Classification and Coverage-Based Falsification for Embedded Control Systems

Arvind Adimoolam[1], Thao Dang[1(✉)], Alexandre Donzé[2], James Kapinski[3], and Xiaoqing Jin[3]

[1] CNRS/Verimag, Grenoble, France
{santosh.adimoolam,thao.dang}@univ-grenoble-alpes.fr
[2] Decyphir, Inc., San Francisco, CA, USA
alex@decyphir.com
[3] Toyota Motors North America R&D, Gardena, CA, USA
{james.kapinski,xiaoqing.jin}@toyota.com

Abstract. Many industrial cyber-physical system (CPS) designs are too complex to formally verify system-level properties. A practical approach for testing and debugging these system designs is falsification, wherein the user provides a temporal logic specification of correct system behaviors, and some technique for selecting test cases is used to identify behaviors that demonstrate that the specification does not hold for the system. While coverage metrics are often used to measure the exhaustiveness of this kind of testing approach for software systems, existing falsification approaches for CPS designs do not consider coverage for the signal variables. We present a new coverage measure for continuous signals and a new falsification technique that leverages the measure to efficiently identify falsifying traces. This falsification algorithm combines global and local search methods and uses a classification technique based on support vector machines to identify regions of the search space on which to focus effort. We use an industrial example from an automotive fuel cell application and other benchmark models to compare the new approach against existing falsification tools.

1 Introduction

Cyber-physical systems integrate heterogeneous components whose descriptions in high level modeling languages involve a wide array of specification paradigms, such as differential equations, difference equations, automata, and data flow graphs. Although the behavior of individual cyber-physical components may be amenable to rigorous mathematical reasoning and analysis, the complex interactions between the components are still not well-understood and pose major theoretical hurdles in formal reasoning. Also, the scalability of the existing formal verification methods and tools (see [1,3,4,12,15,16,18,26] and references therein) is still limited and therefore not suited for verifying industrial scale cyber-physical systems. Testing is an alternate approach for detecting errors, whose advantage over formal verification methods is that it can treat a system

R. Majumdar and V. Kunčak (Eds.): CAV 2017, Part I, LNCS 10426, pp. 483–503, 2017.
DOI: 10.1007/978-3-319-63387-9_24

as a black box, meaning that no internal description of the system is required. In black box testing, only an interface of the system with the external environment is described. Although testing can be applied to large scale cyber-physical systems, as attested by its use in industry, it does not provide proofs of correctness. In other words, black box testing can only detect bugs, and when it does so successfully, it means that the system design has to be corrected. Nevertheless, when the testing process does not find any bugs, we cannot draw any conclusion about its correctness. If the falsification is unsuccessful, then information about the potential validity of the correct behavior of the system would be of great interest to the designer. This information can be provided in terms of a testing coverage measure.

In the existing research on cyber-physical systems testing, the focus was generally on *state-coverage measures*, that is measures to characterize the portion of the state space covered by a test suite. An example is star discrepancy [6,11], a notion borrowed from statistics that indicates how equi-distributed are a set of tested points in the state space. Some other measures are dispersion [13], which indicates the size of the largest unexplored areas, and grid-cell count [25]. Although these state-coverage measures can serve as a possible means to compare coverage of testing data generated by different algorithms, these measures exhibit the following drawbacks. Typically, a test generation algorithm guided by a state coverage measure tries to sample test cases in the areas that are not well explored; however, in industrial scale system models describing interactions among a large number of heterogeneous components, information about the state can be hard to obtain. Additionally, such systems can have low controllability, meaning that it is difficult or impossible to reach some regions of the state space. In such a case, the algorithms can expend a large amount of time attempting to explore unreachable regions. So, state coverage measures are not appropriate for analysis or guidance of the testing effort on many cyber-physical systems.

The present work addresses the shortcomings of the state-coverage-based techniques by instead focusing on *coverage of input signal spaces*. We develop a new test generation technique that is based on covering the input signal space rather than the state space. Previous test generation methods have considered coverage of a parameter space (such as in [9]); the way that we handle input signals is directly related, as we consider the class of finitely parameterized input signals.

While coverage is important during testing for providing confidence in correctness of the system behavior, bug detection is still an important goal of testing. Usually, there is a mutual tradeoff between satisfying the two criteria. Achieving good coverage entails exploring a large portion of the search space, most of which would correspond to correct behaviors. Whereas, the objective of a falsification procedure is to find incorrect behaviors, which would require focusing on behaviors close to incorrectness. Most falsification methods are based on minimizing the behavioral robustness with respect to a property under test; the robustness measure here indicates how far the behavior is from violating the property. A common drawback of such falsification methods is that the optimization procedures can spend a significant amount of computing time near local optima that may not correspond to a false behavior. Therefore, a criterion like coverage can help overcome this drawback, since seeking to improve the global coverage would drive the search process out of the areas of local optima. One way to achieve a good compromise between coverage-driven and local search-driven testing is to initiate the search procedures from points that are separated by some threshold distance. This insight was used previously in the tabu search method, which ensures that all the starting points are well separated [7]; however, apart from ensuring that the starting points are well separated, it is also desirable that they are chosen in regions in which one can reasonably expect to find an incorrect behavior. That is, heuristically speaking, a starting point should have a low robustness value.

Based on the above observations, in this work we present a falsification algorithm that combines the following three essential ideas:

– Defining a coverage measure for quantifying the exploration of input signal space during testing.
– Guiding a randomized global search procedure by performing robustness classification: the classification divides the search space into regions with different potentials of falsification characterized by the robustness of evaluated test cases. Our classification is inspired from linear *support vector machines* [8,17].
– Using local search in regions classified as less robust. The above-mentioned global search together with an iterative classification procedure does converge towards an incorrect behavior, if it exists. However, to speed up the convergence, instead of continually classifying, we can use the information obtained from classification to efficiently initialize a local search within each classified region. Note that in general, local search with arbitrary initialization can perform poorly. Therefore, by alternating classification and local search we can achieve a better convergence while assuring a good coverage of the input signal space (because in general local search does not take into account this coverage criterion).

Before proceeding further, we note that our idea of combining global and local search for black box falsification is independent of the work [22] that is concerned with falsification based on state trajectories. In the latter work [22], although the motivation is to combine local and global search, the state trajectories have to be computed, in which case the system is not a black-box. On the other hand, our work is concerned with black-box kind of systems, i.e., complex systems where the information about the state of the system is very hard, if not impossible, to know.

For implementation and evaluation purposes, for local search we use a method, called the CMA-ES (Covariance Matrix Adaptation Evolution Strategy) [21], also used by the tool Breach [9]. The CMA-ES algorithm is considered as the state-of-the-art in evolutionary computation and has been used for industrial optimization applications. The experimental results obtained using a MATLAB implementation of our falsification algorithm on some benchmark systems demonstrate its good performance and, in addition, its efficiency improvements over search algorithms like the CMA-ES. Indeed, our algorithm was tested on a difficult property of the PTC benchmark [11] and could falsify in all the tested random seeds while the methods based on pseudo-random sampling or only on the CMA-ES could not. Also, we demonstrate that the technique can be successfully applied to industrial problems by presenting results for a prototype automotive hydrogen fuel cell application.

