Testing Cyber-Physical Systems Using a Line-Search Falsification Method

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

07-05-2021 / 06-07-2021

CITATION

Ramezani, Zahra; Claessen, Koen; Smallbone, Nicholas; Fabian, Martin; Åkesson, Knut (2021): Testing Cyber-Physical Systems Using a Line-Search Falsification Method. TechRxiv. Preprint.
<https://doi.org/10.36227/techrxiv.14555826.v3>

DOI

[10.36227/techrxiv.14555826.v3](https://doi.org/10.36227/techrxiv.14555826.v3)

Testing Cyber-Physical Systems Using a Line-Search Falsification Method

Zahra Ramezani, Koen Claessen, Nicholas Smallbone, Martin Fabian, and Knut Åkesson

Abstract—Cyber-physical systems (CPSs) are complex and exhibit both continuous and discrete dynamics, hence it is difficult to guarantee that they satisfy given specifications, i.e., the properties that must be fulfilled by the system. Falsification of temporal logic properties is a testing approach that searches for counterexamples of a given specification that can be used to increase the confidence that a CPS does fulfill its specifications. Falsification can be done using random search methods or optimization methods, both of which have their own benefits and drawbacks. This paper introduces two methods that exploit randomness to different degrees: the optimization-free *Hybrid-Corner-Random* (HCR), and the direct-search method *Line-search Falsification* (LSF). HCR combines randomly chosen parameter values with parameter extreme values, which performs surprisingly well on benchmark evaluations. The gradient-free optimization-based LSF optimizes over line segments through a vector of inputs in the n -dimensional parameter space. The two methods are compared to the Nelder-Mead and SNOBFIT methods, using a well-known set of benchmark problems and LSF shows better performance than any of the evaluated methods.

Index Terms—Cyber-Physical Systems, Testing, Falsification, Simulation-based Optimization

I. INTRODUCTION

Cyber-physical systems (CPSs), [1], bridge the cyber-world of communications and computing to the physical world. CPSs are widely adopted in many areas like autonomous systems and smart grids. These systems exhibit both *continuous* and *discrete* dynamics. CPSs are often safety-critical systems, e.g., autonomous cars and medical devices. Two commonly used methods for assessing the correctness of CPSs, see [2], are *formal verification* and *testing*. The main problem is to decide if it is possible to construct *counterexamples*, i.e., inputs that make the system under test (SUT) violate the specifications.

Formal verification is used to prove or disprove the correctness of the system with respect to a formal specification. This is typically done using model-checking or deductive verification [3]. However, for many industrial systems, formal verification is not a viable approach because of three reasons: (i) typically there are no formal models available to do verification on; (ii) even if formal models were available, the time and space complexity of the verification algorithms make them intractable for large systems; (iii) even if formal models were available and the algorithmic complexity tractable, verification of systems exhibiting a combination of discrete and continuous dynamics is, in general, an undecidable problem [4]. Though

formal verification is both possible and practical for a certain type of systems that are limited in their behavior and/or size, for large systems comprising other systems that all have different behavior, it is not a tractable approach.

Testing is a less formal approach that evaluates if the SUT behaves correctly with respect to given inputs. In testing, it is often assumed that the system can be simulated, but it is not assumed that a mathematical model of the SUT is available. With testing, it is possible to prove the presence of counterexamples but not the absence. The main challenge with testing is to reduce the number of tests done so as to find *as many* counterexamples for a given specification as *early* as possible. Thus, there is a need for rigorous methods to cleverly search for inputs that break the specification.

Simulation-based falsification approaches [5] are a class of methods that can be used to test CPSs given only a black-box model of the SUT, but with formal specifications of the closed-loop behavior available. These specifications can, for example, be given in Metric Interval Temporal Logic (MITL) [6] or Signal Temporal Logic (STL) [7], but the specification can also be defined using high-level languages. [8] presents how STL specifications can be automatically generated from Simulink charts, hence supporting engineers in the specification of formal specifications without the need for being trained in temporal logic. The combination of black-box models for the SUT, together with white-box formal specifications, is a reasonable assumption in many industrial applications.

CPSs are typically developed using a model-based development paradigm. The models are often used for control-design or optimization, but can also be used in simulation-based falsification. In *simulation-based falsification using optimization*, the goal is to reduce the number of tests by using an optimization method to decide on the next set of input signals from the evaluation of the simulation results from the previous simulations. A key challenge here is to choose the optimization method and the kind of information that the optimization method should use to decide on the next set of parameters. Since the number of input signals might be large, and optimization methods are sensitive to the number of parameters, the allowed input signals are often parameterized using relatively few parameters. For example, for sinusoidal signals, the parameters can be the period and the amplitude.

The falsification process is performed using *quantitative semantics* [9], [10], of the specification. Quantitative semantics define an objective function that expresses a virtual distance to falsifying the specification. Given input and output traces from a simulation, the objective function is used to calculate an objective value that can guide the falsification process towards an input that is more likely to falsify the specification. In [11],

Z. Ramezani, M. Fabian and K. Åkesson are with the Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden. E-mail: {rzahra, fabian, knut}@chalmers.se

K. Claessen and N. Smallbone are with the Department of Computer Science and Engineering, Chalmers University and Technology, Gothenburg, Sweden. E-mail: {koen, nicsma}@chalmers.se

the concept of valued booleans (VBools) is introduced to allow expression of different quantitative semantics, e.g., *Max* and *Additive* semantics.

The choice of optimization method affects the efficiency of the falsification. By assuming that the SUT is given as a black-box model, the optimization approach are restricted to simulation optimization or black-box optimization (BBO) methods [12], [13], where no gradients can be analytically computed. BBO are generally divided into *direct search methods*, and *model-based methods*.

Direct-search methods are defined as the sequential examination of trial solutions generated by a certain strategy in [14]. A direct-search method uses and compares only objective function values at a collection of input parameters (points) to directly determine new candidate points for future exploration without any gradient approximation. Direct-search methods include the classic Nelder-Mead (NM) procedure [15].

Model-based search methods build a surrogate model of the objective function to guide the optimization process [13]. Bayesian optimization [16] is a global optimization method that has shown to be efficient for black-box optimization when the objective function is expensive to evaluate. Bayesian optimization is best suited for continuous domains and a relatively moderate number of input parameters. For CPSs, the system dynamics might switch between different discrete modes, each having different system dynamics and constraints, see for example [17]. This behavior of the system dynamics affects negatively the usefulness of a model-based approach for falsification of CPSs, and has to be explicitly handled for a Bayesian optimization approach to be successfully used for simulation-based falsification.

In [18], seventeen gradient-free optimization methods are reviewed and evaluated on a set of benchmark problems. This includes both direct-search methods, like NM and Mesh adaptive direct search algorithms [19], and model-based methods, like SNOBFIT [20]. The results show that no single optimization method consistently outperforms the others, but SNOBFIT was one of the best-evaluated methods.

Simulation-based falsification using different combinations of optimization algorithms and quantitative semantics were evaluated on a set of benchmark problems in [21]. The evaluation showed that quantitative semantics matter but also that the choice of optimization methods is very important. Furthermore, the study showed that NM and SNOBFIT have the best performance, with SNOBFIT as the overall top performer.

Three quantitative semantics, *Max*, *Additive*, and constant, were evaluated in [21]. The latter semantics works by assigning a constant positive value if the specification is fulfilled and a constant negative value if it is falsified. A constant objective value does not hold any information for the optimization methods about how far away from or close to falsifying the specification the current point is. In these experiments, it was observed that when SNOBFIT uses a constant semantics and thus does not get any hints in which direction to continue the search, it tends to explore new parameter values that are towards the extreme values of the allowed parameter ranges. We refer to these extreme values as the *corner points*. Surprisingly, SNOBFIT using a constant objective value performed better

than NM with *Max* and *Additive* semantics.

Random testing [22], is an optimization-free approach that, despite its simplicity, has proven to be useful for large systems. In [23], random testing is used to test space mission software and was shown to be a good complement to formal verification. In [24], feedback-directed random test generation had better fault detection and coverage than the structured test generations used in the study. Extensions of random testing include *adaptive random testing* [25], where empirical observations show that many faults occur in contiguous areas of the input domain. In [26], the risk of using coverage-based criteria are discussed where completely randomized test-suites identified 13.5% more faults than test suites that were generated to specifically achieve coverage. Random search has also been explored as a strategy for hyper-parameter optimization in machine learning approaches. [27] shows that randomized trials are more efficient than uniform trials for hyper-parameter optimization. One reason for this is that only a few parameters really matter for many data sets, but which ones differ from case to case. Together, these observations from practical software systems highlight the benefits of randomness in testing applications.

This paper focuses on CPSs with software components integrating with a physical environment typically described using differential equations. Evaluation results for falsification of CPSs in this paper show that randomized testing is an efficient strategy also for CPSs. Hence, this paper introduces two methods that explore to different degrees the search space randomly. The first one is *Hybrid-Corner-Random* (HCR), an optimization-free method that will serve as a baseline method to compare with. The second one is *Line-search Falsification* (LSF), a direct-search method. The strong dependence on randomness in these methods is motivated by the complex dynamic behavior of the SUT, which complicate model-based optimization approaches. However, as is shown in this paper for many falsification problems, it is also useful to consider corner points.

While HCR only relies on random and corner points, LSF combines random exploration with local search, by randomly generating lines in the n -dimensional parameter space for local search. The implementation of LSF is very small and straightforward, and evaluation on benchmark falsification problems indicates that LSF performs as well as or better than more complex optimization methods.

For simulation-based falsification of CPSs, several toolboxes are available. S-TaLiRo [28] and Breach [29] are both MATLAB/Simulink toolboxes that can be used for the falsification of large scale industrial systems. S-TaLiRo finds counterexamples using specifications expressed in MITL, and Breach performs falsification using specification in STL. In this paper, Breach is used to evaluate the proposed methods, but they are generic and could be implemented in most other tools.

The main contributions of this paper are:

- The introduction of LSF, an optimization-based method that exploits corners and random points, combined with local search. These techniques are well chosen for efficient falsification of CPSs.
- The introduction of the optimization-free method HCR, that combines using corner points of the parameter space

with random search. HCR is proposed as a baseline method to benchmark more sophisticated techniques to.

- Benchmarking of the optimization-free methods, Corners, Random, and HCR with the optimization-based methods, LSF, NM, SNOBFIT. The benchmark problems are published CPS falsification problems [30], [31].

This paper is organized as follows: STL and falsification of temporal logic are introduced in Section II. Section III proposes the suggested HCR method. Section IV introduces the LSF method. Section V introduces the benchmark problems evaluated in this paper. Section VI evaluates the performance of the suggested methods on the chosen benchmark problems. Finally, Section VII summarizes the contributions.

II. SIGNAL TEMPORAL LOGIC FOR FALSIFICATION

Falsification of temporal specifications needs a *quantitative semantics*, that defines an objective function to measure the distance to the specification being falsified. In optimization-free methods, the objective function is used only to evaluate if the specification is falsified or not. In optimization-based methods, though, the objective function is used to guide the falsification process by choosing the next set of input values such that they are more likely to falsify the specification.

