Efficient Guiding Strategies for Testing of Temporal Properties of Hybrid Systems

Tommaso Dreossi[1], Thao Dang[1], Alexandre Donzé[2],
James Kapinski[3], Xiaoqing Jin[3], and Jyotirmoy V. Deshmukh[3]

[1] Verimag {tommaso.dreossi, thao.dang}@imag.fr
[2] University of California, Berkeley donze@berkeley.edu
[3] Toyota Technical Center firstname.lastname@tema.toyota.com

Abstract. Techniques for testing cyberphysical systems (CPS) currently use a combination of automatic directed test generation and random testing to find undesirable behaviors. Existing techniques can fail to efficiently identify bugs because they do not adequately explore the space of system behaviors. In this paper, we present an approach that uses the rapidly exploring random trees (RRT) technique to explore the state-space of a CPS. Given a Signal Temporal Logic (STL) requirement, the RRT algorithm uses two quantities to guide the search: The first is a robustness metric that quantifies the degree of satisfaction of the STL requirement by simulation traces. The second is a metric for measuring coverage for a dense state-space, known as the star discrepancy measure. We show that our approach scales to industrial-scale CPSs by demonstrating its efficacy on an automotive powertrain control system.

1 Introduction

Model-Based Development (MBD) for cyberphysical systems (CPS) is a paradigm based on an end-to-end use of high-level executable models of physical systems interacting with software. These models facilitate a wide array of design and analysis techniques such as accurate simulation, control design, test generation, and code generation. This allows validating the behavior of the CPS early in development cycles, which not only enhances product reliability but also brings significant cost savings.

Techniques for testing CPSs typically employ a combination of automatic directed test generation and random testing [23, 20] to detect undesirable behaviors. The prevalent practice is to use coverage metrics such as Modified-Condition Decision Coverage (MCDC) inspired from software testing [23]. Such metrics help quantify the degree to which the models of software components of the CPS (typically embedded/control software) have been tested. Models of CPS, however, often contain physics-based models (also called plant models) tightly coupled with the models of embedded software. In contrast to models of control software (that are akin to standard computer programs), plant models typically represent behaviors that evolve over continuous time and state-space. Existing code-coverage metrics are thus not applicable when reasoning about the "coverage" of the possibly infinite state-space of a CPS. Furthermore, the idea of using temporal logic to specify behavioral specifications of CPSs has

been gaining traction [11, 16, 24]. Existing directed testing tools are inadequate as they typically try to generate test inputs to satisfy code-coverage criteria or inputs to violate static assertions in the model [23, 20].

In this paper, we propose a testing-based approach that hopes to fill these two lacunae. It builds on two recent results: the first is a hybrid systems test generation technique that is based on the rapidly exploring random trees (RRT) algorithm and guided by the star-discrepancy coverage measure [5]. The other is a technique for robust monitoring of properties expressed in Signal Temporal Logic (STL) [7, 8]. The combination of coverage and robustness analysis allows increasing the effectiveness of generated test suites in terms of error detection.

Another goal of this work is to create a tool that supports industrial CPS models by considering modeling environments that are prevalent in industry, such as Simulink®. Simulink has become a *de facto* standard in the automotive engine controls domain. It is used as an MBD platform to perform system modeling, simulation, and automatic code generation. Tools that can identify design-bugs in Simulink models offer a significant benefit over testing-based techniques, as the cost of addressing problems during the later stages of development is significantly greater than at the modeling stage.

Our contribution can be summarized as follows. Given a user-provided temporal logic specification φ, our technique can automatically identify examples of system behaviors that falsify φ. Our method uses an RRT algorithm, guided by a combination of two metrics: a metric quantifying state-space coverage and a metric quantifying the robust satisfaction degree of the given specification. We provide experimental results that compare the performance of our implementation with the falsification tool S-TaLiRo.

Related Work Our technique brings together two lines of work. The first is the development of efficient sampling-based exploration algorithms. The use of Rapidly-exploring Random Trees (RRT) has been popularized in the context of path planning in robotics [17, 13] and applied and extended later for falsification of safety properties, using the observation that the latter reduces to finding a path from an initial state to some unsafe state [14, 4, 22].

The second line of work is falsifying temporal specifications on CPS models. This is a very active area of research, and tools based on search guided by stochastic and nonlinear optimization (as implemented in the tools S-TaLiRo [1] and Breach [6]) have been already applied in an industrial context to Simulink models [11, 9]. There are two key differences between falsification tools and our approach. First, S-TaLiRo and Breach essentially assume a parameterized representation of input signals, and at the beginning of a simulation run, fix a valuation for the parameters.

By contrast, our RRT-based algorithm allows the flexibility to change input values based on robustness values of a partial system trajectory; in Section 5, we demonstrate how this flexibility leads to superior falsification performance compared to S-TaLiRo for some examples. The second difference is that falsification tools typically do not measure the coverage of the hybrid state-space by the set of test inputs explored, while our approach uses the star-discrepancy based coverage metric to quantify coverage.

The dual of falsification, the problem of control with temporal logic specifications, has been considered recently [15, 12]. The work in [3] is similar to our work, in that RRT algorithms are considered to derive control inputs enforcing temporal logics goals. However, most of the work in this area assumes either simpler (linear) dynamical models, or simpler specification formalisms such as LTL. To the best of our knowledge, our work is the first to consider the problem of STL falsification based on a modified RRT algorithm.

2 Preliminaries

2.1 Dynamical System Model

We assume that a system model $\mathcal{M}$ is specified as a gray-box, that is, we assume we know qualitative information about $\mathcal{M}$ (e.g., the number of state variables) but we do not know detailed information (e.g., the closed-form analytic representation of system dynamics). We assume that the underlying system described by the gray-box is a continuous-time hybrid system. We also assume that the system is provided with a *simulator* that takes as input discrete-time input sequences, and returns discrete-time output sequences.

