THESIS

EFFICIENT INPUT SPACE EXPLORATION FOR FALSIFICATION OF CYBER-PHYSICAL SYSTEMS

Submitted by

Meetkumar Savaliya

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2022

Master's Committee:

Advisor: Vinayak Prabhu

Sudeep Pasricha
Sudipto Ghosh

ABSTRACT

EFFICIENT INPUT SPACE EXPLORATION FOR FALSIFICATION OF CYBER-PHYSICAL SYSTEMS

In recent years black-box optimization based search testing for Signal Temporal Logic (STL) specifications has been shown to be a promising approach for finding bugs in complex Cyber-Physical Systems (CPS) that are out of reach of formal analysis tools. The efficacy of this approach depends on efficiently exploring the input space, which for CPS is infinite. Inputs for CPS are defined as functions from some time domain to the domain of signal values. Typically, in black-box based testing, input signals are constructed from a small set of parameters, and the optimizer searches over this set of parameters to get a falsifying input. In this work we propose a heuristic that uses the step response of the system – a standard system characteristic from Control Engineering – to obtain a smaller time interval in which the optimizer needs to vary the inputs, enabling the use of a smaller set of parameters over which the optimizer needs to search over. We evaluate the heuristic on three complex Simulink model benchmarks from the CPS falsification community, and we demonstrate the efficacy of our approach.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Dr. Vinayak Prabhu. I am really thankful for his guidance from drafting to accomplishing this work. I am really honored to present this work to the Master's Committee who are actively working in this area.

I am extremely thankful to Dr. Sudipto Ghosh and Dr. Sudeep Pasricha for their guidance.

Many thanks to Bradly Zellefrow, Anderson Worcester, and Shlok Gondalia for help with thesis language improvement.

I am very thankful to my family, especially my elder brother, Bhavin Savaliya, and friends who always motivate, and support me at all times.

Again, thank you Dr. Vinayak Prabhu, the thesis would not have been possible without your guidance!

DEDICATION

To my elder brother Bhavin Savaliya

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Cyber-Physical Systems (CPS) combine physical systems including hydraulics, mechanical, and electrical with computational devices, which are the software systems primarily involved in gathering, sending, controlling, and manipulating sensory data. These complex industrial control systems are deployed in numerous domains such as aerospace, automotive, Internet of Things (IoT), and medical devices and therefore need to be robust enough to operate reliably in real-world environments. To ensure this robustness, testing and verification of these large systems are an integral part of the design process, and have a tangible impact on total development costs [1]. Testing and verification are necessary to increase the confidence concerning the performance of the product and verify that all of the safety standards are met.

The complexity of the system should not lead to the erroneous controller design in the implementation phase. The controller should work as a same way as envisioned in the design phase. Traditionally, the software development process of CPS involved developing a monolithic manner code base generation and there was no clear separation between software modules. This was an expensive approach and led to sub-optimal controller design because the testing of the systems was only conducted for experimental test suites.

The model-based design (MBD) paradigm is widely adopted by many organizations. MBD is a mathematical design of complex industrial control systems. The V-model is given in Figure 1.1 depicting the MBD process. This process often starts with the specifications of systems. The merit of using MBD over the traditional software development process is the mitigation of complexity by providing a unified framework for creating, documenting, testing, and deploying reliable software systems for CPS [1]. Moreover, this framework not only helps to design flexible controllers but also allows for early detection of unforeseen errors that might occur in real-time.

Initially, in the design phase of the V-model, the designer starts with the plant design of dynamic characteristics outlined in the specifications using differential, algebraic and logical equations.

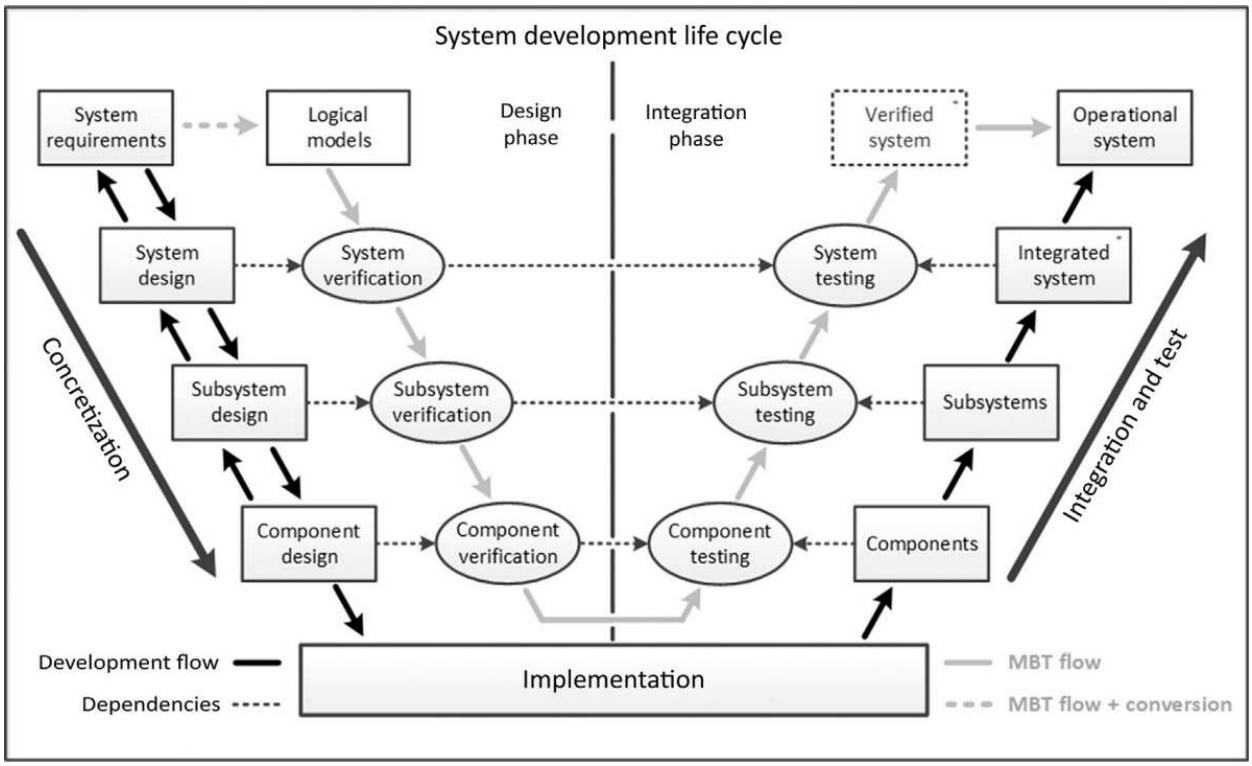

Figure 1.1: V-Model for MBD and MBT from [2]

Next, the design of the controller subsystem that regulates the physical components that is done from design principles from embedded systems, distributed control, and control theory. Lastly, the design of an environmental model that encapsulates the behavior of external entities that may be directly or indirectly interacting with the system [3]. However, these three steps may require refinements in all systems and subsystems design. For example, an inherently modular system that is system might need subsystem design and a clear separation of its modules.

On the right side of the V-model in the Figure 1.1, in the integration phase, all models, systems, or subsystems will be tested individually. The component tests ensure that the individual components are working without fault as per specifications. Model-Based Testing (MBT) is a method of testing the systems, subsystems, or models that are developed in the MBD paradigm in the system development life cycle of the CPS. MBT also ensures that communications between subsystems are also tested. The integration tests are performed in MBT to ensure that the specifications are met and a fully verified system will be released. Both the design and integration phases are involved

in MBD and MBT, respectively. The V-Model System Development Life Cycle helps in faster, error-free, and reusable system development and mitigates most of the defects that could lead to hazards in the real-world applications [2].

However, MBT does not guarantee the correctness of the system through component, network, and integration testing at different levels. There could be an input for which the corresponding output of the system demonstrates undesired behavior. If this happens, then at this point in the system development the refactoring or re-engineering costs could be much higher. The simulation-based testing approach has proven to be promising since no physical implementation of the plant is required. The dynamics of the working model are captured in a Simulink based mathematical model. Note that this mathematical model is the same system that is released in MBT in Figure 1.1. Verification and testing of the system are two major analysis problems that ensure the correctness of safety measures and correctness of the integrated systems [3].

A formal definition of testing is to test the output of a system for an experimental input to the system. There could be multiple inputs from the input space, but the test set in the testing activity is always a finite set from the input space. As the test inputs are not exhaustive, testing does not guarantee the absence of bugs. For example, the testing of the internal combustion engine in the in-house testing facility does not ensure the absence of errors while the vehicle is on the road.

In contrast, verification activity provides the formal proof of correctness of the system for possibly infinite inputs [4]. Formal approaches such as model checking and theorem proving can be used in verification of software systems [3]. A well-known technique is reachability analysis in the CPS domain. The set of reachable states in a given finite time horizon is computed with a particular set of input signals and a set of initial conditions in reachability analysis [3]. The idea is that the system should not reach any unsafe states. These unsafe states are provided in specifications. Also, a common assumption in this technique is the system state is fully observable.

1.1 Black-Box Optimization for Testing

Industrial Cyber-Physical are large and complex, and fully formal verification approaches are usually infeasible for such systems. *Falsification*, based on black-box optimization, has been proposed in order to detect bugs in such complex systems, Falsification is defined as a process of systematically searching for a valid input for that the given CPS model does not hold desired behavior. This input is called counterexample. If the system is given this input counterexample input, then the system output violates the specified system specification, and in this case the system is said to be falsified.

In practice, these behavior specifications are represented using temporal logic. Temporal logic provides flexibility in modeling the behavior of the system concerning time. In falsification, the different inputs are being tested against the system. Moreover, the selection of input is not random. Overall, the provided input will be used in simulating the system and generating outputs that will then be checked against the specifications. Temporal logic includes many extensions such as Linear Temporal Logic (LTL) [5], [6], Metric Temporal Logic (MTL) [7], [8]. In this thesis, we consider the setting where the specifications are encapsulated in Signal Temporal Logic (STL) [9], [10]. STL is an extension of LTL. STL allows us to define the constraints in real-time over real-valued signals.

In the process of simulation-based falsification, the system is considered as a black-box. We only know the interfaces of the system in terms of inputs and outputs. The advantage of considering the system as a black-box is that the system dynamics do not need to not be known. Also, the falsification activity can be constructed as a problem of optimization. The system only sees the input signals and returns some outputs using some underlying unseen logic. In optimization-based falsification, the key work is to generate the potential signal of choice that can falsify the system. In this framework, the falsification of the system will be decided by the *robustness function* from the evaluation of STL specifications against the output of the system. The notion of a robustness function, which is a quantitative measure of how well a system behavior satisfies an STL specification that has been developed in [10] and used for falsification activity in [11], [12]. If the robustness

function returns a negative value then the system is falsified, otherwise, the system behavior is within specification. Hence, for the optimizer, the robustness function is the main function to be minimized when subject to inputs. These inputs are parameterizations from which the input signals will be generated. Intuitively, the system takes input signals and generates the output signals (for example signals $x(t) : [0, T] \rightarrow \mathbb{R}$, and $y(t) : [0, T] \rightarrow \mathbb{R}$), however, the optimizer takes an input parameterization (for example $\alpha_1^x, \alpha_2^x, ..., \alpha_5^x$ for signal $x(t)$ and $\alpha_1^y, \alpha_2^y ..., \alpha_5^y$ for signal $y(t)$ each α_i^x and α_i^y from $\mathbb{R}$ for $i \in \{1, 2, 3, 4, 5\}$), and minimize the robustness functions. To glue the optimizer to the system execution, there needs to be a signal generator which generates the signals $x(t)$ and $y(t)$ from the parameter values generated by the optimizer. For example, the full signal could be completed using some form of interpolation (in such a signal generation scheme from the parameters, the parameters are also known as *control points*). We note that the optimizer knows nothing about signal generation and specification evaluation. Overall, optimization-based falsification has been shown to be promising approach in recent work by the falsification community [12–16]. A more detailed explanation is given in Chapter 2.

1.2 Our Contributions

A basic ingredient of optimization based falsification approaches is the translation of the search space over input signals – defined as functions $[0, T] \rightarrow \mathbb{R}^n$ over some time domain $[0, T]$ – to a finite parameter space over which the black-box optimizers typically operate. Our main contribution in this work is the exploration of a heuristic to reduce the number of parameters for more efficient exploration of the input signal space. Our heuristic takes inspiration from the *step response* concept from the theory of Control Systems. The step response of a system from a given initial state is the response of the system when given a step input. Step response behavior is a critical part of the analysis of control systems [17], and overshoot, rise time, and settling time data from the step response are key characteristics for behavior analysis of dynamical systems. Our heuristic is to vary the input signal not over the entire simulation horizon, but over a time duration equal to the *settling time* for the step response (defined as the time taken for the system output to reach and stay

within a specified error band). For example, if the simulation horizon is 80 seconds, and we wish to have a uniform placement of control points at 2 second intervals, then this would result in 41 control points. If the settling time is 20 seconds, then our heuristic would require 12 control points (we fix a special control point to be at the beginning of the simulation). The placement of this 20 second interval which contains the control points is itself variable, with the offset being determined by the optimizer. Such a heuristic, if successful has two benefits: (1) it reduces the search space for the optimizer, and (2) the falsifying inputs are simpler, as they change over a smaller number of time points, and hence are more desirable for debugging.

We implemented our algorithms in Breach [18], a state of the art falsification tool for Simulink models, and ran benchmarks over three widely used complex Simulink models from the CPS falsification community. Our results show our proposed method to be effective for falsification over these benchmarks. If we consider the least robustness values obtained, for all of the properties in our experiments, except for one, our heuristic either performs better (that is, it achieves better robustness values), or is within 5% of the baseline (where the baseline generates input signals from a uniform placement of control points).

1.3 Related work

The formal definitions of the verification and the falsification are given in [3]. Verification can be viewed as proof of all possible outputs of the system satisfying given specification φ for all possible inputs. Falsification is an inverse process of verification. The traditional method of verification of the CPS, such as model checking is a powerful framework for verifying specifications of finite-state systems, and works for a partial specification where theorem proving does not work [19]. However, model checking based approaches suffer from scalability problems, because of the infinite state space of CPS [20]. In such an infinite state system, the formal verification based approaches are undecidable.

Hence, in the past decades, researchers have been working on testing based approaches [21] that explore good strategies to guide the test case selection process. This is thankful to robustness

guided falsification [22] where a real value robustness function, quantitative semantics over MTL, and its variants like STL is employed which guide the falsification process. A notion of Boolean semantics will provide the true/false answer for φ satisfaction for system behavior. The adoption from MTL semantics, STL semantics is also known as quantitative robustness value, and is a numeric measurement of how "robust" a specification φ hold in the CPS model. Therefore, the notion of robustness function that returns the quantitative robustness value by evaluating the specification φ over the system behavior, and needs to be minimized in robustness guided falsification.

Furthermore, robustness guided falsification can be classified into black-box and grey-box approaches based on the exposure of the underlying CPS model and strategies used for input space exploration. In the black-box approach, the model is completely hidden, and only interfacing of the model such as inputs and outputs are available. However, the gray-box approach is more relevant to the temporal structure of inputs/outputs and the availability of internal dynamics to some extent [23]. Furthermore, the black-box approach is classified into the global optimization problem that uses different optimization methods to minimize the robustness function, and statistical modeling uses different probability distribution based methods [23]. Our work is based on black-box falsification, which is in short evaluation of the specification φ by simulating the system without looking into the system. Overall, the robustness guided falsification requires the simulation of the system, while other traditional verification methods like model checking, theorem proving and reachability analysis do not require any simulations.

MBT constitutes the testing of individual components and their integration before the final release. Hence, some automated test case generation techniques and algorithms have been used that try to cover as much input space as possible. However, there is still a need for inputs that may violate the specification. The falsification community for CPS has been exploring different aspects of the falsification process in recent years. For example, the Markov Chain Monte Carlo method has been used to optimized the problem over input generation to those areas where there are maximum probabilities for violation of specifications [11].

The paper [12] talks about the role of input constraints in the falsification process. Many constraints have been imposed on input signals so that these constraints must be respected by the optimizer in the falsification process. The evaluation of input constraints is contributing to the global cost function. If the input constraints are violated then some penalty will be added in global cost function, otherwise, the cost will be left only to the violation or satisfaction of specifications [12]. This global cost is a robustness value that must be minimized. The global cost function is defined using lexicographical methods for multi-objective optimization with first priority to input constraints satisfaction and then specification evaluation.

