Hierarchical Optimization for Hybrid System Falsification

by

Zhenya Zhang

Dissertation

submitted to the Department of Informatics

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

The Graduate University for Advanced Studies, SOKENDAI

September 2020

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Ichiro Hasuo, for his continuous supervisions and supports to me. Every time when I arrive at bottlenecks in my research, it is always his patient guidance and encouragement that help me overcome them. Also I am grateful for his help and care in my daily life. I really enjoy the three years living here.

I would like to thank Paolo Arcaini for his detailed guidance all along. He taught me so much, both in science and in life. He is always so patient to help me review my writings and correct my mistakes. I would like to thank Gidon Ernst and Sean Sedwards for their guidance in my current research topic. Also thank other colleagues and friends in the laboratory. There is so much happy time in the laboratory life. I am proud of being a member of this team.

I would like to thank my families for their supports all along. They always warm my heart and encourage me when I am frustrated. Thanks for always giving me suggestions and supporting the choices made by myself.

This work is supported by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST; it is also supported by Grant-in-Aid for JSPS Fellows No. 19J15218.

Abstract

Cyber-Physical Systems (CPS) are physical systems integrated with digital control. Quality assurance of CPS is a problem of great importance, but it is also challenging due to the hybrid nature of CPS in which both discrete and continuous dynamics exist. While formal verification suffers from a severe scalability issue, stochastic optimization-based falsification, which aims to find a counterexample input to refute the system specification, is a viable approach to solving the problem. This technique turns the problem into an optimization one based on the robust semantics of the specification language, namely Signal Temporal Logic (STL), and employs stochastic optimization algorithms to search for an answer.

Although falsification has proved to be an effective approach, many methodological weaknesses are still there, limiting its usage in practice. In this work, we address three important ones, namely, improper balance between exploration and exploitation during search, superposing robustness values from signals of different scales in STL robust semantics, and inability of handling input constraints.

In order to tackle those problems, we propose a general two-layered hierarchical optimization framework, in which a problem is firstly decomposed into a set of sub-problems, and then solved via a two-layered methodology: the top layer selects a sub-problem as the next step to proceed based on the information given by the bottom layer; the bottom layer performs numerical optimization with the selected sub-problem and returns feedback to the top layer. In this way, the two layers collaborate with each other and work together to solve the problem.

This framework is instantiated to three techniques, each addressing one specific weakness in the existing falsification workflow. In summary, these techniques are:

- A two-layered optimization framework that combines Monte Carlo Tree Search and hill-climbing optimization for balancing exploration and exploitation during the search;

- A framework for falsifying safety properties with Boolean connectives via the introduction of the Multi-Armed Bandit model;
- A search space transformation approach integrated with the Multi-Armed Bandit model for handling constraints on input signals.

Experimental results show the effectiveness of our approaches. Together, these approaches enhanced the existing falsification technique. Moreover, these approaches also exemplify our hierarchical optimization framework, which is potentially applicable in other contexts.

Contents

List of Figures

List of Tables

1 Introduction: Quality Assurance of Cyber-Physical Systems

1.1 Cyber-Physical Systems

Cyber-Physical Systems (CPS) are physical systems with digital control of computers. With the rapid development of computer technology, especially artificial intelligence these years, the collaboration between physical and digital components of systems is getting much closer than ever before, giving birth to many applications of CPS. These applications are facilitating people's daily life: for example, domestic robots are saving people's time from household duties; autonomous driving cars are giving people easier and safer driving experiences; unmanned aerial vehicles (UAV) are being used for photography, package delivery, or even fire fighting; some advanced medical devices, such as artificial pancreas[1], are helping people fight with diseases. Actually, it

[1]https://en.wikipedia.org/wiki/Artificial_pancreas

is already such an era: machines are equipped with “brain”s, so they behave not only mechanically, but also intelligently.

CPS are usually referred to as *hybrid systems* in the abstract sense: they combine discrete and continuous dynamics together in a single system. Discrete dynamics exhibits stepwise behaviors, and they are empowered by computer software. Continuous dynamics exhibits continuous behaviors, and they are composed of traditional physical systems. The roles that computer software play are usually *controlling* and *monitoring*: they monitor the behaviors of mechanical systems, and send commands to control them based on a set of business logical rules. Mechanical systems execute the commands, and also feedback information to the computer system. Therefore, they do not work independently, but interact with each other actively.

1.2 Quality Assurance of CPS

Quality assurance of CPS products is a topic of great significance. On the one hand, many of the CPS applications are life-critical; system failures can cause unacceptable consequences. For example, many CPS serve as transportation tools, such as airplane, train, car, etc. System faults of these applications can cause huge financial damages or even loss of human lives. Moreover, software systems are prone to error; it is still a challenging job to guarantee the quality of software products. In the history, there are many tragedies due to system failures caused by software errors, such as the well-known Apollo 13 Accident[1].

Verifying software systems is a difficult problem, and it is still one of the major research topics in computer science. CPS, as a family of more complicated systems, pose many new challenges in their quality assurance. We list some of the major concerns below:

- Infinite state space. Due to the existence of physical components, the system state space of CPS is usually infinite. This feature poses challenges in modeling and reasoning about the system. New formalisms have been invented for modeling hybrid systems, such as hybrid automata [1], Simulink[2], etc. However,

[1]https://nssdc.gsfc.nasa.gov/planetary/lunar/ap13acc.html
[2]https://www.mathworks.com/products/simulink.html

verification of such models is usually undecidable.

- Complex dynamics. White-box methods based on rigorous mathematical proofs are highly reliable approaches to quality assurance. However, the dynamics equations of CPS are typically so complicated that even if they are known, there is still a barrier to a tractable way of verifying them.

- Scalability. Scalability refers to the ability of extending reasoning techniques that are applicable to small models to systems of much larger magnitude. As real-world CPS products are usually gigantic and complex, developing techniques that are scalable remains a big challenge.

- Black-box components. Nowadays, many CPS products are produced by commercialized collaborations, so they contain black-box components. A black box component indicates that the internal dynamics of the component are not accessible. One reason is that those components include confidential techniques; also modern CPS are produced by interdisciplinary collaborations, so one individual is not supposed to hold all the domain expertise. As a consequence, black box gives rise to more uncertainties and more uninterpretabililities compared to white-box models.

- Environmental uncertainty. In practice environmental noises can be introduced into the system execution in many ways, and the uncertainty brought by them is an important issue that needs to be considered. Even worse, they are typically difficult to model or handle. Firstly, it is hard to precisely describe them with a formal model; secondly, taking them into consideration during reasoning is also a complicated work.

Specification In quality assurance, properties related to safety, user experience, industrial international standards etc. are referred to as *specifications*; violation of a specification is undesired. Quality assurance consists in guaranteeing that all the system behaviors satisfy the specification; however, since rigorous guarantee is most of the times infeasible, some quality assurance methods try to show the existence of a system behavior that violates the specification.

Specifications can be described in natural language on some informal occasions. However, natural languages usually suffer from the ambiguity issue: the same sentence can be interpreted completely differently by different people. Moreover, natural language processing is a huge burden for computer systems; it is unreasonable to involve this process into the loop of quality assurance. Therefore, a formal language that is easily understandable by computer systems is needed. This is not a trivial task. There has been a lot of research attention paid on building up such formal language. Many of the languages are in the form of mathematical logic, such as propositional logic, first-order logic, etc. Specifically, in the context of reasoning about temporal properties, people usually use *temporal logics*. They feature the use of temporal operators, such as *always* (or $\Box$), *eventually* (or $\Diamond$)—these allow reasoning about events of *presence* or *persistence* on a timeline.

In the next section, we show a usage scenario about a quality assurance task of an automotive system. The model is a hybrid system, and the specification is a temporal property about safety.

1.2.1 A Usage Scenario: an Automotive System

In this section, we present a usage scenario which engineers may encounter in practice. Suppose an engineer is designing a car in a Simulink development environment (an introduction to Simulink is given in §1.3). As Fig. 1.1 shows, the car[1] is composed of many blocks; it accepts *throttle* and *brake* as inputs, and outputs *rounds per minute (RPM)*, *speed*, *gear*, etc. Note that this is an automatic transmission system, so *gear* is an output here.

Now a task comes to the engineer: he/she would like this car to satisfy such a safety property—whenever the *gear* is 4 (a relatively high *gear* value), the *speed* should never be too low, say lower than 35 km/h. This is a reasonable requirement for the car because, once the car runs at a high *gear* with a low *speed*, it will harm some components and even threaten the safety of the entire system. However, this is not a trivial problem for the engineer, as the car consists of so many blocks that operate according to complicated dynamics. It sounds hard to give a mathematical proof to

[1]This model is from: https://www.mathworks.com/help/simulink/slref/modeling-an-automatic-transmission-controller.html

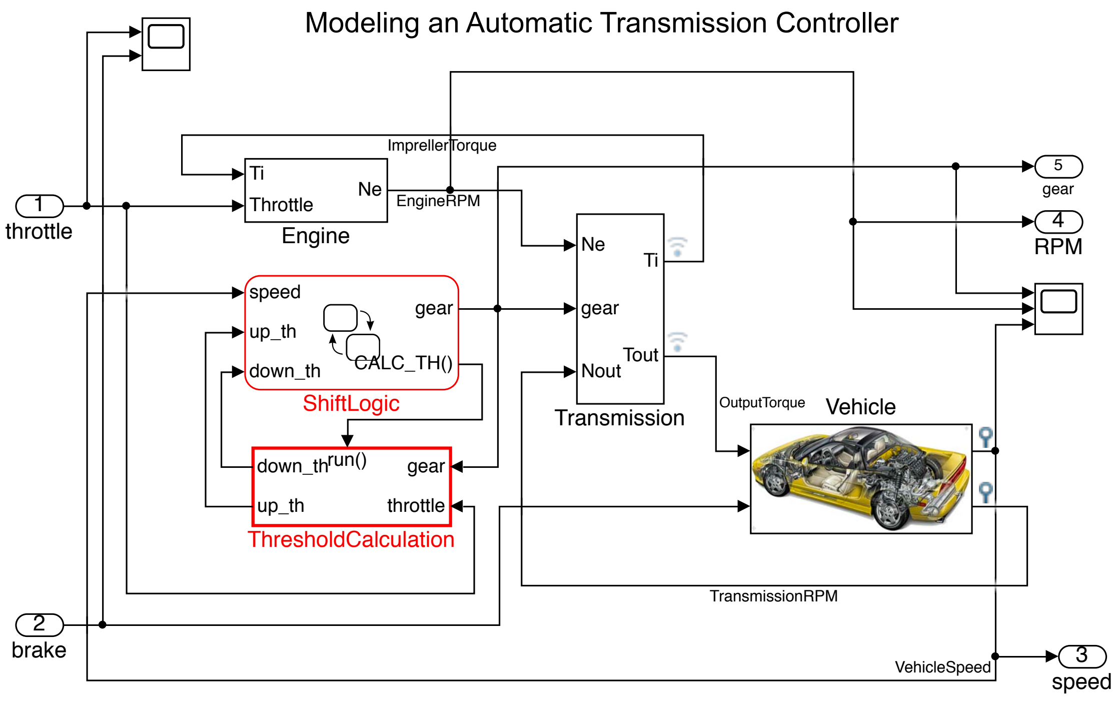


Figure 1.1: Simulink model: automatic transmission

rigorously guarantee that property. As a trial, the engineer comes up with several instances of *throttle* and *brake*. He/she gives these instances as inputs to the car, gets the car model run, and collects the logs of output signals. Then he/she does an analysis on output data. Through the observation, the car seems not violating that property. Nevertheless, the engineer is not convinced, because the number of test cases is too small. Here the problem arises—is there a good way to determine if the car satisfies that property? It will be helpful either saying yes with a proof, or giving an instance of *throttle* and *brake* values under which the car violates the property.

For discrete systems, typical approaches to quality assurance include *verification* and *testing*. In the following sections, we discuss on the technical details of these approaches and the problems that arise from applying them to CPS applications.

1.2.2 Verification

Formal verification is a quality assurance approach that uses mathematical formal methods to guarantee the correctness of systems. It has been widely used in applications such as cryptographic protocols, digital circuits, etc. The advantage of formal verification is that it gives rigorous proofs to the properties of interest based on abstract mathematical models, and thus the result is reliable.

Model checking [2] is a major verification approach to quality assurance. This term is usually connected to *reachability analysis*, a method that aims to determine if a goal state (usually an unsafe state) can be reached by the system. A discrete system, such as a software program, can be modeled as an automaton. The negation of the system specification in the form of temporal logics, which represents the unexpected situation, can be transformed to another automaton. Model checking solves the problem in the following way: firstly it combines the two automata; then it checks, starting from the initial states, if the combined automaton can reach the goal state. This approach is known to be rigorous, but it does not always scale very well.

In the context of hybrid systems, *hybrid automata* [1] are a widely-used formalism. Fig. 1.2 gives an example of a hybrid automaton (taken from [3]), in which the dynamics of a bouncing ball is formally represented. Here x_1 denotes the vertical position of the ball, and x_2 denotes the vertical velocity of the ball. It has two types of state transitions: x_1 and x_2 flow continuously when the ball is in the air; the system jumps to the next state when the ball impacts against the ground. This is an elegant way to formalize hybrid systems, but model checking, even on quite simple hybrid automata, is usually undecidable due to the presence of continuous dynamics.

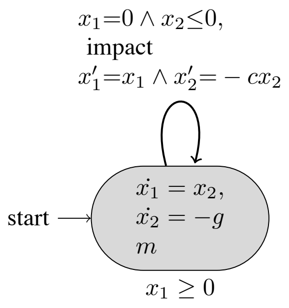

$x_1{=}0 \wedge x_2{\leq}0$,
impact
$x_1'{=}x_1 \wedge x_2'{=}-cx_2$
start
$\dot{x_1} = x_2$,
$\dot{x_2} = -g$
m
$x_1 \geq 0$

Figure 1.2: Hybrid automaton: a bouncing ball

Timed automata [4] are a sub-class of hybrid automata. They capture the temporal features of systems using several clocks. Each clock flows as the time elapses, and when they reach certain conditions, one state jumps to the next one. Timed automata are mainly used to model and analyze real-time systems. They are severe restrictions of hybrid automata, for which reachability is decidable [4]. There have been several successful tools for verification of timed automata, such as UPPAAL [5], Kronos [6], IMITATOR [7], etc.

Other verification techniques for software systems include *theorem proving*, which also has a counterpart technique in hybrid systems. For example, KeYmaera [8] and KeYmaera (X) [9] are automated and interactive theorem provers. KeYmaera (X) supports a first-order logic, named differential dynamic logic (DL) [10, 11], as the specification language. Hybrid systems are expressed in the form of *hybrid programs* [10, 11], and the tools can automatically prove if the system satisfies a given property in DL.

Extension of programming logics by *non-standard analysis* is studied for reasoning about hybrid systems in [12–14]. It adds a notation of *infinitesimal* to the traditional programming language, enabling it to express continuous dynamics and thus describe the executions of hybrid systems. Then, it develops the counterpart of *Hoare Logic* in that context for reasoning about the correctness of the proposed formalism.

Through the aforementioned instances, we can see that, although verification techniques for hybrid systems give elegant formalization and rigorous guarantees for the problem, they also inherit the weaknesses from traditional verification techniques: firstly, scalability is still a major problem, therefore, the class of models and specifications that can be practically handled is limited; secondly, usually they do not handle environmental uncertainty well, restricting their usage to the theory level.

1.2.3 Testing

Testing is a dual process to verification. Instead of verifying that a system satisfies a specification, testing aims to find a counterexample showing that the system is possible to violate the specification. In this way, testing exposes the fact that the system is able to behave unexpectedly, so it requires repairing or redesigning. Verification is usually difficult (if not impossible) to many applications in the context of quality assurance of CPS; on these occasions, testing is a viable approach to understanding the behaviors of the system. Indeed, searching for one specific case is much easier than exploring the whole system thoroughly.

Testing usually relies on system executions. However, directly testing on the real system is expensive and inefficient. For instance, if we test a real car on the road, it is not only time and resource consuming, but it may also lead to catastrophic accidents. Therefore, engineers usually develop a model and run simulations on that one instead [15]. A widely-used modeling tool for CPS is Simulink, such as the example shown in Fig 1.1. It provides a graphical interface that allows engineers to quickly prototype systems. It is based on MATLAB, and thus it is convenient to integrate other toolboxes of MATLAB. Other simulation environments include some game engine applications, such as Unreal Engine[1], especially for automated driving systems. See a related work in [16]. Although using those simulators can be costly, they provide better interfaces and more powerful functionalities that make simulations closer to the real world.

In order to test a system against specifications, *search-based testing* is a commonly-used technique. It usually employs optimization algorithms, such as genetic algorithms, to search for test cases that violate the given property, guided by a well-defined fitness function. This technique has been widely applied in many applications, and there have been a lot of research efforts put in that direction. See [17,18] for surveys. Search-based testing is also considered as an approach that is well-suited for testing CPS: one reason is that it works well with black-box models; another reason is that it is more efficient compared to naive random sampling, and thus it saves a lot of simulation costs. In fact, the theme of this work, namely the falsification technique, is also considered as an instance of search-based testing.

1.3 Optimization-Based Falsification

As we mentioned, falsification is a search-based testing approach for quality assurance of CPS products. Compared to verification techniques, it needs no exploration of the whole system states, but instead pursues one counterexample to refute the system specification. We firstly introduce a basic problem setting of falsification that has been considered in many literature [19–30].

Model Falsification problem concerns about a model, i.e., the object under test, such as a car. We usually treat the model as a black box; in other words, we do not access its internal dynamics, but the only way we evaluate it is via providing an input signal and observing the corresponding output signal. We have elaborated on the reason in §1.2.

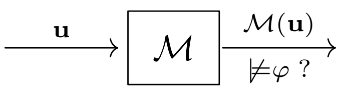


Figure 1.3: Falsification problem

This partially explains why falsification is a hard problem: indeed, no other clue is given by the model; the only means to access it is through sampling of input and output

[1]https://www.unrealengine.com/

signals. On the other hand, this setting also gives the technique a better practicality, in that its usage is not limited to any specific type of model.

In industry, Simulink, a toolbox developed by MathWorks, is a commonly used tool for modeling the CPS products. It provides a visualized interface for fast prototyping. It is also convenient to transform Simulink models to C programs. An example of Simulink model is shown in Fig. 1.1. The ports 1 and 2 are input signals for the system, namely *throttle* and *brake*; the ports 3, 4 and 5 are output signals, namely *RPM* (rounds per minute), *gear* and *speed*. It is composed of several blocks, and each of them can have several sub-blocks embedded. When an input signal is provided, the model can be executed and then output signals will be generated. In this work, all the experiments are done with Simulink models.

Input signal With the black-box setting, the problem boils down to searching for a specific input signal such that the corresponding output signal violates the system specification. Here, a signal is a time-variant function. It is impossible to search for values point by point in the time domain, as the domain is continuous. Therefore, we adopt parameterized representations of input signals as approximations of them. Commonly-used representations include piecewise constant (see Def. 2), piecewise linear (see Def. 3), pulse signals and so on. Fig. 1.4 shows an example of piecewise constant signals, in which the signal is composed of three constants. In practice, piecewise constant signals can be obtained by sampling the real signals at intervals; as a result, the number of samples, usually called *control points*, decides how closely the piecewise constant signal approximates the real one. For falsification, by fixing the type of representation and hyperparameters such as control points, the task is reduced to searching for a finite set of parameters that identify an input signal (counterexample).

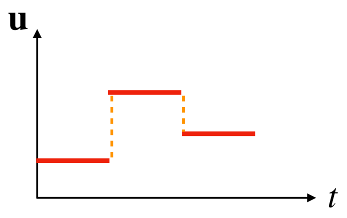


Figure 1.4: Piecewise constant signal

Specification System specifications (as introduced in §1.2) can be used to formalize the criteria, such as safety concerns, that the system needs to satisfy. Research has been heavily performed on formal languages for expressing specifications, such as [31–40]. Temporal logics are a family of formal languages for describing temporal properties. A well-known one for reasoning about discrete systems is Linear Temporal Logic (LTL) [41], which is used to express properties in a discrete domain. One of its counterparts for reasoning about hybrid systems is Signal Temporal Logic (STL) [32]. The feature that enables them to express temporal properties is the use of temporal operators, namely, *always* (or $\square$), *eventually* (or $\diamond$) and *until* (or $\mathcal{U}$). For example, STL is able to express the property in §1.2.1 as $\square(\mathit{gear} = 4 \rightarrow \mathit{speed} > 35)$ (see the syntax of STL in §2.2). Here, $\square$ is the *always* temporal operator, requiring the proposition $\mathit{gear} = 4 \rightarrow \mathit{speed} > 35$ to be true at every moment. Moreover, STL introduces quantitative semantics, of which the optimization algorithm can make use as a guidance to the search. In the next section, we will elaborate on how the quantitative STL robustness works.

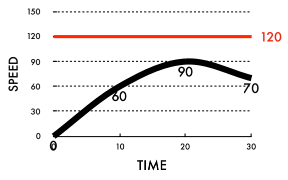
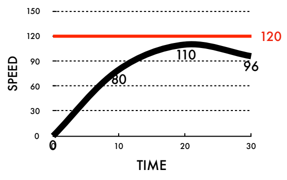
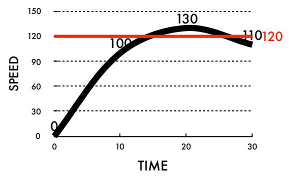
1.3.1 Quantitative STL Robustness

The Boolean satisfaction of temporal logics is well-known and easy to understand. In the context hybrid systems, it is a relation between a signal $\mathbf{u}$ and an STL formula φ—we write $\mathbf{u} \models \varphi$ if $\mathbf{u}$ satisfies φ, and $\mathbf{u} \nvDash \varphi$ otherwise.

Quantitative semantics of STL introduces quantities into the relation. This quantity is usually called *robustness*, and written as $[\![\mathbf{w}, \varphi]\!]$. We will introduce its formal definition in §2.2. The quantitative semantics does not only say that $\mathbf{u}$ satisfies φ or not, but also indicates *how robustly* $\mathbf{u}$ satisfies φ. See the examples in Table 1.1. There are three signals of *speed*, and we reason about their satisfaction to the temporal property $\square_{[0,30]}(\mathit{speed} < 120)$, that is, "during the time bound $[0, 30]$, the *speed* is always below 120". It is clear that the first 2 signals satisfy the property, while the third one violates it, which corresponds to the Boolean satisfaction shown in the second row. Furthermore, although both of the first two signals satisfy the property, intuitively the second one is more vulnerable to perturbations than the first one, in the sense that once a small change happens to the signal, it may violate the property. This intuition is embodied by the robustness in the third row. These values are computed from the margin between the peak point of each signal and the threshold 120. We can further see that when this value is negative (as the case of the third signal), the property is violated.

This is just a simple example of STL robustness; in practice, we need to deal with system specifications that are much more complicated. STL provides a well-established

Table 1.1: Boolean satisfaction $\mathbf{w} \models \varphi$, and quantitative robustness values, of three signals of speed for the STL formula $\varphi \equiv \square_{[0,30]}(speed < 120)$

signal $\mathbf{w}$			
$\mathbf{w} \models \varphi$	True	True	False
$[\![\mathbf{w}, \varphi]\!]$	30	10	−10

definition of robust semantics (see §2.2), which can always return a real number, given a signal and a property.

1.3.2 Stochastic Optimization-Based Falsification

One application of the STL robust semantics is the stochastic optimization-based falsification technique. Stochastic optimization is a family of optimization methods that implement diverse strategies of metaheuristics for the purpose of effective search. These algorithms introduce randomness into the search process, so that the optimization is stochastic—multiple executions of one algorithm with different random seeds lead to different performances. This feature gives the algorithms more possibilities to reach the optimization goal.

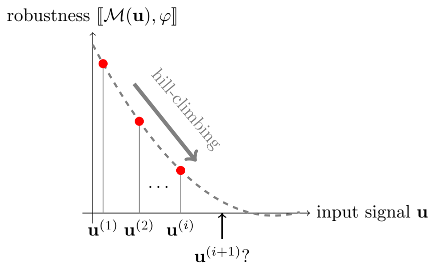

robustness $[\![\mathcal{M}(\mathbf{u}), \varphi]\!]$
hill-climbing
...
input signal $\mathbf{u}$
$\mathbf{u}^{(1)}$ $\mathbf{u}^{(2)}$ $\mathbf{u}^{(i)}$
$\mathbf{u}^{(i+1)}$?

Figure 1.5: Hill-climbing optimization

Metaheuristics refer to strategies that generate and utilize heuristic rules for quickly achieving optimization goals. Many of these algorithms are inspired from natural processes, such as *hill climbing*, *genetic algorithms*, *simulated annealing*, etc. (see the detailed introduction to these algorithms in §2.3). For example, hill climbing works with a process as shown in Fig 1.5: it starts with initial random samplings and obtains their objective function values; then it analyzes these values and conjectures the direction where the objective function potentially descends; based on that, it comes up with the next sample following that direction. This loop is repeated until a time budget is run up; hopefully by that time the optimization goal can be achieved. Just as the name indicates, this process is pretty like climbing (upwards or downwards) a hill. Other metaheuristics take different strategies, but all of them aim at the same goal. In this work, we refer to this body of techniques as hill climbing.

Falsification is turned into an optimization problem in the following way: robustness is treated as the objective function, and optimization aims to minimize it as much as possible. Once a negative robustness value is observed, then it indicates that the system specification is violated.

In order to solve that optimization problem, we employ stochastic optimization algorithms. The key reason why we select them is that they are applicable to black-box models: providing output data of the model is sufficient for them to work. Besides, efficiency and scalability issues are also important. Usually the system simulation that, given an input signal, generates the corresponding output signal is computationally expensive, so in practice it is preferred if an algorithm can work out efficiently with a small number of simulations. We also expect the algorithms to be scalable enough, so that the same workflow can be smoothly applied to systems of larger scale (e.g., with more input signals, system components, etc.). Although hill-climbing optimization algorithms also suffer from this scalability issue, the effect is less severe compared to other exhaustive techniques such as verification.

As the optimization methods are stochastic, different executions can result in different outcomes, depending on the random seeds. Therefore, in the experiments when we assess the algorithms, usually we repeat one experiment many times and then compute the percentage of the number of successful runs (that manages to return a counterexample signal) over the total runs.

So far we have gone through the workflow of the optimization-based falsification technique, as illustrated in Fig 1.6. The model accepts an input signal given by the optimizer, and runs simulation to generate an output signal; the robustness that reflects the relation between the output signal and the system specification is computed according to the STL robust semantics; the optimizer then tries to minimize the robustness by proposing a new input signal. This loop keeps working until a falsifying input is found, or a given time budget is run up.

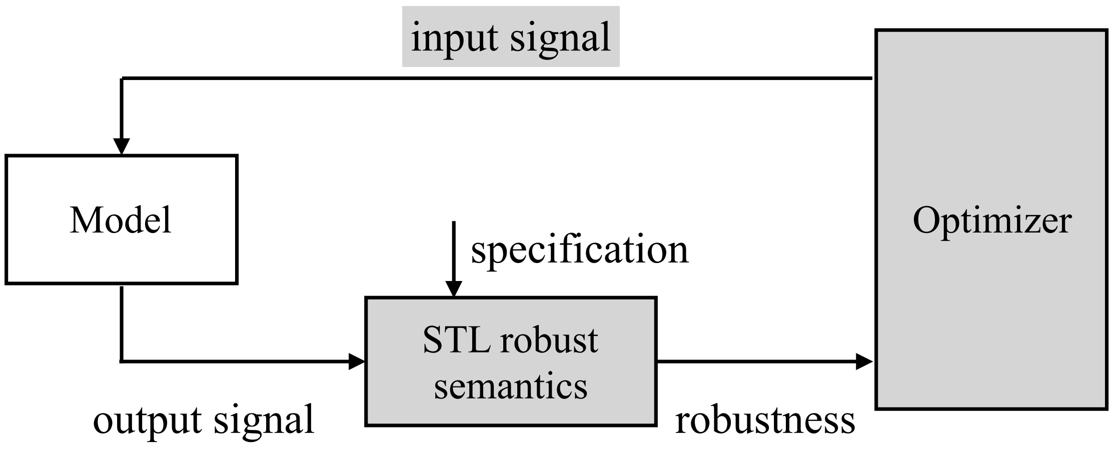


Figure 1.6: Workflow of stochastic optimization-based falsification

1.3.3 Usage Scenario of Falsification

In this section we come back to the usage scenario introduced in §1.2.1. We show how falsification can help the engineer with that problem.

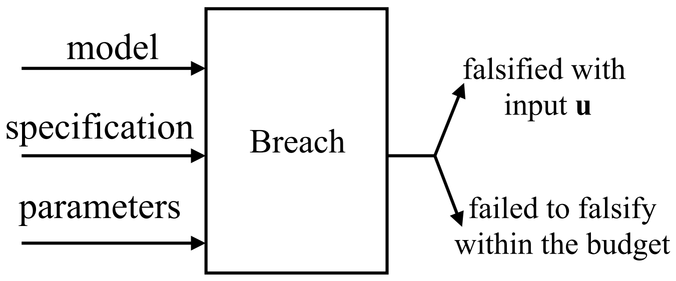
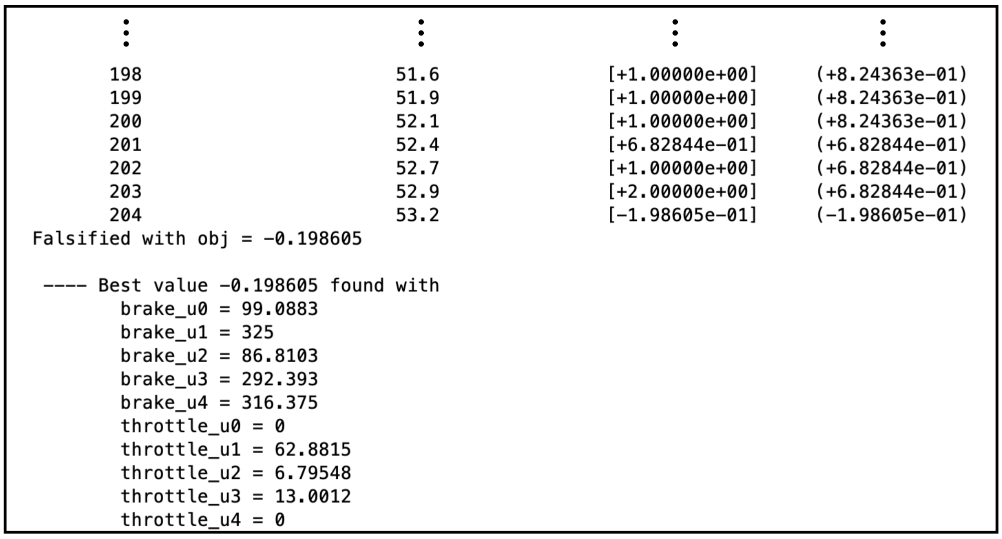

model
specification
parameters
Breach
falsified with input **u**
failed to falsify within the budget

Figure 1.7: Illustration of the tool Breach

Let us say the engineer decided to use a falsification tool Breach [22], one of the most commonly used among the existing falsification tools. As shown in Fig. 1.7, Breach takes a model (usually expressed in Simulink), a specification (in STL), and some other parameters, such as time budget, signal type, choice of an optimization algorithm, etc., as inputs; it then performs the falsification workflow shown in Fig. 1.6, and outputs a result—either succeeding in falsifying the specification with an input signal **u** or failing to do so.

In an experiment, the engineer gives the automatic transmission model in Fig. 1.1 to Breach as the system model under test, and then expresses the property in STL, namely



```
        ⋮                       ⋮                  ⋮                ⋮
       198                    51.6           [+1.00000e+00]     (+8.24363e-01)
       199                    51.9           [+1.00000e+00]     (+8.24363e-01)
       200                    52.1           [+1.00000e+00]     (+8.24363e-01)
       201                    52.4           [+6.82844e-01]     (+6.82844e-01)
       202                    52.7           [+1.00000e+00]     (+6.82844e-01)
       203                    52.9           [+2.00000e+00]     (+6.82844e-01)
       204                    53.2           [-1.98605e-01]     (-1.98605e-01)
 Falsified with obj = -0.198605

  ---- Best value -0.198605 found with
        brake_u0 = 99.0883
        brake_u1 = 325
        brake_u2 = 86.8103
        brake_u3 = 292.393
        brake_u4 = 316.375
        throttle_u0 = 0
        throttle_u1 = 62.8815
        throttle_u2 = 6.79548
        throttle_u3 = 13.0012
        throttle_u4 = 0
```

Figure 1.8: Screenshot of a trial by Breach

$\square_{[0,30]}(gear = 4 \rightarrow speed > 35)$ (see §2.2 for STL syntax); the engineer sets the time budget as 100 seconds, and uses CMA-ES (see §2.3) as the optimizer. The engineer then can command the program to start running, and wait for a while to see the results.

Breach shows a running instance as Fig. 1.8. On the top of the figure, there are four columns, respectively recording the number of simulations, total time consumed so far, the robustness value resulting from the current input signal, and the best (i.e., smallest) robustness over the input signal history. It is clear that the best robustness is decreasing, which means that Breach is approaching a falsifying input gradually. After 204 simulations (53.2 seconds), Breach succeeded in finding a falsifying input, shown on the bottom of the figure, with the robustness -0.198605. This result demonstrates that the car designed in Fig. 1.1 is not so safe as expected, and the engineer needs to redesign related components to fix some existing problems.

1.3.4 Current Status of Falsification in Academia and Industry

Over the years, the methodology of falsification has proved to be effective in practice. There have been several tools developed following the workflow depicted in Fig. 1.6, such as Breach [22] (see §1.3.3), S-TaLiRo [19], and our tool FalStar [27, 42], etc. These tools take Simulink models as systems under test. They differ in many implementation details. For example, S-TaLiRo implements MITL (Metric Interval Temporal Logic) [31] as the specification language, while Breach and FalStar adopt STL. MITL is originally designed for reasoning about discrete systems, but it has a well-defined robust semantics; STL is a counterpart of MITL in a continuous setting. These tools are also used for research purpose—their authors include new research outcomes in them, so they are evolving all the time. In academia, there is an annual friendly competition, held with the workshop of Applied Verification for Continuous and Hybrid Systems (ARCH)[1], for these tools to compete on their performances. See [43, 44] for competition results in recent years.

Falsification techniques are increasingly applied to testing complicated systems, such as autonomous driving systems, aviation systems, etc. These systems are typically large-scale and complex. More system parameters are needed to consider, and simulations are more time-consuming. Regarding these features, falsification provides an efficient way to discover system defects; meanwhile, successful applications to these systems have also proved the strength of falsification and enriched its theory. In many cases, falsification is introduced as a technique for test generation for autonomous vehicles, such as [45–47]. Due to the large system magnitude, reinforcement learning-based falsification techniques are heavily applied in testing those systems [48–51]. Moreover, testing systems with machine learning components is attracting more and more research attention. On the one hand, these components are being increasingly applied in modern CPS, bringing many uncertainties into the system behaviors; on the other hand, these components are known as uninterpretable and difficult to reason about. Many works [52–55] have studied the related topics.

Furthermore, the methodology has been adopted by many industrial manufacturers, such as Toyota [56], Bosch [57], Airbus [58], etc. Toyota is one of the earliest manufacturers that apply this methodology in verifying their products. In [56], they present a fuel control system, and list several requirements that the system is supposed to satisfy; they test those requirements with S-TaLiRo, and experiments show that in most cases S-TaLiRo does not find any violation, so they conclude that "the quality of the manual abstractions that we performed vis-à-vis high-level requirements is reasonable". Bosch has also been working on this for several years. They have been trying to apply the technique in testing autonomous driving systems. In [57] they present their practical experience, in particular they give several interesting tips,

[1]https://cps-vo.org/group/ARCH/FriendlyCompetition

such as "simulation is costly", "many properties can be formalized by formulas of simple form", etc. which are helpful for practitioners. Airbus [58] recently collaborates with academia on a project of testing their products. They apply Monte Carlo Tree Search (MCTS), a technique introduced to falsification by us [42] (to be presented in Chapter 3), in their work.

1.4 Motivation: Existing Problems and Related Works

As we introduced, the methodology introduced in §1.3 has proved to be effective, and it has been adopted by manufacturers such as Toyota [56], Bosch [57], etc. However, it does not mean that the methodology is already perfect. Many shortcomings emerge from both industrial practice and academic research. In this section, we list and elaborate on these problems as the motivation for this work; we also review the existing literature to show the recent progress on those topics.

The problems arise from the following aspects, corresponding to the three shaded boxes in Fig. 1.6.

1.4.1 Exploration and Exploitation

The trade-off between exploration and exploitation is the core of search-based techniques. Exploration prefers to cover the search space as broadly as possible, rather than focusing on a specific local area; exploitation is the other way around that looks into a local area as deeply as possible, without considering big jumps in the search space. It is a challenge how to balance exploration and exploitation, especially given a limited time budget for the search.

In falsification, we usually employ hill-climbing optimization algorithms as the optimizer (see §1.3.2). Nevertheless, these algorithms are usually known for their bias on exploitation, i.e., they tend to perform more exploitation to a specific local area rather than explore the entire search space. This feature leads these algorithms to the "local optimum" trap, as shown in Fig. 1.9. Suppose that an algorithm starts with the

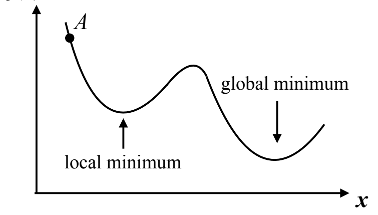


Figure 1.9: "Local optimum"

sample at point A, and that it performs hill climbing to proceed; after several loops, it is not hard to reach the local optimum point. However, the algorithm is usually unable to jump out of the local optimum after that, and as a consequence, it returns the local optimum as the optimization result, missing the global one elsewhere.

This problem affects the falsification performance severely, and thus it becomes one of the main directions of the research in falsification. The works that handle this problem can be classified according to the techniques used as a back end, as follows.

Metaheuristics This line of works makes use of metaheuristics to avoid pure exploitation. Many metaheuristic-based approaches implement such a mechanism that triggers a jump when the search falls into a local optimum. For example, *Simulated Annealing* [59] is an algorithm inspired from the process of annealing in metallurgy. Compared to the naive hill climbing, it has a possibility to jump out of the local search. Its application to falsification is studied in [60, 61], and it now has become a basic optimizer for falsification in both Breach and S-TaLiRo. We will give a more detailed introduction in §2.3. In [62], the authors apply the famous *ant colony optimization* to falsification, in which the search takes a small probability to start a new exploration. In [20], the authors propose the use of *tabu search* for falsification. The algorithm maintains a *tabu list* during the search, which is a memory structure that records the sampling history, so that it will not fall into the same local optimum twice. *Cross entropy* [63] is an *importance sampling* method that initially explores the search space and then performs biased sampling in the promising area. Its application to falsification is studied in [64]. S-TaLiRo implements the stochastic optimization with the *adaptive restart* mechanism [65] to avoid trapping into local optima. Some hill-climbing algorithms are able to change the portion of exploration/exploitation through modifying some parameters, such as CMA-ES as studied in [66].

