Falsifying Cyber-Physical Systems – a Hybrid Optimization-Free and an Optimization-Based Line-Search Approach

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, which 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. In this paper, a method based on combining random parameters together with considering extreme combinations of parameter values is proposed. Evaluation results on benchmark problems show that this method performs well on many of the problems. Optimization methods are needed when optimization-free methods do not perform well in falsification. The efficiency of the falsification is affected by the optimization methods used to search for inputs that might falsify the specifications. This paper presents a new optimization method for falsification, *Line-search falsification*, where optimization is done over line segments through a vector of inputs in the n -dimensional parameter space. The evaluation results on the benchmark problems show that using this method improves the falsification performance by reducing the number of simulations necessary to falsify a specification.

Index Terms—Cyber-Physical Systems, Testing, Falsification, Hybrid Corner-Random, Line-search Falsification

I. INTRODUCTION

Cyber-physical systems (CPSs) 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 are formal verification and testing. Formal verification of the systems exhibiting a combination of discrete and continuous dynamics is an undecidable problem, in general, as discussed in [1]. Thus, testing is a used approach to the design process of complex systems.

CPSs are often developed using a model-based development (MBD) paradigm [2]. For many industrial applications, there are no mathematical models to analyze, and it is only possible to simulate the system under test (SUT). Thus formal verification is not a viable option.

Testing as a process of software verification and validation can only prove the presence of faults, not prove the absence of faults. Testing consists of a set of activities that tries to detect faults that can be fixed later. If a fault has not been identified during the testing runs, it does not mean a fault did not occur.

Simulation-based approaches [3] are methods that can be used to test CPSs, like the falsification of temporal logic specifications. Falsification of temporal logic properties is an approach to find *counterexamples* for given specifications of CPSs that can be used to increase the confidence that a CPS does fulfill its specifications. The falsification process is performed over a *quantitative semantics* of the specification. Quantitative semantics uses an objective function that models the distance to falsifying the specification. An objective function value is calculated to guide the falsification process towards an input that falsifies the specification. A numerical optimiser searches for an input that minimises the objective function, thus falsifying the specification if possible. An objective function is determined by the definition of quantitative semantics using temporal logic formalisms [4]. Metric Interval Temporal Logic (MITL) [5] and Signal Temporal Logic (STL) [6] are common temporal logic formalisms that have been used in the literature.

For testing purposes, SUTs are considered to be *black box* where only the input-output behavior of the SUT can be observed. S-TaLiRo [7] and Breach [8] are two MATLAB/Simulink toolboxes used for testing and falsification. S-TaLiRo finds counterexamples using MITL properties, while Breach performs the falsification using STL. In this paper, Breach is used.

In [9], the concept of valued booleans (VBools) was introduced to express two different quantitative semantics, *Max* and *Additive* semantics. In the *Additive* semantics, the robustness of each clause in conjunction can affect the final robustness of the whole formula where the robustness is a value assigned in the VBools. Disjunction and temporal operators are defined in terms of conjunction for the *Additive* semantics.

The objective functions and the optimization methods affect the performance of the falsification, as discussed in [10]. When gradients are not available, reliable, or practical to obtain, simulation-based approaches are often used for falsification, hence gradient-free optimization methods are used during falsification. In [11], different gradient-free optimizations are reviewed by a systematic comparison and evaluation of some problems. The results show that there is no single optimization method that outperforms the others. Moreover, all the evaluated optimization methods provide the best solution possible for at

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

least some of the test problems.

A direct-search method is a gradient-free method that uses and compares only function values to directly determine the candidate points which are vectors of input parameters, and it does not require the explicit calculation of derivatives [12]. Thus, this method can be applied to non-smooth optimization problems [13]. Nelder-Mead (NM) [14] is one direct-search method.

Model-based search methods are another class of gradient-free methods that construct and use a surrogate model of the objective function to guide the search process [15]. Bayesian optimization [16] is a sequential model-based approach and a global optimization method for black-box functions [17] that is able to deal with stochastic noise in the function evaluations. Bayesian optimization is best suited for continuous domains and a relatively low number of dimensions. For falsification the number of parameters is typically quite high; industrial problems can have 20 to 100 different optimization variables during the falsification and include both continuous and discrete domains. Thus, Bayesian optimization methods are not directly applicable to falsification problems.