Our approach draws inspiration from the approaches implemented in the tools S-Taliro [2] and Breach [9]. These approaches seek the worst case behaviors using the notion of robustness metrics, which are defined with respect to properties specified using the languages MTL (Metric Temporal Logic) [14] and STL (Signal Temporal Logic) [10], respectively. The tools identify property violations by employing global optimization methods to search for behaviors that minimize robustness, where negative robustness values correspond to property violations. Robustness-based approaches can be seen as complementary to coverage-based approach, since the former try to find a worst case behavior while the latter tries to cover a large number of possible behaviors. When a robustness-based approach cannot find an erroneous behavior due to the limitations of global optimization algorithms, the observed error absence cannot be used as a formal correctness proof; in this case good coverage would be desirable to enhance the confidence that the system is free from errors. By combining robustness-based and coverage-guided explorations, our approach enhances the overall testing effectiveness by providing confidence that important or representative behaviors are tested.

2 Preliminaries

We consider system models defined by a mapping from parameters and input signals to output signals,

$$y = \Phi(v, u), \tag{1}$$

where $v \in \mathcal{V}$ is a valuation of a finite collection of parameters, and $u \in \mathcal{U}$ is an input signal used to simulate the system. In this setting, v could contain a set of system initial conditions as well as some finite set of system parameters. Each input signal $u \in \mathcal{U}$ is a function $\mathcal{I}_u \mapsto U$, where $\mathcal{I}_u$ is an interval (either discrete or continuous) from 0 to some finite value, and U is some metric space of finite dimension. Similarly, we assume that each output signal $y \in \mathcal{Y}$ is a function $\mathcal{I}_y \mapsto Y$, where $\mathcal{I}_y$ is an interval (either discrete or continuous) from 0 to some finite value, and Y is some metric space of finite dimension. We assume that $\mathcal{V}$, $\mathcal{U}$, and $\mathcal{Y}$ are metric spaces. Note that the system defined by (1) does not explicitly model the behaviors of the internal system states. State behaviors could be modeled using this framework by ensuring that v includes the system state and all of the states map to system outputs, but we do not require this.

We assume that signals are finitely parameterized, i.e., an input signal u can be uniquely determined by a finite set of m parameters, whose valuation $\widehat{u}$ is a in a subset $\widehat{\mathcal{U}}$ of an m-dimensional metric space. For example, a right-continuous piecewise constant input signal $u : \mathcal{I}_u \rightarrow \mathbb{R}$, where $\mathcal{I}_u = [0, T]$, with discontinuities occurring at monotonically increasing instants $\tau_1, \ldots, \tau_m$, where

$0 = \tau_1 < \tau_m < T$, can be uniquely defined by the m values $u(\tau_i)$. Subsequently, our system can be defined as a mapping from a finite set of parameters to the output signals, as follows:

$$y = \widehat{\Phi}(v, \widehat{u}),\ v \in \mathcal{V} \text{ and } \widehat{u} \in \widehat{\mathcal{U}} \tag{2}$$

We call $\mathcal{V}$ the space of *nominal parameters* and $\widehat{\mathcal{U}}$ the space of *input signal parameters.*

Signal Temporal Logic. To specify correct behavior of a system defined by (1), we use signal temporal logic (STL) [23]. STL can capture behaviors of real valued signals over discrete or dense time. We present here an informal description of STL (see [23] for more details). A formula in STL consists of atomic predicates, Boolean, and temporal operators. Atomic predicates are inequalities over signal values, as in $\mu = f\,(y(t)) \sim 0$, where f is a scalar-valued function over the signal y evaluated at time t, and $\sim \in \{<, \leq, >, \geq, =, \neq\}$. Temporal operators "always" ($\Box$), "eventually" ($\Diamond$), and "until" ($\mathcal{U}$) have the usual meaning and are scoped using intervals of the form (a, b), $(a, b]$, $[a, b)$, $[a, b]$, (a, ∞), or (a, ∞), where $a, b \in \mathbb{R}_{\geq 0}$ and $a < b$. If I is such an interval, then the language of STL is given by the following grammar:

$$\varphi \ := \ \top \mid f(y(t)) \sim 0 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2 : \quad \sim \in \{<, \leq, >, \geq, =, \neq\} \tag{3}$$

The $\Diamond$ and $\Box$ operators are defined as follows: $\Diamond_I \varphi \triangleq \top \mathcal{U}_I \varphi$, $\Box_I \varphi \triangleq \neg\Diamond_I \neg\varphi$. When omitted, the interval I is assumed to be $[0, \infty)$. The semantics are described informally as follows. The signal u satisfies $f(u) > 0$ at time t if $f(u(t)) > 0$. It satisfies $\varphi = \Box_{[0,1)}(f(u) = 0)$ if for all time $0 \leq t < 1$, $f(y(t)) = 0$. The signal satisfies $\varphi = \Diamond_{[1,2)} f(u) < 0$ iff there exists a time t such that $1 \leq t < 2$ and $u(t) < 0$. The two-dimensional signal $y = (y_1, y_2)$ satisfies the formula $\varphi = (y_1 > 0)\mathcal{U}_{[2.3,4.5]}(y_2 < 0)$ iff there is some time t where $2.3 \leq t \leq 4.5$ and $y_2(t) < 0$, and $\forall t'$ in $[2.3, t)$, $y_1(t')$ is greater than 0.

Quantitative Semantics for STL. The quantitative semantics of STL tells how far a signal is from satisfying a formula. In this respect, we use the quantitative interpretation presented in [10], which we describe informally as follows. The semantics relies on a function ρ such that a positive sign of $\rho(\varphi, y, t)$ indicates that (y, t) satisfies φ, and its absolute value estimates the *robustness* of this satisfaction. If ϕ is a simple inequality of the form $f(y) > b$, then its robustness is $\rho(\varphi, y, t) = f(y(t)) - b$. For the conjunction of two formulas $\varphi := \varphi_1 \wedge \varphi_2$, we have $\rho(\varphi, y, t) = \min\left(\rho(\varphi_1, y, t), \rho(\varphi_2, y, t)\right)$, while for the disjunction $\varphi := \varphi_1 \wedge \varphi_2$, we have $\rho(\varphi, y, t) = \max\left(\rho(\varphi_1, y, t), \rho(\varphi_2, y, t)\right)$. For a formula with until operator as $\varphi := \varphi_1 \mathcal{U}_I \varphi_2$, the robustness is computed as $\rho(\varphi, y, t) = \max_{t' \in t+I}\left(\min\left(\rho(\varphi_2, y, t), \min_{t' \in [t,t']}\left(\rho(\varphi_1, y, t'')\right)\right)\right)$.

