Multi-Requirement Testing Using Focused Falsification

Citation for the original published paper (version of record):
Lidén Eddeland, J., Donzé, A., Åkesson, K. (2022). Multi-Requirement Testing Using Focused Falsification. HSCC 2022 - Proceedings of the 25th ACM International Conference on Hybrid Systems: Computation and Control, Part of CPS-IoT Week 2022.
http://dx.doi.org/10.1145/3501710.3519521

(article starts on next page)

Multi-Requirement Testing Using Focused Falsification

Johan Lidén Eddeland
Volvo Car Corporation
Gothenburg, Sweden
johan.eddeland@volvocars.com

Alexandre Donzé
Decyphir SAS
Moirans, France
alex@decyphir.com

Knut Åkesson
Chalmers University of Technology
Gothenburg, Sweden
knut@chalmers.se

ABSTRACT

Testing of Cyber-Physical Systems (CPS) deals with the problem of finding input traces to the systems such that given requirements do not hold. Requirements can be formalized in many different ways; in this work requirements are modeled using Signal Temporal Logic (STL) for which a quantitative measure, or *robustness value*, can be computed given a requirement together with input and output traces. This value is a measure of how far away the requirement is from not holding and is used to guide falsification procedures for deciding on new input traces to simulate one after the other. When the system under test has multiple requirements, standard approaches are to falsify them one-by-one, or as a conjunction of all requirements, but these approaches do not scale well for industrial-sized problems. In this work we consider testing of systems with multiple requirements by proposing focused multi-requirement falsification. This is a multi-stage approach where the solver tries to sequentially falsify the requirements one-by-one, but for every simulation also evaluate the robustness value for all requirements. After one requirement has been focused long enough, the next requirement to focus is selected by considering the robustness values and trajectory history calculated thus far. Each falsification attempt makes use of a prior sensitivity analysis, which for each requirement estimates the parameters that are unlikely to affect the robustness value, in order to reduce the number of parameters that are used by the optimization solver. The proposed approach is evaluated on a public benchmark example containing a large number of requirements, and includes a comparison of the proposed algorithm against a new suggested baseline method.

CCS CONCEPTS

• **Computer systems organization → Embedded and cyber-physical systems**; • **Computing methodologies →** *Modeling and simulation.*

KEYWORDS

CPS, Testing, Formal Requirements, Falsification

ACM Reference Format:
Johan Lidén Eddeland, Alexandre Donzé, and Knut Åkesson. 2022. Multi-Requirement Testing Using Focused Falsification. In *25th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '22), May 4–6, 2022, Milan, Italy*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3501710.3519521

1 INTRODUCTION

Requirement-based testing of Cyber-Physical Systems (CPSs) is a testing method supported by monitoring of formally specified requirements. In contrast to other formal methods such as reachability analysis of hybrid systems, which is notoriously undecidable [14], such testing is always possible in practice for industrial-scale systems, with the drawback that in general it cannot prove the absence of faults. It can however find them efficiently through, e.g., falsification which is the most common requirement-based testing approach and consists of trying to find an input trace to the system that falsifies a given requirement. Requirements are typically written in temporal logic, and in this paper, specifically, *Signal Temporal Logic* (STL) [18]. This leads to the possibility of calculating the *quantitative measure*, or robustness value [9], of the requirement for a given trace. The robustness value of an STL requirement is a measure of the distance to falsify the requirement, and it is used to guide the test case generation in the falsification process. A positive robustness value means the requirement is not falsified, while a negative robustness value means that it is falsified. The magnitude of the robustness value indicates how far away the requirement is from the point of falsification and can be used as an objective function for some minimization algorithm to guide the system toward a requirement violation.

A lot of work has been done in recent years to propose new algorithms or improve on existing ones for falsification and an academic informal friendly competition has been going on since 2019 [12] to compare current approaches. This work is part of the same effort, but also aims at embedding it into a more general formulation suitable for its application in an industrial context of complex CPS design. We look more generally at multi-requirement testing, which arises inevitably at a certain phase during the development of a complex system, when parts of the design have been implemented and already tested, while others are still missing or at a prototype stage. At this point, testers are given not one but several requirements corresponding to these different parts, and while the primary goal is still to detect faults as quickly as possible, a secondary goal is to perform regression testing and ensure a good testing coverage for the parts that are considered sufficiently mature. Since simulation budget, or more generally the computational testing budget, is always limited, these different goals have to be fulfilled as much as possible in parallel. In this work we propose a novel approach considering a multi-requirement falsification problem. We present specific difficulties related to such a formulation, in particular, the high dimensionality of the search space, and the "competition" between and within requirements leading to the so-called masking problem and a difficult multi-objective formulation. We propose several improvements on falsification and a multistage approach based on sequentially solving several simplified falsification problems.

1.1 Related work

The purpose of this paper is to further adapt the falsification process to industrial-scale problems. Two tools commonly used for falsification are S-TaLiRo [3] and Breach [8]. Recent works in the area of falsification have explored falsification with constrained signal spaces using Timed Automata [4], improved falsification using multiple objective functions [22], and falsification of CPSs using deep reinforcement learning [2]. For reviews on testing of CPSs, see [1, 25].

A recent work [19] investigated a similar problem formulation as the one presented in this paper, namely falsification of CPSs with multiple conjunctive requirements. The work uses Bayesian optimization by modeling each separate sub-requirement with a Gaussian Process in order to avoid the *masking problem* commonly present in robustness-based falsification. In this paper, we use additive semantics of STL [5] and local rescaling of predicate robustness values to diminish the masking problem in all levels of the multiple requirement structure, and we also use heuristics to reduce the dimensionality of the falsification problem by focusing on different sub-requirements at a time. Another recent paper [24] investigated the scaling problem of falsification, where certain signals affect the robustness of requirements more than others. A previous work [23] tackled the scaling problem by using a Multi-Armed Bandit problem formulation of the falsification problem. In another paper [7], the authors perform a sequence of optimizations using stochastic local search over increasing numbers of parameters, which is similar to how we reduce the dimensionality of each falsification problem on focused sub-requirements. In [4] a high level algorithm that can switch between different optimization heuristics (or metaheuristics) in order to balance exploitation and exploration is introduced. This balance is even more important and delicate to achieve in our case where multiple requirements are involved.

Evaluating falsification algorithms on benchmarks is an important part in creating credible research. A benchmark used often in the research community is the one used by the ARCH workshop friendly competition on falsification [12]. Other benchmark models commonly used are a fuel-control system [17] and a $\Delta - \Sigma$ modulator [6]. In this paper, we use a variant on one of the ARCH benchmark models, using a larger set of requirements [10][1]. The main reason for using a single benchmark model, as opposed to many recent works which typically choose a set of benchmark models to evaluate on, is because the model is the only available falsification benchmark with a large enough set of requirements in order to properly capture the multi-requirement behaviour of industrial-scale systems. Even though there is only one model to be simulated, we argue that the rich set of requirements makes it an interesting use case to consider.

1.2 Contributions

The contributions of this work are

- a problem formulation of multi-requirement testing motivated by real-world industrial systems,
- an addition to the falsification procedure to reduce the masking problem by using local predicate normalization,
- a method, using sensitivity analysis, for selecting influential parameters in order to reduce the dimensionality of the falsification optimization problem,
- an algorithm (referred to as the MRF algorithm) using the above techniques for solving multi-requirement falsification problems by sequentially selecting and focusing on some requirements while still doing random exploration for the other parameters and requirements,
- an evaluation on a public benchmark example containing a large number of requirements obtained by extensively comparing the proposed algorithm against a proposed baseline approach using corners and uniform random sampling, and
- finally, the observation that even though our method overall performs better, the corners and random-based technique also proved to be surprisingly efficient for multi-requirement testing.

2 PRELIMINARIES

In this section we describe the formalism chosen to describe the contribution of this work. We introduce a system, its inputs and outputs signals, the parametrization of its inputs, and what we call a trace. We define requirements, the multi-requirement problem we consider, signal temporal logic, the quantitative semantics we use and the new local predicate normalization.