Two optimization methods that are suitable for falsification and were evaluated in [10] are NM and SNOBFIT [18]. Both are gradient-free optimization methods. NM starts with a simplex set of $n + 1$ points where here n is the number of dimensions. In each iteration, the points are sorted from the lowest to highest objective function value. Then, 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 function value, when n new points are generated by NM.

SNOBFIT 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 is a method 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 [18] for a detailed description of the inner working of SNOBFIT.

The three semantics evaluated in [10] were *Max*, *Additive*, and constant quantitative semantics. 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. Surprisingly, SNOBFIT using a constant objective value performed better than NM with *Max* and *Additive* semantics. This interesting and unexpected result leads us to ask: how can SNOBFIT perform better than other optimizations without any information from the objective values? The answer seems to be that when SNOBFIT does not get any information from the constant semantics, it tends to drive the new parameter values towards the corner points, i.e., the combination of extreme values of the allowed parameter ranges of the SUT. This was the primary inspiration for us to in this paper suggest a new optimization

method that features the benefits of SNOBFIT but is less complex. This method will be suggested as a free-optimization method for falsification and it is not a general gradient-free optimization.

Evaluation results on the benchmark problems [19], [20] in this paper show that some specifications are easily falsified by the extreme values of the parameter ranges, i.e., the corner points, or with simple random searches for falsification. In these two cases, the falsification process can be done without using any optimization method. Hence, to combine the advantages of both corner points and random methods, a combination of the corner point and random approach, called Hybrid Corner-Random (HCR), is evaluated in this paper. This method works well for specifications where at least one of the corner points or random points in the parameter space is successful in the falsification of the specifications. This leads us to introduce three algorithms: (i) Corners, (ii) Uniform-Random, (iii) Hybrid Corner-Random. In our evaluation, the latter method surprisingly outperforms the NM method on the benchmark problems, irrespectively of which semantics is used.

For specifications and examples where HCR does not work, an optimization-based approach for falsification is better. In this paper, we propose an alternative optimization algorithm, called *Line-search Falsification* (LSF), for falsification that has a performance on par with, or better than, SNOBFIT but with a less complex design. The algorithm is inspired by the crawling procedure of NM for local optimization but can also get out of local optima and continue the optimization from a new but related point. Evaluation results on the benchmark examples in this paper show that the LSF method performs better than NM and SNOBFIT.

The main contributions of this paper are:

- The Hybrid Corner-Random method which uses a combination of the corner points and random methods.
- *Line-search Falsification*, a gradient-free optimization method.
- An evaluation of the proposed methods on the ARCH benchmark [19] and additional benchmarks problems [20].

This paper is organized as follows: Section II introduces the quantitative semantics and different ways to define the objective functions used by the falsification process. Section III introduces the requirements and definitions. Section IV proposes the suggested HCR method. Section V introduces the LSF method. Section VI introduces the evaluated benchmark problems in this paper. Section VII evaluates the performance of the suggested methods on the chosen benchmark problems. Finally, Section VIII summarizes the contributions.

II. QUANTITATIVE SEMANTICS AND OBJECTIVE FUNCTIONS

In this paper, STL [6] is used to model the specifications. The syntax of STL is defined as follows:

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

where the predicate μ is $\mu \equiv \mu(s) > 0$ and 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 s at the discrete-time instant k is defined as:

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

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. A Valued Boolean (VBool) [9] (v, y) 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. However, in this paper, we use the convention that y is negative when v is false, and positive otherwise.

A. 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>
$(\top, y) \wedge (\top, z) =$	$(\top, \min(y, z))$	$(\top, \frac{1}{\frac{1}{y} + \frac{1}{z}})$
$(\top, y) \wedge (\perp, z) =$	$(\perp, z)$	
$(\perp, y) \wedge (\top, z) =$	$(\perp, y)$	
$(\perp, y) \wedge (\perp, z) =$	$(\perp, \max(y, z))$	$(\perp, y + z)$

Using the de Morgan laws, the *or* operator can be defined in terms of *and*, as: $(v_y, y) \vee (v_z, z) = \neg_v(\neg_v(v_y, y) \wedge \neg_v(v_z, z))$, where VBool negation is defined as $\neg_v(v_y, y) = (\neg v_y, -y)$.