Since the output signal is determined by the set of nominal parameters and input signal parameters according to the mapping $\widehat{\Phi}$, we can define a robustness function over the space of parameters, called *parametric robustness*, as $\widehat{\rho}(\varphi, v, \widehat{u}, t) = \rho\left(\varphi, \widehat{\Phi}\left(v, \widehat{u}\right), t\right)$.

Falsification. Finding a counterexample of φ means finding a parameter value $v \in \mathcal{V}$ and an input parameter value $\widehat{u} \in \widehat{\mathcal{U}}$ such that $y \not\models \varphi$, where $y = \widehat{\Phi}(v, \widehat{u})$. Equivalently, the counterexample is identified when its parametric robustness is less than zero, i.e., $\widehat{\rho}(\varphi, v, \widehat{u}, t) < 0$ for some time point t in the time horizon of the signal. We call any $v \in \mathcal{V}$ and $\widehat{u} \in \widehat{\mathcal{U}}$ for which $y \not\models \varphi$ a *counterexample* and we call this task of finding a counterexample as a *falsification problem.* We say that a counterexample y (that is $y \not\models \varphi$) is *robust* if there exists a neighborhood around y, $\mathcal{N}_y$, such that for all $y' \in \mathcal{N}_y$, $y' \not\models \varphi$. We call a corresponding neighborhood $\mathcal{N}_y$ a robustness neighborhood of counterexample y. If a counterexample has a robustness neighborhood that contains a closed ball of radius ϵ, then we say that the counterexample is ϵ-*robust*.

Continuity of Robustness. Recall that our input signals are assumed to be finitely parametrized and correspondingly we defined the parametric robustness function. If we assume that the predicates of an STL formula are defined by functions f in (3) which are continuous w.r.t. the value of y at any time t, and the mapping Φ defining the system dynamics is continuous w.r.t. the parameter and input signal, then we can prove that the parametric robustness is continuous w.r.t. v and $\widehat{u}$. Indeed, for any atomic predicate $\varphi = f\,(y(t)) \sim 0$, the parametric robustness $\widehat{\rho}\,(\varphi, v, \widehat{u}, t)$ is continuous because f and $\widehat{\Phi}$ are continuous. Next, for any general formula as defined in (3), the robustness is computed by a composition of *min* and max operators of subformulas. By using induction we thus can deduce that the parametric robustness, given the aforementioned assumptions, is continuous in the input parameter $\widehat{u}$ and the nominal parameter v.

3 Input Space Coverage - Cell Occupancy

This section presents a metric that we use to measure the coverage of signal spaces. The notion is intended to be used to define the coverage of input signals used to stimulate a dynamical system. We define a measure called cell occupancy, which has the following desirable properties:

- The measure is *monotonic*, in the sense that it is guaranteed not to decrease in value when new signals are added to an existing set;
- The measure permits computation with *efficient algorithms*;
- The measure provides numbers in *reasonable ranges*, in the sense that, for both low dimension and high dimension problems, the measure results in values that are neither too large nor too small so as to be accurately represented with standard floating point numbers.

Henceforth, we define a measure called *cell occupancy* as follows. Let M be a set of signals, which corresponds to a set of parameter vectors X_M. We call elements of X_M points. We use p to denote the size of sets M and X_M.

Choose a partition of X, $\omega = \{\omega_i | i = 1, \ldots, l\}$. For now, we assume that each partition element, which we call a *cell*, is rectangular, with each side of equal length, $\varDelta$, called *grid cell size*[1]. A vector that indicates how many points are in each cell is called a *distribution*, $D = (n_1, \ldots, n_l)$, where each n_i indicates how many points are located in cell i. Cell occupancy is based on the relative number cells occupied by points, compared to the total number of cells. Consider the total number of occupied cells, that is, the number of cells that contain at least one point, i.e., $N_c = \sum_{i=1}^{l} g_i$ where $g_i = 1$ if $n_i \geq 1$, and $g_i = 0$ otherwise. Then, the proposed cell occupancy measure is given as

$$H_c(D) = \frac{\log N_c}{\log l}.$$

Logarithm functions are used due to the fact that the total number of cells could be very large as compared to the number of occupied cells. The logarithms provide two key features for the cell occupancy measure: (1) they maintain the monotonicity of the measure, and (2) they result in reasonable measure values even for cases where the dimension m is large.

Guarantee for Finding Counterexample. We consider here falsification algorithms based on an iterative search on the nominal parameter and input signal parameter spaces. We assume that the functions in the atomic predicates of STL formulas are continuous in the value of the output signal at any fixed time point. Also, the system mapping $\widehat{\varPhi}$ is assumed to be continuous w.r.t. the input signal parameters and nominal parameters. In this case, if a falsification algorithm is such that the cell occupancy is guaranteed to increase after a finite number of robustness evaluations for any partition, then because of the continuity of parametric robustness (explained in Sect. 2), there exists a sufficiently small upper bound on the grid cell size below which, the algorithm is guaranteed to find a counterexample. This is summarized by the following lemma. However, note that in general such falsification algorithms may be used for non-continuous systems as well. The following lemma gives a theoretical insight about why coverage may be taken into account for designing efficient falsification algorithms.

Lemma 1. *Given a falsification algorithm and a partition ω with l grid cells of size $\varDelta > 0$, let $D(\kappa, \varDelta)$ denote the cell distribution after κ robustness evaluations by the algorithm. Let us consider that there exists $\alpha \in \mathbb{Z}_{>0}$ for which the algorithm guarantees that $\forall \kappa \in \mathbb{Z}_{\geq 0}\ H_c\left(D(\kappa + \alpha, \varDelta)\right) > H_c\left(D(\kappa, \varDelta)\right)$. Let us also consider that the system mapping $\widehat{\varPhi}$ is continuous and an STL formula φ is formed by continuous predicates with respect to the signal value at a fixed time. In this case, if an ϵ-robust counterexample of φ exists, then there exists an upper bound $\overline{\varDelta} > 0$ on the grid cell size of ω such that $\forall \varDelta < \overline{\varDelta}$, the algorithm finds a counterexample after a finite number of robustness evaluations.*

[1] We note that in the setting in which we intend to apply the following coverage metrics, we will expect to select points in X that are no closer than some ϵ distance from each other, based on some metric between signals, but this rectangularity will not be exploited in the following. Further, we assume that $\epsilon \ll \varDelta$.

4 Falsification Techniques

We use the term *sampling a point* to mean selecting a parameter vector x in the parameter set X_M, to uniquely define an input signal in M. Such signals are then used as stimuli to simulate the system and determine the robustness values of the corresponding output traces. For simplicity of notation, for a given sampled parameter vector x, we write $\rho(x)$ to denote the robustness value of the corresponding output trace. And for a set S of sampled parameter vectors, $\rho(S) = \min\{\rho(x) \mid x \in S\}$. A parameter vector x is called a *falsifier* if $\rho(x) < 0$.