2.1 Signal Traces

We use discrete time signals of the form $y : k \longrightarrow y[k]$ mapping an integer k to some real value $y[k]$. Bold font symbols represent multidimensional signals, e.g., $\mathbf{y}[k] = (y_1[k], y_2[k], \ldots, y_s[k])$. Throughout this paper we assume we are given a deterministic system S mapping *input signals* $\mathbf{u} = (u_1, \ldots, u_r)$ to *output signals* $\mathbf{y} = (y_1, y_2, \ldots, y_s)$. Furthermore we assume we are given a parameterization of inputs, which maps a finite real parameter vector $\mathbf{p} = (p_1, \ldots, p_m)$ to a signal $\mathbf{u}(\mathbf{p})$. This parameterization can be partitioned into r independent parameterizations $\mathbf{p} = (\mathbf{p}_1, \ldots, \mathbf{p}_r)$ where $\mathbf{p}_i = (p_{i,1}, \ldots, p_{i,m_i})$ for each input signal u_i, $i = 1, \ldots, r$. We have $\sum m_i = m$ and $m_i \geq 1$ for all i. The mapping from parameters to output signals is summarized in Equation (1).

$$\begin{pmatrix} p_{1,1}, \ldots, p_{1,m_1} \rightarrow & u_1[k] \\ \vdots & \vdots \\ p_{r,1}, \ldots, p_{r,m_r} \rightarrow & u_r[k] \end{pmatrix} = \mathbf{u}[k] \rightarrow \mathbf{y}[k] = \begin{pmatrix} y_1[k] \\ \vdots \\ y_s[k] \end{pmatrix} \quad (1)$$

To illustrate this notation, we provide the actual parameterization used in our experimental evaluation. The system we consider is an automatic transmission system where inputs are the two signals *brake* and *throttle*, and output signals are *RPM*, *speed* and *gear*. Inputs signals are interpolated using the `pchip` option in MATLAB with three regular intervals for *brake* and seven for *throttle*. Instantiating Equation (1) and using default parameter names from our

[1]https://github.com/decyphir/ARCH20_ATwSS

tool in that case gives

$$\begin{pmatrix} \mathit{throttle\_u}0, \dots, \mathit{throttle\_u}6 & \rightarrow \mathit{throttle}[k] \\ \mathit{brake\_u}0, \dots, \mathit{brake\_u}2 & \rightarrow \mathit{brake}[k] \end{pmatrix} = \mathbf{u}[k]$$

$$\rightarrow \mathbf{y}[k] = \begin{pmatrix} \mathit{gear}[k] \\ \mathit{RPM}[k] \\ \mathit{speed}[k] \end{pmatrix} \quad (2)$$

A trace tr is a 3-tuple $tr = (\mathbf{p}, \mathbf{u}, \mathbf{y})$, where $\mathbf{u} = \mathbf{u}(\mathbf{p})$ and $\mathbf{y} = S(\mathbf{u})$.

A *requirement* φ for system S maps traces to Boolean values. We denote $tr \models \varphi$ if $\varphi(tr) = \top$ and $tr \not\models \varphi$ if $\varphi(tr) = \bot$. In this work we consider requirements expressed as Signal Temporal Logic (STL) formulas, which will be described in the next sections.

2.2 Signal Temporal Logic

The grammar of STL formulas is defined as

$$\varphi ::= \pi^\mu \mid \neg\pi^\mu \mid \varphi \wedge \psi \mid \Box_{[a,b]}\psi \mid \varphi\, \mathcal{U}_{[a,b]}\psi.$$

Here, π^μ is a predicate $\mathbb{R} \rightarrow \mathbb{B}$ whose truth value is determined by the sign of a function $\mu : \mathbb{R}^n \rightarrow \mathbb{R}$, and φ and ψ are STL formulas. $\wedge$ denotes logical *and*, $\Box_{[a,b]}$ is the timed *globally* (or *always*) operator, and $\mathcal{U}_{[a,b]}$ is the timed *until* operator. We define logical *or* $\varphi \vee \psi$ as $\neg(\neg\varphi \wedge \neg\psi)$, and the timed *eventually* operator $\Diamond_{[a,b]}\varphi$ as $\neg(\Box_{[a,b]}\neg\varphi)$. Note that we define future operators for STL, but it can also be extended to past operators as in [10]. Like in [21], we define the validity of a formula φ with respect to the discrete-time signal $\mathbf{x}$, that in this work consists of elements from $\mathbf{u}$ and $\mathbf{y}$, at time instant k as

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

2.3 Falsification

Falsification of CPSs refers to the process of finding counterexamples to models of CPSs given a temporal logic requirement. With the use of quantitative semantics for the temporal logic requirement, the problem of generating test cases for the system is transformed into an optimization problem, where the quantitative measure aims to be minimized as a negative quantitative measure means that the requirement has been falsified. Figure 1 illustrates the common falsification procedure as performed by falsification tools.

The *Generator* takes as input the finite set of parameters $\mathbf{p}$ that parameterize the input to the simulation model. The *System* simulates the model with the generated input to give an output trace $\mathbf{y}[k]$. The *Quantitative evaluation* is used to calculate an objective function value, which is fed to a *Parameter optimizer* that tries to generate new parameter values that will falsify the requirement in subsequent simulations of the system.

2.4 Quantitative semantics for STL

We present different variations of quantitative semantics for STL, which indicate not just whether a requirement is fulfilled or not, but how *robustly* it is fulfilled. This is the reason why we also refer to the quantitative measure as the robustness value. For STL robustness, a positive value indicates that the requirement is fulfilled, and a negative value indicates that the requirement is not fulfilled. The quantitative semantics are presented as in previous works [21] as a real-valued function ρ^φ of a signal $\mathbf{x}$ and time index k such that $(\mathbf{x},k) \models \varphi \equiv \rho^\varphi(\mathbf{x},k) > 0$.

We refer to the standard semantics defined in earlier works [9, 13] as the *max* semantics (similar to the definition of Valued Booleans [5]). Apart from this, we also present additive quantitative semantics.

2.4.1 Max semantics. The max semantics are defined as

$$\begin{aligned}
\rho^\mu(\mathbf{x},k) &= \mu(\mathbf{x}[k]) \\
\rho^{\neg\mu}(\mathbf{x},k) &= -\mu(\mathbf{x}[k]) \\
\rho^{\varphi\wedge\psi}(\mathbf{x},k) &= \min(\rho^\varphi(\mathbf{x},k), \rho^\psi(\mathbf{x},k)) \\
\rho^{\Box_{[a,b]}\varphi}(\mathbf{x},k) &= \min_{k' \in [k+a,k+b]}(\rho^\varphi(\mathbf{x},k')) \\
\rho^{\varphi\mathcal{U}_{[a,b]}\psi}(\mathbf{x},k) &= \max_{k' \in [k+a,k+b]}(\min(\rho^\varphi(\mathbf{x},k'), \\
&\qquad \min_{k'' \in [k,k']}\rho^\varphi(\mathbf{x},k'')))
\end{aligned}$$

2.4.2 Additive semantics. The additive semantics are originally defined for Valued Booleans [5]. Note that while a Valued Boolean contains both a Boolean value and a non-negative robustness value, and STL robustness is just a real-valued number, the two are in practice equivalent and their only difference is technical.