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

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

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

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

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

The quantitative semantics are used to assign an objective function value given a set of n parameters. The formal relations between the satisfaction of a specification φ and value of the function f are [4]:

$$\begin{aligned}
f(\varphi, x, t) > 0 &\implies (x, t) \models \varphi \\
f(\varphi, x, t) < 0 &\implies (x, t) \not\models \varphi \\
(x, t) \models \varphi &\implies f(\varphi, x, t) \geq 0 \\
(x, t) \not\models \varphi &\implies f(\varphi, x, t) \leq 0
\end{aligned}$$

where $x(t)$ is a signal, and $f(x)$, for the function $f: \mathbb{R}^n \rightarrow \mathbb{R}$, refers to the objective function value.

III. REQUIREMENTS AND DEFINITIONS

For the falsification process of a SUT, each input parameter, x_i , to the system is selected within a finite range, (l_i, u_i) , with a lower l_i and upper bound u_i , where $l_i < u_i$. Since we have n input parameters, there is a lower bound $l = (l_1, l_2, \dots, l_n)^T$ and an upper bound $u = (u_1, u_2, \dots, u_n)^T$. A vector of inputs $x = (x_1, x_2, \dots, x_n)^T$ are defined within the range (l, u) .

To clarify the meaning of a corner point, an example is given here. Assume a system with three inputs x_1, x_2, x_3 . Each input is defined in a specific range, $x_1 \in [0, 10]$, $x_2 \in [-1, 2]$, and $x_3 \in [1, 50]$. Therefore, we have a vector of inputs in three dimensions $x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$ with lower bound $l = \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix}$ and upper bound $u = \begin{pmatrix} 10 \\ 2 \\ 50 \end{pmatrix}$.

In this example, there are $2^3 = 8$ corner points from the combinations of lower and upper bounds. These are: $\begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix}$,

$$\begin{pmatrix} 10 \\ 2 \\ 50 \end{pmatrix}, \begin{pmatrix} 10 \\ -1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \\ 50 \end{pmatrix}, \begin{pmatrix} 0 \\ 2 \\ 50 \end{pmatrix}, \begin{pmatrix} 10 \\ -1 \\ 50 \end{pmatrix}, \begin{pmatrix} 10 \\ 2 \\ 1 \end{pmatrix}.$$

In this paper, it is assumed that we can only simulate the system and do not have access to the internals of the system, thus a gradient-based optimization method cannot be used. Instead, we have to use a gradient-free optimization method.

IV. HYBRID CORNER-RANDOM METHOD

Falsification can be done with or without using an optimization method. Optimization-based approaches are more complex than optimization-free methods, since it is necessary to associate objective function values to simulation traces together with a strategy to choose new parameter combinations to explore based on the objective function values. To motivate the extra complexity in optimization-based approaches we start by investigating how far we can get by using a simple strategy based on using random-search together with exploring extreme parameter values.

So in this section, we introduce the HCR falsification method, an optimization-free 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.

We have observed that a successful strategy in many cases is to focus on the corner points. However, if more simulations are allowed a random strategy will eventually falsify more specifications than a corner-based strategy. A corner-based strategy performs well on some of the models and specifications, while a random strategy performs better on some other problems. We thus propose combining the two strategies. The HCR falsification method is presented in Algorithm 1.