Temporal specifications are typically expressed in linear temporal logic (LTL) [32]. However, LTL cannot reason about time intervals, something that is necessary for many specifications of CPSs. For this, MITL [6] and STL [7] have been introduced. In our work, STL specifications are used, but the proposed methods work with any quantitative semantics, though the performance of the falsification process might depend on the particular quantitative semantics. Thus, we evaluate the proposed methods using two different quantitative semantics defined for STL, *Max*, and *Additive*. Both these can be expressed in terms of VBools [11].

A. Signal Temporal Logic

$$\varphi ::= \mu \mid \neg\mu \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \Box_{[a,b]}\psi \mid \Diamond_{[a,b]}\psi$$

where the predicate μ is $\mu \equiv \mu(x^s) > 0$ and x^s is a signal; φ and ψ are STL formulas; $\Box_{[a,b]}$ denotes the *globally* operator between times a and b (with $a < b$); $\Diamond_{[a,b]}$ denotes the *finally* operator between a and b .

The satisfaction of the formula φ for the discrete signal x^s , consisting of both inputs and outputs to the SUT, at the discrete-time instant k is defined as:

$$\begin{aligned} (x^s, k) \models \mu & \Leftrightarrow \mu(x^s[k]) > 0 \\ (x^s, k) \models \neg\mu & \Leftrightarrow \neg((x^s, k) \models \mu) \\ (x^s, k) \models \varphi \wedge \psi & \Leftrightarrow (x^s, k) \models \varphi \wedge (x^s, k) \models \psi \\ (x^s, k) \models \varphi \vee \psi & \Leftrightarrow (x^s, k) \models \varphi \vee (x^s, k) \models \psi \\ (x^s, k) \models \Box_{[a,b]}\varphi & \Leftrightarrow \forall k' \in [k+a, k+b], (x^s, k') \models \varphi \\ (x^s, k) \models \Diamond_{[a,b]}\varphi & \Leftrightarrow \exists k' \in [k+a, k+b], (x^s, k') \models \varphi \end{aligned}$$

Instead of only checking the boolean satisfaction of an STL formula, the notion of quantitative value, i.e., an objective value, will be defined to measure how far away a specification

is from being falsified. VBool [11] $\langle v, y \rangle$ is a combination of a Boolean value v (true $\top$, or false $\perp$) together with a real number y that is a measure of how true or false the VBool is. This value will be used as a measure of how convincingly a test passed, or how severely it failed, respectively. In the original VBool definition, y is defined to always be non-negative.

B. Quantitative Semantics

Using VBools, we define two quantitative semantics: *Max*, which is essentially the same as the standard STL quantitative semantics, and *Additive*. For these semantics we define the respective *and*, *or*, *always*, and *eventually* operators.

For conjunction, the semantics differ only in the two cases where the truth values are the same:

	<i>Max</i>	<i>Additive</i>
$\langle \top, y \rangle \wedge \langle \top, s \rangle =$	$\langle \top, \min(y, s) \rangle$	$\langle \top, \frac{1}{\frac{1}{y} + \frac{1}{s}} \rangle$
$\langle \top, y \rangle \wedge \langle \perp, s \rangle =$	$\langle \perp, s \rangle$	
$\langle \perp, y \rangle \wedge \langle \top, s \rangle =$	$\langle \perp, y \rangle$	
$\langle \perp, y \rangle \wedge \langle \perp, s \rangle =$	$\langle \perp, \max(y, s) \rangle$	$\langle \perp, y + s \rangle$

Using the de Morgan laws, the *or* operator can be defined in terms of *and*, as: $\langle v_y, y \rangle \vee \langle v_s, s \rangle = \neg_v \langle \neg_v \langle v_y, y \rangle \wedge \neg_v \langle v_s, s \rangle \rangle$, where VBool negation is defined as $\neg_v \langle v, y \rangle = \langle \neg v, -y \rangle$.

For the *Max* semantics, the *always* operator over an interval $[a, b]$ is straightforwardly defined in terms of *and* operator, as

$$\Box_{[a,b]}\varphi = \bigwedge_{k=a}^b \varphi[k],$$

where φ is a finite sequence of VBools defined for all the discrete time instants in $[a, b]$.

For the *Additive*, though, *always* is defined as:

$$\Box_{[a,b]}\varphi = \bigwedge_{k=a}^b \varphi[k] \# \delta t,$$

where δt is the simulation step size that makes the quantitative value independent of the simulation time, and $\#$ is defined as

$$\begin{aligned} \langle \perp, y \rangle \# \delta t &= \langle \perp, y \cdot \delta t \rangle, \\ \langle \top, y \rangle \# \delta t &= \langle \top, y / \delta t \rangle. \end{aligned}$$

Furthermore, the *eventually* operator is for both semantics defined over an interval $[a, b]$ in terms of *always*, as: $\Diamond_{[a,b]}\varphi = \neg \langle \Box_{[a,b]} \langle \neg_v \varphi \rangle \rangle$.

C. Falsification

The main falsification procedure is shown in Fig. 1 for the optimization-based falsification approach. The possible input signals are parameterized with an n -dimensional vector, x , where each element is allowed to be within a defined range. A discrete sequence of inputs is generated by the *Generator* module, that given the parameters x , generates an input trace describing a sequence of input vectors, $x_i^s[k]$, for the input signals. Each element in the sequence is indexed by k , where k ranges from the start of the simulation to the end of the simulation, with x_i^s we denote the full sequence. Note, that the dimension of the input vector x_i^s is often lower than

Fig. 1: A flowchart of optimization-based falsification. The input and output signals are denoted by x_i^s and x_o^s , respectively.

the dimension of x , since multiple parameters are used to parameterize each input signal. The SUT is simulated with the x_i^s as inputs, and its corresponding output x_o^s is generated by the *Simulator*. In the evaluation of specifications both the input and output signals are used, with x^s we denote the combination of x_i^s and x_o^s .

The input x_i^s and output vectors x_o^s and the specification φ are used together with the objective function $f^\varphi(x^s)$ to evaluate whether the specification φ is falsified or not.

By using a quantitative semantics the specification can be determined to be satisfied or not. If it is satisfied, the semantics will also give a value of how convincingly the test passed. If the specification is not fulfilled, the current set of input and output traces is a counterexample, and thus the falsification process can terminate. However, in the sequel, we will consider the situation where the specification is fulfilled and the aim of the falsification is to lower the value in order to try to find counterexamples.

Let φ be the specification possibly containing temporal operators. Given φ , the traces x^s , and the choice of quantitative semantics, the evaluation of x^s will return a VBool $\langle v, y \rangle$, where v denotes if φ is satisfied or not, and y is the measure.

Define $f^\varphi(x^s)$ such that

$$f^\varphi(x^s) = \begin{cases} y & \text{if } v = \top \\ -y & \text{if } v = \perp \end{cases}$$

A negative objective function value, $f^\varphi(x^s) < 0$, means that the specification is falsified; thus, the falsification procedure may stop. A non-negative objective function value, $f^\varphi(x^s) \geq 0$, means that the specification is not falsified; it leads to new parameters being sampled, and the process is repeated. The parameter optimizer generates new parameter values within given ranges to find lower objective function values.

The optimization problem is formulated as follows. Let f^φ be the objective function defined above. The lower and upper bounds on the n -dimensional parameter vector x are defined as $l = (l_1, l_2, \dots, l_n)^T$ and $u = (u_1, u_2, \dots, u_n)^T$, respectively. The falsification problem is to find input parameters x^* such that

$$x^* = \arg \min_{l \leq x \leq u} f^\varphi(x^s). \quad (1)$$

With slight abuse of notation we will write $f(x)$ as shorthand for $f^\varphi(x^s)$, where x_i^s is defined by the *Generator* by using the parameters x , and x_o^s is the output of the *Simulator* with x_i^s as the input trace.

It is important to note that for falsification of CPSs, the objective function value is defined by the input and output traces of the CPS. The output traces are defined by the inputs and the system dynamics. In the simulation-based falsification, the objective function is fully known, but the system dynamics is given as a black-box model.

The optimization methods that are compared in this paper are NM and SNOBFIT, both briefly described below.

Nelder-Mead (NM) [15] is a direct-search method that starts with a simplex set of $n + 1$ points where here n is the number of dimensions of the optimization problem. In each iteration, the points are sorted from the lowest to highest objective function value. NM attempts to replace the point with the highest objective function value with a new point obtained by reflection, expansion or contraction. If all of these fail, the entire simplex shrinks towards the point with the lowest objective function value, and n new points are generated, and the above process is continued.

SNOBFIT is a model-based method that works by building surrogate models around each evaluated point. The optimization proceeds by processing of parameter values and the corresponding function values and recommending a new set of parameters values to evaluate. SNOBFIT that typically has a fast local search and contains parameters that controls the balance between the local and global search of the optimization. The reader is referred to [20] for a detailed description of the inner workings of SNOBFIT.

III. HYBRID CORNER-RANDOM METHOD (HCR)

Falsification can be done with or without using an optimization method. We start by investigating how efficient optimization-free methods are for falsification of benchmark problems. In this section, we introduce the HCR falsification method that explores corner points, i.e., extreme values within the allowed parameter ranges, in combination with evaluating random parameter values also within the allowed parameter ranges.

To clarify the meaning of a corner point, assume a system with n input parameters $x_1, \dots, x_n$, where each parameter has a lower and upper bound. There are 2^n corner points, where a corner point only contains values that are at the lower or upper bound for each parameter.

The HCR falsification method is presented in Algorithm 1.

In Algorithm 1, the quantitative semantics used does not matter. The algorithm just evaluates whether the specification is falsified or not at the currently selected point.

The output of Algorithm 1 is the tuple $\langle \text{Falsified/Not Falsified}, x \rangle$, where the first element describes if the specification was falsified or not, and the second element x is an n -dimensional parameter point with the following properties.

- 1) If the specification is falsified, $f(x) < 0$.
- 2) If the maximum number of simulations is reached and the specification is not falsified, x is the last evaluated point.

Algorithm 1 Hybrid Corner-Random Method for Falsification

```

1: curr_simulation = 0;
2: Pick a corner point,  $x$ .
3: while curr_simulation < max_simulations do
4:   Simulate system with  $x$  as input.
5:   curr_simulation++;
6:   Evaluate spec at  $x$ .
7:   if the spec is falsified then
8:     Return (Falsified,  $x$ ).
9:   else
10:    if curr_simulation mod 2 == 0 or corners exhausted then
11:      Pick a random point  $x$ .
12:    else
13:      Pick a corner point  $x$ , not previously selected.
14:    end if
15:  end if
16: end while
17: Return (Not Falsified, the last evaluated  $x$ ).

```

A. Description

Algorithm 1 starts with a corner point, x , at line 2. In each iteration, while the number of simulations, curr_simulation is less than the max_simulations, the algorithm picks a corner or random point x . Then, the system is simulated at the current point x to evaluate the specification. If the specification is falsified, the algorithm terminates, lines 7-8. Otherwise, if the specification is not falsified, for the next iteration, a new random or corner point x will be picked, lines 9-15.