For the additive semantics, we only redefine robustness for $\wedge$, $\Box$ and $\mathcal{U}$. For clarity, we denote $\rho^\varphi(\mathbf{x},k)$ as ρ^φ and $\rho^\psi(\mathbf{x},k)$ as ρ^ψ in the following definitions.

$$\rho_+^{\varphi\wedge\psi}(\mathbf{x},k) = \begin{cases} \frac{1}{\frac{1}{\rho^\varphi} + \frac{1}{\rho^\psi}} & \text{if } \rho^\varphi, \rho^\psi > 0 \\ 0 & \text{if } \rho^\varphi = 0 \text{ or } \rho^\psi = 0 \\ \rho^\varphi + \rho^\psi & \text{if } \rho^\varphi, \rho^\psi < 0 \\ \min(\rho^\varphi, \rho^\psi) & \text{otherwise} \end{cases}$$

$\rho^{\Box\varphi}(\mathbf{x},k)$ follows by considering the always operator as conjunction over the time axis and then applying the additive semantics for $\wedge$.

$$(\mathbf{x},k) \models \Box_{[a,b]}\varphi \;\Leftrightarrow\; \bigwedge_{k'=k+a}^{k+b} (\mathbf{x},k') \models \varphi.$$

$\rho^{\varphi\mathcal{U}_{[a,b]}\psi}(\mathbf{x},k)$ follows in similar fashion by defining the until operator according to

$$(\mathbf{x},k) \models \varphi\,\mathcal{U}_{[a,b]}\,\psi \;\Leftrightarrow\; \bigvee_{k'=k+a}^{k+b} \left((\mathbf{x},k') \models \psi \wedge \left(\bigwedge_{k''=k}^{k'} (\mathbf{x},k'') \models \varphi \right) \right)$$

which is equivalent to the previous definition for discrete time.

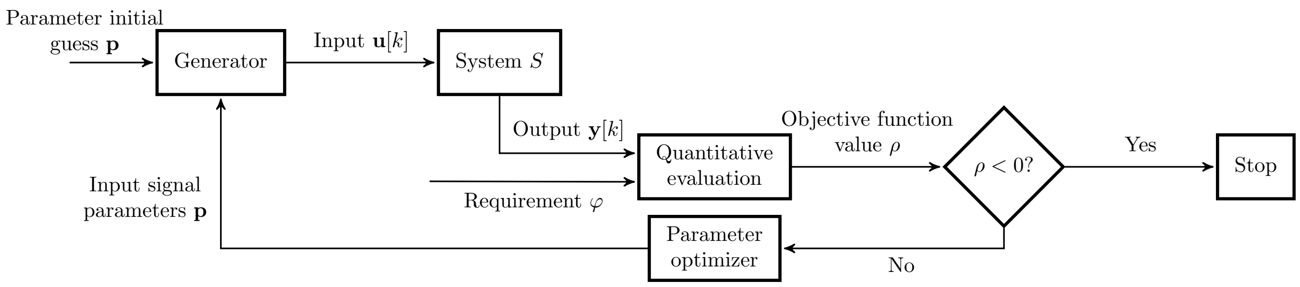

Figure 1: A flowchart depicting the typical optimization-based falsification procedure.

2.4.3 Local predicate normalization. The additive semantics defined above can act as a measure against the masking and scaling problems of temporal logic robustness. For example, the robustness of a conjunction of two negative robustness values will add both values and therefore change as soon as one of the sub-robustnesses change, while the standard max semantics only changes when the sub-robustness with the lowest value changes.

Another measure to prevent the masking and scaling problems is the *local predicate normalization* which we introduce here. Recall that while earlier works have discussed scaling on a global level, i.e., rescaling signals based on expert system knowledge [24], here we scale each predicate without any system knowledge or even information about other potential predicates in the requirement.

Each predicate π^μ has its truth value determined by the sign of the function $\mu(\mathbf{x}[k])$ for a signal $\mathbf{x}$ at time instant k. However, as the execution of the system S under test yields signals defined for a set of time instants, predicates will typically also be defined for a set of time instants. Assume we wish to calculate the robustness values of a predicate for time instants $k_1, \dots, k_{kmax}$. If we define the standard robustness as $\boldsymbol{\mu} = (\mu(\mathbf{x}[k_1]), \dots, \mu(\mathbf{x}[k_{kmax}]))$, we propose rescaling $\boldsymbol{\mu}$ into $\bar{\boldsymbol{\mu}}$ according to

$$\bar{\boldsymbol{\mu}} = \frac{\boldsymbol{\mu}}{\max\left(\left|\min_{k_1,\dots,k_{kmax}} \boldsymbol{\mu}\right|, \left|\max_{k_1,\dots,k_{kmax}} \boldsymbol{\mu}\right|\right)} \tag{3}$$

The effect of this rescaling is that the values of $\bar{\boldsymbol{\mu}}$ are bounded in $[-1, 1]$. As an example, consider the predicate $\pi^\mu := x_1[k] > 0$, with time instants $(k_1, k_2, k_3) = (0, 1, 2)$ and $(x_1[k_1], x_1[k_2], x_1[k_3]) = (5, -15, 20)$. This would give $\mu(x_1[k]) = x_1[k]$ with resulting values $\boldsymbol{\mu} = (5, -15, 20)$ and $\bar{\boldsymbol{\mu}} = (0.25, -0.75, 1)$.

3 MULTI-REQUIREMENT TESTING

In the previous section, we recalled the basics of falsification, where we consider a system with one given requirement and try to falsify it. In this work we extend the scope of this problem to a situation where multiple requirements are defined for some complex design. This typically happens after several stages of development, when some new part of the system is implemented, along with their corresponding requirements, while the requirements for earlier implementations still have to hold and be tested. It is often inefficient to handle the testing of each requirement individually - if only because simulation is likely very costly and it would be an obvious waste of resources to not reuse traces to evaluate the satisfaction of several requirements at the same time - thus we are concerned with the problem of what is the most efficient strategy to falsify or certify to some extent a given set of requirements, which we formalize as follows.

Definition 3.1 (Multi-Requirement Set). A requirement set R is a triplet (R_s, R_a, act) where

- R_s is a finite set of *safety* requirements,
- R_a is a finite (possibly empty) set of *activation* requirements,
- *act* is a mapping from R_s to 2^{R_a}, i.e., *act* maps each safety requirement φ to a finite subset of activation requirements.

This captures an industrial practice observed by the authors for testing with iterative design steps. Safety requirements must not be violated while activation requirements should be satisfied by at least one trace in order to meet a certain confidence level in the testing. In other words, the satisfaction of activation requirements provides a measure of coverage of a testing set. Next we define test runs.

As an example, consider the use case where the system under test is a car, and the specification for the car is that its speed v should never exceed 120 miles per hour. For this example, given a test simulation time of T we could have the safety requirement $\varphi := \Box_{[0,T]} v \leq 120$ and the corresponding activation requirement $\varphi_{act} := \Diamond_{[0,T]} v > 100$. To elaborate, we consider the specification φ "activated" when the speed at some point exceeds 100 miles per hour.

Definition 3.2 (Multi-requirement Test Run). A multi-requirement test set is a pair (T, R) where T is a finite set of traces and R is a multi-requirement set.

We need to characterize a "good" test run against a "bad" one. With classic falsification in mind, an obvious quality measure is the number of safety requirements that the test run manages to falsify. In the following, we note with $card(E)$ the cardinality of any finite set E. Also we write $T \nvDash \varphi$ iff $\exists tr \in T, tr \nvDash \varphi$. Then we define

$$False(T, R) \quad = \frac{card(\{\varphi \in R_s, T \nvDash \varphi\})}{card(R_s)} \tag{4}$$

This however does not take into account activation requirements, for which, in our context, the goal is that they be satisfied by at least one trace. We can measure how well this goal is satisfied by a test run using the following measure.

$$Cover(T, R) \quad = \frac{card(\{\varphi_a \in R_a, \exists tr \in T, tr \models \varphi_a\} \cup \{\varphi_a \in act(\varphi), T \nvDash \varphi\})}{card(R_a)} \tag{5}$$

In words, $False(T, R)$ measure the ratio of falsified safety requirements, and $Cover(T, R)$ measures the ratio of satisfied activation requirements. Note that we count activation requirements as automatically satisfied if the safety requirement they are associated with is falsified. A good test run should catch as many errors as possible and ensure a perfect coverage, so $Cover(T, R)$ should be 1 or close to 1 and $False(T, R)$ should be as high as possible. In the rest of this paper, we fix a given test budget N and multi-requirement R and evaluate methods to create test sets T of cardinality n_{total} that maximize $False$ and $Cover$.