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  $x$ ).

```

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

A. Output

The output of Algorithm 1 is the tuple $\langle \text{Falsified/Not Falsified}, x \rangle$, where the first item tells if the specification was falsified or not and the second parameter 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 last evaluated point.

B. Hybrid Corner-Random Process

Algorithm 1 starts with a corner point, x , in 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 number of system inputs. 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.

Although this method, in general, performs better than the corners and random methods used individually, as will be shown in Section VII, for many of the benchmark problems, the method could not falsify in a reasonable number of iterations. Hence, for the examples where using Corners, Uniform-Random (UR) and HCR is not successful, an optimization method can be used. In the next section, a new method, LSF, will be introduced.

V. LINE-SEARCH FALSIFICATION

LSF, proposed in this paper, is a gradient-free optimization-based approach that combines a local search with the ability to explore new areas of the parameter space in case the local optimization is unable to falsify the specification. This method needs to initially pick a random line and start optimizing over that line to find a local minimum on that line. The method is presented in Algorithm 2, and has been implemented for the falsification of CPSs using Breach [8]. In the next sections, the method will be compared with two optimization methods, NM and SNOBFIT, and also HCR.

A. Output

The output of LSF is the tuple $\langle \text{Falsified/Not Falsified}, x \rangle$, where the first item tells if the specification was falsified or not and the second parameter 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, $f(x) \geq 0$.

B. Line-search Falsification Process

LSF needs three points during the optimization process, and a new point is generated in each iteration. All points have to be inside the parameter range, or on a lower or upper bound.

The LSF method is presented in Algorithm 2. 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. Also, note that this point could instead be chosen randomly within (l, u) . The SUT is then simulated with x as input 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.

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

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:  ( $x_L, x_M, x_R$ ) = 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  $f(x)$ 

```

(x_L, x_M, x_R) = **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  $x$ 

```

2) **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,

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

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

