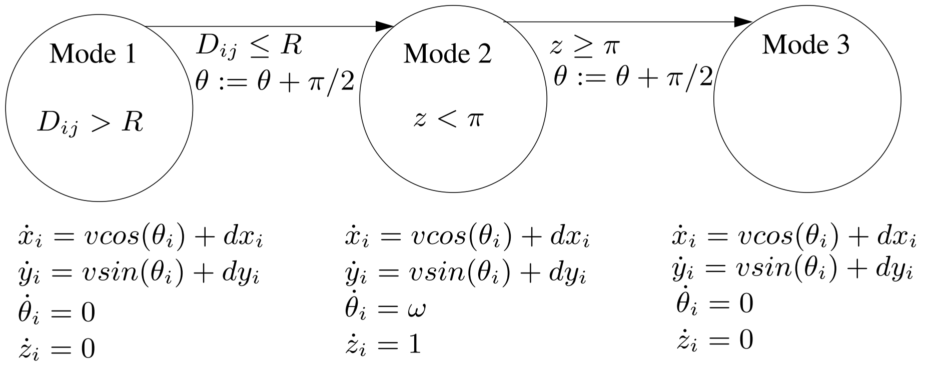


Fig. 11 Aircraft behavior in the three modes [18]


Mode 1
$D_{ij} > R$
$D_{ij} \leq R$
$\theta := \theta + \pi/2$
Mode 2
$z < \pi$
$z \geq \pi$
$\theta := \theta + \pi/2$
Mode 3
$\dot{x}_i = vcos(\theta_i) + dx_i$
$\dot{y}_i = vsin(\theta_i) + dy_i$
$\dot{\theta}_i = 0$
$\dot{z}_i = 0$
$\dot{x}_i = vcos(\theta_i) + dx_i$
$\dot{y}_i = vsin(\theta_i) + dy_i$
$\dot{\theta}_i = \omega$
$\dot{z}_i = 1$
$\dot{x}_i = vcos(\theta_i) + dx_i$
$\dot{y}_i = vsin(\theta_i) + dy_i$
$\dot{\theta}_i = 0$
$\dot{z}_i = 0$
Fig. 12 System dynamics for the three modes

As shown in Fig. 11, all the aircrafts 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).

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

The dynamics of the system are shown in Fig. 12. The guard transition between mode 1 and mode 2 is given by $Dij < R$, which means that the aircraft i is at R distance from the aircraft j. The dynamics of each aircraft is as follows:

$$\dot{x}_i = v\cos(\theta_i) + dx_i,$$

$$\dot{y}_i = v\sin(\theta_i) + dy_i,$$

$$\dot{\theta}_i = \omega.$$

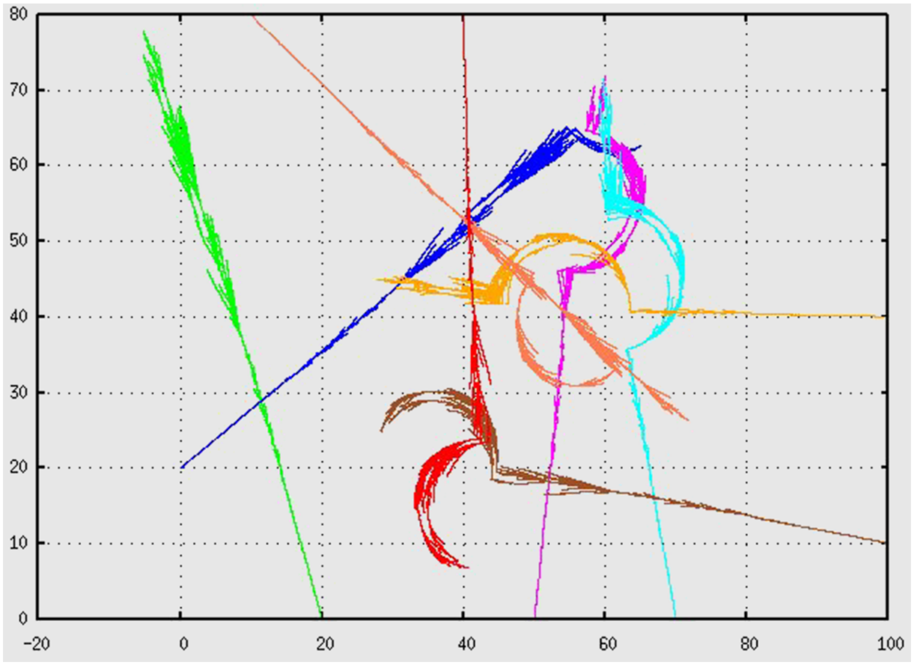


Fig. 13 Eight-aircraft collision avoidance (50000 visited states, computation time: 10 min)