To cast this into a multi-requirement falsification problem, we introduce $R_{\bar{a}} = \{\varphi_{\bar{a}} = \neg\varphi_a, \varphi_a \in R_a\}$, i.e., the set of negations of formulas in R_a. We can then finally compare tests run by counting the number of falsified requirements and negated activation requirements:

$$HitRate(T, R) \quad = \frac{card(\{\varphi \in R_s, T \nvDash \varphi\} \cup \{\varphi_{\bar{a}} \in R_{\bar{a}}, T \nvDash \varphi_{\bar{a}}\})}{card(R_{\bar{a}} \cup R_s)} \quad (6)$$

We now focus on solving the multi-requirement falsification problem. We first present a baseline approach, then move to our main algorithm which improves on the baseline.

3.1 Baseline Algorithm: Corners and Random Search

Our baseline approach presented in Algorithm 1 consists of two phases: in the first phase, we sample $n_{corners}$ corner values, i.e., values for which each parameter p_i is at either its minimum $\underline{p_i}$ or maximum $\overline{p_i}$ value. Indeed, it is well known [22] that this strategy makes it possible to quickly falsify many requirements. After the corners phase, the algorithm removes the falsified requirements and samples the remaining of the n_{total} new traces from a uniform random distribution. In this work, $n_{corners}$ typically comprises around 10% of n_{total}.

Algorithm 1 Corners and random search, CornersRandom

```
Require: $S, R, n_{corners}, n_{total}$
1: $curR \leftarrow \{R_s \cup R_{\bar{a}}\}$            # Current requirements
2: $curR \leftarrow$ cornerFalsification$(S, curR, n_{corners})$
3: $curR \leftarrow$ randomFalsification$(S, curR, n_{total} - n_{corners})$
4: return curR
```

The algorithm is rather straightforward except for the choice of the corner samples. The number m of parameters that we consider is typically too large for us to consider all 2^m possible combinations of minimum and maximum values (i.e., $2^m \gg n_{corners}$). We thus have to use some heuristic to rank which corner to test first. The main trick we use here is to take advantage of the partitioning of parameters with respect to signals. Recall that $\mathbf{p} = (\mathbf{p}_1, \ldots, \mathbf{p}_r)$ where r is the number of input signals. We note $\overline{\mathbf{p}} = (\overline{p_1}, \ldots, \overline{p_m})$ and $\underline{\mathbf{p}} = (\underline{p_1}, \ldots, \underline{p_m})$, and similarly for $\mathbf{p}_i$. We define *signal corners* to be the corners for which for all $i \leq r$, $\mathbf{p}_i = \underline{\mathbf{p}_i}$ or $\mathbf{p}_i = \overline{\mathbf{p}_i}$. Signal corners are in general in much smaller number, so they can often be tested exhaustively. Also, decoupling signals make it possible to detect opposite monotonicities. For example, in the automatic transmission example, some requirements are typically increasingly true with throttle *increasing*, and with brake *decreasing*, thus violations can often be found at maximum throttle and minimum braking. Thus we pick the corners in the following order: first we test $\underline{\mathbf{p}}$ and $\overline{\mathbf{p}}$. Then, as long as the number of samples is lower than $n_{corners}$, we pick a random signal corner. If signal corners are depleted, we pick random corners until reaching $n_{corners}$.

3.2 Focused Multi-requirement Falsification

It turns out that the Corners-Random approach performs very well in a multi-requirement setting, particularly when compared to other naïve approaches that would, e.g., try to falsify each requirement individually in a sequence. Indeed, such an algorithm has an inherent risk – if one requirement turns out to be difficult to falsify or not falsifiable, the simulations used to attack it are wasted, in a sense. Corners-Random, on the other hand, is agnostic to the requirements under test, and does not need more customization by the tester other than setting a reasonable input parameterization for the system under test. To find an approach that performs reliably better than the baseline approach requires efficient heuristics targeting the right requirement(s) and reusing each trace as much as possible.

The multi-requirement falsification can be cast as a multi-objective optimization problem, where one tries to minimize the robustness ρ^{φ} of all requirements in $\{R_s \cup R_{\bar{a}}\}$ together and check the sign of their minimum indicating a violation. In general, one can define a function $F(\{R_s \cup R_{\bar{a}}\})$ and minimize F over $(p_1, \ldots, p_m)$, hence resorting to a standard optimization problem. Typical choice for F include the minimum function ($\min_i \rho^{\varphi_i}$) or some weighted sum of ρ^{φ_i}. However, since m is large, using a standard optimization algorithm might not be for the best – in particular if for each φ, there is a set of parameters of which ρ^{φ} is independent.

As an alternative, we propose a multistage approach: we solve sequentially smaller falsification problems of the form

$$\min_{p_1^k, \ldots, p_{m^k}^k} \rho^{\varphi^k},$$

where $\{p_i^k\}_{m_k}$ is a subset of $\{p_i\}_m$ with $m_k \ll m$. The rationale is that each individual φ_k is likely sensitive to only a subset of the parameters, hence the optimization algorithm needs only consider those as optimization variables, resulting in a search space of reduced dimensions. The problem can then be solved using an efficient falsification engine. Two questions remain to make this a suitable approach: For each φ, which parameters should be included in its optimization problem? For each stage, which φ is the most likely to be easy to falsify? To answer the first question, we make use of an efficient global sensitivity analysis which we briefly summarize in the next section. For the second question, we collect robustness values for all requirements at every step and use some decision rule to pick the most likely candidate, helped by the fact that normalization of robustness values makes it possible to compare robustness values between different requirements. The overall approach is depicted in Figure 2, and our algorithm is further detailed in the next sections.

3.3 Sensitive Parameters Selection

A set of parameters to which a requirement is likely insensitive can be obtained efficiently by using a one-factor global sensitivity

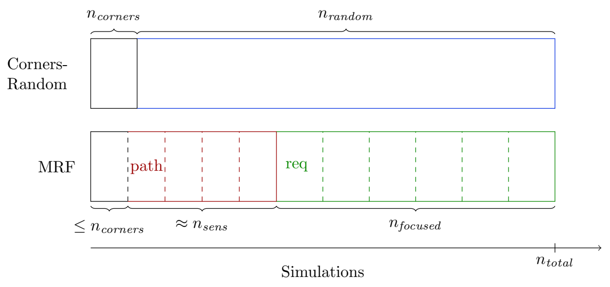

Figure 2: An overview of the baseline Corners-Random algorithm and the proposed MRF algorithm. The Corners-Random algorithm always uses $n_{corners}$ simulations for corner samples and the rest of the n_{total} simulations for random samples. MRF uses up to $n_{corners}$ samples, but may exit the adaptive corners phase early. The sensitivity analysis lasts for approximately n_{sens} samples, rounded down to the nearest number to finish a whole number of paths. The last phase, focused falsification, uses all remaining simulations in the total budget of n_{total}.

analysis, more specifically the elementary effects screening method [20]. With the given mapping from input parameters to robustness values, we try to quantify how much each input parameter affects the robustness of each safety and activation requirement in R. We do so by executing a number of *paths*, each consisting of $m+1$ samples. A path starts with a randomly sampled point in the parameter space. Each subsequent sample in the path varies exactly one parameter while holding all other parameters fixed. The *elementary effect* $d_{i,k}$ of parameter p_i to requirement R_k is then defined as

$$d_{i,k} = \frac{\rho^{\varphi}\left(p_1, \ldots, p_{i-1}, p_i + \Delta, p_{i+1}, \ldots, p_m\right) - \rho^{\varphi}(\mathbf{p})}{\Delta}, \quad (7)$$

where Δ is the change in the parameter p_i, and all changes in all parameter during the path ensures that the chosen parameter values are always within their respective domains. Note that Δ is arbitrarily chosen so that it covers a significant enough portion of the input parameter range. By calculating the mean μ_d and variance σ_d^2 of the distribution of the elementary effects for each parameter, we approximate how much each parameter affects each requirements' robustness value. One natural way of selecting parameters to include when solving the focused falsification problem is to discard all parameters with $|\mu_d|$ below some threshold.