A rudimentary approach to search for a falsifier is repeatedly select randomly an unoccupied cell with uniform probability distribution and evaluate one point inside it. This way, we ensure that the cell occupancy always keeps increasing until we eventually find a robust bug, if it exists (see Lemma 1); however, this approach may not be efficient because the uniform search does not differentiate regions that are more likely to contain an input that falsifies from those that are less likely. Therefore, we propose to enhance it using two concepts:

1. *Using classification to bias random search.* We use robustness based classification to classify less falsifiable regions from more falsifiable regions. Then, the probability distribution of random samples is biased according to the coverage and robustness information in different regions.
2. *Combining global search and local search.* Local search approaches (such as Hill climbing, Gradient methods, Simulated annealing, Genetic algorithms) (see for example [19,24]) can be very efficient if the search procedures are appropriately initialized. Finding good initializations constitutes a major difficulty that limits the efficiency of these approaches. In our framework, the classification based global search provides useful hints at appropriate initializations for the local search. Indeed, the least robust points in regions with high potential of falsification can be used to initialize a number of local searches.

Thus, our falsification algorithm involves two phases. The first phase is a global search guided by hyperplane classifiers, coverage and the robustness information. Next is a local search phase which runs a number of local searches initiated at the least robust points of different regions formed by the classification process during the global search. We now explain the aforementioned ingredients of the algorithm.

4.1 Classification Using Hyperplane Subdivision

In the following, we say that two regions are *separate* if their intersection can only lie on their boundaries. Our classification problem can be intuitively described as follows: given a rectangle R representing a search space and a set S of sampled points in R, iteratively subdivide it, according to the robustness values of the sampled points, to obtain a rectangular partition, the elements of which have different average robustness levels. We define the average robustness of the set of samples S in R as $\mu = \frac{\sum_{x \in S} \rho(x)}{|S|}$. Our objective is thus to separate a region of R having higher potential of containing low robustness samples. To this end, we define a hyperplane, in view of separating samples below the average robustness μ from those above μ. Obviously, such a hyperplane does not always exist, and we therefore choose a hyperplane that does this separation as best as possible. To address this problem, we draw inspiration from *soft margin support vector machines* [5], where hyperplanes are determined so that the misclassification error is minimized. In general, a misclassification error is defined according to the locations of the misclassified samples; however, in our approach we define a misclassification error that gives weightage to the robustness values of the misclassified samples in addition to their locations. Furthermore, since it is easy to sample uniformly in rectangles, we only use axis-aligned hyperplanes, which generate only rectangular subregions. Otherwise, when allowing non-axis aligned classifiers, we generate polyhedral regions in which uniform random sampling as well as partition manipulation could be more expensive.

To explain the essence of our classification method, let us consider one rectangle R in the partition as the product of intervals $R = [a_1, b_1] \times \ldots \times [a_n, b_n]$. Let S be the set of samples in R. We denote an axis-aligned hyperplane inside R by a tuple (d, r) where $d \in \{1, \ldots, n\}$ is the axis normal to the separating hyperplane, while $r \in [a_d, b_d]$ is a coordinate at which the hyperplane is drawn. The hyperplane (d, r) subdivides R into two subrectangles $A^-(R, d, r)$ and $A^+(R, d, r)$ such that $A^-(R, d, r) = [a'_1, b_1] \times \ldots \times [a'_n, b_n]$ where $a'_j = r$ if $j = d$, and $a'_j = a_j$ otherwise; and $A^+(R, d, r) = [a_1, b'_1] \times \ldots \times [a_n, b'_n]$ where $b'_j = r$ if $j = d$, and $b'_j = b_j$ otherwise (Fig. 1).

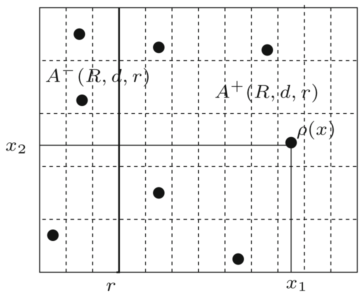

Fig. 1. Classification by subdivision. The samples in a 2-dimensional rectangle R are represented by black points labeled with their robustness values; R is divided by the hyperplane (d, r) where the axis is $d = 1$, to minimise the misclassification error. This division produces two subrectangles $A^-(R, d, r)$ (on the left) and $A^+(R, d, r)$ (on the right).

An ideal separation of samples below the average robustness from those above the average robustness by the hyperplane can be described in one of the following scenarios, identified by the following notion of *polarity*. Hyperplane (d, r)

has polarity $p = 1$ w.r.t. S, if the left subrectangle $A^-(R,d,r)$ contains all samples below the average μ, while the right subrectangle $A^+(R,d,r)$ contains all samples above the average μ. Similarly, (d,r) has polarity $p = -1$ w.r.t. S, if $A^+(R,d,r)$ contains all samples below μ, while $A^-(R,d,r)$ contains all samples above μ. When an ideal separation as above is not feasible, we identify misclassified samples as follows.

Definition 1. *A point $x \in R$ is misclassified w.r.t. a hyperplane (d,r), polarity $p \in \{-1,1\}$ and the sampled set S, if* $\operatorname{sgn}(p(\rho(x)-\mu)(x_d - r)) = 1$ *where x_d is the d^{th} coordinate of x, and* sgn *denotes the sign function.*

For a misclassified sample, the misclassification error is measured according to its location and robustness value. If a misclassified sample is farther from the hyperplane, it is considered to entail a higher misclassification error. Also, since the classification is based on the average robustness, samples with robustness values farther from the average get higher weightage in measuring the misclassification error. Accordingly, we define the misclassification error for a point $x \in R$ w.r.t. a hyperplane (d,r), polarity $p \in \{-1,1\}$, and a set of samples S as $e_{d,r}(x,R,S,p) = \max\{p(\rho(x)-\mu)(x_d - r), 0\}$. Then the total misclassification error is the sum of the misclassification errors of all the samples:

$$\Gamma_{d,r}(R,S,p) = \sum_{x \in S} e_{d,r}(x,R,S,p). \tag{4}$$

An appropriate hyperplane (d_*, r_*) traversing the rectangle R, chosen for the desired separation of S, is one that minimises the total misclassification error for either a positive or negative polarity, *i.e.*,

$$\left(d_*^{R,S}, r_*^{R,S}\right) = \operatorname*{argmin}_{r \in [a_d,b_d], d \in [n]} \left(\min\{\Gamma_{d,r}(R,S,-1), \Gamma_{d,r}(R,S,1)\}\right). \tag{5}$$

We denote $A_*^-(R,S) = A^-(R, d_*^{R,S}, r_*^{R,S})$ and $A_*^+(R,S) = A^+(R, d_*^{R,S}, r_*^{R,S})$ as the subrectangles formed by dividing R by the above optimal hyperplane.