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

$$dx_i = d_1 \sin(\theta_i) + d_2 \cos(\theta_i),$$

$$dy_i = -d_1 \cos(\theta_i) + d_2 \sin(\theta_i),$$

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

Results For N aircrafts, the system has $3N + 1$ continuous variables (one for modeling a clock). For the case of $N = 2$ aircrafts, 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 $N = 8$ aircrafts with the disturbance bound $\delta = 0.06$ is shown in Fig. 13, where we show the projected positions of the eight aircrafts on a 2-dimensional space. For this example, the computation time for 50000 visited states was 10 min and a collision was found. For a similar example with $N = 10$ aircrafts, the computation time was 14 min and a collision was also found.

8.2.3 A robotic vehicle benchmark

This example is adapted from the robotic navigation system benchmark [22]. We consider a car with the following continuous dynamics with 5 variables: $\dot{x} = v\cos(\theta)$, $\dot{y} = v\sin(\theta)$, $\dot{\theta} = v\tan(\phi)/L$, $\dot{v} = u_0$, $\dot{\phi} = u_1$ where x, y, θ describe the position and heading of the car, v is its speed and ϕ is its steering angle. The car can be in one of three car modes (smooth car, smooth unicycle, smooth differential drive). In this work, we consider only the smooth car mode.

The inputs of the system are u_0 and u_1 which are respectively the acceleration and steering control. The system uses a hybrid control law with 3 driver modes. In the first driver mode, called *RandomDriver*, the control inputs are selected uniformly at random between their lower and upper bounds. In the second driver mode, called *StudentDrive*, when the speed is low, u_0 is randomly chosen as in first mode; otherwise, the strategy is to reduce the speed. In the third driver mode, called *HighwayDrive*, the strategy is to reduce the speed when it is high and increase it when it is low. A detailed description of this control law can be found in [22].

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

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

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

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

9 Related work

Classical model-based testing frameworks use Mealy machines or finite labeled transition systems and their applications include testing of digital circuits, communication protocols and software. Recently, these frameworks have been extended to real-time systems and hybrid systems. Here we only discuss related work in hybrid systems testing. The paper [23] proposed a framework for generating test cases from simulation of hybrid models specified using the language CHARON [2]. 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 validation method. In [11], 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. 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 proposed.

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 [4, 8, 9, 11, 22] and references therein). Here we only discuss a comparison of our approach with some existing RRT-based approaches for the validation of continuous and hybrid systems. Concerning the problem of defining a hybrid distance, our hybrid distance is close to that proposed in [11]. The difference is that we use the centroids of the guard sets to define the distance between these sets, while the author of [11] 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 into account the system dynamics. It is based on the spatial positions of the states. In [11] 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 [11] 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 [8, 11] and in our hRRT algorithm, the problem of optimally steering the system towards the goal states was not addressed. In other words, the evolution of the tree is mainly determined by the selection of nearest neighbors. In [9], the problem of computing optimal successors was considered more carefully, and approximate solutions for linear dynamics as well as for some particular cases of non-linear dynamics were proposed. The authors of [22] proposed a search on a combination of the discrete structure and the coarse-grained decomposition of the continuous state space into regions, in order to determine search directions. This can be thought of as an implicit way of defining a hybrid distance as well as a guiding heuristics.

Concerning test coverage for continuous and hybrid systems, in [8] 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 while in our work, the coverage measure is used to guide the simulation, in [8] it is used as a termination criterion. The paper [10] addresses the problem of robust testing by quantifying the robustness of some properties under parameter perturbations. This work also considers the problem of how to generate test cases with a number of initial state coverage strategies.

Concerning guided exploration, sampling the configuration space has been one of the fundamental issues in probabilistic motion planning. Our idea of guiding the test generation via the sampling process has some similarity with the sampling domain control [27]. As mentioned earlier, the RRT exploration is biased by the Voronoi diagram of the vertices of the tree. If there are obstacles around such vertices, the expansion from them is limited and choosing them frequently can slow down the exploration. In the dynamic-domain RRT algorithm, the domains over which the goal points are sampled need to reflect the geometric and differential constraints of the system, and more generally, the controllability of the system. In [17], 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 [8] where the number of successful iterations is used to define an adaptive biased sampling. To sum up, the novelty in our guiding method is that we use the information about the current coverage of the visited states in order to improve the coverage quality.