3.4 MRF algorithm

The MRF algorithm we propose is given in Algorithm 2.

The algorithm consists of several different parts – for an illustration, see Figure 2. First is the *adaptive corners phase*, then the *sensitivity analysis*, followed by the *focused falsification phases* which consist of two functions. The different parts of the algorithm are detailed in the following subsections.

3.4.1 Adaptive corners phase. During the entire algorithm, we keep track of the requirements still to be falsified in a variable *curR*, which is initialized to the entire requirement set R. For each part

Algorithm 2 Multi-Requirement Falsification, MRF

```
Require: $S, R, n_{corners}, n_{sens}, n_{total}$
 1: $curR \leftarrow \{R_s \cup R_{\bar{a}}\}$                  # Current requirements
 2: $h \leftarrow \emptyset$          # Variable to store robustness and trace history
 3: $[curR, h] \leftarrow$ ADAPTIVECORNERS$(S, curR, n_{corners}, h)$
 4: $[M, curR, h] \leftarrow$ SENSANALYSIS$(S, curR, n_{sens}, h)$
 5: $tbL \leftarrow \emptyset$                                  # Tabu list
 6: while $curR \neq \emptyset$ and budget $n_{total}$ not exhausted do
 7:     $reqFoc \leftarrow$ SELECTREQTOFOCUS$(curR \setminus tbL, h)$
 8:     $tbL.append(reqFoc)$
 9:     $[curR, h] \leftarrow$ FOCFALSIFICATION$(M, curR, R, reqFoc, h)$
10: end while
```

of the algorithm (adaptive corners phase, sensitivity analysis, and focused falsification phases) we always store the robustness and trace history in h of all current requirements in *curR*, which is necessary to have in each focused falsification phase. Note that ADAPTIVECORNERS and SENSANALYSIS do not *need* h; it is rather included so that the functions can update the variable h with new robustness values and traces.

The adaptive corners phase is similar to the corners falsification in the baseline (Algorithm 1), with the addition that it can exit early if there are no new falsified samples for a set number of trajectories in a row. Specifically, we set a threshold parameter η_{adapt}, such that $0 < \eta_{adapt} < 1$ and if $\eta_{adapt} n_{corners}$ samples pass without any falsifications of the current requirements, ADAPTIVECORNERS returns without exhausting its entire budget of $n_{corners}$ simulations. The updated *curR* variable which is returned from *curR* contains all the requirements that have not been falsified by the adaptive corners phase.

Note that the main purpose of starting the MRF algorithm with the adaptive corners phase is to reduce the number of requirements to monitor for the sensitivity analysis. Based on previously published results on benchmarks [12], there are typically a set of requirements that can be easily falsified by corners, so taking advantage of this fact can reduce the total computation time needed for monitoring requirements during sensitivity analysis.

3.4.2 Sensitivity analysis. This step of the algorithm performs one-factor global sensitivity analysis, the elementary effects screening method, for a maximum of n_{sens} simulations. We approximate the sensitivity of each requirement in $\{R_s \cup R_a\}$ to each input parameter $p_i \in \mathbf{p} = (p_1, \ldots, p_m)$ by evaluating a certain number of *paths*, where each path contains $m+1$ simulations.

The $m \times N_{Rc}$ matrix M returned from SENSANALYSIS is the collection of means of the distribution of elementary effects of each input parameter, i.e., μ_d in the elementary effects screening method. Here, N_{Rc} is the number of requirements in *curR* when SENSANALYSIS is invoked. The returned *curR* is the collection of requirements that were not falsified so far during the adaptive corners or sensitivity analysis phase.

3.4.3 Focused falsification. This phase repeatedly uses an optimization solver, SNOBFIT [16] in our experiments, to try to falsify specific requirements, one at a time. Each iteration of focused falsification consists of two functions. The first function, SELECTREQTOFOCUS, takes the list of current requirements and the history of all previous simulations (from adaptiveCorners, sensAnalysis, and potentially earlier iterations of focFalsification). Based on the robustness and trajectory history, a requirement is selected to be *focused* in this iteration. Selection criteria can be for example

- minimum robustness, i.e., select $R_i \in curR \setminus tbL$ such that $\min(\rho^{R_i}) \leq \min(\rho^{R_j})$ for all other $R_j \in curR \setminus tbL$,
- maximum robustness gap, i.e., select $R_i \in curR \setminus tbL$ such that $(\max(\rho^{R_i}) - \min(\rho^{R_i})) \geq (\max(\rho^{R_i}) - \min(\rho^{R_i}))$ for all other $R_j \in curR \setminus tbL$, or
- minimum number of sensitive input parameters, i.e., select $R_i \in curR \setminus tbL$ such that the total number of occurrences where μ_d has a value below a given selection threshold is higher than or equal to all other $R_j \in curR \setminus tbL$.

The tabu list tbL is used to ensure that no requirement is focused more than once (at least until other requirements have been focused). A separate falsification problem is then created for the focFalsification function, which also takes into account the sensitivity information in the matrix M. The focFalsification function is detailed in Algorithm 3.

Algorithm 3 Focused falsification, focFalsification

```
Require: $M, curR, R, reqFoc, h$
1: $\mathbf{p}_s \leftarrow$ selectSensitiveParams($M, reqFoc$)
2: $\mathbf{p}_r \leftarrow \mathbf{p} \setminus \mathbf{p}_r$                # Set of random parameters
3: $h_r \leftarrow h.reqFoc$          # Robustness corresponding to reqFoc
4: $T \leftarrow$ falsification($h_r, \mathbf{p}_s, \mathbf{p}_r$)              # Set of traces
5: $h \leftarrow$ monitorAllReqs($curR$)                 # Store history
6: $curR.remove(\{\varphi \in curR, T \nvDash \varphi\} \cup \{\varphi_{\bar{a}} \in act(\varphi_s), T \nvDash \varphi_s\})$
7: return [$curR, h$]
```

First, we select only the input parameters $\mathbf{p}_s \subseteq \mathbf{p}$ that show sensitivity for *reqFoc* in the matrix M from the sensitivity analysis phase. These are the parameters that will be given as the optimization variables in the falsification problem formulation. The parameters that are not selected as optimization variables, i.e., all the parameters in $\mathbf{p}$ that show no sensitivity towards the robustness of *reqFoc*, are selected as *random parameters* $\mathbf{p}_r$ for the optimization problem. The random parameters will have random values inside their respective domains for each new trace generated by the falsification, but the optimization solver does not include it in the optimization problem. As such, if $\mathbf{p}_s$ is a proper subset of $\mathbf{p}$, we reduce the dimension of the optimization problem, while still varying the random parameters randomly so that we can "get lucky" and happen to falsify another requirement in *curR* while focusing on *reqFoc*. Another motivation for varying the values of $\mathbf{p}_r$ is that the sensitivity analysis step is only an approximation of the true sensitivity from input parameters to robustness values of requirements, so a parameter that was found to be non-sensitive to a requirement could still affect the robustness value of that requirement in some cases.

We select the robustness history h_r, i.e., the robustness history corresponding to *reqFoc*, from the complete robustness and trace history h of all requirements in *curR*. From the optimization solver, we require that we can include previous objective function values to potentially increase the performance of the solver. We also use a method for generating random values of all random parameters in $\mathbf{p}_r$ even though they are not included as optimization variables.

After the focused falsification concludes, either by falsifying *reqFoc* or by running out of its simulation budget, we monitor all requirements in *curR* over the set of traces T generated by the optimization solver. We store the robustness and trace history in h, remove any requirements found to be falsified by the trace, and return the current set of requirements together with the complete history h. Note that for each violated safety requirement, we also remove all its corresponding activation requirements, as the activation criteria are no longer deemed important in relation to falsifying the safety part of the requirement.