It should be mentioned here that the number of corners depends on the dimension of the input parameters, x . Since the number of corners is constant, if there are no new corners to select, the algorithm will continue working with only random points, lines 10-14, until the maximum number of iterations have been executed.

IV. LINE-SEARCH FALSIFICATION (LSF)

In the previous section, HCR, an optimization-free approach was presented. From the evaluation in Section VI we will see that while the HCR method is quite successful in falsifying many of the specifications, there are also harder problems where an optimization-based approach might be more efficient. In this section, LSF is introduced, LSF is a gradient-free direct-search optimization-based approach. While HCR only relies on random and corner points, LSF combines random exploration with local search, by randomly generating lines in the n -dimensional parameter space. On these lines, corner points for different sets of parameters are evaluated together with a local search to find the minimal value of the objective function along the lines. When no improvements are reported for a number of iterations, a new line is generated and the search space explored LSF combines random search, with corner points, with a local search without being dependent on building a surrogate model of the objective function.

The method is presented in Algorithm 2, and has been implemented in Breach. The output of LSF is the tuple $\langle \text{Falsified/Not Falsified}, x \rangle$, where the first element describes if the specification was falsified or not, and the second element x is an n -dimensional parameter point where:

- 1) If the specification is falsified, $f(x) < 0$.
- 2) If the maximum number of simulations is reached and the specification is not falsified, x is the point with the minimum positive objective value, $f(x) \geq 0$.

A. Description

The method consists of the following steps. The method is generating random lines through the parameter space and then does local search along the line but always within the allowed parameter range. Technically, the local search is done along a *line segment* since it has two end points, and not along a *line* that extends infinitely in both directions. However, in the description below *line* is used to refer to both line segments and lines when the exact meaning is clear from the context.

1) *Initial Points*: LSF needs three parameter points during the optimization process, and a new point is generated in each iteration. First one point has to be identified, line 1. This point is the middle point $x = \frac{l+u}{2}$ in each of the n dimensions. Note that this point could instead be chosen randomly within the range (l, u) . The SUT is then simulated with x as input parameters, that define x^s as the discrete input signals, and the corresponding objective function value $f(x)$ is computed. This is handled by the call to Eval(x), which returns $f(x)$. If $f(x) < 0$, then the specification is falsified and the algorithm terminates. Otherwise, if curr_simulations is larger than one, the heuristic H_1 is called, lines 7-9. When curr_simulations is not larger than 1, H_1 is not called, as will be discussed later. Then, SelectPoints(x) is called, line 10.

Algorithm 2 Line-search Falsification Approach

```

1:  $x = \frac{(l+u)}{2}$ ;
2: global curr_simulation = 0;
3: while Eval( $x$ )  $\geq 0$  do
4:   if curr_simulation  $\geq$  max_simulations then
5:     Return (Not Falsified,  $x$ )
6:   end if
7:   if curr_simulations > 1 then
8:      $x = \text{Heuristic } H_1(x_{old}, x)$ 
9:   end if
10:   $(x_L, x_M, x_R) = \text{SelectPoints}(x)$ 
11:   $x_{old} = x$ ;
12:   $x = \text{FalsifyLine}(x_L, x_M, x_R, x)$ 
13: end while
14: Return (Falsified,  $x$ )

```

 $f(x) = \text{Eval}(x)$

```

15: map( $x, f(x)$ ) evaluations;
16: if  $x$  is not in the evaluations map then
17:   Simulate with  $x$  as input, compute  $f(x)$  using the quantitative semantics.
18:   Store  $(x, f(x))$  in the evaluations map;
19:   curr_simulations++;
20: end if
21: Look up the value,  $f(x)$ , of  $x$  in the evaluations map.
22: Return  $f(x)$ 

```

2) *SelectPoints*: The three points, x_M , x_L , and x_R , are defined and generated in the SelectPoints(x) function. SelectPoints(x)

 $(x_L, x_M, x_R) = \text{SelectPoints}(x)$

23: Pick a random line that goes through x .
 24: Compute the end-points x_L and x_R by using one of Options 1-4.
Option 1: $x_L = x$, $x_M = \frac{(x_R + x_L)}{2}$, x_R is a point where one of the dimensions is on the boundary of l or u .
Option 2: $x_L = x$, $x_M = \frac{(x_R + x_L)}{2}$, x_R is a point where half of the dimensions are on the boundary of l and u .
Option 3: $x_L = x$, $x_M = \frac{(x_R + x_L)}{2}$, x_R is a corner point.
Option 4: $x_M = x$, x_R and x_L are the points where at least one of the dimensions are on the boundary of l or u , or can be a corner points.
 25: **Return** (x_L, x_M, x_R)

 $x = \text{Heuristic } H_1(x_{old}, x)$

26: **if** $f(x_{old}) < f(x)$ **then**
 27: x = be the second point with lowest objective function value of the last call to **FalsifyLine**.
 28: **end if**
 29: **Return** x

 $x = \text{FalsifyLine}(x_L, x_M, x_R, x)$

30: iterations_without_improvement = 0;
 31: **while** iterations_without_improvement < max_iterations **do**
 32: **if** Eval(x_L) < 0 **then Return** x_L **end**
 33: **if** Eval(x_R) < 0 **then Return** x_R **end**
 34: **if** Eval(x_M) < 0 **then Return** x_M **end**
 35: **if** Eval(x_L) < Eval(x_R) **and** Eval(x_L) < Eval(x_M) **then**
 36: Let $x_{new} = \frac{(x_L + x_M)}{2}$,
 37: **if** Eval(x_{new}) < Eval(x_L) **then**
 38: $x = x_{new}$
 39: **end if**
 40: $x_R = x_M$, $x_M = x_{new}$
 41: **else if** Eval(x_R) < Eval(x_L) **and** Eval(x_R) < Eval(x_M) **then**
 42: Let $x_{new} = \frac{(x_M + x_R)}{2}$,
 43: **if** Eval(x_{new}) < Eval(x_R) **then**
 44: $x = x_{new}$
 45: **end if**
 46: $x_L = x_M$, $x_M = x_{new}$
 47: **else**
 48: **if** ($\|x_L - x_M\|_2 \geq \|x_M - x_R\|_2$) **then**
 49: Let $x_{new} = \frac{(x_L + x_M)}{2}$,
 50: **if** Eval(x_{new}) < Eval(x_M) **then**
 51: $x = x_{new}$
 52: $x_R = x_M$, $x_M = x_{new}$
 53: **else**
 54: $x_L = x_{new}$
 55: **end if**
 56: **else**
 57: Let $x_{new} = \frac{(x_M + x_R)}{2}$,
 58: **if** Eval(x_{new}) < Eval(x_M) **then**
 59: $x = x_{new}$
 60: $x_L = x_M$, $x_M = x_{new}$
 61: **else**
 62: $x_R = x_{new}$
 63: **end if**
 64: **end if**
 65: **end if**
 66: **if** $x == x_{new}$ **then**
 67: iterations_without_improvement = 0;
 68: **else**
 69: iterations_without_improvement++;
 70: **end if**
 71: **end while**
 72: **Return** x

picks a random line that goes through x and computes end-points x_L and x_R , lines 23-24. There are four options for how to pick these end-points, which we now describe.

3) *Pick line segments through point x :* Let $x = (x_1, \dots, x_n)^T$ where $l_i \leq x_i \leq u_i$ be an n -dimensional point within the given boundaries. Pick a random direction vector $d = (d_1, \dots, d_n)^T$ where $d_i \neq 0$ for $1 \leq i \leq n$. Define the function $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$ where each dimension i , $1 \leq i \leq n$, of g is defined by g_i as

$$g_i(z) = \begin{cases} u_i & \text{if } x_i + z d_i > u_i \\ x_i + z d_i & \text{if } l_i \leq x_i + z d_i \leq u_i \\ l_i & \text{if } x_i + z d_i < l_i \end{cases}$$

Let $z_k^+ \in \mathbb{R}_{>0}$ be the smallest positive value such that for at least k distinct dimensions,

$$g_i(z_k^+) = u_i \text{ or } g_i(z_k^+) = l_i.$$

Let $z_k^- \in \mathbb{R}_{<0}$ for $1 \leq k \leq n$ be the smallest negative value such that for at least k distinct dimensions,

$$g_i(z_k^-) = u_i \text{ or } g_i(z_k^-) = l_i.$$

In the sequel, the end-points defined by z_1^+ as Option 1, $z_{\lfloor \frac{n}{2} \rfloor}^+$ as Option 2 and z_n^+ as Option 3. Let $z_{j,k} = (z_j^-, z_k^+)$, $1 \leq j, k \leq n$ for Option 4.

For options 1-3, define end-points for a given z_k^+ as $x_L = x$, $x_R = g(z_k^+)$, and $x_M = \frac{(x_L + x_R)}{2}$. For Option 4, define end-points for a given $z_{j,k}$ as $x_L = g(z_j^-)$, $x_R = g(z_k^+)$, and $x_M = x$.

Note that which option to use is predetermined before Algorithm 2 starts, and the same option is used during the entire execution.

Fig. 2 exemplifies these four options in two dimensions $n = 2$, i.e., two input parameters with respective lower and upper bounds. We are allowed to take points within the input parameter box (l, u) . Thus, the four corner points are:

$$\begin{pmatrix} l_1 \\ l_2 \end{pmatrix}, \begin{pmatrix} l_1 \\ u_2 \end{pmatrix}, \begin{pmatrix} u_1 \\ l_2 \end{pmatrix}, \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}.$$

Fig. 2a shows an example for Option 1 where $k = 1$. Since this is a two-dimensional case, Option 1 and Option 2 will be the same. In this graph, the chosen line starts from x_L in the box and ends up at the point x_R where it cuts off at the upper bound of dimension 1.

Fig. 2b shows an example for Option 3 where $k = 2$. In this graph, the chosen line starts from x_L in the box and ends up at the point x_R where it cuts off the corner point $\begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$.

To get a clear understanding of how to choose a line according to Option 4, figures 2c and 2d are given. In Fig. 2c where $k = 1$ and $j = 1$, we work with a line segment where x_R is on the upper bound of the first dimension, u_1 , and x_L is on the lower bound of the first dimension, l_1 . The line can be a line that passes through x_M , but it can also be two line segments that connect x_M to x_L and x_M to x_R as is shown in Fig. 2d where $k = 2$ and $j = 1$. In this graph, x_L is on the lower bound of the first dimension, l_1 and x_R is a corner point $\begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$.

Fig. 2: Examples in two dimensions for the four options: a) Example for Option 1 and Option 2 where $k = 1$ and $x = x_L$; b) Example for Option 3 where $k = 2$ and $x = x_L$; c) and d) Examples for Option 4 where $k = 1, j = 1$ and $k = 2, j = 1$, respectively and $x = x_M$.

4) Comparison among the four options to choose a line:

Which option performs best for a given SUT and specification is heavily application dependent, and thus comparing them has no clear outcome.