While it is relatively the same idea described in the paper [12], the work in [24] proposes a new framework for calculating robustness by explicitly specifying the input and the output interfaces. This new notion of relative robustness has been proposed for inputs and outputs. The notion of vacuity is used to address relative robustness for inputs. Classically, Vacuity is defined as follows: for a given specification φ contains implications ($request \rightarrow response$), the system violates the specification if φ suffers from antecedent failure [25]. In other words, the robustness value of the post-condition ($response$) is not affecting the falsification if the pre-condition ($request$) is not satisfied. The work of [24] presents a proposal for extending this vacuity notion to the case where the formula does not fall into a request-response template. The output robustness measures the degree of satisfying the specification those are defined over output signal variables. In Reactive Systems, the inputs are coming from the environment in which the system operates, while the outputs are the computational outcome of the environmental inputs and some internal states. In addition to relative robustness, after the falsification, for finding an input where the worst (negative) robustness value originated, the work in [12] also proposed a method for narrowing down the time interval in the output trace responsible for the system specification violation. For this, worst case diagnostics and the epoch diagnostics algorithms were used. The worst case diagnostics marks a point in the trace π where the worst robustness value originated, while epoch diagnostics identify the entire trace or the sub traces in which the worst robustness value originates [24].

Another aspect in the falsification of CPS, discussed in the paper [26], is only considering the MTL properties in the form of request $\rightarrow$ response. In other words, if there is an antecedent then there is at least one consequent. Intuitively, this is falsified when there is such an input where the antecedent is satisfied and the consequent is violated. To generate such input, the 2 stage algorithm has been proposed. During stage 1, the algorithm checks for **AF(φ)**. AF stands for antecedent failure. If the antecedent ($request$) is true, then falsification is only left to the consequent. In stage 2, first the input condition and the input prefix will be extracted from the input signal, then using this same prefix, which is a part of input where the antecedent is satisfied the deterministic CPS model will generate another input with different suffixes so that the consequent in the property can be falsified [26].

The black-box checking (BBC) is a combined approach of model checking and automata learning proposed in [27]. The key concept was to apply what had been learned from past falsification trails. For example, in Breach [18] tool, the multiple specifications will be tested sequentially. In this paper, knowledge of the falsification process of one specification could help falsify the other specifications. Intuitively, the learned model will check if it satisfies the specification φ through the model checking, if it does not then it generates a counterexample and checks that the counterexample is also witnessed by the original CPS model. If it does then the counterexample has been found. Otherwise, we have a learned model that violates the specification and original model that satisfies the specification. Hence this contradiction leads to a better learning process using the counterexample.

There are different aspects of verifying the CPS, such as input generation strategies concerning algorithms or the optimizer, control points placements over simulation horizon, and evolution of conjunctive specifications. Recently, significant amount of work has been done in the input space exploration for better input generation strategies [28]. [29] introduces methods for exploiting randomness in random search. [30] talks about the input search space transformation between the unconstrained search space to the constrained search space. Moreover, the recent work also shows

the input and output signal variable importance in the formalism of specifications, [14] mentioned the vacuity performance on the falsification process.

The work of [31] explores the a non-control point parameterization scheme for input space exploration. Periodic square wave signals are used for inputs with five parameters for pulse generation. These five parameters are period, base, amplitude, delay and width. However, not all parameters were varied in input generation, base was fixed at a lower value from the range in the input signal. All input signals go from low to high and stays at high for provided width duration. In experiments, they show the importance of fixing some control point values and their impact on the performance of falsification. The work concludes the parameterization scheme requires domain expertise.

There are many recent advancements in tools and technologies that can help to implement the desired heuristics. Such tools provide many in-build features and functions, including signal generator of choice, optimizer, and many more toolboxes of machine learning based developments. These are the tools used in recent work like Breach [18], S-TaLiRo [32], ST-Lib [33], RRT-REX [13], FalStar [34], and falsify [23]. These tools are implemented based on different optimization and search strategies.

1.4 Thesis Organisation

Chapter 2 gives an overview of the black-box optimisation based falsification framework. Chapter 3 presents a formal introduction to Signal Temporal Logic. Chapter 4 explores the core internals of the Breach falsification tool on which we implemented our falsification strategy. Chapter 5 presents our proposed method for a parameterisation strategy to more efficiently explore the input signal space for falsification. Chapter 6 gives the benchmarks on which we test our proposed method. Chapter 7 gives the details of our experiments over the benchmarks, and presents the results of the experiments. Chapter 8 concludes the thesis.

Chapter 2

Overview of Black-Box Optimization based Falsification

Falsification is an inverse process of verification. In falsification, we need to find an input for which the system output violate a given specification. Falsification is a testing of the system over infinite input space from which the input is selected using some strategy such that the system does not hold desired behavior [4]. Property specification can be achieved using different extensions of temporal logic such as LTL, and MTL. Since most of the system works with real values and real-time inputs, the specifications are described in STL

A key notion for STL is that of quantitative robustness, which is a measure of how far system behavior is from violating or satisfying the specification. If the robustness is positive then system behavior satisfies the specification, otherwise it violates it. For example, the system takes one input, let us say x, and gives output y. Both x and y are real-valued signals in the domain of $[0, 100]$. The STL specification is formalized as $y \leq 50$. Then, the robustness for this specification is just an evaluation with different values that y held during the simulation of the system. A detailed explanation is given later in this Chapter.

However, the above process is experimenting with one input by simulating the system. Falsification involves changing the inputs until the negative robustness has been found. Hence, there is some sort of automatic input generation required that iteratively generates inputs from input space. There are many tools and methods available that can do this. Please note that experimenting with different inputs does not mean generating random inputs from input space.

The black-box falsification process is depicted in Figure:2.1. As discussed earlier the falsification is a process of finding an input signal such that the system behavior violates the specification. In practice, the input signal will be generated by a signal generator and given to the system. Moreover, this signal generator plays a crucial role in the generation of signals. Black-box based

optimization is a problem of minimizing the quantitative robustness value subject to input signals and in the best case, finding the negative robustness value as early as possible. The key technique that enables the optimization-based falsification is the quantitative robustness value [3, 12]. Hence, designing a signal generation of a choice that generates the input signals for the falsification of the system using some heuristic is key [11]. The heuristic in this work is the input parameterization from which an input signal will be generated and given to the system. For a corresponding input signal from the signal generator, the system will hold an output execution. The design of a signal generator of a choice is possible due to several recently developed tools that use an optimization-based falsification approach in the testing of the CPS such as Breach [18]. Other tools are used in this area such as S-TaLiRo [32], RRT-REX [13], FalStar [34], and falsify [23]. These tools are implemented using different optimization and search strategies. In this work, we have used Breach, a state-of-the-art industrial falsification tool.

The last step is to check the system behavior by evaluating the specification with the output execution. This evaluation gives a real value called quantitative robustness value that indicates a violation or satisfaction of a specification. The big box given in Figure 2.1 is a black-box for a optimizer. The optimizer tries different values of input parameterization with the aim of minimizing the robustness. In this black-box based falsification, the optimizer is not aware about any internal structure of input signals, the system or output execution. The only interfaces that the optimizer is aware of are input parameterization with their ranges and quantitative robustness value that needs to be minimized. Once the negative robustness value has found the optimizer stops searching the values for input parameterization. The rest of this chapter is detailed explanation of black-box based falsification process depicted in Figure 2.1.

2.1 Black-Box based optimization

The optimizer in the optimization is a common term that tries to optimize something. So, the first question that arises in the mind is what to optimize, which is what is often provided as an objective. Mainly, the optimizer works to either minimize or maximize the objective, given as

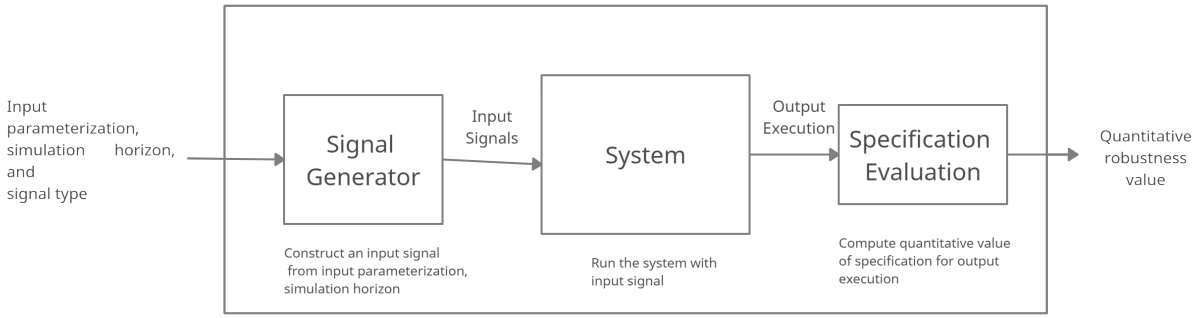

Figure 2.1: The black-box falsification process

a function. Let us say a function f that takes finite n parameters in $\mathbb{R}$. So, the parameters are $\alpha_1, \alpha_2, ...\alpha_n$ and $\alpha_i \in \mathbb{R}$. If the objective is to minimize the function f then optimization problem can be formulated as below:

$$\underset{\alpha_i \in R}{\text{minimize}}\ f(\alpha_1, \alpha_2, ..., \alpha_n) \tag{2.1}$$

where function f is defined as follows:

$$f : \mathbb{R}^n \rightarrow \mathbb{R} \tag{2.2}$$

The function f maps the n-dimensional input to a real value and the structure of f is given in Figure: 2.2.

Figure 2.2: Structure of a function f given in 2.2

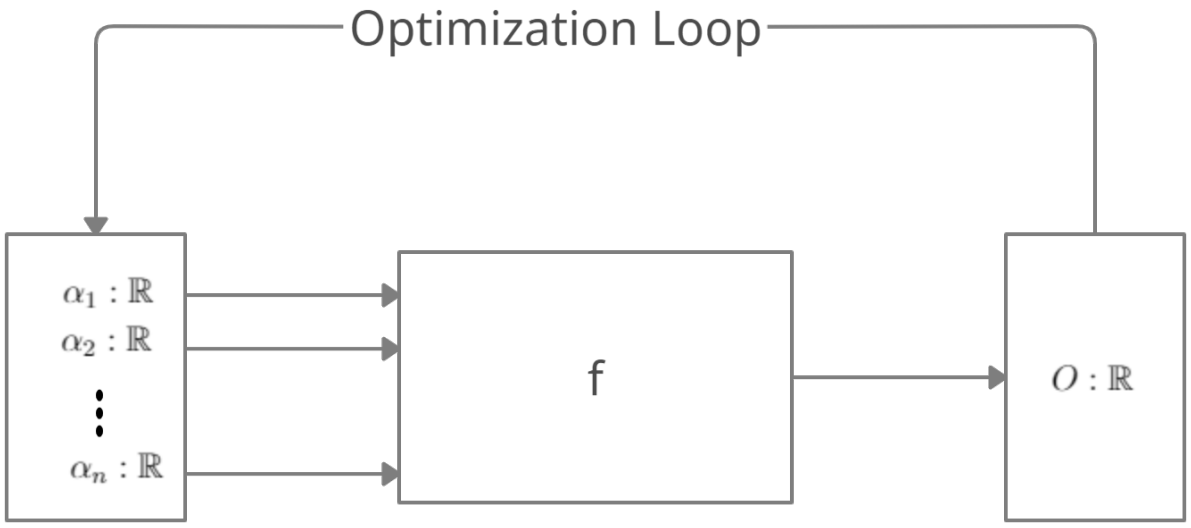

Figure 2.3: Optimization loop on function f given in 2.1

As depicted in Figure:2.3, the optimizer will try minimizing $O \in \mathbb{R}$ for different parameters. Please note that we do not know what function f looks like, we only know that function f takes n parameters and returns a single real value. Hence, for an optimizer, the function f is a black-box. Below are the terminology that makes some distinctions that are required to understand subsequent sections.

2.2 Black-Box based optimization over signals

The example given in 2.3 is black-box optimization over a set of variables ($\{\alpha_1, \alpha_2, ..., \alpha_n\}$) in $\mathbb{R}$. However, the system takes input signals generated by a signal generator, hence, the black-box optimization is over the input signals that are exercised by the system.

Parameterization:

A Parameterization is to construct an input signal from finite parameters. For example, these finite parameters are certain signal valuations placed over the simulation horizon, using linear interpolation on those placed signal valuations the full input signal will be generated. These particular signal point placements and valuations are called *control points*. The notion

of control points is defined as a certain number of signal values from the signal range are placed over the simulation time domain. The control points are sampled values from the input signal range [35]. Moreover, this is not the only example of parameterization. There are also other possible heuristics in the signal generation process. For example, the one way of parameterizing a $\cos$ signal generation would be amplitude and frequency. In Figure: 2.4, α_i^x and α_i^y where $i \in \{1, 2, ..., 5\}$ are parameters for input signal $x(t)$ and $y(t)$ respectively. Also, we may also consider some parameters which are directly required by the system but not part of signal generation. Those parameters do not contribute to the input signal, however, the system does allow changes to their values during runtime for tuning. The α_1^{param} and α_2^{param} are examples of such parameters. Moreover, the number of parameters in each signal generation is subject to the simulation horizon in our work.

Signal Variables:

Signal variables are the symbolic representation or the identifiers for a signal.

Signal:

A (discrete-time) signal is a sequence of signal valuations with respect to time. For example, for a corresponding signal variable x, a (discrete-time) signal $x(t)$ is a sequence of signal valuations $x_i : [0, T] \to \mathbb{R}$. $x(t) = x_0 x_1 x_2 .....x_{m-1}$ where x_i is a signal value at time $i * \Delta$; for Δ the sample time period. $m = T/\Delta$ is a positive integer.

Signal Valuations:

For a real-valued signal variable, signal valuation is a real value that the signal variable holds at a particular time. x_i is a signal valuation at time $i * \Delta$, for Δ the sample time period.

2.3 Signal Generator

The signal generator in figure 2.1 is a function that generates a signal over the simulation horizon from a given parameterization. Generally, a signal generator is responsible for generating

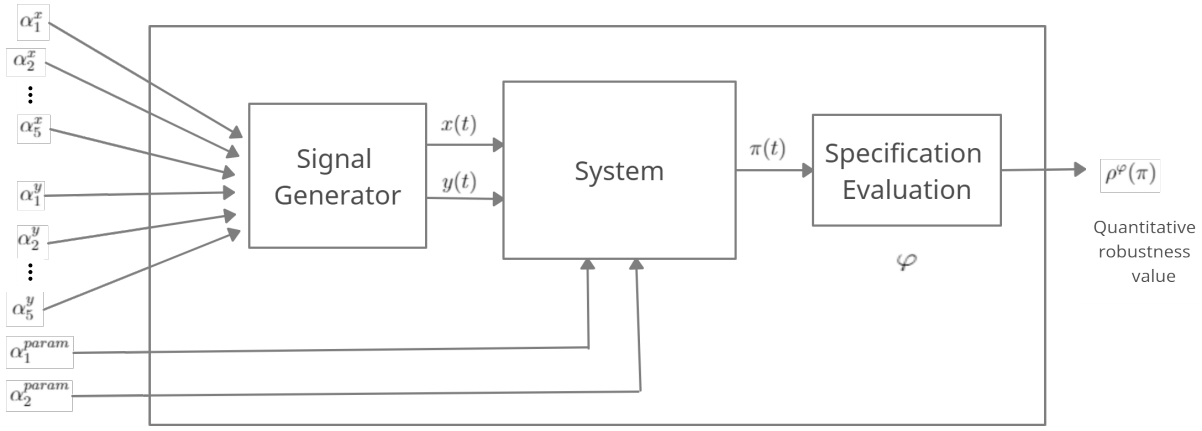

Figure 2.4: A falsification process simplified from 2.1

a valid signal that can be given to the system. A signal generator is often seen as a function of time that assigns multiple dimensional values to the signal variables with respect to time and, using linear interpolation, the full signal will be generated. For example, let us say a system has 2 input signals and 3 output signals. Then, the function of time is defined as

$$\pi_{in} : [0, T] \rightarrow \mathbb{R}^2 \tag{2.3}$$

Here, π_{in} is a trace or signal, further let's say those 2 signal variables x and y are in $[0, 100]$ and $[0, 325]$ respectively and our simulation horizon has $T = 10$ seconds. With a sampling period of 2 sec, a total 6 signal valuations will be created by π_{in}.