It is important to remark that in order for the classification to reflect the robustness distribution over the whole dense space, the number of samples should be sufficiently large. Henceforth, only rectangles in which the number of samples is not smaller than a (user-defined) threshold number, are subdivided as above. The classification procedure takes as input a partition encoded as a list of k rectangles and the set of points in each respective rectangle. For each rectangle if the number of points is not smaller than the threshold K_c, the rectangle is subdivided by a hyperplane that minimizes the classification error. The rectangle is replaced by the left subrectangle and the right subrectangle is added to the list of rectangles. After all rectangles are considered for subdivision, the samples inside them are updated.

4.2 Global Search

Each iteration of the global search performs 3 successive procedures:

1. Classification using hyperplanes, the goal of which is to partition the state space into regions with different robustness levels.
2. Coverage and robustness guided sampling of input signal parameters.
3. Singularity based sampling of input signal parameters inside rectangles containing very low robust samples. Here, we use *singularity* to refer to a partition element that contains a point in a low robustness range with low frequency of occurance.

Note that the term *sampling* in the description of our method refers to the consecutive execution of three steps (1) defining the input signals from the sampled parameters, (2) simulating the system under the defined input signals, and (3) evaluating the robustness of the corresponding simulated output traces.

Coverage and Robustness Based Sampling. We randomly select a number of unoccupied cells, such that the probability of picking a cell in each rectangle is based on two components: coverage based probability and robustness based probability. Then the probability of cell sampling is determined as a weighted sum of the former components. Once a cell is sampled, a point is selected by a uniform sampling inside the cell.

Coverage Based Probability Distribution. Let $\{R_1, \ldots, R_k\}$ be the set of rectangles of a partition of the parameter space. We now consider the collection of grid cells intersecting with R_i, that is $\{\omega_j : \omega_j \cap R_i \neq \emptyset\}$, and we index them as $\beta^i = \left\{\beta^i_1, \ldots, \beta^i_{q_i}\right\}$ where q_i is the number of such cells. Let $D(R_i, S_i)$ be the vector denoting the distribution of samples S_i in cells of β^i, that is $\forall j \in \{1, \ldots, l\}$ (l is the total number of grid cells), the j^{th} component $D_j(R_i, S_i) = \left|S_i \cap \beta^i_j\right|$. Then the coverage based sampling probability in R_i is proportional to the number of unoccupied cells in this rectangle:

$$P^i_c = \frac{1 - H_c\left(D(R_i, S_i)\right)}{\sum_{j=1}^{m}(1 - H_c\left(D(R_j, S_j)\right))}, \tag{6}$$

where $H_c\left(D(R_i, S_i)\right)$ is the *local cell occupancy* of R_i, i.e., $H_c\left(D(R_i, S_i)\right) = \frac{\log(N_{ci})}{\log(l_i)}$. where l_i is the number of grid cells intersecting with R_i and N_{ci} is the number of unoccupied cells intersecting with R_i.

Robustness Based Probability Distribution. A probability distribution takes into consideraton the average robustness as well as the potential reduction in robustness below the average. The potential reduction in robustness below the average is defined as $\lambda_i = \frac{1}{|S_i|}\sum_{x \in S_i} \max(\mu_i - \rho(x), 0)$. Then a potentially reduced robustness value below the average is $\theta_i = \mu_i - \lambda_i$. Then we define a robustness based probability in a rectangle R_i as inversely proportional to θ_i, as follows.

$$P^i_r = \frac{\frac{1}{\theta_i}}{\sum_{j=1}^{m} \frac{1}{\theta_j}}. \tag{7}$$

Sampling Probability Distribution. The probability distribution for sampling is a weighted sum of the probability based on robustness and the probability based on coverage. The weightage given to either probability is a user defined constant. Let the weight assigned to the robustness based probability be denoted by w_r such that $w_r \in [0, 1]$. Then, the overall probability of sampling in rectangle R_i is

$$P_t^i = w_r P_r^i + (1 - w_r) P_c^i. \tag{8}$$

Singularity Based Sampling. Certain rectangles may contain samples whose robustness is very low compared to the lowest robustness values in other rectangles. We refer to them as *singular samples*, which we heuristically define as follows.

Let $\gamma = \{\gamma_1, \ldots, \gamma_k\}$ be the vector of lowest robustness values in each rectangle, defined as $\gamma_i = \min_{x \in S_i} \rho(x)$. The mean of γ and the average deviation below the mean are respectively defined as $\mu_\gamma = \frac{\sum_{i=1}^{k} \gamma_i}{k}$ and $\lambda_\gamma = \frac{\sum_{i=1}^{k} max\,(0, (\mu_\gamma - \gamma_i))}{k}$. If the robustness of a sample is less than λ then it is an indication that the sample may be close to a counterexample. Also, samples with very low frequency and sufficiently low robustness are also considered singular. To select such rare samples, we use the following heuristic. If γ were a large set of random samples selected from a normal distribution, then less than 15% of the samples tend to lie below the value $\mu_\gamma - 3\lambda_\gamma$. Although the actual set of samples in γ may not follow the pattern of a normal distribution, this also can be used as a heuristic to define a singular sample.

Definition 2. *A point $x \in \bigcup_{i=1}^{k} S_i$ for which $\rho(x) \leq max\,(\mu_\gamma - 3\lambda_\gamma, \lambda)$ is called a singular sample.*

We call the rectangles containing singular samples as *singular rectangles.* Since the frequency of singular samples can be very small, the robustness based probability in (7) may not give adequate weightage to singular rectangles. So, we have to perform additional sampling in the singular rectangles.

Overall Global Search. Suppose that we have a partition of rectangles $R_1, \ldots, R_k$ containing sets of samples $S_1, \ldots, S_k$, respectively. Let C_i be the set of unoccupied cells intersecting with a rectangle R_i, i.e., $C_i = \{\omega_j \in \omega : \omega_j \cap R_i \neq \emptyset \wedge \omega_j \cap S_i = \emptyset\}$. Let N be the number of samples to be added during probabilistically biased random sampling. We compute the probability distribution of sampling among different rectangles P^t, where the user defines a weightage w_r given to the robustness based probability distribution P^r. Then we select $(\min\{\max\{1, \lfloor P_i^t N \rfloor\}, |C_i|\})$ number of cells among C_i and sample one point in each cell. Note that if there is an unoccupied cell in a rectangle, then at least one sample is added to each rectangle irrespective of the probability P_i^t. Then update the sets of samples $S_1, \ldots, S_k$ by adding the new samples and also the sets of unoccupied cells C_i $\forall i \in \{1 \ldots k\}$.