4 RESULTS

In this section, we first define our experimental setup, with choices of hyper-parameters that affect how the presented algorithms work in practice. We also present a set of tables with results and discuss the contents of these tables. Note that additional detailed tables, along with the code to generate all the results in the paper, are uploaded to a public Github repository[2].

4.1 Experimental setup

We run our proposed Algorithm 2 on a benchmark of industrially-inspired requirements [10] for a model of the automatic transmission system of a vehicle (see [15] for further details on the model). The benchmark has two different variations of the requirements; one easier configuration called *base*, and one harder configuration called *hard*. The structure of the requirements are the same for both base and hard configuration, but certain parameter values in the requirements differ. There are also two different versions of the model that can be simulated, one *standard* (or *non-artificial*) model and one *artificial* model including *artificial signals*. In both versions of the model, there is one input for the throttle and one input for the brake of the vehicle, which will make the vehicle accelerate and decelerate, respectively. There are 11 artificial signals, which is a large number of input signals compared to usual falsification benchmarks. These signals are meant to increase the complexity of the falsification problem, but they are only part of the requirements and not the system dynamics. The corresponding artificial requirements were designed in particular to be resilient to corner falsification, i.e., they can be falsified only if the artificial signals stay in some middle range for some duration. In total, the standard model has 35 requirements, and the artificial model has 70 requirements. Out of these 70 requirements, 35 are the same as in the standard model, and 35 contain additional logic each including a subset of the artificial signals in the artificial model.

The test cases are generated as follows. Each simulation lasts for 30 seconds. The throttle and brake inputs are generated as described in Section 2.1. The artificial signals are generated in the same way as the brake signal, i.e., by interpolating with the `pchip` setting in MATLAB between three regular intervals. The throttle input and each artificial signal have parameters in the range $[0, 100]$, while the brake input has parameters in the range $[0, 325]$.

We choose to use additive semantics with normalized predicates, both of these choices are made in order to avoid the masking problem of robustness for STL requirements. Additive semantics and normalization are presented in Section 2.2.

[2] https://github.com/JohanEddeland/Focused_Falsification_REP

For the requirement selection during the focused falsification phase, i.e., the function selectReqToFocus in Algorithm 2, we select the requirement in the history with the lowest robustness value (excluding all requirements in the tabu list *tbL*). In our earlier experiments, choosing lowest robustness as the selection criterion gave better performance than, e.g., choosing the requirement with the minimum number of sensitive input parameters. The total simulation budget is $n_{total} = 3000$, which for the baseline Corners-Random algorithm is split between the corners phase and the random phase. For the non-artificial model, $n_{corners} = 150$, while for the artificial model $n_{corners} = 260$ – these number are explicitly dependent on the number of input parameters in the models. The number of random samples take the rest of the total budget. For MRF, $n_{corners}$ is the same as in Corners-Random (though recall that the adaptive corners phase can exit early), and n_{sens} is around 1000 but set so that a whole number of paths are taken. For the non-artificial model, this means $n_{sens} = 990$ (90 paths with 10 input parameters), and for the artificial model, $n_{sens} = 968$ (22 paths with 43 input parameters). We note that using approximately a third of the total simulations for sensitivity analysis is an arbitrary choice, but one that has shown good performance in our earlier experiments. The focused falsification phase takes the rest of the total budget. We use the solver SNOBFIT [16] during this phase, a solver that has provided good results in similar applications before [11].

4.2 Results and discussion of MRF

For each of the requirement variations and each of the models, we run the baseline algorithm (corners and random search) and our proposed MRF algorithm for 3000 simulations each. This procedure is repeated for 10 different random seeds to produce Table 1 (noting that one seed takes approximately 48 hours to complete on a recent computer). In this table, the average number of simulations include simulations used for sensitivity analysis as well. We have previously tried a version of MRF without the sensitivity analysis phase, but this resulted in worse performance than the current version.

The columns are for the different combinations of non-artificial or artificial model, base or hard configuration, and Corners-Random (Algorithm 1) or MRF (Algorithm 2). The rows show for different performance measures, including *False*, *Cover*, and *HitRate* defined in Section 3. The combination of non-artificial model and base scenario is considered the easiest of all four combinations, and the combination of artificial model and hard scenario is considered the most difficult.

The first row shows the average of the *False* measure, i.e., the proportion of safety requirements that were falsified. Overall, MRF performs slightly better than the baseline algorithm in most cases, but not with a big margin.

On the second row, we report the average of the *Cover* measure, which is the proportion of activation requirements that either were activated or had their corresponding safety requirement falsified. The results here are closer than for the *False* measure: there is similar performance for the easiest combination, MRF is slightly better for the hardest combination, and Corners-Random is slightly better for the remaining two combinations.

The third row presents the average of the *HitRate* measure, which takes both activation and safety requirements into account. We see that MRF is slightly better than Corners-Random for all combinations except for the artificial model and base scenario.

The fourth row shows the average of the number of simulations used for each requirement. Recall that a requirement is removed from the active set of requirements as soon as it is falsified, so not falsifying *any* requirements would result in 3000 average simulations in total. We can see that the baseline algorithm of Corners-Random has on average fewer simulations than MRF. This is largely due to MRF spending many simulations (around 1000) on sensitivity analysis, where only one input parameter is varied at a time. As a result, it takes longer for MRF to find cases of requirements to falsify that the baseline algorithm finds during the first 1000 simulations, which substantially increases the average number of simulations needed.

The fifth row shows the average number of simulations needed, but only for the requirements that are ever falsified. It is once again clear that Corners-Random uses less simulations on average to falsify requirements, and the most important factor is the sensitivity analysis phase of MRF.

To summarize the results in the table, MRF is overall better at falsifying the more difficult requirements, but slower at falsifying the requirements which are easier to falsify. The combination where Corners-Random performs better than MRF for all measures is for the artificial model and base scenario. This scenario is characterized by many input parameters (the artificial model has many artificial inputs), but requirements that are easier to falsify than in the hard scenario. One explanation for why MRF does not perform as well for this scenario is that the method for selecting which requirement to focus chooses sub-optimal requirements, i.e., requirements that are too difficult or impossible to falsify. As mentioned earlier, selecting the "wrong" requirement in a focused falsification results in many potentially wasted simulations, especially if this results in the algorithm not having enough simulations to focus on another requirement that was possible to falsify if given the chance.

Another perspective of the results are shown in Table 2. In this table, we compare falsification capabilities of Corners-Random and MRF by counting cases when one falsifies a requirement and the other does not, etc. This is done for all scenarios reported in Table 1.

We can once again see that Corners-Random is overall faster, since there are more cases where, when both algorithms falsify the requirement, Corners-Random has an earlier falsification index than MRF. We also see that it occurs more times that MRF falsifies when Corners-Random does not, compared to the other way around. This also fits the results in Table 1. An interesting point is that we also see that for the non-falsified requirements, MRF typically has lower minimum robustness values compared to Corners-Random. While lower robustness value does not necessarily mean that we are closer to finding a falsifying trace, it should be a good thing for analysis more often than not (otherwise one would likely want to revise the quantitative semantics used).

During all 40 runs, a total of 243 requirements were focused in the focused falsification phase of Algorithm 2 (meaning that on average, around 6 requirements were focused in each run). Out of these 243 focused requirements, 75 were falsified, with 51 out of these 75 being falsified during their "focus period", i.e., during the falsification problem where the requirements robustness value was used as objective function.

Table 1: Table comparing the baseline Corners-Random algorithm with our proposed MRF algorithm using the measures $False, Cover$, and $HitRate$ defined in Section 3. There are four combinations of models and requirements on which we compare the algorithms; each of non-artificial and artificial model, as well as each of the base and hard settings for the requirements. The numbers presented are the averages taken over 10 runs with different random seeds, each run having a maximum simulation budget of 3000.