Coverage-guided metrics Coverage is an important notion in software testing. Originally, it is used to measure how much portion of a program is executed by test suites. In the case when no error is found from the program, the larger coverage the test suites achieve, the more reliable the testing result can be considered. Some falsification techniques are developed based on the coverage notion for the aim of avoiding local optimum. Those techniques employ coverage as a guidance for the search—with that guidance, the search then aims not only to minimize the robustness, but also to enlarge the coverage to the search space. In this way, the search can avoid pure exploitation because that does not help to obtain a good coverage.

As hybrid systems deal with infinite search space, it remains the problem how to define the coverage metric in this context. Different coverage definitions have been proposed. In [67], the coverage is defined based on the internal states of Simulink models, and it is included as a part of the objective function so that the search is guided by both robustness and coverage. However, it is unclear how the defined robustness affects the balance between exploration and exploitation. In [23], the coverage is defined by *star discrepancy*, a measure for evaluating how well-distributed a set of samples are. This coverage definition thus guarantees exploration; however, as it considers input space only, it emphasizes purely on exploration. In [68], a coverage notion is defined based on classification in input space according to robustness values. This work gives a better balance between exploration and exploitation because it takes not only input space but also robustness into consideration.

Machine learning Machine learning techniques are developing rapidly, and they are also applied to falsification for enhancing the search effectiveness. In general, these techniques learn models or objective functions from sampling data; based on that, they come up with new samples that explore the search space effectively. Specifically two machine learning techniques have been used heavily in falsification, that is, Bayesian optimization and reinforcement learning.

Bayesian optimization makes use of Gaussian Process Regression to learn the objective function (i.e., robustness in our context); this results in a Gaussian process over the search space. Gaussian process is a model in which every finite collection of variables form a multivariate normal distribution; in our context, it predicts the probability distribution of robustness at each point. Bayesian optimization then samples at some potentially interesting places, that is, those predicted to have low robustness. The sampling strategy, known as *acquisition function*, guarantees the balance between exploration and exploitation. This technique has been studied in [69–72]. In [69], the authors apply a dimension reduction method to mitigate the scalability issue. In [71], the authors exploit the causality between the sub-formulas and the global formula of the specification to accelerate the search. There are also other machine learning techniques that are based on Gaussian Process Regression, such as *active learning* that is applied to falsification in [25].

Reinforcement learning is a hot spot in the machine learning community in recent years. It tries to learn the best policy—a sequence of actions—that an agent can take, from the interactions between the agent and the environment. There is also a trade-off between exploration and exploitation in reinforcement learning: when the agent decides which action to take for the next step, the choice is either to explore an action that has not been taken before, or to exploit an already taken action further to get more reliable feedback. In recent years, this technique has been applied to falsification [26,73]. In [26], the authors study the application of reinforcement learning to falsification, where they discretize the input signal as a sequence of actions and search in an incremental way. In [73], the authors also study the technique, with the aim of falsifying a family of models rather than a single one. Reinforcement learning is also applied in the context of testing autonomous vehicles [74] for detecting dangerous scenarios, going through a similar workflow with falsification.

1.4.2 Robust Semantics Definition

The definition of robust semantics in our context plays the role of objective function for optimization, and thus it affects the performance of the entire framework significantly. The current widely-used STL robust semantics definition captures only spatial robustness (see §2.2 for the definition and see the example in Table. 1.1). The formalism [32] that captures the temporal robustness also exists. It formalizes such an intuition: for example, compare the firstly two signals in Table 1.1; the first one arrives at the *speed* 90 much later than the second one, therefore, the former is more robust than the latter. In [33], the authors propose a way that takes both perspectives into account, resulting in the *Averaged STL (AvSTL)*. While the computation of spatial or temporal robustness is just a matter of obtaining a vertical or horizontal distance, AvSTL requires to compute the integration of the signal over certain interval; the authors then design a specific algorithm for that purpose. They also experimentally show the strength of that semantics applied in the falsification problems.

In this work, we tackle another problem, that is, the robust semantics for Boolean connectives. In the existing definition, when it comes to the robustness for Boolean connectives, it makes a comparison between the robustness values of different sub-formulas—for conjunctive it takes the minimum, and for disjunctive it takes the maximum (see §2.2 for details). However, this yields new problems. Intuitively, if the sub-formulas concern with different signals ranging over different scales, then the comparison becomes unfair, because one can always beat the other one. This is the so-called *scale problem*, and we elaborate on that with a concrete example in §4.1. There has been one work handling that problem [75]. In that work, they explicitly declare the input and output signals so that they manually introduce a bias on the comparison. This method solves the problem in many cases, however, it requires domain expertise and human intervention.

1.4.3 Input Constraints

As we introduced in §1.3.2 the way hill-climbing optimization works, it relies on random sampling to explore the search space. Therefore, the search space must be unconstrained (usually a hyperrectangle). In this way, falsification can return any input signal that violates the system specification in the search space, once it manages to find one. However, in reality, there usually exist some logical constraints among input signals. One example is that, in an automotive system like the one in Fig. 1.1, the *throttle* and *brake* cannot be pushed simultaneously. Such constraints also exist in other literature about CPS products; for example in [76], the authors test an assisted driving system under different environment and system parameters; there is a constraint—"when there is no fog, the visibility range is set to maximum"—that restricts the system inputs. Sometimes, the engineers also desire to impose some conditions on the input when testing systems; for example in [56], the authors aim to test a system under the condition that *throttle* increases monotonically. In the presence of such constraints, input signals produced by falsification should be guaranteed to satisfy them; otherwise, those signals would be meaningless.

Few works have addressed this issue before. To the best of our knowledge, [77] is the only one. It utilizes a timed automaton, that implements the input constraints, to generate meaningful words, and then it applies Monte Carlo sampling methods to produce input signals. This work does solve the input constraint problem; however, it cannot be integrated with hill-climbing optimization, that is more efficient in finding counterexamples than Monte Carlo sampling.

In the optimization community, the constrained optimization problem has been heavily studied. Many among those works are white-box methods, but they are not applicable in our context. Methods based on black-box models can be classified into the following categories according to [78]:

- Methods based on penalty functions. The intuition is simple: once the input does not satisfy the constraint, add a penalty value to the objective function. In this way, the constrained optimization problem can be converted into an unconstrained one. Common methods include death penalty [79], static penalty [80], dynamic penalty [81], etc. As these methods change the objective function quite much, the performance of them is very problem-dependent. In §5.2, we will present our study [82] of penalty-based approaches to falsification.
- Methods based on search of feasible solutions. These methods make use of heuristic rules that advise the search to proceed towards the feasible areas. Examples include [83, 84], etc. This is also a commonly-used approach, but as it depends on heuristics, there is no guarantee on the performance.
- Methods based on preserving feasibility of solutions. These methods somehow transform the infeasible samples to feasible ones, with the help of encoding and decoding techniques. This proposal has already studied in [85]. Our contribution in Chapter 5, §5.3 is also based on this idea: we define a new transformation method, and extend it with introduction of multi-armed bandit model.
- Hybrid methods. They are combinations of the methods in the aforementioned categories. Examples include [86, 87], etc.

1.5 A Hierarchical Optimization Framework

In this work, we propose a hierarchical optimization framework, used as a general idea for solving the problems that we discussed in §1.4. We then instantiate this framework to three techniques, each addressing one specific problem of §1.4. In this section, we introduce the philosophy of the hierarchical framework, and briefly preview the contributions in the following chapters.

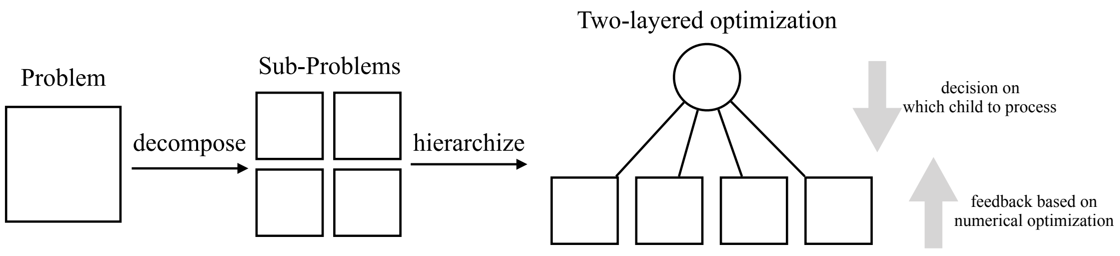

Problem
decompose
Sub-Problems
hierarchize
Two-layered optimization
decision on which child to process
feedback based on numerical optimization

Figure 1.10: Solving problems using a hierarchical optimization framework

Hierarchical framework The hierarchical optimization framework is shown as in Fig. 1.10. It firstly decomposes the problem into a set of sub-problems, and then uses a hierarchical methodology: the top layer makes a decision on selecting one child problem as the next step to proceed, based on the information coming from the bottom layer; the bottom layer performs numerical optimization on the sub-problem suggested by the top layer, and it returns feedback to the top layer. This process takes place iteratively, until the problem is solved. One feature of this framework is the interaction between the two layers: the top layer makes high-level decisions and advises the bottom layer; the bottom layer performs concrete numerical evaluation and gives feedback to the top layer to facilitate the future decisions.

This framework is instantiated into three techniques addressing the problems in §1.4. Concretely, they are:

- A Monte Carlo Tree Search (MCTS)-based technique for balancing exploration and exploitation during the search. MCTS is originally an artificial intelligence technique, and is booming in recent years due to its application to computer Go games. In Chapter 3, we apply MCTS to explore the search space that is organized as a tree by *time staging* and *space discretization.* A key notion of MCTS is *reward*, as the asymmetric search performed by MCTS is guided by rewards attached with branches. We connect the definition of reward with robustness—the branches exposing high robustness have low reward, and vice versa. The robustness values are obtained by running hill-climbing optimization on the sub-spaces identified by branches. In this way, we combine MCTS and hill-climbing optimization, which gives rise to a two-layered optimization framework.

- A Multi-Armed Bandit (MAB)-based technique solving the problem in the definition of STL robust semantics. The MAB problem originally models the problem of how to maximize the rewards that a gambler can earn in front of a row of bandit machines. In Chapter 4, we take different sub-formulas of a Boolean connective as "bandit machines", and connect the rewards with robustness. We then apply MAB algorithms, such as UCB1, to govern the hill-climbing optimization running with each "machine": once the robustness of one sub-formula descends smoothly, then MAB takes more efforts on that one; otherwise, less budget will be assigned. In this work, MAB algorithms work on the high level, governing the hill-climbing optimization running on the low level.

- A search space transformation-based technique extended with MAB for solving input constrained falsification. Search space transformation is a technique that allows optimization to sample in an unconstrained space. When a sample comes, search space transformation maps it to a point in the constrained space, and computes its fitness according to the robustness of the mapped point. Once a sample with negative fitness is found, the mapped point in the constrained space will be returned as a falsifying input; as that point is from the constrained space, it is guaranteed to satisfy the constraint. The performance of this framework is subject to a hyperparameter on the choice of a transformation map, more specifically, a total order over dimensions of the search space. In order to achieve the best performance, we apply MAB algorithms in this context again to select the optimal order.

1.6 Organization

The rest of the work is organized as follows:

- Chapter 2 describes a formal model for falsification.
- Chapter 3 talks about the exploration and exploitation problem, and particularly presents the application of Monte Carlo Tree Search in falsification. This chapter is based on two works: [88] and [42].
- Chapter 4 addresses the problem in STL robust semantics definition for Boolean connectives, and proposes a novel approach based on the Multi-Armed Bandit model to solve it. This chapter is based on the work [89].

- Chapter 5 discusses on the input constraint problem, and introduces a technique named search space transformation and three parameter selection approaches based on that technique. This chapter is based on two works: [82] and [90].
- Chapter 6 concludes this work.

2 Preliminaries: Optimization-Based Falsification

In this chapter, we review the widely-accepted methodology of stochastic optimization-based falsification. Mainly, we give a formal description of the framework.

2.1 System Models

We treat the system model as a black box—the system behaviors are only observed from inputs and their corresponding outputs. We firstly define signals, and later introduce system model.

Definition 1 ((Time-bounded) signal) *Let $T \in \mathbb{R}_+$ be a positive real. An M-*dimensional signal *with a time horizon T is a function $\mathbf{w}\colon [0, T] \to \mathbb{R}^M$.*

Let $\mathbf{w}\colon [0, T] \to \mathbb{R}^M$ and $\mathbf{w}'\colon [0, T'] \to \mathbb{R}^M$ be M-dimensional signals. Their concatenation $\mathbf{w} \cdot \mathbf{w}'\colon [0, T + T'] \to \mathbb{R}^M$ *is the M-dimensional signal defined by $(\mathbf{w} \cdot \mathbf{w}')(t) = \mathbf{w}(t)$*

if $t \in [0, T]$, *and* $(\mathbf{w} \cdot \mathbf{w}')(t) = \mathbf{w}'(t - T)$ *if* $t \in (T, T + T']$.

Let $0 < T_1 < T_2 \leq T$. *The* restriction $\mathbf{w}|_{[T_1,T_2]} \colon [0, T_2 - T_1] \to \mathbb{R}^M$ *of* $\mathbf{w} \colon [0, T] \to \mathbb{R}^M$ *to the interval* $[T_1, T_2]$ *is defined by* $(\mathbf{w}|_{[T_1,T_2]})(t) = \mathbf{w}(T_1 + t)$.

The time domain of signals in Def. 1 is continuous. This derives the difference between discrete systems and hybrid systems: in discrete systems all the state transitions are “jumps”, while in hybrid systems physical components exhibit such continuous “flows”.

In order for computer systems to handle signals, such continuity is a barrier. In practice, engineers consider parameterized representations of continuous signals for approximation. Examples of such approximations include *piecewise constant*, *piecewise linear*, etc. See the following definitions.

Definition 2 (Piecewise constant signal) *Let* K *be a positive integer. A signal* $\mathbf{w} \colon [0, T] \to \mathbb{R}^M$ *is* piecewise constant *if for all* $k \in \{0, \ldots, \mathsf{K} - 1\}$, $\mathbf{w}(t)$ *is a vector of* M *constant values in the interval* $t \in [k\frac{T}{\mathsf{K}}, (k+1)\frac{T}{\mathsf{K}}]$.

Definition 3 (Piecewise linear signal) *Let* K *be a positive integer. A signal* $\mathbf{w} \colon [0, T] \to \mathbb{R}^M$ *is* piecewise linear *if for all* $k \in \{0, \ldots, \mathsf{K} - 1\}$, *each dimension of* $\mathbf{w}(t)$ *is linear in the interval* $t \in [k\frac{T}{\mathsf{K}}, (k+1)\frac{T}{\mathsf{K}}]$.

The parameter K is known as the number of *control points*. Apparently, once K is fixed as a positive natural number, a piecewise constant/linear signal can be identified by $\mathsf{K} \cdot M$ parameters of real number.

Definition 4 (Signal range) *Let* $\mathbf{w} \colon [0, T] \to \mathbb{R}^M$ *be an* M*-dimensional signal. The range of* $\mathbf{w}$ *is an* M*-dimensional hyperrectangle* Ω *such that* $\mathbf{w}(t) \in \Omega$ *for all* $t \in [0, T]$.

We give a formal description of the system model, which we treat as a black box. We simply define the system model as a function.

Definition 5 (System model $\mathcal{M}$) *A* system model, *with* M*-dimensional input and* N*-dimensional output, is a function* $\mathcal{M}$ *that takes an input signal* $\mathbf{u} \colon [0, T] \to \mathbb{R}^M$ *and returns a signal* $\mathcal{M}(\mathbf{u}) \colon [0, T] \to \mathbb{R}^N$. *Here the common time horizon* $T \in \mathbb{R}_+$ *is arbitrary, and the input signal* $\mathbf{u}$ *is bounded by a range* Ω.

Furthermore, we impose the following *causality* condition on $\mathcal{M}$: for any time-bounded signals $\mathbf{u}\colon [0,T] \to \mathbb{R}^M$ and $\mathbf{u}'\colon [0,T'] \to \mathbb{R}^M$, we require that $\mathcal{M}(\mathbf{u}\cdot\mathbf{u}')\big|_{[0,T]} = \mathcal{M}(\mathbf{u})$.

Note that $\mathcal{M}(\mathbf{u}\cdot\mathbf{u}') = \mathcal{M}(\mathbf{u})\cdot\mathcal{M}(\mathbf{u}')$ does not hold in general: feeding $\mathbf{u}$ can change the internal state of $\mathcal{M}$. This motivates the following definition.

Definition 6 (Continuation $\mathcal{M}_{\mathbf{u}}$) *Let $\mathcal{M}$ be a system model and $\mathbf{u} : [0,T] \to \mathbb{R}^M$ be a signal. The continuation of $\mathcal{M}$ after $\mathbf{u}$, is defined as follows. For an input signal $\mathbf{u}' : [0,T] \to \mathbb{R}^M$, $\mathcal{M}_{\mathbf{u}}(\mathbf{u}')(t) := \mathcal{M}(\mathbf{u}\cdot\mathbf{u}')(T+t)$.*

2.2 Robust Semantics for STL

In this work, we select Signal Temporal Logic (STL) as our specification language. This is a common choice in the falsification community, and it is also adopted by the tool Breach [22]. In this section, we review the syntax and robust semantics of STL.

Our definitions here are taken from [31, 32].

Definition 7 (STL syntax) *We fix a set* $\mathbf{Var}$ *of variables. In STL,* atomic propositions *and* formulas *are defined as follows, respectively: $\alpha ::\equiv f(x_1,\dots,x_N) > 0$, and $\varphi ::\equiv \alpha \mid \bot \mid \neg\varphi \mid \varphi\wedge\varphi \mid \varphi\vee\varphi \mid \varphi\,\mathcal{U}_I\,\varphi$. Here f is an N-ary function $f : \mathbb{R}^N \to \mathbb{R}$, $x_1,\dots,x_N \in \mathbf{Var}$, and I is a closed non-singular interval in $\mathbb{R}_{\geq 0}$, i.e. $I = [a,b]$ or $[a,\infty)$ where $a,b\in\mathbb{R}$ and $a<b$.*

We omit subscripts I for temporal operators if $I = [0,\infty)$. Other common connectives such as $\to, \top$, $\Box_I$ (always) and $\Diamond_I$ (eventually), are introduced as abbreviations: $\varphi_1 \to \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$, $\Diamond_I\varphi \equiv \top\,\mathcal{U}_I\,\varphi$ and $\Box_I\varphi \equiv \neg\Diamond_I\neg\varphi$. An atomic formula $f(\vec{x}) \leq c$, where $c\in\mathbb{R}$, is accommodated using $\neg$ and the function $f'(\vec{x}) := f(\vec{x}) - c$.

Below we will introduce the robust semantics of STL. More precisely, this is the definition of spatial robustness.

Definition 8 (Robust semantics [32]) *Let $\mathbf{w}\colon [0,T] \to \mathbb{R}^N$ be an N-dimensional signal, and $t \in [0,T)$. The t-*shift *of $\mathbf{w}$, denoted by $\mathbf{w}^t$, is the time-bounded signal $\mathbf{w}^t\colon [0,T-t] \to \mathbb{R}^N$ defined by $\mathbf{w}^t(t') := \mathbf{w}(t+t')$.*

Let $\mathbf{w}\colon [0,T] \to \mathbb{R}^{|\mathbf{Var}|}$ be a signal, and φ be an STL formula. We define the robustness *$[\![\mathbf{w},\varphi]\!] \in \mathbb{R}\cup\{\infty,-\infty\}$ as follows, by induction on the construction of formulas. Here $\bigsqcap$*

and $\bigsqcup$ denote infimums and supremums of real numbers, respectively. Their binary version $\sqcap$ and $\sqcup$ denote minimum and maximum.

$$
\begin{aligned}
&[\![\mathbf{w}, f(x_1, \cdots, x_n) > 0]\!] \;:=\; f\big(\mathbf{w}(0)(x_1), \cdots, \mathbf{w}(0)(x_n)\big)\\
&[\![\mathbf{w}, \bot]\!] \;:=\; -\infty \qquad [\![\mathbf{w}, \neg\varphi]\!] \;:=\; -[\![\mathbf{w}, \varphi]\!]\\
&[\![\mathbf{w}, \varphi_1 \wedge \varphi_2]\!] \;:=\; [\![\mathbf{w}, \varphi_1]\!] \sqcap [\![\mathbf{w}, \varphi_2]\!] \qquad [\![\mathbf{w}, \varphi_1 \vee \varphi_2]\!] \;:=\; [\![\mathbf{w}, \varphi_1]\!] \sqcup [\![\mathbf{w}, \varphi_2]\!]\\
&[\![\mathbf{w}, \varphi_1 \,\mathcal{U}_I\, \varphi_2]\!] \;:=\; \textstyle\bigsqcup_{t \in I \cap [0,T]} \big([\![\mathbf{w}^t, \varphi_2]\!] \sqcap \bigsqcap_{t' \in [0,t)} [\![\mathbf{w}^{t'}, \varphi_1]\!] \big)
\end{aligned}
$$

For atomic formulas, $[\![\mathbf{w}, f(\vec{x}) > c]\!]$ stands for the vertical margin $f(\vec{x}) - c$ for the signal $\mathbf{w}$ at time 0. A negative robustness value indicates how far the formula is from being true. It follows from the definition that the robustness for the eventually modality is given by $[\![\mathbf{w}, \Diamond_{[a,b]}(x > 0)]\!] = \bigsqcup_{t \in [a,b] \cap [0,T]} \mathbf{w}(t)(x)$.

The above robustness notion taken from [32] is therefore *spatial*. Other robustness notions [32] take *temporal* aspects into account, too, such as "how long before the deadline the required event occurs." A robustness definition that integrates both temporal and spatial features is introduced in [33]. Our choice of spatial robustness in this paper is for the sake of simplicity, and is thus not essential.

The original semantics of STL is Boolean, given as usual by a binary relation $\models$ between signals and formulas. The robust semantics refines the Boolean one in the following sense: $[\![\mathbf{w}, \varphi]\!] > 0$ implies $\mathbf{w} \models \varphi$, and $[\![\mathbf{w}, \varphi]\!] < 0$ implies $\mathbf{w} \not\models \varphi$, see [31, Prop. 16]. Optimization-based falsification via robust semantics hinges on this refinement.

2.3 Hill Climbing-Guided Falsification

We firstly make clear the basic problem setting in this work, and then introduce the optimization-based solution to it.

Definition 9 (Falsification problem) *Let $\mathcal{M}$ be a system model, and φ be an STL formula. Falsification aims to find an input signal* $\mathbf{u}$ *such that the corresponding output signal $\mathcal{M}(\mathbf{u})$ violates φ, i.e., $\mathcal{M}(\mathbf{u}) \not\models \varphi$. Here, input signal* $\mathbf{u}$ *is called a falsifying input.*

In general, it is infeasible to search for a continuous falsifying input $\mathbf{u}$. Therefore, we use parameterized representations of input signals, such as piecewise constant/linear (see Def. 2 and Def. 3), and thus the task is reduced to searching for a finite set of parameters that identify $\mathbf{u}$. In this work, we use piecewise constant signals for the sake of simplicity.

The solution to falsification problem is via transforming it into an optimization problem, shown as follows.

Definition 10 (Optimization problem derived from falsification) *The optimization problem derived from falsification is shown as follows:*

$$\begin{aligned} \underset{\mathbf{u}}{\textit{minimize}} \quad & [\![\mathcal{M}(\mathbf{u}), \varphi]\!] \\ \textit{subject to} \quad & \mathbf{u}(t) \in \Omega \end{aligned}$$

where the robustness $[\![\mathcal{M}(\mathbf{u}), \varphi]\!] \in \mathbb{R} \cup \{\infty, -\infty\}$ serves as the objective function (a.k.a. fitness function), and the input signal $\mathbf{u}$ is bounded by a signal range Ω. The goal of the optimization is to find an input signal $\mathbf{u}$ such that $[\![\mathcal{M}(\mathbf{u}), \varphi]\!] < 0$.

In order to solve the optimization problem in Def. 10, we apply *hill-climbing optimization.* It is a metaheuristic-based stochastic optimization algorithm. We present the workflow of hill-climbing optimization in Def. 11.

Definition 11 (Hill climbing-guided falsification) *Assume the setting in Def. 9. For finding a falsifying input, the methodology of* hill climbing-guided falsification *is presented in Algorithm 2.1.*

The algorithm requires a system model $\mathcal{M}$, an STL formula φ as the system specification, and a budget $\mathbf{K}$ that can be in the form of time limit or the number of simulations.

Here the function Hill-Climb *makes a guess of an input signal $\mathbf{u}_k$, aiming at minimizing the robustness $[\![\mathcal{M}(\mathbf{u}_k), \varphi]\!]$. It does so, learning from the previous observations $\big(\mathbf{u}_l, [\![\mathcal{M}(\mathbf{u}_l), \varphi]\!]\big)_{l\in[1,k-1]}$ of input signals $\mathbf{u}_1, \ldots, \mathbf{u}_{k-1}$ and their corresponding robustness values (cf. Table 1.1).*

The Hill-Climb function can be implemented by various metaheuristic strategies. These strategies are usually inspired by natural processes or phenomena; we will see some examples below.

Algorithm 2.1 Hill climbing-guided falsification

```
Require: a system model $\mathcal{M}$, an STL formula $\varphi$, and a budget $\mathbf{K}$
 1: function Hill-Climb-Falsify($\mathcal{M}, \varphi, \mathbf{K}$)
 2:     $\mathsf{rb} \leftarrow \infty$ ;   $k \leftarrow 0$        ▷ rb is the smallest robustness so far, initialized to ∞
 3:     while $\mathsf{rb} \geq 0$ and $k \leq \mathbf{K}$ do
 4:         $k \leftarrow k + 1$
 5:         $\mathbf{u}_k \leftarrow \textsc{Hill-Climb}\Big( \big(\mathbf{u}_l, [\![\mathcal{M}(\mathbf{u}_l), \varphi]\!]\big)_{l\in[1,k-1]} \Big)$
 6:         $\mathsf{rb}_k \leftarrow [\![\mathcal{M}(\mathbf{u}_k), \varphi]\!]$
 7:         if $\mathsf{rb}_k < \mathsf{rb}$ then
 8:             $\mathsf{rb} \leftarrow \mathsf{rb}_k$
 9:     $\mathbf{u} \leftarrow \begin{cases} \mathbf{u}_k & \text{if } \mathsf{rb} < 0 \text{, that is, } \mathsf{rb}_k = [\![\mathcal{M}(\mathbf{u}_k), \varphi]\!] < 0 \\ \text{Failure} & \text{otherwise, that is, no falsifying input found within budget } \mathbf{K} \end{cases}$
10:     Return u
```

- Global Nelder-Mead [91]. Nelder-Mead method makes use of the simplex to perform local search. It is specialized in searching for local optimum, but unable to jump out of it to search for the global optimum. Global Nelder-Mead introduces a probabilistic restart mechanism, so that the search is possible to be reinitialized elsewhere and thus not trapped in the local optimum.

- Simulated annealing. Simulated annealing is inspired by the physical process of heating a material and then cooling it down to decrease defects. The algorithm starts with random sampling, and accepts all new points that have a lower temperature (i.e., robustness in our context). The feature of the algorithm is that it also accepts points that have higher temperature with a small probability, therefore the algorithm is able to jump out of the local optimum.

- CMA-ES [92] CMA-ES is the abbreviation for Covariance Matrix Adaption Evolutionary Strategy, belonging the family of evolutionary algorithm (genetic algorithm). Generally, genetic algorithm is inspired by Darwin's theory of evolution: it maintains a set of solutions, known as *population*. Different individuals in the population consist of different combinations of properties; this combination is known as *chromosome*. The genetic algorithm proceeds much like the evolution process of creatures: mutations and crossovers can happen to chromosomes, so new individuals are created; population rule out the individuals that have low fitness and keep those good ones, like the *natural selection* theory. In this way, the algorithm proceeds towards the optimization goal gradually. CMA-ES, based on the general evolutionary algorithm, employs a *multivariate normal distribution* to approximate the objective function, and updates its covariance matrix according to the information learned by performing mutations and crossovers.

Compared to naive random sampling, hill climbing learns from sampling history and based on that comes up with the next samplings; therefore, it is a more intelligent and effective approach. Moreover, it suits well for black box—learning from input and output data is sufficient.

2.4 Evaluation Metrics of Falsification Algorithms

In this section, we introduce the metrics for assessing falsification algorithms. As falsification employs stochastic optimization as the optimizer (see §1.3.2), different falsification trials with different random seeds can produce different results. The metrics we use in this work thus take this feature into account, so they evaluate falsification algorithms by running them repeatedly and analyzing the results statistically.

We consider two metrics, namely effectiveness (if an algorithm can find an answer) and efficiency (how fast an algorithm manages to find an answer). In general, we prioritize effectiveness rather than efficiency of an algorithm, because the ultimate goal of falsification is to find out the falsifying inputs, as long as the time budget is not run out. These two aspects are embodied by the following two measurements.

Falsification success rate It refers to the percentage of the trials that succeed in finding falsifying inputs within the budget, over the total number of trials. A high success rate certificates the effectiveness of an algorithm, in the sense that it manages to find falsifying inputs with a high probability. Otherwise, it indicates that the algorithm is not effective, or vulnerable to random seeds, thus not stable.

Time consumption For successful trials, time consumption is counted until the time when the algorithm returns a falsifying input; for unsuccessful trials, usually they will run out of the budget, or it can also be the case that the optimization algorithm meets its internal termination criteria, e.g., local optimum, and thus quits halfway. Therefore, there are two ways of computing time consumption: one is by computing the average time consumption of only successful trials; the other one is by taking also unsuccessful trials into consideration, and thus computing the average time consumption of both successful and unsuccessful trials, in the latter case taking time budget as their values.

The advantage of the first method is that it reflects the efficiency of algorithms better, e.g., in the case that a “greedy” algorithm is fast but not effective; the second method integrates effectiveness into efficiency, exemplifying the principle that only if one is effective can it be efficient. Both methods are adopted by literature, especially the former one. In this work, we also mainly use the former one.

Note that these two metrics are not independent—as the success rate metric is subject to the parameter of budget, in many cases efficiency leads to effectiveness. Therefore, we report both falsification success rate and time consumption in our experiments.

3 Balancing Exploration and Exploitation Using Monte Carlo Tree Search

Balance between exploration and exploitation is the core issue in search-based techniques. Pure exploration is beneficial to covering the entire search space, but it is unlikely to hit the goal state if that is rare; pure exploitation is able to reach the local optimum, but it may miss the global one if that is elsewhere. The collaboration between exploration and exploitation is thus of great importance, especially in the case where budget is limited, as in falsification.

In this chapter, we tackle the problem of balancing exploration and exploitation during the search by using a Monte Carlo Tree Search (MCTS) based technique. This technique discretizes the search space, and then combines MCTS with hill-climbing optimization, yielding a two-layered optimization framework that improves the search effectiveness. We introduce the framework in §3.2. Before that, we firstly introduce an idea of time staging in §3.1 that preludes the main contribution of this chapter.

The material in this chapter is based on [88] and [42]

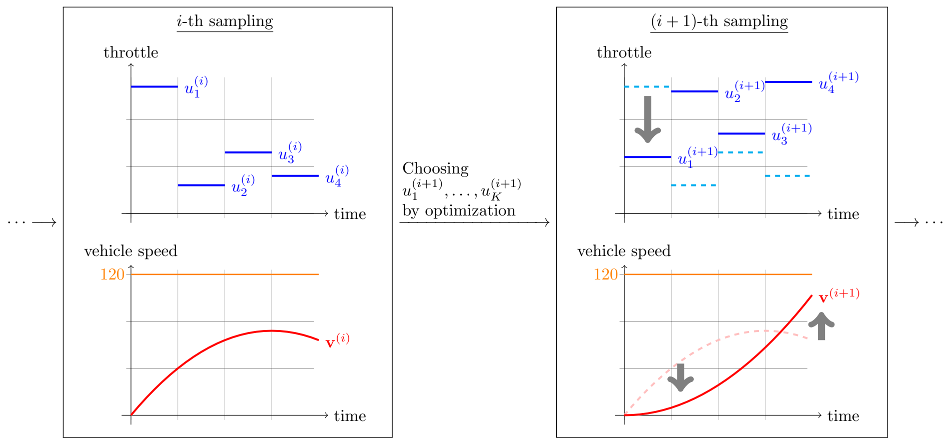


Figure 3.1: Falsification by global optimization

3.1 Exploiting Time Causality via Time Staging

The falsification technique introduced in §1.3 has a weakness, that is, it does not make use of *time causal* information in the problem. Time causality has been introduced in §2.1; it says that the continuation of a model $\mathcal{M}$ after a signal $\mathbf{u}$ does not equal to the initial $\mathcal{M}$, since feeding $\mathbf{u}$ to $\mathcal{M}$ changes the state of $\mathcal{M}$. This derives a property of *time monotonicity*: an input prefix that achieves smaller robustness is more likely to extend to a full falsifying input signal. However, this property is ignored by the existing falsification framework.

See Fig. 3.1 for an illustration. This trial aims to falsify a simple property $\square(\mathbf{v} < 120)$, i.e., it aims to find a piecewise constant signal *throttle* such that the *vehicle speed* $\mathbf{v}$ is over 120 at some moments. Compare the i-th and the $(i+1)$-th samplings. We can observe that $\mathbf{v}^{(i+1)}$ is better than $\mathbf{v}^{(i)}$ in that it has a (globally) lower robustness. This is thanks to the application of hill-climbing optimization that helps the sampling approach the falsification goal. However, if we take a closer look at these two samples, we can find that the prefix of $\mathbf{v}^{(i+1)}$ in the first interval is not as good as that of $\mathbf{v}^{(i)}$, due to that the value of $u_1^{(i+1)}$ is lower than $u_1^{(i)}$. It is reasonable to hypothesize that keeping the value of $u_1^{(i)}$ would have achieved an even better robustness. This example exposes the weakness in the way of applying hill-climbing optimization—as hill climbing treats variables (that identify input signals) of different stages independent, it does not maintain those ones that result in good prefixes.

In addition, it brings about another problem of state space explosion: if we denote

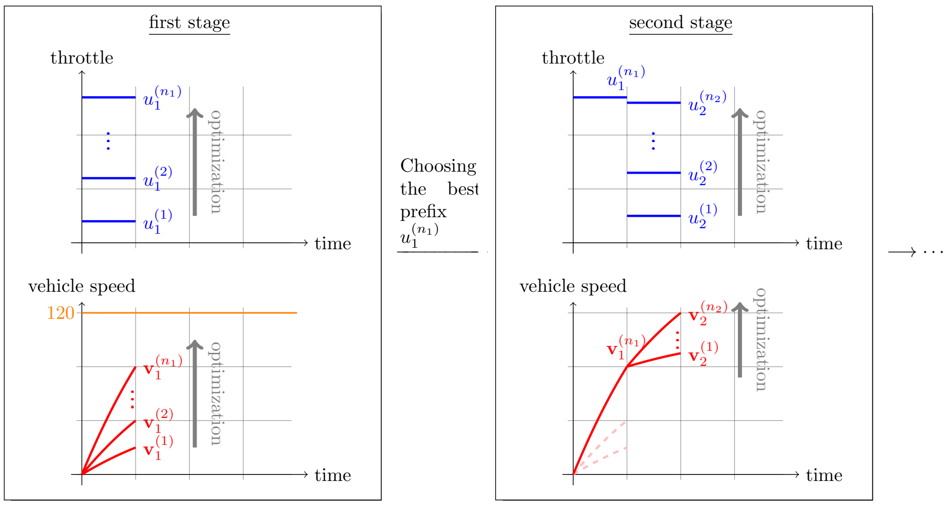


Figure 3.2: The time-staging strategy

U as a sub-space of one interval, then the size of the entire search space is $|U|^{\mathsf{K}}$ where K is the number of control points; apparently, it increases exponentially with respect to K.

3.1.1 Time-Staging Approach

We proposed a time-staging approach to mitigate the problem. *Time staging* refers to discretizing the time domain of the input signal $\mathbf{u}$ and then searching for each segment incrementally. In the case that $\mathbf{u}$ is piecewise constant, a natural way of discretization is to break it at the K control points.

The algorithm is shown in Alg. 3.1. The function TIME-STAGE starts with an empty signal $\mathbf{u}$. Then it calls the function HILL-CLIMB-FALSIFY for K times; at each loop, HILL-CLIMB-FALSIFY figures out the best signal segment $\mathbf{u}'$, and concatenate it to $\mathbf{u}$. Finally, if the constructed $\mathbf{u}$ is a falsifying input, then just return it; otherwise, report a failure of the trial. The function HILL-CLIMB-FALSIFY differs from Alg. 2.1 a bit in the following aspects: it has one more input argument $\mathbf{u}^*$, which drives the model to an initial state; the signal that HILL-CLIMB searches for in Line 11 is not a complete one but a segment; the returned $\mathbf{u}'$ is a signal segment that results in the smallest robustness over the loops rather than a falsifying input. This process is also illustrated by Fig. 3.2, in which the optimization algorithm searches for the best signal segment at

Algorithm 3.1 The time-staging approach for falsification

```
Require: a system model $\mathcal{M}$ that accepts input signal $\mathbf{u} : [0,T] \to \mathbb{R}^M$ and gives output signal
    $\mathcal{M}(\mathbf{u}) : [0,T] \to \mathbb{R}^N$, an STL formula $\varphi$, the time budget $\mathbf{K} \in \mathbb{R}_+$ and the number of control points
    (time stages) $\mathsf{K} \in \mathbb{N}$

 1: function TIME-STAGE($\mathcal{M}, \varphi, \mathbf{K}, \mathsf{K}$)
 2:     $\mathbf{u} \leftarrow ()$                                        ▷ start with the empty signal ()
 3:     for $j \in \{1, \dots, \mathsf{K}\}$ do
 4:         $\mathbf{u}' \leftarrow$ HILL-CLIMB-FALSIFY($\mathcal{M}, \varphi, \mathbf{u}, \frac{\mathbf{K}}{\mathsf{K}}$)          ▷ synthesizing the $j$-th input segment
 5:         $\mathbf{u} \leftarrow \mathbf{u} \cdot \mathbf{u}'$                                    ▷ concatenate $\mathbf{u}'$
 6:     Return $\begin{cases} \mathbf{u} & \text{if } \mathsf{rb}_k = [\![\mathcal{M}(\mathbf{u}), \varphi]\!] < 0 \\ \text{Failure} & \text{otherwise, that is, no falsifying input found within budget } K \end{cases}$

 7: function HILL-CLIMB-FALSIFY($\mathcal{M}, \varphi, \mathbf{u}^*, \mathbf{K}$)
 8:     $\mathsf{rb} \leftarrow \infty$ ;   $\mathbf{u}' \leftarrow ()$ ;   $k \leftarrow 0$     ▷ rb, $\mathbf{u}'$ record the best robustness and signal segment so far
 9:     while $\mathsf{rb} \geq 0$ and $k \leq \mathbf{K}$ do
10:         $k \leftarrow k + 1$
11:         $u_k \leftarrow$ HILL-CLIMB$\Big( \big( u_l, [\![\mathcal{M}(\mathbf{u}^* \cdot u_l), \varphi]\!] \big)_{l \in [1, k-1]} \Big)$     ▷ HILL-CLIMB searches for a segment
12:         $\mathsf{rb}_k \leftarrow [\![\mathcal{M}(\mathbf{u}^* \cdot u_k), \varphi]\!]$
13:         if $\mathsf{rb}_k < \mathsf{rb}$ then                                ▷ update rb and $\mathbf{u}'$ if applicable
14:             $\mathsf{rb} \leftarrow \mathsf{rb}_k$
15:             $\mathbf{u}' \leftarrow u_k$
16:     Return $\mathbf{u}'$                                   ▷ return the $\mathbf{u}$ with the least robustness
```

each interval, and concatenate them in the end.