Now let's say those valuations given below are created at the sample time period in incremental order.

$$\pi_{in} = \{(0, 70), (50, 100), (34, 325), (76, 10), (0, 0), (10, 235)\} \tag{2.4}$$

More specifically, $\Delta = 2\ sec$, then total valuations $m = \lfloor (T/\Delta) \rfloor + 1 = \lfloor 10/2 \rfloor + 1 = 6$, the system is starting the execution from 0, so, the first control point will be placed at 0 in the simulation horizon.

For signal $x(t)$ and $y(t)$, the m valuations from π_{in} are $\{0, 50, 34, 76, 0, 10\}$ and $\{70, 100, 325, 10, 0, 235\}$, respectively.

Then, the input signals $x(t)$ and $y(t)$ will be created over those 6 signal valuations by linear interpolation.

These signals will be further given to the system. Earlier in this section, we mentioned a valid signal and that is, the signal valuations are from given signal ranges.

2.4 System

A system is a mathematical model having exact system dynamics as defined in the specifications of the system. The underlying layers of CPS are complex. Mathematically, a system that takes an input signal defined in 2.3 and gives output signal valuations in the same fashion. Let us now say that those 3 output signals have some ranges. Then for provided inputs, the system generates the outputs. A system as a function is defined as the following:

$$M : ([0, T] \to \mathbb{R}^2) \to ([0, T] \to \mathbb{R}^3) \tag{2.5}$$

2.5 Temporal Logic Specification

The specifications are the system behaviors that must be held with respect to time for a system under consideration. *Temporal logic* is a formalism that allows us to represent the specification with respect to time. For example, for a system under consideration, some phenomena will take place in some time. As given earlier in this chapter, the specification evaluation is simply checking the behavior of the system with respect to the specification. Temporal logic have many extensions such as Linear Temporal Logic (LTL) [5], Metric Temporal Logic (MTL) [8]. The Signal Temporal Logic (STL) is an extension of MTL, allowing us to develop the specification on continuous-time, real-valued signals. In this work, the specifications are represented in STL. A simple example of an STL using a temporal operator is given below:

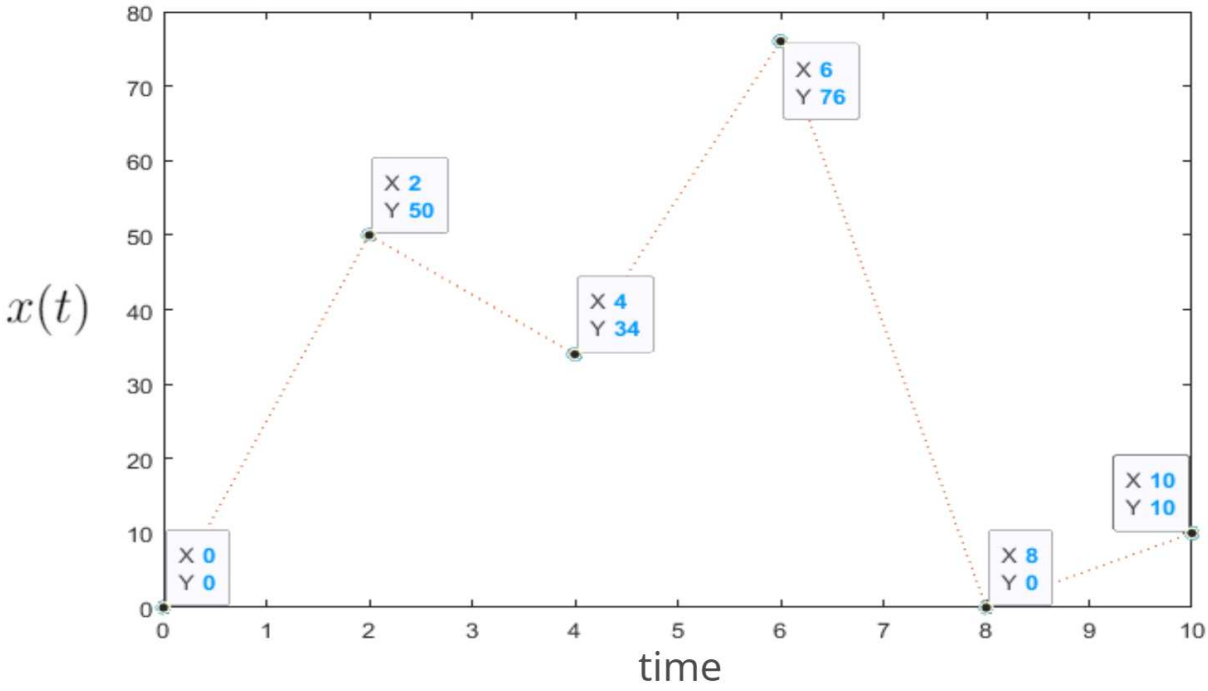

Figure 2.5: An input signal $x(t)$ using linear interpolation from the signal valuations given in equation 2.4

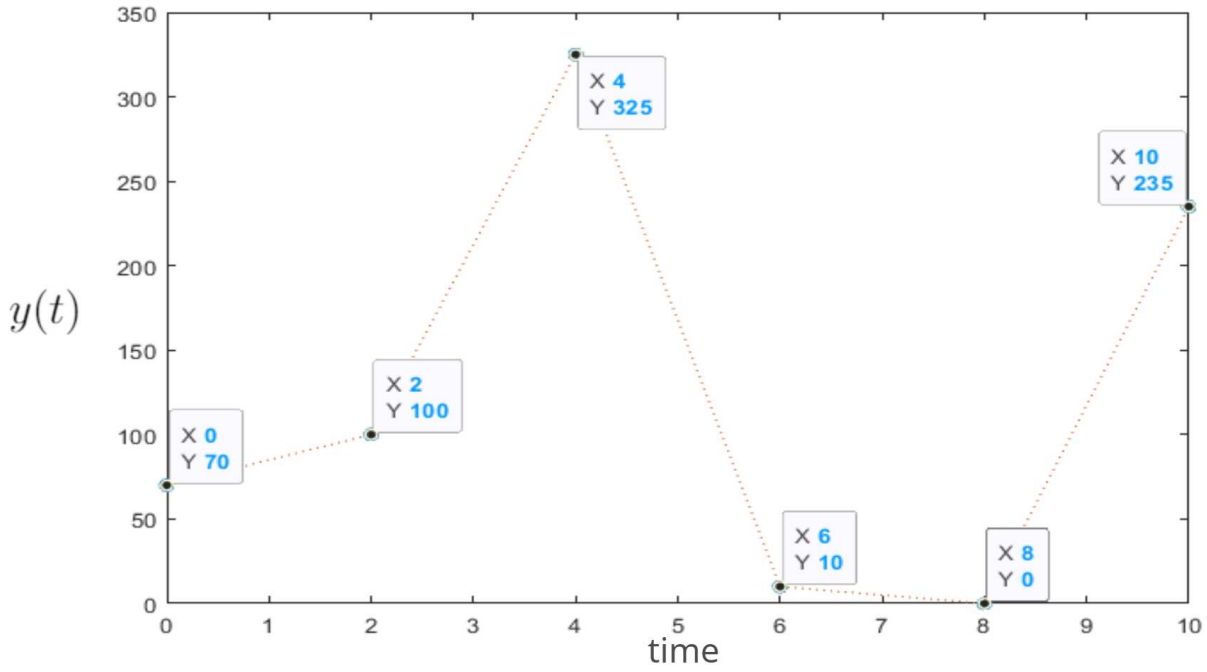

Figure 2.6: An input signal $y(t)$ using linear interpolation from the signal valuations given in equation 2.4

$$\square \left(x(t) > 32 \rightarrow \Diamond_{[0,200]} x(t) < 25\right) \tag{2.6}$$

Intuitively, if the antecedent is satisfied, then the consequent will be served. For example, in the equation 2.6 a signal $x(t)$ represents the room temperature, if the room temperature goes higher than 32 °C, then the air conditioning will start, and the room temperature will go lower than 25 °C within 200 seconds. The temporal operators $\square$ and $\Diamond$ are meant for *always* and *eventually*. Initially, the quantitative interpretation of MTL was studied in [16], [36]. An adaption to STL, where the satisfaction of an STL formula in terms of real-valued *robustness degree* indicates how far an observed signal is from satisfying or violating the specification proposed in [10]. This alternative to binary satisfaction has been seen in many of our Related Work papers. This robustness degree or robustness value or robustness indicates violation or satisfaction of the specification.

Proposition 1. *[10] For a given STL formula φ and an output execution π, if the quantitative robustness value $\rho_{\varphi}(\pi) < 0$ then the system behavior π violates the specification φ, otherwise for the quantitative robustness value $\rho_{\varphi}(\pi) > 0$ means the system behavior π satisfies the specification φ*

If the robustness value is positive, that means the system behavior satisfied the specification, otherwise, it violates it [10]. For example, consider a specification saying the temperature of a room is never above 35 °C. If, however, the temperature of the room is 37 °C then binary satisfaction is false and the robustness value is -2. If the temperature of the room is 25 °C then binary satisfaction is true and the robustness value is 10. The robustness function is given in Figure: 2.4 as $\rho^{\varphi}(\pi)$. The procedure of calculating robustness for different temporal operators for STL is given in Chapter 3.

2.6 Integration falsification and optimization

As discussed earlier the signal to the system is a function of time and using linear interpolation, the input signal will be generated. As per the falsification definition and depiction shown in Figure 2.4, this activity is all about systematically searching for an input where system behavior

violates the specifications. Hence, a higher number of input signals leads to a more complex input search space. The robustness $\rho^{\varphi}(\pi)$ is a function that needs to be minimized over the input space. So, falsification is often seen as a problem of optimization.

In this work, the system is considered a black-box, meaning we only know the interfacing of the system in terms of inputs and outputs. This black-box consideration of the system makes for easy integration of optimization and falsification. Hence, the Figure: 2.7, the big middlebox now can be given as a function to the optimizer, where the optimizer tries minimizing the robustness. Intuitively, the black-box takes signal generation parameters and returns the robustness by evaluating the specifications. If the robustness is negative then the optimizer stops minimizing the function. Then, the system is falsified. Otherwise, the robustness will help us to pick a better input parameterization. Moreover, these input parameterizations are from an input space. The input space is denoted as U. Therefore, the optimization problem can be structured as below:

$$\underset{u \in U}{\text{minimize}}\ \rho^{\varphi}(\pi)$$

where $\rho^{\varphi}(\pi)$ is a robustness value. u is an input parameterization from input space U

In Figure: 2.7, the overall black-box based falsification is explained. In this work, we have made a signal generator of a choice that is aware of the specification that we are testing. Moreover, the number of control points considered in input parameterization and their placement is critical for generating the input. So, the placement strategy of those control points in terms of sparse or dense control points over the simulation horizon is a key work in this thesis. Existing works usually take a uniform, fixed number of control points in input parameterization, which we have changed to allow any number of control points at any time point in this work.

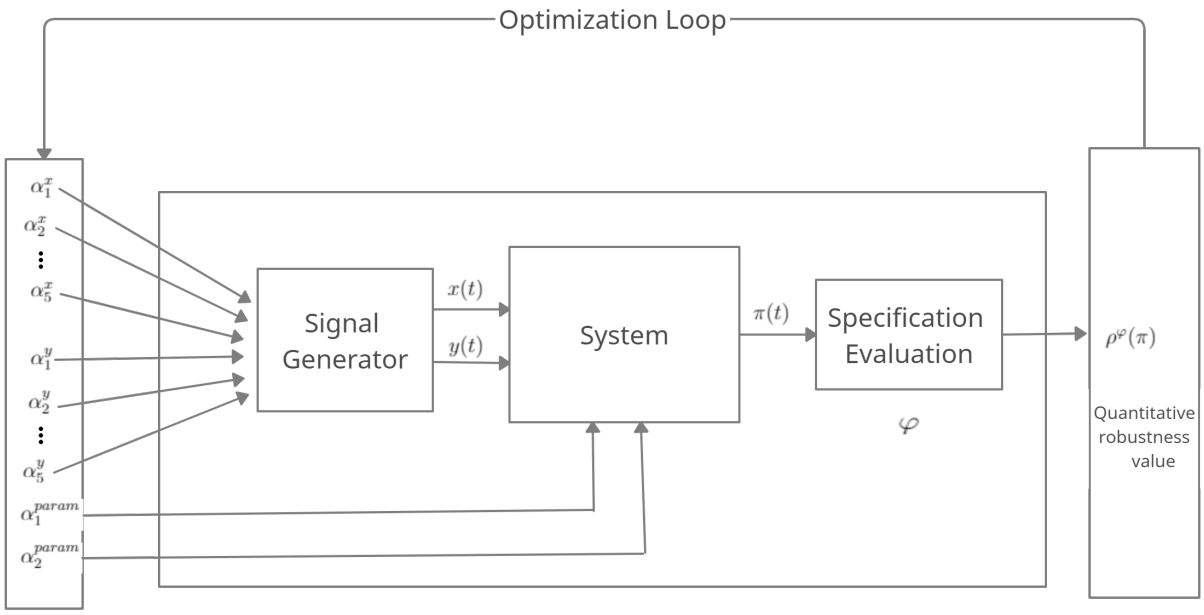

Figure 2.7: Falsification as a optimization problem

Chapter 3

Signal Temporal Logic

In this chapter we present Signal Temporal Logic, and the formal definition of the quantitative robustness function $\rho^{\varphi}(\pi)$ that is the function to b e optimized by the optimizer.

3.1 Preliminaries

Systems, signals, traces. (finite) *trace* or a *signal* $\pi : [0,T] \to \mathbb{R}^n$ of arity n is a mapping from a finite closed interval $[0,T]$ of $\mathbb{R}_+$ to $\mathbb{R}^n$. The time-domain of π is the time interval $[0,T]$ over which it is defined. We partition signals into *input* signals, and *output* signals. A (continuous-time) *system* $\mathcal{S} : \left(\mathbb{R}_+^{[\,]} \to \mathbb{R}^{n_I}\right) \to \left(\mathbb{R}_+^{[\,]} \to \mathbb{R}^{n_O}\right)$, where $\mathbb{R}_+^{[\,]}$ is the set of finite closed intervals $[0,T]$ of $\mathbb{R}_+$, transforms input signals $\pi_{\mathsf{ip}} : [0,T] \to \mathbb{R}^{n_I}$ into output traces $\pi_{\mathsf{op}} : [0,T] \to \mathbb{R}^{n_O}$ (over the same time domain). At time we refer to the input-trace output-trace combination as $\pi : [0,T] \to \mathbb{R}^{n_I+n_O}$, with π_{ip} and π_{op} being the corresponding projections; that is if $\pi_{\mathsf{ip}}(t) = (a_1,\ldots,a_{n_I})$ and $\pi_{\mathsf{op}}(t) = (b_1,\ldots,b_{n_O})$, then $\pi(t) = (a_1,\ldots,a_{n_I},b_1,\ldots,b_{n_O})$. For the signal value $\pi(t) = (a_1,\ldots,a_{n_I},b_1,\ldots,b_{n_O})$, we refer to each a_j as the value of the j-th input *signal variable*, and similarly for output signal variables. variables. We refer to the j-th dimension of the signal π as π_j, that is $\pi_j : [0,T] \to \mathbb{R}$ such that $\pi_j(t) = d_j$ where $\pi(t) = (d_1,\ldots,d_k)$.

3.2 Signal Temporal Logic

Signal Temporal Logic (STL), introduced in [9], extends Metric Interval Temporal Logic (MITL) [7] with real-time signals. We consider STL formulas with bounded-time temporal operators defined recursively according to the grammar

$$\varphi ::= \mathrm{T} \mid z + c \geq 0 \mid z + c \leq 0 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathcal{U}_{[a,b]} \varphi_2,$$

where $\top$ is the *true* predicate; z is an input or output signal variable, and c is a constant; $\varphi, \varphi_1, \varphi_2$ are STL formulas; $\neg$ and $\wedge$ are Boolean connectives that respectively indicate negation and conjunction; and $\mathcal{U}_{[a,b]}$ with $a, b \in \mathbb{R}$ such that $0 \leq a \leq b$ is the *until* operator. We define additional temporal operators in the standard way: the "eventually" operator $\Diamond_{[a,b]}\varphi$ stands for $\mathrm{T}\mathcal{U}_{[a,b]}\varphi$; and the "always" operator $\Box_{[a,b]}\varphi$ stands for $\neg\Diamond_{[a,b]}\neg\varphi$. The semantics for STL are standard (see *e.g.* [9]), we give the semantics here for convenience.