The system model $\mathcal{M}$ is defined as a tuple $(\mathcal{X}, \mathcal{U}, \mathsf{sim})$, where $\mathcal{X}$ is a finite or infinite set of states, $\mathcal{U}$ is a finite or infinite set of input values. Let $\boldsymbol{x}$ denote a function that maps a given time point t over a given time domain to the state $\boldsymbol{x}(t)$ of $\mathcal{M}$ at time t; we also call $\boldsymbol{x}$ as the *state variable*. Let $\boldsymbol{u}$ denote a function mapping a given time point t to the input value $\boldsymbol{u}(t) \in \mathcal{U}$; we also call this function an *input signal*. The function sim maps a state and an input at a given time t_k, i.e., $\boldsymbol{x}(t_k)$ and $\boldsymbol{u}(t_k)$, and a rational time-step $h_k > 0$, to a new state at time $t_{k+1} = t_k + h_k$. In other words, $\mathcal{M}$ can be viewed as a nonautonomous discrete-time dynamical system, with the update function given by sim:

$$\boldsymbol{x}(t_{k+1}) = \mathsf{sim}(\boldsymbol{x}(t_k), \boldsymbol{u}(t_k), h_k) \tag{1}$$

In general, $\mathcal{X}$ can be a product of n different domains (such as the Boolean $\{true, false\}$ domain, $\mathbb{Z}$, $\mathbb{R}$, etc.). We say that the state dimension of $\mathcal{M}$ is n. Similarly, $\mathcal{U}$ can be a product of m different input domains, and the input dimension is called m.

A *simulation trace* of the model $\mathcal{M}$ is defined as a sequence of times and pairs of state and input values:

$$(t_0, \boldsymbol{x}(t_0), \boldsymbol{u}(t_0)),\ (t_1, \boldsymbol{x}(t_1), \boldsymbol{u}(t_1)),\ \ldots,\ (t_N, \boldsymbol{x}(t_N), \boldsymbol{u}(t_N))$$

where $\forall i : \boldsymbol{x}(t_{i+1}) = \mathsf{sim}(\boldsymbol{x}(t_i), \boldsymbol{u}(t_i), h_i)$, and $t_{i+1} = t_i + h_i$.

We stress that we do not require the function sim to be known in an analytic or symbolic form, but assume that there is a simulator that returns the states computed by sim. Formally, a *simulator* is a program that, given an initial time t_0, an initial state value $\boldsymbol{x}(t_0)$, a sequence of non-zero time-steps $h_0, \ldots, h_N$, and a sequence of input values $\boldsymbol{u}(t_0), \ldots, \boldsymbol{u}(t_N)$ is able to compute the corresponding simulation trace of $\mathcal{M}$.

In the case of Simulink models, the simulator can be provided with a fixed time step h_i. Simulation traces are computed by performing numerical integration of the differential (or difference) algebraic equations corresponding to the continuous or hybrid dynamical systems that the models describe.

Simulator assumptions We assume that the inputs of $\mathcal{M}$ are controllable by the user and the full state of $\mathcal{M}$ is observable. We also assume that it is possible to reset any stateful element of the model to a permissible value in $\mathcal{X}$. These assumptions are meaningful in the context of model-based testing as they allow a user or a test program to start a simulation trace from an arbitrary initial state. Note that, in theory, our technique could be applied to a testing scenario, where the result of sim is provided by performing a particular test case. While our assumptions are plausible for this case, they would be quite difficult to enforce, as observing and resetting the state of the system is often difficult or impossible in a testing scenario.

2.2 Signal Temporal Logic

We require a formal language to specify how a model $\mathcal{M}$ is expected to behave and employ Signal Temporal Logic (STL) for this purpose. STL was proposed in [19] as a specification language for properties of signals (i.e., functions from a time domain to some domain of values). In the following we present the syntax and semantics of STL for a continuous time domain ($\mathbb{R}_0^+$). In practice, STL semantics have to be adapted to discrete time signals (such as the simulation traces of a system $\mathcal{M}$) on a dense time domain by use of linear interpolation (i.e., for t between t_k, t_{k+1}, we define $\boldsymbol{x}(t)$ by linear interpolation between $\boldsymbol{x}(t_k)$ and $\boldsymbol{x}(t_{k+1})$).

An STL formula is formed of *atomic predicates* connected with Boolean and *temporal* operators. Atomic predicates can be reduced to inequalities of the form $\mu = f(\boldsymbol{x}) \sim 0$, where f is a scalar-valued function over the signal $\boldsymbol{x}$, $\sim \in \{<, \leq, \geq, >, =, \neq\}$. Temporal operators are "always" (denoted as $\Box$), "eventually" (denoted as $\Diamond$) and "until" (denoted as $\mathcal{U}$). Each temporal operator is indexed by intervals of the form (a, b), $(a, b]$, $[a, b)$, $[a, b]$, (a, ∞) or $[a, \infty)$ where each of a, b is a non-negative real-valued constant, and $a < b$. If I is an interval, then an STL formula is written using the following grammar:

$$\varphi := \top \mid \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \,\mathcal{U}_I\, \varphi_2$$

The always and eventually operators are defined as follows: $\Box_I \varphi \triangleq \neg\Diamond_I \neg\varphi$, $\Diamond_I \varphi \triangleq \top \,\mathcal{U}_I\, \varphi$. When the interval I is omitted, we use the default interval of $[0, +\infty)$. The semantics of STL formulas are defined informally as follows. The signal $\boldsymbol{x}$ satisfies $f(\boldsymbol{x}) > 0$ at time t (where $t \geq 0$) if $f(\boldsymbol{x}(t)) > 0$. It satisfies $\varphi = \Box_{[0,2)}\ (x - 1 > 0)$ if for all time $0 \leq t < 2$, $x(t) - 1 > 0$. The signal x_1 satisfies $\varphi = \Diamond_{[1,2)}\ x_1 + 0.5 > 0$ iff there exists time t such that $1 \leq t < 2$ and $x_1(t) > -0.5$. The two-dimensional signal $\boldsymbol{x} = (x_1, x_2)$ satisfies the formula $\varphi = (x_1 > 0)\ \mathcal{U}_{[2.3,4.5]}\ (x_2 < 0)$ iff there is some time t where $2.3 \leq t \leq 4.5$ and $x_2(t) < 0$, and $\forall t'$ in $[2.3, t)$, $x_1(t')$ is greater than 0. The complete semantics is given in the appendix.