In Option 1 and Option 2, we work with lines that cut off at the upper or lower bound in one and half of the dimensions, respectively. In these options, one of the points, x_R , is on the bound of the box. In Option 3, one of the points, x_R , is on a corner point. On the other hand, in Option 4, x_R and x_L are either on a bound of the box or on corner points.

Option 1 and Option 2 search more points on the boundaries, while Option 3 searches more corner points and towards them if the optimization process guides in that direction. Since lines are chosen randomly, options 1-3 may lead to using short lines that do not guide the process towards falsification or might get stuck for some iterations in a local area. On the other hand, Option 4 works with long lines, extending between the boundaries, which has a higher chance of finding a better point. A comprehensive comparison from the practical perspective based on the evaluated examples in this paper will be given in Section VI.

Admittedly, there are many possible ways to pick the lines. Heuristics for how to choose a line is an open question for future research.

5) *Algorithm's main loop in FalsifyLine:* Now it is time to perform the falsification process over a line by calling FalsifyLine, line 12. Before calling this function, we need to save the current point x in a variable x_{old} at line 11. FalsifyLine tries to find a new point with lower objective function value. The ultimate goal being to find a point with negative objective value.

The FalsifyLine algorithm is the local search loop in LSF. The SUT is simulated and evaluated at the three parameter

points x_M , x_L , and x_R by calling the Eval function, lines 32-34. If the specification is falsified, i.e., the objective function value is negative, the algorithm terminates. Otherwise, if the specification is not falsified at any of the points x_L , x_M , or x_R , the algorithm starts the local search by looking for new points that can potentially have lower object values.

The variable `iterations_without_improvement` is used to count the number of iterations that we stay with a line inside FalsifyLine. The local search along the line is completed when `iterations_without_improvement` has reached `max_iterations`.

There are three cases, depending on which of the points x_M , x_L , and x_R has the smallest objective function value. In each iteration, a new point is searched that is the middle point between x_M and x_L , or x_M and x_R . This new point will first be evaluated to decide if it falsifies the specification or not. If it does not, the new point will be replaced by one of the three points, x_L , x_M , or x_R , based on some rules presented later. The optimization will now be done on a shorter line segment than the initial line segment, defined by the selected end-points, for the next iteration. How to select the new point and the shorter line segment are based on the function evaluation presented below.

1. If the point x_L has the lowest objective value among the three points, lines 35-40, i.e.

$$f(x_L) < f(x_M) \text{ and } f(x_L) < f(x_R).$$

If the above condition is satisfied then let $x_{new} = \frac{x_L + x_M}{2}$ be defined as the middle point. Then, check whether x_{new} has a lower objective function value than x_L or not, if it does, then update $x = x_{new}$. For the next iteration, x_R will be omitted and replaced to work with a shorter line that connects x_M to x_L and passing from x_{new} . The new points are

$$x_R = x_M \text{ and } x_M = x_{new}. \quad (2)$$

The point x_L , that was the point with the lowest objective function value is not changed.

2. When the point x_R has the lowest objective value among the three points, lines 41-46 i.e.,

$$f(x_R) < f(x_M) \text{ and } f(x_R) < f(x_L).$$

Compute the middle point $x_{new} = \frac{x_M + x_R}{2}$. Then, check to see whether x_{new} has a lower objective function value than x_R or not, if it does thus update $x = x_{new}$. For the next iteration, x_L will be omitted and replaced to work with a shorter line segment that connects x_M to x_R and passing from x_{new} . The three points for the next iteration are

$$x_L = x_M \text{ and } x_M = x_{new}. \quad (3)$$

The point x_R , with the lowest objective function value, will not change.

3. If the point x_M has the lowest objective value among the three points, lines 47-62 i.e.,

$$f(x_M) \leq f(x_L) \text{ and } f(x_M) \leq f(x_R).$$

Here, we need to decide on which line segment to continue the search, i.e., the line segment (x_M, x_L) or (x_M, x_R) . We do an

evaluation that depends on which interval (x_L, x_M) or (x_M, x_R) that is longer, for this purpose we use the Euclidean distance ($\|\cdot\|_2$). x_{new} is selected to be the middle point on the longer of the two segments.

$$x_{new} = \begin{cases} \frac{x_L + x_M}{2}, & \text{if } \|x_L - x_M\|_2 \geq \|x_M - x_R\|_2 \\ \frac{x_M + x_R}{2}, & \text{otherwise.} \end{cases} \quad (4)$$

When $\|x_L - x_M\|_2 \geq \|x_M - x_R\|_2$ the end-points are updated as follows.

$$\begin{cases} x_R = x_M \text{ and } x_M = x_{new}, & \text{if } f(x_{new}) < f(x_M), \\ x_L = x_{new}, & \text{otherwise.} \end{cases} \quad (5)$$

When $\|x_L - x_M\|_2 < \|x_M - x_R\|_2$ the end-points are updated as follows.

$$\begin{cases} x_L = x_M \text{ and } x_M = x_{new}, & \text{if } f(x_{new}) < f(x_M), \\ x_R = x_{new}, & \text{otherwise.} \end{cases} \quad (6)$$

Those points that are not explicitly updated in (5) and (6) will keep their previous value. Note that in this case, where x_M has the lowest objective function value, the decision about which line segment to continue on is based solely on the length of the line segments. This is for two reasons: (1) The longest line segment is a sensible choice because this is the line segment that has the largest uncertainty in the meaning being least explored. (2) The second reason is to reduce the number of line segments to consider if also the objective function values of x_L and x_R were used. This would add complexity since it would require to balance between distance and objective function value.

Based on which of the above cases happens, in each iteration, one of the new points $\frac{(x_L + x_M)}{2}$ or $\frac{(x_M + x_R)}{2}$ is searched, and one of the old three points will be replaced with the new point.

If we can improve the point x and find a better point with lower objective function value, where $x = x_{new}$, the iterations_without_improvement resets to zero, line 67. This assumption leads to work more with the lines that can search points with lower objective function values. On the other hand, if we cannot improve the point x , iterations_without_improvement is increased by 1, line 69. This assumption leads to less work with the lines that do not help get the points with lower objective function values. When iterations_without_improvement reaches max_iterations, we get out from the current chosen line and pick a new one.

When iterations_without_improvement has reached max_iterations, the FalsifyLine process is finished and it returns a value for x . Now we are at line 3. If the specification is not falsified yet and curr_simulation has not reached max_simulations and curr_simulation > 1, then Heuristic H1 is called at line 8.

6) *Heuristic H₁*: In function Heuristic H₁, we check the objective function of x_{old} with x . If $f(x_{old}) < f(x)$, FalsifyLine could not give a better point with less objective function value than x_{old} . Hence, to force the algorithm to never return the same x as was given to FalsifyLine, we consider x to be the point with the *second lowest* objective function value of the

last call to FalsifyLine. Our experimental evaluations show that this heuristic helps the algorithm to avoid getting stuck in local minima.

7) *Stopping condition*: This algorithm works until a falsified point is found, i.e., a point with a negative objective function value, or curr_simulation reaches max_simulations. If a falsified point is not found, the algorithm returns the point with the lowest objective function value found so far.

In the following section the benchmark problems that will be used to evaluate the suggested methods, are presented.

V. BENCHMARK PROBLEMS

We consider the simulation-based falsification for all benchmark examples of the ARCH workshop [30] and additional benchmark problems present in [31]. The paper [30] describes a friendly competition in the ARCH19 workshop for the falsification of temporal logic specifications over CPSs. Table I demonstrates the STL specifications for all the example models. These examples are briefly introduced in the following.

TABLE I: Specifications to falsify for all benchmark examples

Spec.	Formula
φ_1^{AT}	$\Box_{[0,20]}(v < 120)$
φ_2^{AT}	$\Box_{[0,10]}(\omega < 4750)$
φ_3^{AT}	$\Box_{[0,30]}((\neg g_1 \wedge g_1) \Rightarrow \Diamond_{[0,2.5]} g_1)$
φ_4^{AT}	$\Box_{[0,30]}((\neg g_2 \wedge g_2) \Rightarrow \Diamond_{[0,2.5]} g_2)$
φ_5^{AT}	$\Box_{[0,30]}((\neg g_3 \wedge g_3) \Rightarrow \Diamond_{[0,2.5]} g_3)$
φ_6^{AT}	$\Box_{[0,30]}((\neg g_4 \wedge g_4) \Rightarrow \Diamond_{[0,2.5]} g_4)$
φ_7^{AT}	$(\Box_{[0,30]} \omega < 3000) \Rightarrow (\Box_{[0,4]} v < 35)$
φ_8^{AT}	$(\Box_{[0,30]} \omega < 3000) \Rightarrow (\Box_{[0,8]} v < 50)$
φ_9^{AT}	$(\Box_{[0,30]} \omega < 3000) \Rightarrow (\Box_{[0,20]} v < 65)$
φ_1^{AFC}	$\Box_{[11,50]}(((\theta < 8.8) \wedge (\Diamond_{[0,0.05]}(\theta > 40)) \vee (\theta > 40) \wedge (\Diamond_{[0,0.05]}(\theta < 8.8)) \Rightarrow (\Box_{[1,5]} \mu < 0.008))$
φ_2^{AFC}	$\Box_{[11,50]} \mu < 0.007$
φ_1^{NN}	$\Box_{[1,37]}(Pos - Ref > 0.005 + 0.03 Ref \Rightarrow \Diamond_{[0,2]} \Box_{[0,1]} (\neg 0.005 + 0.03 Ref \leq Pos - Ref))$
φ_2^{NN}	$\Box_{[1,37]}(Pos - Ref > 0.005 + 0.04 Ref \Rightarrow \Diamond_{[0,2]} \Box_{[0,1]} (\neg 0.005 + 0.04 Ref \leq Pos - Ref))$
φ_1^{WT}	$\Box_{[30,630]} \theta \leq 14.2$
φ_2^{WT}	$\Box_{[30,630]} 21000 \leq M_{g,d} \leq 47500$
φ_3^{WT}	$\Box_{[30,630]} \Omega \leq 14.3$
φ_4^{WT}	$\Box_{[30,630]} \Diamond_{[0,5]} \theta - \theta_d \leq 1.6$
φ_1^{CC}	$\Box_{[0,100]} (y_5 - y_4 \leq 40)$
φ_2^{CC}	$\Box_{[0,100]} \Diamond_{[0,30]} y_5 - y_4 \geq 15$
φ_3^{CC}	$\Box_{[0,80]} ((\Box_{[0,20]} y_2 - y_1 \leq 20) \vee (\Diamond_{[0,20]} y_5 - y_4 \geq 40))$
φ_4^{CC}	$\Box_{[0,65]} \Diamond_{[0,30]} \Box_{[0,20]} y_5 - y_4 \geq 8$
φ_5^{CC}	$\Box_{[0,72]} \Diamond_{[0,8]} ((\Box_{[0,5]} y_2 - y_1 \geq 9) \Rightarrow (\Box_{[5,20]} y_5 - y_4 \geq 9))$
φ^{F16}	$\Box_{[0,15]} altitude > 0$
φ^{SC}	$\Box_{[30,35]} (87 \leq pressure \wedge pressure \leq 87.5)$
$\varphi_1^{AT'}$	$\Diamond_{[0,T]} (\omega \geq 2000)$
$\varphi_2^{AT'}$	$\Box_{\Diamond_{[0,T]}} (\omega \leq 3500 \vee \omega \geq 4500)$
$\varphi_3^{AT'}$	$\Box_{[0,T]} (\neg (gear == 4))$
$\varphi_4^{AT'}$	$\Diamond_{\Box_{[0,T]}} (gear == 3)$
$\varphi_5^{AT'}$	$\bigwedge_{i=1,\dots,4} \Box((\neg (gear == i) \wedge \Diamond_{[0,\epsilon]} (gear == i)) \Rightarrow (\Box_{[\epsilon,T+\epsilon]} (gear == i)))$
$\varphi_6^{AT'}$	$\Box_{[0,T]} (v \leq 85) \vee \Diamond (\omega \geq 4500)$
$\varphi_7^{AT'}$	$\neg((\Box_{[0,1]} gear == 1) \wedge (\Box_{[2,4]} gear == 2) \wedge (\Box_{[5,7]} gear == 3) \wedge (\Box_{[8,10]} gear == 3) \wedge (\Box_{[12,15]} gear == 2))$
$\varphi_8^{AT'}$	$\Box_{[0,20]} ((gear == 4 \wedge throttle > 45 \wedge throttle < 50) \Rightarrow \omega < \bar{\omega})$
$\varphi^{A-\Sigma}$	$\Box (\bigwedge_{i=1}^3 (-1 \leq x_i \wedge x_i \leq 1))$
φ^{SS}	$\Box (y \geq 0)$