```

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 ( $\|x_L - x_M\| \geq \|x_M - x_R\|$ ) 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  $x$ 

```

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, $z_{\lceil \frac{n}{2} \rceil}^+$ 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. 1 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:

Fig. 1: 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 $x = x_M$.

Fig. 1a 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. 1b 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 (u_1, u_2) .

To get a clear understanding of how to choose a line according to Option 4, figures 1c and 1d are given. In Fig. 1c 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 , but it can also be two line segments that connect x to x_L and x to x_R as is shown in Fig. 1d 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 (u_1, u_2) .

3) **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 VII.

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

4) **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} in 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 main loop of the LSF approach. 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 searching for new points.

In this algorithm, we work with the chosen line and three points generated in SelectPoints function inside FalsifyLine. A iterations_without_improvement is used to count the number of iterations that we stay with a line inside FalsifyLine until the 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:

- 1) **The point x_L has the lowest objective value among the three points, lines 35-40:** Compute the middle point $x_{new} = \frac{x_L + x_M}{2}$. Now check to see whether x_{new} has a lower objective function value than x_L or not, if it does thus update $x = x_{new}$. Consider $x_R = x_M$ and $x_M = x_{new}$.
- 2) **The point x_R has the lowest objective value among the three points, lines 41-46:** Compute the middle point $x_{new} = \frac{x_M + x_R}{2}$. Now check to see whether x_{new} has a lower objective function value than x_R or not, if it does thus update $x = x_{new}$. Consider $x_L = x_M$ and $x_M = x_{new}$.
- 3) **The point x_M has the lowest objective value among the three points, lines 47-62:** There are two cases depending on which interval (x_L, x_M) or (x_M, x_R) is larger.
 - if the interval (x_L, x_M) is larger than (x_M, x_R) , Euclidean distance is calculated in line 48, compute the middle point $x_{new} = \frac{x_L + x_M}{2}$. Now check to see if the $f(x_{new}) < f(x_M)$ thus $x = x_{new}$, Consider $x_R = x_M$ and $x_M = x_{new}$, lines 48-52. Otherwise if $f(x_{new}) > f(x_M)$, thus consider $x_L = x_{new}$, line 54.
 - if the interval (x_M, x_R) is larger than (x_L, x_M) , compute the middle point $x_{new} = \frac{x_M + x_R}{2}$, lines 57. Now check to see if the $f(x_{new}) < f(x_M)$ thus $x = x_{new}$, consider $x_L = x_M$ and $x_M = x_{new}$. Otherwise if $f(x_{new}) > f(x_M)$, thus consider $x_R = x_{new}$, line 62.

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, the `iterations_without_improvement` resets to zero, line 67. Otherwise, `iterations_without_improvement` is increased by 1, line 69. 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 in 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 in line 8.

5) **Heuristic H_1** : In function Heuristic H_1 , we check the objective function of x_{old} with x . If $f(x_{old}) < f(x)$, it means that `FalsifyLine` could not give a better point with less objective function than x_{old} . Hence, to force the algorithm to never return the same x as was given as to `FalsifyLine`, we consider x be equal to the second point with lowest objective function value of the last call to `FalsifyLine`. This heuristic helps the algorithm to work better by helping it to avoid getting stuck in a local minimum. For this purpose, first we save the $x_{old} = x$ in line 11. This procedure is presented in Heuristic H_1 . It should be mentioned here that we did not present Heuristic H_1 earlier because it can not be applied until the `FalsifyLine` function has been called at least once.

This algorithm works until a falsified point is found, i.e., a point with a negative objective function value or the `curr_simulation` reaches `max_simulations`. If the falsified point is not found, the algorithm returns the point with the lowest objective function value.

The next section describes the benchmark problems that will be used to evaluate the suggested methods.

VI. BENCHMARK PROBLEMS

In this paper, we consider the simulation-based falsification for all benchmark examples of the ARCH workshop [19] and some additional benchmark problems [20]. The paper [19] 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.

A. ARCH Benchmark Problems

For the ARCH Benchmark problems [19], 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

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}	$\Box_{[1,37]}(Pos - Ref > 0.005 + 0.03 Ref \Rightarrow \Diamond_{[0,2]} \Box_{[0,1]} (-0.005 + 0.03 Ref \leq Pos - Ref))$
φ_2^{NN}	$\Box_{[1,37]}(Pos - Ref > 0.005 + 0.04 Ref \Rightarrow \Diamond_{[0,2]} \Box_{[0,1]} (-0.005 + 0.04 Ref \leq Pos - Ref))$
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}	$\Box_{[30,35]} (87 \leq pressure \wedge pressure \leq 87.5)$
$\varphi_1^{AT'}$	$\Diamond_{[0,T]}(\omega \geq 2000)$
$\varphi_2^{AT'}$	$\Box_{[0,T]}(\omega \leq 3500 \vee \omega \geq 4500)$
$\varphi_3^{AT'}$	$\Box_{[0,T]}(\neg(gear == 4))$
$\varphi_4^{AT'}$	$\Diamond(\Box_{[0,T]}(gear == 3))$