Quantitative semantics We define a quantitative semantics of STL as a function ρ such that the sign of $\rho(\varphi, \boldsymbol{x}, t)$ determines whether $(\boldsymbol{x}, t)$ satisfies φ and its absolute value estimates the *robustness* of this satisfaction. A common way of defining such functions, as presented in [7], is as follows. For a predicate $\mu =$

$x > 2$, one can simply use the value $x(t) - 2$ as a robustness estimate. For the conjunction of two formulas $\varphi = \varphi_1 \wedge \varphi_2$, with robustness ρ_1 and ρ_2, we can use $\rho = \min(\rho_1, \rho_2)$. For $\Diamond_{[0,2]}\ (x - 1 > 0)$, the robustness at time 0 can be estimated by the maximum for $t \in [0, 2]$ of $x(t) - 1$. The inductive formal definition of ρ is given in the appendix.

3 Coverage-Based Testing

The original RRT algorithm is a technique for quickly exploring the state space for systems with differential constraints [18]. In this section, we recall the gRRT version of RRT, which is a testing approach used to maximize state space coverage [5]. The gRRT 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.

We will now present how the algorithm is applied to a Simulink model. The procedure takes as input a Simulink model $\mathcal{M}$, an initial state value $\boldsymbol{x}_{init} \in \mathcal{X}$, and an iteration limit k_{max}. The function sample samples a goal-state $\boldsymbol{x}_{goal}$ from $\mathcal{X}$. The goal-state is intended to indicate the direction towards which the tree is expected to evolve. Then, a starting state $\boldsymbol{x}_{near}$ is determined as a neighbor of $\boldsymbol{x}_{goal}$ using some predefined distance. The point $\boldsymbol{x}_{near}$ is expanded towards $\boldsymbol{x}_{goal}$ as follows:

- The function findInput is used to select a sample from the input set $\boldsymbol{u} \in \mathcal{U}$. This can be performed by randomly selecting a point in $\mathcal{U}$.
- The initial condition for $\mathcal{M}$ is set to $\boldsymbol{x}_{near}$, and a simulation is performed for h seconds using the input $\boldsymbol{u}$. A new edge from $\boldsymbol{x}_{near}$ to $\boldsymbol{x}_{new}$, labeled with the associated input $\boldsymbol{u}$, is then added to the tree.

Unlike the original RRT algorithm, the goal state sampling in the gRRT algorithm is not uniform, and the function sample is used to guide the exploration to improve the star-discrepancy coverage, which we define below.

3.1 Star-Discrepancy Coverage

The star discrepancy is a notion in equidistribution theory (see for example [2]) that has been used in quasi-Monte Carlo techniques for error estimation. Let P be a set of k points inside $\mathcal{B} = [l_1, L_1] \times \ldots \times [l_n, L_n] \subset \mathbb{R}^n$. Let $p = (\beta_0, \ldots, \beta_n)$ be a point inside $\mathcal{B}$, this point together with the bottom left vertex of $\mathcal{B}$ forms a subbox $J = [l_1, \beta_1] \times \ldots \times [l_n, \beta_n]$. The local discrepancy of the point set P with respect to the sub-box J is:

$$D(P, J) = \Big| \frac{A(P, J)}{k} - \frac{vol(J)}{vol(\mathcal{B})} \Big| \tag{2}$$

where $A(P, J)$ is the number of points of P that are inside J, and $vol(J)$ is the volume of the box J. We use $\mathcal{J}$ to denote the set of all subboxes J such that $l_i \leq \beta_i \leq L_i$ for each $i \in 1, \ldots, n$. The star discrepancy of a point set P with respect to the box $\mathcal{B}$ is defined as:

$$D^*(P, \mathcal{B}) = \sup_{J \in \mathcal{J}} D(P, J). \tag{3}$$

Algorithm 1 The gRRT Algorithm

1: **function** gRRT($\mathcal{M}$,$\boldsymbol{x}_{init}$,k_{max},h)
2: $k \leftarrow 0$; $\mathcal{T}$.init($\boldsymbol{x}_{init}$)
3: **repeat**
4: $\boldsymbol{x}_{goal} \leftarrow \mathcal{T}$.sample()
5: $\boldsymbol{x}_{near} \leftarrow \mathcal{T}$.neighbor($\boldsymbol{x}_{goal}$)
6: $\boldsymbol{u} \leftarrow \mathcal{T}$.findInput($\boldsymbol{x}_{near}, \boldsymbol{x}_{goal}$)
7: $\boldsymbol{x}_{new} \leftarrow$ sim($\boldsymbol{x}_{near}, \boldsymbol{u}, h$)
8: $\mathcal{T}$.addState($\boldsymbol{x}_{new}, \boldsymbol{u}$)
9: $k \leftarrow k + 1$
10: **until** $k \geq k_{max}$
11: **end function**

Algorithm 2 The Rrt-Rex Algorithm

1: **function** Rrt-Rex($\mathcal{M}$,$\boldsymbol{x}_{init}$,φ,k_{max}, h)
2: $k \leftarrow 0$; $\mathcal{T}$.init($\boldsymbol{x}_{init}$)
3: **repeat**
4: $\boldsymbol{x}_{goal} \leftarrow \mathcal{T}$.sample()
5: $\boldsymbol{x}_{near} \leftarrow \mathcal{T}$.neighbor($\boldsymbol{x}_{goal}$)
6: $\boldsymbol{u} \leftarrow \mathcal{T}$.findInput($\boldsymbol{x}_{near}, \boldsymbol{x}_{goal}$)
7: $\boldsymbol{x}_{new} \leftarrow$ sim($\mathcal{M}$, $\boldsymbol{x}_{near}$, $\boldsymbol{u}$, h)
8: $traj \leftarrow \mathcal{T}$.getTrajectory($\boldsymbol{x}_{new}$)
9: $v \leftarrow \rho^{est}(traj, \varphi)$
10: $\mathcal{T}$.addState($\boldsymbol{x}_{new}$,$\boldsymbol{u}$,v)
11: $k \leftarrow k + 1$
12: **until** $(v < 0)$ or $(k \geq k_{max})$
13: **end function**

The star discrepancy of a point set P with respect to a box $\mathcal{B}$ satisfies $0 < D^*(P, \mathcal{B}) \leq 1$. We define the coverage of P as $\gamma(P, \mathcal{B}) = 1 - D^*(P, \mathcal{B})$. 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 distributed well over $\mathcal{B}$.

We use the star discrepancy to evaluate the coverage of a set of states. Since there is no efficient way to compute the star discrepancy, we approximate it with an upper and lower bound. The estimation is based on a finite box partition, $\mathcal{C}$, of the box $\mathcal{B}$ (see [5] for more detail). Below, we describe how this information is used to guide the exploration of the system behaviors to the elements of $\mathcal{C}$ that are not as well explored as other elements.