A. ARCH Benchmark Problems

For the ARCH Benchmark problems [30], two variants of input signals are considered, called Instance 1 and Instance 2.

- *Instance 1: Arbitrary piece-wise continuous input signals.* This variant considers the entire set of piece-wise continuous input signals where the values for each dimension are from a given range. Finite number of discontinuities and finite variability for all continuous parts of inputs are considered as additional constraints.
- *Instance 2: Constrained input signals.* This variant fixes the format of the input signal, allowing discontinuities. An example input signal would be a piece-wise constant signal with k equally spaced control points, with ranges for each dimension of the input.

For more details about the two variants of input signals, see [30].

1) *Automatic Transmission (AT):* The controller of this model selects a gear 1 to 4, depending on inputs *throttle* and *brake*; rotations per minute (ω); and car speed (v) [33]. Two different control inputs are considered for this example:

- Instance 1: $0 \leq \text{throttle} \leq 100$ and $0 \leq \text{brake} \leq 325$, where both can be active at the same time.
- Instance 2: Constrained input signals with discontinuities at most every 5 time units.

2) *Chasing Cars (CC):* The model is a simple model of an automatic chasing car [34]. The CC model consists of five cars, where the first car is driven by inputs (*throttle* and *brake*), and the other four cars are driven by Hu et al.'s algorithm. The location of the five cars y_1, y_2, y_3, y_4, y_5 is the system output. The inputs are:

- Instance 1: The input specifications allow any piece-wise continuous signals.
- Instance 2: The input specifications constrain inputs to piece-wise constant signals with control points for every 5 seconds, 20 segments.

3) *Fuel Control of an Automotive Power Train (AFC):* This system is modeled on [35]. The constrained input signal fixes the throttle θ to be piece-wise constant with 10 uniform segments over a time horizon of 0 with two modes and the engine speed ω to be constant with $900 \leq \omega < 1100$.

4) *Neural Network Controller (NN):* This benchmark is based on the neural network of the NARMA-L2 [36] controller. It is designed for a system that levitates a magnet above an electromagnet at a reference position. A reference value Ref for the position, where $1 \leq Ref \leq 3$, is the only input of this model. The current position of the levitating magnet Pos is the output. The two variants of inputs considered for this model are:

- Instance 1: The input specification requires discontinuities to be at least 3 time-units long.
- Instance 2: An input signal with exactly three constant segments is required.

5) *Aircraft Ground Collision Avoidance System (F16):* The F16 aircraft and its inner-loop controller are modeled for Ground Collision avoidance. 16 continuous variables with piece-wise nonlinear differential equations are modeled [37]. The

system is required to always avoid hitting the ground during its maneuvers, starting from all the initial conditions for roll, pitch, and yaw in the range $[0.2\pi, 0.2833\pi] \times [-0.5\pi, -0.54\pi] \times [0.25\pi, 0.375\pi]$.

6) *Steam Condenser with Recurrent Neural Network Controller (SC):* This is a dynamic model of a steam condenser based on energy balance and cooling water mass balance, controlled with a Recurrent Neural network in feedback [38]. The input can vary in the range $[3.99, 4.01]$, and the input signal should be piece-wise constant with 20 evenly spaced segments.

7) *Wind Turbine (WT):* A simplified wind turbine model from [39] is considered. The input is the wind speed v , while the outputs are blade pitch angle θ , generator torque $M_{g,d}$, rotor speed Ω , and demanded blade pitch angle θ_d . The wind speed is constrained by $8.0 \leq v \leq 16.0$.

B. Additional Benchmark Problems

1) *Automatic Transmission (AT'):* The inputs to the model are the throttle and brake of a vehicle. The outputs of the model are the vehicle speed v , the engine speed ω , and the gear, see [40] for details. This example has different specifications from the ARCH example presented in Section V-A1.

2) *Third Order $\Delta - \Sigma$ Modulator:* The third order $\Delta - \Sigma$ modulator is a model of a technique for analog to digital conversion. It has one input U , three states x_1, x_2, x_3 , and three initial conditions $x_1^{init}, x_2^{init}, x_3^{init}$, see [41] for details.

3) *Static Switched (SS) System:* The static switched system is a model without any dynamics that is included as a simple case. The model is inspired by [42].

VI. EXPERIMENTAL SETUP AND RESULTS

In this section, the Corners and random methods are compared with the HCR approach. Note that the Uniform-Random (UR) method is used in this paper. The LSF method is compared with NM, SNOBFIT, and HCR. Finally, all methods are compared together. The two semantics *Max* and *Additive* are considered in this paper. The results are shown in tables II to XI. Each falsification is set to have $\text{max_simulations} = 1000$. There are 20 falsification runs for each method and objective function to account for most algorithms' random nature.

For Corners, UR, or HCR, it does not matter which semantics is used, thus they are evaluated by only one semantics.

1) *Corners Implementation Setup:* Each specification and example has a different number of corners. For some examples, the number of corners is less than the maximum number of simulations. Then, the Corners method terminates when all corners have been evaluated.

2) *Uniform-Random Implementation Setup:* For each of 20 falsification runs, uniformly distributed random points are generated from different seeds, i.e., 20 different seeds are considered.

3) *Hybrid Corner-Random Implementation Setup:* The HCR method starts with the Corners point, and the next point is the UR point. It switches between the Corners and UR until the maximum number of simulations, 1000 here, is reached or a falsified point is found. The number of corners is limited, and

it depends on how many corners points the SUT has. If the maximum number of corners is reached, the HCR algorithm continues using only the UR points.

4) *Nelder-Mead Implementation Setup*: This algorithm is implemented as *fminsearch* [43] in Matlab. Before starting NM, 100 random sampling points for all examples are generated. After evaluating 100 random sampling points, the point with the lowest objective value is picked as a starting point if none falsifies the specification. NM needs a simplex of $n + 1$ points for n -dimensional vectors, while we have one point. The n points will be generated in the *fminsearch* of MATLAB using that minimum point. The minimum point is similar to x_0 when calling *fminsearch*(f, x_0) in MATLAB, where it starts at the point x_0 and tries to find a local minimum.

Some parameter values are used by *fminsearch*. The first one is the maximum number of iterations. The default is $200 \times n$. The second one is the maximum number of evaluations of the function. The default is 150 in this paper.

Unless falsification occurs, NM exits when one of the following conditions hold:

- The maximum number of iterations has been reached.
- The maximum number of evaluations of the function has been reached.
- $f(x_{n+1}) - f(x_1) < \epsilon$, $\epsilon > 0$ is a small predetermined tolerance.

If one of the above conditions is fulfilled, 100 new random points are generated, and the NM process restarts.

5) *Line-search Falsification Implementation Setup*: The value of `max_iterations` is set to 3.

In the tables II-XI, the first row denotes the specification, see Table I, the second row is the semantics, the third row contains the optimization method, and the last row the optimization-free methods. For each specification the first column show the results with the *Max* quantitative semantics, and the second column the results with the *Additive* quantitative semantics. For each semantics, two values are presented. The first value is the relative success rate of falsification in percent. There are 20 falsification runs for each parameter value and specification, thus the success rate will be a multiple of 5%. The second value, inside parentheses, is the average number of simulations (rounded) *per successful falsification*. The presented results for LSF are using Option 4 presented previously.

A. Optimization-free vs. Optimization-based falsification

First, we compare the optimization-free methods Corners, UR, and HCR. Based on the results shown in tables II-XI, for many specifications, one of Corners and UR are able to successfully. Hence, HCR outperforms for many of specifications. For a comparison between Corners, UR, and HCR, a cactus plot is shown in Fig 3. The results presented in this plot relate to all examples. As can be seen in Fig 3, while UR is more successful than Corners, for those cases that are falsified at the corner points, Corners falsifies much faster and with fewer simulations. Thus, HCR manages to falsify more examples than either of the other two.

To get a better understanding of the difference in performance between optimization-free and optimization-based methods, we

go through some examples in more detail to compare HCR with NM, SNOBFIT, and LSF.

For φ_2^{AT} in Instance 2, Table II, NM is not 100 percent successful, and SNOBFIT and LSF need more simulations than Corners and HCR. On the other hand, HCR manages to falsify the property with just a few simulations. As can be seen in Table III, all optimization-based methods needed more simulations to falsify φ_2^{CC} for Instance 2. Also, using *Max* semantics is not 100% successful. On the other hand, with only one simulation, HCR falsifies this specification. Similarly, HCR performs better than the optimization-based methods for specifications $\varphi_1^{AT'} (T = 30)$ and $(T = 40)$, $\varphi_3^{AT'} (T = 4.5)$, $\varphi_4^{AT'} (T = 1)$, $\varphi_8^{AT'} (\bar{\omega} = 3000)$ and $(\bar{\omega} = 3500)$ in Table IX. There is a significant advantage for HCR for the specifications $\varphi_1^{\Delta-\Sigma} (U \in [-0.40, 0.40])$ and $(U \in [-0.45, 0.45])$ in Table X.; and all specifications in Table XI where HCR manages to falsify them with fewer simulations.

To compare Corners, UR, and HCR with NM, SNOBFIT and LSF, a cactus plot is shown in Fig 4 for those examples that using an optimization-based approach to improve the falsification process is not needed. As can be seen in this figure, HCR beats all methods and manages to falsify more examples than either of the optimization methods and requires fewer simulations than all other methods.