The suffix of the (input-output) signal π from time t (for $t \leq T$, the simulation horizon) is said to satisfy an STL formula φ, denoted $(\pi, t) \models \varphi$ as follows.

$(\pi, t) \models \mathrm{T}$

$(\pi, t) \models \neg\varphi$ iff $(\pi, t) \not\models \varphi$

$(\pi, t) \models \varphi_1 \vee \varphi_2$ iff $(\pi, t) \models \varphi_1$ or $(\pi, t) \models \varphi_2$

$(\pi, t) \models \varphi_1 \wedge \varphi_2$ iff $(\pi, t) \models \varphi_1$ and $(\pi, t) \models \varphi_2$

$(\pi, t) \models z_j + c \geq 0$ iff $\pi_j(t)) + c \geq 0$; where z_j corresponds to the j-th dimension of the signal π

$(\pi, t) \models z_j + c \leq 0$ iff $\pi_j(t)) + c \leq 0$; where z_j corresponds to the j-th dimension of the signal π

$(\pi, t) \models \Diamond_{[l,u]}\varphi$ iff there is some $\delta \in [l, u]$ such that $(\pi, t + \delta) \models \varphi$

$(\pi, t) \models \Box_{[l,u]}\varphi$ iff for all $\delta \in [l, u]$ we have $(\pi, t + \delta) \models \varphi$

$(\pi, t) \models \varphi_1 \mathcal{U}_{[l,u]}\varphi_2$ iff there is some $\delta \in [l, u]$ such that :

$(\pi, t + \delta) \models \varphi_2$; and for all $0 \leq \delta' < \delta$, we have $(\pi, t + \delta') \models \varphi_1$

A trace π is said to satisfy an STL formula φ, denoted $\pi \models \varphi$ if $(\pi, 0) \models \varphi$.

3.2.1 Signal Temporal Logic Robustness

In the previous section, we defined the satisfaction relation for STL. In this section, we give the quantitative robustness function for STL ρ^{φ} which quantifies the degree of satisfaction of the formula φ over trace π. For $T \in \mathbb{R}_+$ and finite-time signals on the real segment $[0, T]$, a quantitative semantics for STL can be defined similar to the quantitative semantics in [10], which deals with infinite-time signals. We assign a *robustness value* $\rho^{\varphi}(\pi, t) \in \mathbb{R} \cup \{-\infty, +\infty\}$ to every STL formula φ over a trace π at time t, where negative robustness values represent falsifications of φ:

$$
\begin{aligned}
\rho^{z_j+c\geq 0}(\pi,t) &= \pi_j(t) + c; \qquad \text{where } z_j \text{ corresponds to the } j\text{-th dimension of the signal } \pi \\
\rho^{z_j+c\leq 0}(\pi,t) &= -(\pi_j(t) + c); \qquad \text{where } z_j \text{ corresponds to the } j\text{-th dimension of the signal } \pi \\
\rho^{\neg\varphi}(\pi,t) &= -\rho^{\varphi}(\pi,t) \\
\rho^{\mathrm{T}}(\pi,t) &= +\infty \\
\rho^{\varphi_1 \wedge \varphi_2}(\pi,t) &= \min\left(\rho^{\varphi_1}(\pi,t), \rho^{\varphi_2}(\pi,t)\right) \\
\rho^{\varphi_1 \vee \varphi_2}(\pi,t) &= \max\left(\rho^{\varphi_1}(\pi,t), \rho^{\varphi_2}(\pi,t)\right) \\
\rho^{\varphi_1 \,\mathcal{U}_{[\delta_1,\delta_2]} \varphi_2}(\pi,t) &= \sup_{\tau \in (t+[\delta_1,\delta_2]) \cap [0,T]} \left(\min\left(\rho^{\varphi_2}(\pi,\tau), \inf_{s \in [t,\tau\rangle \cap [0,T]} (\rho^{\varphi_1}(\pi,s)) \right) \right) \\
\rho^{\Diamond_{[\delta_1,\delta_2]} \varphi}(\pi,t) &= \sup_{\tau \in (t+[\delta_1,\delta_2]) \cap [0,T]} \rho^{\varphi}(\pi,\tau) \\
\rho^{\Box_{[\delta_1,\delta_2]} \varphi}(\pi,t) &= \inf_{\tau \in (t+[\delta_1,\delta_2]) \cap [0,T]} \rho^{\varphi}(\pi,\tau)
\end{aligned}
$$

The robustness value $\rho^{\varphi}(\pi)$ of φ over π is defined to be $\rho^{\varphi}(\pi, 0)$.

Chapter 4

The Breach Falsification Platform

Breach is an open source state-of-the-art falsification tool widely used by the falsification community. The main features of Breach are test case generation, formal specifications monitoring, optimization-based falsification, and mining of requirements for hybrid systems [37]. There are some other tools used in recent work like S-TaLiRo [32], FALSTAR and `falsify` (deep learning approach). Moreover, Breach can be interfaced with any Simulink model. In this work, Breach version 1.8.0 has been used for experiments. Breach also has official developer documents with getting started tutorials. However, this chapter is a result of our hands-on experience with Breach including core understanding of Breach architecture, Breach modules and their interfacing that was required for implementing our developed methods.

4.1 Breach Architecture

To run the falsification problem using Breach, we primarily need 3 main classes.

- **Breach Simulink System**: This class will generate an interface on the Simulink model.
- **Falsification Problem**: This class will run a falsification problem on Breach interface.
- **Signal Generation**: Signal generation is a generic class that allows generating any other signal generation of a choice.

For the sake of simplicity, we have divided Breach tool into 3 parts for better understanding. Firstly, `Breach Simulink System`. The `Breach Simulink System` is a class used for interfacing Simulink system with the Breach. The class consists of a constructor and some methods for setting some parameters for the simulink system. Some of the methods are sim time, plot signals, print signal, etc. The `Breach Simulink System` is a derived class of `Breach System`. Secondly, The `Breach Problem` class is a derived class from `Breach Set`. The `Breach`

`Problem` class is responsible for running the optimization problem. It has 3 child classes which are `Falsification Problem`, `MaxSat Problem` and `ParamSynth Problem`. The `Falsification Problem` takes Breach Object (which is an object of `Breach Simulink System` class), and STL requirements as an input and run the optimization strategy. However, some solver options have to be set before running the optimization problem and we discussed later in this chapter. Lastly, the `signal_gen` is a base class of all signal generation classes. The `Breach Simulink System` uses the this class. The `signal_gen` class is responsible for generating different input signals as guided by the solver. The diagram is given in Figure: 4.1 with primarily 3 components of Breach that we discussed above.

4.1.1 Breach Simulink System

This section contains information about how to interface model with Breach.

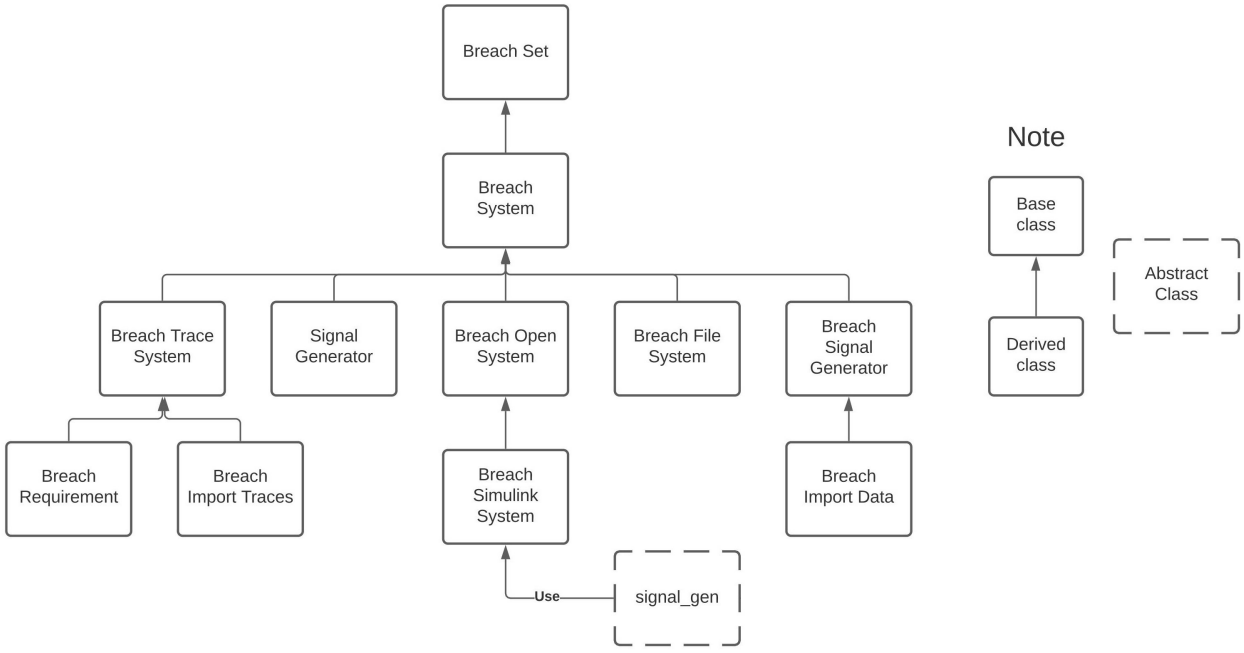

Figure 4.1: An Overview of Breach

In practice, the main class is `Breach Simulink System`. To interface the Simulink model with Breach, we provide the model name to the `Breach Simulink System` constructor. One

thing to note is that we should prepare the model first such that we log all required signals those need to be monitored. More information about the interfacing is available in the Prepare The Model section later in this chapter. `PrintAll()` and `PlotSignals()` are useful methods to check if a model interfaced correctly.

```
1 InitBreach;  % initialize Breach
2 model_name = 'AbstractFuelControl_M1'; % Model name
3 Breach_object = BreachSimulinkSystem(model_name); % Breach Object
4 Breach_object.PrintAll(); % Print signals and parameters
5 Breach_object.Sim(); % Run simulation one time
6 Breach_object.PlotSignals(); %Plot all log signals
```

Figure 4.2: Interfacing Breach

In the above code snippet, `InitBreach` at line 1 is always required to use when we run Breach. Otherwise, Breach will throw an error. `PrintAll()` at line 4 will print all parameters, signals and signals' properties as an input (model_input) or output (model_output). On the other hand, `PlotSignals()` at line 6 will plot all signals that are logged in the model. Please note that it must use the `Sim()` method before plotting signals. Moreover, please initialize any constants before you interface the model with Breach in a script. Below is an example of both methods that help to see if interfacing is correct.

It is obvious that no input signals are provided to the model in the code snippet. Because of that $Engine\_Speed$ and $Pedal\_Angle$ are constant at zero. These are where signal generation comes into the picture. More information about signal generation is given in the Signal Gen section later in this Chapter.

4.1.2 Breach Problem

The section contains information about the `Breach Problem`. That is further derived into the `Falsification Problem`. Once we have the `Breach Simulink System` object and a set of properties (STL) defined, we can run a `Falsification Problem` with the opti-

```
---  SIGNALS  ---
AF (model_output)
AFref (model_output)
Kappa (model_output)
MAF (model_output)
Omega (model_output)
controller_mode (model_output)
cyl_air (model_output)
cyl_aircharge (model_output)
cyl_fuel (model_output)
manifold_pressure (model_output)
tau_ww (model_output)
Pedal_Angle (model_input)
Engine_Speed (model_input)

-- PARAMETERS --
AF_sensor_tol=1
MAF_sensor_tol=1
fault_time=50
fuel_inj_tol=1
kappa_tol=1
ki=0.14
kp=0.04
pump_tol=1
tau_ww_tol=1
Pedal_Angle_u0=0
Engine_Speed_u0=0
```

Figure 4.3: Output of `PritnAll()` method

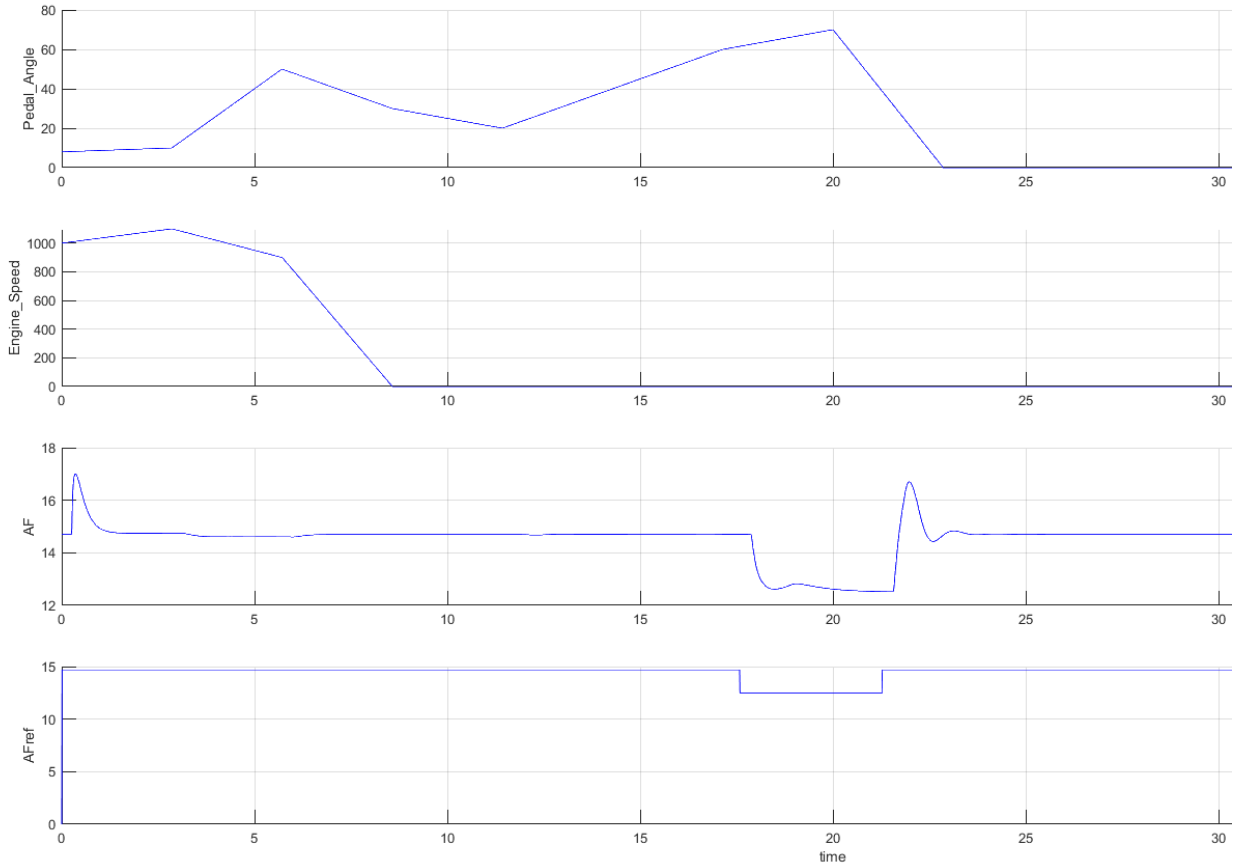

Figure 4.4: Plotting of signals using `PlotSignals()` method

mization strategy. The solver needs to be set up for optimization. Breach in-built provides some solver and below is the list of those.