4 Combining Coverage and Robustness

In this section we show how to combine the robustness and coverage information to guide the test generation process. To guide the exploration towards the states violating the property, robustness information can be used to select a starting state with a low robustness value; however, this objective can result in poor performance, as the initial robustness values can lead the exploration to some local positive minima. The star discrepancy coverage information can be used to ameliorate this problem by steering the algorithm to other parts of the feasible space. Intuitively, the algorithm uses the robustness to bias towards critical behaviors and the coverage to explore freely in the search space.

Note that while the coverage is defined for a set of states, robustness is defined for a trace. Each state in the RRT tree stems from a unique trace from the initial state (at the root of the tree). This trace may be *incomplete*, that is, the trace is not long enough to determine the true robustness value; however, a "predictive" value can be estimated and used for the purpose of guiding the search. The estimation will be discussed in Section 4.2. When a trace is *complete*, the robustness value indicates whether it satisfies the STL property in question.

We now describe the sampling method used in the function sample. The search space is partitioned into a set $\mathcal{C}$ of rectangular regions called *cells*. Each cell

c is associated with a local star discrepancy value, denoted by $D(c)$, determined by the geometric distribution of the current set of states with respect to the cell (see the definition (2)). When a new state is added, the estimates for the affected cells are updated.

4.1 Guiding Strategies

Using the above information, the main steps of the goal state sampling process are as follows:

- Step 1: a goal cell $c_{goal} \in \mathcal{C}$ is selected, based on the coverage estimate;
- Step 2: a goal state $\boldsymbol{x}_{goal}$ is randomly selected from inside the chosen cell c_{goal}.

The goal cell selection in Step 1 is performed by defining a probability distribution for the cells in the partition. We create a discrete probability distribution by assigning a probability value to each c based on the estimated star discrepancy. Let $\mathcal{C}$ be the current set of cells. For each cell we define a weight:

$$w(c) = s(\rho(c))$$

where $s(\rho(c)) = \frac{1}{1+e^{-\rho(c)}}$ is the sigmoid function. A goal cell is then sampled according to the following probability distribution:

$$Prob[c_{goal} = c] = \frac{w(c)}{\sum_{c \in \mathcal{C}} w(c)}.$$

The computation of the neighbor is biased by the robustness. Indeed, we can choose m nearest neighbors of the $\boldsymbol{x}_{goal}$ and then pick the one with the lowest robustness estimate. So the robustness plays a role in the selection of the neighbor which determines the initial state for the next iteration.

4.2 Defining Branch Robustness

The syntax and semantics of STL introduced in Section 2.2 are defined for traces with a possibly unbounded time horizon. In practice, one usually assumes that simulation time is long enough to estimate the satisfaction of a property. However, this assumption cannot be made in our case since we construct a tree of incomplete trajectories and estimate the satisfaction of a property for each node of the tree corresponding to incomplete trajectories. In this section, we introduce an *interval* quantitative semantics which is well-defined even for partial traces.

Consider the simple property $\varphi \equiv \Box_{[10,20]}(x > 1)$ and assume that we simulated the system until some time T_{sim}. There can be three situations: $T_{\text{sim}} < 10$, $10 \leq T_{\text{sim}} < 20$ or $20 \leq T_{\text{sim}}$. In the first case, the trace does not contain any information relevant to the property, hence its satisfaction cannot be determined in any way. In the last case, the trace contains all the information needed for computing the Boolean and quantitative satisfaction. In the middle case, we cannot know what happens between T_{sim} and $20s$, but the values

of x between $10s$ and T_{sim} provide some information. For instance, we have $\rho(\varphi, x, t) = \min_{t\in[10,20]}(x(t) - 1) \leq \min_{t\in[10,T_{\text{sim}}]}(x(t) - 1)$, hence the minimum of $x(t) - 1$ in $[10, T_{\text{sim}}]$ is an upper bound of $\rho(\varphi, x, t)$. If this upper bound is negative, we know that the property is falsified, even though we do not know the actual robust satisfaction value. Similarly, if $\varphi \equiv \Diamond_{[10,20]}(x > 1)$, then we can easily deduce that the *maximum* of $x(t) - 1$ over $[10, T_{\text{sim}}]$ is a *lower bound* of $\rho(\varphi, x, t)$. If it is positive, we know already that no matter what happens between T_{sim} and 20, the property will be satisfied. The computation of upper and lower bounds for ρ can be done automatically by induction on formulas. Let $\boldsymbol{x}$ be a signal defined on $\mathbb{R}^+$, $\boldsymbol{x}_{|T_{\text{sim}}}$ its restriction to $[0, T_{\text{sim}}]$ for some $T_{\text{sim}} > 0$ and φ an STL formula.

Formally, we define the function $\bar{\rho}$ for $\boldsymbol{x}_{|T_{\text{sim}}}$ and an arbitrary STL formula as:

$$\bar{\rho}(\mu, \boldsymbol{x}_{|T_{\text{sim}}}, t) = f(\boldsymbol{x}(t)) \text{ if } t \leq T_{\text{sim}},\ +\infty \text{ otherwise.} \tag{4}$$

$$\bar{\rho}(\neg\varphi, \boldsymbol{x}_{|T_{\text{sim}}}, t) = -\underline{\rho}(\varphi, \boldsymbol{x}_{|T_{\text{sim}}}, t) \tag{5}$$

$$\bar{\rho}(\varphi_1 \wedge \varphi_2, \boldsymbol{x}_{|T_{\text{sim}}}, t) = \min(\bar{\rho}(\varphi_1, \boldsymbol{x}_{|T_{\text{sim}}}, t), \bar{\rho}(\varphi_2, \boldsymbol{x}_{|T_{\text{sim}}}, t)) \tag{6}$$

$$\bar{\rho}(\varphi_1 \mathcal{U}_{[a,b]} \varphi_2, \boldsymbol{x}_{|T_{\text{sim}}}, t) = \sup_{t'\in[t+a,t+b]} \big(\min(\bar{\rho}(\varphi_2, \boldsymbol{x}_{|T_{\text{sim}}}, t'), \inf_{t''\in[t,t')} \bar{\rho}(\varphi_1, \boldsymbol{x}_{|T_{\text{sim}}}, t'')\big). \tag{7}$$