On the other hand, there are specifications that neither Corners nor UR can falsify. Hence, the HCR method will not be successful either. These are the specifications 7-9 of the AT example for Instance 2, Table II; specification 4 of both instances 1 and 2 of the CC example, Table III; specification 2 of the NN example for Instance 1, Table V; the F16 example Table VI; specifications 6 for both $T = 10, T = 12$ and 7 of AT' , Table IX; and the first specification of the Modulator example, Table X.

B. Line-search Falsification using the four options

Section IV introduced four different options for LSF to pick a line. A cactus plot comparing these options is given in Fig 5.

Option 1, where we work with the lines that cut off at the upper or lower bound in one of the dimensions, does not work as well as the other options. Option 2 and Option 3 perform approximately the same, where Option 2 works with the lines that cut off at the half of the dimensions, and Option 3 searches more corner points. As can be seen in Fig. 5, Option 3 is more successful and requires fewer simulations. Option 3 is a good approach for those easy benchmark examples that can be falsified on the corners. On the other hand, Option 4 is more successful in the falsification process because it uses the combination of three options. Hence, in the tables, we just quote the results using Option 4 for LSF.

C. Line-search Falsification vs. Nelder-Mead and SNOBFIT

This part compares the results of the new LSF method with NM and SNOBFIT for those specifications and examples that it is not possible to falsify by Corners and UR. It is only for a subset of the specifications and models that an optimization methods can perform better than an optimization-free approach. These are specifications 1 and 7-9 of the AT example for

TABLE II: Results for the AT system, Instances 1 and 2.

Instance 1										
Specifications	φ_1^{AT}		φ_2^{AT}		φ_3^{AT}		φ_4^{AT}		φ_5^{AT}	
Semantics	Max	Add	Max	Add	Max	Add	Max	Add	Max	Add
Nelder-Mead	100 (213)	100 (227)	100 (17)	100 (8)	100 (24)	100 (19)	100 (14)	100 (15)	100 (13)	100 (15)
SNOBFIT	100 (40)	100 (38)	100 (6)	100 (6)	100 (8)	100 (8)	100 (14)	100 (30)	100 (11)	100 (9)
Line-search Falsification	100 (34)	100 (40)	100 (9)	100 (10)	100 (37)	100 (34)	100 (31)	100 (36)	100 (21)	100 (17)
Corners	100 (3)	0 (-)	100 (2)	100 (10)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)
Uniform-Random	100 (5)	100 (3)	100 (3)	100 (3)	100 (13)	100 (27)	100 (13)	100 (25)	100 (11)	100 (23)
Hybrid Corner-Random	100 (5)	100 (3)	100 (3)	100 (3)	100 (13)	100 (27)	100 (13)	100 (25)	100 (11)	100 (23)
Instance 2										
Specifications	φ_6^{AT}		φ_7^{AT}		φ_8^{AT}		φ_9^{AT}		φ_{10}^{AT}	
Semantics	Max	Add	Max	Add	Max	Add	Max	Add	Max	Add
Nelder-Mead	100 (57)	100 (73)	100 (43)	100 (48)	100 (57)	100 (103)	100 (43)	100 (29)	100 (45)	100 (23)
SNOBFIT	100 (106)	100 (87)	100 (54)	100 (38)	100 (91)	100 (70)	100 (52)	100 (1)	100 (17)	100 (16)
Line-search Falsification	100 (63)	100 (112)	100 (13)	100 (24)	100 (26)	100 (28)	100 (22)	100 (12)	100 (67)	100 (93)
Corners	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)
Uniform-Random	100 (59)	100 (117)	100 (65)	100 (130)	100 (96)	100 (178)	100 (31)	100 (61)	100 (21)	100 (17)
Hybrid Corner-Random	100 (59)	100 (117)	100 (65)	100 (130)	100 (96)	100 (178)	100 (31)	100 (61)	100 (21)	100 (17)
Instance 1										
Specifications	φ_1^{AT}		φ_2^{AT}		φ_3^{AT}		φ_4^{AT}		φ_5^{AT}	
Semantics	Max	Add	Max	Add	Max	Add	Max	Add	Max	Add
Nelder-Mead	0 (-)	0 (-)	80 (263)	90 (234)	100 (5)	100 (5)	100 (1)	100 (1)	100 (1)	100 (1)
SNOBFIT	0 (-)	0 (-)	100 (76)	100 (59)	100 (7)	100 (5)	100 (2)	100 (1)	100 (1)	100 (1)
Line-search Falsification	0 (-)	0 (-)	100 (187)	100 (96)	100 (10)	100 (13)	100 (3)	100 (3)	100 (2)	100 (2)
Corners	0 (-)	0 (-)	100 (2)	100 (2)	100 (5)	100 (5)	100 (3)	100 (3)	100 (3)	100 (3)
Uniform-Random	0 (-)	0 (-)	100 (288)	100 (3)	100 (5)	100 (7)	100 (2)	100 (2)	100 (1)	100 (1)
Hybrid Corner-Random	0 (-)	0 (-)	100 (3)	100 (3)	100 (5)	100 (7)	100 (3)	100 (3)	100 (2)	100 (2)
Instance 2										
Specifications	φ_6^{AT}		φ_7^{AT}		φ_8^{AT}		φ_9^{AT}		φ_{10}^{AT}	
Semantics	Max	Add	Max	Add	Max	Add	Max	Add	Max	Add
Nelder-Mead	100 (2)	100 (2)	0 (-)	5 (597)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)
SNOBFIT	100 (2)	100 (2)	45 (357)	45 (304)	15 (621)	30 (369)	75 (194)	65 (243)	100 (75)	100 (41)
Line-search Falsification	100 (4)	100 (6)	100 (121)	100 (122)	100 (127)	100 (214)	100 (75)	100 (41)	100 (75)	100 (41)
Corners	100 (3)	100 (3)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)
Uniform-Random	100 (2)	100 (2)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)
Hybrid Corner-Random	100 (3)	100 (3)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)

TABLE III: Results for the CC system, Instances 1 and 2.

Instance 1										
Specifications	φ_1^{CC}		φ_2^{CC}		φ_3^{CC}		φ_4^{CC}		φ_5^{CC}	
Semantics	Max	Add	Max	Add	Max	Add	Max	Add	Max	Add
Nelder-Mead	100 (6)	100 (11)	100 (5)	100 (5)	100 (22)	100 (21)	0 (-)	0 (-)	100 (45)	100 (23)
SNOBFIT	100 (4)	100 (5)	100 (5)	100 (3)	100 (5)	100 (5)	55 (382)	60 (334)	100 (17)	100 (16)
Line-search Falsification	100 (8)	100 (9)	100 (15)	100 (10)	100 (14)	100 (19)	40 (381)	35 (623)	100 (19)	100 (37)
Corners	100 (3)	100 (3)	100 (1)	100 (1)	100 (3)	100 (3)	0 (-)	0 (-)	0 (-)	0 (-)
Uniform-Random	100 (8)	100 (8)	100 (5)	100 (5)	100 (17)	100 (17)	0 (-)	0 (-)	100 (18)	100 (18)
Hybrid Corner-Random	100 (5)	100 (5)	100 (1)	100 (1)	100 (5)	100 (5)	0 (-)	0 (-)	100 (36)	100 (36)
Instance 2										
Specifications	φ_6^{CC}		φ_7^{CC}		φ_8^{CC}		φ_9^{CC}		φ_{10}^{CC}	
Semantics	Max	Add	Max	Add	Max	Add	Max	Add	Max	Add
Nelder-Mead	100 (131)	100 (142)	90 (245)	100 (96)	100 (21)	100 (37)	0 (-)	0 (-)	100 (152)	100 (55)
SNOBFIT	100 (32)	100 (29)	90 (129)	100 (125)	100 (8)	100 (12)	0 (-)	0 (-)	100 (48)	100 (38)
Line-search Falsification	100 (32)	100 (26)	90 (304)	100 (218)	100 (13)	100 (11)	10 (620)	10 (755)	100 (67)	100 (93)
Corners	100 (3)	100 (3)	100 (1)	100 (1)	100 (3)	100 (3)	0 (-)	0 (-)	100 (21)	100 (21)
Uniform-Random	100 (67)	100 (67)	100 (157)	100 (1)	100 (24)	100 (5)	0 (-)	0 (-)	100 (96)	100 (96)
Hybrid Corner-Random	100 (5)	100 (5)	100 (1)	100 (1)	100 (5)	100 (5)	0 (-)	0 (-)	100 (38)	100 (38)

Fig. 3: A cactus plot showing the performance of Corners, UR, and HCR for all examples. The plotted values show how many successful falsifications (x -axis) were completed in less than 1000 simulations (y -axis, logarithmic scale).

Fig. 4: A cactus plot showing performance of Corners, UR, and HCR against NM, SNOBFIT, and LSF for those examples that using an optimization-based approach to improve the falsification process is not needed. The plotted values show how many successful falsifications (x -axis) were completed in less than 1000 simulations (y -axis, logarithmic scale).

Fig. 5: A cactus plot showing the performance of each combination of quantitative semantics and LSF, using different options for all examples. The plotted values tell how many successful falsifications (x -axis) were completed in less than 1000 simulations (y -axis, logarithmic scale).

TABLE IV: Results for the AFC system

Specifications	φ_1^{AFC}		φ_2^{AFC}	
	Max	Add	Max	Add
Semantics				
Nelder-Mead	100 (148)	95 (318)	100 (17)	100 (13)
SNOBFIT	100 (33)	100 (27)	100 (8)	100 (7)
Line-search Falsification	100 (6)	100 (9)	100 (2)	100 (3)
Corners		100 (3)		100 (3)
Uniform-Random		100 (364)		100 (16)
Hybrid Corner-Random		100 (5)		100 (5)

Instance 2, Table II; the fourth specification of both instances 1 and 2 of the CC example, Table III; the second specification of the NN example for Instance 1, Table V; the F16 example Table VI; specifications 6 for both $T = 10, T = 12$ and 7 of AT' , Table IX; and the first specification of the Modulator example, Table X. For some specifications and systems, there is a clear tendency for a specific optimization method and semantics to perform better.

TABLE V: Results for the NN system, Instances 1 and 2.

Instance 1				
Specifications	φ_1^{NN}		φ_2^{NN}	
	Max	Add	Max	Add
Semantics				
Nelder-Mead	100 (71)	100 (81)	15 (487)	10 (434)
SNOBFIT	100 (77)	100 (87)	35 (379)	20 (440)
Line-search Falsification	100 (65)	100 (62)	45 (400)	25 (289)
Corners		0 (-)		0 (-)
Uniform-Random		100 (117)		0 (-)
Hybrid Corner-Random		100 (125)		0 (-)
Instance 2				
Specifications	φ_1^{NN}		φ_2^{NN}	
	Max	Add	Max	Add
Semantics				
Nelder-Mead	100 (28)	100 (19)	0 (-)	5 (164)
SNOBFIT	100 (18)	100 (20)	75 (342)	85 (361)
Line-search Falsification	100 (25)	100 (18)	80 (253)	90 (354)
Corners		100 (36)		100 (42)
Uniform-Random		100 (29)		5 (506)
Hybrid Corner-Random		100 (39)		100 (83)

Instance 2 of the AT example, Table II, is an excellent example to show the performance of LSF. Except for φ_1^{AT} , which can be falsified using a method not considered by

TABLE VI: Results for the F16.