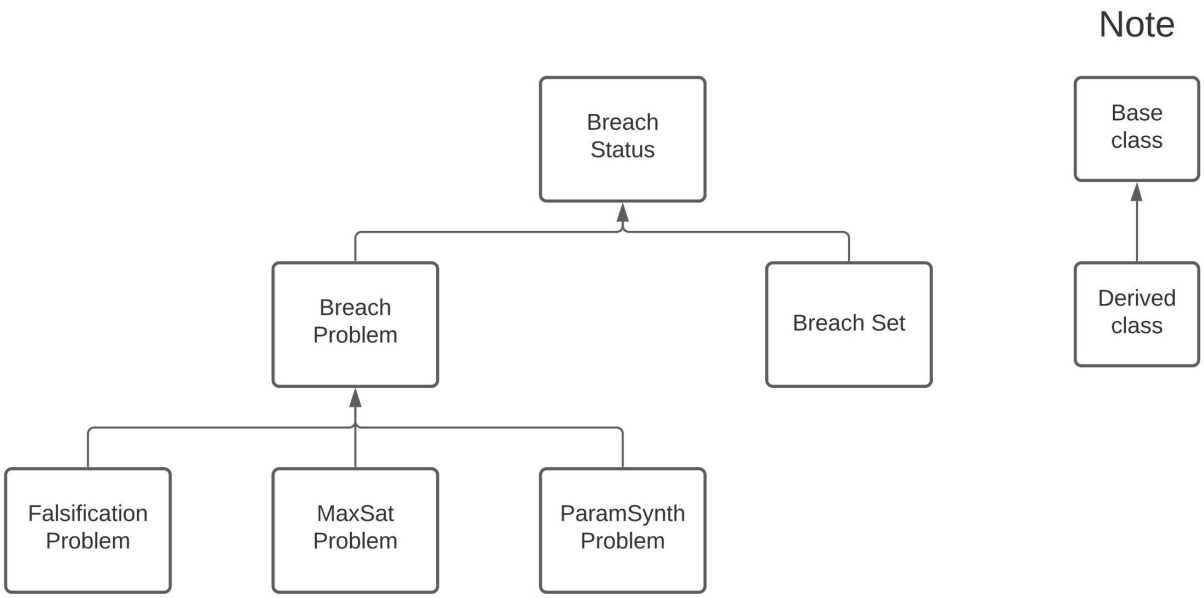

Figure 4.5: An Overview of Breach Problem

Please note that this work is only limited to `Falsification Problem`. The other two features of `Breach Problem` depicted in Figure: 4.5 are out of scope for this thesis. Moreover, we have used the CMA_ES solver in our experiments. The other solvers listed down below are just a glimpse of the solver extension that Breach allows for optimizations.

- **Quasi-random sequence solver** This is a quasi-random (systematic) sampling method. The control point valuations are initially ordered in ascending or descending order in this sampling procedure. The first individual in the order is chosen at random, with succeeding individuals being chosen at a regular interval known as a period [38]. This period is the nearest value of Nn, where N is the population size, and n is the sample size. We must set the value of n when we set the solver option.

- **Random solver** This is unbiased and hence the best method of sampling. This solver uses a simple random sampling method to sample values from input space. This method's sampling feature is that every item in the population is almost certainly included in the sample [38]. The solver allows many distributions in the setup option in Breach. These are binomial, exponential, Poisson and many more methods.

- **Covariance matrix adaptation evolution strategy (CMA_ES)** CMAES implements an Evolution Strategy with Covariance Matrix Adaptation (CMA-ES) for nonlinear function minimization [18].

- **Optimization toolbox (optimtool)** This is the Matlab optimization toolbox. The solver that default setup in Breach is 'fmincon'. Users can try other optimization algorithms as peruse. This it is a standard Matlab tool available with a graphical user interface [39].

- **Find minimum of constrained nonlinear multivariable function (fmincon)** FMINCON finds a constrained minimum of a function of several variables. FMINCON attempts to solve problems of the form where linear, nonlinear, and bounded constraints are given. FMINCON implements four different algorithms: interior point, SQP, active set, and trust region reflective [40]. This solver does allow one to choose a algorithm.

- **Find minimum of unconstrained multivariable function using derivative-free method (fminsearch)** This solver uses Nelder-Mead algorithm. For the function `f` of a variable X, the solver starts with X0 and attempts to find a local minimizer X of the function `f` [41].

- **Find minimum of function using simulated annealing algorithm (simulannealbnd)** This is also a **fminsearch** solver with simulated annealing algorithms for finding a local minimum. The only difference is that the variable X is bounded in some LB and UB and the function `f` is needed to be optimized over the range of X [42].

- **Corners** The corner point-based algorithm helps to solve the multi-objective optimization problem [43]. This solver depends on which evolutionary algorithms are used to solve the problem. The main idea in this solver is to cover the corner points in the input space.

- **Meta Heuristic** This is available under the Metaheuristics and Machine learning package in Matlab. A metaheuristic is a process of finding, generating or selecting a good heuristic that is a partial search over input space. Some partial search algorithms are particle swarm optimization, firefly, harmony search, and others [44].

- **Genetic Algorithm (ga)** This solver is about constrained optimization using genetic algorithms. Same as 'fmincon', this solver solves the linear, nonlinear, and bounded constraints defined over the variables [45].

- **Global Nelder Mead** Nelder Mead optimization works for local search. The Global Nelder Mead works for global search. The probabilistic restart makes Nelder Mead a global Nelder Mead [46].

Each solver has its own properties and those need to be acknowledged as the experiment's characteristics. All solvers behave according to the input provided to the solver. So, the one way to choose a solver is by looking at multiple definitions and picking one accordingly. We have chosen CMA-ES in out experiments and we have acknowledged some solver properties as per our need. For example, for the solver CMA-ES available in Breach [18], the range between LB and UB should be more than 3, otherwise, CMA-ES will throw 'insigma' error. However, using the solver setup feature of Breach, you can relax such constraints. Some solver information about CMA-ES available in Breach [18] is given below.

In the above listing, lines number 4, 10, 11, and 20 are important in running experiments, we should set up the values for running error-free experiments. We have explained the importance of those in Chapter 7.

```
StopFitness: '-Inf' % stop if f(xmin) < stopfitness, minimization
MaxFunEvals: 100
MaxIter: '1e3*(N+5)^2/sqrt(popsize)' % maximal number of iterations
StopFunEvals: 'Inf'  % stop after resp. evaluation, possibly resume later
StopIter: 'Inf'  % stop after resp. iteration, possibly resume later
TolX: '1e-11*max(insigma)' % stop if x-change smaller TolX
TolUpX: '1e3*max(insigma)' % stop if x-changes larger TolUpX
TolFun: '1e-12' % stop if fun-changes smaller TolFun
TolHistFun: '1e-13' % stop if back fun-changes smaller TolHistFun
StopOnStagnation: 'off'
StopOnWarnings: 'off'
StopOnEqualFunctionValues: '0'
DiffMaxChange: 'Inf'  % maximal variable change(s), can be Nx1-vector
DiffMinChange: '0'   % minimal variable change(s), can be Nx1-vector
WarnOnEqualFunctionValues: 'yes'  % 'no'=='off'==0, 'on'=='yes'==1
LBounds: [50x1 double]
UBounds: [50x1 double]
EvalParallel: 'no'  % objective function FUN accepts NxM matrix, with M>1?
EvalInitialX: 'yes'  % evaluation of initial solution
Restarts: '1'
IncPopSize: '2'  % multiplier for population size before each restart
PopSize: '(4 + floor(3*log(N)))'  % population size, lambda
ParentNumber: 'floor(popsize/2)'  % AKA mu, popsize equals lambda
RecombinationWeights: 'superlinear decrease' % or linear, or equal
DiagonalOnly: '0*(1+100*N/sqrt(popsize))+(N>=1000)'  % C is diagonal for given
    iterations, 1==always
Noise: [1x1 struct]
CMA: [1x1 struct]
Resume: 'no'   % resume former run from SaveFile
Science: 'on'  % off==do some additional (minor) problem capturing, NOT IN USE
ReadSignals: 'on'  % from file signals.par for termination, yet a stumb
Seed: 2.1168e+05
DispFinal: 'off'
DispModulo: 0
SaveVariables: 'on'
SaveFilename: 'variablescmaes.mat'  % save all variables, see SaveVariables
LogModulo: 0
LogTime: '25'   % [0:100] max. percentage of time for recording data
LogFilenamePrefix: 'outcmaes'  % files for output data
LogPlot: 'off'    % plot while running using output data files
UserData: 'for saving data/comments associated with the run'
UserDat2: ''
```

Figure 4.6: Solver option for CMA-ES

4.1.3 Signal Gen

The signal generation is responsible for generating the signal and given to the system as input through the `Breach Simulink System`. This class name is given as `signal_gen` in Breach. There are many in-built signals available in Breach. The figure given below contains all signals generation classes that are implemented from the `signal_gen` abstract class. Each signal generation class has its own constructor. It depends on signals' characteristics for what to provide as an argument to a constructor. For example, `fixed_cp_signal_gen` takes a total number of control points, values of those control points from the input range and the interpolation method. The control points are defined as a sampled value from the signal range.

Breach also provides flexibility to implement its own signal generation method. The only thing is that the generic signal generation class should be derived from the `signal_gen` abstract class. Below is an example of a generic signal generator where we we define the generic class afc_sig_gen for the Abstract Fuel Control model (more information about this model is given in next chapter). The Figure: 4.8 illustrates signal classes available in the Breach.

4.2 Prepare a model with Breach

This is a critical step before using a particular Breach. Breach comes with many capabilities, and it has an open space to extend as a user would want them to be. However, there are bare minimum modifications required to the Simulink model so that Breach can easily interface itself with the model. The manual work needed to be done still depends how well the Simulink-based model has been defined. Sometimes, Breach can pick the interfacing by itself. Latter is the thorough guide for preparing the model and some taken care points for using Breach easily.

When you prepare the model, Please make sure you log all signals that have to be monitored. This process is easy: you open a model in Simulink software; then you right-click on the signals (any connecting line between two blocks of Simulink), and log the signals. If the signal is logged then a wifi like of figure will appears on the signal. Moreover, make sure you name the signal per your choice. However, it is advisable to make a consistent name that is given to the model. For

```
classdef AFC_sig_gen < signal_gen
    properties
       lambda  % some parameter for the signal generator - this won't be visible from Breach API
    end
    methods
        % The constructor must name the signals and the parameters needed to construct them.
        function this = AFC_sig_gen(lambda)
            this.lambda = lambda;
            this.signals = {'Engine_Speed', 'Pedal_Angle'};
            this.params = {'my_param_for_Engine', 'my_param_for_Pedal'};
            this.p0 = [1000 50]; % default values
        end

        % The class must implement a method with the signature below
        function [X, time] = computeSignals(this, p, time)
            % p contains values for the parameters in the declared order
            my_param_for_Engine = p(1);
            my_param_for_Pedal = p(2);

            % Constructs signals as some function of time and parameters
            Engine_Speed = this.lambda*cos(time)+ my_param_for_Engine;
            Pedal_Angle = this.lambda*sin(time)+ my_param_for_Pedal;

            % The signals must be returned as rows of X, in the declared order
            X = [ Engine_Speed; Pedal_Angle ];
        end
    end
end
```

Figure 4.7: Generic Signal Generator for AFC

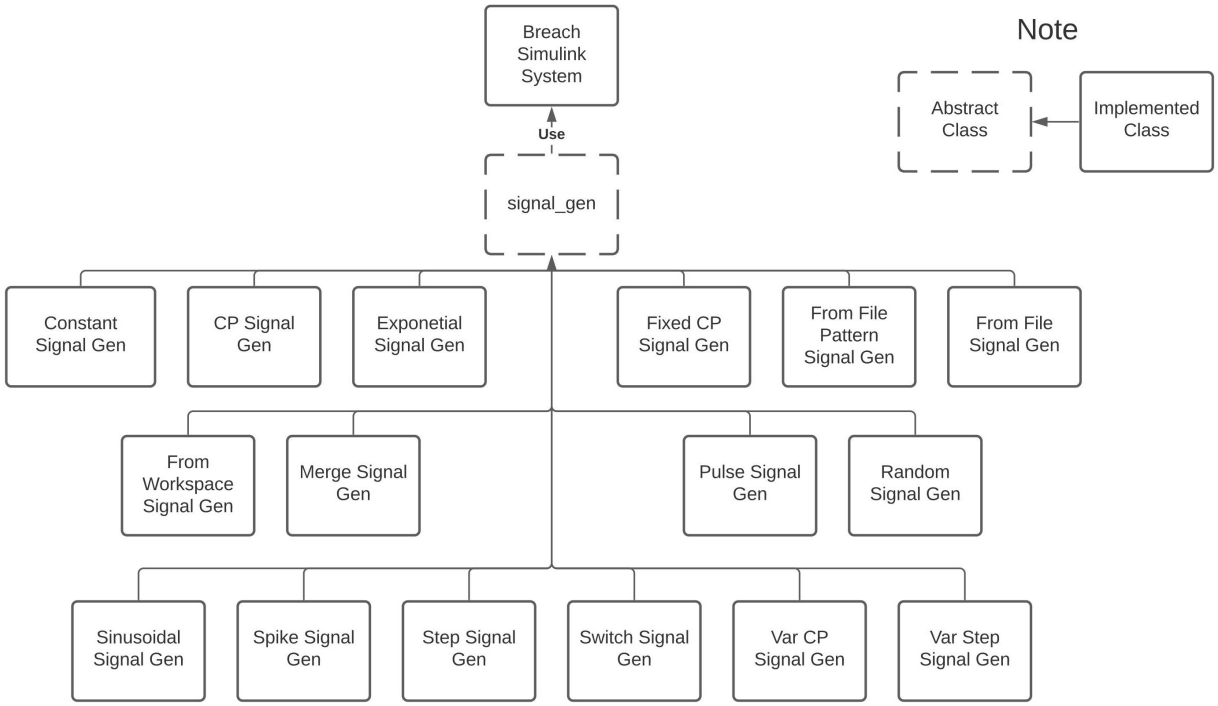

Figure 4.8: Types of signal generators available in Breach

example, a wind turbine model takes wind speed as an input. So make sure the signal that is given from the input port to any Simulink block has a wind name otherwise at a time of simulation it will not show any signal that is generated. Make sure you have saved the model after this modification. Please take a look at the figure given below. The blue highlighted connecting line depicts a signal in Simulink, after right-clicking on the blue line, the log signal option be available.

4.3 Notable points for application of Breach

- Breach follows the folder structure. The model must be under Breach/Example/Simulink/your_folder
  - A copy of just a simulink model *.mdl* or *.slx* extension file must be kept inside Breach/Ext/Models/your_model
  - You are advisable to make own *.stl* file instead of writing STL in *.m* file

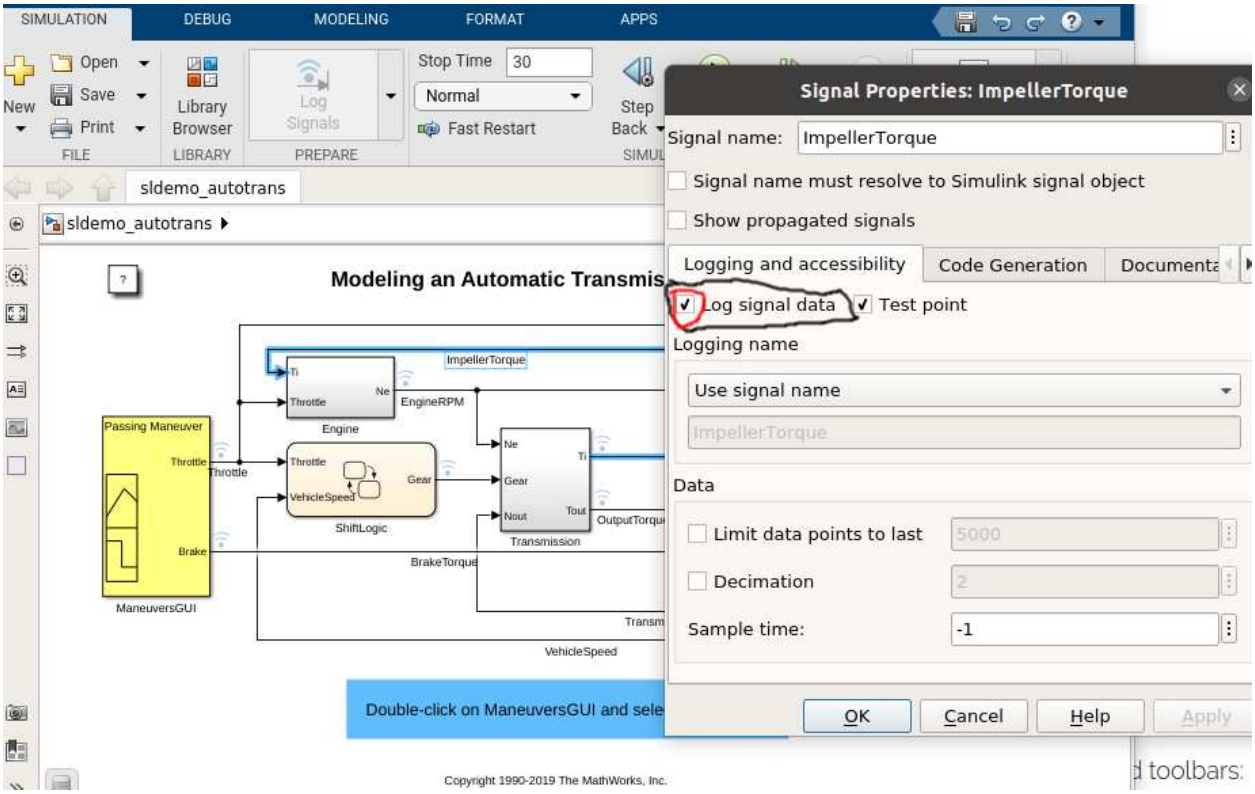

Figure 4.9: An example of Logging signal in Simulink for monitoring in Breach

  - A copy of STL file with a *.stl* extension must be kept inside Breach/Ext/Specs/your_stl_file

- Please take care about the 'addpath' function when you are playing with Breach. I would right-click on my Breach_root folder add all folders and its subfolders so that I can smoothly follow between other scripts. Otherwise, "files not found" error will occur each time you use functions from different scripts.