10 Conclusion

The main contributions of the paper can be summarized as follows. We proposed a formal framework for conformance testing of hybrid systems, using the international standard for formal conformance testing [26]. 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 test coverage problem. We proposed a novel coverage measure, which is useful not only as a criterion to evaluate testing quality but also to guide the test generation process. Our coverage-guided test generation algorithms are based on a combination of the ideas from robotic path planning, equidistribution theory, algorithmic geometry, and numerical simulation. The experimental results obtained using an implementation of the test generation algorithm show its scalability to high dimensional systems and good coverage quality.

A number of directions for future research can be identified. First, we are interested in defining a measure for trace coverage. Partial observability also needs to be considered. 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. Finally, 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.

Appendix

Proof of the Reachability Completeness Theorem (Theorem 2) We first prove the following intermediate result: given a reachable set R with positive volume, the probability that there exists $k > 0$ such that $V^k \cap R \neq \emptyset$ is non-null. Recall that V^k is the states at the vertices of the tree at iteration k. We observe that since the whole set R is reachable and each set R^j defined in (10) has positive volume (by the condition (H3)), there must exist $j > 0$ such that the intersection $R \cap R^j$ has positive volume. It thus suffices to prove that for any set $P^j \subseteq R^j$ with positive volume,

$$\Pr[\exists k > 0 : V^k \cap P^j \neq \emptyset] > 0. \tag{13}$$

Let W^j be the set of all sequences of discrete transitions that steer the system from the initial state $s_{init} = (q_{init}, x_{init})$ to P^j. If W^j is empty, which means that P^j is reached only by the continuous dynamics at the location q_{init}. Note that, using the conditions (H1), (H2) and (H3) and the idea of the completeness proof[5] for the continuous systems [14], we can prove that for any reachable set Y at location q_{init} with positive volume, there is a non-null probability that after some $k > 0$ iterations $V^k \cap Y$ is non-empty (that is, the set Y is visited by the algorithm). Hence, (13) can be proven.

We consider now the case where W^j is not empty. By the definition of R^j in (10), the length of each sequence in W^j is at most j. To prove (13), it suffices to show that there is a non-null probability that a sequence in W^j is visited by the algorithm after $k > 0$ iterations. To do so, let $p = e_1, e_2, \ldots, e_l$ be a sequence in W^j. The transition e_1 from the initial location can be visited if its guard set can be reached. By the condition (H4), the set of reachable states enabling the transition e_1 satisfies the positive volume neighborhood property. Again, combining this with the completeness proof for continuous systems, we can conclude that there is a non-null probability that the guard set of e_1 is visited by the algorithm. Similarly, we can prove the same for the subsequent transitions of p. Consequently, there is a non-null probability that the sequence p is visited by the algorithm after $k > 0$ iterations.

We proceed with the proof of the theorem. We define the set

$$B_{reach}(s) = \mathrm{Reach}_b(s) \cap \mathcal{N}(s, \varepsilon). \tag{14}$$

Recall that $\mathrm{Reach}_b(s)$ is the backward reachable set from s, defined as in (11). Using the assumption (H3), the set $B_{reach}(s)$ has positive volume.

We define the distance from $s = (q, x)$ to V^k as $d^k(s) = \min_{s' \in V^k} d_H(s', s)$. Let $D^k(s)$ be a random variable whose value is $d^k(s)$.

Let $k \geq 0$ be an integer such that V^k does not contain a vertex inside $B_{reach}(s)$. Because the whole set $B_{reach}(s)$ is reachable, using the above intermediate result, we have

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

Note that the fact $V^{k'}$ contains a state in $B_{reach}(s)$ implies that $d^{k'}(s) < d^k(s)$, since none of the states in V^k is in $B_{reach}(s)$. In addition, $d^k(s)$ is non-increasing with respect to k; therefore there exists a strictly positive constant c such that $D^{k'}(s) - D^k(s) > c$. Therefore, $\lim_{k\to\infty} \Pr[d^k(s) \leq \varepsilon] = 1$, which means that $\lim_{k\to\infty} \Pr[\exists s' \in V^k : s' \in \mathcal{N}(s, \varepsilon)] = 1$. This establishes the proof of the theorem. □

References

1. Alur R, Courcoubetis C, Halbwachs N, Henzinger TA, Ho P-H, Nicollin X, Olivero A, Sifakis J, Yovine S (1995) The algorithmic analysis of hybrid systems. Theor Comput Sci 138:3–34

[5] The idea of the completeness proof for the continuous systems was pointed out in Sect. 7.1.