Specifications	φ^{F16}	
Semantics	Max	Add
Nelder-Mead	25 (343)	20 (319)
SNOBFIT	75 (189)	65 (420)
Line-search Falsification	65 (530)	60 (399)
Corners	0 (-)	
Uniform-Random	5 (759)	
Hybrid Corner-Random	5 (767)	

TABLE VII: Results for the SC system.

Specifications	φ^{SC}	
Semantics	Max	Add
Nelder-Mead	0 (-)	0 (-)
SNOBFIT	0 (-)	0 (-)
Line-search Falsification	0 (-)	0 (-)
Corners	0 (-)	
Uniform-Random	0 (-)	
Hybrid Corner-Random	0 (-)	

this paper [30]. LSF manages to falsify 100% of the other specifications, while NM cannot falsify any of the φ_i^{AT} with $i = 7, 8, 9$. There is also a significant advantage for LSF relative SNOBFIT for these specifications.

In the CC example Instance 1, Table III, φ_4^{CC} while NM is not successful in the falsification process, SNOBFIT and LSF are successful in some runs where SNOBFIT works better than LSF. In Instance 2 of this example, a 10% improvement happens in CC, Instance 2 using LSF, it gives this result that it is possible to falsify this specification where none of NM and SNOBFIT are successful. In the NN example, for Instance 1 of specification φ_2^{NN} (Table V), LSF and SNOBFIT perform similarly and better than NM, respectively. In the F16 example, Table VI, LSF and SNOBFIT work similarly and better than NM, respectively. In the AT' , Table IX, only in the two specifications $\varphi_6^{AT'} (T = 10)$, and $(T = 12)$, NM performs better than two others. Compared to SNOBFIT, which is not successful for $\varphi_6^{AT'} (T = 12)$, LSF performs better and it manages to falsify this specification in some runs. $\varphi_7^{AT'}$ is the only specification of this example where SNOBFIT works better than LSF. In the $\Delta - \Sigma$ Modulator example, Table X, LSF manages to falsify to 100%, which is a slight improvement over SNOBFIT, and a big improvement over NM.

An aggregated comparison is shown in Fig. 6. As can be seen, and is evident from the discussion above, LSF outperforms NM. LSF can falsify using fewer simulations, which is important as simulations are costly. Also, LSF also works better than SNOBFIT while being simple. Comparing the semantics, *Additive* works slightly better than *Max*.

D. Evaluation and Discussion

An aggregated comparison among all methods presented and discussed in this paper for all examples is shown in Fig. 7. As can be seen, HCR not only manages to falsify more examples than Corners and UR but also has a better performance than NM using both *Max* and *Additive* semantics.

LSF, compared to the other evaluated optimization-free and optimization-based approaches, falsifies the most examples, as shown in Fig. 7. The LSF method covers the corners points and combines randomly generated lines in the n -dimensional parameter range with local search to guide the search. This method has a simple strategy compared to SNOBFIT, but

a performance that is on par with or better than SNOBFIT on the benchmark problems. LSF can falsify some examples that none of the other evaluated methods could successfully falsify. Instance 2 of CC is one example that shows the better performance of LSF.

The main focus in this paper has been to develop and describe an approach that works well for falsification problems by investigating and exploiting the typical properties of these problems. It is beyond the scope of this paper to discuss statistical guarantees or local and/or global convergence. However, we note that global convergence can typically be shown for random search algorithms, but local convergence requires assumptions on the underlying function. The evaluation is instead done using an extensive evaluation on available benchmark problems. The nature of the complexity of the SUT present in many industrial applications, typically having discrete modes where the dynamics might change completely, invalidate the assumptions typically necessary for giving any guarantees.

We have evaluated the approach on the set of benchmark problems that are available within the falsification community. We encourage researchers to post more problems, problems of larger complexity, and problems with more involved specifications, since this would increase the quality of the evaluations. However, it is important to note that problems used are not cherry-picked to show the benefits of the proposed approach, but they are the set of problems available to us.

VII. CONCLUSIONS

In this paper, two optimization methods have been proposed to enhance the falsification of CPSs. The first method is the combination of Corners and Uniform-Random, Hybrid Corner-Random. For systems that are falsified on the parameter corners or by randomly choosing parameters, this method manages to falsify in the fewest number of simulations. On the other hand, there are specifications and models for which neither Corners nor UR are efficient, and then optimization-based methods are needed.

The second proposed method is a gradient-free optimization-based method, called *Line-search Falsification*, that optimizes over line segments through a vector of inputs in the n -dimensional parameter space. The method is designed by using insights from the optimization-free approaches in order to combine randomness, corner points, and local search. This method is compared to the NM and SNOBFIT methods. The proposed method has been evaluated on standard benchmark problems. LSF has better performance than all other evaluated methods in this paper.

A conclusion that can be drawn from the combinations of methods and quantitative semantics is that the choice of optimization method appears to matter more than the choice of quantitative semantics. This conclusion is counter intuitive, but the evaluation of the benchmark problems shows that for falsification of CPSs the problem is not only about solving the optimization problem as efficiently as possible but also to explore corner cases and the parameter space.

Testing benefits from running a large number of simulations. If the closed-loop system can be simulated, testing can take

TABLE VIII: Results for the WT system.

Specifications	φ_1^{WT}		φ_2^{WT}		φ_3^{WT}		φ_4^{WT}	
Semantics	Max	Add	Max	Add	Max	Add	Max	Add
Nelder-Mead	100 (1)	100 (1)	100 (1)	100 (1)	100 (1)	100 (1)	100 (109)	100 (79)
SNOBFIT	100 (1)	100 (1)	100 (1)	100 (1)	100 (1)	100 (1)	100 (62)	100 (65)
Line-search Falsification	100 (3)	100 (3)	100 (2)	100 (2)	100 (2)	100 (2)	100 (66)	100 (62)
Corners	100 (3)	100 (3)	100 (3)	100 (3)	100 (3)	100 (3)	100 (16)	100 (16)
Uniform-Random	100 (2)	100 (1)	100 (1)	100 (1)	100 (1)	100 (1)	100 (115)	100 (115)
Hybrid Corner-Random	100 (3)	100 (2)	100 (2)	100 (2)	100 (2)	100 (2)	100 (30)	100 (30)

TABLE IX: Results for the (AT') system.

Specifications	$\varphi_1^{AT'} (T = 20)$		$\varphi_2^{AT'} (T = 30)$		$\varphi_3^{AT'} (T = 40)$		$\varphi_4^{AT'} (T = 10)$		$\varphi_5^{AT'} (T = 4.5)$	
Semantics	Max	Add	Max	Add	Max	Add	Max	Add	Max	Add
Nelder-Mead	100 (138)	100 (156)	85 (264)	95 (365)	35 (315)	65 (557)	100 (33)	100 (14)	100 (141)	90 (323)
SNOBFIT	100 (27)	100 (16)	100 (44)	100 (34)	100 (80)	100 (53)	100 (11)	100 (10)	100 (89)	100 (46)
Line-search Falsification	100 (63)	100 (56)	100 (172)	100 (104)	100 (241)	100 (219)	100 (18)	100 (20)	100 (173)	100 (151)
Corners	100 (1)	100 (1)	100 (1)	100 (1)	100 (1)	100 (1)	100 (2)	100 (2)	100 (11)	100 (11)
Uniform-Random	100 (168)	100 (168)	60 (396)	60 (396)	25 (267)	25 (267)	100 (20)	100 (20)	100 (183)	100 (183)
Hybrid Corner-Random	100 (1)	100 (1)	100 (1)	100 (1)	100 (1)	100 (1)	100 (3)	100 (3)	100 (21)	100 (21)
Specifications	$\varphi_3^{AT'} (T = 5)$		$\varphi_4^{AT'} (T = 1)$		$\varphi_5^{AT'} (T = 2)$		$\varphi_6^{AT'} (T = 1)$		$\varphi_7^{AT'} (T = 2)$	
Semantics	Max	Add	Max	Add	Max	Add	Max	Add	Max	Add
Nelder-Mead	100 (65)	100 (95)	60 (505)	35 (358)	100 (21)	100 (20)	95 (407)	75 (516)	100 (5)	100 (4)
SNOBFIT	100 (41)	100 (24)	100 (170)	60 (328)	100 (17)	100 (18)	100 (54)	100 (59)	100 (3)	100 (4)
Line-search Falsification	100 (81)	100 (61)	60 (376)	75 (307)	100 (29)	100 (24)	100 (37)	100 (127)	100 (6)	100 (7)
Corners	100 (11)	100 (11)	100 (1)	100 (1)	100 (1)	100 (1)	100 (71)	100 (71)	100 (22)	100 (22)
Uniform-Random	100 (79)	100 (79)	70 (407)	70 (407)	100 (24)	100 (24)	95 (212)	95 (212)	100 (5)	100 (5)
Hybrid Corner-Random	100 (21)	100 (21)	100 (1)	100 (1)	100 (1)	100 (1)	100 (120)	100 (120)	100 (10)	100 (10)
Specifications	$\varphi_6^{AT'} (T = 10)$		$\varphi_7^{AT'} (T = 12)$		$\varphi_8^{AT'} (\bar{\omega} = 3000)$		$\varphi_9^{AT'} (\bar{\omega} = 3500)$			
Semantics	Max	Add	Max	Add	Max	Add	Max	Add	Max	Add
Nelder-Mead	50 (722)	45 (483)	100 (236)	100 (215)	20 (766)	75 (383)	100 (13)	100 (12)	30 (439)	90 (375)
SNOBFIT	0 (-)	0 (-)	80 (481)	90 (344)	65 (546)	95 (295)	100 (7)	100 (9)	25 (462)	85 (341)
Line-search Falsification	30 (543)	25 (317)	95 (173)	100 (265)	25 (568)	75 (345)	100 (11)	100 (9)	55 (324)	100 (178)
Corners	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	100 (3)	100 (3)	100 (3)	100 (3)
Uniform-Random	0 (-)	0 (-)	0 (-)	0 (-)	45 (421)	45 (421)	100 (9)	100 (9)	30 (389)	30 (389)
Hybrid Corner-Random	0 (-)	0 (-)	60 (297)	60 (297)	30 (621)	30 (621)	100 (5)	100 (5)	100 (5)	100 (5)

TABLE X: Results for $\Delta - \Sigma$ Modulator system.