- Breach is a big software with so many functions call and it is highly coupled between different modules. If you want to track the function call, then you are advised to use the debugging functionality of Matlab. To do so, place some breakpoints in your script. If required, please also use the 'disp("print something")' built-in function of Matlab and print something in the different script to see the call of the functions.

- As advised in an earlier section, do not create a falsification problem directly. Please run a couple of simulations and use PrintAll() and PlotSignals() to see if your model is perfectly interfaced with Breach.

- It is recommended to use the 'disp()' function to see if everything works well. Although, you have corrected some scripts and changes may not apply. This is recommended steps because running the falsification problem takes time and you do not want to waste your time with old errors.

- Breach will generate a 'yourModelName_breach.slxc' which is a cache of your model and it is generated when you run the simulation using Breach. That file should be removed manually since it is a cache. If you are running many experiments simultaneously then remove them once you are done with the experiment.

- Each time you run the experiment makes sure you put the 'Simulink.sdi.clear' command without quotes in your main experiment file that has a *.m* extension. This step is highly recommended since Matlab with Simulink generates a large number of temporary files and those must be removed for the smooth running of an experiment.

- If you want to monitor some internal functions or intermediate data that the optimizer generates, please extend Breach by saving your intermediate data in some emph.txt files. Matlab's online community has many ways mentioned doing this.

4.4 CMA-ES solver

Breach allows the use of external black-box optimizers. We use the external CMA-ES optimization library, a state of the art black-box optimization routine [47]. In this section we discuss the options that were relevant for our experimental setup.

- **Seed**

  There is a static 0 value given to the seed option in Breach for CMA-ES solver, hence it was generating the same robustness in all experiments. To mitigate this, we explicitly made the

seed to be `sum (100*clock)` such that the seed is always unique and depends on the clock. In the solver option printed in chapter 4, the seed value was `2.1168e+05`.

- **Equal Function Valuation**

  CMA-ES was stopped evaluating objective function when robustness was getting the same values.

```
-----------------------------------
# objective   time          Robustness        Minimum Robustness
evaluation
 26        220.0       [+1.00000e+00]     (+1.00000e+00) \n
-----------------------------------
 27        229.0       [+1.00000e+00]     (+1.00000e+00) \n
-----------------------------------
 28        238.1       [+1.00000e+00]     (+1.00000e+00) \n
-----------------------------------
 29        247.3       [+1.00000e+00]     (+1.00000e+00) \n
-----------------------------------
 30        256.4       [+1.00000e+00]     (+1.00000e+00) \n
-----------------------------------
 31        265.5       [+2.00000e+00]     (+1.00000e+00) \n
-----------------------------------

*****************Alert Stopflag Occured ********************
                equalfunvals

```

Figure 4.10: equalfunvals stopflag occurred after 31 objective evaluations

Resolving this solver option does allow to set the flag to be false such that CMA-ES will not be stopped for an equal value of the objective. Specifically, this flag is `StopOnEqualFunctionValues`. There are some other flags like `StopOnWarnings` and `StopOnStagnation`. However, we have not seen any issues related to stagnation. The warnings were ignored. The next in this section also contains the steps we took to capture this behavior of CMA-ES. For that, we introduce `stopflag` in each evaluation of objective during the experiment.

- **Restart**

```
-------------------------------------
# objective   time          Robustness        Minimum Robustness
evaluation
-------------------------------------
463         7919.4      [+1.00000e+00]     (+8.02562e-01) \n
   -------------------------------------
465        7957.7       [+2.00000e+00]     (+8.02562e-01) \n
-------------------------------------
466        7976.8       [+2.00000e+00]     (+8.02562e-01) \n
-------------------------------------
467        7995.6       [+1.00000e+00]     (+8.02562e-01) \n
-------------------------------------
468        8015.0       [+1.00000e+00]     (+8.02562e-01) \n
-------------------------------------

 ******************Alert Stopflag Occured ********************
                 maxfunevals

```

Figure 4.11: maxfunevals stopflag occurred after 8000 sec time limit reached

Technically, we have not seen any issue that causes by the restart option. However, the internal working mechanism of the CMA-ES solver is not part of this work. So, without investing much time, for the sake of simplicity, we keep this `restart` flag to be *true*.

To capture the behavior of the CMA-ES, we have extended Breach tool and allowed to write the intermediate data in the .txt file. This intermediate data are robustness values for a particular objective evaluation with input values of input signals. Moreover, we explicitly capture the flag that stops the execution and we call this an alertflag. Overall, we have seen three cases where CMA-ES is stopped running experiments.

- **Maximum Time Reached**

  CMA-ES has stopped evaluation objectives when the maximum time limit for an experiment is reached. The maximum time limit is important since a solver may be stucked due to plateau, limitation of hardware on which experiments running, or allocation of shared memory storage where no more space is available to store MATLAB generated files. Below is an example of max time 8000 sec reached and the stop flag occurred.

- **equalfunvaluation**

This flag occurred due to some objective function values over time, in other words, the same robustness value. Moreover, different values of inputs were tried and robustness was the same. This specifically happened in a particular form of STL that consists of $\rightarrow$ in the formula. For example, in STL formula: $gear \geq 3 \rightarrow speed \geq 20$, this flag occurred because $gear$ could not reach 3 or more and execution was stopped.

- **# Objective Function Evaluation Reached**

  CMA-ES stopped an experiment when given objective functions are evaluated a certain number of times specified in the problem setup. For example, for AT and AFC benchmarks, each experiment should have 800 objective function evaluations, hence after 800 objective function evaluation, this flag occurred. Below is an example of when 1000 times objective function evaluated in a trial for AT benchmark.

```
-----------------------------------
# objective   time          Robustness        Minimum Robustness
evaluation
-----------------------------------
994        6282.6           [-8.10304e-01]     (-2.27383e+00) \n
-----------------------------------
995         6295.2          [-1.30548e+00]     (-2.27383e+00) \n
-----------------------------------
996         6316.2          [-1.02274e+00]     (-2.27383e+00) \n
-----------------------------------
997         6320.2          [-9.34768e-01]     (-2.27383e+00) \n
-----------------------------------
998         6332.8          [-1.22785e+00]     (-2.27383e+00) \n
-----------------------------------
999         6345.4          [-1.25231e+00]     (-2.27383e+00) \n
-----------------------------------
1000     6358.1             [-2.00307e+00]     (-2.27383e+00) \n
-----------------------------------
 ******************Alert Stopflag Occured ********************
                 maxfunevals

```

Figure 4.12: maxfunevals stopflag occurred after 1000 objective evaluation reached

Overall, to stop at `maximum time reached` and `number of objective function evaluation reached` are normal behavior of CMA-ES. However, `equalfunvaluation` is the only flag where we allowed some change in the solver option.

Chapter 5

Parameterization for Efficient Input Space Search

Recall the black-box optimization falsification framework presented in Chapter 2. Falsification is done by the use of a black-box optimizer that is searching over the space of the parameters (control points in our case). It is desirable to have a strategy which uses as few a number of control points as possible as (1) a smaller number of control points gives a n_I (where n_I is the dimension of the input signal) multiplicative benefit for the number of parameters for the optimizer to search over – this significantly improves optimizer performance; and (2) a smaller number of control points increases the interpretability, and hence usefulness, of falsifying inputs if any, as this facilitates debugging of the system under test. In order to reduce the number of control points, we use the step response concept from Control Engineering.

5.1 Utilizing the Step Response

The *step response* of a system from a given initial state is the response of the system when given a step input $u(t)$ which is 0 for $t < 0$ and A for $t \geq 0$ for some constant A for single input single output systems, with appropriate extensions for multiple input multiple output (MIMO) systems. Step response behavior is a critical part of the analysis of control systems [17], and overshoot, rise time, and settling time are part of key characteristics for behavior analysis of dynamical systems.

In our work we propose a heuristic to reduce the number of control points based on the step response, notably based on the settling time of the system. The settling time is defined as the time elapsed from when a step input is applied to when the system enters and stays within a band of the final steady state value. Matlab® provides a function `stepinfo`, that computes settling time values (in addition to the other step response entities). Note that we do not propose *monitoring* the system over a settling time period, rather a setting time period is where we place the control points and hence vary the input signal. A factor to consider is the choice of where this settling time period should be. If the settling time is t_{settle} time units, the naive choice is have the control

point placement interval as $[0, t_{\mathsf{settle}}]$. However, this choice is suboptimal, for instance when the STL property is monitoring later in the simulation period. We propose the control point placement interval as $[t_\alpha, t_\alpha + t_{\mathsf{settle}}]$, where t_α is itself a variable, hence the offset of the t_{settle}-length control point placement interval is also determined by the optimizer. In addition to placing the control points in this interval, we also need to place two additional control points, one at 0 and one at the simulation horizon T so that the input signal over the entire simulation horizon can be constructed via interpolation.

For MIMO systems, we compute the settling time for each input variable, output variable pair (holding the other input signals at some nominal value in the middle of their range), and then take the maximum of these settling times as t_{settle}. For systems with a fast response time, t_{settle} may be a small number, and placing control points in this interval may lead to input signals with high variance (compared to the simulation horizon); such input signals may not be realistic in an engineering context. If this is the case, we can place a small number, for example 8, of uniformly placed control points in a larger interval. Hence, if the chosen inter control point distance is a constant Δ_{cp}, then we place the control points in an interval of length $\Delta_{\mathsf{cp}} \cdot 8$ (excluding the control point at 0 and at the simulation horizon T). The control point number can be chosen as an initial guess about the number of control points needed to achieve sufficient input signal variability, while balancing the need to keep the number of optimization variables tractable for the optimizers. In addition, we chose to make the value of the input signal at 0 also variable as the initial signal values can have long lasting effects. The signal value after the control point placement interval was kept constant to the value at the last control point. Note that if the input signal is over n_I variables, and we have 8 control points in the control point placement interval, then such a strategy results in $n_I \cdot \Delta_{\mathsf{cp}} \cdot 9 + 1$ optimization variables: $n_I \cdot \Delta_{\mathsf{cp}} \cdot 9$ due to the $8 + 1$ control points, and an additional variable due to t_α.

5.2 Encoding Special Input Constraints

In this section we consider requirements of the form $\square\left(\theta_{\mathsf{ip}} \rightarrow \square_{[0,b]}\theta_{\mathsf{op}}\right)$, where θ_{ip} is an input variable constraint, and θ_{op} is an output variable constraint. A naive approach is to encode the entire specification in STL, and this has been the approach used previously, for example in [48]. However, some commonly occurring signals are hard to encode in temporal logics. We propose a method where the search over inputs satisfying the input constraints is guaranteed by the signal generation module (see Figure 2.7) directly, circumventing the need to encode in STL. For example, consider a formula

$$\square_{[11,50]}\big(\text{rise} \rightarrow \big(\square_{[1,5]}\ |AF - AFref| < 0.02 \cdot AFref\big)\ \big)$$

where "rise" is an input constraint which requires that the input be low for some threshold time, then go high in a very short amount of time, and then stay high for some time. Specifying such a constraint in STL is cumbersome, however, we can easily design a signal generator with control points to ensure that the generated signal satisfies this constraint, and moreover, by construction, we can directly state *when* the "rise" triggering constraint occurs. In such a case, we can modify the optimization formula to be only over the output signal variables, with the temporal constraints being different in each iteration of the optimization loop. For our example, we will use the output variable formula

$$\square_{[1+t,5+t]}\ |AF - AFref| < 0.02 \cdot AFref$$

where t is a variable denoting when the "rise" trigger happens in each iteration of the optimization loop in Figure 2.7. We will give more details in Chapter 7.

Chapter 6

Benchmarks

Our benchmarks are Simulink based models of complex industrial control systems used in falsification experiments. This section contains descriptions and STL specifications of the well-known benchmarks. We have employed the proposed method in chapter 5 in 3 known benchmarks: Automatic Transmission (AT) [49] (the modelling of AT in Matlab and Simulink given in [50]), Abstract Fuel Control (AFC) [51], and Wind Turbine (WT) [52]. These three benchmarks are widely used in many works of falsification, which include AT [53–57], AFC [13,34,54–56,58,59], and WT [48, 60] for experimental set-ups.

The main idea in this thesis work claims that either the falsification could not be achieved using the existing method and the proposed method can falsify the system or both methods find the falsification. Latter is a case where our heuristics should be able to find the falsification at the earlier stage compared to the existing method. More specifically, for benchmark AT in Table 7.1, a custom heuristics works well if falsification is found in nearly 200-300 objective function evaluations using custom heuristics and 300+ using the existing method.

However, not all specifications are falsified in the existing method. So, to make specifications difficult to falsify, we have changed constants for most specifications. The simple example is $speed \leq 120$, so 120 is a constant and that has been relaxed to 130 such that $speed \leq 130$ is difficult to falsify.

We have identified these constants by running experiments multiple times (trial and error). The deciding factor to relax the boundary in multiple trials depends on the minimum robustness found in the experiment and the number of objective function evaluation it is found at. Moreover, how quickly the robustness goes negative also helped us change the constant. We have also encapsulated the case where robustness should not be more positive after modifying the constants. Intuitively, we want the constant such that the robustness is in desired bound, neither too low (positively) nor

high. Below are the three benchmarks and their specifications that we have used in this work. Table 6.1 contains information about the benchmarks and their interface.

Table 6.1: Benchmarks with Input and Output Signals

Model	Input Signals	Input Ranges	Output Signal
AFC	Pedal_Angle	[0,100]	AF (air-to-fuel ratio)
	Engine_Speed	[900,1100]	
AT	throttle	[0,100]	speed
	brake	[0,325]	RPM
			gear
Wind Turbine	wind	[8,16]	Theta (Θ)
			Theta_d (Θ_d)
			Omega (Ω)
			Mg (M_{gd})

6.1 Automatic Transmission (AT)

This benchmark has been experimented in [11, 12, 26, 27]. The model is a closed-loop model, and it contains 3 blocks for imitating the automatic transmission in a vehicle. The main components are the model for the engine, the transmission, and the vehicle. The closed-loop engine block computes the engine's RPM (this is output signal RPM) from the input signals $throttle$ and impeller torque. Further, the transmission block computes the output torque and impeller torque from the engine's RPM (computed by the engine block), gear value (output signal $gear$) and transmission RPM. The gear status and transmission RPM are computed by a closed-loop gear block and the vehicle block, respectively. The vehicle block computes the output signal $speed$ and the transmission RPM from the output torque (coming from the transmission block) and the input signal $brake$. Meanwhile, the gear block takes vehicle speed, and commands to up-shift or down-shift the gear from the provided $throttle$ [54].

The gear calculation is done in the threshold calculation block, which takes $throttle$ and the current gear as input and decided to up-shift or down-shift using a look-up table given for gear values and applied throttle. If the current gear is let's say i then for a up-shift command, the gear

becomes $i + 1$, and $i - 1$ for a down-shift command [54]. Once the gear value is updated, the transmission block continues with further iteration in the closed-loop model. This model is mimicking the actual working mechanism of automatic transmission in the vehicle, and the falsification community widely used this model for the experimental setups [53–57].

Overall, in other words, the $throttle$ and $brake$ are the inputs and RPM, $gear$ and $speed$ are the outputs of the system. The output $gear$ takes values from $\{1, 2, 3, 4\}$, while other outputs are real-valued. The range of $throttle$ and $break$ are [0,100] and [0,325] respectively [26]. The applied $throttle$ increases RPM of the engine and in parallel it is shifting the $gear$ in increment order. As a result, the vehicle $speed$ increases. These calculations are done using some internal dynamics of the model. Meanwhile, $brake$ is reducing $speed$ of the vehicle, and internally the affected parameters shift down the $gear$ and reduce the engine RPM [11, 12, 26, 27].