	Non-artificial				Artificial			
	Base		Hard		Base		Hard	
	Corners-R	MRF	Corners-R	MRF	Corners-R	MRF	Corners-R	MRF
Avg. $False(T,R)$	14.9 / 16	**15.5 / 16**	12.8 / 16	**13.1 / 16**	**23.9 / 32**	**23.9 / 32**	18.3 / 32	**19.3 / 32**
Avg. $Cover(T,R)$	**19.0 / 19**	**19.0 / 19**	**16.9 / 19**	16.8 / 19	**32.7 / 38**	32.4 / 38	26.5 / 38	**27.1 / 38**
Avg. $HitRate(T,R)$	29.9 / 35	**30.4 / 35**	26.8 / 35	**27.0 / 35**	**52.2 / 70**	50.7 / 70	41.7 / 70	**42.2 / 70**
Avg. #sim	**546.0**	575.2	**829.9**	891.9	**1091.1**	1199.9	**1426.6**	1510.0
Avg. #sim (successful)	**137.0**	262.3	**186.9**	267.3	**478.9**	622.6	**377.6**	732.7

Table 2: Table summarizing comparison between Corners-Random and MRF. For each requirement in all of the falsification runs presented in Table 1, we count which algorithm falsified the requirement faster, or if no algorithm falsified it, which algorithm yielded the lowest robustness value for that requirement.

Description	Occurrences
Both falsify, Corners-Random faster	385
Both falsify, same first falsification index	670
Both falsify, MRF faster	355
Corners-Random falsifies, MRF does not	62
MRF falsifies, Corners-Random does not	93
Neither falsifies, Corners-Random lower rob	129
Neither falsifies, same lowest rob	40
Neither falsifies, MRF lower rob	211
MRF removes $act(\varphi_s)$ because φ_s is falsified, Corners-Random falsifies either φ_s or $act(\varphi_s)$	154
MRF removes $act(\varphi_s)$ because φ_s is falsified, Corners-Random does not falsify either φ_s or $act(\varphi_s)$	1

4.3 Sensitivity analysis

One of the big inspirations for this work is the fact that large systems in industrial settings can have "groups" of inputs that affect different requirements. Using sensitivity analysis is one way explored in this paper to reduce the number of dimensions in the falsification problem for a specific requirement, given that there is a set of input parameters that does not affect its robustness value. We note that the actual calculation of the sensitivity between inputs and requirements, including calculating the elementary effects, takes negligible time in comparison to simulation of the system and monitoring of the requirements.

To provide an insight into how useful sensitivity analysis actually can be for such a system as the one used for the experiments in the paper, Figure 3 shows an overview of the sensitivity information calculated for one run of the MRF algorithm. The sensitivity analysis shows that a majority of the input/requirement pairs have no correlation, which is actually by construction in this case due to artificial signals. Nevertheless, this clearly shows that one can substantially reduce the number of optimization variables used in the falsification problem, which is necessary to make an algorithm that outperforms the baseline Corners-Random algorithm for systems with a large number of inputs. We can also see that one requirement is sensitive to no inputs at all, since the entire column is red. This can happen if, for example, the requirement is Boolean in nature and can only change robustness value if it is actually falsified. Since we know nothing about which parameters could potentially influence the robustness value of the requirement, an intuitive way to work around its insensitive nature is to add it to the tabu list directly after the sensitivity analysis, ensuring that we never focus on the requirement (in practice this means only random exploration will be used to falsify this requirement).

Another use case of sensitivity analysis is to give an overview of the system under test or under development. Industrial systems can take a long time for an expert to get an overview of, so in the case of new developers or testers, an illustration of the sensitivity matrix for the system, such as in Figure 3, can provide a valuable reference sheet for how clusters of inputs affect clusters of requirements. We have seen this be the case for a large-scale model at Volvo Car Corporation.

5 CONCLUSION AND FUTURE WORK

In this paper, we propose a problem formulation of multi-requirement testing, as well as a focused falsification algorithm that solves the problem better than a baseline falsification algorithm using a combination of corners and random samples. In a multi-requirement setting, we found that the Corners-Random algorithm actually performs very well and is difficult to beat performance-wise. In particular it is usually faster in falsifying requirements, especially the ones considered "easy", i.e., the ones that are falsified in each experiment. Nevertheless, our multi-requirement falsification algorithm has equal or better performance compared to Corners-Random for most cases of our experiments evaluated on a benchmark of an automatic transmission system with a large set of industrially-inspired requirements.

Our proposed algorithm uses one-factor global sensitivity analysis to approximate the effect each input parameter of the system has on each requirement. This allows for two novel additions to the

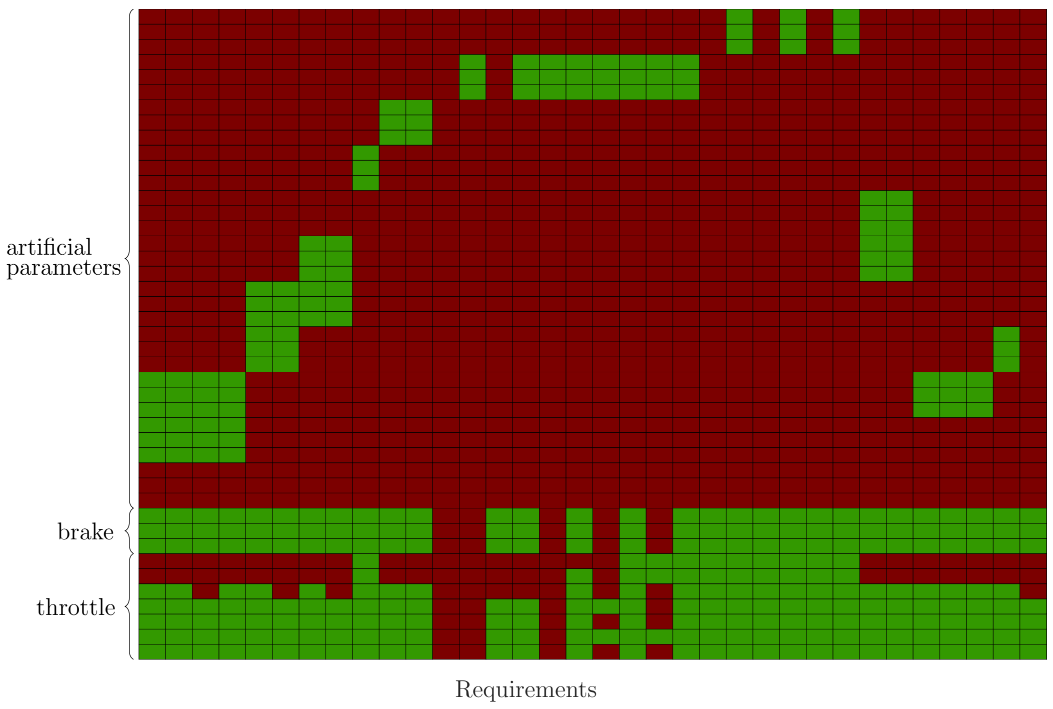

Figure 3: Illustration of approximated sensitivities between input parameters (rows) and requirements (columns) for one instance of MRF, where artificial requirements and the artificial model was used. A cell is red if the requirement is not sensitive to the input parameter ($\mu_d = 0$), and green otherwise. Note that this snapshot is taken from the sensitivity analysis phase of Algorithm 2, meaning that requirements that were falsified in the adaptive corners phase are excluded from this figure.

falsification procedure; firstly, we can remove non-sensitive input parameters when focusing on a specific requirement, which reduces the number of dimensions in the optimization problem. Secondly, we still vary the removed input parameters as random parameters, since we afterwards also monitor all non-focused requirements on the simulation traces resulting from the optimization solver in the focused falsification. In addition to this, the results of the sensitivity analysis results provide an overview of the system behaviour, something which can be very useful when dealing with large-scale industrial systems.