This approach makes use of the time causality: it maintains the best prefix at each stage, and proceeds based on that. In this way, the search is more efficient. In addition, this approach reduces the search space, from $|U|^{\mathsf{K}}$ to $\mathsf{K} \cdot |U|$, because the search does not take place at all stages simultaneously, but in a one-by-one manner.

Apparently, this strategy is greedy. It does not work in such an occasion, that is, the best prefix is not included in any falsifying input. Actually, this is common, especially when it comes to complicated systems or properties. This gives rise to a new problem, i.e., the trap of "local optimum" brought from time staging.

In order to solve this problem, an ideal way is to introduce a "backtracking" mechanism to this approach: when the concatenated signal is not a falsifying input, we backtrack to one stage earlier to search for a substitution, and then proceed again. This derives a tree search structure, as shown in Fig. 3.3: the hierarchy of the tree comes from time staging; and each path identifies a signal prefix. This idea is also challenging, in that the tree has infinitely many paths and thus it is impossible to explore the whole

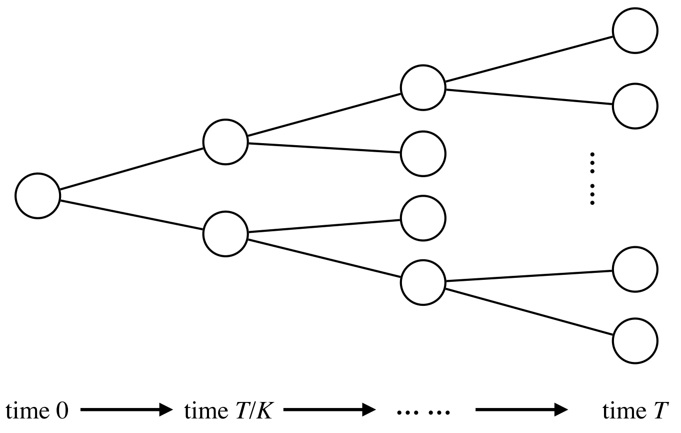


Figure 3.3: A hypothetical tree that implements "backtracking"

tree. An "intelligent" technique is needed that helps identify the essential parts of the tree and explore only there within the limited budget.

In the following sections, we introduce a falsification technique based on Monte Carlo Tree Search, which solves the problem.

3.2 Falsification with Monte Carlo Tree Search

In this section we present the main contribution of this chapter, namely, a two-layered optimization framework for hybrid system falsification. It combines: *Monte Carlo tree search* (MCTS) for high-level planning in the upper layer; and hill-climbing optimization for local search in the lower layer. The upper layer steers the lower layer using the UCT strategy [93], an established method in machine learning for balancing exploration and exploitation. We present two algorithms: the *basic* two-layered algorithm (Alg. 3.2), and a version enhanced with *progressive widening* (Alg. 3.4).

3.2.1 Monte Carlo Tree Search

We firstly give an introduction to Monte Carlo Tree Search (MCTS) in this section. MCTS is a famous algorithm in the artificial intelligence community. Especially in recent years, its successful application on the AlphaGo[1] [94], the AI board game player

[1]https://deepmind.com/research/case-studies/alphago-the-story-so-far

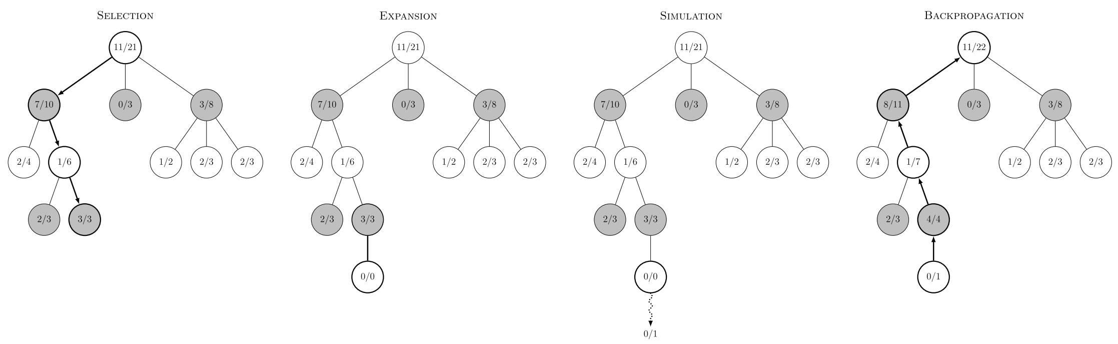


Figure 3.4: An example of the workflow of MCTS

who beat the world champion Lee Sedol[1], leads to prevalence of the algorithm.

MCTS has a long history, but nowadays it mainly refers to the *Upper Confidence Bounds applied to Trees (UCT)* [93] algorithm. It concerns with a problem of search in a huge tree. In the basic setting, the tree yields a huge search space, but its number of nodes is still finite. The aim of MCTS is to select the most promising child of the root node to move, with which the search can gain the best reward. Here, the definition of reward depends on the problem context. For example, in the Go game, reward is correlated to the result of the game, i.e., winning or losing. In the following paragraphs, we use the example in Fig. 3.4[2] to elaborate on how MCTS works. In the figure, there is a fraction attached with each node; the denominator is the number of total visits to that node, while the numerator is the number of winnings. Therefore, the fractions represent the rewards of nodes.

Selection The first step is to select a child node to reason about. Initially, only the root node is available; when there are several children that have already been expanded, the selection is based on the *Upper Confidence Bound version 1 (UCB1)* [95] algorithm. It selects the best candidate according to the following equation:

$$a_{best} = \arg\max_{a \in A} \left(R_a + c\sqrt{\frac{2 \ln N}{N_a}} \right)$$

[1]https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol
[2]The figure is from https://en.wikipedia.org/wiki/Monte_Carlo_tree_search

where A is the set of children, R_a is the reward of child a, c is a scalar parameter, and N, N_a are the numbers of visits to the parent and child a respectively. This algorithm exemplifies the balance between exploration and exploitation: on the one hand, if a child holds a good reward, that one is more likely to be exploited further; on the other hand, if a child has not been visited much often, it will get a better chance to be explored in the future. This balance can also be adjusted by tuning c, so that the user can impose a bias on either exploration or exploitation. For example in Fig. 3.4, when the search visits the root, there are three children to be selected; by applying UCB1, it selects the one with the best reward 7/10.

Expansion If all the children of a selected node have been expanded, then it just continues selection; otherwise, an unexpanded child will be expanded. This is usually done by choosing randomly from the list of the unexpanded children. Then, the expanded child will be initialized with the reward 0/0, as shown in Fig. 3.4.

Simulation (a.k.a. Playout or Rollout) Simulation aims to evaluate a node that has just been expanded. This is by applying Monte Carlo methods: firstly, it collects the sequence of nodes from the root to the node just expanded; then this sequence is concatenated by the nodes randomly chosen from the lower layers of the tree, ending with a leaf node. The sequence can be evaluated according to the context. For example, in a board game, this sequence identifies a way how a player plays the game, and thus a result, either “winning” or “losing”, comes out. In the case of “winning”, we update the numerator of the reward with 1; otherwise, it will be 0. In both cases, the number of visits will be updated to 1. In the example of Fig. 3.4, it loses the game, so the reward is updated to 0/1.

Backpropagation In this step, the information obtained from simulation is used to update the nodes along the path from the root to the expanded node. Namely, for all the nodes along that path, their denominators of rewards will be added by 1, and their numerators of rewards will be added by 0 or 1 depending on the simulation result.

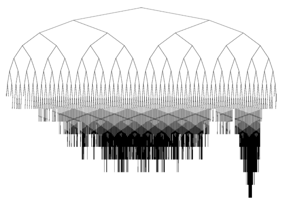


Figure 3.5: Asymmetric tree growth

MCTS repeats this loop for many times, until reaching the given budget. Then, it can select the best child of the root that has the best reward as the output of the algorithm. The generated tree in the end is an asymmetric tree, as the one shown in Fig. 3.5[1]: it has rich branches in those promising areas; meanwhile it does take care of other areas, though barely.

Although MCTS is an effective search method, it is not trivial to apply it in the falsification context. There are several challenges as listed below:

- Falsification reasons about an infinite search space, which is a different setting from the original one of MCTS. Therefore, it is a problem how to build up the search tree.
- Intuitively, the concept of "winning" in falsification may refer to finding a falsifying input. However, the latter is the ultimate goal in falsification, which is rare and hard to reach. Therefore, rewards must be redefined.
- Following up the last item, it is also a challenge to perform simulation (playout) in this context in order to evaluate each node.

In the following sections, we address these issues with the proposal of our two-layered optimization framework.

3.2.2 The Basic Two-Layered Algorithm

We start with the basic two-layered algorithm.

Time staging Similar to the method presented in §3.1, we search for a falsifying input signal, focusing on piecewise constant signals (see Def. 2). The interval $[0, T]$ is divided into K equal sub-intervals, where K is the number of control points. We call this discretization to the time domain *time staging*. Our goal is therefore to find a sequence $\mathbf{u}_1, \dots, \mathbf{u}_{\mathsf{K}}$, where each $\mathbf{u}_i = (u_{i1}, \dots, u_{iM})$ is an M-dimensional real vector (M is the number of input signal dimensions for the model $\mathcal{M}$), so that the corresponding piecewise constant signal is a falsifying one (Def. 9).

We assume intervals $I_i = [u_i^{\min}, u_i^{\max}]$, $i \in \{1, \dots, M\}$, for the ranges of input $u_1, \dots, u_M$ of the model $\mathcal{M}$.

[1]The figure is from [96].

Algorithm 3.2 *Basic* Two-Layered Algorithm

```
Require: a system model $\mathcal{M}$, an STL formula $\varphi$, intervals $I_i = [u_i^{\min}, u_i^{\max}]$, $i \in \{1, \dots, M\}$, for the
    ranges of input $u_1, \dots, u_M$ of $\mathcal{M}$, time horizon $T \in \mathbb{R}_+$, and the following tunable parameters: the
    number K of control points, the number $L_i$ of partitions of the input range $[u_i^{\min}, u_i^{\max}]$ for each
    $i \in \{1, \dots, M\}$, the scalar $c$ in Line 2 of Alg. 3.3, and an MCTS budget (the maximum number of
    MCTS samples, Line 9)
 1: function MCTSPreprocess
 2:     $A \leftarrow \{1, \dots, L_1\} \times \cdots \times \{1, \dots, L_M\}$          ▷ the set of actions
 3:     $\mathcal{T} \leftarrow \{\varepsilon\}$          ▷ the MCTS search tree, initially root-only
 4:     $N \leftarrow (\varepsilon \mapsto 0)$          ▷ visit count $N$ initialized, defined only for $\varepsilon$
 5:     $R \leftarrow (\varepsilon \mapsto \infty)$          ▷ reward function $R$ initialized
 6:     $\overrightarrow{\mathbf{u}} \leftarrow$ null          ▷ place holder for a falsifying input
 7:     $R_{\min} \leftarrow \infty$          ▷ place holder for a minimum reward
 8:     $\overrightarrow{a}_{\min} \leftarrow$ null          ▷ the most promising action sequence
 9:     while $R(\varepsilon) \geq 0$ and within the MCTS budget do
10:         MCTSSample($\varepsilon$)
11:     if $\overrightarrow{\mathbf{u}} \neq$ null then  Return $\overrightarrow{\mathbf{u}}$          ▷ a falsifying input is found already in preprocessing
12:     else Return $\overrightarrow{a}_{\min}$          ▷ return the most promising action sequence

13: function MCTSSample($w$)          ▷ let $w = a_1 \dots a_d$ with $a_i \in A$
14:     $N(w) \leftarrow N(w) + 1$
15:     if $|w| <$ K then
16:         if $wa' \in \mathcal{T}$ for all $a' \in A$ then          ▷ if all children have been expanded
17:             $a \leftarrow$ UCBSample($w$)          ▷ pick a child $wa$ by UCB
18:             MCTSSample($wa$)          ▷ recursive call
19:             $R(w) \leftarrow \min_{a' \in A} R(wa')$          ▷ back-propagation
20:         else
21:             randomly sample $a \in A$ from $\{a \mid wa \notin \mathcal{T}\}$          ▷ expand a random unexpanded child $wa$
22:             $\mathcal{T} \leftarrow \mathcal{T} \cup \{wa\}$
23:             $\mathbf{u}_1, \dots, \mathbf{u}_\mathrm{K} \leftarrow \displaystyle\mathop{\arg\min^{\mathrm{HillClimb}}}_{\substack{\mathbf{u}_1 \in \mathrm{Reg}(a_1), \dots, \mathbf{u}_d \in \mathrm{Reg}(a_d), \\ \mathbf{u}_{d+1} \in \mathrm{Reg}(a), \\ \mathbf{u}_{d+2}, \dots, \mathbf{u}_\mathrm{K} \in I_1 \times \cdots \times I_M}} [\![\mathcal{M}(\mathbf{u}_1 \dots \mathbf{u}_\mathrm{K}), \varphi]\!]$          ▷ playout by hill-climbing
24:             $N(wa) \leftarrow 0$
25:             $R(wa) \leftarrow [\![\mathcal{M}(\mathbf{u}_1 \dots \mathbf{u}_\mathrm{K}), \varphi]\!]$
26:             if $R(wa) < 0$ then
27:                 $\overrightarrow{\mathbf{u}} \leftarrow \mathbf{u}_1 \dots \mathbf{u}_\mathrm{K}$          ▷ a falsifying input is found and stored in $\overrightarrow{\mathbf{u}}$
28:             if $R(wa) < R_{\min}$ then
29:                 $R_{\min} \leftarrow R(wa)$
30:                 $\overrightarrow{a}_{\min} \leftarrow a_1 \dots a_d a$
31:             $R(w) \leftarrow \min_{a' \in A} R(wa')$          ▷ back-propagation

32: function Main
33:     $\overrightarrow{x} \leftarrow$ MCTSPreprocess
34:     if $\overrightarrow{x} = \overrightarrow{\mathbf{u}}$, an input signal then          ▷ Line 11
35:         Return $\overrightarrow{\mathbf{u}}$
36:     else          ▷ $\overrightarrow{x} = a_1 a_2 \dots a_{\mathrm{K}'} \in A^*$ with some $\mathrm{K}' \leq \mathrm{K}$, Line 12
37:         Return $\displaystyle\mathop{\arg\min^{\mathrm{HillClimb}}}_{\substack{\mathbf{u}_1 \in \mathrm{Reg}(a_1), \dots, \mathbf{u}_{\mathrm{K}'} \in \mathrm{Reg}(a_{\mathrm{K}'}), \\ \mathbf{u}_{\mathrm{K}'+1}, \dots, \mathbf{u}_\mathrm{K} \in I_1 \times \cdots \times I_M}} [\![\mathcal{M}(\mathbf{u}_1 \dots \mathbf{u}_\mathrm{K}), \varphi]\!]$
```

Algorithm 3.3 Auxiliary Functions for Algs. 3.2 & 3.4

```
1: function UCBSAMPLE($w$)
2:     Return $\arg\max_{a \in A} \left( \left(1 - \frac{R(wa)}{\max_{w' \in \mathcal{T}} R(w')}\right) + c\sqrt{\frac{2 \ln N(w)}{N(wa)}} \right)$
3: function REG($a$)    ▷ The input region for an action $a \in A$ is of the form $(k_1, \ldots, k_M)$, see Line 2 of Alg. 3.2
4:     Return $\prod_{i=1}^{M} \left[ u_i^{\min} + \frac{k_i - 1}{L_i}(u_i^{\max} - u_i^{\min}),\ u_i^{\min} + \frac{k_i}{L_i}(u_i^{\max} - u_i^{\min}) \right]$
```

The search tree A search tree in MCTS has a branching degree $|A|$, where the set A is called an *action set* in the MCTS literature. In Go, for example, an action set A consists of possible moves.

We use, as the action set A, a *partitioning* of the input space $I_1 \times \cdots \times I_M$. We partition the input space into $L_1 \times \cdots \times L_M$ hyperrectangles of equal size, according to predetermined parameters $L_1, \ldots, L_M$, where M is the number of input signals of the system and L_i indicates how finely the i-th input should be partitioned. In Fig. 3.6 we present an example where $M = 2$ and $L_1 = L_2 = 2$. There we have four actions in the set A, corresponding to the four square regions.

An edge in our search tree represents a choice of an input region—from which we choose the input value $\mathbf{u}_i$—for a single control point $\frac{(i-1)T}{\mathsf{K}}$. The depth of the tree is K (the number of control points). We follow the usual convention and specify a node of a $|A|$-branching tree by a word $w = a_1 a_2 \ldots a_j$ over the alphabet A, where $j \leq \mathsf{K}$. That is: the root is ε (the empty word), its child in the direction $a_1 \in A$ is a_1, its children are $a_1 a_1, a_1 a_2, \ldots$, and so on.

In general, a node in an MCTS search tree is decorated by two values: *reward R* and *visit count N*. In our case, R stores the current estimate of the smallest (i.e. the best) robustness value. Both values are updated explicitly during backpropagation (see below).

Monte Carlo Tree Search sampling Much like usual MCTS, Alg. 3.2 iteratively expands the search tree $\mathcal{T}$. Initially the tree $\mathcal{T}$ is root-only (Line 3), and in each iteration—called *MCTS sampling*—the invocation of MCTSSAMPLE on Line 10 adds one new node to $\mathcal{T}$. In the MCTS literature, *expanding* a child means adding the child to $\mathcal{T}$. We repeat MCTS sampling until a counterexample is found, or the MCTS budget is

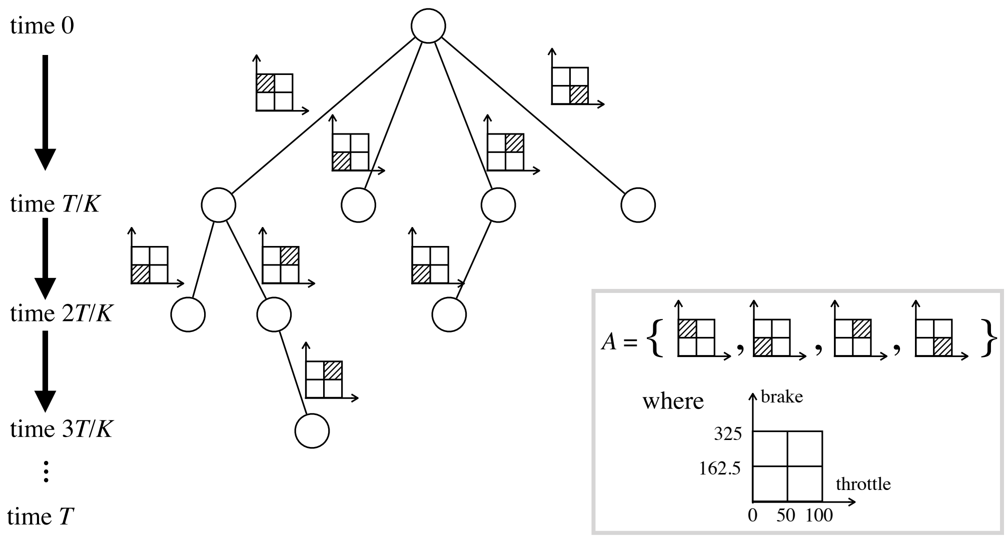


Figure 3.6: Our MCTS search tree for a system model $\mathcal{M}$ with two input signals, throttle and brake, whose ranges are $[0, 100]$ and $[0, 325]$, respectively. We partition each range into two intervals, i.e. $L_1 = L_2 = 2$, hence the branching degree $|A|$ is 2×2.

used up after the maximum number of iterations (Line 9).

The exploration-exploitation trade-off in MCTS comes in the choice of the node to add. In each MCTS sampling, we start from the root (Line 10), walk down the tree $\mathcal{T}$, choosing already expanded nodes (Lines 17–18), until we expand a child (Lines 21–22). Growing a wider tree means exploration, while a deeper tree means exploitation.

We use the UCT strategy [93], the most commonly used strategy in MCTS, to resolve the dilemma. UCT is based on the UCB strategy for the multi-armed bandit problems [95]; Line 2 of Alg. 3.3 follows UCB1, where the exploitation score $1 - \frac{R(wa)}{\max_{w' \in \mathcal{T}} R(w')}$ and the exploration score $\sqrt{\frac{2 \ln N(w)}{N(wa)}}$ are superposed using a scalar c. Recall that our rewards $R(wa)$ for w's children are given by robustness estimates from previous simulations, and that falsification favors smaller R. Note also that values of R can be greater than 1. In the exploitation score $1 - \frac{R(wa)}{\max_{w' \in \mathcal{T}} R(w')}$, therefore, we normalize rewards to the interval $[0, 1]$ and reverse their order.[1] The exploration score $\sqrt{\frac{2 \ln N(w)}{N(wa)}}$ is taken from UCB1: the visit count $N(w)$ gives how many times the node w has been visited, that is, how many offspring the node w currently has in $\mathcal{T}$. The scalar, for the trade-off, is a tunable parameter, as usual in MCTS.

[1]We can assume nonnegative values of R , otherwise we already have a falsifying input.

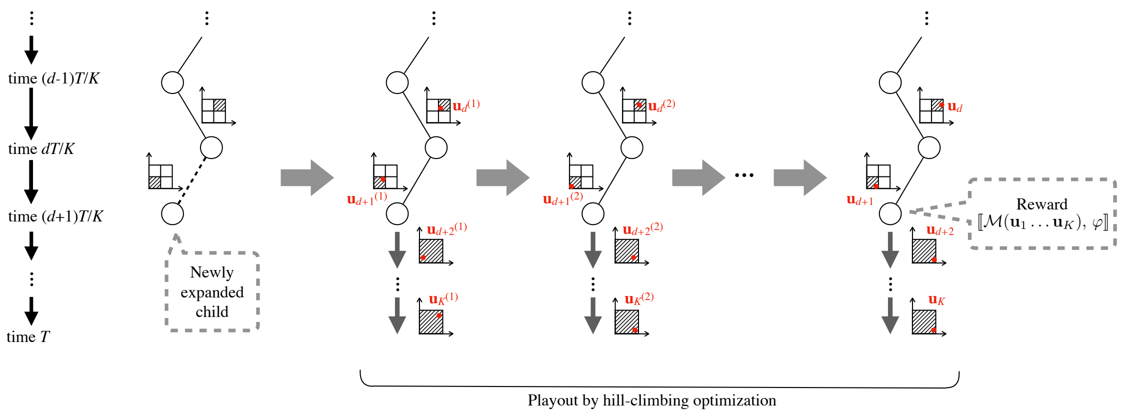


Figure 3.7: Playout by hill-climbing optimization

Playout and backpropagation In MCTS, the reward of a newly expanded node $a_1 a_2 \dots a_d a$ (see e.g. Line 22) is computed by an operation called *playout* (or *simulation* as in §3.2.1). The result is then *back-propagated*, in a suitable manner, to the ancestors: $a_1 \dots a_d, a_1 \dots a_{d-1}, \dots$, and finally ε.

In our MCTS algorithms for falsification we use hill-climbing optimization (e.g. SA, GNM and CMA-ES) for playout. See Line 23, where input values $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_K$ are sampled by stochastic hill-climbing optimization, so that the resulting robustness value of the specification φ becomes smaller. The regions from which to sample those values are dictated by the MCTS tree: $\mathbf{u}_1 \in \textsc{Reg}(a_1), \dots, \mathbf{u}_d \in \textsc{Reg}(a_d)$ follow the actions $a_1, \dots, a_d$ determined so far (here $\textsc{Reg}$ is from Alg. 3.3); $\mathbf{u}_{d+1} \in \textsc{Reg}(a)$ follows the newly chosen action a (Line 21); and the remaining values $\mathbf{u}_{d+2}, \dots, \mathbf{u}_K$ can be chosen from the whole input range $I_1 \times \cdots \times I_M$.

Figure 3.7 illustrates an example of playout by hill-climbing optimization. Smaller gray squares represent actions, and red dots represent input values (notice that they are chosen from the gray regions). The values $\mathbf{u}_1, \dots, \mathbf{u}_K$ are sampled repeatedly so that the robustness value $[\![\mathcal{M}(\mathbf{u}_1 \dots \mathbf{u}_K), \varphi]\!]$ becomes smaller.

An intuition of this playout operation is that we sample the best input signal, $\mathbf{u}_1 \dots \mathbf{u}_K$, under the constraints imposed by the MCTS search tree (namely, the input regions prescribed by the actions). The least robustness value thus obtained is assigned to the newly expanded node wa as its reward (Line 25). If $R(wa) < 0$ then this means we have already succeeded in falsification (Line 27).

Backpropagation is an important operation in MCTS. Following the intuition that

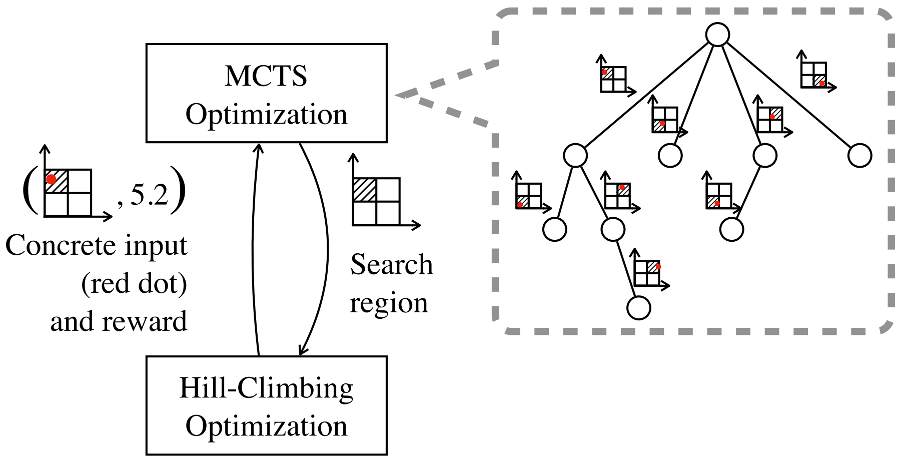

MCTS Optimization
, 5.2
Concrete input (red dot) and reward
Search region
Hill-Climbing Optimization

Figure 3.8: Our two-layered optimization framework

the reward $R(w)$ is the smallest robustness achievable at the node w, we define the reward of an internal node w by the minimum of its children's rewards. See Lines 19 and 31. Note that, via recursive calls of MCTSSAMPLE (Line 18), the result of playout is propagated to all ancestors.

A two-layered framework In Alg. 3.2, hill-climbing optimization occurs twice, in Lines 23 and 37. The first occurrence is in playout of MCTS—this way we interleave MCTS optimization (by growing a tree) and hill-climbing optimization. See Fig. 3.8. MCTS optimization is considered to be a *preprocessing* phase in Alg. 3.2 (Line 33): its principal role is to find an action sequence $\overrightarrow{a}_{\mathrm{min}}$, i.e. a sequence of input regions, that is most promising. In the remainder of the MAIN function, the second hill-climbing optimization is conducted for falsification, where we sample according to $\overrightarrow{a}_{\mathrm{min}}$.

The two occurrences of hill-climbing optimization therefore have different roles. Given also the fact that the first occurrence is repeated every time we expand a new child, we choose to spend less time for the former than the latter. In our implementation, we set the timeout to be 5–15 seconds for the first hill-climbing sampling in Line 23 ($\mathrm{TO}_{\mathrm{po}}$ in §3.3), while for the second hill-climbing sampling in Line 37 the timeout is 300 seconds.

A falsifying input $\overrightarrow{\mathbf{u}}$ is often found already in the preprocessing phase. In this case the MAIN function simply returns $\overrightarrow{\mathbf{u}}$ (Line 35).

3.2.3 The Two-Layered Algorithm with Progressive Widening

Our second algorithm (Alg. 3.4) differs from the basic one (Alg. 3.2) in two ways:

Algorithm 3.4 Two-Layered Algorithm with *Progressive Widening*

```
Require: The algorithm is the same as Alg. 3.2, except that the function MCTSSAMPLE is replaced by
    the following one.
Require: The same data as required in Alg. 3.2, and additionally, constants $C, \alpha$ (used in Line 4)
 1: function MCTSSAMPLE($w$)                                        ▹ let $w = a_1 \dots a_d$ with $a_i \in A$
 2:     $N(w) \leftarrow N(w) + 1$
 3:     if $|w| < K$ then
 4:         if $\left( \begin{array}{l} \left|\{a' \in A \mid wa' \in \mathcal{T}\}\right| \geq C \cdot N(w)^{\alpha} \\ \text{or } wa' \in \mathcal{T} \text{ for all } a' \in A \end{array} \right)$ then      ▹ progressive widening
 5:             $a \leftarrow$ UCBSAMPLE($w$)                             ▹ pick a child $wa$ by UCB
 6:             MCTSSAMPLE($wa$)                                        ▹ recursive call
 7:             $R(w) \leftarrow \min_{a' \in A} R(wa')$                          ▹ back-propagation
 8:         else
 9:             $S \leftarrow$ (a maximal convex subset of $\bigcup_{wa' \notin \mathcal{T}} \text{REG}(a')$)
10:             $\mathbf{u}_1, \dots, \mathbf{u}_K \leftarrow \mathop{\arg\min^{\text{HillClimb}}}_{\substack{\mathbf{u}_1 \in \text{REG}(a_1), \dots, \mathbf{u}_d \in \text{REG}(a_d), \\ \mathbf{u}_{d+1} \in S, \\ \mathbf{u}_{d+2}, \dots, \mathbf{u}_K \in I_1 \times \cdots \times I_M}} [\![\mathcal{M}(\mathbf{u}_1 \dots \mathbf{u}_K), \varphi]\!]$   ▹ playout by hill-climbing
11:             $a \leftarrow \big(a \in A$ such that $\mathbf{u}_{d+1} \in \text{REG}(a)\big)$
12:             $\mathcal{T} \leftarrow \mathcal{T} \cup \{wa\}$
13:             $N(wa) \leftarrow 0$
14:             $R(wa) \leftarrow [\![\mathcal{M}(\mathbf{u}_1 \dots \mathbf{u}_K), \varphi]\!]$
15:             if $R(wa) < 0$ then
16:                 $\overrightarrow{\mathbf{u}} \leftarrow \mathbf{u}_1 \dots \mathbf{u}_K$
17:             if $R(wa) < R_{\min}$ then
18:                 $R_{\min} \leftarrow R(wa)$
19:                 $\overrightarrow{a}_{\min} \leftarrow a_1 \dots a_d a$
20:             $R(w) \leftarrow \min_{a' \in A} R(wa')$                          ▹ back-propagation
```

Progressive widening Alg. 3.4 uses *progressive widening* [97]; see Line 4. Unlike in the basic algorithm (Line 16 of Alg. 3.2), we do not always expand a new child, even if there are unexpanded ones; the threshold $C \cdot N(w)^{\alpha}$ is computed using the visit count $N(w)$ and tunable parameters C, α.

Progressive widening is a widely employed technique in MCTS for coping with a large or infinite action set A—in such a case expanding all children incurs a lot of computational cost. See e.g. [48]. In our Alg. 3.4 the action set A can be large, depending on the numbers $L_1, \dots, L_m$ of input range partitions.

Hill-climbing optimization for expanding children In progressive widening, since we may not expand all the children, it makes sense to be selective about which child to expand. This is in contrast to random sampling in Alg. 3.2 (Line 21). See Line 10 of Alg. 3.4, where we first playout by hill-climbing optimization. The value $\mathbf{u}_{d+1}$ thus obtained is then used to determine which child wa to expand, in Line 11. In order to ensure that the new child wa is indeed previously unexpanded, the value $\mathbf{u}_{d+1}$ is sampled from the set $\bigcup_{wa' \notin \mathcal{T}} \textsc{Reg}(a')$; in fact, we restrict to its convex subset (Line 9), because many hill-climbing optimization algorithms work best in a convex domain. See Fig. 3.9 for illustration.

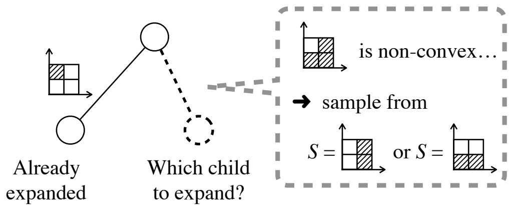


Figure 3.9: Lines 9–11 of Alg. 3.4

3.3 Experimental Evaluation

We have implemented our *basic* algorithm (Alg. 3.2, denoted "Basic") and our *progressive widening* algorithm (Alg. 3.4, denoted "P.W.") in MATLAB, using Breach [22] as a front-end for hill-climbing optimization and for its implementation of the robust semantics.

The experiments have two goals. Firstly, in §3.3.2, we evaluate the falsification performance of our proposal in comparison to the state-of-the-art. Since our MCTS enhancement emphasizes coverage, our interest is in the success rate in hard problem instances rather than in execution time. Secondly, in §3.3.3, we evaluate the impact of different choices of parameters for our algorithms (such as the UCB scalar c in Alg. 3.3).

3.3.1 Experiment Setup

The experiments are based on the following benchmarks.

The *automatic transmission* (AT) model is exactly the Simulink model in Fig. 1.1, and it was proposed as a benchmark for falsification in [98]. It has input signals *throttle* $\in [0, 100]$ and *brake* $\in [0, 325]$, and computes the car's speed *speed*, the engine rotation *rpm*, and the selected gear *gear*. We consider the following specifications, taken in part from [98].

S1 $\equiv \Box_{[0,30]}\ (\mathit{speed} < 120)$ can be falsified easily by hill climbing with an input $\mathit{throttle} = 100$ and $\mathit{brake} = 0$ throughout.

S2 $\equiv \Box_{[0,30]}\ (\mathit{gear} = 3 \longrightarrow \mathit{speed} \geq 20)$ states that in gear three, the speed should not get too low. The difficulty arises from the lack of guidance by robustness as long as $\mathit{gear} \neq 3$: we follow [98] and take $\mathit{gear} = 1, \ldots, \mathit{gear} = 4$ as Boolean propositions, instead of taking *gear* as a numeric variable. In contrast to [98], we use a more difficult speed threshold of 20 instead of 30.

S3 $\equiv \Diamond_{[10,30]}\ (\mathit{speed} \notin [53, 57])$ states that it is not possible to maintain a constant speed after 10s. A falsifying trace needs precise inputs to hit and maintain the narrow speed range.

S4 $\equiv \Box_{[0,29]}(\mathit{speed} < 100) \lor \Box_{[29,30]}(\mathit{speed} > 65)$ is a specification designed to demonstrate the limitation of robustness-guided falsification by hill-climbing optimization only. Here, a falsifying trajectory has to reach high speed before braking. Similarly to S2, the speed 100 has to be reached much earlier than the indicated time bound of 29 to give sufficient time for deceleration. However, by using the maximum as semantics for the $\lor$-connective, the robustness computation can shadow either of the disjuncts .

S5 $\equiv \Box_{[0,30]}(\mathit{rpm} < 4770 \lor \Box_{[0,1]}(\mathit{rpm} > 600))$ aims to prevent systematic sudden drops from high to low *rpm*. It is falsified if an *rpm* peak above 4770 is immediately followed by a drop to $\mathit{rpm} \leq 600$.

The second benchmark is the *Abstract Fuel Control* (AFC) model [56]. It takes two input signals, *pedal angle* and *engine speed*, and outputs the critical signal *air-fuel ratio* (*AF*), which influences fuel efficiency and car performance. The value is expected to be close to a reference value *AFref*. The pedal angle varies in the range $[0, 61.1]$ and the engine speed varies in the range $[900, 1100]$. According to [56], this setting corresponds to *normal mode*, where $\mathit{AFref} = 14.7$.

The basic requirement of the AFC is to keep the air-to-fuel ratio *AF* close to the reference *AFref*. However, changes to the pedal angle cause brief spikes in the output signal *AF* before the controller is able to regulate the engine. Falsification is used to discover the amplitude and periods of such spikes.

The formal specification Sbasic is $\Box_{[11,30]}(\neg(|\mathit{AF} - \mathit{AFref}| > 0.05 * 14.7))$. It is violated when *AF* deviates from its *AFref* too much. Another specification is Sstable: $\neg(\Diamond_{[6,26]}\Box_{[0,4]}(|\mathit{AF} - \mathit{AFref}| > 0.01 * 14.7))$. The goal is to find spikes where the ratio is off by a fraction 0.01 of the reference value for at least t' seconds during the interval $[6, 26]$.

The third benchmark model is called *Free Floating Robot* (FFR) that has been considered as a falsification benchmark in [69]. It is a robot vehicle powered by four boosters and moving in two spatial dimensions. It is governed by the following second-order differential equations:

$$\begin{aligned}
\ddot{x} &= 0.1 \cdot (u_1 + u_3)\cos(\varphi) - 0.1 \cdot (u_2 + u_4)\sin(\varphi) \\
\ddot{y} &= 0.1 \cdot (u_1 + u_3)\sin(\varphi) + 0.1 \cdot (u_2 + u_4)\cos(\varphi) \\
\ddot{\varphi} &= 5/12 \cdot (u_1 + u_3) - 5/12 \cdot (u_2 + u_4)
\end{aligned}$$

The goal of the robot is to steer from $(x, y, \varphi) = (0, 0, 0)$ to $x = y = 4$, with a tolerance of 0.1, such that $\dot{x}$ and $\dot{y}$ are within $[-1, 1]$, given a time horizon of $T = 5$. The four inputs $u_i \in [-10, 10]$ range over the same domain. We run falsification on the negated requirement: Strap $\equiv \neg \Diamond_{[0,5]}\, x, y \in [3.9, 4.1] \wedge \dot{x}, \dot{y} \in [-1, 1]$.

The experiments use Breach version 1.2.9 and MATLAB R2017b on an Amazon EC2 c4.large instance (2.9 GHz Intel Xeon E5-2666, 2 virtual CPU cores, 4 GB main memory).

3.3.2 Performance Evaluation

The results are shown in Table 3.1 and are grouped with respect to the method: uniform random sampling ("Random") as a baseline, Breach, our "Basic" algorithm (Alg. 3.2) and our "P.W." algorithm (Alg. 3.4), as well as with respect to the underlying hill-climbing optimization solver (CMA-ES, Global Nelder-Mead (GNM) and Simulated Annealing (SA)). Run times are shown in seconds. Since the algorithms are stochastic, we give the success rate out of a number of trials—here it is 10 trials for each experiment.