Specifications	$\varphi_1^{\Delta-\Sigma} (U \in [-0.35, 0.35])$		$\varphi_2^{\Delta-\Sigma} (U \in [-0.40, 0.40])$		$\varphi_3^{\Delta-\Sigma} (U \in [-0.45, 0.45])$	
Semantics	Max	Add	Max	Add	Max	Add
Nelder-Mead	65 (459)	25 (320)	100 (301)	90 (399)	100 (196)	100 (140)
SNOBFIT	100 (180)	95 (160)	100 (38)	100 (47)	100 (21)	100 (28)
Line-search Falsification	100 (249)	100 (254)	100 (52)	100 (48)	100 (39)	100 (50)
Corners	0 (-)	0 (-)	100 (3)	100 (3)	100 (6)	100 (6)
Uniform-Random	0 (-)	0 (-)	95 (313)	95 (313)	100 (262)	100 (262)
Hybrid Corner-Random	0 (-)	0 (-)	100 (5)	100 (5)	100 (11)	100 (11)

TABLE XI: Results for the SS system.

Specifications	$\varphi_1^{SS} (thresh = 0.7)$		$\varphi_2^{SS} (thresh = 0.8)$		$\varphi_3^{SS} (thresh = 0.9)$	
Semantics	Max	Add	Max	Add	Max	Add
Nelder-Mead	100 (62)	100 (68)	100 (80)	100 (130)	80 (437)	95 (295)
SNOBFIT	100 (22)	100 (22)	100 (128)	100 (63)	80 (304)	75 (208)
Line-search Falsification	100 (66)	100 (47)	100 (60)	100 (119)	100 (121)	100 (192)
Corners	100 (2)	100 (2)	100 (2)	100 (2)	100 (2)	100 (2)
Uniform-Random	100 (42)	100 (42)	100 (84)	100 (84)	90 (261)	90 (261)
Hybrid Corner-Random	100 (3)	100 (3)	100 (3)	100 (3)	100 (3)	100 (3)

Fig. 6: A cactus plot showing the performance of each combination of semantics and optimization-based methods for those examples that using an optimization-based approach to improve the falsification process is needed. The plotted values tell how many successful falsifications (x -axis) were completed in less than 1000 simulations (y -axis, logarithmic scale).

Fig. 7: A cactus plot showing the performance of each combination of quantitative semantics and all evaluated methods in this paper for all examples. The plotted values tell how many successful falsifications (x -axis) were completed in less than 1000 simulations (y -axis, logarithmic scale).

advantage of computing clusters to run the models in parallel, using thousands of computing nodes. The proposed methods can easily be adapted to execute on computing clusters, thus raising the confidence in the correctness of the SUT.

For future work, LSF uses heuristics in its internal process, and it would be interesting to evaluate the efficiency of using different heuristics. In particular, it would be interesting to evaluate the suggested LSF method on several large-scale industrial applications.

ACKNOWLEDGMENT

This work was supported by the Swedish Research Council (VR) project SyTeC VR 2016-06204 and from the Swedish Governmental Agency for Innovation Systems (VINNOVA) under project TESTRON 2015-04893.

The simulation results for all benchmark problems in this paper are performed on resources at High Performance Computing Center North (HPC2N), Umeå University, a Swedish national center for Scientific and Parallel Computing.

REFERENCES

- [1] R. Alur, *Principles of cyber-physical systems*. MIT press, 2015.
- [2] S. Mitra, *Verifying Cyber-Physical Systems: A Path to Safe Autonomy*. MIT Press, 2021.
- [3] E. M. Clarke, E. A. Emerson, and J. Sifakis, "Model checking: algorithmic verification and debugging," *Communications of the ACM*, vol. 52, no. 11, pp. 74–84, 2009.
- [4] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" in *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing*, 1995, pp. 373–382.
- [5] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, and S. Sankaranarayanan, "Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications," in *Lectures on Runtime Verification*. Springer, 2018, pp. 135–175.
- [6] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Systems*, vol. 2, no. 4, pp. 255–299, 1990.
- [7] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, 2004, pp. 152–166.
- [8] J. L. Eddeland, K. Claessen, N. Smallbone, Z. Ramezani, S. Miremadi, and K. Åkesson, "Enhancing temporal logic falsification with specification transformation and valued booleans," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 5247–5260, 2020.
- [9] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia, "Reactive synthesis from signal temporal logic specifications," in *Proceedings of the 18th international conference on hybrid systems: Computation and control*. ACM, 2015, pp. 239–248.
- [10] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262 – 4291, 2009.
- [11] K. Claessen, N. Smallbone, J. Eddeland, Z. Ramezani, and K. Åkesson, "Using valued booleans to find simpler counterexamples in random testing of cyber-physical systems," *IFAC-PapersOnLine*, vol. 51, no. 7, pp. 408–415, 2018.
- [12] S. Amaran, N. Sahinidis, B. Sharda, and S. Bury, "Simulation optimization: A review of algorithms and applications," *Annals of Operations Research*, vol. 240, 05 2016.
- [13] C. Audet and W. Hare, *Derivative-free and blackbox optimization*, ser. Springer Series in Operations Research and Financial Engineering. Springer, 2017.
- [14] R. Hooke and T. A. Jeeves, "'Direct search' solution of numerical and statistical problems," *Journal of the ACM (JACM)*, vol. 8, no. 2, pp. 212–229, 1961.
- [15] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 01 1965.
- [16] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.
- [17] J. Eddeland, J. G. Cepeda, R. Fransen, S. Miremadi, M. Fabian, and K. Åkesson, "Automated mode coverage analysis for cyber-physical systems using hybrid automata," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 9260–9265, 2017, 20th IFAC World Congress.
- [18] L. Rios and N. Sahinidis, "Derivative-free optimization: a review of algorithms and comparison of software implementations," *Journal of Global Optimization*, vol. 56, no. 3, pp. 1247–1293, 2013.
- [19] C. Audet and J. E. Dennis Jr, "Mesh adaptive direct search algorithms for constrained optimization," *SIAM Journal on optimization*, vol. 17, no. 1, pp. 188–217, 2006.
- [20] W. Huyer and A. Neumaier, "Snobfit—stable noisy optimization by branch and fit," *ACM Transactions on Mathematical Software (TOMS)*, vol. 35, no. 2, pp. 1–25, 2008.
- [21] J. L. Eddeland, S. Miremadi, and K. Åkesson, "Evaluating optimization solvers and robust semantics for simulation-based falsification," in *ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems*, 2020.
- [22] A. Arcuri, M. Z. Iqbal, and L. Briand, "Random testing: Theoretical

- results and practical implications,” *IEEE Transactions on Software Engineering*, vol. 38, no. 2, pp. 258–277, 2012.
- [23] A. Groce, G. Holzmann, and R. Joshi, “Randomized differential testing as a prelude to formal verification,” in *29th International Conference on Software Engineering (ICSE’07)*, 2007, pp. 621–631.
 - [24] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball, “Feedback-directed random test generation,” in *29th International Conference on Software Engineering (ICSE’07)*, 2007, pp. 75–84.
 - [25] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. Tse, “Adaptive random testing: The art of test case diversity,” *Journal of Systems and Software*, vol. 83, no. 1, pp. 60–66, 2010.
 - [26] G. Gay, M. Staats, M. Whalen, and M. P. E. Heimdahl, “The risks of coverage-directed test case generation,” *IEEE Transactions on Software Engineering*, vol. 41, no. 8, pp. 803–819, 2015.
 - [27] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *Journal of Machine Learning Research*, vol. 13, no. 10, pp. 281–305, 2012.
 - [28] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, “S-TaLiRo: A tool for temporal logic falsification for hybrid systems,” in *Tools and Algorithms for the Construction and Analysis of Systems*, 2011, pp. 254–257.
 - [29] A. Donzé, “Breach, a toolbox for verification and parameter synthesis of hybrid systems,” in *Computer Aided Verification*, T. Touili, B. Cook, and P. Jackson, Eds. Springer Berlin Heidelberg, 2010, pp. 167–170.
 - [30] G. Ernst, P. Arcaini, A. Donzé, G. Fainekos, L. Mathesen, G. Pedrielli, S. Yaghoubi, Y. Yamagata, and Z. Zhang, “Arch-comp 2019 category report: Falsification,” in *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems*, vol. 61. EasyChair, 2019, pp. 129–140.
 - [31] Z. Ramezani, J. L. Eddeland, K. Claessen, M. Fabian, and K. Åkesson, “Multiple objective functions for falsification of cyber-physical systems,” *IFAC-PapersOnLine*, vol. 53, no. 4, pp. 417–422, 2020.
 - [32] A. Pnueli, “The temporal logic of programs,” in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, 1977, pp. 46–57.
 - [33] B. Hoxha, H. Abbas, and G. Fainekos, “Benchmarks for temporal logic requirements for automotive systems,” in *ARCH@CPSWeek*, 2014.
 - [34] J. Hu, J. Lygeros, and S. Sastry, “Towards a theory of stochastic hybrid systems,” in *Hybrid Systems: Computation and Control*, N. Lynch and B. H. Krogh, Eds. Springer Berlin Heidelberg, 2000, pp. 160–173.
 - [35] A. Dokhanchi, S. Yaghoubi, B. Hoxha, G. Fainekos, G. Ernst, Z. Zhang, P. Arcaini, I. Hasuo, and S. Sedwards, “Arch-comp18 category report: Results on the falsification benchmarks,” in *ARCH@ADHS*, 2018.
 - [36] MathWorks, “Design NARMA-L2 neural controller in simulink,” <https://au.mathworks.com/help/deeplearning/ug/design-narma-l2-neural-controller-in-simulink.html>, 2020, Online: accessed 1 March 2021.
 - [37] G. Frehse, A. Abate, D. Adzkiya, L. Bu, M. Giacobbe, M. S. Mufid, and E. Zaffanella, “Arch-comp18 category report: Hybrid systems with piecewise constant dynamics,” in *ARCH18. 5th International Workshop on Applied Verification of Continuous and Hybrid Systems*, vol. 54. EasyChair, 2018, pp. 1–13.
 - [38] S. Yaghoubi and G. Fainekos, “Gray-box adversarial testing for control systems with machine learning components,” in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, ser. HSCC ’19, 2019, p. 179–184.
 - [39] S. Schuler, F. D. Adegas, and A. Anta, “Hybrid modelling of a wind turbine,” in *ARCH16. 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems*, ser. EPIc Series in Computing, G. Frehse and M. Althoff, Eds., vol. 43. EasyChair, 2017, pp. 18–26.
 - [40] X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda, and K. Butts, “Powertrain control verification benchmark,” in *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC ’14. Association for Computing Machinery, 2014, p. 253–262.
 - [41] T. Dang, A. Donzé, and O. Maler, “Verification of analog and mixed-signal circuits using hybrid system techniques,” in *International Conference on Formal Methods in Computer-Aided Design*. Springer, 2004, pp. 21–36.
 - [42] A. Dokhanchi, A. Zutshi, R. T. Sriniva, S. Sankaranarayanan, and G. Fainekos, “Requirements driven falsification with coverage metrics,” in *Proceedings of the 12th International Conference on Embedded Software*, 2015, pp. 31–40.
 - [43] J. Lagarias, J. Reeds, M. Wright, and P. Wright, “Convergence properties of the Nelder–Mead simplex method in low dimensions,” *SIAM J. on Optimization*, vol. 9, pp. 112–147, 12 1998.