Then $\underline{\rho}$ satisfies the same self inductive rules except that $+\infty$ is replaced by $-\infty$ in (4) and $\bar{\rho}$ and $\underline{\rho}$ are switched in (5). The following lemma is true by construction:

Lemma 1. *Define the time horizon $T(\varphi)$ of φ inductively by $T(\mu) = 0$, $T(\neg\varphi) = T(\varphi)$, $T(\varphi_1, \varphi_2) = \max(T(\varphi_1), T(\varphi_2))$ and $T(\varphi_1 \mathcal{U}_{[a,b]} \varphi_2) = \max(T(\varphi_1), T(\varphi_2)) + b$. Then*

$$T_{sim} > 0 \Rightarrow \bar{\rho}(\varphi, \boldsymbol{x}_{|T_{sim}}, 0) \geq \rho(\varphi, \boldsymbol{x}, 0) \geq \underline{\rho}(\varphi, \boldsymbol{x}_{|T_{sim}}, 0) \tag{8}$$

$$T_{sim} \geq T(\varphi) \Rightarrow \bar{\rho}(\varphi, \boldsymbol{x}_{|T_{sim}}, 0) = \underline{\rho}(\varphi, \boldsymbol{x}_{|T_{sim}}, 0) = \rho(\varphi, \boldsymbol{x}, 0) \tag{9}$$

From (8), it follows that if $\bar{\rho}$ is negative for $\boldsymbol{x}_{|T_{\text{sim}}}$, we can conclude that $\boldsymbol{x}$ falsifies φ. Similarly, a non-negative $\underline{\rho}$ for $\boldsymbol{x}_{|T_{\text{sim}}}$ means that $\boldsymbol{x}$ satisfies φ, which means we can stop expanding the tree from $\boldsymbol{x}_{|T_{\text{sim}}}$. From (9), it follows that if a trace issued from a node in the exploration tree is longer than $T(\varphi)$, its satisfaction status is fully determined. If it falsifies φ, then we are done, otherwise, the branch issued from this node does not need to be explored any further.

In practice, it is often the case that $\bar{\rho}$ and/or $\underline{\rho}$ do not provide any useful information (e.g., if they are equal to $+\infty$ or $-\infty$). For guiding purposes, we use an intermediate estimate obtained by computing ρ on the unbounded signal defined by constant extrapolation of the last value. Since such extrapolation does not provide any guarantee on the Boolean satisfaction of φ, we use ρ only for guiding, $\bar{\rho}$ to determine falsification and $\underline{\rho}$ to determine definite satisfaction.

4.3 Algorithm for Testing a Simulink Model against an STL formula

We describe an implementation of our approach called RRT-REX (RRT Robustness-guided EXplorer), which uses Simulink as a simulation engine. The pseudocode in Algorithm 2 summarizes the new algorithm. The algorithm receives as input a Simulink model $\mathcal{M}$, an STL formula φ, and the maximum number of iterations $k_{max} \in \mathbb{N}$. The algorithm iterates until a negative robustness estimate is found, i.e., a trace that does not satisfy the property is discovered, or the maximum number of iterations is reached. In this algorithm we use a fixed time step h for simplicity of presentation but it is possible to use a variable time step for more accurate simulation performance.

In RRT-REX, as the sim function, we use the numerical differential equation solvers within Simulink to produce an updated state $\boldsymbol{x}_{new}$. This is performed by first resetting the solver state to $\boldsymbol{x}_{near}$ and then setting the input to $\boldsymbol{u}$; the solver then provides a solution over h seconds. For the Simulink tool, this is computationally costly, due to a time-consuming model-compilation step that is performed before each simulation is computed. This is an inefficiency in the solver implementation, but it does not reflect a fundamental drawback of our technique.

Our implementation of the function neighbor uses the Approximate Nearest Neighbors (ANN) library [21]. The ANN library can quickly identify members of the elements of $\mathcal{T}$ in close proximity to a given point $\boldsymbol{x}_{near}$. Also, we use a part of *HTG* (Hybrid Test Generation Tool [5]) to build and maintain the $\mathcal{T}$ structure, as well as to implement the function sample. The Breach [6] tool is used to implement the ρ^{est} function. The function addState primarily adds a new point to the RRT $\mathcal{T}$; it additionally updates the partitions in the tree with the star- discrepancy information as well as the robustness estimate. These quantities are then subsequently used as part of the functions sample and neighbor. All of the $\mathcal{T}$-related functions use the implied partition $\mathcal{C}$. We note that this partition can be dynamically refined for more accurate star discrepancy estimation, faster neighbor computation, and better cell robustness estimation.

5 Case Studies

This section discusses case studies of the application of our technique. We consider four case studies: an academic model of a system that measures how much longer a signal remains positive than it remains negative, a mode-selection example that uses a number of Boolean connectives to determine an operating regime and accordingly sets a reference value, an abstract model for the closed-loop fuel-control in an internal combustion gasoline engine, and a closed-loop model of the airpath in a diesel engine.

For each model, we try to falsify a given STL requirement and compare the performance of RRT-REX with a state-of-the-art falsification tool: S-TaLiRo. S-TaLiRo uses robust satisfaction values of an MTL requirement computed on complete simulation traces in iteration i to guide the choice of inputs and initial conditions of the model in iteration $i+1$. This guidance is provided by stochastic optimization algorithms such as simulated annealing, cross-entropy, and ant-colony optimization. Recall that S-TaLiRo chooses a fixed parameterization of

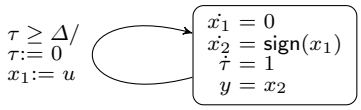

Fig. 1: Sampled polarity integrator system

the input signal-space and at the beginning of each simulation picks a set of valuations for the parameters to define the concrete input signals.

The results of the study are summarized in Table 1. Note that for each item in the table, an average over 10 cases is reported. For the Falsified columns (denoted *Fals.*), all of the 10 cases for each item were either falsified or not, except for the footnoted item.