For all the experiments, input signals are chosen to be piecewise constant, with $\mathsf{K} = 5$ control points for AT and AFC, and $\mathsf{K} = 3$ control points for FFR (due to the shorter time horizon). These numbers coincide with the depth of the MCTS search trees. In Breach, this is achieved with the "UniStep" input generator, with its `.cp` attribute set to K. The timeout for Breach was set to 900 seconds (which is well above all successful falsification trials) with no upper limit on the number of simulations. For our P.W. algorithm, we used the parameters $C = 0.7$ and $\alpha = 0.85$ (Line 4 of Alg. 3.4).

Table 3.1: Comparison of uniform random sampling and Breach against Algs. 3.2 (Basic) and 3.4 (P.W.).

(a) Experimental results of Automatic Transmission

		Parameters			AT model									
					S1		S2		S3		S4		S5	
	Algorithm	M_b	TO_{po}	c	succ.	time	succ.	time	succ.	time	succ.	time	succ.	time
	Random				10/10	108.9	10/10	289.1	1/10	301.1	0/10	-	0/10	-
CMA-ES	Breach				10/10	21.9	6/10	30.3	**10/10**	**193.9**	4/10	208.8	3/10	75.5
CMA-ES	Basic	40	15	0.20	10/10	15.8	10/10	108.5	10/10	697.1	7/10	786.8	9/10	384.4
CMA-ES	P.W.	40	15	0.20	**10/10**	**10.8**	**10/10**	**65.7**	10/10	728.6	**7/10**	**767.8**	**10/10**	**648.1**
GNM	Breach				**10/10**	**5.4**	**10/10**	**151.4**	0/10	-	0/10	-	0/10	-
GNM	Basic	20	5	0.20	10/10	12.4	10/10	162.3	**10/10**	**185.6**	7/10	261.9	7/10	163.7
GNM	P.W.	20	5	0.05	10/10	60.8	9/10	110.7	8/10	211.2	**8/10**	**313.0**	**10/10**	**178.7**
SA	Breach				**10/10**	**160.1**	0/10	-	3/10	383.7	0/10	-	3/10	80.4
SA	Basic	20	15	0.05	10/10	264.8	9/10	236.1	**8/10**	**385.6**	**8/10**	**505.3**	7/10	341.2
SA	P.W.	40	15	0.20	10/10	208.7	**10/10**	**377.6**	8/10	666.0	7/10	795.4	**10/10**	**624.2**

(b) Experimental results of Abstract Fuel Control and Free Floating Robot

		Parameters			AFC model				FFR model	
					Sbasic		Sstable		Strap	
	Algorithm	M_b	TO_{po}	c	succ.	time	succ.	time	succ.	time
	Random				6/10	278.7	10/10	242.6	4/10	409.3
CMA-ES	Breach				**10/10**	**111.7**	3/10	256.3	**10/10**	**119.8**
CMA-ES	Basic	40	15	0.20	10/10	182.0	7/10	336.9	10/10	338.0
CMA-ES	P.W.	40	15	0.20	10/10	177.1	**8/10**	**272.9**	10/10	473.9
GNM	Breach				**10/10**	**171.4**	0/10	-	0/10	-
GNM	Basic	20	5	0.20	10/10	227.1	2/10	378.5	**10/10**	**162.2**
GNM	P.W.	20	5	0.05	10/10	252.0	**6/10**	**153.2**	6/10	197.4
SA	Breach				0/10	-	6/10	307.0	3/10	92.8
SA	Basic	20	15	0.05	5/10	391.3	**8/10**	**273.8**	**10/10**	**273.2**
SA	P.W.	40	15	0.20	**8/10**	**665.7**	6/10	293.7	10/10	390.9

The choice of parameters for our two MCTS-based algorithms is as follows: for each combination with the hill-climbing optimization solvers, we present a set of parameters that give good results over all the specifications. This is justified, because the performance is quite dependent on these parameters, and one choice that works for a given combination of a falsification algorithm and a hill-climbing solver might just not work for another combination. However, note that we do *not* change the settings across the specifications.

As we discuss at the end of §3.2.2, different timeouts are set for hill-climbing in playout (Line 23 of Alg. 3.2) and to hill-climbing at the end (Line 37 of Alg. 3.2). Specifically, the timeout for the former is $\mathrm{TO}_{\mathrm{po}}$ in Table 3.1 (5–15 seconds) while the timeout for the latter is globally 300 seconds. M_b (MCTS budget) is the maximum visit count for the root of the MCTS search tree (i.e. the maximum number of nodes of the tree).

The cells with bold fonts are local best performers w.r.t. each hill-climbing solver, and green backgrounded cells are the global performers w.r.t. each property. Here, the ranking criterion takes success rate as first priority, and average time as second priority.

The results in Table 3.1 indicate, at a high-level, that for seemingly hard problems, the benefit of the extra exploration done by the MCTS layer significantly increases the falsification rate. This is most evident in S4 and S5, where Breach (with any of CMA-ES, GNM or SA) has at most 30–40% success rates. Our MCTS enhancements succeed much more often.

For easy problems, the increased exploration typically increases the falsification times, which is expected. One reason is that falsification is in general a hard problem that can only be tackled by heuristics. We note from Table 3.1 that the additional execution time is often not prohibitively large. We also note that there is generally no single algorithm that works on all instances equally well. For example, for Sstable, both Breach and our algorithms are even weaker than random testing. However, our algorithms still increase the falsification rate compared to Breach.

The choice of a hill-climbing optimization solver has a great influence on the outcome. CMA-ES has built-in support for some exploration before the search converges in the most promising direction. Nevertheless, we see that the upper-layer optimization by MCTS can improve success rates (S4, S5, Sstable). The Nelder-Mead variant GNM has very little support for exploration and furthermore, Breach's implementation is not stochastic (it uses deterministic low-discrepancy sequences as a source of quasi-randomness). For this reason, the method quickly converges to non-falsifying minima that are local and cannot be escaped without extra measures. Thus, using MCTS pays off especially with GNM; see for example S3 and S4. Conversely, SA heavily relies on exploration and keeps just a single good trace found so far, limiting its exploitation. In combination with MCTS, SA shows mixed performance. In some cases falsification time becomes longer (S1, S3), whereas for S4, MCTS is able to overcome this particular limitation, presumably because it maintains several good prefixes. For the free floating robot, we observe that our approach needs additional time in comparison to Breach with CMA-ES (within an order of magnitude), which is reasonable given the added exploration on the exponentially larger state space. However, it does increase the falsification rate with GNM and SA, for the same reasons as before.

The difference between the two variants, Algs. 3.2 and 3.4 (the latter with progressive widening), is not significant on most of the examples. However, progressive widening has a positive effect on the success rate and falsification time for S2 and S5.

In the experiments, we set the MCTS budget (number of iterations of the main loop) to be 20–40. Note that the number of all possible nodes is much greater: it is $(1 + |A| + |A|^2 + \cdots + |A|^{\mathrm{K}})$. For AT and AFC (2 input signals, $L = 3 \times 5$ and $\mathrm{K} = 5$), it is 813616; and for FFR (4 input signals, $L = 2 \times 2 \times 2 \times 2$ and $\mathrm{K} = 3$), it is 4369. The overall success rates seem to suggest that, not only in computer Go but also in hybrid system falsification, MCTS is very effective in searching in a vast space with limited resources.

3.3.3 Evaluation of Parameter Choices

We evaluate the effect of the parameters using the specification S4 for the AT model, where the success of falsification varies strongly. For the experiments in this section we focus on Alg. 3.2 (Basic).

Table 3.2 contains 4 sub-tables, each showing the results for the different optimization solvers when varying a hyperparameter. The results are shown in terms of success rate and average time in seconds for those successful trials. The default parameter settings are: maximum tree size (MCTS budget) is 60, and $c = 0.2$, $L = 2$, $\mathrm{TO}_{\mathrm{po}} = 10$,

$\mathsf{K} = 5$ (gray headed columns). The green cells are the best performers w.r.t. each solver.

The first concern is about the scalar c for exploration/exploitation. We observe that there is a general trend that falsification rate improves with increased focus on exploration. It is particularly evident when comparing the results of $c = 0.02$ and $c = 0.5$. However, no significant performance gap is observed between $c = 0.5$ and $c = 1.0$, indicating that $c = 0.5$ is already sufficient for optimization solvers to benefit from exploration.

Next, consider the results for different partitioning of the input space, where $L = n \times m$ means that the throttle range is partitioned into n actions and the brake range into m actions (for the AT model; pedal and engine for the AFC model). We note that the different choices have much less influence than the scalar c. However, there are some differences, e.g., GNM seems to cope badly with the coarse partitioning 2×2 in the first column, which could be attributed to its reliance on guidance by the MCTS layer.

With respect to the timeout for individual playouts $\mathrm{TO}_{\mathrm{PO}}$, we observe that it is correlated with overall falsification time. This is expected, as we spend more time in non-falsifying regions of the input space as well.

Varying the number of control points K (and therefore the depth of the MCTS tree), shows that for the respective requirement, $\mathsf{K} = 3$ is insufficient but the results for more control points are not clear. As more control points make the problem harder due to the larger search space, the falsification rate drops (specifically for $\mathsf{K} = 10$). Note that we purposely keep the MCTS budget and playout time consistent to expose this effect, whereas in practice one might want to increase the limits when the problem is more complex.

3.4 Discussion

Our algorithms interleave MCTS optimization and hill-climbing optimization: the latter is used in the playout operation of the former, for sampling and estimating the reward of a high-level input-synthesis strategy. This high-level strategy is concretely given by a sequence $a_1 a_2 \ldots a_d$ of input regions. Via the UCT tree search strategy, we ensure that our search in a search tree is driven not only by depth, but also by width. This way we enhance exploration in search-based falsification, in the sense that

Table 3.2: Parameter variation for Alg. 3.2 (Basic)

	$c = 0.02$		$c = 0.2$		$c = 0.5$		$c = 1.0$	
Solver	succ.	time	succ.	time	succ.	time	succ.	time
CMA-ES	6/10	826.1	7/10	728.7	8/10	725.7	9/10	744.3
GNM	0/10	-	4/10	807.3	3/10	779.4	3/10	791.4
SA	1/10	719.5	8/10	733.5	9/10	736.3	8/10	799.1
	$L = 2 \times 2$		$L = 3 \times 3$		$L = 3 \times 5$		$L = 5 \times 5$	
Solver	succ.	time	succ.	time	succ.	time	succ.	time
CMA-ES	7/10	728.7	9/10	674.4	9/10	740.2	8/10	743.4
GNM	4/10	807.3	3/10	712.3	9/10	721.6	10/10	724.2
SA	8/10	733.5	6/10	755.7	8/10	832.0	6/10	832.8
	$\mathrm{TO}_{\mathrm{po}} = 5$		$\mathrm{TO}_{\mathrm{po}} = 10$		$\mathrm{TO}_{\mathrm{po}} = 15$		$\mathrm{TO}_{\mathrm{po}} = 20$	
Solver	succ.	time	succ.	time	succ.	time	succ.	time
CMA-ES	8/10	431.8	7/10	728.7	9/10	776.2	7/10	1330.1
GNM	3/10	502.6	4/10	807.3	4/10	809.4	2/10	1397.1
SA	7/10	510.5	8/10	733.5	7/10	1108.0	8/10	1342.5
	$\mathrm{K} = 3$		$\mathrm{K} = 5$		$\mathrm{K} = 7$		$\mathrm{K} = 10$	
Solver	succ.	time	succ.	time	succ.	time	succ.	time
CMA-ES	0/10	-	7/10	728.7	6/10	711.5	5/10	777.9
GNM	0/10	-	4/10	807.3	1/10	664.3	6/10	892.8
SA	0/10	-	8/10	733.5	8/10	709.7	3/10	750.9

different regions of the input space are sampled in a structured and disciplined manner. It is an interesting topic for future work to quantify the *coverage guarantees* that can potentially be achieved by our approach.

In falsification of hybrid systems, it is often the case that *simulation*, i.e. running a model $\mathcal{M}$ under a given input signal, is computationally the most expensive operation. In our algorithm this occurs in Lines 23 and 37, since a hill-climbing optimization algorithm tries many samples of $\mathbf{u}_1, \ldots, \mathbf{u}_\mathrm{K}$. Simplifying Line 23, e.g. by decimating the control points, can result in a useful variation of our algorithm.

Among the tunable parameters of the algorithm is the scalar c, used for the UCB sampling (Line 2 of Alg. 3.3). Having this parameter is unique to our falsification framework, in comparison to simple robustness-guided optimization (with hill-climbing only). Specifically, the parameter c endows our algorithm with *flexibility* in the exploration-exploitation trade-off. Given the diversity of instances of the hybrid system falsification problem, it is unlikely that there is a single value of c that is optimal for all falsification examples. An engineer can then use her/his expert domain knowledge to tune the parameter c.

4 Multi-Armed Bandits for Boolean Connectives in STL

In this chapter, we tackle the *scale problem* that arises from the definition of STL robust semantics for Boolean connectives (conjunctive and disjunctive) in the existing falsification framework. We propose a novel technique based on the Multi-Armed Bandit (MAB) problem to falsify *safety properties* embedded with Boolean connectives. Our solution consists of integration of the MAB algorithms in hill climbing-guided falsification: it takes different sub-formulas of the Boolean connective as bandit machines, and applies MAB algorithms, namely UCB1 and ε-Greedy, to govern the hill-climbing optimization processes running on different machines. As a result, the MAB algorithms tend to select the machine that is more likely to falsify the property, and therefore the problem can be solved effectively.

We firstly explain the problem caused by the improper definition of robust semantics for Boolean connectives in §4.1, and then present the proposed approach in §4.2.

The material in this chapter is based on [89]

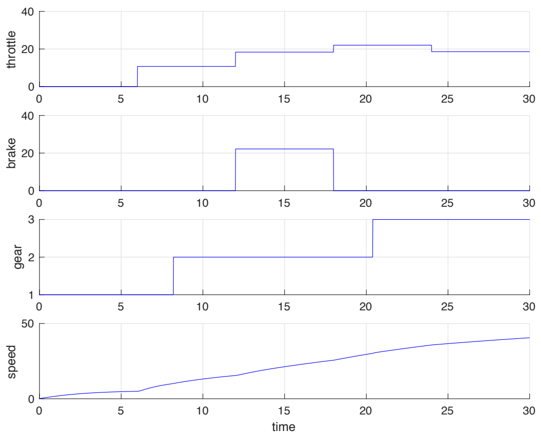
4.1 Motivation: the Scale Problem

Safety properties are a class of properties of great importance. Intuitively, they require that "something bad will never happen". In STL, safety properties are expressed by the "always" modality, i.e., they are in the form $\square(\varphi')$, where φ' refers to that "something bad does not happen". The examples we use in §1.2.1 and Table 1.1 are both safety properties. In industrial practice, safety properties are usually used in combination with *conjunctive* or *disjunctive*. In other words, these two forms of formulas are usually tested against, $\square(\varphi_1 \wedge \varphi_2)$ and $\square(\varphi_1 \vee \varphi_2)$; the former requires a system to satisfy more than one property, while the latter indicates a logical constraint between properties.

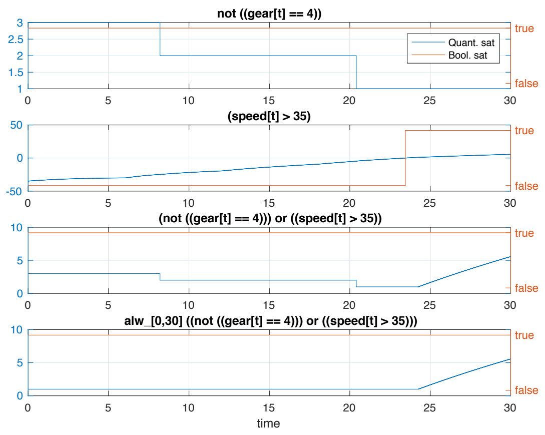
However, the existing falsification framework is problematic in handling such properties. The problem arises from the definition of STL robust semantics that has been presented in §2.2. As robustness plays the role of objective function for hill-climbing optimization, it affects the performance of the framework significantly. We firstly take a glance at the problem via an example.

Recall the usage scenario introduced in §1.2.1; it regards the following property: when the *gear* of the car is 4, the *speed* should not be lower than 35km/h. In STL, it is written as $\varphi \equiv \square(gear = 4 \rightarrow speed > 35)$, and thus equivalent to $\square(\neg(gear = 4) \vee speed > 35)$, i.e., it belongs to the form $\square(\varphi_1 \vee \varphi_2)$. Fig. 4.1 and Fig. 4.2 show two samples, *Sample1* and *Sample2*, that happen chronologically during a falsification process to φ. In each figure, on the left (Fig. 4.1a and Fig. 4.2a) are the input and output signals, and on the right (Fig. 4.1b and Fig. 4.2b) are the Boolean satisfaction and quantitative robustness for sub-formulas. Compare *Sample1* and *Sample2*; in both cases, the robustness to φ is 1. This is troublesome to hill-climbing optimization: recall the way how hill-climbing optimization works (see Alg. 2.1); it compares the robustness of different samples, and figures out the direction for future sampling based on that comparison. In the case that robustness stays the same over different samples, hill climbing will get lost and report a failure.

We analyze the reason why this situation of *flat robustness* happens. The robustness computation processes for *Sample1* and *Sample2* are presented in Fig. 4.1b and Fig. 4.2b respectively. In both cases, the final robustness 1 is obtained from the robustness to φ itself, i.e., the last sub-figures in Fig. 4.1b and Fig. 4.2b. Those two are derived from the second last sub-figures, i.e., the robustness to $\neg(gear = 4) \vee speed > 35$. The

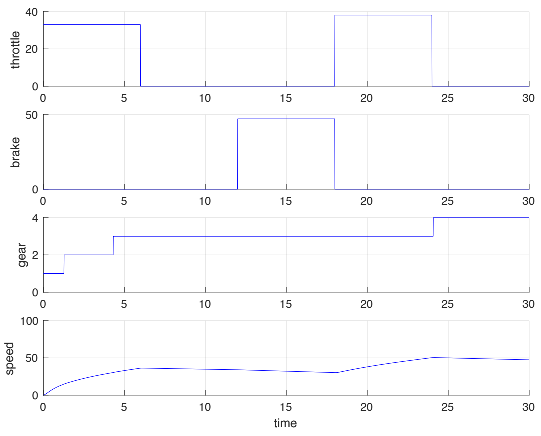


(a) Input signals *throttle*, *brake* and output signals *gear*, *speed* of *Sample1*

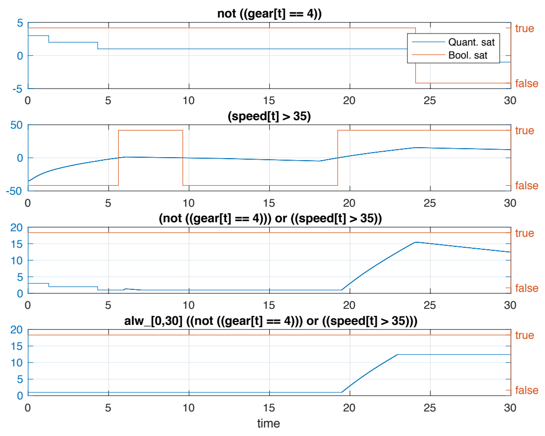


(b) Boolean satisfaction and quantitative robustness of Sample1 to sub-formulas of φ

Figure 4.1: *Sample1*: A sample from hill-climbing optimization during falsification to $\varphi \equiv \Box(gear = 4 \longrightarrow speed > 35)$



(a) Input signals *throttle*, *brake* and output signals *gear*, *speed* of *Sample2*



(b) Boolean satisfaction and quantitative robustness of *Sample2* to sub-formulas of φ

Figure 4.2: *Sample2*: A sample occurring after *Sample1*, from the same process with that in Fig. 4.1

computations of these two ones are interesting: they are based on the robustness to $\neg gear = 4$ and $speed > 35$ (the first 2 sub-figures), and computed by applying the robust semantics for disjunctive in Def. 8:

$$[\![\mathbf{w}, \varphi_1 \vee \varphi_2]\!] \ := \ [\![\mathbf{w}, \varphi_1]\!] \sqcup [\![\mathbf{w}, \varphi_2]\!]$$

Namely, taking the maximum of the robustness of two sub-formulas. Therefore, the robustness to $\neg(gear = 4) \vee speed > 35$ (the third sub-figures in Fig. 4.1b and Fig. 4.2b) comes from comparing the robustness of $\neg(gear = 4)$ and that of $speed > 35$ point by point and taking the maximum accordingly. We can see that the robustness to $\neg(gear = 4) \vee speed > 35$ is composed of the robustness to $\neg(gear = 4)$ and $speed > 35$; and in both cases the lowest parts come from the robustness to $\neg(gear = 4)$. In conclusion, the problematic situation results from the robust semantics for disjunctive; more precisely, it results from comparing the robustness concerned with *gear* and the robustness concerned with *speed*.

Here it gives rise to the problem: *gear* usually ranges from 1 to 5, while *speed* ranges over about [0, 150]; therefore, it is not fair to compare the robustness values of them directly, as one can always win the comparison. In this example, when we compare the robustness values at the point where both are positive, the one w.r.t. *speed* is likely to be larger, e.g., in the interval [25,30] of *Sample1* or the interval [20,30] of *Sample2*. This is the so-called *scale problem*. Even worse, *gear*, of which the robustness contributes the final robustness, is a discrete variable—tuning an input signal may not lead to a different robustness value. That is the direct reason why the failure in this section takes place.

Few existing work has tackled the problem. In [75], the authors propose a method that explicitly declares input signals and output signals, so it manually introduces a bias when comparing the robustness values of sub-formulas concerned with different signals. This method mitigates the problem in the presence of knowledge about input/output signals; however, it cannot handle the following cases. Firstly, domain expertise is not available. Although it is generally not difficult to classify signals into input and output, sometimes that is not the case, e.g., *gear* can be an input signal in some applications, but in Fig. 1.1, it is an output signal. Secondly, if both signals are output signals, their method becomes not working unless manually assigning a priority, but that does not give any guarantee on the performance.

Our work proposes a novel approach to handling the problem by introducing the Multi-Armed Bandit (MAB) model into the falsification framework. During the falsification to a Boolean connective, we take sub-formulas as bandit machines as shown in Fig. 4.3, and apply MAB algorithms on the high level to govern the hill-climbing optimization processes running on the low level. The proposed approach overcomes the weakness of [75], in that it does not need human intervention but it learns the falsifying priority between sub-formulas automatically.

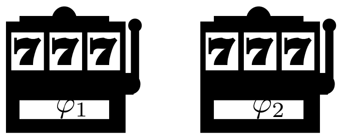


Figure 4.3: Sub-formulas treated as bandit machines

4.2 Multi-Armed Bandit-Based Falsification Algorithm

In this section, we present our contribution, namely a falsification algorithm that addresses the scale problem in Boolean superposition. The main novelties in the algorithm are as follows.

1. **(Use of MAB algorithms)** For binary Boolean connectives, unlike most works in the field, we do not superpose the robustness values of the constituent formulas φ_1 and φ_2 using a fixed operator (such as $\sqcap$ and $\sqcup$ in Def. 8). Instead, we view the situation as an instance of the Multi-Armed Bandit problem (MAB): we use an algorithm for MAB to choose one formula φ_i to focus on (here $i \in \{1, 2\}$); and then we apply hill climbing-guided falsification to the chosen formula φ_i.
2. **(*Hill-climbing gain* as rewards in MAB)** For our integration of MAB and hill-climbing optimization, the technical challenge is find a suitable notion of reward for MAB. We introduce a novel notion that we call *hill-climbing gain*: it formulates the (downward) robustness gain that we would obtain by applying hill-climbing optimization, suitably normalized using the scale of previous robustness values.

Later, in §4.3, we demonstrate that combining those two features gives rise to falsification algorithms that successfully cope with the scale problem in Boolean superposition.

Our algorithms focus on a fragment of STL as target specifications. They are called *(disjunctive and conjunctive) safety properties.* In §4.2.1 we describe this fragment of

STL, and introduce necessary adaptation of the semantics. After reviewing the MAB problem in §4.2.2, we present our algorithms in §4.2.3–4.2.4.

4.2.1 Conjunctive and Disjunctive Safety Properties

Definition 12 (conjunctive/disjunctive safety property) *An STL formula of the form* $\Box_I(\varphi_1 \wedge \varphi_2)$ *is called a* conjunctive safety property*; an STL formula of the form* $\Box_I(\varphi_1 \vee \varphi_2)$ *is called a* disjunctive safety property.

It is known that, in industry practice, a majority of specifications is of the form $\Box_I(\varphi_1 \rightarrow \varphi_2)$, where φ_1 describes a trigger and φ_2 describes a countermeasure that should follow. This property is equivalent to $\Box_I(\neg\varphi_1 \vee \varphi_2)$, and is therefore a disjunctive safety property.

In §4.2.3–4.2.4, we present two falsification algorithms, for conjunctive and disjunctive safety properties respectively. For the reason we just discussed, we expect the disjunctive algorithm should be more important in real-world application scenarios. In fact, the disjunctive algorithm turns out to be more complicated, and it is best introduced as an extension of the conjunctive algorithm.

We define the restriction of robust semantics to a (sub)set of time instants. Note that we do not require $\mathcal{S} \subseteq [0,T]$ to be a single interval.

Definition 13 ($[\![\mathbf{w},\psi]\!]_{\mathcal{S}}$, robustness restricted to $\mathcal{S} \subseteq [0,T]$) *Let* $\mathbf{w}\colon [0,T] \rightarrow \mathbb{R}^{|\mathbf{Var}|}$ *be a signal,* ψ *be an STL formula, and* $\mathcal{S} \subseteq [0,T]$ *be a subset. We define the* robustness *of* $\mathbf{w}$ *under* ψ restricted to $\mathcal{S}$ *by*

$$[\![\mathbf{w},\psi]\!]_{\mathcal{S}} \;:=\; \textstyle\bigsqcap_{t\in\mathcal{S}} [\![\mathbf{w}^t,\psi]\!] \quad . \tag{4.1}$$

Obviously, $[\![\mathbf{w},\psi]\!]_{\mathcal{S}} < 0$ implies that there exists $t \in \mathcal{S}$ such that $[\![\mathbf{w}^t,\psi]\!]_{\mathcal{S}} < 0$. We derive the following easy lemma; it is used later in our algorithm.

Lemma 1 *In the setting of Def. 13, consider a disjunctive safety property* $\varphi \equiv \Box_I(\varphi_1 \vee \varphi_2)$*, and let* $\mathcal{S} := \{t \in I \cap [0,T] \mid [\![\mathbf{w}^t,\varphi_1]\!] < 0\}$*. Then* $[\![\mathbf{w},\varphi_2]\!]_{\mathcal{S}} < 0$ *implies* $[\![\mathbf{w},\Box_I(\varphi_1 \vee \varphi_2)]\!] < 0$. □

4.2.2 The Multi-Armed Bandit (MAB) Problem

The *Multi-Armed Bandit* (MAB) problem describes a situation where,

- a gambler sits in front of a row $A_1, \dots, A_n$ of slot machines;
- each slot machine A_i gives, when its arm is played (i.e. in each attempt), a reward according to a prescribed (but unknown) probability distribution μ_i;
- and the goal is to maximize the cumulative reward after a number of attempts, playing a suitable arm in each attempt.

The best strategy of course is to keep playing the best arm $A_{\max}$, i.e. the one whose average reward $\mathrm{avg}(\mu_{\max})$ is the greatest. This best strategy is infeasible, however, since the distributions $\mu_1, \dots, \mu_n$ are initially unknown. Therefore the gambler must learn about $\mu_1, \dots, \mu_n$ through attempts.

The MAB problem exemplifies the "learning by trying" paradigm of *reinforcement learning*, and is thus heavily studied. The greatest challenge is to balance between *exploration* and *exploitation.* A greedy (i.e. exploitation-only) strategy will play the arm whose empirical average reward is the maximum. However, since the rewards are random, this way the gambler can miss another arm whose real performance is even better but which is yet to be found so. Therefore one needs to mix exploration, too, occasionally trying empirically non-optimal arms, in order to identity their true performance.

The relevance of MAB to our current problem is as follows. Falsifying a conjunctive safety property $\Box_I(\varphi_1 \wedge \varphi_2)$ amounts to finding a time instant $t \in I$ at which either φ_1 or φ_2 is falsified. We can see the two subformulas (φ_1 and φ_2) as two arms, and this constitutes an instance of the MAB problem. In particular, playing an arm translates to a falsification attempt by hill climbing, and collecting rewards translates to spending time to minimize the robustness. We show in §4.2.3–4.2.4 that this basic idea extends to disjunctive safety properties $\Box_I(\varphi_1 \vee \varphi_2)$, too.

A rigorous formulation of the MAB problem is presented for the record.

Definition 14 (the Multi-Armed Bandit problem) *The* Multi-Armed Bandit *(MAB) problem is formulated as follows.*
Input: *arms* $(A_1, \dots, A_n)$*, the associated probability distributions* $\mu_1, \dots, \mu_n$ *over* $\mathbb{R}$*, and a time horizon* $H \in \mathbb{N} \cup \{\infty\}$*.*
Goal: *synthesize a sequence* $A_{i_1} A_{i_2} \dots A_{i_H}$*, so that the cumulative reward* $\sum_{k=1}^{H} \mathrm{rew}_k$ *is*

Algorithm 4.1 The ε-greedy algorithm for multi-armed bandits

```
Require: the setting of Def. 14, and a constant $\varepsilon > 0$ (typically very small)
    At the $k$-th attempt, choose the arm $A_{i_k}$ as follows
1: $j_{\text{emp-opt}} \leftarrow \arg\max_{j \in [1,n]} R(j, k-1)$        ▷ the arm that is empirically optimal
2: Sample $i_k \in [1,n]$ from the distribution
       $\begin{bmatrix} j_{\text{emp-opt}} & \longmapsto & (1-\varepsilon) + \frac{\varepsilon}{n} & \\ j & \longmapsto & \frac{\varepsilon}{n} & \text{for each } j \in [1,n] \setminus \{j_{\text{emp-opt}}\} \end{bmatrix}$
3: Return $i_k$
```

maximized. Here the reward rew_k *of the k-th attempt is sampled from the distribution* μ_{i_k} *associated with the arm* A_{i_k} *played at the k-th attempt.*

We introduce some notations for later use. Let $(A_{i_1} \dots A_{i_k}, \mathsf{rew}_1 \dots \mathsf{rew}_k)$ *be a* history, *i.e. the sequence of arms played so far (here* $i_1, \dots, i_k \in [1,n]$*), and the sequence of rewards obtained by those attempts* (rew_l *is sampled from* μ_{i_l}*).*

For an arm A_j*, its* visit count $N(j, A_{i_1}A_{i_2} \dots A_{i_k}, \mathsf{rew}_1\mathsf{rew}_2 \dots \mathsf{rew}_k)$ *is given by the number of occurrences of* A_j *in* $A_{i_1}A_{i_2} \dots A_{i_k}$*. Its* empirical average reward $R(j, A_{i_1}A_{i_2} \dots A_{i_k}, \mathsf{rew}_1\mathsf{rew}_2 \dots \mathsf{rew}_k)$ *is given by* $\sum_{l \in \{l \in [1,k] \mid i_l = j\}} \mathsf{rew}_l$*, i.e. the average return of the arm* A_j *in the history. When the history is obvious from the context, we simply write* $N(j,k)$ *and* $R(j,k)$*.*

MAB Algorithms

There have been a number of algorithms proposed for the MAB problem; each of them gives a *strategy* (also called a *policy*) that tells which arm to play, based on the previous attempts and their rewards. The focus here is how to resolve the exploration-exploitation trade-off. Here we review two well-known algorithms.

The ε-Greedy algorithm This is a simple algorithm that spares a small fraction ε of chances for empirically non-optimal arms. The spared probability ε is uniformly distributed. See Algorithm 4.1.

The UCB1 algorithm The UCB1 (*upper confidence bound*) algorithm is more complex; it comes with a theoretical upper bound for *regrets*, i.e. the gap between the expected cumulative reward and the optimal (but infeasible) cumulative reward (i.e. the

Algorithm 4.2 The UCB1 algorithm for multi-armed bandits

```
Require: the setting of Def. 14, and a constant $c > 0$
    At the $k$-th attempt, choose the arm $A_{i_k}$ as follows
1:  $i_k \leftarrow \arg\max_{j \in [1,n]} \left( R(j, k-1) + c\sqrt{\frac{2\ln(k-1)}{N(j,k-1)}} \right)$
2:  Return $i_k$
```

result of keep playing the optimal arm $A_{\max}$). It is known that the UCB1 algorithm's regret is at most $O(\sqrt{nH \log H})$ after H attempts, improving the naive random strategy (which has the expected regret $O(H)$).

See Alg. 4.2. The algorithm is deterministic, and picks the arm that maximizes the value shown in Line 1. The first term $R(j, k-1)$ is the *exploitation* factor, reflecting the arm's empirical performance. The second term is the *exploration* factor. Note that it is bigger if the arm A_j has been played less frequently. Note also that the exploration factor eventually decays over time: the denominator grows roughly with $O(k)$, while the numerator grows with $O(\ln k)$.

4.2.3 Our MAB-Guided Algorithm I: Conjunctive Safety Properties

Our first algorithm targets at conjunctive safety properties. It is based on our identification of MAB in a Boolean conjunction in falsification—this is as we discussed just above Def. 14. The technical novelty lies in the way we combine MAB algorithms and hill-climbing optimization; specifically, we introduce the notion of *hill-climbing gain* as a reward notion in MAB (Def. 15). This first algorithm paves the way to the one for disjunctive safety properties, too (§4.2.4).

The algorithm is in Algorithm 4.3. Some remarks are in order.

Algorithm 4.3 aims to falsify a conjunctive safety property $\varphi \equiv \Box_I(\varphi_1 \wedge \varphi_2)$. Its overall structure is to *interleave* two sequences of falsification attempts, both of which are hill climbing-guided. These two sequences of attempts aim to falsify $\Box_I\varphi_1$ and $\Box_I\varphi_2$, respectively. Note that $[\![\mathcal{M}(\mathbf{u}), \varphi]\!] \leq [\![\mathcal{M}(\mathbf{u}), \Box_I\varphi_1]\!]$, therefore falsification of $\Box_I\varphi_1$ implies falsification of φ; the same holds for $\Box_I\varphi_2$, too.

In Line 5 we run an MAB algorithm to decide which of $\Box_I\varphi_1$ and $\Box_I\varphi_2$ to target at

Algorithm 4.3 Our MAB-guided algorithm I: *conjunctive* safety properties

```
Require: a system model $\mathcal{M}$, an STL formula $\varphi \equiv \Box_I(\varphi_1 \wedge \varphi_2)$, and a budget $\mathbf{K}$
 1: function MAB-FALSIFY-CONJ-SAFETY($\mathcal{M}, \varphi, \mathbf{K}$)
 2:     $\mathsf{rb} \leftarrow \infty$ ;   $k \leftarrow 0$
                          ▷ rb is the smallest robustness seen so far, for either $\Box_I\varphi_1$ or $\Box_I\varphi_2$
 3:     while $\mathsf{rb} \geq 0$ and $k \leq \mathbf{K}$ do      ▷ iterate if not yet falsified, and within budget
 4:         $k \leftarrow k + 1$
 5:         $i_k \leftarrow \mathrm{MAB}\Big( (\varphi_1, \varphi_2),\ \big(\mathcal{R}(\varphi_1), \mathcal{R}(\varphi_2)\big),\ \varphi_{i_1} \dots \varphi_{i_{k-1}},\ \mathsf{rew}_1 \dots \mathsf{rew}_{k-1} \Big)$
                          ▷ an MAB choice of $i_k \in \{1, 2\}$ for optimizing the reward $\mathcal{R}(\varphi_{i_k})$
 6:         $\mathbf{u}_k \leftarrow \textsc{Hill-Climb}\Big( \big( (\mathbf{u}_l,\ \mathsf{rb}_l) \big)_{l \in [1, k-1] \text{ such that } i_l = i_k} \Big)$
            ▷ suggestion of the next input $\mathbf{u}_k$ by hill climbing, based on the previous
              observations on the formula $\varphi_{i_k}$ (those on the other formula are ignored)
 7:         $\mathsf{rb}_k \leftarrow [\![ \mathcal{M}(\mathbf{u}_k), \Box_I \varphi_{i_k} ]\!]$
 8:         if $\mathsf{rb}_k < \mathsf{rb}$ then
 9:             $\mathsf{rb} \leftarrow \mathsf{rb}_k$
10:     $\mathbf{u} \leftarrow \begin{cases} \mathbf{u}_k & \text{if } \mathsf{rb} < 0 \\ \text{Failure} & \text{otherwise, that is, no falsifying input found within budget } \mathbf{K} \end{cases}$
11:     Return u
```

Algorithm 4.4 Our MAB-guided algorithm II: *disjunctive* safety properties

```
Require: a system model $\mathcal{M}$, an STL formula $\varphi \equiv \Box_I(\varphi_1 \vee \varphi_2)$, and a budget $\mathbf{K}$
 1: function MAB-FALSIFY-DISJ-SAFETY($\mathcal{M}, \varphi, \mathbf{K}$)
    The same as Algorithm 4.3, except that Line 7 is replaced by the following Line 7'.
    7':  $\mathsf{rb}_k \leftarrow [\![ \mathcal{M}(\mathbf{u}_k), \varphi_{i_k} ]\!]_{\mathcal{S}_k}$   where $\mathcal{S}_k = \big\{ t \in I \cap [0, T] \,\big|\, [\![ \mathcal{M}(\mathbf{u}^t_k), \varphi_{\overline{i_k}} ]\!] < 0 \big\}$
                          ▷ here $\varphi_{\overline{i_k}}$ denotes the other formula than $\varphi_{i_k}$, among $\varphi_1, \varphi_2$
```

in the k-th attempt. The function MAB takes the following as its arguments: 1) the list of arms, given by the formulas φ_1, φ_2; 2) their rewards $\mathcal{R}(\varphi_1), \mathcal{R}(\varphi_2)$; 3) the history $\varphi_{i_1} \dots \varphi_{i_{k-1}}$ of previously played arms ($i_l \in \{1, 2\}$); and 4) the history $\mathsf{rew}_1 \dots \mathsf{rew}_{k-1}$ of previously observed rewards. This way, the type of the MAB function in Line 5 matches the format in Def. 14, and thus the function can be instantiated with any MAB algorithm such as Algorithms 4.1–4.2.

The only missing piece is the definition of the rewards $\mathcal{R}(\varphi_1), \mathcal{R}(\varphi_2)$. We introduce the following notion, tailored for combining MAB and hill climbing.