Below is the list of STL properties that we have used in our experiments.

$$\varphi_1^{AT} = \square\big(gear = 3 \rightarrow (speed \geq 20)\big) \tag{6.1}$$

$$\varphi_2^{AT} = \square(speed \leq 160) \tag{6.2}$$

$$\varphi_3^{AT} = \square(RPM \leq 4800) \tag{6.3}$$

$$\varphi_4^{AT} = \square\big(gear = 4 \rightarrow (speed \geq 35)\big) \tag{6.4}$$

The original φ_2^{AT} and φ_3^{AT} are available in [12] and [49], pg-28, table(1), respectively. The φ_1^{AT} and φ_4^{AT} are taken from [12], pg-10, table(a).

6.2 Air Fuel Control (AFC)

This benchmark is widely used for experiments and evaluations in falsification community [13,34,54–56,58,59]. This is a mean value gasoline engine model. Before this model was proposed for the falsification activity, it was initially used for Bayesian statistical model checking [61]. The model contains a controller that keeps *air-to-fuel* ratio close to the *stoichiometric* ratio of 14.7

[61]. This is the ideal for the vehicle performance concerning the fuel consumption and is a key to maintaining vehicle response, carbon emission, and vehicle efficiency.

Inputs of this system model are $Pedal\_Angle$ and $Engine\_Speed$. Outputs are AF, $AFref$ and controller mode. Some non-linear and linear differential equations with a switching condition explain the simulink section of the system [61]. Furthermore, using four sensor measurements, the system estimates the correct fuel rate to reach the required *stoichiometric* ratio. The $Engine\_Speed$ and the $Pedal\_Angle$ are two that are directly related to the inputs. The remaining two sensors give critical feedback: the EGO sensor measures the amount of residual oxygen in the exhaust gas, and the MAP sensor measures the absolute pressure in the intake manifold. The *air-to-fuel* ratio is tied to the EGO value, whereas the air mass rate is related to the MAP value. If two or more sensors fail due to hardware failure, the engine is shut down since the system cannot consistently control the air-fuel ratio. [57].

Overall, the Simulink diagram is made up of numerous subsystems containing various types of continuous and discrete blocks, including look-up tables and a hybrid automaton. Hence, this model gives a concise description of the key characteristics of hybrid systems [54]. In this work, we have used the following 4 properties in the experiments. The θ is a signal variable for input $Pedal\_Angle$ in STL specifications given below.

$$\varphi_1^{AFC} = \square_{[11,50]} \left(|AF - AFref| < 0.03 \cdot AFref \right) \tag{6.5}$$

$$\varphi_2^{AFC} = \square_{[11,50]} \left(|AF - AFref| < 0.06 \cdot AFref \right) \tag{6.6}$$

For φ_1^{AFC} and φ_2^{AFC}, inputs are required to be normal mode $0 \leq \theta < 61.2$ and in power mode $61.2 \leq \theta < 81.2$, respectively.

$$\varphi_3^{AFC} = \square_{[11,50]} \big(\text{rise} \rightarrow \left(\square_{[1,5]} \, |AF - AFref| < 0.02 \cdot AFref \right) \big) \tag{6.7}$$

$$\varphi_4^{AFC} = \square_{[11,50]} \big(\text{fall} \rightarrow \left(\square_{[1,5]} \, |AF - AFref| < 0.02 \cdot AFref \right) \big) \tag{6.8}$$

For φ_3^{AFC} and φ_4^{AFC}, inputs are required to be only in normal mode $0 \leq \theta < 61.2$ In the above specifications, the "rise" triggering condition happens if the input $Pedal\_Angle$ has a low value that is constant for some time, then rises to a high value within a very short time period, and then stays at this high value again for some time. The "fall" triggering condition is defined similarly.

6.3 Wind Turbine (WT)

This model is in the early stage of a wide adoption in the falsification community [48,60]. The need of green energy is in demand due to it being the fastest-growing source of electricity, and this mechanism has great potential to fulfill that. The proposal of this model was published in [52]. The model was then officially adopted for formal analysis because of the hybrid characteristics of the model. The wind speed is the model's input, and the blade pitch angle θ, generator torque M_{gd}, rotor speed Ω, and demanded blade pitch angle θ_d are the model's outputs [48, 60]. The model is simplified with some modification for easy use in falsification. The Simulink model contains 5 subsystems: Aero-elastic subsystem, Servo-elastic subsystem, Pitch actuator subsystem, Torque controller, and Coll Pitch Controller. These three subsystems Aero-elastic subsystem, Servo-elastic subsystem, Pitch actuator subsystem are combined closed-loop system called WindTurbine that takes $wind$, θ_d, and M_{gd} as inputs and computes Ω and θ as outputs. Further, the θ and Ω are taken by the Coll Pitch Controller to compute θ_d. This θ_d and Ω are used by the Torque Controller to compute M_{gd}. Again, θ_d and Ω are fed forward to the close-loop system WindTurbine [52].

The input wind speed is constrained by $8 \leq v \leq 16$. The benchmark has 2 instances that could be configured at simulation time. The first instance is a "singlerun", which has a v= 11.2, and the second instance has the different values of v, called "Allruns". It is left to the user to configure any instances per experimental requirements. This work is only limited to the "Allruns" instance.

The original specifications are taken from the annual ARCH-COMP falsification report [48]. We have modified the constants to make properties harder to falsify.

$$\varphi_1^{WT} = \square_{[30,200]} \ (\theta \leq 14.4) \tag{6.9}$$

$$\varphi_2^{WT} = \Box_{[30,200]} \ (\Omega \leq 14.5) \tag{6.10}$$

$$\varphi_3^{WT} = \Box_{[30,195]} \ \big(\Diamond_{[0,5]} \ |\theta - \theta_d| \leq 1.6 \ \big) \tag{6.11}$$

$$\varphi_4^{WT} = \Diamond_{[190,195]} \ (|\theta - \theta_d| \leq 1.6) \tag{6.12}$$

φ_4^{WT} is an extended version φ_3^{WT}. Overall, WT is a complex model with a higher simulation horizon compared to AFC and AT, the experiments take a large amount of time. Hence, we have shortened the simulation time for WT from 630 secs to 200 secs.

Chapter 7

Experiments

7.1 Experiment Setup

We implemented our heuristics in the Breach tool. We used the three benchmarks from Chapter 6. Each benchmark model has various associated temporal logic specifications. We employed the CMA-ES black-box optimizer in Breach, and gave a fixed simulation budget, that is, the number of input signals over which the system was executed was fixed. While the high level goal is falsification, engineers are concerned with how far away the system is from failure, or the degree of failure, that is they are concerned with the actual robustness values, not just whether the robustness value is negative or not. Thus, in addition to whether falsification was achieved or not (equivalently, whether the robustness value was negative or not), we also looked at the actual quantitative values. Due to the stochastic nature of the optimizers, the resulting best robustness values (the smaller the value, the better for falsification) in repeated experiments were different. Bearing this is mind, we ran each experiment multiple times, and used two aggregate measures for comparing performance: avg – the average robustness over different experiments (with the same parameters; and avg_2 – the average robustness over the top 50% of the experiments. We used the second measure as in some benchmarks, the worst 30-40% of the robustness values were heavily skewing the results overall otherwise in the standard average.

7.1.1 Baseline and Proposed Strategy

For the baseline input signal exploration strategy, we generated signals using a uniform placement of control point over the simulation horizon, with a constant inter control point (CP) time distance. This inter CP distance was chosen heuristically. For our heuristic, we keep the inter CP distance the same, but reduced the interval in which the CPs were placed from the entire simulation horizon of the baseline, to a time interval approximately corresponding to the settling time as

proposed in Chapter 5. We performed two sets of experiments for this reduced time interval; one where the reduced interval had variable placement (the placement of the variable interval being an optimization variable), and one where the reduced interval had a fixed placement from the start of the simulation.

Two experiments had input triggering constraints, and for these the input triggering condition was directly encoded in the signal generator for our heuristic as proposed in Section 5.2. For these two experiments, for the baseline, we generated a fixed number of input signals with one rise event, where the rise event for the j-th signal occurred at j seconds; and we took the best robustness value corresponding to this set of fixed signals. Hence the baseline in this case did not involve an optimizer.

7.1.2 Experiment Parameters

CMA-ES allows setting both the maximum number of function evaluations (which is the same as the maximum number of simulations for the Simulink model), as well as the maximum time limit per experiment. Table 7.1 gives the number of experiments for each logical specification, and the maximum number of function evaluations allowed per experiment. The ballpark experiment times were 10 minutes for AT, 7 minutes for AFC, and 2 hours for WT. The experiment time was primarily determined by the simulation time for the corresponding Simulink model.

Table 7.1: A table of experimental setup for each benchmark

Benchmark	# Experiment per property	# Objective evaluation per experiment
AT	20	800
AFC	30	800
WT	20	200

As discussed in Section 5.2, the Breach tool can stop when the first negative robustness is found. We relaxed this constraint and allow to reach maximum objective evaluation regardless of

robustness. We set WT to have fewer objective evaluations compared to AT and AFC, the reason is the time taken for Simulink to compute a single simulation for WT which was about two orders of magnitude more than that for AT and AFC.

Next we give the benchmark specific parameters.

Automatic Transmission (AT) Parameters. We used a simulation horizon of 120 seconds and placed CPs at 2 second intervals. For our heuristic, we identified 34 seconds as the settling time (for a band of 15% of the final value) – the output signal given a step input was gradually increasing, and hence we chose a larger band in this case. We placed CPs at 2 second intervals in this smaller time interval, the placement of the time interval was itself an optimization variable. We also placed a CP at $t = 0$. The full signal was constructed using linear interpolation.

Air Fuel Control (AFC) Parameters. The simulation horizon was 50 seconds. The inter-CP distance was kept at 2 seconds for the formulae without input constraints. For our heuristic, the settling time from the step response was computed to be approximately 12 seconds for a 2% band. For the formulae that had the "rise" and "fall" input triggering constraints, we used CPs for the corresponding signal variable (pedal angle) as follows (the engine speed CP placement was as before). We placed 5 CPs at an inter-CP distance of 2 as before, and then we used 2 variable and 2 dummy CPs to generate the rise signal in a time interval of 1.05 seconds, and 2 more after that to generate a normal tail end of the signal. In all cases we also placed a CP at $t = 0$. Here is a an example placement of the CPs on the time line:

- One at time 0
- 4 CPs at $t_\alpha, t_\alpha + 2, t_\alpha + 2 \cdot 2, \ldots, t_\alpha + 3 \cdot 2$, with t_α being chosen by the optimizer.
- 1 CP at time $t_\alpha + 4 \cdot 2$, and another dummy CP at time $t_\alpha + 4 \cdot 2 + 0.5$, both of which took the *same* signal value in the "low" range $[0, 8.7]$.
- 1 CP at time $t_\alpha + 4 \cdot 2 + 0.55$, and another dummy CP at time $t_\alpha + 4 \cdot 2 + 1.05$, both of which took the *same* signal value in the "high" range $[40.1, 60.1]$.
- 2 normal CPs at times $t_\alpha + 5 \cdot 2 + 1.05$ and $t_\alpha + 6 \cdot 2 + 1.05$.

Thus, for pedal angle, there were in total 9 CPs whose signal values were variable (not taking the dummy CPs). An example generated input is shown in Figure 7.1.

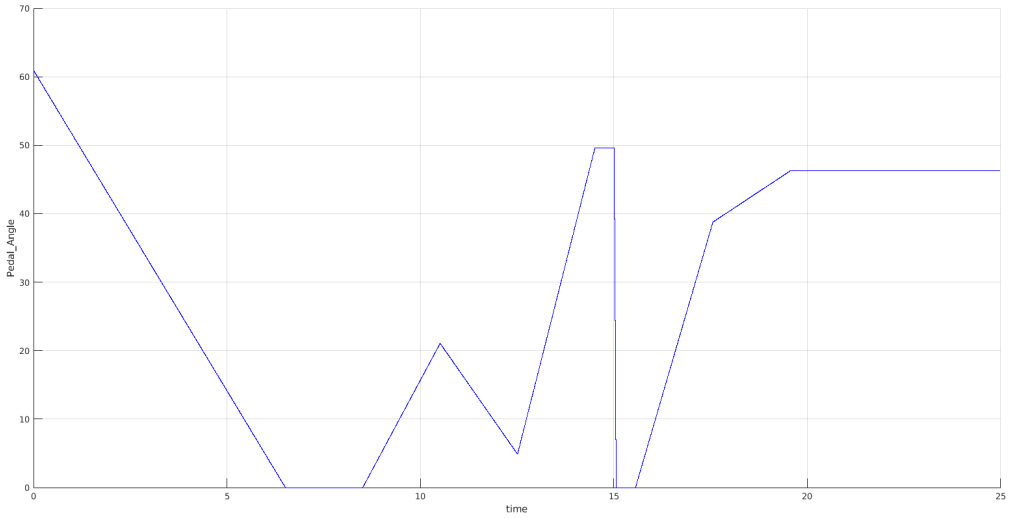

Figure 7.1: Example: $rise$ Input Signal

Wind Turbine Parameters. We used a simulation horizon of 200 seconds, with an inter-CP distance of 5 seconds. The largest settling time from the step response data was 30 seconds, and the CPs were placed in this time interval (in addition to the CP at time 0).

7.2 Results

Table 7.2 presents the falsification results of our experiments corresponding to the formulae from Chapter 6. Our proposed method varies the placement of the control point interval with the placement being governed by the optimizer. For the AT benchmarks, our proposed method performs significantly better than the baseline strategy. For φ_2^{AFC}, we also perform significantly better. For φ_1^{AFC} while we have a comparable falsification rate as the baseline, the simulation at which we falsify is somewhat higher. The other two AFC experiments had input triggering constraints, the $rise$ and $fall$ trigerring constraints, and our method with circumventing STL for the input constraints gets a perfect falsification rate compared to no falsifications in the baseline.

For WT, we perform somewhat worse in two of the cases, for the other two, both the baseline and our method were unable to falsify.

Table 7.2: Falsification rate and average number of objective evaluations where first negative was found. **FR**: % Successfully falsified; **Avg # Obj eval**: Average Number of Objective Evaluations at which first negative found

Model	Property	Baseline		Input Variation Strategy	
		FR %	Avg # Obj eval	FR %	Avg # Obj eval
AT	φ_1^{AT}	15	180	100	3
	φ_2^{AT}	0	–	100	38
	φ_3^{AT}	0	–	95	72
	φ_4^{AT}	50	364	100	5
AFC	φ_1^{AFC}	76.66	208	73.33	400
	φ_2^{AFC}	0	–	66.67	334
	φ_3^{AFC}	0	–	83.33	39
	φ_4^{AFC}	0	–	70	48
WT	φ_1^{WT}	80	79	20	102
	φ_2^{WT}	100	35	85	58
	φ_3^{WT}	0	–	0	–
	φ_4^{WT}	0	–	0	–

While the high level goal is falsification, engineers are concerned with how far away the system is from failure, or the degree of failure, that is they are concerned with the actual robustness values, not just whether the robustness value is negative or not. Table 7.3 presents the robustness values for the benchmarks (the objective was to get a low a robustness value as possible). "Variable interval placement" is for the heuristic where the placement of the control point interval was variable, and decided by the optimizer. "Fixed Interval Placement" is for the fixed placement of this interval, with the interval starting from time $t = 0$. We also normalized the robustness values. Normalization was done by multiplying the robustness value by $100/c$ where c was the constant the (error) signal (*e.g.*, the error signal $|AF - AFref|$ in φ_3^{AFC}) was compared to in the formula. Some of the experiments did not see much improvement from a floating CP interval compared to a fixed interval, some did. Our heuristic either performed better than the baseline, or was within a 5% normalized avg_2 value of the baseline, except for φ_4^{WT} for which we performed significantly worse.