$\varphi_5^{AT'}$	$\bigwedge_{i=1,\dots,4} \Box((\neg(gear == i) \wedge \Diamond_{[0,e]}(gear == i)) \Rightarrow (\Box_{[e,T+e]}(gear == i)))$
$\varphi_6^{AT'}$	$\Box_{[0,T]}(v \leq 85) \vee \Diamond(\omega \geq 4500)$
$\varphi_7^{AT'}$	$\neg((\Box_{[0,1]} gear == 1) \wedge (\Box_{[2,4]} gear == 2) \wedge (\Box_{[5,7]} gear == 3) \wedge (\Box_{[8,10]} gear == 3) \wedge (\Box_{[12,15]} gear == 2))$
$\varphi_8^{AT'}$	$\Box_{[0,20]} ((gear == 4 \wedge throttle > 45 \wedge throttle < 50) \Rightarrow \omega < \bar{\omega})$
$\varphi^{\Delta-\Sigma}$	$\Box(\bigwedge_{i=1}^3 (-1 \leq x_i \wedge x_i \leq 1))$
φ^{SS}	$\Box(y \geq 0)$

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 [19].

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) [21]. Two different control inputs are considered for this example:

- Instance 1: $0 \leq throttle \leq 100$ and $0 \leq brake \leq 325$, 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 [22]. 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 [23]. 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 [24] 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 [25]. 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 [26]. 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 [27] 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 [28] for details. This example has different specifications from the ARCH example presented in Section VI-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 [29] 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 [30].

VII. EXPERIMENTAL SETUP AND RESULTS

In this section, the Corners and UR methods are compared with the HCR approach. The LSF method are compared with NM and SNOBFIT and also, 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 $\max_simulations = 1000$. There are 20 falsification runs for each method and objective function to account for most algorithms' random nature.

Using 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* [31] 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 constant and equal to 3.

6) *Tables Setup*: The first column denotes the specifications, semantics, optimization method, and free-optimization methods;

the next columns show the *Max*, and *Additive* semantics. For each method and 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 objective function; thus, the success rate will be a multiple of 5%. The second value, inside parentheses, is the average number of simulations *per successful falsification*. The presented results in the tables for LSF are related to Option 4.

A. Hybrid Corner-Random vs. Corners and Uniform-Random

First, we compare the Corners, UR, and HCR methods are given. According to tables II-XI, for many specifications, at least one of Corners and UR work well. Now, we go through all examples in detail.

The AT example, Instance 1 Table II, is one of the examples where for all specifications, one of Corners and UR, or even both successfully falsify the specification with rate 100%. φ_1^{AT} is falsified with Corners. Both Corners and UR work well for φ_2^{AT} . For the other specifications, UR works well. Thus, if we only have one of Corners or UR, we cannot get a falsified point for all specifications, but the HCR method is fully successful (100%), for all specifications of this example. As can be seen, for Instance 2 of AT example, the specifications φ_1^{AT} , φ_7^{AT} , φ_8^{AT} , φ_9^{AT} are hard to falsify using Corners and UR. Thus, the HCR method does not perform as well. In this situation, optimization methods are needed, as will be discussed later. On the other hand, the specifications φ_i^{AT} where $i = 2, \dots, 6$ are falsified by both Corners and UR. Thus, HCR is also good for these specifications.

In the CC example for both Instances 1 and 2, Table III, all specifications (except φ_4^{CC} where an optimization method is needed) can be falsified easily. Also, for φ_5^{CC} of Instance 1, UR beats Corners, thus HCR improves on Corners.

In the AFC example in Table IV, the Corners method is the best method for both specifications of this example requiring fewer simulations than to UR, Table IV. HCR helps here to improve UR and beats UR, improving falsification.

In the NN example, Table V, Instance 1 for specification φ_1^{NN} , HCR improves the falsification and beats the Corners method. In Instance 2 of this specification, both Corners and UR work well, thus HCR performs well. On the other hand, for Instance 1 of φ_2^{NN} , none of the optimization-free methods can falsify. For Instance 2 of this specification, Corners is successful thus HCR works well.

For the F16 example, Table VI, using an optimization-based approach improves the falsification process. UR finds only one out of 20 runs of testing; in this case, it is not a good method and does not help HCR to be more efficient. In this example, the number of corners is 8 so that most iterations of HCR are done using random points.

The SC example, Table VII, is one example where it does not matter which semantics or optimization method is used, since none of them can falsify the specification. [26] used a Simulated Annealing global search in combination with an optimal control based local search on the infinite-dimensional input space to successfully falsify this specification.

The WT example, Table VIII, is falsified easily by both Corners and UR. Compared to the first three specifications,

φ_4^{WT} is hardest and needed more simulations, but still, Corners, UR, and HCR are all successful in falsifying the specification.