Next, we perform sampling in each of the singular rectangles as follows. Let R_j be a singular rectangle, currently containing the set S_j of samples and a set C_j of unoccupied cells. Then we select $\min\{\max\{K_c - |S_j|, 0\}, |C_j|\}$ cells among the unoccupied cells C_i. Therefore, in the next iteration R_j contains at least K_c samples (if it has unoccupied cells) and is consequently subdivided. The procedure is repeated in each iteration until the time limit $\overline{T}_g$ on global search is reached. Alternatively, we can also set a limit on the total number of samples for which robustness is evaluated. If we have not falsified yet, then we perform a number of local searches initialized at the lowest robustness samples of all the separate subrectangles, as described below.

4.3 Local Search

Suppose that we have K subrectangles $R_1, \ldots, R_K$ after running global search for $\overline{T}_g$ time. Let L be the set of the lowest robustnes points of different rectangles. If the property is not yet falsified, then we use the lowest robustness points of different rectangles to initialize a local search based falsification algorithm. In our implementation, we used the state-of-the-art *Covariance matrix adaptive evolutionary search* (CMA-ES) algorithm [21] in the local search phase. The essence of the CMA-ES algorithm can be briefly described as follows. It is a randomized black box method which selects samples based on a multivariate normal distribution having a mean and a covariance matrix as parameters. Based on the robustness of a population of points evaluated in an iteration, the mean and covaraiance matrix of the search distribution are updated for the next sampling iteration. The procedure generally coverges to a locally optimum point or finds a counterexample. It may happen that the set of sampled points do not contain enough information to derive a reliable estimation of a covariance matrix for an efficient update. Therefore, good initializations of the mean and covariance matrix are crucial. In our algorithm, the global search provides initialization guidance as follows. We have a number of subrectangles formed by classification, that contain sets of samples. So, we can initialize in one the following ways: (1) Each of the lowest robustness points of different rectangles, i.e., the points in L can be selected for initialization with covariance as identity (the order of selection is according to their robustness values, with the lowest robustness tested first); (2) The mean and covariance are initialized as that of those points in L whose robustness is less than the average robustness of samples in L. (3) The mean and covariance are initialized as that of all the points in L. With such initialization guidance from our global search procedure described earlier, this local search procedure can become more effective in falsification.

4.4 Overall Falsification Algorithm

The overall falsification algorithm consists of iteratively doing global search for a threshold time and then doing local search. We give an outline of the algorithm below.

- Step 1: *Initialization.* In the first step, we evaluate the robustness of N uniformly selected points in the search space R and store them as a set of samples S.
- Step 2: *Global search phase*: We perform a number of global search iterations until a time limit is attained. Each global search iteration consists of the following three steps executed one after the other. (i) The first step is classification, where new rectangles are constructed by classifying and consequently subdividing the existing rectangles that contain more than a threshold K_c of samples. (ii) The second step involves probabilistically biased sampling based on the coverage and robustness values of samples in different rectangles. The specific procedure is explained earlier in the Section on overall global search. (iii) The third step is singularity based sampling. This procedure is also explained earlier in the Section on overall global search.
- Step 3: *Local search phase.* If no counterexample is found in the global search phase, then we perform the local search based on the set of low robustness points in different rectangles. The specific procedure is explained earlier in the Section on local search.
- Step 5: If not falsified during local search, then continue global search iterations i.e. go to Step 3.
- Step 6: If not falsified, then alternate with local search, i.e. go to Step 4.

We can now state an important completeness property of our overall falsification algorithm for the class of the systems (2) satisfying the assumption that the mapping $\widehat{\Phi}$ is continuous in the nominal parameters and the input signal parameters.

Theorem 1. *If an ϵ-robust counterexample exists, then there exists a grid cell size Δ and a global search time $\overline{T}_g$ so that our algorithm finds a counterexample.*

Sketch of proof. The theorem can be directly established from Lemma 1. Indeed, the condition in this lemma is always satisfied by our algorithm since, by construction, after each iteration the cell occupancy of the samples always strictly increases. So, for sufficient $\overline{T}_g$, the falsification is guaranteed if an ϵ-robust counterexample exists.

5 Experimental Results

In our experiments, we compare the performance of a MATLAB implementation[2] with the following standard approaches: CMA-ES, Simulated Annealing, Global Nelder-Mead algorithm implementations (integrated in Breach [9]), and the S-TaLiRo tool [2] by setting Simulated Annealing as optimization algorithm[3].

[2] We use the robustness evaluation function from the Breach toolbox available in October 2016, on the site https://people.eecs.berkeley.edu/~donze/breach_page.html.

[3] We used the latest version available in October 2016, on the site https://sites.google.com/a/asu.edu/s-taliro/home.

Experiments were performed on a computer with 1.4 GHz processor with 4 GB RAM, running MATLAB R2015 64-bit version. Also, we compare with a random sampling method, where in each iteration a pseudo-randomly selected point is tested only if it falls in a grid cell wherein no other point has been previously tested. The grid used in this method is the same as the grid chosen in our falsification approach. We will call this method as grid based random sampling, for the sake of reference during comparison.

5.1 Automative Powertrain Control

We consider a Simulink model of a closed loop of an Automative Powertrain Control subsystem (PTC). The model contains a representation of an internal combustion engine and an embedded software controller for the air-to-fuel ratio within the engine (see [11] for more details). Here, we focus on the input-output behavior, considering the internal model as a blackbox. The model has three input signals, Pedal Angle Engine Speed and Sensor Offset. The air-to-fuel (A/F) ratio, denoted by η, is an output signal for which the following safety requirement was stated in [11]: $\phi = \Box_{[5,10]}\,(\eta < 0.5)$.

Input Signal Settings. Compared to [11], we consider a smaller input range for the Pedal Angle as $[0, 40]$ and fix the Engine Speed and Sensor Offset as 1000 and 1, respectively. Reducing the ranges makes the properties more robust and consequently difficult to falsify. The time horizon is 50s. We use piecewise constant signal for testing, where the Pedal Angle is parameterized by 10 uniformly spaced control points in the time horizon. Thus, we have a 10 dimensional search space X.

Algorithm Setting. For our algorithm, the threshold number of samples for hyperplane classification K_c is 100. The global search time is $\overline{T}_g = 2000s$. The local search is initialized with the lowest robustness point found during global search and allowed to run until falsification. Cell partitioning ω consists of hypercubes of side length $\epsilon = 4$. We consider equal weightage for robustness based probability and coverage based probability for sampling during global search, i.e., $w_r = 0.5$.

Results. Our algorithm (classification guided global search + local search) successfully found a counterexample in less than 3000s for all seeds. As an estimate of the classification frequency, the final number of separate rectangles constructed for while testing the first seed were 30. In comparison, the tool S-TaLiRo could falsify but took 4481s. The grid based random sampling found a falsifier for only the seeds 15000 and 20000, but failed to do so on the other seeds before maximum time limit was reached. The other methods were not successful in finding a falsifier within the default stopping time of 5000s. Both the CMA-ES and Nelder-Mead became stuck without reduction in robustness value until the default stopping time was reached. The results are presented in Table 1. We note that for any fixed seed for random sampling, these results are reproducible.