Table 7.3: Results comparison for Baseline, Variable Interval Placement, and Fixed Interval Placement. avg: Normalized Average Robustness avg_2: Normalized Average Robustness of top 50%

Model	Property	Baseline		Variable Interval Placement		Fixed Interval Placement	
		avg	avg_2	avg	avg_2	avg	avg_2
AT (19 cps)	φ_1^{AT}	3.8040	2.6085	-0.8768	-0.8775	-0.8765	-0.8780
	φ_2^{AT}	1.5701	1.0184	-1.0867	-1.1090	-1.0999	-1.1288
	φ_3^{AT}	0.6341	0.5144	-4.5483	-4.5641	-4.5461	-4.5759
	φ_4^{AT}	1.2303	-0.2760	-0.5240	-0.5244	-0.5240	-0.5246
AFC (8 cps)	φ_1^{AFC}	-8.0045	-15.1020	-5.6463	-13.9909	14.8919	-4.4429
	φ_2^{AFC}	32.1542	27.0295	2.8005	-12.9252	32.0189	24.0242
(10 cps)	φ_3^{AFC}	98.9206	97.5533	-41.5658	-74.7188	–	–
	φ_4^{AFC}	98.0212	95.5147	-29.6429	-62.4059	–	–
WT (8 cps)	φ_1^{WT}	-0.0069	-0.0660	0.2521	0.0896	0.3559	0.2169
	φ_2^{WT}	-0.3676	-0.4503	-0.2386	-0.3959	-0.0904	-0.2374
	φ_3^{WT}	5.4750	3.4687	10.5675	8.2592	16.5144	12.7954
	φ_4^{WT}	43.3062	36.2000	63.1534	52.3287	100	100

Table 7.4 presents the results of our heuristic when we increased the number of control points by 2 to see whether this would have an effect on the results. Increasing the number of control points had a negative impact in some cases, and a negligible impact in others.

The results demonstrate the efficacy of out approach. As an added bonus, the reduction of the number of control points from our heuristic leads to simpler falsifying input signals.

Table 7.4: Results for varying number of CPs: avg : Normalized Average Robustness avg_2 : Normalized Average Robustness of top 50%

Model	Property	# CPs	Input Variation Strategy	
			avg	avg_2
AT	φ_1^{AT}	19	-0.8768	-0.8775
		21	-0.8764	-0.8769
	φ_2^{AT}	19	-1.0867	-1.1090
		21	-0.9989	-1.0398
	φ_3^{AT}	19	-4.5483	-4.5641
		21	-4.4986	-4.5203
	φ_4^{AT}	19	-0.5240	-0.5244
		21	-0.5238	-0.5243
AFC	φ_1^{AFC}	8	-5.6463	-13.9909
		10	-1.1565	-8.8889
	φ_2^{AFC}	8	2.8005	-12.9252
		10	-1.5986	-12.7551
	φ_3^{AFC}	10	-41.5658	-74.7188
		11	-33.9887	-63.6077
	φ_4^{AFC}	10	-29.6429	-62.4059
		11	-14.4376	-38.7619
WT	φ_1^{WT}	8	0.2521	0.0896
		10	0.2889	0.0972
	φ_2^{WT}	8	-0.2386	-0.3959
		10	-0.1386	-0.3179
	φ_3^{WT}	8	10.5675	8.2592
		10	15.1609	11.6013
	φ_4^{WT}	8	63.1534	52.3287
		10	63.7391	53.5200

Chapter 8

Conclusions

In this work we explored the falsification problem for Cyber-Physical Systems with Signal Temporal Logic specifications based on the black-box optimization framework. Efficient falsification in such frameworks depends on a good parameterization scheme for exploring the input signal space. We proposed a heuristic based on the settling time for the step response of dynamical systems as a method to reduce the number of parameters to search over, and hence as a heuristic for dimensionality reduction for input space exploration for falsification of CPS. We demonstrated the efficacy of our approach on three commonly used complex Simulink benchmarks from the CPS falsification community. If we consider the falsification rate, out of the twelve properties we tested over these benchmarks, for eleven, our proposed method either achieved a better falsification rate, or obtained a rate comparable to the baseline (where the baseline explored the input signal space via a uniform placement of control points). The largest falsification rate improvement was 100% (the baseline could not falsify in any of the instances in this case); the three largest falsification rate deteriorations were 3%, 15% and 60%. If we consider the best (lowest) robustness values obtained, for eleven properties, our heuristic either performed better in terms of the robustness value obtained, or was within 5% of the baseline. We note that our heuristic generates signals with fewer control points compared to a uniform placement (the actual reduction depends on the settling time), and hence leads to simpler signals that are more desirable for debugging purposes. For our experiment settings, the largest improvement was a reduction in control points from 61 to 19, the smallest reduction was 26 to 8 control points. Our work shows that falsification strategies could benefit from leveraging standard control systems concepts in order to obtain better parameterization strategies for more efficient exploration of the input space.

Future work directions would be to run our heuristic on more benchmarks in order to see how well our heuristic generalizes, explore other signal interpolation strategies, use different optimizers in the falsification loop, and explore other signal classes that are relevant for engineers.

Bibliography

[1] James Kapinski, Jyotirmoy V. Deshmukh, Xiaoqing Jin, Hisahiro Ito, and Ken Butts. Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques. *IEEE Control Systems Magazine*, 36(6):45–64, 2016.

[2] A. Aerts, Michel Reniers, and Mohammad Mousavi. *Model-Based Testing of Cyber-Physical Systems*, pages 287–304. 01 2017.

[3] Jyotirmoy Deshmukh and Sriram Sankaranarayanan. *Formal Techniques for Verification and Testing of Cyber-Physical Systems*, pages 69–105. 05 2019.

[4] James Kapinski, Jyotirmoy Deshmukh, Xiaoqing Jin, Hisahiro Ito, and Ken Butts. Simulation-guided approaches for verification of automotive powertrain control systems. In *2015 American Control Conference (ACC)*, pages 4086–4095, 2015.

[5] A. Pnueli. The temporal logic of programs. *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57, 1977.

[6] Kristin Y. Rozier. Survey: Linear temporal logic symbolic model checking. *Comput. Sci. Rev.*, 5(2):163–203, may 2011.

[7] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996.

[8] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2:255–299, 2005.

[9] Oded Maler and Dejan Nickovic. Monitoring properties of analog and mixed-signal circuits. *STTT*, 15(3):247–268, 2013.

[10] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In Krishnendu Chatterjee and Thomas A. Henzinger, editors, *Formal Modeling and Analysis of Timed Systems*, pages 92–106, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[11] A. Aerts, B. Tong Minh, M. R. Mousavi, and M. A. Reniers. Temporal logic falsification of cyber-physical systems: An input-signal-space optimization approach. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 214–223, 2018.

[12] Zhenya Zhang, Paolo Arcaini, and Ichiro Hasuo. Constraining counterexamples in hybrid system falsification: Penalty-based approaches. *ArXiv*, abs/2001.05107, 2020.

[13] Tommaso Dreossi, Thao Dang, Alexandre Donzé, James Kapinski, Xiaoqing Jin, and Jyotirmoy V. Deshmukh. Efficient guiding strategies for testing of temporal properties of hybrid systems. In Klaus Havelund, Gerard Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods*, pages 127–142, Cham, 2015. Springer International Publishing.

[14] Logan Mathesen, Giulia Pedrielli, and Georgios Fainekos. Efficient optimization-based falsification of cyber-physical systems with multiple conjunctive requirements. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 732–737, 2021.

[15] Houssam Abbas, Georgios Fainekos, Sriram Sankaranarayanan, Franjo Ivančić, and Aarti Gupta. Probabilistic temporal logic falsification of cyber-physical systems. *ACM Trans. Embed. Comput. Syst.*, 12(2s), may 2013.

[16] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.*, 410(42):4262–4291, September 2009.

[17] Kang-Zhi Liu and Yu Yao. *Robust Control: Theory and Applications*. Wiley Publishing, 1st edition, 2016.

[18] Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, pages 167–170, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[19] Edmund M Clarke and Paolo Zuliani. Statistical model checking for cyber-physical systems. In *International symposium on automated technology for verification and analysis*, pages 1–12. Springer, 2011.

[20] Edmund M. Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. *Model Checking and the State Explosion Problem*, pages 1–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[21] Alexandre Donzé and Oded Maler. Systematic simulation using sensitivity analysis. In Alberto Bemporad, Antonio Bicchi, and Giorgio Buttazzo, editors, *Hybrid Systems: Computation and Control*, pages 174–189, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[22] Sriram Sankaranarayanan and Georgios Fainekos. Falsification of temporal properties of hybrid systems using the cross-entropy method. In *Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '12, pages 125–134, New York, NY, USA, 2012. Association for Computing Machinery.

[23] Yoriyuki Yamagata, Shuang Liu, Takumi Akazaki, Yihai Duan, and Jianye Hao. Falsification of cyber-physical systems using deep reinforcement learning. *IEEE Transactions on Software Engineering*, 47(12):2823–2840, 2021.

[24] Thomas Ferrère, Dejan Nickovic, Alexandre Donzé, Hisahiro Ito, and James Kapinski. Interface-aware signal temporal logic. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, pages 57–66, New York, NY, USA, 2019. Association for Computing Machinery.

[25] Thomas Ball and Orna Kupferman. Vacuity in testing. In *Proceedings of the 2nd International Conference on Tests and Proofs*, TAP'08, pages 4–17, Berlin, Heidelberg, 2008. Springer-Verlag.

[26] A. Dokhanchi, S. Yaghoubi, B. Hoxha, and G. Fainekos. Vacuity aware falsification for mtl request-response specifications. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, pages 1332–1337, 2017.

[27] Masaki Waga. Falsification of cyber-physical systems with robustness-guided black-box checking. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, HSCC âĂŹ20, New York, NY, USA, 2020. Association for Computing Machinery.

[28] Fute Shang, Buhong Wang, Tengyao Li, Jiwei Tian, and Kunrui Cao. Cpfuzz: Combining fuzzing and falsification of cyber-physical systems. *IEEE Access*, 8:166951–166962, 2020.

[29] Zahra Ramezani, Koen Claessen, Nicholas Smallbone, Martin Fabian, and Knut Akesson. Testing cyber-physical systems using a line-search falsification method. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2021.

[30] Zhenya Zhang, Paolo Arcaini, and Ichiro Hasuo. Hybrid system falsification under (in)equality constraints via search space transformation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3674–3685, 2020.

[31] Zahra Ramezani, Alexandre Donzé, Martin Fabfian, and Knut Akesson. Temporal logic falsification of cyber-physical systems using input pulse generators. In Goran Frehse and Matthias Althoff, editors, *8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)*, volume 80 of *EPiC Series in Computing*, pages 195–202. EasyChair, 2021.

[32] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In Parosh Aziz Abdulla

and K. Rustan M. Leino, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 254–257, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[33] James Kapinski, Xiaoqing Jin, Jyotirmoy V. Deshmukh, Alexandre Donzé, Tomoya Yamaguchi, Hisahiro Ito, Tomoyuki Kaga, Shunsuke Kobuna, and Sanjit A. Seshia. St-lib: A library for specifying and classifying model behaviors. 2016.

[34] Zhenya Zhang, Gidon Ernst, Sean Sedwards, Paolo Arcaini, and Ichiro Hasuo. Two-layered falsification of hybrid systems guided by monte carlo tree search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2894–2905, 2018.

[35] Jyotirmoy Deshmukh, Marko Horvat, Xiaoqing Jin, Rupak Majumdar, and Vinayak S. Prabhu. Testing cyber-physical systems through bayesian optimization. *ACM Trans. Embed. Comput. Syst.*, 16(5s), September 2017.

[36] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications. In Klaus Havelund, Manuel Núñez, Grigore Roşu, and Burkhart Wolff, editors, *Formal Approaches to Software Testing and Runtime Verification*, pages 178–192, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[37] Gidon Ernst, Paolo Arcaini, Ismail Bennani, Alexandre Donzé, Georgios Fainekos, Goran Frehse, Logan Mathesen, Claudio Menghi, Giulia Pedrielli, Marc Pouzet, Shakiba Yaghoubi, Yoriyuki Yamagata, and Zhenya Zhang. Arch-comp 2020 category report: Falsification. In Goran Frehse and Matthias Althoff, editors, *ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20)*, volume 74 of *EPiC Series in Computing*, pages 140–152. EasyChair, 2020.

[38] https://www.rajgunesh.com/resources/downloads/statistics/samplingmethods.pdf.

[39] https://www.mathworks.com/products/optimization.html.

[40] https://www.mathworks.com/help/optim/ug/fmincon.html.

[41] https://www.mathworks.com/help/matlab/ref/fminsearch.html.

[42] https://www.mathworks.com/help/gads/simulannealbnd.html.

[43] Xiaobing Yu and Yiqun Lu. A corner point-based algorithm to solve constrained multi-objective optimization problems. *Applied Intelligence*, 48(9):3019–3037, sep 2018.

[44] http://www.scholarpedia.org/article/metaheuristic_optimization.

[45] https://en.wikipedia.org/wiki/geneti_algorithm.

[46] https://www.emse.fr/ leriche/gbnm_cas.pdf.

[47] Nikolaus Hansen. The CMA evolution strategy: A tutorial. *CoRR*, abs/1604.00772, 2016.

[48] Gidon Ernst, Paolo Arcaini, Alexandre Donzé, Georgios Fainekos, Logan Mathesen, Giulia Pedrielli, Shakiba Yaghoubi, Yoriyuki Yamagata, and Zhenya Zhang. Arch-comp 2019 category report: Falsification. In Goran Frehse and Matthias Althoff, editors, *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 61 of *EPiC Series in Computing*, pages 129–140. EasyChair, 2019.

[49] Bardh Hoxha, Houssam Abbas, and Georgios Fainekos. Benchmarks for temporal logic requirements for automotive systems. In Goran Frehse and Matthias Althoff, editors, *ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems*, volume 34 of *EPiC Series in Computing*, pages 25–30. EasyChair, 2015.

[50] MathWorks. https://www.mathworks.com/videos/modeling-an-automatic-transmission-and-controller-68823.html.

[51] Xiaoqing Jin, Jyotirmoy V. Deshmukh, James Kapinski, Koichi Ueda, and Ken Butts. Powertrain control verification benchmark. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, HSCC '14, pages 253–262, New York, NY, USA, 2014. Association for Computing Machinery.

[52] Simone Schuler, Fabiano Daher Adegas, and Adolfo Anta. Hybrid modelling of a wind turbine. In Goran Frehse and Matthias Althoff, editors, *ARCH16. 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems*, volume 43 of *EPiC Series in Computing*, pages 18–26. EasyChair, 2017.

[53] Hanghang Tong, Christos Faloutsos, and Jia-yu Pan. Fast random walk with restart and its applications. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 613–622, 2006.

[54] Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V. Deshmukh, and Sanjit A. Seshia. Mining requirements from closed-loop control models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(11):1704–1717, 2015.

[55] Takumi Akazaki and Ichiro Hasuo. Time robustness in mtl and expressivity in hybrid system falsification. In Daniel Kroening and Corina S. Păsăreanu, editors, *Computer Aided Verification*, pages 356–374, Cham, 2015. Springer International Publishing.

[56] Jyotirmoy Deshmukh, Alexandre Donzé, Shromona Ghosh, Xiaoqing Jin, Garvit Juniwal, and Sanjit Seshia. Robust online monitoring of signal temporal logic. *Formal Methods in System Design*, 51:1–26, 08 2017.

[57] Giuseppe Bombara, Cristian-Ioan Vasile, Francisco Penedo, Hirotoshi Yasuoka, and Calin Belta. A decision tree approach to data classification using signal temporal logic. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, HSCC '16, pages 1–10, New York, NY, USA, 2016. Association for Computing Machinery.

[58] Chuchu Fan, Bolun Qi, Sayan Mitra, Mahesh Viswanathan, and Parasara Sridhar Duggirala. Automatic reachability analysis for nonlinear hybrid models with c2e2. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification*, pages 531–538, Cham, 2016. Springer International Publishing.

[59] Jyotirmoy Deshmukh, Xiaoqing Jin, James Kapinski, and Oded Maler. Stochastic local search for falsification of hybrid systems. In Bernd Finkbeiner, Geguang Pu, and Lijun Zhang, editors, *Automated Technology for Verification and Analysis*, pages 500–517, Cham, 2015. Springer International Publishing.

[60] Sota Sato, Masaki Waga, and Ichiro Hasuo. Constrained optimization for hybrid system falsification and application to conjunctive synthesis. *IFAC-PapersOnLine*, 54(5):217–222, 2021.

[61] Paolo Zuliani, André Platzer, and Edmund M. Clarke. Bayesian statistical model checking with application to simulink/stateflow verification. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '10, pages 243–252, New York, NY, USA, 2010. Association for Computing Machinery.