5.1 Sampled polarity integrator system (SPI)

A sampled polarity integrator is an academic model that highlights the advantages of using intermediate robustness values as a search heuristic for guiding the RRT search. As shown in Fig. 1, every Δ seconds, the system samples the input $u(t)$, and stores the value in state x_1. We assume that u is a signal that ranges over $[-v, v]$ for some real number v. The state x_2 evolves with rate -1 if $x_1(t) < 0$, and $+1$ if $x_1(t) > 0$. Finally, the output of the system (denoted y) is the state x_2. We pick $\Delta = 1$ for our experiment, and initial conditions $x_1(0) = 0$, $x_2(0) = 0$, $\tau(0) = 0$.

We introduce the following artificial safety requirement, where n, k are fixed positive integers, $k \in [-v, v]$, and $n > k$: $\Box_{[0,n\Delta]}(y < k)$. We assume the following fixed parameterization of the input signals: $u(t) = u_i$ if $(i-1)\Delta \leq t < i\Delta$ for $1 < i \leq n$, and $u_i \in [-v, v]$. Each u_i is called a control point. We observe that in order to falsify the requirement, the required input u would have to be positive at greater than $\lceil \frac{n+k}{2} \rceil$ control points. For a tool based on Markov-chain Monte-Carlo based random testing techniques, the probability of at least $\lceil \frac{n+k}{2} \rceil$ of n uniform-randomly chosen numbers in the interval $[-v, v]$ being positive is tiny (when k is comparable[4] to n). For example, for $k = 30$, and $n = 50$, the probability is about 10^{-5}.

One of the RRT-REX heuristics is to pick multiple goal points and select one based on lower branch robustness. The next step is then to perform a local optimization in order to choose an input that drives the RRT towards the goal point. In this variant, in each step, the probability that RRT-REX selects a positive input effectively exceeds that in the previous step (starting with a probability of 0.5 in the first step). Thus, as a result of the robustness-guided goal-point selection and local optimization, the probability that RRT-REX discovers a sequence containing greater than $\frac{n+k}{2}$ positive values is much better than that for MCMC techniques. This is reflected in the experimental results shown in Table 1. For the

[4] One could use the (conservative) Chernoff bounds on the sum of binomial coefficients, which yields the following bound on probability: $\exp\left(\frac{-k^2}{2n}\right)$.

results in the table, we use a time step of $h = 1.0$ seconds for the RRT-REX algorithm over each of the various time horizons (50,200,500). For the associated S-TaLiRo tests, the time horizon is equivalent to the dimension of the search space explored by the falsification engine.

5.2 Mode-specific Reference Selection Model (MRS)

Next, we consider a model that selects an operating regime based on the input signals, and then sets a reference value based on the operating regime. We present a simplified version of the mode-selection logic occuring in an actual model for proprietary reasons. The simplified model takes as input 9 signals $u_1, \ldots, u_9$, where the range for each input in $u_1, \ldots, u_8$, is $[0, 100]$, and the range for u_9 is $[-5, 5]$. The model has three outputs, and we consider it a violation if any one of the outputs is less than a specified bound for that output. Thus, we consider three requirements of the form:

$$\Box_{[\tau, T_{hoz}]}(y_i > -\rho_i). \tag{10}$$

In the above, we assume $\tau = 5$ seconds, $T_h = 10$ seconds, and $\rho_1 = 8$, $\rho_2 = 100$ and $\rho_3 = 20$. Analyzing the structure of the models, we observe that for each requirement, selection of the mode corresponding to the following condition leads to the failing case:

$$\bigwedge_{i \in [1..4]} ((u_{2i}(t) > 90) \wedge (u_{2i-1}(t) < 10)). \tag{11}$$

We thus know that is is possible to falsify each of the requirements by setting the appropriate input values, so as to enable the particular failing mode. However, such a configuration is difficult to find since the probability of finding a time t at which (11) is *true* is 10^{-8} (8 inputs, and for each input there is a probability of $\frac{1}{10}$ for choosing the right value for that input).

We select a time step of $h = 2.5$ seconds for the RRT-REX algorithm over the 10 second time horizon. For the associated S-TaLiRo tests, this corresponds to 4 decision variables per each of the 9 inputs (i.e., a 36 dimension search space). As expected, neither RRT-REX nor S-TaLiRo were able to falsify any of the requirements with their default configurations.

5.3 Fuel Control System (AFC)

Next we consider a closed-loop model of an automotive powertrain control (PTC) subsystem. The system consists of two separate parts: (1) a plant model that describes some key physical processes in an internal combustion engine, and (2) a controller model that represents the embedded software used to regulate the ratio of air-to-fuel (A/F) within the engine. A detailed description of this model can be found in [10], section 3.1.; here, we only focus on features relevant to this case study. The model has 7 continuous state variables (5 for the plant,

Model	Spec.	RRT			S-TaLiRo		
		Fals.	Time (sec)	Iter.	Fals.	Time	Iter.×Run
SPI	$\square_{[0,50]}(y < 20)$	yes	60.81	199.10	yes	104.95	1176.60
	$\square_{[0,200]}(y < 50)$	yes	432.07	558.00	no	1103.46	2000×10
	$\square_{[0,500]}(y < 150)$	yes	2251.01	2014.30	no	9771.45	5000×10
MRS	$\square_{[5,10]}(y_i > -8)$	no	401.70	5000	no	813.36	1000×10
	$\square_{[5,10]}(y_i > -100)$	no	372.71	5000	no	997.80	1000×10
	$\square_{[5,10]}(y_i > -20)$	no	379.73	5000	no	905.93	1000×10
AFC	$\square_{[5,50]}(\lvert\lambda\rvert \leq 0.05)$	yes	1162,13	185.33	no	5737.43	5000×10
	$\square_{[5,50]}(\lvert\lambda\rvert \leq 0.02)$	yes	1658.48	290.67	no	6755.09	5000×10
	$\Diamond_{[50,50]}(\lambda_{rms} \leq 0.05)$	no	6359.22	1000	yes	928.97	10.00
DAP	$\square_{[1.1,50]}(\mathbf{x} < 2)$	yes	14.23	2.3	yes	35.01	1
	$\square_{[1.1,50]}(\mathbf{x} < 4)$	yes	155.55	15.3	yes	46.30	1.4
	$\square_{[1.1,50]}(\mathbf{x} < 6)$	yes/no[5]	416.38	40.0	yes	204.36	7.6

Table 1: Results of comparison between RRT approach and S-TaLiRo tool[6].

and 2 for the controller), a *delay* function[7], 4 discrete modes of operation in the controller, and two exogenous inputs. In this case study we focus on: (1) the plant state λ denoting the measured, normalized A/F ratio, (2) a fixed engine speed $\omega \in [900, 1100]$, which is treated as a parameter, and (3) the exogenous user input θ representing the throttle angle command. We assume that $\theta(t_i) \in [0, 61.2]$, and is permitted to change value at a rate of $h = 1.0$ seconds.