2. Alur R, Dang T, Esposito J, Hur Y, Ivancic F, Kumar V, Lee I, Mishra P, Pappas G, Sokolsky O (2002) Hierarchical modeling and analysis of embedded systems. In: Proceedings of the IEEE, October 2002
3. Beck J, Chen WWL (1988) Irregularities of distribution. Cambridge Univ Press, Cambridge
4. Branicky MS, Curtiss MM, Levine J Morgan S (2005) Sampling-based reachability algorithms for control and verification of complex systems. In: Proc thirteenth Yale workshop on Adaptive and Learning Systems, New Haven, CT
5. Cheng P, LaValle S (2002) Resolution complete rapidly-exploring random trees. In: Proc IEEE International Conference on Robotics and Automation, pp 267–272
6. Dang T, Nahhal T (2006) Randomized simulation of hybrid systems. Technical report, Verimag, May 2006
7. Dobkin D, Eppstein D (1993) Computing the discrepancy. In: Proceedings of the Ninth Annual Symposium on Computational Geometry, pp 47–52
8. Esposito JM, Kim J, Kumar V (2004) Adaptive RRTs for validating hybrid robotic control systems. In: Int workshop on the Algorithmic Foundations of Robotics
9. Bhatia A, Frazzoli E (2004) Incremental search methods for reachability analysis of continuous and hybrid systems. In: Hybrid Systems: Computation and Control HSCC. LNCS, vol 2993. Springer, Berlin, pp 142–156
10. Julius AA, Fainekos GE, Anand M, Lee I, Pappas GJ (2007) Robust test generation and coverage for hybrid systems. In: Hybrid Systems: Computation and Control HSCC. LNCS. Springer, Berlin, pp 329–342
11. Kim J, Esposito JM, Kumar V (2006) Sampling-based algorithm for testing and validating robot controllers. Int J Rob Res 25(12):1257–1272
12. LaValle SM, Branicky MS, Lindemann SR (2004) On the relationship between classical grid search and probabilistic roadmaps. Int J Rob Res 23(7–8):673–692
13. Faure H (1978) Discrépance de suites associees à un système de numération. General theory of distribution modulo 1. In: Irregularities of distribution
14. Kuffner J, LaValle S (2000) 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
15. LaValle SM, Kuffner JJ (2001) Rapidly-exploring Random Trees: progress and prospects. In: Algorithmic and computational robotics: new directions. AK Peters, Wellesley, pp 293–308
16. Lee D, Yannakakis M (1996) Principles and methods of testing finite state machines—A survey. In: Proceedings of the IEEE, vol 84, pp 1090–1123
17. Lindemann SR, LaValle SM (2004) Incrementally reducing dispersion by increasing voronoi bias in RRTs. In: Proceedings IEEE International Conference on Robotics and Automation
18. Mitchell I, Tomlin C (2000) Level set methods for computation in hybrid Systems. In: Hybrid Systems: Control and Computation HSCC. LNCS, vol 1790. Springer, Berlin
19. Moore A (1991) A tutorial on kd-trees. Computer laboratory technical report no 209, University of Cambridge, http://www.cs.cmu.edu/~awm/papers.html
20. Nahhal T, Dang T (2007) Guided randomized simulation. In: Hybrid Systems: Control and Computation. LNCS, vol 4416. Springer, Berlin, pp 731–735
21. Nahhal T (2007) Model-based testing of hybrid systems. PhD thesis, Joseph Fourier University, October 2007
22. Plaku E, Kavraki LE, Vardi MY (2007) Hybrid systems: From verification to falsification. In: Damm W, Hermanns H (eds) International Conference on Computer Aided Verification (CAV). LNCS, vol 4590. Springer, Berlin, pp 468–481
23. Tan L, Kim J, Sokolsky O, Lee I (2004) Model-based testing and monitoring for hybrid embedded systems. In: IRI, pp 487–492
24. Thiémard E (2001) An algorithm to compute bounds for the star discrepancy. J Complex 17(4):850
25. Tretmans J (1999) Testing concurrent systems: a formal approach. In: Int Conference on Concurrency Theory CONCUR. LNCS, vol 1664. Springer, Berlin
26. Tretmans J (1994) A formal approach to conformance testing. In: Proceedings of the IFIP TC6/WG6.1 sixth international workshop on Protocol Test Systems VI. North-Holland, Amsterdam, pp 257–276
27. Yershova A, Jaillet L, Simeon T, LaValle SM (2005) Dynamic-domain RRTs: efficient exploration by controlling the sampling domain. In: Proc IEEE International Conference on Robotics and Automation
28. Wang X, Hickernell F (2000) Randomized Halton sequences. Math Comput Model 32:887–899
29. Zhu H, Hall PAV, May JHR (1997) Software unit test coverage and adequacy. In: ACM Computing Surveys, 29, 4 Dec 1997