Definition 15 (hill-climbing gain) *In Algorithm 4.3, in Line 5, the reward* $\mathcal{R}(\varphi_i)$ *of* *the arm φ_i (where $i \in \{1,2\}$) is defined by*

$$\mathcal{R}(\varphi_i) = \begin{cases} \dfrac{\mathsf{max\text{-}rb}(i,k-1) - \mathsf{last\text{-}rb}(i,k-1)}{\mathsf{max\text{-}rb}(i,k-1)} & \textit{if } \varphi_i \textit{ has been played before} \\ 0 & \textit{otherwise} \end{cases}$$

Here $\mathsf{max\text{-}rb}(i,k-1) := \max\{\mathsf{rb}_l \mid l \in [1,k-1], i_l = i\}$ *(i.e. the greatest* rb_l *so far, in those attempts where φ_i was played), and* $\mathsf{last\text{-}rb}(i,k-1) := \mathsf{rb}_{l_{\mathrm{last}}}$ *with l_{last} being the greatest $l \in [1,k-1]$ such that $i_l = i$ (i.e. the last* rb_l *for φ_i).*

Since we try to minimize the robustness values rb_l through falsification attempts, we can expect that rb_l for a fixed arm φ_i decreases over time. (In the case of the hill-climbing algorithm CMA-ES that we use, this is in fact guaranteed). Therefore the value $\mathsf{max\text{-}rb}(i,k-1)$ in the definition of $\mathcal{R}(\varphi_i)$ is the first observed robustness value. The numerator $\mathsf{max\text{-}rb}(i,k-1) - \mathsf{last\text{-}rb}(i,k-1)$ then represents how much robustness we have reduced so far by hill climbing—hence the name "hill-climbing gain." The denominator $\mathsf{max\text{-}rb}(i,k-1)$ is there for normalization.

In Algorithm 4.3, the value rb_k is given by the robustness $[\![\mathcal{M}(\mathbf{u}_k), \Box_I \varphi_{i_k}]\!]$. Therefore the MAB choice in Line 5 essentially picks i_k for which hill climbing yields greater effect (but also taking exploration into account—see §4.2.2).

In Line 6 we conduct hill-climbing optimization—see §2.3. The function Hill-Climb learns from the previous attempts $\mathbf{u}_{l_1}, \dots, \mathbf{u}_{l_m}$ regarding the same formula φ_{i_k}, and their resulting robustness values $\mathsf{rb}_{l_1}, \dots, \mathsf{rb}_{l_m}$. Then it suggests the next input signal $\mathbf{u}_k$ that is likely to minimize the (unknown) function that underlies the correspondences $\big[\, \mathbf{u}_{l_j} \mapsto \mathsf{rb}_{l_j} \,\big]_{j \in [1,m]}$.

Lines 6–8 read as follows: the hill-climbing algorithm suggests a single input $\mathbf{u}_k$, which is then selected or rejected (Line 8) based on the robustness value it yields (Line 7). We note that this is a simplified picture: in our implementation that uses CMA-ES (it is an evolutionary algorithm), we maintain a population of some ten particles, and each of them is moved multiple times (our choice is three times) before the best one is chosen as $\mathbf{u}_k$.

4.2.4 Our MAB-Guided Algorithm II: Disjunctive Safety Properties

The other main algorithm of ours aims to falsify a *disjunctive* safety property $\varphi \equiv \Box_I(\varphi_1 \vee \varphi_2)$. We believe this problem setting is even more important than the conjunctive case, since it encompasses conditional safety properties (i.e. of the form $\Box_I(\varphi_1 \rightarrow \varphi_2)$). See §4.2.1 for discussions.

In the disjunctive setting, the challenge is that falsification of $\Box_I\varphi_i$ (with $i \in \{1, 2\}$) does *not* necessarily imply falsification of $\Box_I(\varphi_1 \vee \varphi_2)$. This is unlike the conjunctive setting. Therefore we need some adaptation of Algorithm 4.3, so that the two interleaved sequences of falsification attempts for φ_1 and φ_2 are not totally independent of each other. Our solution consists of *restricting* time instants to those where φ_2 is false, in a falsification attempt for φ_1 (and vice versa), in the way described in Def. 13.

Algorithm 4.4 shows our MAB-guided algorithm for falsifying a disjunctive safety property $\Box_I(\varphi_1 \vee \varphi_2)$. The only visible difference is that Line 7 in Algorithm 4.3 is replaced with Line 7'. The new Line 7' measures the quality of the suggested input signal $\mathbf{u}_k$ in the way restricted to the region $\mathcal{S}_k$ in which the other formula is already falsified. Lem. 1 guarantees that, if $\mathsf{rb}_k < 0$, then indeed the input signal $\mathbf{u}_k$ falsifies the original specification $\Box_I(\varphi_1 \vee \varphi_2)$.

The assumption that makes Alg. 4.4 sensible is that, although it can be hard to find a time instant at which both φ_1 and φ_2 are false (this is required in falsifying $\Box_I(\varphi_1 \vee \varphi_2)$), falsifying φ_1 (or φ_2) individually is not hard. Without this assumption, the region $\mathcal{S}_k$ in Line 7' would be empty most of the time. Our experiments in §4.3 demonstrate that this assumption is valid in many problem instances, and that Alg. 4.4 is effective.

4.3 Experimental Evaluation

4.3.1 Experiment Setup

We name MAB-UCB and MAB-ϵ-greedy the two versions of MAB algorithm using strategies ϵ-Greedy (see Alg. 4.1) and UCB1 (see Alg. 4.2). We compared the proposed approach (both versions MAB-UCB and MAB-ϵ-greedy) with a state-of-the-art falsification framework, namely Breach [22]. Breach encapsulates several hill-climbing optimization algorithms, including *CMA-ES (covariance matrix adaptation evolution strategy)* [92], *SA (simulated annealing)*, *GNM (global Nelder-Mead)* [91], etc. According to our experience, CMA-ES outperforms other hill-climbing solvers in Breach, so the experiments for both Breach and our approach rely on the CMA-ES solver.

Experiments have been executed using Breach 1.2.13 on an Amazon EC2 c4.large instance, 2.9 GHz Intel Xeon E5-2666, 2 virtual CPU cores, 4 GB RAM.

Benchmarks We selected three benchmark models from the literature, each one having different specifications. The first one is the *Automatic Transmission* (AT) model [44, 98]. It has two input signals, $throttle \in [0, 100]$ and $brake \in [0, 325]$, and computes the car's *speed*, engine rotation in rounds per minute *rpm*, and the automatically selected *gear*. The specifications concern the relation between the three output signals to check whether the car is subject to some unexpected or unsafe behaviors. The second benchmark is the *Abstract Fuel Control* (AFC) model [44, 56]. It takes two input signals, *pedal angle*$\in[8.8, 90]$ and *engine speed*$\in[900, 1100]$, and outputs the critical signal *air-fuel ratio* (*AF*), which influences fuel efficiency and car performance. The value is expected to be close to a reference value $AFref$; $mu \equiv |AF - AFref| / AFref$ is the deviation of AF from $AFref$. The specifications check whether this property holds under both *normal mode* and *power enrichment mode*. The third benchmark is a model of a *magnetic levitation system with a NARMA-L2 neurocontroller* (NN) [44, 99]. It takes one input signal, $Ref \in [1, 3]$, which is the reference for the output signal *Pos*, the position of a magnet suspended above an electromagnet. The specifications say that the position should approach the reference signal in a few seconds when these two are not close.

We built the benchmark set Bbench, as shown in Table 4.1a that reports the name of the model and its specifications (ID and formula). In total, we found 11 specifications. In order to increase the benchmark set and obtain specifications of different complexity, we artificially modified a constant (turned into a parameter named τ if it is contained in a time interval, named ρ otherwise) of the specification: for each specification S, we generated m different versions, named as S_i with $i \in \{1, \ldots, m\}$; the complexity of the specification (in terms of difficulty to falsify it) increases with increasing i.[1] In total, we

[1]Note that we performed this classification based on the falsification results of Breach.

Table 4.1: Benchmark sets Bbench and Sbench

(a) Bbench (here $\delta_{t'}(\mathbf{w})$ represents $\mathbf{w}^t(t') - \mathbf{w}^t(0)$).

Bench	Specification		Parameter
	ID	Formula	
AT	AT1	$\Box_{[0,30]}((gear = 3) \rightarrow (speed > \rho))$	$\rho \in \{20.6, 20.4, 20.2, 20, 19.8\}$
	AT2	$\Box_{[0,30]}((gear = 4) \rightarrow (speed > \rho))$	$\rho \in \{43, 41, 39, 37, 35\}$
	AT3	$\Box_{[0,30]}((gear = 4) \rightarrow (rpm > \rho))$	$\rho \in \{700, 800, 900, 1000, 1100\}$
	AT4	$\Box_{[0,30-\tau]}((\delta_{10}(rpm) > 2000) \rightarrow (\delta_\tau(gear) > 0))$	$\tau \in \{15, 16, 17, 18, 19\}$
	AT5	$\Box_{[0,30]}((speed < \rho) \wedge (RPM < 4780))$	$\rho \in \{130, 131, 132, 133, 134, 135, 136, 137\}$
	AT6	$\Box_{[0,26]}((\delta_4(speed) > \rho) \rightarrow (\delta_4(gear) > 0))$	$\rho \in \{20, 25, 30, 35, 40\}$
	AT7	$\Box_{[0,30-\tau]}((\delta_\tau(speed) > 30) \rightarrow (\delta_\tau(gear) > 0))$	$\tau \in \{2, 3, 4, 5, 6, 7, 8\}$
AFC	AFC1	$\Box_{[11,50]}((controller\_mode = 0) \rightarrow (mu < \rho))$	$\rho \in \{0.16, 0.17, 0.18, 0.19, 0.2\}$
	AFC2	$\Box_{[11,50]}((controller\_mode = 1) \rightarrow (mu < \rho))$	$\rho \in \{0.222, 0.224, 0.226, 0.228, 0.23\}$
NN		$close \equiv \|Pos - Ref\| <= \rho + \alpha * \|Ref\|$	
		$reach \equiv \Diamond_{[0,2]}(\Box_{[0,1]}(close))$	
	NN1	$\Box_{[0,18]}(\neg close \rightarrow reach)$, $\alpha = 0.04$	$\rho \in \{0.001, 0.002, 0.003, 0.004, 0.005\}$
	NN1	$\Box_{[0,18]}(\neg close \rightarrow reach)$, $\alpha = 0.03$	$\rho \in \{0.001, 0.002, 0.003, 0.004, 0.005\}$

(b) Sbench

Spec ID	scaled output	factor 10^k
$AT1_1$		
$AT1_2$		
$AT1_3$	*speed*	$k \in$\{-2,0,1,3\}
$AT1_4$		
$AT1_5$		
$AT5_4$		
$AT5_5$		
$AT5_6$	*speed*	$k \in$\{-2,0,1,3\}
$AT5_7$		
$AT5_8$		
$AFC1_1$		
$AFC1_2$		
$AFC1_3$	*mu*	$k \in$\{0,1,2,3\}
$AFC1_4$		
$AFC1_5$		

produced 60 specifications. Column *parameter* in the table shows which concrete values we used for the parameters ρ and τ. Note that all the specifications but one are disjunctive safety properties (i.e., $\Box_I(\varphi_1 \vee \varphi_2)$), as they are the most difficult case and they are the main target of our approach; we just add AT5 as example of conjunctive safety property (i.e., $\Box_I(\varphi_1 \wedge \varphi_2)$).

Our approach has been proposed with the aim of tackling the scale problem. Therefore, to better show how our approach mitigates this problem, we generated a second benchmark set Sbench as follows. We selected 15 specifications from Bbench (with concrete values for the parameters) and, for each specification S, we changed the corresponding Simulink model by multiplying one of its outputs by a factor 10^k, with $k \in \{-2, 0, 1, 2, 3\}$ (note that we also include the original one using scale factor 10^0); the specification has been modified accordingly, by multiplying with the scale factor the constants that are compared with the scaled output. This makes sense, because we may use a different measurement in real life, such as m/s instead of km/h for *speed.* We implement this by adding a product block to the Simulink model. Fig. 4.4 shows an example, where *speed* is amplified by 10. We name a specification S scaled with factor 10^k as S^k. Table 4.1b reports the IDs of the original specifications, the output that has been scaled, and the used scaled factors; in total, the benchmark set Sbench contains 60 specifications .

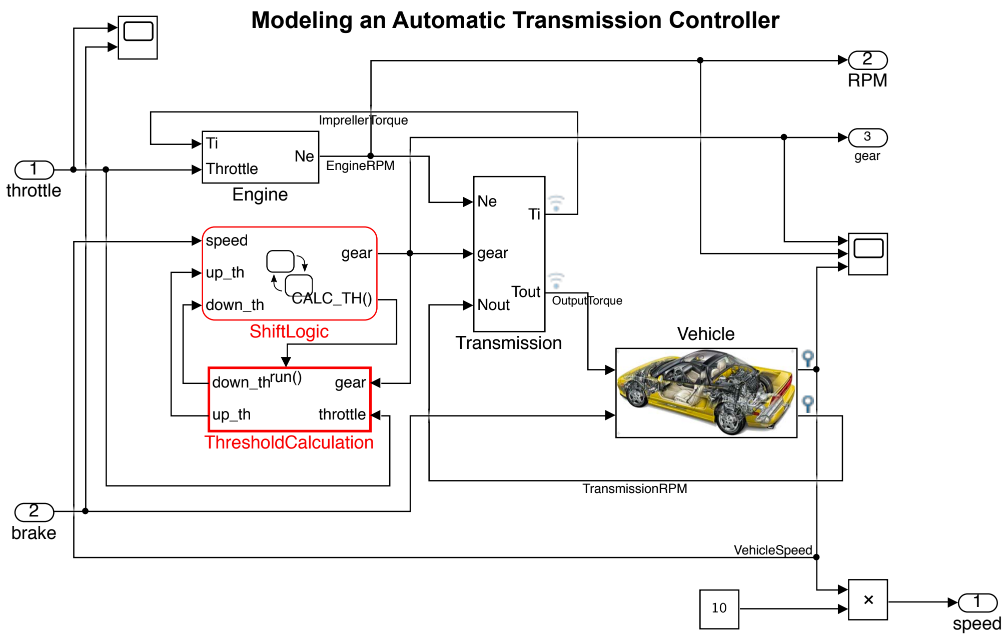


Figure 4.4: Automatic transmission model with *speed* amplified by 10

Experiment In our context, an *experiment* consists in the execution of an approach A (either Breach, MAB-ϵ-greedy, or MAB-UCB) over a specification S for 30 *trials*, using different initial seeds. For each experiment, we record the *success* SR as the number of trials in which a falsifying input was found, and average execution *time* of the trials. Complete experimental results are reported in Appendix §A.2[1]. We report aggregated results in Table 4.2.

For benchmark set Bbench, it reports aggregated results for each group of specifications obtained from S (i.e., all the different versions S_i obtained by changing the value of the parameter); for benchmark set Sbench, instead, results are aggregated for each scaled specification S^k (considering the versions S_i^k obtained by changing the parameter value). We report minimum, maximum and average number of successes SR, and time in seconds. For MAB-ϵ-greedy and MAB-UCB, both for SR and time, we also report the average percentage difference[2] (Δ) w.r.t. to the corresponding value of Breach.

[1]The code, models, and specifications are available online at https://github.com/ERATOMMSD/FalStar-MAB.

[2]$\Delta = ((m-b)*100)/(0.5*(m+b))$ where m is the result of MAB and b the one of Breach.

Table 4.2: Aggregated results for benchmark sets Bbench and Sbench (SR: # successes out 30 trials. Time in secs. Δ: percentage difference w.r.t. Breach). Outperformance cases are highlighted, indicated by positive Δ of SR, and negative Δ of time.

Spec.	Breach						MAB-ϵ-greedy								MAB-UCB							
ID	SR (/30)			time (sec.)			SR (/30)				time (sec.)				SR (/30)				time (sec.)			
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Δ	Min	Max	Avg	Δ	Min	Max	Avg	Δ	Min	Max	Avg	Δ
AT1	14	25	20.2	125	361.2	223.1	24	30	28.6	35.7	62.7	213.4	106.4	−73.4	28	30	29.2	37.8	45.1	146.8	77.4	−97.1
AT2	11	30	20.2	14	390.6	209.8	30	30	30	43.9	11.9	126.3	54.5	−96.9	27	30	29.4	42.2	17.7	92.5	36.8	−112.1
AT3	29	30	29.4	2.3	22.2	14.2	30	30	30	2	2.5	7	3.5	−82.9	30	30	30	2	2.5	3.6	3	−88.6
AT4	18	30	25.8	19.5	265.3	109.6	29	30	29.8	16	7.8	45.1	24.4	−105	30	30	30	16.6	6.2	36.2	22.2	−113.5
AT5	6	23	14.1	203.1	525.9	366.2	26	30	28.5	72.1	35.2	149	93.7	−120.6	26	30	28.2	71.4	37.7	154.1	99.2	−116.8
AT6	5	29	22.8	30.1	509.5	157	21	30	27	28	2.3	300	95.1	−98.3	22	30	27	27.7	2.9	247.3	86.1	−99.4
AT7	15	30	26.6	12.2	314	81.5	20	30	28.6	8.4	2.9	283.9	49.9	−92	23	30	29	10.3	5.5	223.3	42.9	−88.3
AFC1	6	30	14.4	124.8	565.6	413.5	4	28	12	−28.4	171	568.4	446	10.8	5	30	16.4	9.7	98.7	559.8	389.9	−9.3
AFC2	2	30	18	80.7	582.3	343.4	5	30	20	23.8	43.2	547.8	301.9	−23.8	5	30	20	22.9	59.4	568.4	320.5	−11.1
NN1	17	25	20.8	212.9	384.7	292.9	14	27	20.2	−4.5	189.5	422.8	320.3	6.2	17	28	22.6	7.3	148.2	403.3	272.3	−11.8
NN2	27	28	27.2	55.5	93.4	73.1	30	30	30	9.8	11	39.3	26.3	−97.8	30	30	30	9.8	14.6	38.2	27.4	−92.3
$AT1^{-2}$	30	30	30	42.5	97.4	56.9	28	30	29	−3.4	75.6	178.3	118.7	68.7	28	30	29.4	−2.1	54.3	136.3	80.3	33.3
$AT1^{0}$	14	25	20.2	125	361.2	223.1	24	30	28.6	35.7	62.7	213.4	106.4	−73.4	28	30	29.2	37.8	45.1	146.8	77.4	−97.1
$AT1^{1}$	4	21	15.4	204.5	527.6	310.2	25	30	29	68.4	49	234.7	102.1	−108	27	29	28.2	64.5	77.5	128.7	105.1	−93
$AT1^{3}$	8	24	19.8	164	471.7	240.1	29	30	29.8	44.6	67.5	170.6	101.9	−77.3	29	30	29.4	43.4	55.4	104.8	80.6	−93.6
$AT5^{-2}$	29	30	29.6	61.1	163.7	102	25	30	27.8	−6.4	76.9	139.5	111.9	12.6	28	30	29.4	−0.7	48.5	131.9	85.7	−17
$AT5^{0}$	6	18	11.2	291.1	525.9	423.1	28	30	28.4	90.5	80.2	151.3	107.4	−117.7	26	30	28	89.4	68.3	154.1	114.9	−114.5
$AT5^{1}$	0	2	0.4	566.4	600	593.3	27	30	28.4	194.8	70.7	184.5	110.3	−138.5	25	30	27.6	194.1	83.1	150	123.7	−131.2
$AT5^{3}$	0	1	0.2	586.4	600	597.3	27	30	28.6	197.2	66.8	163.3	102.5	−142.3	27	29	28	197.2	80.4	160.9	111.9	−137.4
$AFC1^{0}$	6	30	14.4	124.8	565.6	413.5	4	29	16.4	8.5	115.1	559.9	411.1	−2.8	5	30	16.4	9.7	98.7	559.8	389.9	−9.3
$AFC1^{1}$	7	30	16.6	99	548.2	393.3	3	29	10.8	−60.9	198.1	587.6	465.8	24.6	7	29	17.8	10.3	105.7	527.3	354.3	−10.3
$AFC1^{2}$	0	12	5.2	434.4	600	535.8	3	28	11.6	96.2	180.8	577.6	463	−20.7	4	30	17	127	73.7	556.3	374.5	−47.3
$AFC1^{3}$	1	12	4.8	425.7	587.4	532.6	3	30	14.4	109	138	585.5	436.5	−28	7	30	15	113	77.1	553.4	403.7	−39.9

Comparison In the following, we compare two approaches $A_1, A_2 \in$ {Breach, MAB-ϵ-greedy, MAB-UCB } by comparing the number of their successes SR and average execution *time* using the non-parametric Wilcoxon signed-rank test with 5% level of significance[1] [100]; the null hypothesis is that there is no difference in applying A_1 A_2 in terms of the compared measure (SR or time).

4.3.2 Evaluation

We evaluate the proposed approach with some research questions.

RQ1 *Which is the best MAB algorithm for our purpose?*

In §4.2.2, we described that the proposed approach can be executed using two different strategies for choosing the arm in the MAB problem, namely MAB-ϵ-

[1]We checked that the distributions are not normal with the non-parametric Shapiro-Wilk test.

Table 4.3: Experimental results – Sbench (SR: # successes out of 30 trials. Time in secs)

Spec.	Breach		MAB-UCB		Spec.	Breach		MAB-UCB		Spec.	Breach		MAB-UCB	
ID	SR	time	SR	time	ID	SR	time	SR	time	ID	SR	time	SR	time
	(/30)	(sec.)	(/30)	(sec.)		(/30)	(sec.)	(/30)	(sec.)		(/30)	(sec.)	(/30)	(sec.)
$AT1_1^{-2}$	30	51.3	30	54.3	$AT5_4^{-2}$	30	61.1	30	48.5	$AFC1_1^0$	30	124.8	30	98.7
$AT1_1^0$	25	125	29	75	$AT5_4^0$	18	291.1	28	94.5	$AFC1_1^1$	30	99	29	105.7
$AT1_1^1$	20	221.1	28	107.9	$AT5_4^1$	2	566.4	25	150	$AFC1_1^2$	12	434.4	30	73.7
$AT1_1^3$	23	170	29	55.4	$AT5_4^3$	1	586.4	28	96.2	$AFC1_1^3$	12	425.7	30	77.1
$AT1_2^{-2}$	30	49	29	67.5	$AT5_5^{-2}$	30	71.3	29	67.8	$AFC1_2^0$	16	421.5	23	346.8
$AT1_2^0$	22	187.5	30	45.1	$AT5_5^0$	15	369.1	27	114	$AFC1_2^1$	25	345.9	27	227.9
$AT1_2^1$	21	204.5	29	77.5	$AT5_5^1$	0	600	29	83.1	$AFC1_2^2$	8	497.2	25	320.5
$AT1_2^3$	24	164	30	61	$AT5_5^3$	0	600	27	113.8	$AFC1_2^3$	5	518.1	21	364
$AT1_3^{-2}$	30	42.5	30	62.4	$AT5_6^{-2}$	29	110.2	28	103.3	$AFC1_3^0$	11	457.7	15	442
$AT1_3^0$	19	239.5	29	62.5	$AT5_6^0$	10	438.2	30	68.3	$AFC1_3^1$	13	479.2	14	455.5
$AT1_3^1$	16	296.2	27	128.7	$AT5_6^1$	0	600	27	126.7	$AFC1_3^2$	2	590.7	15	453.2
$AT1_3^3$	21	209.8	30	93.4	$AT5_6^3$	0	600	29	80.4	$AFC1_3^3$	5	545.6	8	510.6
$AT1_4^{-2}$	30	44.5	30	80.8	$AT5_7^{-2}$	30	103.6	30	77.3	$AFC1_4^0$	9	498.2	9	502.1
$AT1_4^0$	21	202.2	30	57.4	$AT5_7^0$	7	491.4	26	154.1	$AFC1_4^1$	8	494	12	455
$AT1_4^1$	16	301.7	28	119.5	$AT5_7^1$	0	600	27	134.3	$AFC1_4^2$	4	556.8	11	468.7
$AT1_4^3$	23	185.1	29	88.3	$AT5_7^3$	0	600	29	108	$AFC1_4^3$	1	587.4	9	513.4
$AT1_5^{-2}$	30	97.4	28	136.3	$AT5_8^{-2}$	29	163.7	30	131.9	$AFC1_5^0$	6	565.6	5	559.8
$AT1_5^0$	14	361.2	28	146.8	$AT5_8^0$	6	525.9	29	143.6	$AFC1_5^1$	7	548.2	7	527.3
$AT1_5^1$	4	527.6	29	91.9	$AT5_8^1$	0	600	30	124.2	$AFC1_5^2$	0	600	4	556.3
$AT1_5^3$	8	471.7	29	104.8	$AT5_8^3$	0	600	27	160.9	$AFC1_5^3$	1	586	7	553.4

greedy and MAB-UCB. We here assess which one is better in terms of SR and time. From the results in Table 4.2, it seems that MAB-UCB provides slightly better performance in terms of SR; this has been confirmed by the Wilcoxon test applied over all the experiments (i.e., on the non-aggregated data reported in Appendix, §A.2): the null hypothesis that using anyone of the two strategies has no impact on SR is rejected with p-value equal to 0.005089, and the alternative hypothesis that SR is better is accepted with p-value=0.9975; in a similar way, the null hypothesis that there is no difference in terms of time is rejected with p-value equal to 3.495e-06, and the alternative hypothesis that is MAB-UCB is faster is accepted with p-value=1. Therefore, in the following RQs, we compare Breach with only the MAB-UCB version of our approach.

RQ2 *Does the proposed approach effectively solve the scale problem?*

We here assess if our approach is effective in tackling the scale problem. Table 4.3 reports the complete experimental results over Sbench for Breach and MAB-UCB; for each specification S, all its scaled versions are reported in increasing order of the scaling factor. We observe that changing the scaling factor affects (sometimes greatly) the number of successes SR of Breach; for example, for $AT5_5$ and $AT5_7$ it goes from 30 to 0. For MAB-UCB, instead, SR is similar across the scaled versions of each specification: this shows that the approach is robust w.r.t. to the scale problem as the "hill-climbing gain" reward in Def. 15 eliminates the impact of scaling and UCB1 algorithm balances the exploration and exploitation of two sub-formulas. The observation is confirmed by the Wilcoxon test over SR: the null hypothesis is rejected with p-value=1.808e-09, and the alternative hypothesis accepted with p-value=1. Instead, the null hypothesis that there is no difference in terms of time cannot be rejected with p-value=0.3294.

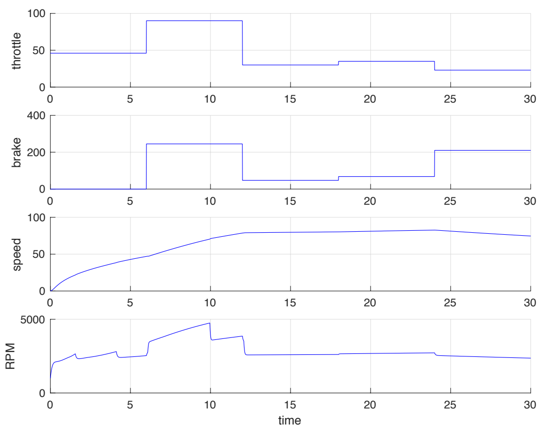
RQ3 *How does the proposed process behave with not scaled benchmarks?*

In RQ2, we checked whether the proposed approach is able to tackle the scale problem for which it has been designed. Here, instead, we are interested in investigating how it behaves on specifications that have not been artificially scaled (i.e., those in Bbench). From Table 4.2 (upper part), we observe that MAB-UCB is always better than Breach both in terms of SR and time, which is shown by the highlighted cases. This is confirmed by Wilcoxon test over SR and time: null hypotheses are rejected with p-values equal to, respectively, 6.02e-08 and 1.41e-08, and the alternative hypotheses that MAB-UCB is better are both accepted with p-value=1. This means that the proposed approach can also handle specifications that do not suffer from the scale problem, and so it can be used with any kind of specification.

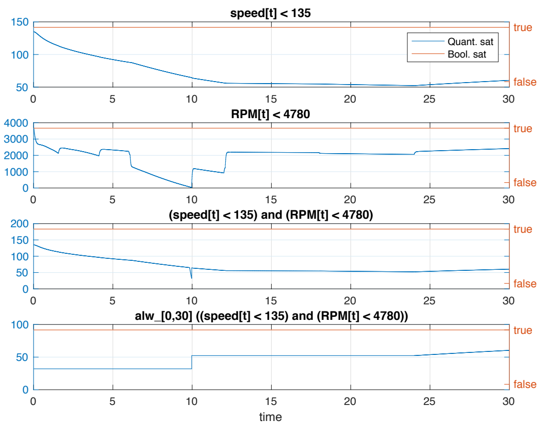
4.3.3 A Comparison to a Normalization-Based Approach

A naïve solution to the scale problem could be to rescale the signals used in specification at the same scale. This is a normalization approach. For example, instead of falsifying $(\neg(\mathit{gear} = 4) \vee \mathit{speed} > 35)$, we can falsify $(\neg(\mathit{gear} = 4) \vee (\gamma \cdot \mathit{speed} > \gamma \cdot 35))$, where γ is a *rescaling factor* that normalizes the robustness value of $\mathit{speed} > 35$ to the same scale as that of $\mathit{gear} = 4$.

The Sbench already gives an implementation of the approach with manual selections of rescaling factors. We thus can compare the performance of our approach to this possible baseline. In some cases, this baseline approach performs quite well. For example, the performance of $AT1^{-2}$ in Table 4.2 are the cases where *speed* is rescaled by 0.01. In these cases, the falsification performance in terms of SR is quite good



(a) Input signals *throttle*, *brake* and output signals *speed*, *RPM* of *a sample* during falsifying $\mathrm{AT5}_6^0$



(b) Boolean satisfaction and quantitative robustness of the sample in Fig. 4.5a to sub-formulas of $\mathrm{AT5}_6^0$

Figure 4.5: A sample from hill-climbing optimization during falsification to $\mathrm{AT5}_6^0 \equiv \Box(\mathit{speed} < 135 \wedge \mathit{rpm} < 4780)$

(SR being 30/30), compared to the cases with other rescaling factors. However, the baseline approach does not always work well. For example, the specifications of AT5 give restriction to *speed* and *rpm* together, and these properties suffer from the scale problem as *speed* is one order of magnitude less than *rpm*. However, from Table 4.2, we observe that the baseline approach (i.e., running Breach over $\mathrm{AT5}^1$) is not effective, as SR is 0.4/30, that is much lower than the original SR 14.1/30 of the unscaled approach using Breach. Our approach, instead, raises SR to 28.4/30 and to 27.6/30 using the two proposed versions. By monitoring Breach execution, we notice that the baseline approach fails because it tries to falsify $\mathit{rpm} < 4780$, which, however, is not falsifiable; our approach, instead, understands that it should try to falsify $\mathit{speed}<\rho$ thanks to the application of MAB algorithms.

Here, it gives rise to the problem: how to select the rescaling factor. The quick answer is by comparing the scales of different signals. However, the example of AT5 proves that this method does not work. Instead, the experimental results in Table 4.2 show that $10^{-2}(0.01)$ is the best choice. Let us take a further look into the example. Fig. 4.5 presents a sample during the process falsifying $\mathrm{AT5}_6^0$. We can see that the final robustness comes from the robustness to the sub-formula $\mathit{rpm} < 4780$. This is opposite to our intuition, since robust semantics for conjunctive (see Def. 8) selects the minimum one between sub-formulas, but *rpm* is the one of larger magnitude. The explanation is as follows: although *rpm* has a larger scale, it is less variant than *speed*; in other words, $[\![\mathcal{M}(\mathbf{u}), \mathit{rpm} < 4780]\!]$ is more likely to be smaller than $[\![\mathcal{M}(\mathbf{u}), \mathit{speed} < 135]\!]$, because it is not hard to drive *rpm* to a high value. Therefore, the larger rescaling factor to *speed* we select, the more probably that *rpm* takes the final robustness. That is why it performs the best when the rescaling factor is $10^{-2}(0.01)$.

5 Constraining Counterexamples via Search Space Transformation

In this chapter, we consider the falsification problem in the presence of logical constraints on input signals. Typical hill-climbing optimization algorithms rely on random samplings, therefore, introduction of input constraints increases the occurrence of infeasible samplings and thus makes the search more difficult.

We firstly show two naive penalty-based approaches that are, though able to solve the problem, not very effective. We then present the main contribution of this chapter, that is, a framework based on search space transformation. It consists of a *space mapping* that maps points in an unconstrained search space to the constrained input space, and a fitness definition that assigns fitness values to points in the search space according to the robustness values of points in the input space. In this way, it allows the search performing in an unconstrained space, and when a negative fitness is detected, it returns the mapped point in the input space as a falsifying input. An

The material in this chapter is based on [82] and [90]

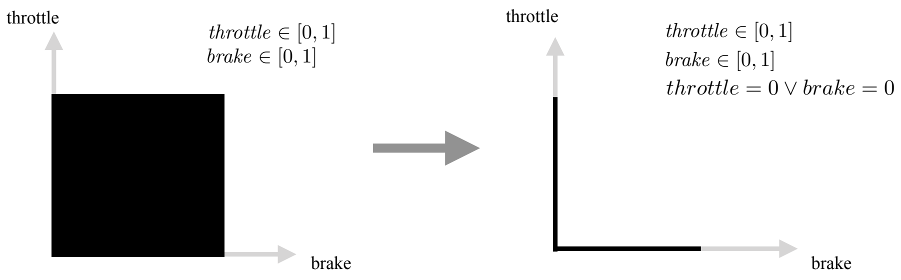


Figure 5.1: Feasible areas without/with considering input constraints

instance of space mapping, named *proportional transformation*, is then defined. We propose three approaches that make use of the proportional transformation, and we experimentally show that the one based on Multi-Armed Bandit (MAB) model performs better than others.

5.1 Motivation and Problem Definition

The problem setting of falsification introduced in Def. 10 does not take into consideration possible constraints over the input signals. Therefore, the search takes place in an unconstrained space, and it can return any input signal that violates the system specification in that space provided that it manages to find one. However, in the presence of constraints, the feasible search space is much more restricted. For example, in an automotive system, like the one in Fig. 1.1, the *throttle* and *brake* cannot be pushed simultaneously. If we ignore this constraint, then hill climbing can search freely in the black rectangle in the left sub-figure of Fig. 5.1; however, if the constraint is considered, the search space shrinks to the black area in the right sub-figure. Actually, typical hill-climbing optimization algorithms only support searching in a rectangle as the case of the left sub-figure, so they do not work for the case of the right one. As a consequence, the falsifying inputs they return are meaningless, e.g., a case in which pushing *throttle* and *brake* simultaneously. Apparently, they are not helpful for engineers to debug the system.

Indeed, some works [56, 76] report such input constraints in CPS. In [56], the authors aim to test a *powertrain control system*, under the condition that *throttle* increases or decreases monotonically; in that case, the values of input signals should be dependent on their prefixes. In [76], the authors test an *assisted driving system* with different initial conditions as the system inputs. There is an constraint about the environment and the system parameter—"when there is no fog, the visibility range is set to maximum". The authors should handle this constraint; otherwise the test does not help the practical use.

In §5.1.1, we formally define the constrained falsification problem, in comparison the the one in Def. 10. Particularly, we introduce two ways to formalize the constraints. They will be used in different technical contexts later, but they are equivalent in expressivity given the condition that input signals are piecewise constant.

5.1.1 Problem Definition

The input constrained falsification problem considered in this chapter is defined as follows.

Definition 16 (Input constrained falsification problem) *The input constrained falsification problem can be stated as the following constrained optimization problem, where ψ are input constraints over the input signals* $\mathbf{u}$.

$$\begin{array}{rl} \underset{\mathbf{u}}{\textit{minimize}} & [\![\mathcal{M}(\mathbf{u}), \varphi]\!] \\ \textit{subject to} & \mathbf{u} \models \psi \\ & \mathbf{u} \in \Omega \end{array}$$

In this chapter, we use two formalisms of constraints ψ. This first one used in §5.2 is simple. We just use STL to express constraints, because ψ is a temporal property that the input signal $\mathbf{u}$ is supposed to satisfy. See the syntax of STL in Def. 7. The second one is introduced in Def. 19, given in the form of propositional logic formula. We use that form for the search space transformation approach introduced in §5.3, §5.4 and §5.5. Note that these two formalisms have equivalent expressivity provided that we use piecewise constant as input signals, as we show in §5.6.1, where we explain how we can transform one into the other one.

5.2 Penalty-Based Approaches

In this section, we introduce two penalty-based approaches. A simple approach based on the modification of the specification under study is presented in §5.2.1, while a more complicated approach based on the lexicographic method are proposed in §5.2.2.

5.2.1 Constraint Embedding Approach

A straightforward penalty-based approach to the constrained falsification problem consists in embedding the input constraints ψ as a prerequisite of the system specification φ. In this way, we obtain the STL formula $\psi \rightarrow \varphi$ as a new falsification goal.

The constrained problem of Def. 16 can be stated as the following unconstrained problem.

$$\begin{aligned} &\underset{\mathbf{u}}{\textit{minimize}} && [\![\langle \mathbf{u}, \mathcal{M}(\mathbf{u})\rangle, \psi \rightarrow \varphi]\!] \\ &\textit{subject to} && \mathbf{u} \in \Omega \end{aligned}$$

The falsification approach must now evaluate the robustness of a formula that predicates both over the input and output signals, formally denoted as $\langle \mathbf{u}, \mathcal{M}(\mathbf{u})\rangle$.

The soundness of the approach is given by Thm. 1.

Theorem 1 (Soundness & completeness of the Constraint Embedding Approach) *For all input signals* $\mathbf{u}$, $[\![\langle \mathbf{u}, \mathcal{M}(\mathbf{u})\rangle, \psi \rightarrow \varphi]\!] < 0$ *if and only if the input constraints* ψ *are satisfied and the specification* φ *is falsified.*

The proof directly comes from the robustness definition of STL and the semantics of the implication.

5.2.2 Lexicographic Method Approach

While the constraint embedding approach can be effective in some cases, it does not dictate a search algorithm to first satisfy input constraints ψ and then falsify the specification φ. We here propose a method that imposes a strict prioritization between the satisfaction of the input constraints and the optimization of the objective function for falsification. This method is based on the use of a *lexicographic method* [101] for defining the fitness function of the optimization problem.

A lexicographic method [101] can be applied for a multi-objective optimization problem that aims at minimizing objective functions $f_1, \ldots, f_N$, and for which there exists a preference order in the optimization of the objective functions, i.e., functions with higher priorities must be optimized first. Formally, there exists a total order of priorities $p_1, \ldots, p_N$, where $p_k = N - k$ for each $k \in \{1, \ldots, N\}$; the larger p_k is, the higher priority f_k has.

$$\begin{aligned} &\underset{\mathbf{x}}{\textit{minimize}} && f_1(\mathbf{x}), \ldots, f_N(\mathbf{x}) \\ &\textit{subject to} && \mathbf{x} \in \Omega \end{aligned}$$

The method defines a global cost function *GCF* in the following way:

$$GCF(\mathbf{x}) = \sum_{k=1}^{N} B^{p_k} \lceil (B-1)\mathcal{T}_k\big(f_k(\mathbf{x})\big) \rceil \tag{5.1}$$

where $B \in \mathbb{R}_+$ with $B > 1$ is a base number, $\lceil\rceil$ is the regular ceiling operator, and each $\mathcal{T}_k$ is a transformation function. Note that $\lceil (B-1)\mathcal{T}_k\big(f_k(\mathbf{x})\big) \rceil$ is needed to map the transformed value of the objective function f_k in B quantization levels. Such a quantization is required by the lexicographic method to maintain the total order of the inputs [102] w.r.t. the priorities of the objective functions, i.e., the fitness value of a unachieved function with higher priority always dominates the fitness values of functions of lower priority. Note that the value of B can have an effect on the efficiency of the search [102], as also noted during the application of the lexicographic methods in other contexts [103]. In the experiments, we will evaluate such effect using different values for B.