We provide three requirements for the closed-loop model. The first two are specified in (12). Here, φ_1 specifies the bounds on the worst-case overshoot or undershoot on λ, while φ_2 characterizes the settling time on λ (time it requires λ to return to a small neighborhood of the reference value λ_{ref} after a perturbation). In (13), we introduce the signal λ_{rms} to help measure the RMS error between λ and λ_{ref}. In the definition of λ_{rms}, we exclude model behaviors for an initial τ_I seconds of time in order to discard transients in the startup mode and transients arising from a mode switch to the normal mode of operation. Note that $\mathsf{u}(t)$ denotes the Heaviside step function. The third requirement (shown in (14)) specifies the bounds on the RMS error incurred in the A/F ratio state in the normal model of operation.

[7] A continuous-time delay is a function described by the input/output relation $y(t) = u(t - \Delta)$ for some $\Delta \in \mathbb{R}^{\geq 0}$. Systems with delays pose a significant challenge to techniques such as RRT-REX, as they correspond to systems with infinite state variables. We assume that we can simulate such systems, but do not assume that we can measure the states associated with the delays.

$$\varphi_1 \equiv \Box_{[0,T_h]}(|\lambda| \leq 0.05) \qquad \varphi_2 \equiv \Box_{[\tau,T_h]}(|\lambda| \leq 0.02) \tag{12}$$

$$\lambda_{rms}(t) = \sqrt{\frac{1}{t-\tau_I}\int_0^t (\lambda(\tau) - \lambda_{ref})^2 \cdot \mathsf{u}(t-\tau_I)d\tau} \tag{13}$$

$$\varphi_3 \equiv \Diamond_{[T_h,T_h]}(\lambda_{rms} \leq 0.05) \tag{14}$$

In the above formulas, we select parameter values $\tau = 11$ seconds and $T_h = 50$ seconds, and we use a time step of $h = 1.0$ seconds for the RRT-REX algorithm. In this study RRT-REX and S-TaLiRo present mixed results (see Table 1). RRT-REX was able to falsify both the specifications 1) and 2) but not 3); however, S-TaLiRo was not able to falsify the properties 1) nor 2), but found a counterexample for the specification 3). The results support the theory that constructing input traces incrementally, as is the case for RRT-REX, can offer performance benefits over other approaches for some cases; however, such incremental approach may be a drawback in some instances. For instance, specification 3) imposes a constraint on a precise time instant rather then on a time interval. This does not provide RRT-REX with incremental quantitative information that will lead to the failing trace, which makes it difficult for RRT-REX to identify a falsifying instance.

5.4 Diesel Air-Path Model (DAP)

In this section, we consider an industrial closed-loop Simulink model of a prototype airpath controller for a diesel engine. The model contains more than $3,000$ blocks. It has a detailed plant model and a controller with more than 20 lookup tables and function blocks containing customized Matlab functions and legacy code. Moreover, more than half of these lookup tables are high dimensional (greater than one dimension). One of the challenges is, due to the high model complexity, obtaining simulation traces for this model is computationally expensive.

For this case study, we choose a safety property specifying upper bounds on the overshoot of a particular signal. This is represented by the following STL formula: $\Box_{[1.1,50]}(\mathbf{x} < c)$. We select a time step of $h = 5.0$ seconds, and compare the results of running RRT-REX and S-TaLiRo on this model, using three different values for c (2, 4, and 6). These values for c are not realistic, in the sense that the actual worst-case bounds on the system behavior are much larger. We select these smaller bounds due to the significant computational costs required to find falsifying traces using either RRT-REX or S-TaLiRo. The c values we select are adequate to study the performance of the RRT-REX approach versus S-TaLiRo as a function of the relative difficulty of the falsification task.

Table 1 indicates the results of the experiments. For the $c = 2$ case, RRT-REX is able to identify falsifying traces early in the simulation runs, and so explores simulation runs that are significantly shorter in length than the 50 second time horizon. By contrast, S-TaLiRo has to complete at least one simulation trace over the 50 second time horizon to determine that a trace falsifies the property. Since this requirement is easily falsified, RRT-REX is able to identify falsifying

traces by exploring a small number of short simulation traces. The result is that RRT-REX performs better than S-TaLiRo for this case.

For the $c = 4$ and $c = 6$ cases, S-TaLiRo performs better than RRT-REX. The reason for this is that the requirements are such that the robustness values are sufficient to guide the global optimizer within S-TaLiRo directly to falsifying traces. This is in contrast to the RRT-REX approach, which introduces a significant amount of randomness into the search. This additional randomness incurs computational costs (due to the increased number of required simulations) that are larger than those incurred by S-TaLiRo.

6 Conclusions

In this paper we proposed a testing-based technique to find bugs in CPS systems. Given a specification expressed in terms of an STL formula, the search for an input sequence that causes the system to exhibit behaviors that violate the formula (falsifying behaviors) is guided using a combination of two criterion: the coverage of the system state space and the satisfaction robustness value of the specification. The coverage indicates "how well" we are exploring the state space of the system, while the robustness reflects "how much" the specification is satisfied, giving us the numerical intuition of how far we are from falsifying the formula. We incrementally build a simulation tree to best cover the state space, favoring those branches that correspond to low robustness value (with respect to the specification).

We implemented our framework in a prototype tool called RRT-REX, we applied it to both academic and industrial models, showing its applicability to practical systems, and we compared it with the S-TaLiRo tool.

Our experiments reveal that the relative performance of RRT-REX and S-TaLiRo depends on the nature of the model and the associated specification. We demonstrated that the incremental random RRT-REX search performs well on system with large input spaces and long input sequence; however, the RRT-REX approach is weak in the case of specifications defined of precise temporal instants. This drawback suggests directions for future developments. In subsequent work, we plan to study a dynamic exploration technique triggered by the formula itself, where the sampling space from which RRT-REX selects the exploration directions varies according to logical subformulas within the specification.