Future work includes further evaluating the effect of different hyper-parameters in our proposed algorithm, such as the method for choosing which requirement to focus on, and maybe more importantly the optimizer to choose during focused falsification (other than SNOBFIT that we used) and how much budget it should be allocated as compared to the total budget. Also, related to the focused requirement choice, there are potential extensions where one could target specific test quality measures to maximize for, for example test coverage (focusing on activation requirements) or falsification (focusing on safety requirements). It would also be of interest to evaluate our algorithm against other potential solvers in the multi-requirement setting, not just the baseline Corners-Random algorithm. We have made the choice to use additive quantitative semantics for STL, and this choice affects the sensitivity analysis as well as the focused falsification part of our algorithm. Future work could include investigating exactly how max and additive quantitative semantics affect sensitivity analysis, as well as if there are other reasonable choices to make for the functions to use in a sensitivity analysis setting.

ACKNOWLEDGMENTS

This work has been performed with support from the Swedish Governmental Agency for Innovation Systems (VINNOVA) project TESTRON 2015-04893 and from the Swedish Research Council (VR) project SyTeC 2016-06204. This support is gratefully acknowledged.

REFERENCES

[1] Sara Abbaspour Asadollah, Rafia Inam, and Hans Hansson. 2015. A Survey on Testing for Cyber Physical System. In *Testing Software and Systems*, Khaled El-Fakih, Gerassimos Barlas, and Nina Yevtushenko (Eds.). Springer International Publishing, Cham, 194–207.

[2] Takumi Akazaki, Shuang Liu, Yoriyuki Yamagata, Yihai Duan, and Jianye Hao. 2018. Falsification of cyber-physical systems using deep reinforcement learning. In *International Symposium on Formal Methods*. Springer, Springer International Publishing, Cham, 456–465.

[3] Yashwanth Annpureddy, Che Liu, Georgios E Fainekos, and Sriram Sankaranarayanan. 2011. S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems.. In *TACAS*, Vol. 6605. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 254–257.

[4] Benoît Barbot, Nicolas Basset, Thao Dang, Alexandre Donzé, James Kapinski, and Tomoya Yamaguchi. 2020. Falsification of Cyber-Physical Systems with Constrained Signal Spaces. In *NASA Formal Methods Symposium*. Springer, Springer International Publishing, Cham, 420–439.

[5] Koen Claessen, Nicholas Smallbone, Johan Eddeland, Zahra Ramezani, and Knut Åkesson. 2018. Using Valued Booleans to Find Simpler Counterexamples in Random Testing of Cyber-Physical Systems. *IFAC-PapersOnLine* 51, 7 (2018), 408–415.

[6] Thao Dang, Alexandre Donzé, and Oded Maler. 2004. Verification of analog and mixed-signal circuits using hybrid system techniques. In *International Conference on Formal Methods in Computer-Aided Design*. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 21–36.

[7] Jyotirmoy Deshmukh, Xiaoqing Jin, James Kapinski, and Oded Maler. 2015. Stochastic local search for falsification of hybrid systems. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, Springer International Publishing, Cham, 500–517.

[8] Alexandre Donzé. 2010. Breach, a toolbox for verification and parameter synthesis of hybrid systems.. In *CAV*, Vol. 10. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 167–170.

[9] Alexandre Donzé and Oded Maler. 2010. Robust Satisfaction of Temporal Logic over Real-Valued Signals. In *Formal Modeling and Analysis of Timed Systems: 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010. Proceedings*, Krishnendu Chatterjee and Thomas A. Henzinger (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 92–106.

[10] Johan Lidén Eddeland, Alexandre Donzé, Sajed Miremadi, and Knut Åkesson. 2020. Industrial Temporal Logic Specifications for Falsification of Cyber-Physical Systems. In *ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20) (EPiC Series in Computing)*, Goran Frehse and Matthias Althoff (Eds.), Vol. 74. EasyChair, 267–274. https://doi.org/10.29007/r74f

[11] Johan Lidén Eddeland, Sajed Miremadi, and Knut Åkesson. 2020. Evaluating Optimization Solvers and Robust Semantics for Simulation-Based Falsification. In *ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20) (EPiC Series in Computing)*, Goran Frehse and Matthias Althoff (Eds.), Vol. 74. EasyChair, 259–266. https://doi.org/10.29007/f4vs

[12] Gidon Ernst, Paolo Arcaini, Ismail Bennani, Alexandre Donze, Georgios Fainekos, Goran Frehse, Logan Mathesen, Claudio Menghi, Giulia Pedrielli, Marc Pouzet, Shakiba Yaghoubi, Yoriyuki Yamagata, and Zhenya Zhang. 2020. ARCH-COMP 2020 Category Report: Falsification. In *ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20) (EPiC Series in Computing)*, Goran Frehse and Matthias Althoff (Eds.), Vol. 74. EasyChair, 140–152. https://doi.org/10.29007/trr1

[13] Georgios E Fainekos and George J Pappas. 2009. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science* 410, 42 (2009), 4262–4291.

[14] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. 1998. What's Decidable about Hybrid Automata? *J. Comput. System Sci.* 57, 1 (1998), 94–124. https://doi.org/10.1006/jcss.1998.1581

[15] Bardh Hoxha, Houssam Abbas, and Georgios Fainekos. 2015. Benchmarks for Temporal Logic Requirements for Automotive Systems. In *ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems (EPiC Series in Computing)*, Goran Frehse and Matthias Althoff (Eds.), Vol. 34. EasyChair, 25–30. https://doi.org/10.29007/xwrs

[16] Waltraud Huyer and Arnold Neumaier. 2008. SNOBFIT-Stable Noisy Optimization by Branch and Fit. *ACM Trans. Math. Softw.* 35, 2 (2008), 9–1.

[17] Xiaoqing Jin, Jyotirmoy V. Deshmukh, James Kapinski, Koichi Ueda, and Ken Butts. 2014. Powertrain Control Verification Benchmark. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control (HSCC '14)*. Association for Computing Machinery, New York, NY, USA, 253–262. https://doi.org/10.1145/2562059.2562140

[18] Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 152–166.

[19] Logan Mathesen, Giulia Pedrielli, and Georgios Fainekos. 2021. Efficient Optimization-Based Falsification of Cyber-Physical Systems with Multiple Conjunctive Requirements. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*. 732–737. https://doi.org/10.1109/CASE49439.2021.9551474

[20] Max D Morris. 1991. Factorial sampling plans for preliminary computational experiments. *Technometrics* 33, 2 (1991), 161–174.

[21] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M. Murray, and Sanjit A. Seshia. 2015. Reactive Synthesis from Signal Temporal Logic Specifications. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control (HSCC '15)*. Association for Computing Machinery, New York, NY, USA, 239–248. https://doi.org/10.1145/2728606.2728628

[22] Zahra Ramezani, Johan Lidén Eddeland, Koen Claessen, Martin Fabian, and Knut Åkesson. 2020. Multiple objective functions for falsification of cyber-physical systems. *IFAC-PapersOnLine* 53, 4 (2020), 417–422.

[23] Zhenya Zhang, Ichiro Hasuo, and Paolo Arcaini. 2019. Multi-armed Bandits for Boolean Connectives in Hybrid System Falsification. In *Computer Aided Verification*, Isil Dillig and Serdar Tasiran (Eds.). Springer International Publishing, Cham, 401–420.

[24] Zhenya Zhang, Deyun Lyu, Paolo Arcaini, Lei Ma, Ichiro Hasuo, and Jianjun Zhao. 2021. On the Effectiveness of Signal Rescaling in Hybrid System Falsification. In *NASA Formal Methods*, Aaron Dutle, Mariano M. Moscato, Laura Titolo, César A. Muñoz, and Ivan Perez (Eds.). Springer International Publishing, Cham, 392–399.

[25] Xin Zhou, Xiaodong Gou, Tingting Huang, and Shunkun Yang. 2018. Review on testing of cyber physical systems: Methods and testbeds. *IEEE Access* 6 (2018), 52179–52194.