The definition of a $\mathcal{T}_k$ is specific to the type of optimization problem; for example, we will see later how to define it for the constraint satisfaction problem and the falsification problem. In any case, the definition of a $\mathcal{T}_k$ must at least satisfy the monotonicity property, i.e., given two values $v_1 \leq v_2$, then $\mathcal{T}_k(v_1) \leq \mathcal{T}_k(v_2)$. Usually, a transformation function $\mathcal{T}_k$ is implemented as a normalization function between [0,1]: in such a case, the values of f_k that are mapped to 0 are those that *achieve* the objective.[1]

[1]Note that, in general, it is not always possible to specify when an objective function is “achieved”. However, the lexicographic methods require that for functions $f_1, \ldots, f_{N-1}$, this is possible, and this is

We apply the lexicographic method to the constrained falsification problem introduced in Def. 16. To do this, we first turn the constrained falsification problem in a unconstrained multi-objective problem as follows.

$$\underset{\mathbf{u}}{\textit{minimize}} \quad [\![\mathbf{u}, \neg\psi]\!] \tag{5.2}$$

$$\underset{\mathbf{u}}{\textit{minimize}} \quad [\![\mathcal{M}(\mathbf{u}), \varphi]\!] \tag{5.3}$$

$$\textit{subject to} \quad \mathbf{u} \in \Omega$$

The constraint satisfaction problem has been turned into an optimization problem by exploiting the robust semantics of STL (recall that also the input constraints are expressed in STL). Since in a lexicographic method all objective functions must be minimized (see Eq. 5.2.2), we consider the negation of the input constraints (negative robustness of $\neg\psi$ corresponds to positive robustness of ψ).

We can now combine the two objectives (Eq. 5.2 and Eq. 5.3) into a single global cost function, following Eq. 5.1. Since we want to prioritize the satisfaction of the input constraints, we take $[\![\mathbf{u}, \neg\psi]\!]$ as f_1, and $[\![\mathcal{M}(\mathbf{u}), \varphi]\!]$ as f_2. The definition of the global cost function is as follows.

Definition 17 (Lexicographic fitness function GCF_{fal} **for falsification)** *Let* $f_1(\mathbf{u}) :=$ $[\![\mathbf{u}, \neg\psi]\!]$, *and* $f_2(\mathbf{u}) := [\![\mathcal{M}(\mathbf{u}), \varphi]\!]$. *The definition of the global cost function for the constrained falsification problem is as follows:*

$$GCF_{\mathsf{fal}}(\mathbf{u}) = B\lceil (B-1)\mathcal{T}_1(f_1(\mathbf{u}))\rceil + (B-1)\mathcal{T}_2(f_2(\mathbf{u}))$$

As explained before, the definition of a transformation function $\mathcal{T}_k$ is specific to the kind of optimization problem. In our context, the transformation function $\mathcal{T}_1$ considers values r given by the robustness evaluation of the input constraints: for any negative value of the robustness, the input constraints are satisfied, while positive values indicate the degree of violation of the input constraints ψ. Therefore, $\mathcal{T}_1$ is

applicable in our context.

defined as a normalization function as follows:

$$\mathcal{T}_1(r) = \begin{cases} 0 & r < 0 \\ \dfrac{r}{R^{\psi}_{max}} & \text{otherwise} \end{cases} \tag{5.4}$$

where R^{ψ}_{max} is the possible maximum value of r. The identification of a correct R^{ψ}_{max} requires minimum effort by sampling the input space. We will present how we come up with R^{ψ}_{max} later in §5.6.

The transformation function $\mathcal{T}_2$, instead, considers values r given by the robustness evaluation of the specification φ. Also in this case, negative values of the robustness mean that the objective is achieved (i.e., the specification is falsified). Therefore, the definition of the transformation function for $\mathcal{T}_2$ is as follows:

$$\mathcal{T}_2(r) = \begin{cases} 0 & r < 0 \\ \epsilon & r = 0 \\ \dfrac{r}{R^{\varphi}_{max}} & \text{otherwise} \end{cases} \tag{5.5}$$

where R^{φ}_{max} is the possible maximum value of r, and ϵ is an arbitarily small positive number[1]. We will also explain later in §5.6 how we select a proper R^{φ}_{max}.

Considering the definitions of the two transformation functions, we can now analyse the behaviour of function GCF_{fal} (see Def. 17). Given an input signal $\mathbf{u}$, if the input constraints ψ are satisfied, the first operand of the sum will be 0 (due to the transformation function $\mathcal{T}_1$ in Eq. 5.4), and therefore the value of GCF_{fal} will only depend on the robustness value of the temporal specification (i.e., the second operand). On the other hand, if the input constraints are not satisfied, the first operand will be positive and guaranteed to be larger than the second one (so driving the search towards the satisfaction of the input constraints).

Note that in the definition of GCF_{fal}, we do not apply the ceiling operator to the robustness evaluation of the specification φ (i.e., f_2). It is indeed known that the ceiling operator is not really needed by the lexicographic method for the last operand of the sum [101, 103], and we take advantage of this. Therefore, since f_2 corresponds to the

[1]Note that this is needed to distinguish inputs having robustness 0 (not falsifying) from those having negative robustness (falsifying).

falsification algorithm, we prefer to remove the ceiling in order to preserve as much information as possible regarding the specification robustness that could be helpful for driving the search. Indeed, removing the ceiling avoids the quantization effect that in general is adversarial for the hill-climbing search.

Theorem 2 (Soundness of the GCF_{fal} fitness function) *If there exists an input signal* $\mathbf{u}$ *such that* $GCF_{\mathsf{fal}}(\mathbf{u}) = 0$*, then the input constraints* ψ *are satisfied and the specification* φ *is falsified.*

The proof directly comes from the definitions of GCF_{fal}, $\mathcal{T}_1$, and $\mathcal{T}_2$, and the robustness definition of STL.

5.2.3 Discussion: Weaknesses of the Penalty-Based Approaches

The approaches in §5.2 exemplify the same idea—adding a penalty to the objective function and searching in a unconstrained space. Moreover, both approaches implement the penalty as a quantitative guidance for input signals to satisfying the constraints. However, these approaches are not sufficiently effective, as proved by the experimental results in §5.6.

The reason is mainly from the following two folds: firstly, the optimization algorithm spends quite much time on searching in the infeasible area, especially if the constraint is non-trivial, or even very hard, to satisfy, like the case in Fig. 5.1; secondly, the penalty added to the objective function changes the fitness domain, which makes the search much more difficult. The penalty can introduce local optimum, flat fitness, etc, to the objective function, in which cases hill-climbing optimization algorithms do not work.

5.3 Problem Setting and Overview of the Search Space Transformation-Based Approach

From this section on, we present the main contribution of this chapter, namely, a framework solving the input constrained falsification problem via search space transformation. This framework consists of a search space transformation that maps a point in an unconstrained search space to the constrained feasible space (which will be introduced in §5.3 and §5.4), and the way of utilizing such transformation (which will be introduced in §5.5).

The search space transformation maps a sample point from an unconstrained space to a point in the constrained space. The latter identifies an input signal **u**, in terms of its parameterized representation. This is reasonable as we consider piecewise constant signals as our input signal. We denote the discretized representation of **u** as a vector $\overrightarrow{u} = (u_{1,1}, \ldots, u_{1,M}, \ldots, u_{\mathsf{K},1}, \ldots, u_{\mathsf{K},M})$, where M is the dimension of **u** and K is the number of control points. We use it from now on to indicate the input signal. We thus restate the problem definition of Def. 16 as follows.

Definition 18 (Input constrained falsification problem) *The* input constrained falsification problem *is defined as:*

$$\begin{aligned} \underset{\overrightarrow{u}}{\textit{minimize}} \quad & [\![\mathcal{M}(\overrightarrow{u}), \varphi]\!] \\ \textit{subject to} \quad & \overrightarrow{u} \models \psi \\ & \overrightarrow{u} \in \Omega \end{aligned}$$

where ψ is a constraint on the input signal $\overrightarrow{u}$. The goal of the problem is to find an input signal $\overrightarrow{u}$ such that $\overrightarrow{u} \models \psi$, $\overrightarrow{u} \in \Omega$, and $[\![\mathcal{M}(\overrightarrow{u}), \varphi]\!] < 0$.

In this work, as ψ, we consider logical combinations of *linear constraints* (both equalities and inequalities). We now give the syntax of the supported constraints. Without loss of generality, we assume the logical constraints to be in Disjunctive Normal Form (DNF). In the following sections, for the sake of presentation, constraints are not given in DNF, but of course they can be transformed to it.

Definition 19 (Syntax of constraints) *We define an n-ary constraint ψ as follows:*

$$\psi ::\equiv \psi \vee \psi \mid \gamma \qquad \gamma ::\equiv \gamma \wedge \gamma \mid \xi$$
$$\xi ::\equiv \sum_{k=1}^{n} a_k x_k + a_{n+1} = 0 \mid \sum_{k=1}^{n} a_k x_k + a_{n+1} < 0 \mid \bot \mid \neg\xi$$

Here $a_1, \ldots, a_{n+1} \in \mathbb{R}$ are coefficients, and $x_1, \ldots, x_n$ variables, each one x_i defined over a domain D_i.

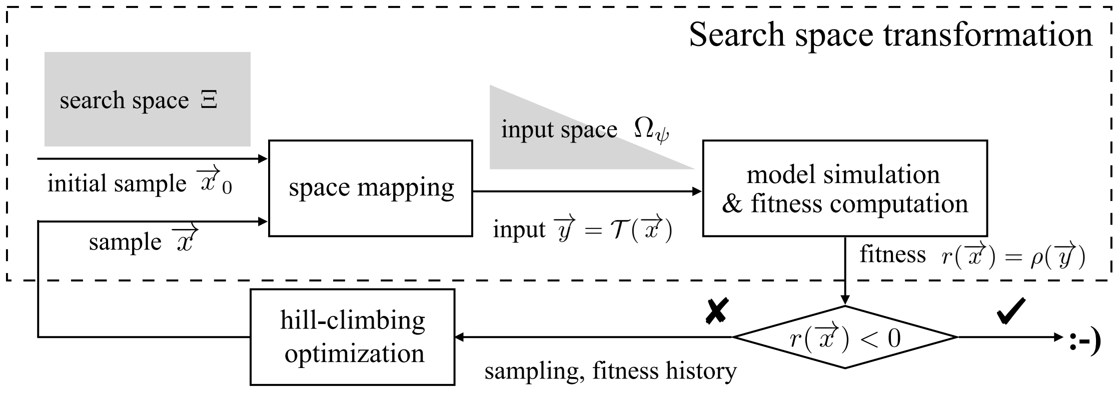

Search space transformation
search space Ξ
initial sample $\vec{x}_0$
sample $\vec{x}$
space mapping
input space Ω_ψ
input $\vec{y} = \mathcal{T}(\vec{x})$
model simulation & fitness computation
fitness $r(\vec{x}) = \rho(\vec{y})$
$r(\vec{x}) < 0$
✗
✓
:-)
hill-climbing optimization
sampling, fitness history

Figure 5.2: Proposed constrained falsification approach

In our context, the number of variables is $n = M\mathsf{K}$, because the input signal is a piecewise constant signal composed of M inputs having K control points.

Example 1 *The aforementioned example of constraint that throttle* and *brake* cannot be positive simultaneously *can be expressed as* $\bigwedge_{k=1}^{\mathsf{K}} (\mathit{throttle}_k = 0 \vee \mathit{brake}_k = 0)$.

In Def. 18, the actual input space is given by the application of ψ to Ω. We denote this *constrained space* as Ω_ψ, which contains all the points in Ω that satisfy ψ, i.e., $\Omega_\psi := \{\vec{x} \in \Omega \mid \vec{x} \models \psi\}$.

5.3.1 Search Space Transformation-Based Approach

In this rest of this chapter, we propose an approach for the input constrained falsification problem. The workflow is shown in Fig. 5.2. Let Ξ be an arbitrary hyperrectangle with the same number of dimensions as Ω_ψ. In the following, we name Ξ as a *search space* and Ω_ψ as an *input space*. The approach allows to perform the falsification search over the search space Ξ (weakly-bounded, so not harming the effectiveness of hill climbing) but, at the same time, it minimizes the fitness computed based on the input space Ω_ψ. To do this, the fitness function r of Ξ is defined in terms of the fitness (robustness) distribution ρ in Ω_ψ. More precisely, it employs a *search space transformation* that firstly maps a point $\vec{x}$ of the search space Ξ into a point $\vec{y}$ of the input space Ω_ψ through a space mapping $\mathcal{T}$ (i.e., $\vec{y} = \mathcal{T}(\vec{x})$), and then defines the fitness accordingly (i.e., $r(\vec{x}) = \rho(\vec{y})$). In this way, the constrained falsification problem is turned into an unconstrained optimization problem:

$$\begin{aligned} \underset{\overrightarrow{x}}{\mathit{minimize}} \quad & r(\overrightarrow{x}) \\ \mathit{subject\ to} \quad & \overrightarrow{x} \in \Xi \end{aligned}$$

Once a point $\overrightarrow{x}$ with negative fitness in Ξ is found, the mapped point $\overrightarrow{y} = \mathcal{T}(\overrightarrow{x})$ in Ω_ψ will be returned as falsifying input. Formally, the process of search space transformation is defined by the two following definitions.

Definition 20 (Space mapping) *Let Ξ be the search space, and Ω_ψ be the input space. We define a* space mapping *function $\mathcal{T}\colon \Xi \longrightarrow \Omega_\psi$ as a total surjective function from Ξ to Ω_ψ.*

We also define the fitness function of the points of the search space Ξ on the base of the fitness of the input space Ω_ψ.

Definition 21 (Fitness function in Ξ) *Let Ω_ψ be the input space, and $\rho\colon \Omega_\psi \longrightarrow \mathbb{R}$ be a fitness function for Ω_ψ. Let $\mathcal{T}\colon \Xi \longrightarrow \Omega_\psi$ be a space mapping from the search space Ξ to Ω_ψ. The fitness function $r\colon \Xi \longrightarrow \mathbb{R}$ in the search space Ξ is defined as $r(\overrightarrow{x}) := \rho\left(\mathcal{T}(\overrightarrow{x})\right)$.*

The search space transformation guarantees two properties necessary in our approach.

Proposition 1 (Soundness and completeness of the Search Space Transformation) *Any falsification algorithm that samples over the search space Ξ using the fitness function r as guidance is guaranteed to be* sound *and* complete*:*

Soundness*: If a sample $\overrightarrow{x}$ with negative fitness ($r(\overrightarrow{x}) < 0$) is found in the search space Ξ, the corresponding input $\overrightarrow{y} = \mathcal{T}(\overrightarrow{x})$ in the input space Ω_ψ is guaranteed to be a falsifying input ($\rho\left(\overrightarrow{y}\right) < 0$). As soon as such an $\overrightarrow{x}$ is found, the falsification process can stop and $\overrightarrow{y}$ can be returned as witness of the falsification.*

Completeness*: For each falsifying input $\overrightarrow{y}$ in the input space Ω_ψ, there is a sample $\overrightarrow{x}$ in the search space Ξ that maps to it, i.e., $\overrightarrow{y} = \mathcal{T}(\overrightarrow{x})$. This guarantees that the search over Ξ can find all the falsifying inputs (if any).*

The soundness comes from the definition of r (see Def. 21): once $r(\overrightarrow{x}) < 0$, it means that $\rho\left(\mathcal{T}(\overrightarrow{x})\right) < 0$ and thus $\rho\left(\overrightarrow{y}\right) < 0$. The completeness is from the surjectiveness of $\mathcal{T}$ (Def. 20).

Remark 1 *Prop. 1 states that* any *space mapping guarantees soundness and completeness of the approach. However, these are not the only desired properties. We would also like that the implemented space mapping does not harm the effectiveness of hill climbing. For guaranteeing this, hill climbing in the search space* Ξ *should get a* faithful *representation of the fitness landscape of the input space* Ω_ψ*.*

Continuity *of a space mapping, i.e., mapping points in proximity again to proximity, is a good criterion. We shall propose a specific class of continuous space mappings. It is called the* proportional transformation.

5.4 Proportional Transformation

Different search space transformations can be identified, that differ in the way they implement the space mapping (see Def. 20). In this section, we propose the *proportional transformation* $\mathcal{T}$ that maps each point of the search space Ξ into a point of the input space Ω_ψ, by proportionally scaling the value of each dimension of Ξ. It is illustrated in Fig. 5.3.

We call a set of non-overlapping intervals as *interval sequence.* Formally, an *interval sequence* over the real domain is defined as $R := (I_1, \ldots, I_q)$, where (a) each $I_j = [I_j^L, I_j^U]$ is a continuous interval with lower bound $I_j^L \in \mathbb{R}$ and upper bound $I_j^U \in \mathbb{R}$, such that $I_j^L \leq I_j^U$; (b) $I_j^U < I_{j+1}^L$ for each $j = 1, \ldots, q-1$. We denote the length of an interval I_j as $|I_j| = I_j^U - I_j^L$, and the *accumulated length* of all the intervals in R as $\mathtt{accLen}(R) = \sum_{j=1}^{q} |I_j|$.

We now provide a definition for determining the bounds of the constrained space identified by the constraints.

Definition 22 (Feasible interval sequence) *Let* Ξ *be the search space, and* $\psi = \vee_{i=1}^{t} \psi_i$ *an n-ary constraint in DNF defined over variables* $\overrightarrow{x} = (x_1, \ldots, x_n)$*, where each* ψ_i *is a conjunction of equalities and/or inequalities. Given a dimension* $d \in \{1, \ldots, n\}$*, we can identify the* bounds *of* ψ *over* d *as follows. For each conjunction* ψ_i*, we identify the* *minimum* Γ_i^L *and the maximum* Γ_i^U *of its feasible area, by solving two linear programming problems*[1] *(note that* ψ_i *only contains equalities and inequalities):*

$$\begin{aligned} \textit{minimize} \quad & x_d \\ \textit{subject to} \quad & \overrightarrow{x} \models \psi_i \\ & \overrightarrow{x} \in \Omega \end{aligned}$$

and

$$\begin{aligned} \textit{maximize} \quad & x_d \\ \textit{subject to} \quad & \overrightarrow{x} \models \psi_i \\ & \overrightarrow{x} \in \Omega \end{aligned}$$

Then, the feasible interval sequence $\mathbf{R}_d$ *of* ψ *on the* d*-th dimension is computed as follows:* $\mathbf{R}_d := \bigcup_{i=1}^{t} [\Gamma_i^L, \Gamma_i^U]$.

In the next definition, we show how a value belonging to a continuous interval can be mapped to an interval sequence.

Definition 23 (Proportional position) *Let* $A = [A^L, A^U]$ *be a continuous interval and* $v \in A$. *The proportional position* $\Theta(v, A, R)$ *of* v *in an interval sequence* $R = (I_1, \ldots, I_q)$ *is defined as follows:*

$$\Theta(v, A, R) \;:=\; p \cdot \texttt{accLen}(R) - \sum_{j=1}^{e} |I_j| + I_{e+1}^L$$

where:

- $p = \frac{v - A^L}{A^U - A^L}$ *is the proportional value of* v *in* A;
- $e \in \{1, \ldots, q\}$ *is the maximum index that satisfies* $p \cdot \texttt{accLen}(R) - \sum_{j=1}^{e} |I_j| > 0$.[2]

Definition 24 (Constraint reduction) *Let* ψ *be an* n*-ary constraint. Given a search space* Ξ *and a point* $\overrightarrow{u} \in \Xi$, *we compute the proportional position* $\pi_d = \Theta(u_d, D_d, \mathbf{R}_d)$ *over the* d*-th dimension of* $\overrightarrow{u}$ *(i.e.,* u_d*). Then, the function* $\texttt{Reduce}(\psi, \pi_d, d) := \psi[x_d \mapsto \pi_d]$ *is used to* reduce ψ *to an* $(n-1)$*-ary constraint.*

[1]They can be easily computed with any linear programming solver.
[2]Note that $\texttt{accLen}(R)$ can be 0. The implementation handles these cases.

Algorithm 5.1 Proportional transformation

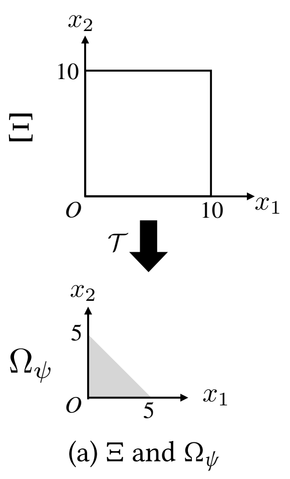
```
Require: a search space $\Xi = \prod_{k=1}^{n} D_k$, a sampled point $\overrightarrow{u} \in \Xi$, a constraint $\psi = \vee_{i=1}^{t}\psi_i$
    in DNF, a priority (permutation) $S$ of the dimensions $\{1, \ldots, n\}$.
 1: function MAP-POINT($\Xi, \psi, S, \overrightarrow{u}$)
 2:     $\overrightarrow{\pi} = (\pi_1, \ldots, \pi_n) \leftarrow (0, \ldots, 0)$                ▷ Initialize $\overrightarrow{\pi}$
 3:     MAP-DIMENSION($\Xi, \psi, S, \overrightarrow{u}, \overrightarrow{\pi}$)
 4:     Return $\overrightarrow{\pi}$                                  ▷ Final mapped point in $\Omega_\psi$

 5: procedure MAP-DIMENSION($\Xi, \psi, S, \overrightarrow{u}, \overrightarrow{\pi}$)
 6:     if length($S$) > 0 then
 7:         $s \leftarrow S$.head                        ▷ Obtain the first dimension in $S$
 8:         $\mathbf{R}_s \leftarrow \bigcup_{i=1}^{t}[\Gamma_i^L, \Gamma_i^U]$                   ▷ Feasible interval sequence
 9:         $\pi_s \leftarrow \Theta(u_s, D_s, \mathbf{R}_s)$                  ▷ Obtain proportional position
10:         $\psi' \leftarrow \mathtt{Reduce}(\psi, \pi_s, s)$                    ▷ Constraint reduction
11:         $S' \leftarrow$ remove $s$ from $S$
12:         MAP-DIMENSION($\Xi, \psi', S', \overrightarrow{u}, \overrightarrow{\pi}$)              ▷ Recursive call
13: end procedure
```

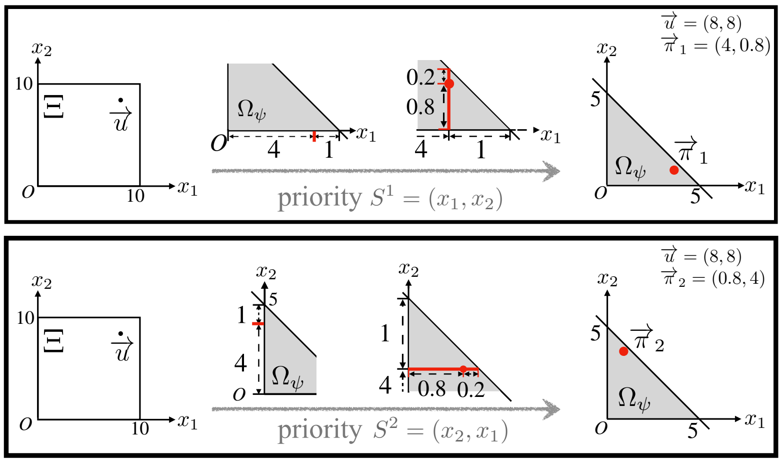
We define S as a permutation of the set $\{1, \ldots, n\}$ of the dimensions of Ξ. Here, S identifies the order in which the dimensions must be considered, and so it will be called *priority* in the following.

The proposed *proportional transformation* $\mathcal{T}$ maps a point $\overrightarrow{u}$ of Ξ into a point $\overrightarrow{\pi}$ of Ω_ψ, such that $\overrightarrow{\pi}$ satisfies the constraint ψ. To do this, it iteratively computes feasible interval sequences (Def. 22), identifies the proportional position (Def. 23), and performs constraint reduction (Def. 24), following a given priority order S, until all values on different dimensions of $\overrightarrow{u}$ have been mapped to their corresponding proportional positions. The computation of the proportional transformation is presented in Alg. 5.1.

The algorithm starts by initializing an n-dimensional point $\overrightarrow{\pi}$ (Line 2), and invoking the procedure MAP-DIMENSION using as arguments the search space Ξ, the constraint ψ, and the priority S (Line 3). The procedure MAP-DIMENSION also receives the point $\overrightarrow{\pi}$, and iteratively modifies its value on each dimension. In each loop, the procedure obtains the first element s of the priority S, and determines the *feasible interval sequence* $\mathbf{R}_s$ of s dimension, following the rules in Def. 22 (Line 8). Then, the proportional position $\Theta(u_s, D_s, \mathbf{R}_s)$ of $\overrightarrow{u}$ on s dimension is computed according to Def. 23 (Line 9), and the constraint ψ is reduced over s dimension following Def. 24 (Line 10). Finally,



(a) Ξ and Ω_ψ



(b) Two proportional transformations

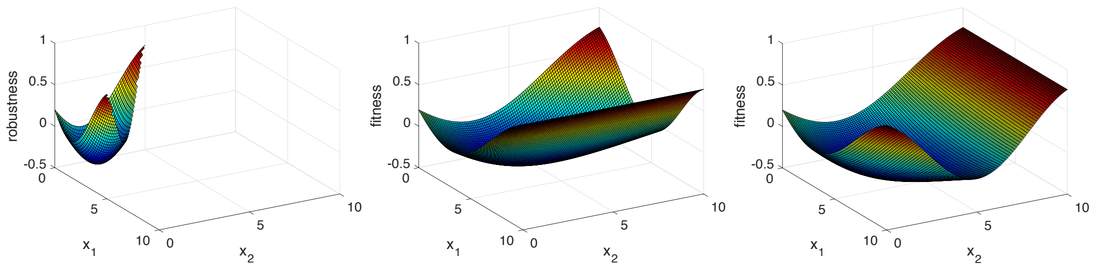
Figure 5.3: Running example–proportional transformation

the priority S is updated by removing the first element s (Line 11), and the procedure Map-Dimension is invoked again to reduce the remaining arguments (Line 12). The recursive call terminates when all dimensions have been mapped. At the end, $\vec{\pi}$ is returned (Line 4) as the mapped point in Ω_ψ that satisfies the constraint ψ.

It is easy to see that the proportional transformation $\mathcal{T}$ is a space mapping in Def. 20, that is, it is a total surjection. Indeed, one can follow its definition and construct, in a step-by-step manner, a right inverse g of $\mathcal{T}$ (i.e. $\mathcal{T} \circ g = \text{id}$). Existence of such g witnesses the surjectiveness of $\mathcal{T}$. Continuity of $\mathcal{T}$ is easily established, too.

Example 2 *We use a simple example to explain our approach. We consider* $\Xi = [0, 10] \times [0, 10]$ *as search space, and* $\psi = (x_1 + x_2 - 5 < 0)$ *as constraint defining the input space* Ω_ψ*, as shown in Fig. 5.3a. Let us consider a point* $\vec{u} = (8, 8) \in \Xi$*. Let us call* x_1 *and* x_2 *the two dimensions of the search space. Using the priority* $S^1 = (x_1, x_2)$*,* $\vec{u}$ *is mapped to point* $\vec{\pi}_1 = (4, 0.8)$*; instead, using the priority* $S^2 = (x_2, x_1)$*, it is mapped to point* $\vec{\pi}_2 = (0.8, 4)$*. The two proportional transformations are shown in Fig. 5.3b.*

Remark 2 *Note that the general transformation process is not specialized to linear constraints, and can be adapted for any type of constraints. What needs to be adapted is Def. 22 to find the bounds over a given dimension: different types of constraints need different solvers (for linear constraints, we use a linear programming solver). In this*



(a) In Ω_ψ (b) In Ξ (priority S^1) (c) In Ξ (priority S^2)

Figure 5.4: Fitness landscape of Ω_ψ and transformed fitness landscapes in Ξ

paper, we focus and perform experiments on linear constraints. Extending the approach to non-linear constraints is left as future work.

5.5 Falsification Based on the Proportional Transformation

In this section, we describe how we use the proportional transformation presented in §5.4 to implement a falsification algorithm that considers the constraints existing among the inputs. Namely, we adapt the hill climbing-guided falsification approach described in §2.3. The approach performs classical hill-climbing optimization over the search space Ξ; sampled points $\overrightarrow{u}_k$ are mapped to points $\overrightarrow{\pi}_k$ of the input space Ω_ψ, using the proportional transformation presented in §5.4. In this context, the fitness function ρ of Ω_ψ is given by the robustness value of the mapped input $\overrightarrow{\pi}_k$ for the specification φ, i.e., $\rho(\overrightarrow{\pi}_k) = [\![\mathcal{M}(\overrightarrow{\pi}_k), \varphi]\!]$. See the whole workflow in Fig 5.2. We can notice that the hill-climbing algorithm (performed over the whole search space Ξ) has a *deformed* view of the fitness landscape of the input space Ω_ψ. The obtained deformed landscape depends on the priority used in the proportional transformation (see §5.4, Alg. 5.1, and the two proportional transformations in Fig. 5.3b).

Example 3 *Let us consider Ex. 2. The fitness landscape of the input space Ω_ψ (produced by a given objective function) is as shown in Fig. 5.4a. By applying the proportional transformation using the two priorities S^1 and S^2 (as shown in Fig. 5.3b), we obtain the fitness landscapes in Fig. 5.4b and Fig. 5.4c.*

Algorithm 5.2 Fixed-Priority approach

```
Require: a system model $\mathcal{M}$, an STL formula $\varphi$, a constraint $\psi$, a priority of dimensions $S$, and a budget $\mathbf{K}$
1: function FALS-FIXED-PRIORITY($\mathcal{M}, \varphi, \mathbf{K}, \psi, S$)
2:     $\mathsf{rb} \leftarrow \infty$ ;   $k \leftarrow 0$          ▷ rb is the smallest robustness so far, initialized to ∞
3:     while $\mathsf{rb} \geq 0$ and $k \leq \mathbf{K}$ do
4:         $k \leftarrow k + 1$
5:         $\overrightarrow{u}_k \leftarrow$ HILL-CLIMB$\Big( \big(\overrightarrow{u}_l, [\![\mathcal{M}(\overrightarrow{u}_l), \varphi]\!]\big)_{l\in[1,k-1]} \Big)$
6:         $\overrightarrow{\pi}_k \leftarrow$ MAP-POINT$(\Xi, \psi, S, \overrightarrow{u}_k)$          ▷ Proportional transformation
7:         $\mathsf{rb}_k \leftarrow [\![\mathcal{M}(\overrightarrow{\pi}_k), \varphi]\!]$          ▷ Robustness value of the mapped point
8:         if $\mathsf{rb}_k < \mathsf{rb}$ then
9:             $\mathsf{rb} \leftarrow \mathsf{rb}_k$
10:    $\overrightarrow{\pi} \leftarrow \begin{cases} \overrightarrow{\pi}_k & \text{if } \mathsf{rb} < 0 \text{, that is, } \mathsf{rb}_k = [\![\mathcal{M}(\overrightarrow{\pi}_k), \varphi]\!] < 0 \\ \text{Failure} & \text{otherwise, that is, no falsifying input found within budget } \mathbf{K} \end{cases}$
11:    Return $\overrightarrow{\pi}$
```

As we will show in the experiments in §5.6, the chosen priority can greatly affect the performance of the falsification. In the following sections, we consider three methods for selecting the priority: selecting one priority, considering all the priorities, or *learning* which priority is better.

5.5.1 Method 1: Fixed-Priority

In this approach, the user must provide a given priority order S. Alg. 5.2 shows how the classical hill climbing-guided falsification has been modified to implement the proportional transformation with Fixed-Priority. There are two differences: firstly, the algorithm now also considers a constraint ψ and, for computing the fitness of an input $\overrightarrow{u}_k$ sampled in the search space Ξ, it first maps it to a point $\overrightarrow{\pi}_k$ in the input space Ω_ψ using the proportional transformation (Line 6), and then uses the robustness of $\overrightarrow{\pi}_k$ as fitness for $\overrightarrow{u}_k$ (Line 7); secondly, the final falsifying input (if any) is a point $\overrightarrow{\pi}_k$ of the input space (Line 10).

Algorithm 5.3 All-Priorities approach

```
Require: a system model $\mathcal{M}$, an STL formula $\varphi$, constraint $\psi$, priorities $Prior$, and a budget $\mathbf{K}$
1: function FALS-ALL-PRIORITIES($\mathcal{M}, \varphi, \mathbf{K}, \psi, Prior$)
2:     $\mathrm{rb} \leftarrow \infty$ ;   $k \leftarrow 0$    ▷ rb is the smallest robustness so far, initialized to $\infty$
3:     while $\mathrm{rb} \geq 0$ and $k \leq \mathbf{K}$ do
4:         $k \leftarrow k + 1$
5:         $\overrightarrow{u}_k \leftarrow$ HILL-CLIMB$\Big( \big(\overrightarrow{u}_l, [\![\mathcal{M}(\overrightarrow{u}_l), \varphi]\!]\big)_{l \in [1,k-1]} \Big)$
6:         $\Pi \leftarrow \{$MAP-POINT$(\Xi, \psi, S, \overrightarrow{u}_k) \mid S \in Prior\}$ ▷ Proportional transformations
7:         $\overrightarrow{\pi}_k \leftarrow \arg\min_{\overrightarrow{\pi} \in \Pi} [\![\mathcal{M}(\overrightarrow{\pi}), \varphi]\!]$    ▷ Selection of best mapped point
8:         $\mathrm{rb}_k \leftarrow [\![\mathcal{M}(\overrightarrow{\pi}_k), \varphi]\!]$    ▷ Robustness value of the selected mapped point
9:         if $\mathrm{rb}_k < \mathrm{rb}$ then
10:            $\mathrm{rb} \leftarrow \mathrm{rb}_k$
11:     $\overrightarrow{\pi} \leftarrow \begin{cases} \overrightarrow{\pi}_k & \text{if rb} < 0\text{, that is, rb}_k = [\![\mathcal{M}(\overrightarrow{\pi}_k), \varphi]\!] < 0 \\ \text{Failure} & \text{otherwise, that is, no falsifying input found within budget } \mathbf{K} \end{cases}$
12:     Return $\overrightarrow{\pi}$
```

5.5.2 Method 2: All-Priorities

Although the proportional transformation is surjective, it changes the distribution of fitness on the base of the selected priority, and so it influences the performance of hill-climbing optimization. Therefore, different priorities lead to different falsification performance and different results. The current method is based on this observation, and so it considers *all* the priorities. Alg. 5.3 shows the implementation of the All-Priorities approach (differences w.r.t. Alg. 2.1). At Line 6, the approach now generates *all* the mapped points Π of $\overrightarrow{u}_k$, using the priorities contained in set *Prior* given as input.

The set *Prior* is built as follows. Initially, all the permutations of the dimensions $\{1, \ldots, n\}$ are added to the set. However, some priorities are guaranteed to map the points in the same way. Therefore, such *equivalent priorities* are identified with these two rules:

(a) Only the variables contained in the constraint ψ affect the result of the priority application. Given two priorities S^1 and S^2, if the relative order of variables contained in ψ is the same in S^1 and S^2, then they are equivalent;

(b) Two variables are *independent* if they occur in the same conjunction ψ_i of ψ (recall that ψ is in DNF) and they are not in the same atomic proposition. Given two priorities S^1 and S^2, if they only differ in the relative order of independent variables, then they are equivalent.

So, *Prior* is the set of all non-equivalent permutations of $\{1, \ldots, n\}$.

Note that, even if two priorities are not equivalent, a point can still be mapped to the same point under both priorities: therefore, at Line 6, duplicated points are removed.

At Lines 7-8, the algorithm determines the point $\overrightarrow{\pi}_k$ in Π having the minimum robustness value rb_k. Finally, at Lines 11-12, it returns a falsifying input $\overrightarrow{\pi}$ in the input space or reports a failure.

5.5.3 Method 3: MAB-Priority

Although the All-Priorities approach guarantees to find the *best* priority (i.e., the one mapping to points with minimum robustness), it is computationally expensive, as it requires to simulate all the mapped points. In this section, we propose a method that tries to *learn* the best priority during execution: it is based on the Multi-Armed Bandit (MAB) problem, that has proven to be effective in other contexts for falsification [89] (our contribution in Chapter 4).

We first provide an introduction to the Multi-Armed Bandit problem, and then we describe how we apply it to our context.

Multi-Armed Bandit problem The *Multi-Armed Bandit* (MAB) problem describes the situation where a gambler sits in front of a row $A_1, \ldots, A_m$ of slot machines, each one giving, when its arm is played (i.e., in each attempt), a reward according to a prescribed (but unknown) probability distribution μ_i. The goal is to maximize the cumulative reward after a number of attempts, playing a suitable arm in each attempt. The best strategy of course is to keep playing the best arm $A_{\max}$, i.e., the one whose average reward $\mathrm{avg}(\mu_{\max})$ is the greatest. However, this best strategy is infeasible, because the distributions $\mu_1, \ldots, \mu_m$ are initially unknown. Therefore, the gambler must learn about $\mu_1, \ldots, \mu_m$ through attempts. A formal definition of the MAB problem is as follows.

Definition 25 (The Multi-Armed Bandit (MAB) problem) ***Input:*** *arms* $(A_1, \dots, A_m)$, *the associated probability distributions* $\mu_1, \dots, \mu_m$ *over* $\mathbb{R}$, *and a time horizon* $H_T \in \mathbb{N} \cup \{\infty\}$.

Goal: *synthesize a sequence* $A_{i_1} A_{i_2} \dots A_{i_H}$, *so that the cumulative reward* $\sum_{k=1}^{H_T} \mathrm{rew}_k$ *is maximized. Here the reward* rew_k *of the* k*-th attempt is sampled from the distribution* μ_{i_k} *associated with the arm* A_{i_k} *played at the* k*-th attempt.*

We introduce some notations for later use. Let $\langle (A_{i_1} \dots A_{i_k}), (\mathrm{rew}_1 \dots \mathrm{rew}_k) \rangle$ *be a* history, *i.e., the sequence of arms played so far (here* $i_1, \dots, i_k \in \{1, \dots, m\}$*), and the sequence of rewards obtained by those attempts (*rew_l *is sampled from* μ_{i_l}*).*

MAB-based falsification In our context, an arm is a priority S. The reward of a priority S is based on the minimum robustness value observed when using S for mapping the sampled point (i.e., playing that arm). The hill-climbing algorithm implementing the MAB approach is shown in Alg. 5.4 (differences w.r.t. Alg. 2.1). In Line 5, an MAB algorithm is run to decide which priority S, taken from a set of priorities *Prior* (see §5.5.2), must be executed in the k-th attempt.

The function MAB takes as inputs: (i) priorities $\mathit{Prior} = \{S^1, \dots, S^m\}$ (i.e., the arms); (ii) the history $(S^{i_1}, \dots, S^{i_{k-1}})$ of previously played arms; and (iii) the history of robustness values $(\mathrm{rb}_1, \dots, \mathrm{rb}_{k-1})$ of the previously selected inputs.