5.2 Automatic Transmission

We consider the benchmark model of an Automatic Transmission control system, which appeared in [20][4]. The system has two input signals, called throttle and break, respectively, and two output signals, called the engine speed, denoted w (RPM), and the vehicle speed, denoted v (mph). The property states that if the engine speed stays below a value $\overline{w}$, then the vehicle speed v does not exceed a threshold $\overline{v}$ within 10s. We specify the values of $\overline{w}$ and $\overline{v}$ to be 2520 and 50, respectively, which gives the following STL property: $\phi = \neg \big((\Diamond_{[0,10]} v > 50) \wedge (\Box w \leq 2520) \big)$ [20].

Input Signal and Parameter Settings. Initially, the vehicle is at rest, when $v = 0$ and $w = 0$. For the input signals, we consider smaller ranges than specified in [20], which makes the property ϕ more robust. Henceforth, the throttle signal is allowed to vary between $[35, 100]$ and the break is allowed to vary between $[0, 40]$. The time horizon is set to 30s. We use piecewise constant input signals for testing, where the throttle signal is parametrized by 7 control points and the break has 3 control points. Thus, we have a $7 + 3 = 10$ dimensional search space. ϕ.

Algorithm Setting. For our algorithm, the threshold number of samples of hyperplane classification K_c is 70. The global search time is $\overline{T}_g = 500$s, while maximum time for local search is $\overline{\tau}_l = 2000$s. Cell partitioning ω consists of hypercubes of side length $\epsilon = 4$. We consider equal weightage for robustness based probability and coverage based probability for sampling during global search, i.e., $w_r = 0.5$.

Results. Our algorithm (classification guided global search + local search) successfully found a counterexample in less than 2000s for all tested seeds. As an indication of the number of classification operations that occurred, the final number of separate rectangles constructed for while testing the first seed were 31. In comparison, the CMA-ES found a falsifier for two seeds 5000 and 15000 within 2000s but failed to do so on the other seeds. The other methods were not successful in finding a falsifier within the default stopping time of 3000s. For this example, S-TaLiRo became stuck around a local optimum without any significant reduction in robustness value. The results are presented in Table 1. We note that for any fixed seed for random sampling, these results are reproducible.

5.3 Industrial Example

We present results for an air path controller for an automotive fuel cell (FC) application. The system contains an FC stack that generates electrical power to provide torque to the vehicle drivetrain. The system is composed of an air compressor and the air path through the FC stack. The system takes as input

[4] The model and property description of this benchmark is available at the site of the workshop Applied Verification for Continuous and Hybrid Systems, ARCH 2014–2015, http://cps-vo.org/node/12116.

Table 1. Experimental results

Solver	Seed	Computation time (secs)		Falsification	
		PTC	Aut. Trans	PTC	Aut. Trans
Hyperplane classification + CMA-ES-Breach	0	2891	996	✓	✓
	5000	2364	1382	✓	✓
	10000	2101	1720	✓	✓
	15000	2271	1355	✓	✓
CMA-ES-Breach	0	T.O. (5000)	T.O. (2000)		
	5000	T.O. (5000)	1302		✓
	10000	T.O. (5000)	T.O. (2000)		
	15000	T.O. (5000)	1325		✓
Grid based random sampling	0	T.O. (5000)	T.O. (2000)		
	5000	T.O. (5000)	T.O. (2000)		
	10000	3766	T.O. (2000)	✓	
	15000	268	T.O. (2000)	✓	
S-TaLiRo (Simulated Annealing)		4481	T.O. (3000)	✓	
S-TaLiRo (Simulated Annealing)		4481	Default stopping (3300)	✓	

T.O.: Exceeded indicated time out limit.
Seed: Index for a sequence of random numbers in MATLAB. *Solver*: Algorithm used for falsification. *Computation time*: Amount of time (in seconds) until falsification or default stopping after the time limit in parentheses. Computation time is reported for a computer with 1.4 GHz processor and 4 GB RAM, running MATLAB R2015 64-bit version. *Falsification*: Boolean variable indicating whether the algorithm could falsify the property.

requested current from the stack and ambient temperature. The outputs are desired air flow rate and the measured air flow rate through the FC stack. The goal is for the stack air flow rate to maintain accurate regulation when current request "disturbances" are presented to the system. System performance (called *responsiveness*) crucially depends on accurate and timely regulation of the air flow to the commanded reference. The corresponding specification for the system can be described informally as follows: when there is a step input of current request, there is a rise-time requirement on the output air flow that should be satisfied. Details about the system and the specifications are proprietary and so are suppressed here.

We analyze a Simulink model of the FC system, which contains representations of the FC system along with its controller. The model is complex, containing several thousands of Simulink blocks; simulations over the selected time horizon are expensive to perform, each taking approximately 1 to 2 min. The MATLAB implementation of the hyperplane classification algorithm with local search is applied to the model, and the results are compared to the same algorithms used in Sects. 5.1 and 5.2.

For our method, we performed the tests using two different cell partitions. Cell partition A is large and corresponds to a small number of grid elements; cell partition B is smaller (each dimension of the search space is 1/5 the size of the grid elements in partition A). Thus, partition B corresponds to a significantly larger number of grid elements.

Table 2 provides the results. As can be seen in the table, using cell partition A with our method performs much better than with partition B. This can be attributed to the fact that, for partition A, the classification phase of the search spends less time in regions close to regions that have already been explored, as compared to partition B. This demonstrates that the selected cell partition size has a significant impact on the performance of our technique.

Table 2. Results for fuel cell example.

Solver	Seed	Computation time (sec.)	Falsification
Hyperplane classification + CMA-ES-Breach (cell partition: A)†	1	406	✓
	2	1383	✓
	3	T.O.	
	4	794	✓
Hyperplane classification + CMA-ES-Breach (cell partition: B)†	1	409	✓
	2	T.O.	
	3	T.O.	
	4	T.O.	
CMA-ES Breach†	1	314	✓
	2	1418	
	3	T.O.	
	4	1316	✓
Uniform random† sampling	1	396	✓
	2	786	✓
	3	2241	✓
	4	T.O.	
S-TaLiRo (Simulated Annealing)‡ sampling	1	310	✓
	2	T.O.	
	3	671	✓
	4	T.O.	
Global Nelder-Mead-Breach†		1501	✓

T.O.: Exceeded time out limit of 2700 s.

†: Times reported are from machines running Dell Precision, with a Xeon processor (2.13 GHz), with 24 GB of RAM, running a 64 bit version of Windows 7 Ultimate, SP1.