References

1. Annapureddy, Y., Liu, C., Fainekos, G.E., Sankaranarayanan, S.: S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems. In: TACAS. pp. 254–257 (2011)
2. Beck, J., Chen, W.: Irregularities of Distribution. Cambridge Studies in Social and Emotional Development, Cambridge University Press (1987)
3. Bhatia, A., Maly, M., Kavraki, E., Vardi, M.: Motion planning with complex goals. Robotics Automation Magazine, IEEE 18(3), 55–64 (2011)
4. Dang, T., Donzé, A., Maler, O., Shalev, N.: Sensitive state-space exploration. In: CDC. pp. 4049–4054 (2008)

5. Dang, T., Nahhal, T.: Coverage-guided test generation for continuous and hybrid systems. Formal Methods in System Design 34(2), 183–213 (2009)
6. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: CAV. pp. 167–170 (2010)
7. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: FORMATS. pp. 92–106 (2010)
8. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for stl. In: CAV. pp. 264–279 (2013)
9. Fainekos, G., Sankaranarayanan, S., Ueda, K., Yazarel, H.: Verification of automotive control applications using s-taliro. In: ACC (2012)
10. Jin, X., Deshmukh, J.V., Kapinski, J., Ueda, K., Butts, K.: Powertrain Control Verification Benchmark. In: HSCC (2014)
11. Jin, X., Donzé, A., Deshmukh, J., Seshia, S.: Mining requirements from closed-loop control models. In: HSCC (2013)
12. Karaman, S., Frazzoli, E.: Linear temporal logic vehicle routing with applications to multi-UAV mission planning. Int. J. of Robust and Nonlinear Control 21(12), 1372–1395 (2011)
13. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. Int. J. of Robotics Research 30(7), 846–894 (2011)
14. Kim, J., Esposito, J.M., Kumar, V.: An rrt-based algorithm for testing and validating multi-robot controllers. In: RSS. pp. 249–256 (2005)
15. Kloetzer, M., Belta, C.: A fully automated framework for control of linear systems from temporal logic specifications. IEEE Trans. Auto. Control 53(1), 287–297 (2008)
16. Kong, Z., Jones, A., Ayala, A.M., Gol, E.A., Belta, C.: Temporal logic inference for classification and prediction from data. In: HSCC (2014)
17. LaValle, S.M.: Planning Algorithms, chap. 5. Cambridge University Press, Cambridge, U.K. (2006), available at http://planning.cs.uiuc.edu/
18. Lavalle, S.M., Kuffner, J.J., Jr.: Rapidly-exploring random trees: Progress and prospects. In: Algorithmic and Computational Robotics: New Directions. pp. 293–308 (2000)
19. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: FORMATS/FTRTFT. pp. 152–166 (2004)
20. Mathworks, T.: Simulink design verifier, `http://www.mathworks.com/products/sldesignverifier/`
21. Mount, D.M., Arya, S.: Ann: A library for approximate nearest neighbor searching, `http://www.cs.umd.edu/~mount/ANN/`
22. Plaku, E., Kavraki, L., Vardi, M.: Hybrid systems: From verification to falsification by combining motion planning and discrete search. Formal Methods in System Design 34(2), 157–182 (2009)
23. Systems, R.: Model based testing and validation with reactis, reactive systems inc., `http://www.reactive-systems.com`
24. Yang, H., Hoxha, B., Fainekos, G.: Querying parametric temporal logic properties on embedded systems. In: Int. Conference on Testing Software and Systems. vol. 7641, pp. 136–151 (2012)

Appendix

Boolean and Quantitative Semantics of STL

Given a signal $\boldsymbol{x}$ and a time t, the Boolean semantics of any STL formula is given inductively by

$$\begin{array}{ll}
(\boldsymbol{x}, t) \models \mu & \text{iff } \boldsymbol{x} \text{ satisfies } \mu \text{ at time } t \\
(\boldsymbol{x}, t) \models \neg\varphi & \text{iff } (\boldsymbol{x}, t) \not\models \varphi \\
(\boldsymbol{x}, t) \models \varphi_1 \wedge \varphi_2 & \text{iff } (\boldsymbol{x}, t) \models \varphi_1 \text{ and } (\boldsymbol{x}, t) \models \varphi_2 \\
(\boldsymbol{x}, t) \models \varphi_1 \, \mathcal{U}_{[a,b]} \, \varphi_2 & \text{iff } \exists t' \in [t+a, t+b] \text{ s.t.} (\boldsymbol{x}, t') \models \varphi_2 \\
 & \quad \text{and } \forall t'' \in [t, t'), (\boldsymbol{x}, t') \models \varphi_1
\end{array}$$

Extending these semantics to other kinds of intervals (open, open-closed, and closed-open) is straightforward. We write $\boldsymbol{x} \models \varphi$ as a shorthand of $(\boldsymbol{x}, 0) \models \varphi$.

The quantitative semantics ρ is defined inductively as follows:

$$\begin{aligned}
\rho(\mu, \boldsymbol{x}, t) &= f(\boldsymbol{x}(t)) \\
\rho(\neg\varphi, \boldsymbol{x}, t) &= -\rho(\varphi, \boldsymbol{x}, t) \\
\rho(\varphi_1 \wedge \varphi_2, \boldsymbol{x}, t) &= \min(\rho(\varphi_1, \boldsymbol{x}, t), \rho(\varphi_2, \boldsymbol{x}, t)) \\
\rho(\varphi_1 \mathcal{U}_{[a,b]} \varphi_2, \boldsymbol{x}, t) &= \\
&\sup_{t' \in [t+a, t+b]} \big(\min(\rho(\varphi_2, \boldsymbol{x}, t'), \inf_{t'' \in [t, t')} \rho(\varphi_1, \boldsymbol{x}, t'') \big),
\end{aligned}$$

where we assume each predicate is of the form $\mu \equiv f(\boldsymbol{x}(t)) \geq 0$. Additionally, by combining the rules above with $\Diamond_I \varphi \triangleq \top \, \mathcal{U}_I \, \varphi$, we get

$$\rho(\Diamond_{[a,b]} \varphi, \boldsymbol{x}, t) = \sup_{t' \in [t+a, t+b]} \rho(\varphi, \boldsymbol{x}, t').$$

For $\Box$, we get a similar expression using inf instead of sup.