Our MAB algorithm is based on the UCB1 (*upper confidence bound*) algorithm. UCB1 algorithm exemplifies the *exploitation* and *exploration* trade-off over the set of arms. It is instantiated at Line 13 of Alg. 5.4: it returns the index i_k of the priority S^{i_k} that has the largest sum of *exploitation* and *exploration* score (Line 14). Given a priority S^z, the *exploitation score* identifies the empirical reward $\mathtt{Rew}(z, k-1)$, and it follows the formal definition in [42], that considers the robustness obtained in previous loops: the lower the observed robustness is, the higher the reward assigned to the arm is. The definition is as follows: $\mathtt{Rew}(z, k-1) = \left(1 - \frac{\min_{l \in \{1,\dots,k-1\} \text{ s.t. } i_l = z} \mathrm{rb}_l}{\max_{l \in \{1,\dots,k-1\}} \mathrm{rb}_l}\right)$.

The *exploration score* is a value negatively correlated to the number of attempts of the arm of priority S^z. At Line 14, $N(z, k-1)$ identifies the number of attempts of priority S^z in the previous $k-1$ steps $S^{i_1} \dots S^{i_{k-1}}$. The scalar c is used to give more importance to either exploration or exploitation.

Algorithm 5.4 MAB-Priority approach

```
Require: a system model $\mathcal{M}$, an STL formula $\varphi$, constraint $\psi$, permutations $Prior = \{S^1, \ldots, S^m\}$ of dimensions $\{1, \ldots, n\}$, and a budget $\mathbf{K}$
1: function FALS-MAB-PRIORITY($\mathcal{M}, \varphi, \mathbf{K}, \psi, Prior$)
2:     $\mathrm{rb} \leftarrow \infty$ ;   $k \leftarrow 0$        ▹ rb is the smallest robustness so far, initialized to $\infty$
3:     while $\mathrm{rb} \geq 0$ and $k \leq \mathbf{K}$ do
4:         $k \leftarrow k + 1$
5:         $i_k \leftarrow \mathrm{MAB}(Prior, \langle (S^{i_1} \ldots S^{i_{k-1}}), (\mathrm{rb}_1 \ldots \mathrm{rb}_{k-1}) \rangle)$  ▹ Selection of the priority
6:         $\overrightarrow{u}_k \leftarrow \textsc{Hill-Climb}\Big( \big( (\overrightarrow{u}_l,\ \mathrm{rb}_l) \big)_{l \in \{1,\ldots,k-1\} \text{ such that } i_l = i_k} \Big)$
                   ▹ Hill climbing suggests $\overrightarrow{u}_k$ based on sampling history of $S^{i_k}$
7:         $\overrightarrow{\pi}_k \leftarrow \textsc{Map-Point}(\Xi, \psi, S^{i_k}, \overrightarrow{u}_k)$        ▹ Proportional transformation
8:         $\mathrm{rb}_k \leftarrow [\![ \mathcal{M}(\overrightarrow{\pi}_k), \varphi ]\!]$        ▹ Robustness value of the mapped point
9:         if $\mathrm{rb}_k < \mathrm{rb}$ then
10:            $\mathrm{rb} \leftarrow \mathrm{rb}_k$
11:    $\overrightarrow{\pi} \leftarrow \begin{cases} \overrightarrow{\pi}_k & \text{if rb} < 0\text{, that is, } \mathrm{rb}_k = [\![ \mathcal{M}(\overrightarrow{\pi}_k), \varphi ]\!] < 0 \\ \text{Failure} & \text{otherwise, that is, no falsifying input found within budget } \mathbf{K} \end{cases}$
12:    Return $\overrightarrow{\pi}$
13: function MAB($Prior, \langle (S^{i_1} \ldots S^{i_{k-1}}), (\mathrm{rb}_1 \ldots \mathrm{rb}_{k-1}) \rangle$)
14:    $i_k \leftarrow \underset{z \in \{1,\ldots,|Prior|\}}{\arg\max} \left( \mathrm{Rew}(z, k-1) + c \sqrt{\frac{2 \ln(k-1)}{N(z, k-1)}} \right)$
15:    Return $i_k$
```

5.6 Experimental Evaluation

In this section, we present the experiments we performed to evaluate the effectiveness of the proposed approaches. We take the penalty-based approaches introduced in §5.2 as baselines comparison. Then, in §5.6.1 we introduce the experiments setup, and in §5.6.2 we illustrate the experiments and evaluate the results using some research questions.

5.6.1 Experiment Setup

Baselines We compare the performance of the search space transformation-based approaches presented in §5.3, §5.4 and §5.5 with two penalty-based approaches—the *Constraint Embedding* method in §5.2.1 and the *Lexicographic Method* in §5.2.2.

In the lexicographic method-based approaches proposed in §5.2.2, we need to choose a proper base number B and transformation functions $\mathcal{T}_1$ and $\mathcal{T}_2$ for the global cost function. Regarding B, we selected 10 in our experiments because it performs the best according to [82]. As for transformation functions $\mathcal{T}_1$ and $\mathcal{T}_2$, we need to determine R^{ψ}_{max} and R^{φ}_{max} in each case (see §5.2.2). We handle this problem as follows. We take a small set of samplings of the input space and compute their robustness values (both for the input constraint and the specification). Then, for the input constraints, we determine R^{ψ}_{max} by multiplying the maximum value of the obtained robustness values by a reasonable factor. For the specification, we determine R^{φ}_{max} in a similar way.

Models, specifications and constraints We experiment our approaches over the benchmarks used in the falsification community [44]. In order to make a comprehensive and reliable comparison, we select 4 Simulink models with 20 system specifications. The hardness of these specifications depends on their parameters; in our experiments, we vary the parameters to obtain problems of different difficulties. Each specification has been experimented with different input constraints taken from [82].

The constraints for the considered models usually predicate about the inputs over time. Therefore, we allow users to specify input constraints in STL, as we stated in §5.1.1; such constraints predicate over variables $u_1, \ldots, u_M$ (one for each input). However, the constraints supported by the search space transformation-based approach are a combination of linear constraints (no temporal operators) defined over variables $\overrightarrow{u} = (u_{1,1}, \ldots, u_{1,M}, \ldots, u_{\mathsf{K},1}, \ldots, u_{\mathsf{K},M})$ (see Defs. 19 and §5.3), where, for each input u_i, there are K variables $u_{1,i}, \ldots, u_{\mathsf{K},i}$ (one for each control point). We present how STL constraints can be translated to our supported format as follows:

Such constraints predicate over variables $u_1, \ldots, u_M$ (one for each input). However, the constraints we support in our approach are a combination of linear constraints (no temporal operators) defined over variables $\overrightarrow{u} = (u_{1,1}, \ldots, u_{1,M}, \ldots, u_{\mathsf{K},1}, \ldots, u_{\mathsf{K},M})$.

(a) each subformula $\Box_{[t_1,t_2]}(\gamma)$ of ψ_{STL} is transformed in $\bigwedge_{k=\lfloor \frac{\mathsf{K}}{T} t_1 \rfloor}^{\lfloor \frac{\mathsf{K}}{T} t_2 \rfloor} (\gamma[u_1 \mapsto u_{k,1}, \ldots, u_M \mapsto x_{k,M}])$;

(b) each subformula $\Diamond_{[t_1,t_2]}(\gamma)$ of ψ_{STL} is transformed in $\bigvee_{k=\lfloor \frac{\mathsf{K}}{T} t_1 \rfloor}^{\lfloor \frac{\mathsf{K}}{T} t_2 \rfloor} (\gamma[u_1 \mapsto u_{k,1}, \ldots, u_M \mapsto x_{k,M}])$;

(c) the formula obtained from the two previous substitutions is transformed in DNF.

Table 5.1: Temporal specifications φ. Here, $\mathbf{w}^t$ represents the *t-shift* of $\mathbf{w}$ (see Def. 8) and $\Delta_t(\mathbf{w})$ represents $\mathbf{w}^t - \mathbf{w}$

Model	Spec. ID	Temporal specification in STL
AT	AT1	$\square_{[0,30]}\ (speed < 120)$
	AT2	$\square_{[0,30]}\ (gear = 3 \rightarrow speed \geq 20)$
	AT3	$\square_{[0,30]}\ (gear = 4 \rightarrow speed \geq 35)$
	AT4	$\neg(\square_{[10,30]}((50 < speed) \wedge (speed < 60)))$
	AT5	$\neg(\square_{[10,30]}((53 < speed) \wedge (speed < 57)))$
	AT6	$\square_{[0,29]}(speed < 100) \vee \square_{[29,30]}(speed > 75)$
	AT7	$\square_{[0,29]}(speed < 100) \vee \square_{[29,30]}(speed > 70)$
	AT8	$\square_{[0,30]}(rpm < 4770 \vee \square_{[0,1]}(rpm > 1000))$
	AT9	$\square_{[0,30]}(rpm < 4770 \vee \square_{[0,1]}(rpm > 700))$
	AT10	$\square_{[0,30]}(rpm < 3000) \rightarrow \square_{[0,20]}(speed < 65)$
	AT11	$\square_{[0,10]}\ (speed < 50) \vee \diamond_{[0,30]}\ (rpm > 2700)$
	AT12	$\square_{[0,10]}\ (speed < 50) \vee \diamond_{[0,30]}\ (rpm > 2520)$
	AT13	$\square_{[0,26]}(\Delta_4(speed) > 40 \rightarrow \Delta_4(gear) > 0)$
	AT14	$\square_{[0,27]}(\Delta_3(speed) > 30 \rightarrow \Delta_3(gear) > 0)$
AFC	AFC1	$\square_{[11,50]}(\mu < 0.23)$
	AFC2	$\square_{[11,50]}(\diamond_{[0,10]}(\lvert\mu\rvert < 0.05))$
NN		$NN\_req \equiv \square_{[0,16]}(\neg close\_ref \rightarrow reach\_ref\_in\_tau)$
		$close\_ref \equiv \lvert Pos - Ref\rvert \leq \alpha_1 + \alpha_2 \cdot \lvert Ref\rvert$
		$reach\_ref\_in\_tau \equiv \diamond_{[0,2]}(\square_{[0,1]}(close\_ref))$
	NN1	NN_req with $\alpha_1 = 0.003, \alpha_2 = 0.04$
	NN2	NN_req with $\alpha_1 = 0.015, \alpha_2 = 0.03$
FFR	FFR1	$\neg\diamond_{[0,5]}(x, y \in [3.9, 4.1] \wedge \dot{x}, \dot{y} \in [-1, 1])$
	FFR2	$\neg\diamond_{[0,5]}(x, y \in [3.95, 4.05] \wedge \dot{x}, \dot{y} \in [-0.5, 0.5])$

The specifications are reported in Table 5.1. The input constraints are reported in Table 5.2, where each constraint is in its two forms, namely STL and format of Def. 19. As the Simulink models have been used in the experiments of chapters before, we just give a brief introduction, as below.

Automatic Transmission (AT) [98] It has two input signals, *throttle* (throttle) and *brake* (brake), and produces outputs signals such as *speed*, *rpm*, *gear*, etc. The model is composed of 6 sub-systems, 1 Stateflow chart, and 72 blocks in total. The *throttle* and *brake* range over $[0, 100]$ and $[0, 325]$ respectively, each with 5 control points. We select specifications AT1, …, AT14, concerned with system's safety, from literature [42,

Table 5.2: Input constraints ψ. Here, $\mathbf{w}^t$ represents the *t-shift* of $\mathbf{w}$ (see Def. 8) and $\Delta_t(\mathbf{w})$ represents $\mathbf{w}^t - \mathbf{w}$

Model	Constr. ID	ψ in STL and in the format of Def. 19
AT	ψ^1_{AT}	$\square_{[0,30]}(throttle = 0 \vee brake = 0)$ $\bigwedge_{k=1}^{K} (throttle_k = 0 \vee brake_k = 0)$
	ψ^2_{AT}	$\square_{[0,30]}(throttle \leq 20 \vee brake \leq 50)$ $\bigwedge_{k=1}^{K} (throttle_k \leq 20 \vee brake_k \leq 50)$
	ψ^3_{AT}	$\square_{[0,30]}(throttle \geq 3 \cdot brake \vee brake \geq 3 \cdot throttle)$ $\bigwedge_{k=1}^{K} (throttle_k \geq 3 \cdot brake_k \vee brake_k \geq 3 \cdot throttle_k)$
	ψ^4_{AT}	$\square_{[0,24]}(throttle \geq 70 \rightarrow throttle^6 \leq 10)$ $\bigwedge_{k=1}^{K-1} (throttle_k \geq 70 \rightarrow throttle_{k+1} \leq 10)$
	ψ^5_{AT}	$\square_{[6,30]}(throttle = 0 \vee brake = 0) \wedge \square_{[0,6]}(brake = 0)$ $\bigwedge_{k=2}^{K} (throttle_k = 0 \vee brake_k = 0) \wedge brake_1 = 0$
AFC	ψ^1_{AFC}	$\square_{[0,50]}(Pedal\_Angle \geq 50 \rightarrow Engine\_Speed \geq 1000)$ $\bigwedge_{k=1}^{K} \left(Pedal\_Angle_k \geq 50 \rightarrow Engine\_Speed_k \geq 1000\right)$
	ψ^2_{AFC}	$\square_{[0,20]}(\Delta_{10}(Pedal\_Angle) \geq 0)$ $\bigwedge_{k=1}^{3} \left(Pedal\_Angle_k \leq Pedal\_Angle_{k+1}\right)$
NN	ψ^1_{NN}	$\square_{[0,14]}(\Diamond_{[0,6]}(Ref \geq 2.5))$ $\bigwedge_{k=1}^{\frac{c}{2}} \left(\bigvee_{j=1}^{2} Ref_k \geq 2.5\right)$
	ψ^2_{NN}	$\square_{[0,15]}(\Delta_5(Ref) \geq 0)$ $\bigwedge_{k=1}^{K} \left(Ref_k \leq Ref_{k+1}\right)$
FFR	ψ^1_{FFR}	$\square_{[0,5]} \left(\left(\bigvee_{\sim \in \{\leq,\geq\}} \bigwedge_{i \in \{1,3\}} u_i \sim 0 \right) \wedge \left(\bigvee_{\sim \in \{\leq,\geq\}} \bigwedge_{i \in \{2,4\}} u_i \sim 0 \right) \right)$ $\bigwedge_{k=1}^{K} \left(\left(\bigvee_{\sim \in \{\leq,\geq\}} \bigwedge_{i \in \{1,3\}} u_{i_k} \sim 0 \right) \wedge \left(\bigvee_{\sim \in \{\leq,\geq\}} \bigwedge_{i \in \{2,4\}} u_{i_k} \sim 0 \right) \right)$

44, 89, 98]. We consider 5 input constraints, covering both equalities and inequalities.

Abstract Fuel Control (AFC) [56] It takes two input signals, *Pedal_Angle* (pedal angle) and *Engine_Speed* (engine speed), and outputs a ratio μ reflecting the deviation of *air-fuel-ratio* from its reference value. In our experiment, we set the range of *Pedal_Angle* $\in [8.8, 70]$ and *Engine_Speed* $\in [900, 1100]$, each with 5 control points. The model is composed of 20 sub-systems, and 271 blocks in total. Specifications AFC1 and AFC2 reason about the expected safety properties of the system. We specify two input constraints, one constraining the value of *Engine_Speed* w.r.t. the value of *Pedal_Angle*, and another one constraining the value of *Pedal_Angle* over time.

Neural Network controller (NN) It is a neural network controller for a magnet system from Mathworks. Specifications NN1 and NN2 formalize the safety requirement about the position *Pos* of the magnet w.r.t. its reference value *Ref*. The input signal *Ref* ranges over $[1, 3]$ with 4 control points. The model is composed of 11 sub-systems, including one neural network-based controller, and 104 blocks in total. We consider two input constraints: the first one requiring *Ref* to be non-decreasing, and the second one requiring *Ref* to be larger of 2.5 in at least one time point.

Free Floating Robot (FFR) It is a model considered as a falsification benchmark in [69] and [42]. The inputs $u_1, u_2, u_3, u_4 \in [-10, 10]$ are four boosters for a robot, and the goal is to steer it from $(x, y) = (0, 0)$ to $(4, 4)$ in a 2-dimensional space. The model contains 32 blocks in total. We take 4 control points for input signals. The constraint we consider is a real one: u_1 and u_3 should be both positive or negative, and so should u_2 and u_4, according to [42]; otherwise, the boosters would conflict with each other.

Experiment platform In our experiments, we use Breach [22] (ver 1.2.13) with CMA-ES (the state of the art). The experiments are executed on an Amazon EC2 c4.2xlarge instance (2.9 GHz Intel Xeon E5-2666 v3, 15 GB RAM).

5.6.2 Evaluation

We performed a set of experiments using the two baseline approaches Constraint Embedding (CE) and Lexicographic Method (LM), and the three proposed approaches Fixed-Priority (`Fix`), All-Priorities (ALL), and MAB-Priority (MAB). Since Fixed-Priority requires to select a given priority, for each benchmark we randomly selected two priorities S^1 and S^2: we name the two settings as $\texttt{Fix}_{S^1}$ and $\texttt{Fix}_{S^2}$.

In our context, an *experiment* consists in the execution of an approach A (CE, LM, $\texttt{Fix}_{S^1}$, $\texttt{Fix}_{S^2}$, ALL, or MAB) over a specification φ for 30 *trials*, with different random seeds; each single trial has been executed with a time budget $\mathbf{K}$ of 900 secs. For each experiment, we define the *success rate* SR as the number of trials in which a falsifying input was found, and measure the average execution *time* of the successful trials. Note that time is correlated with the number of simulations, because simulation is much more computationally expensive than other steps, e.g., proportional transformation.

In the following, we compare two approaches $A_1, A_2 \in \{$CE, LM, $\texttt{Fix}_{S^1}$, $\texttt{Fix}_{S^2}$, ALL, MAB} by comparing SR using the non-parametric Wilcoxon signed-rank test with 5% level of significance [100]. The null hypothesis is that there is no statistical significant difference in applying A_1 or A_2 in terms of SR; if the null hypothesis is rejected, we check the alternative hypothesis that A_1 is better than A_2 (higher SR).

Experimental results are reported in Table 5.3. The gray cells are local best performers: they have the best SR with minimum time. Table 5.4 reports the results of the Wilcoxon signed-rank test between each pair of techniques in terms of SR.

We now analyze the results using three research questions.

RQ1 *Do the proposed approaches outperform the two baseline approaches?*

In this RQ, we want to assess whether we improve w.r.t. the state of the art. From Table 5.3, we observe that sometimes the two baseline approaches CE and LM are not able to find any feasible falsifying input over the 30 trials: for example, AT7 for almost all the constraints, and AT8 and AT9 for ψ_{AT}^4. Our proposed approaches, instead, are almost always successful in at least one trial. Exceptions are $\texttt{Fix}_{S^1}$ with AT6, $\texttt{Fix}_{S^2}$ with AT7, and ALL with AT5, all under constraint ψ_{AT}^1.

Also when the two baseline approaches do find at least a falsifying input, our approaches in general perform better.

The statistical tests in Table 5.4 confirm the previous qualitative evaluation: all our approaches are statistically better than the two baseline approaches.

Note that both the baseline approaches and our proposed approaches modify the fitness landscape. However, in the baseline approaches, the fitness landscape is given by the *composition* of the degree of violation of the constraints and of the robustness; in this way, the falsification task performed by hill-climbing is complicated. Moreover, note that the *scales* (i.e., orders of magnitude) of constraint violation and robustness may be very different, and this has been shown to affect the effectiveness of falsification algorithms [89]. In our approach, instead, hill-climbing operates over a fitness landscape that, although deformed, it is only given by robustness.

We notice that in many cases the proposed approaches improve the baselines in time. This is reasonable: since the baselines make objective functions much more complex, they need more simulations (thus time) to find the falsifying input. Note that all infeasible samplings require simulation, so wasting time for falsification.

We want now to assess the effect of the proportional transformation on the

Table 5.3: Experimental results (SR: the number of successes out of 30 trials. #sim: the number of simulations. Time in secs.)

(a) Automatic Transmission

		AT1			AT2			AT3			AT4			AT5			AT6			AT7		
		SR	#sim	time	SR	#sim	time	SR	#sim	time	SR	#sim	time	SR	#sim	time	SR	#sim	time	SR	#sim	time
ψ^1_{AT}	CE	22	334	130.1	6	320	121.0	1	186	66.1	25	318	120.1	26	741	286.6	7	1094	481.6	0	-	-
	LM	9	477	194.2	0	-	-	0	-	-	15	291	116.0	9	860	348.5	1	399	154.1	0	-	-
	Fix$_{S^1}$	28	111	37.3	27	71	23.6	17	84	28.0	9	246	94.7	7	1014	418.9	0	-	-	2	315	127.0
	Fix$_{S^2}$	24	216	78.1	26	106	36.2	13	87	29.1	19	238	84.2	9	834	331.0	1	193	85.4	0	-	-
	ALL	30	382	124.0	30	145	47.0	29	269	88.2	28	464	161.4	0	-	-	29	988	343.5	19	1143	428.0
	MAB	30	744	251.3	30	57	28.1	30	127	49.5	30	473	164.6	16	2000	741.5	28	1603	604.6	14	1849	726.9
ψ^2_{AT}	CE	28	216	75.6	13	90	31.4	3	78	27.5	30	228	82.3	29	527	193.9	10	620	263.9	0	-	-
	LM	20	402	157.2	9	95	35.0	3	93	35.4	28	224	86.8	29	454	166.7	4	629	272.9	0	-	-
	Fix$_{S^1}$	29	93	32.3	18	109	38.6	21	97	33.9	20	176	66.5	16	917	390.9	7	563	232.0	8	463	200.1
	Fix$_{S^2}$	30	137	46.1	22	76	26.0	13	100	33.9	23	261	93.5	24	689	269.3	4	191	83.9	4	747	290.9
	ALL	30	410	132.1	30	359	117.8	28	308	101.3	24	755	273.0	0	-	-	27	885	306.3	21	1368	500.5
	MAB	30	480	160.5	30	119	47.0	30	228	82.0	30	497	172.1	14	2067	781.6	28	1578	601.9	27	1787	681.7
ψ^3_{AT}	CE	10	211	82.8	23	179	64.9	11	168	61.7	18	372	140.7	15	778	310.9	0	-	-	0	-	-
	LM	1	439	177.4	26	131	48.3	11	167	64.5	26	397	155.2	24	741	292.1	5	530	241.4	6	874	366.7
	Fix$_{S^1}$	30	110	37.2	20	79	27.3	16	78	26.8	29	219	78.6	23	852	337.8	16	489	193.6	13	748	303.6
	Fix$_{S^2}$	29	127	43.1	22	93	32.2	13	100	33.9	27	215	74.3	28	675	257.5	9	537	213.6	4	675	283.0
	ALL	30	540	177.0	27	390	132.2	25	706	240.9	30	431	149.2	2	1432	676.4	12	1309	486.6	3	1865	699.1
	MAB	30	736	249.3	30	114	45.3	30	246	88.0	30	347	122.3	16	2087	767.3	24	1710	634.2	15	1830	708.0
ψ^4_{AT}	CE	17	1105	460.7	23	89	31.4	14	70	24.2	30	114	40.2	29	448	162.8	2	1433	571.5	4	1170	487.7
	LM	14	1154	480.5	21	112	41.9	14	87	31.9	30	120	44.3	30	478	181.3	1	1055	469.6	1	1563	610.5
	Fix$_{S^1}$	14	543	208.0	18	92	30.8	10	127	43.1	28	108	36.0	26	504	180.8	5	228	89.3	4	411	165.9
	Fix$_{S^2}$	21	376	141.9	21	89	29.7	19	78	25.8	30	115	38.4	28	414	149.1	21	187	65.8	17	206	77.4
	ALL	30	795	273.7	21	263	92.3	16	181	63.6	30	191	64.2	27	1104	402.3	22	828	294.9	21	1437	401.2
	MAB	27	583	203.5	30	160	61.8	29	258	95.4	29	164	62.7	29	634	214.1	13	636	242.4	16	592	199.7
ψ^5_{AT}	CE	14	272	106.1	12	167	61.0	4	437	165.4	23	632	258.2	14	919	376.7	9	1093	490.2	0	-	-
	LM	12	706	276.6	0	-	-	0	-	-	28	538	217.1	15	947	402.3	0	-	-	0	-	-
	Fix$_{S^1}$	30	90	30.2	28	66	23.0	20	63	21.5	30	131	45.6	28	973	383.0	13	526	218.2	4	1186	463.6
	Fix$_{S^2}$	25	88	30.2	27	57	19.1	28	69	23.1	30	187	63.4	21	804	326.2	26	151	52.0	28	313	111.8
	ALL	30	435	140.6	23	189	80.3	30	321	106.2	26	318	110.5	6	1565	646.0	30	966	330.7	23	1225	441.1
	MAB	30	503	168.8	30	69	30.5	30	80	35.3	30	260	93.4	28	1351	497.0	26	745	271.8	21	1122	414.8

		AT8			AT9			AT10			AT11			AT12			AT13			AT14		
		SR	#sim	time	SR	#sim	time	SR	#sim	time	SR	#sim	time	SR	#sim	time	SR	#sim	time	SR	#sim	time
ψ^1_{AT}	CE	5	865	371.2	4	1631	655.1	18	541	215.8	7	679	291.2	9	559	229.3	6	325	118.5	16	281	101.5
	LM	3	201	77.5	9	439	170.3	3	747	310.1	2	1688	705.2	2	1519	617.6	9	236	88.2	8	289	109.2
	Fix$_{S^1}$	22	208	74.7	20	226	87.1	29	172	61.7	26	265	96.6	19	423	161.3	25	157	55.8	30	70	23.1
	Fix$_{S^2}$	22	198	70.6	17	342	132.2	28	332	121.2	19	245	86.7	14	316	120.9	22	246	88.2	27	74	25.3
	ALL	27	553	184.6	22	658	232.4	30	454	150.3	28	629	224.5	14	1176	483.2	28	995	351.8	30	281	95.0
	MAB	30	493	177.4	30	997	355.0	30	271	96.5	30	368	126.9	26	1536	570.3	30	1085	398.6	30	207	75.3
ψ^2_{AT}	CE	13	715	289.9	9	909	359.6	27	513	203.4	25	577	229.4	20	878	352.2	10	234	84.2	17	115	39.7
	LM	8	208	86.8	13	255	96.1	25	566	231.0	22	601	250.0	23	575	227.5	9	215	76.3	16	165	60.5
	Fix$_{S^1}$	25	87	30.7	19	111	40.6	29	116	40.3	26	159	57.3	22	380	146.7	27	142	50.4	30	72	25.0
	Fix$_{S^2}$	24	100	33.7	22	112	38.6	27	162	56.1	23	192	69.7	19	350	132.8	26	130	45.4	30	87	29.9
	ALL	29	470	157.6	24	564	197.5	30	444	146.7	28	666	237.0	5	1097	420.9	27	704	244.0	30	400	133.5
	MAB	30	511	180.8	30	704	248.1	30	220	80.5	30	376	133.0	30	1608	598.3	30	942	340.3	30	228	84.0
ψ^3_{AT}	CE	1	507	216.9	0	-	-	22	470	184.2	19	719	293.5	1	574	215.1	5	365	133.9	12	226	81.3
	LM	1	124	56.1	4	658	275.7	23	451	182.4	27	760	295.9	1	1883	831.8	11	412	153.0	14	238	85.1
	Fix$_{S^1}$	20	117	41.1	22	140	49.8	29	212	75.2	27	522	196.4	11	538	213.5	19	178	62.5	26	91	31.2
	Fix$_{S^2}$	19	138	48.2	14	150	52.9	28	275	98.1	26	491	182.6	6	394	161.0	18	177	63.7	28	98	34.0
	ALL	24	502	174.8	15	672	253.4	30	623	209.8	18	873	331.0	1	565	261.2	21	778	285.5	28	377	125.8
	MAB	30	866	315.7	30	851	299.0	30	394	139.9	29	851	296.4	8	1991	742.8	30	762	273.9	30	255	92.0
ψ^4_{AT}	CE	0	-	-	0	-	-	30	106	36.7	29	216	77.0	28	717	274.3	2	361	127.9	5	223	78.5
	LM	0	-	-	0	-	-	30	132	48.5	28	148	76.7	26	758	297.2	7	482	179.9	3	150	53.7
	Fix$_{S^1}$	14	155	52.8	15	156	53.8	30	97	31.5	30	198	67.4	25	814	311.8	8	97	32.3	21	73	24.3
	Fix$_{S^2}$	20	187	66.2	12	130	46.9	30	126	41.7	29	172	57.8	29	553	205.7	11	195	67.8	17	66	22.1
	ALL	13	379	146.1	10	572	172.5	30	180	61.0	30	288	97.9	19	1180	382.3	10	759	183.0	24	462	72.5
	MAB	22	441	164.7	24	514	181.5	30	174	67.4	30	255	94.7	28	862	305.2	18	352	131.1	26	155	56.3
ψ^5_{AT}	CE	11	1209	514.2	7	1050	427.0	26	423	165.8	14	530	222.9	14	617	243.3	5	401	149.0	5	343	124.8
	LM	5	413	173.1	4	405	164.6	20	472	199.8	16	587	253.7	4	873	376.0	6	454	169.6	8	317	116.1
	Fix$_{S^1}$	23	89	30.5	18	131	45.4	30	116	39.9	30	100	34.0	30	251	88.8	7	119	41.7	16	95	32.8
	Fix$_{S^2}$	15	91	30.5	13	130	44.4	30	177	61.1	18	248	92.4	13	400	154.4	13	86	30.1	22	93	32.2
	ALL	21	495	169.3	14	455	168.6	30	346	114.5	26	826	300.2	5	953	388.0	25	694	237.2	29	511	171.7
	MAB	30	571	206.9	30	687	242.1	30	228	82.8	30	291	104.4	28	1045	376.7	28	605	217.8	30	302	108.2

(b) Abstract Fuel Control

		AFC1			AFC2		
		SR	#sim	time	SR	#sim	time
ψ^1_{AFC}	CE	3	1181	771.4	10	438	282.7
	LM	7	901	591.2	10	452	295.4
	Fix$_{S^1}$	4	604	401.3	14	399	257.0
	Fix$_{S^2}$	10	787	516.6	11	215	140.5
	ALL	0	-	-	23	598	386.5
	MAB	0	-	-	4	975	604.0
ψ^2_{AFC}	CE	0	-	-	8	520	343.3
	LM	5	655	427.9	13	642	418.4
	Fix$_{S^1}$	7	492	316.8	14	543	358.2
	Fix$_{S^2}$	3	1028	702.4	11	587	380.3
	ALL	6	840	546.7	9	339	239.6
	MAB	5	550	355.8	13	637	412.1

(c) Neural Network controller

		NN1			NN2		
		SR	#sim	time	SR	#sim	time
ψ^1_{NN}	CE	23	591	271.6	18	723	291.2
	LM	28	407	172.0	20	617	202.7
	Fix$_{S^1}$	30	303	121.7	26	461	192.3
	Fix$_{S^2}$	26	301	118.2	24	583	239.7
	ALL	28	549	211.4	20	556	226.9
	MAB	30	501	197.4	30	737	283.8
ψ^2_{NN}	CE	25	507	217.0	16	826	361.0
	LM	20	478	285.1	12	790	354.9
	Fix$_{S^1}$	26	379	152.6	22	459	192.5
	Fix$_{S^2}$	19	403	167.4	16	463	195.0
	ALL	27	543	218.9	18	716	333.1
	MAB	27	542	204.2	24	763	297.5

(d) Free Floating Robot

		FFR1			FFR2		
		SR	#sim	time	SR	#sim	time
ψ^1_{FFR}	CE	1	1492	477.4	1	1427	548.7
	LM	3	1456	346.1	0	-	-
	Fix$_{S^1}$	30	338	65.1	20	1467	398.6
	Fix$_{S^2}$	30	370	67.9	18	1325	370.1
	ALL	4	1751	460.7	0	-	-
	MAB	30	1034	226.5	3	920	316.9

Table 5.4: Wilcoxon signed-rank test between two considered approaches A_1 and A_2. Legend: (*nh* = null hypothesis that there is no difference between A_1 and A_2)
✓: *nh* is rejected; A_1 is better than A_2 by the alternative hypothesis (*ah*).
≡: *nh* is not rejected.
✗: *nh* is rejected; A_2 is better than A_1 by the alternative hypothesis (*ah*).

		A_2					
		CE	LM	$\texttt{Fix}_{S^1}$	$\texttt{Fix}_{S^2}$	ALL	MAB
A_1	CE	-	≡	✗	✗	✗	✗
	LM	≡	-	✗	✗	✗	✗
	$\texttt{Fix}_{S^1}$	✓	✓	-	≡	≡	✗
	$\texttt{Fix}_{S^2}$	✓	✓	≡	-	✗	✗
	ALL	✓	✓	≡	✓	-	✗
	MAB	✓	✓	✓	✓	✓	-

execution time. For each experiment, we have computed the average simulation time (*time*/*#sim*) for all the techniques (not reported for the sake of space); note that, for the proposed approaches, such value also includes the time required by the transformation. We observed that there is no significant difference between the approaches, meaning that the computational cost of the transformation is negligible.

RQ2 *How do the three proposed approaches compare each other in terms of SR?*

We are here interested in assessing which is the best approach (among the three proposed ones) in terms of SR (in the given time budget). From Table 5.3, we notice (as already observed in RQ1) that in very few cases the `Fix` approach may be not effective: this shows that, in some cases, choosing the *wrong* priority can affect the performance; a more detailed analysis will be given in RQ3. In one case, we observe that also the ALL method is not effective: although this is an exception considering all the other results of ALL, it is a signal that such exhaustive approach may be ineffective, in particular when there are many priorities to consider.

Observing the statistical tests in Table 5.4, we can draw more definitive conclusions. Selecting one particular priority (`Fix`) does not make any difference: $\texttt{Fix}_{S^1}$ and $\texttt{Fix}_{S^2}$ are statistically equivalent. Considering all the priorities (ALL) is sometimes better than considering only one (ALL is better than $\texttt{Fix}_{S^2}$), but sometimes it is equivalent (ALL is equivalent to $\texttt{Fix}_{S^1}$). This means that none of the two techniques, `Fix` and ALL, overcomes the other, because they highly depend on the considered

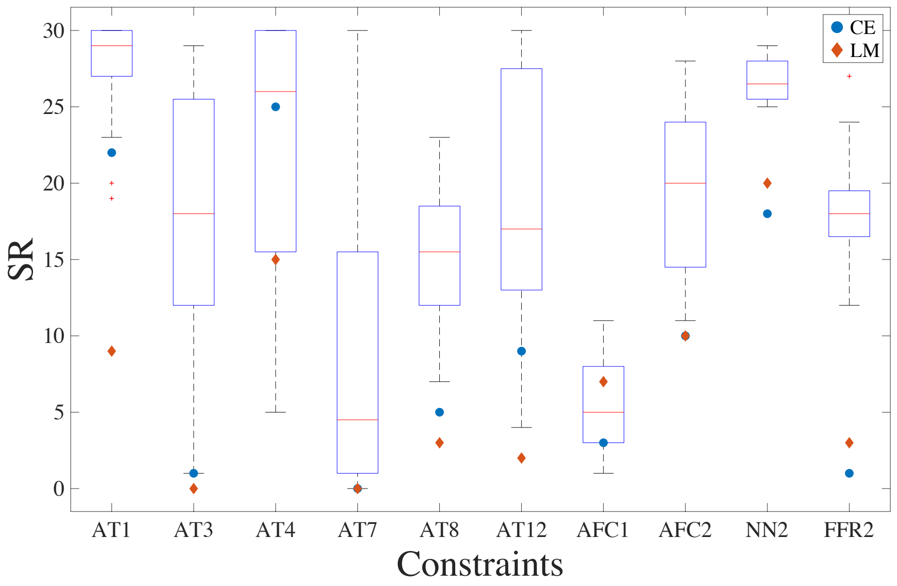


Figure 5.5: Influence of selected priority S in the Fixed-Priority. (ψ^1_{AT} is used for AT specifications, ψ^1_{AFC} for AFC specifications, and ψ^1_{NN} for the NN specification, and ψ^1_{FFR} for the FFR specification.)

specification and constraint. On the other hand, we observe that the MAB approach outperforms all the other proposed approaches: this shows that the Multi-Armed Bandit efficiently learns the priority that must be used because it provides the lower robustness.

RQ3 *How does the selection of priority S in the Fixed-Priority approach affect SR?*

As observed in RQ2, the priority chosen by the Fixed-Priority approach can influence the falsification ability. In this RQ, we want to assess such influence. We selected 8 specifications among those used previously and, for each of these, we also selected one constraint. For each specification, we run the Fixed-Priority approach using all the non-equivalent priorities in set *Prior* (see §5.5.2). For each fixed priority, the falsification is run 30 times. Fig. 5.5 shows how the success rate SR changes by varying priorities. We observe that, for some specifications (AT1, AFC1, NN2, FFR2), the variability is low, meaning that the influence of the selected priority is small. Some of these specifications are simple, such as AT1 and NN2, as we note that SR is high. AFC1 and FFR2 are not simple but they still exhibit small variance. For example in FFR2, this is because the four different inputs (i.e., the four robot boosters) play similar roles in the system, and thus the priority over them does not matter.

On the other hand, for some specifications (e.g., AT3, AT4, AT7 and AT12), the variance is very high, going from SR of (almost) 0 to (almost) 30. This means that, in these cases, some priorities perform much better than others, because they tend to map points towards parts of the input space having low (possibly negative) robustness. In these specifications, the MAB-Priority approach is effective (see RQ2 and Table 5.3): in AT3, AT4 and AT12 with ψ^1_{AT}, MAB-Priority approaches achieve the highest success rate; in AT7 with ψ^1_{AT}, it performs better than most of other approaches.

We also mark the performance of CE and LM methods in Fig. 5.5. We observe that, in almost all the cases, these two baseline methods do not perform as well as the median performance of the Fixed-Priority method; and in many cases, they perform even worse than the worst cases of the Fixed-Priority method. This observation strengthens our conclusion in RQ1 that the proposed approach performs generally better than the baseline approaches.

6 Conclusions

In this work, a general hierarchical optimization framework that can be instantiated for multiple purposes is presented. The framework consists of two layers: a top layer that selects a subset of the whole problem as the next step to proceed, and a bottom layer that performs numerical optimization and returns feedback to the top layer. The decision made by the top layer is dependent on the information provided by the bottom layer; the numerical optimization performed by the bottom layer is bounded by the conditions suggested by the top layer. In this way, the two layers collaborate with each other and work together to solve the given problem.

The framework is instantiated to three techniques, and each of them solves one specific problem that exists in the falsification workflow. Concretely, they are:

- A Monte Carlo Tree Search (MCTS)-based technique for balancing exploration and exploitation during search. Hill-climbing optimization used in the existing falsification technique is a greedy search strategy, and thus can easily fall into the local optimum. In our method, we discretize the search space and structure the sub-spaces as a search tree, and then we perform search in a hierarchical manner: on the top layer, MCTS decides the sub-spaces (identified by branches) that should be further looked into based on the rewards computed by the bottom layer; on the bottom layer, hill-climbing optimization is run in a local space suggested by the top layer to give feedback or find a concrete solution. These two layers work closely together to improve the effectiveness and efficiency of the search.