‡: Times reported are from machines running Dell Precision, with a Xeon processor (2.3 GHz), with 64 GB of RAM, running a 64 bit version of Windows 7 Ultimate, SP1.

Also, the table shows that the CMA-ES fails to find falsifying behaviors in 2 of the 4 cases, which demonstrates better performance than our technique using partition B but poorer performance than our technique using partition A. The uniform random sampling approach is able to find falsifying traces in all but one case, and the computation times for the successful cases are comparable to our technique using partition A, though we note that the computation times for our technique are lower than the uniform random method, for the cases where falsifying traces are found. The S-TaLiRo approach fails to find falsifying traces in 2 of the 4 cases, which is less than the number of times our technique is successful, using partition A. The Nelder-Mead algorithm is able to identify a falsifying trace in about 25 min, which is longer than the 3 successful cases of our technique, using partition A.

The above results show mixed results for our technique for this example, as compared to the other falsification approaches. This could be due to any of several factors. We observe that for this example, comparing against the falsification techniques that we selected, only a relatively small number of simulations are required to find falsifying traces, when they are found at all. This may suggest that either the model is not robust, in the sense that there may be many disconnected regions in the search space that correspond to falsifying behaviors, or that the robustness function is rather monotone or simple. It may be that for systems with these qualities, the benefits provided by the hyperplane classification approach are outweighed (or at least offset) by the overhead that it requires.

6 Conclusions

We have presented a novel falsification algorithm that maintains a balance between convergence towards low robustness points and enhancing global coverage. We accomplish this by intelligently subdividing the search space and subsequently biasing the density of random sampling in different sub-regions. For the subdivision, we use hyperplane classifiers akin to support vector machines, which tries to focus effort on low robustness regions of the search space. We demonstrated the efficiency of our algorithm by falsifying properties on benchmark examples, which other approaches failed to falsify. Also, we demonstrated that the approach could be applied to industrial systems by describing a successful application on an automotive hydrogen fuel cell example. Future work includes investigating new coverage measures, such as the combinatorial entropy notion from the domain of physics to measure the degree of randomness in the distribution of points. In addition, global search and local search can be done in a multi-resolution manner, that is if local search leads to a promising region, global search can then be done within the region using a more refined grid.

References

1. Althoff, M., Krogh, B.: Zonotope bundles for the efficient computation of reachable sets. In: 2011 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC), pp. 6814–6821, December 2011
2. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: a tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 254–257. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19835-9_21
3. Bouissou, O., Goubault, E., Putot, S., Tekkal, K., Vedrine, F.: HybridFluctuat: a static analyzer of numerical programs within a continuous environment. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 620–626. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02658-4_46
4. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: an analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 258–263. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39799-8_18
5. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
6. Dang, T., Nahhal, T.: Coverage-guided test generation for continuous and hybrid systems. Formal Methods Syst. Des. **34**(2), 183–213 (2009)
7. Deshmukh, J., Jin, X., Kapinski, J., Maler, O.: Stochastic local search for falsification of hybrid systems. In: Finkbeiner, B., Pu, G., Zhang, L. (eds.) ATVA 2015. LNCS, vol. 9364, pp. 500–517. Springer, Cham (2015). doi:10.1007/978-3-319-24953-7_35
8. Dietterich, T.G., Lathrop, R.H., Lozano-Pérez, T.: Solving the multiple instance problem with axis-parallel rectangles. Artif. Intell. **89**(1), 31–71 (1997)
9. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 167–170. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14295-6_17
10. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 92–106. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15297-9_9
11. Dreossi, T., Dang, T., Donzé, A., Kapinski, J., Jin, X., Deshmukh, J.V.: Efficient guiding strategies for testing of temporal properties of hybrid systems. In: Havelund, K., Holzmann, G., Joshi, R. (eds.) NFM 2015. LNCS, vol. 9058, pp. 127–142. Springer, Cham (2015). doi:10.1007/978-3-319-17524-9_10
12. Dreossi, T., Dang, T., Piazza, C.: Parallelotope bundles for polynomial reachability. In: Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, HSCC 2016, Vienna, Austria, 12–14 April 2016, pp. 297–306 (2016)
13. Esposito, J.M., Kim, J., Kumar, V.: Adaptive RRTs for validating hybrid robotic control systems. In: Erdmann, M., Overmars, M., Hsu, D., van der Stappen, F. (eds.) Algorithmic Foundations of Robotics VI. STAR, vol. 17, pp. 107–121. Springer, Heidelberg (2005). doi:10.1007/10991541_9
14. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications. In: Havelund, K., Núñez, M., Roşu, G., Wolff, B. (eds.) FATES/RV-2006. LNCS, vol. 4262, pp. 178–192. Springer, Heidelberg (2006). doi:10.1007/11940197_12
15. Fan, C., Qi, B., Mitra, S., Viswanathan, M., Duggirala, P.S.: Automatic reachability analysis for nonlinear hybrid models with C2E2. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 531–538. Springer, Cham (2016). doi:10.1007/978-3-319-41528-4_29

16. Frehse, G., Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22110-1_30
17. Fung, G.M., Mangasarian, O.L., Shavlik, J.W.: Knowledge-based support vector machine classifiers. In: Advances in Neural Information Processing Systems, pp. 521–528 (2002)
18. Gao, S., Avigad, J., Clarke, E.M.: δ-complete decision procedures for satisfiability over the reals. In: Joint Automated Reasoning, pp. 286–300 (2012)
19. Hoos, H., Sttzle, T.: Stochastic Local Search: Foundations & Applications. Morgan Kaufmann Publishers Inc., San Francisco (2004)
20. Hoxha, B., Abbas, H., Fainekos, G.E.: Benchmarks for temporal logic requirements for automotive systems. In: 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems, ARCH@CPSWeek 2014, Berlin, Germany, 14 April 2014/ARCH@CPSWeek 2015, Seattle, WA, USA, 13 April 2015, pp. 25–30 (2014)
21. Igel, C., Suttorp, T., Hansen, N.: A computational efficient covariance matrix update and a (1+1)-CMA for evolution strategies. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation GECCO, pp. 453–460. ACM (2006)
22. Kuřátko, J., Ratschan, S.: Combined global and local search for the falsification of hybrid systems. In: Legay, A., Bozga, M. (eds.) FORMATS 2014. LNCS, vol. 8711, pp. 146–160. Springer, Cham (2014). doi:10.1007/978-3-319-10512-3_11
23. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT-2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30206-3_12
24. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Pearson Education, Upper Saddle River (2003)
25. Skruch, P.: A coverage metric to evaluate tests for continuous-time dynamic systems. Cent. Eur. J. Eng. **1**(2), 174–180 (2011)
26. Testylier, R., Dang, T.: NLTOOLBOX: a library for reachability computation of nonlinear dynamical systems. In: Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 469–473. Springer, Cham (2013). doi:10.1007/978-3-319-02444-8_37