For the AT' example, Table IX, all specification $\varphi_i^{AT'}$ for $i = 1, \dots, 5, 8$, could be falsified using Corners while UR failed for $\varphi_1^{AT'}$ ($T = 30$), ($T = 40$) and $\varphi_4^{AT'}$. Thus, HCR is successful here. On the other hand, UR performs better than Corners and HCR in $\varphi_6^{AT'}$ ($T = 12$) and $\varphi_7^{AT'}$. HCR is more successful than Corners here. Only for $\varphi_6^{AT'}$ for both ($T = 10$), ($T = 12$) and $\varphi_7^{AT'}$, it is better to use an optimization-based approach.

For the modulator example, Table X, both Corners and UR work well for the specifications $\varphi_1^{\Delta-\Sigma}$ ($U \in [-0.40, 0.40]$) and ($U \in [-0.45, 0.45]$) (Corners is much better). Hence, HCR is fully successful for them. On the other hand, for the specification $\varphi_1^{\Delta-\Sigma}$ ($U \in [-0.35, 0.35]$), using an optimization-based approach improves the falsification process.

SS, Table XI, is falsified on the corners, so HCR performs well.

For a summarized comparison between Corners, UR, and HCR, a cactus plot is shown in Fig 2. The results presented in this plot relate to all examples, even those where optimization methods were required to be able to falsify. As can be seen in Fig 2, while UR is more successful than Corners, for those cases that are falsified on the corner points, Corners falsifies much faster and with less number of simulations. Thus, HCR manages to falsify more examples than either of the other two.

B. Hybrid Corner-Random Vs. Optimization Methods

In this section, we discuss the advantages of HCR, and why we should first try to falsify our examples with HCR before using any optimization-based method. We also compare HCR with NM, SNOBFIT and LSF.

For φ_2^{AT} , 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. Similarly, as can be seen in Table III, Instance 2, φ_2^{CC} ; Table IX, $\varphi_1^{AT'}$ ($T = 30$) and ($T = 40$), $\varphi_3^{AT'}$ ($T = 4.5$), $\varphi_4^{AT'}$ ($T = 1$), $\varphi_8^{AT'}$ ($\bar{\omega} = 3000$) and ($\bar{\omega} = 3500$); Table X, $\varphi_1^{\Delta-\Sigma}$ ($U \in [-0.40, 0.40]$) and ($U \in [-0.45, 0.45]$); and all specifications in Table XI, NM, SNOBFIT and LSF need more simulations than Corners and HCR.

To compare Corners, UR, and HCR with NM, SNOBFIT and LSF, a cactus plot is shown in Fig 3 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.

As can be seen in Fig. 2, HCR beats all methods requires fewer simulations than all other methods. As a result, using HCR is a viable method for falsifying those examples where the optimization-based methods do not perform well. Since HCR can falsify many examples with relatively few simulations, and hence take a short time to do this, it is useful to first try HCR on any example, and only if it does not manage to falsify the property, an optimization-based approach can be used to improve the falsification process.

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)		100 (2)		0 (-)		0 (-)		0 (-)	
Uniform-Random	0 (-)		100 (10)		100 (13)		100 (13)		100 (11)	
Hybrid Corner-Random	100 (5)		100 (3)		100 (27)		100 (25)		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 (47)	100 (17)	100 (16)
Line-search Falsification	100 (63)	100 (112)	100 (13)	100 (24)	100 (26)	100 (28)	100 (22)	100 (12)	100 (19)	100 (17)
Corners	0 (-)		0 (-)		0 (-)		0 (-)		0 (-)	
Uniform-Random	100 (59)		100 (65)		100 (96)		100 (31)		100 (11)	
Hybrid Corner-Random	100 (117)		100 (130)		100 (178)		100 (61)		100 (23)	

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 (1)		100 (3)		0 (-)		0 (-)	
Uniform-Random	100 (8)		100 (5)		100 (17)		0 (-)		100 (18)	
Hybrid Corner-Random	100 (5)		100 (1)		100 (5)		0 (-)		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 (1)		100 (3)		0 (-)		100 (21)	
Uniform-Random	100 (67)		100 (157)		100 (24)		0 (-)		100 (96)	
Hybrid Corner-Random	100 (5)		100 (1)		100 (5)		0 (-)		100 (38)	

TABLE IV: Results for the AFC system

Specifications	φ_1^{AFC}		φ_2^{AFC}	
Semantics	Max	Add	Max	Add
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)	

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

Instance 1				
Specifications	φ_1^{NN}		φ_2^{NN}	
Semantics	Max	Add	Max	Add
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}	
Semantics	Max	Add	Max	Add
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)	

C. Line-search Falsification using the four options

Section V introduces four different options for LSF to pick a line. A cactus plot comparing these options is given in Fig 4. Option 1, where we work with the lines that cut off at the

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 (-)	

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 Figure 4, 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

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 (16)	
Uniform-Random	100 (2)		100 (1)		100 (1)		100 (115)	
Hybrid Corner-Random	100 (3)		100 (2)		100 (2)		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 (2)		100 (11)	
Uniform-Random	100 (168)		60 (396)		25 (267)		100 (20)		100 (183)	
Hybrid Corner-Random	100 (1)		100 (1)		100 (1)		100 (3)		100 (21)	
Specifications	$\varphi_6^{AT'} (T = 5)$		$\varphi_7^{AT'} (T = 1)$		$\varphi_8^{AT'} (T = 2)$		$\varphi_9^{AT'} (T = 1)$		$\varphi_{10}^{AT'} (T = 2)$	
Semantics	Max	Add	Max	Add	Max	Add	Max	Add	Max	Add
Nelder-Mead	100 (65)	100 (95)	60 (305)	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 (1)		100 (1)		100 (71)		100 (22)	
Uniform-Random	100 (79)		70 (407)		100 (24)		95 (212)		100 (5)	
Hybrid Corner-Random	100 (21)		100 (1)		100 (1)		100 (120)		100 (10)	
Specifications	$\varphi_{11}^{AT'} (T = 10)$		$\varphi_{12}^{AT'} (T = 12)$		$\varphi_{13}^{AT'} (\bar{\omega} = 3000)$		$\varphi_{14}^{AT'} (\bar{\omega} = 3500)$			
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 (-)		100 (3)		100 (3)	
Uniform-Random	0 (-)		90 (328)		45 (421)		100 (9)		30 (389)	
Hybrid Corner-Random	0 (-)		60 (297)		30 (621)		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 (-)		100 (3)		100 (6)	
Uniform-Random	0 (-)		95 (313)		100 (262)	
Hybrid Corner-Random	0 (-)		100 (5)		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)	
Uniform-Random	100 (42)		100 (84)		90 (261)	
Hybrid Corner-Random	100 (3)		100 (3)		100 (3)	

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.

D. 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. Most of the available benchmark problems in this paper are easily falsified using the Corners or UR methods. It is only for a subset of the specifications and models that optimization methods are necessary. These are specifications 1 and 7-9 of the AT example for 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 that for a specific optimization method and semantics to perform better.

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 this paper [19]. 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

Fig. 2: 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. 3: 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).

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 Figure 5. 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*.

VIII. 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. This method is compared to the NM and SNOBFIT methods. The proposed method has been evaluated on standard benchmark problems. An aggregated comparison among all methods presented and discussed in this paper for all examples is shown in Figure 6. As can be seen, HCR not only manages to falsify more examples than Corners and UR but also has a better performance than NM

Fig. 4: 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).

Fig. 5: 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).

using both *Max* and *Additive* semantics. LSF, compared to the other evaluated optimization-free and optimization-based approaches, falsifies the most examples.

A key strength of using model-based development methods is that the models can be used for various purposes, including testing. Testing benefits from running a large number of simulations. If only software models are needed to run the simulations, testing can take advantage of computing clusters to run the models in parallel, using thousands of computing nodes. Since, in industry, there is the possibility to run the different methods in parallel using a computing cluster, many more examples in a shorter time can be run using optimization-free methods, thus raising the confidence in the correctness of the SUT.

For future work, it would be interesting to evaluate the suggested LSF method on industrial applications as well as

extending the number of benchmark problems.

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] 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.

Fig. 6: 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).

- [2] P. Mosterman, "Model-based design of embedded systems," in *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*, 2007, pp. 3–3.
- [3] 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.
- [4] 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.
- [5] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Systems*, vol. 2, no. 4, pp. 255–299, 1990.
- [6] 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.
- [7] 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.
- [8] 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. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 167–170.
- [9] 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," vol. 51, no. 7, pp. 4-08 – 415, 2018, 14th IFAC Workshop on Discrete Event Systems WODES 2018.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] J. Larson, M. Menickelly, and S. M. Wild, "Derivative-free optimization methods," *arXiv preprint arXiv:1904.11585*, 2019.
- [14] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 01 1965.
- [15] A. R. Conn, K. Scheinberg, and L. N. Vicente, *Introduction to derivative-free optimization*. SIAM, 2009.
- [16] J. Mockus, *Bayesian approach to global optimization: theory and applications*. Springer Science & Business Media, 2012, vol. 37.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] Z. Ramezani, J. L. Eddeland, K. Claessen, M. Fabian, and K. Åkesson, "Multiple objective functions for falsification of cyber-physical systems," in *Accepted to 2020 Workshop on Discrete Event Systems*, 2020.
- [21] B. Hoxha, H. Abbas, and G. Fainekos, "Benchmarks for temporal logic requirements for automotive systems," in *ARCH@CPSWeek*, 2014.
- [22] 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. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 160–173.
- [23] 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.
- [24] 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.
- [25] 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.
- [26] S. Yaghoubi and G. Fainekos, "Gray-box adversarial testing for control systems with machine learning component," *CoRR*, vol. abs/1812.11958, 2018.
- [27] 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.
- [28] B. Hoxha, H. Abbas, and G. Fainekos, "Benchmarks for temporal logic requirements for automotive systems," *Proc. of Applied Verification for Continuous and Hybrid Systems*, 2014.
- [29] 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.
- [30] 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.
- [31] 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.