- Application of Multi-Armed Bandit (MAB) model to handling safety properties with Boolean connectives. The existing definition of STL robust semantics for Boolean connectives superposes the robustness values from different signals. As signals may have different scales, the global robustness will be biased, and this can affect the falsification performance. We propose a novel technique that treats different sub-formulas as different bandit machines, and applies the MAB algorithms (UCB1 and ε-Greedy) to govern the hill-climbing processes running on different machines. We then define *hill-climbing gain* rewards to embody the running status of each machine. It forms such a framework: the MAB algorithms on the top layer select one of the machines according to the rewards of them; the selected machine on the bottom layer runs hill-climbing optimization and returns the running status information for computing rewards. These two layers work together to handle the problem of falsifying safety properties with Boolean connectives.

- Handling input constraints via search space transformation enhanced by Multi-Armed Bandit (MAB) model. The existing falsification framework ignores logical constraints on input signals, and thus produces falsifying inputs that are meaningless. We propose a search space transformation approach, in which the search is allowed to sample in an unconstrained search space, guided by fitness coming from the constrained input space. This is implemented by a surjection that maps points from the unconstrained space to the constrained space. Once a negative fitness is observed, we return the point in the constrained space (as the counterexample for falsification), so that its satisfaction to the input constraints is guaranteed. The performance of this approach is subject to a hyperparameter, namely, a total order over the dimensions of the search space. In order to achieve the best performance, we introduce the MAB model in this

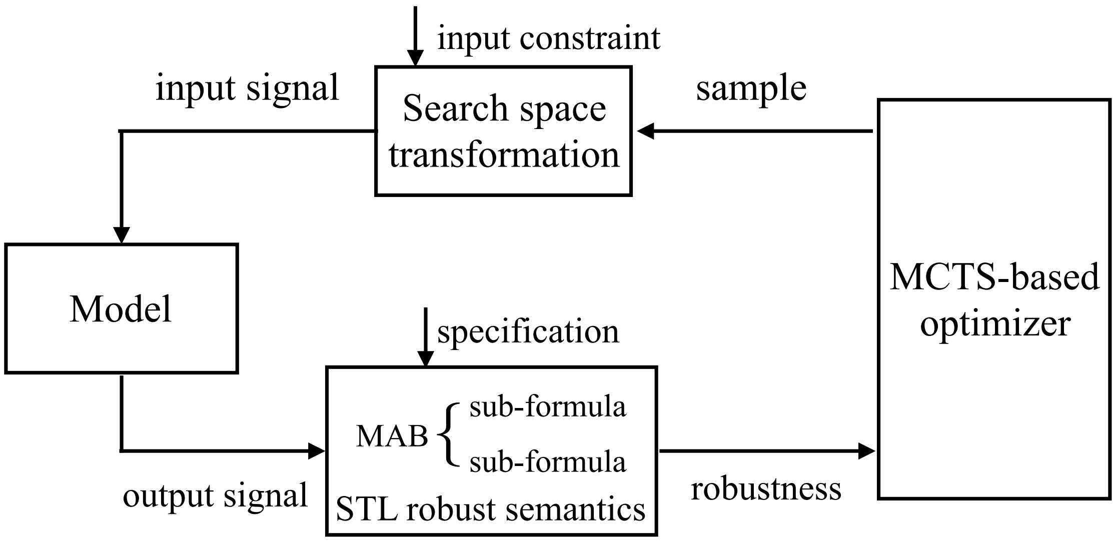


Figure 6.1: A potential framework integrating all the technique

context again, and construct the hierarchical framework: the MAB algorithm on the top layer selects the best order and sends it to the bottom layer; the search space transformation-based optimization on the bottom layer runs following that order and gives feedback to the top layer. Again, they solve the problem through collaboration.

These techniques exemplify the strength of the hierarchical optimization framework. Together, they enhanced the existing falsification framework, and pushed the falsification technique one big step forward to the practical use. In the next section, we discuss on the potential of integrating these different techniques into a single tool, and compare it with the existing state of the art.

6.1 Discussion: Integration of the Techniques

As the proposed techniques work in different phases of the falsification workflow (Fig. 1.6), it is possible to integrate all of them into a single framework, which is depicted in Fig. 6.1. In the presence of input constraints, the framework starts with a sample and maps it to an input signal that satisfies the constraint via search space transformation; then the calculation of robustness of the output signal will be tailored if the specification contains Boolean connectives—it selects one sub-formula rather than the whole formula at each loop according to the MAB algorithm; lastly, the optimizer is replaced by the combination of hill-climbing optimization and MCTS to enhance the search ability.

The framework shown in Fig. 6.1 improves the classic falsification framework in the following aspects: firstly, it strengthens the effectiveness so that the new framework is not vulnerable to problems such as local optima or scale problem; secondly, it provides new functionalities such as support for input constraints.

Moreover, the framework in Fig. 6.1 also benefits the practitioners in the sense that they do not have to strive to select one specific technique for their applications—they can directly apply this framework in any case without additional cost. If their application has no input constraint to deal with, then the sample and the input signal in Fig. 6.1 just correspond; even if it is unknown in advance whether their application suffers from the scale problem, applying the MAB technique for Boolean connectives will not lead to a worse performance as shown in Chapter 4; if the search is so simple that hill climbing suffices, using our MCTS-based technique will introduce a little overhead for search space exploration but that is acceptable. Therefore, our work offers practitioners a more powerful falsification tool than the state of the art.

6.2 Future works

An ongoing work now is to investigate how much confidence can be given by the falsification technique. Especially, in the case that no falsifying input has been found by falsification, how likely it is that no such input exists in the search space. As the search space we consider here is a continuous domain, in general this problem is undecidable. However, some clues are still given by the sampling. For example, if the sampling is biased to exploitation and thus only searches in a local area, then it is not known if the unexplored areas contain the falsifying input. In such a case, the confidence level on non-existence of falsifying inputs should be rather low, even if the algorithm does not manage to find one.

As mid-term future works, related topics about debugging hybrid systems and automatic fault repair will be investigated. Falsification technique in nature is a testing-based approach for the aim of knowing the existence of unexpected behaviors. In the case there exist such behaviors, it is important to conduct fault localization to find the root cause. One instance of the problem is that if the specification is a complicated one, it is required to firstly address the sub-formula that is violated. A more important problem is to find the system components that cause the unexpected behavior. The aim of fault localization is to repair it, and it is preferable if it can be done automatically. This is a direction of great significance, but it has not been explored much yet.

The long-term future work is to extend the philosophy of the hierarchical framework to other contexts. One possibility is to develop new techniques that instantiate the framework. For example, in a setting of white-box information of a Simulink model aware, we can consider combining symbolic execution and numerical analysis in a hierarchical manner for falsification: on the top layer, we perform symbolic execution to obtain the constraints under which an execution path can be visited; on the bottom layer, we run constrained optimization to assess the quality of the corresponding path. We can use techniques such as UCB1 to balance the exploration to different execution paths and the exploitation to some specific paths. Our goal is to find the path where falsifying inputs exist effectively and efficiently.

It is also possible to extend the hierarchical optimization framework to totally different domains. Nowadays, as the scales of many problems are much larger than ever before, techniques based on artificial intelligence or optimization are more and more widely applied. For instance, in deep learning, a critical problem is to tune a large set of parameters using optimization. In general, decomposing the problems and hierarchizing the methodologies can be a generic strategy. Therefore, we believe that our hierarchical framework can also make a success in other contexts.

Bibliography

[1] Thomas A Henzinger. The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*, pages 265–292. Springer, 2000.

[2] Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model checking*. MIT press, 2018.

[3] Shankara Narayanan Krishna and Ashutosh Trivedi. Hybrid automata for formal modeling and verification of cyber-physical systems. *CoRR*, abs/1503.04928, 2015.

[4] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[5] Kim G Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International journal on software tools for technology transfer*, 1(1-2):134–152, 1997.

[6] Conrado Daws, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. The tool kronos. In *International Hybrid Systems Workshop*, pages 208–219. Springer, 1995.

[7] Étienne André, Thomas Chatain, Laurent Fribourg, and Emmanuelle Encrenaz. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(05):819–836, 2009.

[8] André Platzer and Jan-David Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In *International Joint Conference on Automated Reasoning*, pages 171–178. Springer, 2008.

[9] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. Keymaera x: An axiomatic tactical theorem prover for hybrid systems. In *International Conference on Automated Deduction*, pages 527–538. Springer, 2015.

[10] André Platzer. Differential dynamic logic for verifying parametric hybrid systems. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 216–232. Springer, 2007.

[11] André Platzer. Differential dynamic logic for hybrid systems. *Journal of Automated Reasoning*, 41(2):143–189, 2008.

[12] Kohei Suenaga and Ichiro Hasuo. Programming with infinitesimals: A while-language for hybrid system modeling. In *International Colloquium on Automata, Languages, and Programming*, pages 392–403. Springer, 2011.

[13] Ichiro Hasuo and Kohei Suenaga. Exercises in nonstandard static analysis of hybrid systems. In *International Conference on Computer Aided Verification*, pages 462–478. Springer, 2012.

[14] Kohei Suenaga, Hiroyoshi Sekine, and Ichiro Hasuo. Hyperstream processing systems: nonstandard modeling of continuous-time signals. *ACM SIGPLAN Notices*, 48(1):417–430, 2013.

[15] 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. IEEE, 2015.

[16] Rodrigo Queiroz, Thorsten Berger, and Krzysztof Czarnecki. Geoscenario: An open dsl for autonomous driving scenario representation. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 287–294. IEEE, 2019.

[17] Wasif Afzal, Richard Torkar, and Robert Feldt. A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, 51(6):957–976, 2009.

[18] Phil McMinn. Search-based software testing: Past, present and future. In *Fourth IEEE International Conference on Software Testing, Verification and Validation, ICST 2012, Berlin, Germany, 21-25 March, 2011, Workshop Proceedings*, pages 153–163. IEEE Computer Society, 2011.

[19] Yashwanth Annapureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-Taliro: A tool for temporal logic falsification for hybrid systems. In *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems: Part of the Joint European Conferences on Theory and Practice of Software*, TACAS' 11/ETAPS' 11, pages 254–257, Berlin, Heidelberg, 2011. Springer-Verlag.

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

[21] Jan Kuřátko and Stefan Ratschan. Combined global and local search for the falsification of hybrid systems. In *Formal Modeling and Analysis of Timed Systems*, pages 146–160, Cham, 2014. Springer International Publishing.

[22] Alexandre Donzé. Breach, A toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification, 22nd Int. Conf., CAV 2010*, volume 6174 of *LNCS*, pages 167–170. Springer, 2010.

[23] 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 *NASA Formal Methods*, pages 127–142, Cham, 2015. Springer International Publishing.

[24] Aditya Zutshi, Jyotirmoy V. Deshmukh, Sriram Sankaranarayanan, and James Kapinski. Multiple shooting, cegar-based falsification for hybrid systems. In *2014 International Conference on Embedded Software, EMSOFT 2014, New Delhi, India, October 12-17, 2014*, pages 5:1–5:10. ACM, 2014.

[25] Simone Silvetti, Alberto Policriti, and Luca Bortolussi. An active learning approach to the falsification of black box cyber-physical systems. In *Integrated Formal Methods*, pages 3–17, Cham, 2017. Springer International Publishing.

[26] Takumi Akazaki, Shuang Liu, Yoriyuki Yamagata, Yihai Duan, and Jianye Hao. Falsification of cyber-physical systems using deep reinforcement learning. In *Formal Methods - 22nd International Symposium, FM 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 15-17, 2018, Proceedings*, pages 456–465, 2018.

[27] Gidon Ernst, Sean Sedwards, Zhenya Zhang, and Ichiro Hasuo. Fast falsification of hybrid systems using probabilistically adaptive input. In *Quantitative Evaluation of Systems*, pages 165–181, Cham, 2019. Springer International Publishing.

[28] Truong Nghiem, Sriram Sankaranarayanan, Georgios Fainekos, Franjo Ivancić, Aarti Gupta, and George J. Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *Proc. of the 13th ACM Int. Conf. on Hybrid Systems: Computation and Control*, HSCC '10, pages 211–220, NY, USA, 2010. ACM.

[29] Houssam Abbas, Bardh Hoxha, Georgios Fainekos, Jyotirmoy V Deshmukh, James Kapinski, and Koichi Ueda. Conformance testing as falsification for cyber-physical systems. *arXiv preprint arXiv:1401.5200*, 2014.

[30] Aditya Zutshi, Sriram Sankaranarayanan, Jyotirmoy V Deshmukh, and James Kapinski. A trajectory splicing approach to concretizing counterexamples for hybrid systems. In *52nd IEEE Conference on Decision and Control*, pages 3918–3925. IEEE, 2013.

[31] 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.

[32] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *Formal Modeling and Analysis of Timed Systems - 8th Int. Conf., FORMATS 2010*, volume 6246 of *LNCS*, pages 92–106. Springer, 2010.

[33] Takumi Akazaki and Ichiro Hasuo. Time robustness in MTL and expressivity in hybrid system falsification. In *Computer Aided Verification*, pages 356–374, Cham, 2015. Springer International Publishing.

[34] Pavithra Prabhakar, Ratan Lal, and James Kapinski. Automatic trace generation for signal temporal logic. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 208–217. IEEE, 2018.

[35] Alexandre Donzé, Thomas Ferrere, and Oded Maler. Efficient robust monitoring for stl. In *International Conference on Computer Aided Verification*, pages 264–279. Springer, 2013.

[36] Hsi-Ming Ho, Joël Ouaknine, and James Worrell. Online monitoring of metric temporal logic. In *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, pages 178–192, 2014.

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

[38] Bardh Hoxha, Adel Dokhanchi, and Georgios E. Fainekos. Mining parametric temporal logic properties in model-based design for cyber-physical systems. *STTT*, 20(1):79–93, 2018.

[39] Oded Maler, Dejan Nickovic, and Amir Pnueli. From mitl to timed automata. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 274–289. Springer, 2006.

[40] Ezio Bartocci, Roderick Bloem, Dejan Nickovic, and Franz Roeck. A counting semantics for monitoring ltl specifications over finite traces. In *International Conference on Computer Aided Verification*, pages 547–564. Springer, 2018.

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

[42] 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, Nov 2018.

[43] Adel Dokhanchi, Shakiba Yaghoubi, Bardh Hoxha, Georgios Fainekos, Gidon Ernst, Zhenya Zhang, Paolo Arcaini, Ichiro Hasuo, and Sean Sedwards. ARCH-COMP18 category report: Results on the falsification benchmarks. In *ARCH18. 5th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 54 of *EPiC Series in Computing*, pages 104–109. EasyChair, 2018.

[44] 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 *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 61 of *EPiC Series in Computing*, pages 129–140. EasyChair, 2019.

[45] Cumhur Erkan Tuncali, Theodore P Pavlic, and Georgios Fainekos. Utilizing s-taliro as an automatic test generation framework for autonomous vehicles. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1470–1475. IEEE, 2016.

[46] Cumhur Erkan Tuncali and Georgios Fainekos. Rapidly-exploring random trees for testing automated vehicles. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 661–666. IEEE, 2019.

[47] Tommaso Dreossi, Daniel J Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A Seshia. Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems. In *International Conference on Computer Aided Verification*, pages 432–442. Springer, 2019.

[48] Ritchie Lee, Mykel J Kochenderfer, Ole J Mengshoel, Guillaume P Brat, and Michael P Owen. Adaptive stress testing of airborne collision avoidance systems. In *Digital Avionics Systems Conference, 2015 IEEE/AIAA 34th*, pages 6C2–1. IEEE, 2015.

[49] Anthony Corso, Peter Du, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Adaptive stress testing with reward augmentation for autonomous vehicle

validatio. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 163–168. IEEE, 2019.

[50] Mark Koren and Mykel J Kochenderfer. Efficient autonomy validation in simulation with adaptive stress testing. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 4178–4183. IEEE, 2019.

[51] Vahid Behzadan and Arslan Munir. Adversarial reinforcement learning framework for benchmarking collision avoidance mechanisms in autonomous vehicles. *IEEE Intelligent Transportation Systems Magazine*, 2019.

[52] Tommaso Dreossi, Alexandre Donzé, and Sanjit A. Seshia. Compositional falsification of cyber-physical systems with machine learning components. In *NASA Formal Methods*, pages 357–372, Cham, 2017. Springer International Publishing.

[53] Shakiba Yaghoubi and Georgios Fainekos. Gray-box adversarial testing for control systems with machine learning components. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 179–184, 2019.

[54] Cumhur Erkan Tuncali, Georgios Fainekos, Hisahiro Ito, and James Kapinski. Simulation-based adversarial test generation for autonomous vehicles with machine learning components. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1555–1562. IEEE, 2018.

[55] Cumhur Erkan Tuncali, Georgios Fainekos, Danil Prokhorov, Hisahiro Ito, and James Kapinski. Requirements-driven test generation for autonomous vehicles with machine learning components. *IEEE Transactions on Intelligent Vehicles*, 2019.

[56] Xiaoqing Jin, Jyotirmoy V. Deshmukh, James Kapinski, Koichi Ueda, and Ken Butts. Powertrain control verification benchmark. In *Proc. of the 17th Int. Conf. on Hybrid Systems: Computation and Control*, HSCC '14, pages 253–262, NY, USA, 2014. ACM.

[57] Christoph Gladisch, Thomas Heinz, Christian Heinzemann, Jens Oehlerking, Anne von Vietinghoff, and Tim Pfitzer. Experience paper: Search-based testing in automated driving control applications. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*, pages 26–37, 2019.

[58] Rémi Delmas, Thomas Loquen, Josep Boada-Bauxell, and Mathieu Carton. An evaluation of monte-carlo tree search for property falsification on hybrid flight control laws. In *Numerical Software Verification - 12th International Workshop, NSV@CAV 2019, New York City, NY, USA, July 13-14, 2019, Proceedings*, pages 45–59, 2019.

[59] Martin Pincus. Letter to the editor—a monte carlo method for the approximate solution of certain types of constrained optimization problems. *Operations Research*, 18(6):1225–1228, 1970.

[60] Houssam Abbas, Georgios Fainekos, Sriram Sankaranarayanan, Franjo Ivančić, and Aarti Gupta. Probabilistic temporal logic falsification of cyber-physical systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 12(2s):1–30, 2013.

[61] Arend Aerts, Bryan Tong Minh, Mohammad Reza Mousavi, and Michel 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. IEEE, 2018.

[62] Yashwanth Singh Rahul Annapureddy and Georgios E Fainekos. Ant colonies for temporal logic falsification of hybrid systems. In *IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society*, pages 91–96. IEEE, 2010.

[63] Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1(2):127–190, 1999.

[64] 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, pages 125–134, 2012.

[65] Logan Mathesen, Shakiba Yaghoubi, Giulia Pedrielli, and Georgios Fainekos. Falsification of cyber-physical systems with robustness uncertainty quantification through stochastic optimization with adaptive restart. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 991–997. IEEE, 2019.

[66] Shakiba Yaghoubi and Georgios Fainekos. Worst-case satisfaction of stl specifications using feedforward neural network controllers: a lagrange multipliers approach. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–20, 2019.

[67] Adel Dokhanchi, Aditya Zutshi, Rahul T. Sriniva, Sriram Sankaranarayanan, and Georgios E. Fainekos. Requirements driven falsification with coverage metrics. In *2015 International Conference on Embedded Software, EMSOFT 2015, Amsterdam, Netherlands, October 4-9, 2015*, pages 31–40. IEEE, 2015.

[68] Arvind Adimoolam, Thao Dang, Alexandre Donzé, James Kapinski, and Xiaoqing Jin. Classification and coverage-based falsification for embedded control systems. In *Computer Aided Verification*, pages 483–503, Cham, 2017. Springer International Publishing.

[69] Jyotirmoy V. Deshmukh, Marko Horvat, Xiaoqing Jin, Rupak Majumdar, and Vinayak S. Prabhu. Testing cyber-physical systems through bayesian optimization. *ACM Trans. Embedded Comput. Syst.*, 16(5):170:1–170:18, 2017.

[70] Takumi Akazaki. Falsification of conditional safety properties for cyber-physical systems with gaussian process regression. In *Runtime Verification - 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings*, volume 10012 of *Lecture Notes in Computer Science*, pages 439–446. Springer, 2016.

[71] Takumi Akazaki, Yoshihiro Kumazawa, and Ichiro Hasuo. Causality-aided falsification. In *Proceedings First Workshop on Formal Verification of Autonomous*

Vehicles, FVAV@iFM 2017, Turin, Italy, 19th September 2017., volume 257 of *EPTCS*, pages 3–18, 2017.

[72] Yasasa Abeysirigoonawardena, Florian Shkurti, and Gregory Dudek. Generating adversarial driving scenarios in high-fidelity simulators. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8271–8277. IEEE, 2019.

[73] Koki Kato, Fuyuki Ishikawa, and Shinichi Honiden. Falsification of cyber-physical systems with reinforcement learning. In *3rd Workshop on Monitoring and Testing of Cyber-Physical Systems, MT@CPSWeek 2018, Porto, Portugal, April 10, 2018*, pages 5–6, 2018.

[74] Mark Koren, Saud Alsaif, Ritchie Lee, and Mykel J Kochenderfer. Adaptive stress testing for autonomous vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–7. IEEE, 2018.

[75] 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 2019, Montreal, QC, Canada, April 16-18, 2019.*, pages 57–66, 2019.

[76] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. Testing vision-based control systems using learnable evolutionary algorithms. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE ’18, pages 1016–1026, New York, NY, USA, 2018. ACM.

[77] Benoît Barbot, Nicolas Basset, and Thao Dang. Generation of signals under temporal constraints for CPS testing. In *NASA Formal Methods*, pages 54–70, Cham, 2019. Springer International Publishing.

[78] Özgür Yeniay. Penalty function methods for constrained optimization with genetic algorithms. *Mathematical and computational Applications*, 10(1):45–56, 2005.

[79] Thomas Back, Frank Hoffmeister, and Hans-Paul Schwefel. A survey of evolution strategies. In *Proceedings of the fourth international conference on genetic algorithms*, volume 2. Morgan Kaufmann Publishers San Mateo, CA, 1991.

[80] Angel Kuri Morales and Carlos Villegas Quezada. A universal eclectic genetic algorithm for constrained optimization. In *Proceedings of the 6th European congress on intelligent techniques and soft computing*, volume 1, pages 518–522, 1998.

[81] S Kazarlis and Vassilios Petridis. Varying fitness functions in genetic algorithms: Studying the rate of increase of the dynamic penalty terms. In *International conference on parallel problem solving from nature*, pages 211–220. Springer, 1998.

[82] Zhenya Zhang, Paolo Arcaini, and Ichiro Hasuo. Constraining counterexamples in hybrid system falsification: Penalty-based approaches. *arXiv preprint arXiv:2001.05107*, 2020.

[83] Zbigniew Michalewicz and Girish Nazhiyath. Genocop iii: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In *Proceedings of 1995 IEEE International Conference on Evolutionary Computation*, volume 2, pages 647–651. IEEE, 1995.

[84] Kalyanmoy Deb. An efficient constraint handling method for genetic algorithms. *Computer methods in applied mechanics and engineering*, 186(2-4):311–338, 2000.

[85] Slawomir Koziel and Zbigniew Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary computation*, 7(1):19–44, 1999.

[86] Hojjat Adeli and Nai-Tsang Cheng. Augmented lagrangian genetic algorithm for structural optimization. *Journal of Aerospace Engineering*, 7(1):104–118, 1994.

[87] Jong-Hwan Kim and Hyun Myung. Evolutionary programming techniques for constrained optimization problems. *IEEE Transactions on evolutionary computation*, 1(2):129–140, 1997.

[88] Zhenya Zhang, Gidon Ernst, Ichiro Hasuo, and Sean Sedwards. Time-staging enhancement of hybrid system falsification. In *3rd Workshop on Monitoring and Testing of Cyber-Physical Systems, MT@CPSWeek 2018, Porto, Portugal, April 10, 2018*, pages 3–4, 2018.

[89] Zhenya Zhang, Ichiro Hasuo, and Paolo Arcaini. Multi-armed bandits for boolean connectives in hybrid system falsification. In *International Conference on Computer Aided Verification*, pages 401–420. Springer, 2019.

[90] Zhenya Zhang, Paolo Arcaini, and Ichiro Hasuo. Hybrid system falsification under (in)equality constraints via search space transformation. *preprint*, 2020.

[91] Marco A. Luersen and Rodolphe Le Riche. Globalized Nelder–Mead method for engineering optimization. *Computers & Structures*, 82(23):2251–2260, 2004.

[92] Anne Auger and Nikolaus Hansen. A restart CMA evolution strategy with increasing population size. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2005*, pages 1769–1776. IEEE, 2005.

[93] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293, Berlin, Heidelberg, 2006. Springer.

[94] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, 2015.

[95] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

[96] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1):1–43, 2012.

[97] Rémi Coulom. Computing "elo ratings" of move patterns in the game of go. *ICGA Journal*, 30(4):198–208, 2007.

[98] Bardh Hoxha, Houssam Abbas, and Georgios E. Fainekos. Benchmarks for temporal logic requirements for automotive systems. In *1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems,*

ARCH@CPSWeek 2014, Berlin, Germany, April 14, 2014 / ARCH@CPSWeek 2015, Seattle, USA, April 13, 2015., pages 25–30, 2014.

[99] Mark Hudson Beale, Martin T Hagan, and Howard B Demuth. Neural network toolbox™ user's guide. *The Mathworks Inc*, 1992.

[100] Claes Wohlin, Per Runeson, Martin Hst, Magnus C. Ohlsson, Bjrn Regnell, and Anders Wessln. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012.

[101] Kuang-Hua Chang. Chapter 19 - multiobjective optimization and advanced topics. In Kuang-Hua Chang, editor, *e-Design*, pages 1105 – 1173. Academic Press, Boston, 2015.

[102] Matthias Ehrgott. *Multicriteria Optimization*. Springer-Verlag, Berlin, Heidelberg, 2005.

[103] Daniele Pinchera, Stefano Perna, and Marco Donald Migliore. A lexicographic approach for multi-objective optimization in antenna array design. *Progress In Electromagnetics Research*, 59:85–102, 2017.

A Omitted Details

A.1 MATLAB Source Code for the Usage Scenario

```
%Initialize the execution environment of Breach.
InitBreach;

%Specify the model name, and interface with Breach
mdl = 'Autotrans_shift';
Br = BreachSimulinkSystem(mdl);

%Specify the type of Signal. Here, piecewise constant signal.
input_gen.type = 'UniStep';
input_gen.cp = 5;
Br.SetInputGen(input_gen);

%Specify the ranges of the signals
```

```
for cpi = 0:input_gen.cp-1
throttle_sig = strcat('throttle_u', num2str(cpi));
brake_sig = strcat('brake_u', num2str(cpi));
Br.SetParamRanges({throttle_sig},[0 100]);
Br.SetParamRanges({brake_sig},[0 325]);
end

%System specification in Signal Temporal Logic (STL)
phi_str = 'alw_[0,30]((gear[t] == 4) => (speed[t] > 35))'
phi = STL_Formula('phi', phi_str);

%Interface the problem
falsif_pb = FalsificationProblem(Br, phi);

%Budget in terms of timeout or the number of simulations
falsif_pb.max_time = 100;
%falsif_pb.max_obj_eval = 100;

%Select an optimization solver
solver_input = 'cmaes';
falsif_pb.setup_solver(solver_input);

%Executing falsification process
falsif_pb.solve();

%Plot input and output signals
%falsif_pb.BrSys.PlotSignals();

%Plot specification satisfaction
%falsif_pb.BrSys.PlotRobustSat(phi, 2)
```

A.2 Raw Experimental Results for Table 4.2

Table A.1: Raw experimental results – `Bbench` (SR: # successes out 30 trials. Time in secs. Δ: percentage difference w.r.t. BREACH)

Spec	Breach		MAB-ϵ-greedy				MAB-UCB			
	SR	time	SR	time	ΔSR	Δtime	SR	time	ΔSR	Δtime
$AT1_1$	25	125	29	77.4	14.8	−47	29	75	14.8	−50
$AT1_2$	22	187.5	30	62.7	30.8	−99.7	30	45.1	30.8	−122.4
$AT1_3$	19	239.5	30	91.5	44.9	−89.4	29	62.5	41.7	−117.2
$AT1_4$	21	202.2	30	87.1	35.3	−79.5	30	57.4	35.3	−111.6
$AT1_5$	14	361.2	24	213.4	52.6	−51.4	28	146.8	66.7	−84.4
$AT2_1$	30	14	30	11.9	0	−16.2	30	17.7	0	23.3
$AT2_2$	26	96.2	30	37.8	14.3	−87.1	30	19	14.3	−134
$AT2_3$	20	216.3	30	41.1	40	−136.1	30	23	40	−161.5
$AT2_4$	14	331.9	30	55.2	72.7	−143	30	31.9	72.7	−164.9
$AT2_5$	11	390.6	30	126.3	92.7	−102.3	27	92.5	84.2	−123.4
$AT3_1$	29	21.9	30	7	3.4	−103.6	30	3.6	3.4	−143.6
$AT3_2$	30	2.3	30	2.5	0	7.1	30	2.5	0	8.3
$AT3_3$	29	22.2	30	2.6	3.4	−158.4	30	3.4	3.4	−146.9
$AT3_4$	30	3	30	2.8	0	−5.8	30	2.9	0	−3.4
$AT3_5$	29	21.7	30	2.8	3.4	−154	30	2.6	3.4	−157.3
$AT4_1$	30	19.5	30	7.8	0	−85.8	30	6.2	0	−103.5
$AT4_2$	29	48.1	30	19.5	3.4	−84.4	30	13.3	3.4	−113.4
$AT4_3$	29	54.4	30	31.7	3.4	−52.8	30	29.2	3.4	−60.3
$AT4_4$	23	160.5	30	17.8	26.4	−160	30	36.2	26.4	−126.4
$AT4_5$	18	265.3	29	45.1	46.8	−141.9	30	26.3	50	−163.9
$AT5_1$	23	203.1	30	35.2	26.4	−140.9	30	37.7	26.4	−137.4
$AT5_2$	16	320.4	26	126.8	47.6	−86.6	30	39.7	60.9	−155.9
$AT5_3$	18	290.6	30	46.6	50	−144.7	26	141.6	36.4	−68.9
$AT5_4$	18	291.1	27	108.5	40	−91.4	28	94.5	43.5	−102
$AT5_5$	15	369.1	29	71.1	63.6	−135.4	27	114	57.1	−105.6
$AT5_6$	10	438.2	30	75.3	100	−141.3	30	68.3	100	−146
$AT5_7$	7	491.4	28	136.8	120	−112.9	26	154.1	115.2	−104.5
$AT5_8$	6	525.9	28	149	129.4	−111.7	29	143.6	131.4	−114.2
$AT6_1$	28	46.4	30	2.3	6.9	−180.8	30	2.9	6.9	−176.5
$AT6_2$	29	30.1	30	4.3	3.4	−149.5	30	4.3	3.4	−150.1
$AT6_3$	25	111.9	30	9.7	18.2	−168.1	30	11.4	18.2	−163
$AT6_4$	27	86.9	24	159	−11.8	58.6	23	164.8	−16	61.8
$AT6_5$	5	509.5	21	300	123.1	−51.8	22	247.3	125.9	−69.3
$AT7_1$	30	12.2	20	283.9	−40	183.5	23	223.3	−26.4	179.2
$AT7_2$	15	314	30	33.6	66.7	−161.4	30	43.2	66.7	−151.6
$AT7_3$	25	111.9	30	10.5	18.2	−165.6	30	11.4	18.2	−163
$AT7_4$	28	51.1	30	2.9	6.9	−178.6	30	5.8	6.9	−159.5
$AT7_5$	30	13.5	30	5	0	−91.5	30	5.5	0	−84.1
$AT7_6$	30	12.6	30	5.5	0	−78.5	30	5.6	0	−76.2
$AT7_7$	28	54.9	30	7.6	6.9	−151.6	30	5.6	6.9	−162.7
$AFC1_1$	30	124.8	28	171	−6.9	31.2	30	98.7	0	−23.4
$AFC1_2$	16	421.5	15	419.2	−6.5	−0.5	23	346.8	35.9	−19.4
$AFC1_3$	11	457.7	8	506.7	−31.6	10.2	15	442	30.8	−3.5
$AFC1_4$	9	498.2	5	568.4	−57.1	13.2	9	502.1	0	0.8
$AFC1_5$	6	565.6	4	564.7	−40	−0.1	5	559.8	−18.2	−1
$AFC2_1$	30	80.7	30	43.2	0	−60.5	30	59.4	0	−30.5
$AFC2_2$	29	128.1	30	100.5	3.4	−24.2	30	123.3	3.4	−3.8
$AFC2_3$	17	436.1	23	326.1	30	−28.9	24	359.3	34.1	−19.3
$AFC2_4$	12	489.9	12	491.9	0	0.4	11	492	−8.7	0.4
$AFC2_5$	2	582.3	5	547.8	85.7	−6.1	5	568.4	85.7	−2.4
$NN1_1$	25	221.2	27	189.5	7.7	−15.4	28	148.2	11.3	−39.5
$NN1_2$	24	212.9	24	212.6	0	−0.1	28	169	15.4	−23
$NN1_3$	19	300.1	18	401.9	−5.4	29	19	308.7	0	2.8
$NN1_4$	17	384.7	18	374.6	5.7	−2.7	21	332.2	21.1	−14.6
$NN1_5$	19	345.5	14	422.8	−30.3	20.1	17	403.3	−11.1	15.4
$NN2_1$	27	66.8	30	11	10.5	−143.5	30	14.6	10.5	−128.1
$NN2_2$	27	70.7	30	17.3	10.5	−121.4	30	23.7	10.5	−99.5
$NN2_3$	28	55.5	30	26	6.9	−72.2	30	27.8	6.9	−66.6
$NN2_4$	27	79.1	30	39.3	10.5	−67.3	30	32.5	10.5	−83.5
$NN2_5$	27	93.4	30	37.8	10.5	−84.7	30	38.2	10.5	−83.8

Table A.2: Raw experimental results – Sbench (SR: # successes out 30 trials. Time in secs. Δ: percentage difference w.r.t. Breach)

Spec	Breach		MAB-ϵ-greedy				MAB-UCB			
	SR	time	SR	time	ΔSR	Δtime	SR	time	ΔSR	Δtime
$AT1_1^{-2}$	30	51.3	30	75.6	0	38.3	30	54.3	0	5.6
$AT1_1^{0}$	25	125	29	77.4	14.8	−47	29	75	14.8	−50
$AT1_1^{1}$	20	221.1	30	49	40	−127.5	28	107.9	33.3	−68.8
$AT1_1^{3}$	23	170	30	82.5	26.4	−69.3	29	55.4	23.1	−101.6
$AT1_2^{-2}$	30	49	29	115.6	−3.4	80.9	29	67.5	−3.4	31.9
$AT1_2^{0}$	22	187.5	30	62.7	30.8	−99.7	30	45.1	30.8	−122.4
$AT1_2^{1}$	21	204.5	30	59.7	35.3	−109.6	29	77.5	32	−90.1
$AT1_2^{3}$	24	164	30	88.8	22.2	−59.5	30	61	22.2	−91.5
$AT1_3^{-2}$	30	42.5	28	144.4	−6.9	109.1	30	62.4	0	38
$AT1_3^{0}$	19	239.5	30	91.5	44.9	−89.4	29	62.5	41.7	−117.2
$AT1_3^{1}$	16	296.2	30	72.3	60.9	−121.5	27	128.7	51.2	−78.8
$AT1_3^{3}$	21	209.8	29	99.9	32	−71	30	93.4	35.3	−76.8
$AT1_4^{-2}$	30	44.5	30	79.8	0	56.8	30	80.8	0	57.9
$AT1_4^{0}$	21	202.2	30	87.1	35.3	−79.5	30	57.4	35.3	−111.6
$AT1_4^{1}$	16	301.7	30	94.6	60.9	−104.5	28	119.5	54.5	−86.5
$AT1_4^{3}$	23	185.1	30	67.5	26.4	−93.1	29	88.3	23.1	−70.8
$AT1_5^{-2}$	30	97.4	28	178.3	−6.9	58.7	28	136.3	−6.9	33.3
$AT1_5^{0}$	14	361.2	24	213.4	52.6	−51.4	28	146.8	66.7	−84.4
$AT1_5^{1}$	4	527.6	25	234.7	144.8	−76.8	29	91.9	151.5	−140.7
$AT1_5^{3}$	8	471.7	30	170.6	115.8	−93.7	29	104.8	113.5	−127.3
$AT5_4^{-2}$	30	61.1	25	139.5	−18.2	78.1	30	48.5	0	−23
$AT5_4^{0}$	18	291.1	28	106.6	43.5	−92.8	28	94.5	43.5	−102
$AT5_4^{1}$	2	566.4	29	70.7	174.2	−155.6	25	150	170.4	−116.2
$AT5_4^{3}$	1	586.4	28	89.3	186.2	−147.1	28	96.2	186.2	−143.6
$AT5_5^{-2}$	30	71.3	27	113.8	−10.5	45.9	29	67.8	−3.4	−5.1
$AT5_5^{0}$	15	369.1	28	98.2	60.5	−115.9	27	114	57.1	−105.6
$AT5_5^{1}$	0	600	27	115.4	200	−135.5	29	83.1	200	−151.4
$AT5_5^{3}$	0	600	30	66.8	200	−159.9	27	113.8	200	−136.2
$AT5_6^{-2}$	29	110.2	29	76.9	0	−35.5	28	103.3	−3.5	−6.5
$AT5_6^{0}$	10	438.2	28	100.5	94.7	−125.4	30	68.3	100	−146
$AT5_6^{1}$	0	600	29	90.8	200	−147.4	27	126.7	200	−130.3
$AT5_6^{3}$	0	600	30	70	200	−158.2	29	80.4	200	−152.7
$AT5_7^{-2}$	30	103.6	28	116.1	−6.9	11.4	30	77.3	0	−29.1
$AT5_7^{0}$	7	491.4	30	80.2	124.3	−143.9	26	154.1	115.2	−104.5
$AT5_7^{1}$	0	600	30	90	200	−147.8	27	134.3	200	−126.8
$AT5_7^{3}$	0	600	27	123.3	200	−131.8	29	108	200	−139
$AT5_8^{-2}$	29	163.7	30	113	3.4	−36.7	30	131.9	3.4	−21.6
$AT5_8^{0}$	6	525.9	28	151.3	129.4	−110.6	29	143.6	131.4	−114.2
$AT5_8^{1}$	0	600	27	184.5	200	−105.9	30	124.2	200	−131.4
$AT5_8^{3}$	0	600	28	163.3	200	−114.4	27	160.9	200	−115.4
$AFC1_1^{0}$	30	124.8	29	115.1	−3.4	−8.1	30	98.7	0	−23.4
$AFC1_1^{1}$	30	99	29	198.1	−3.4	66.7	29	105.7	−3.4	6.5
$AFC1_1^{2}$	12	434.4	28	180.8	80	−82.4	30	73.7	85.7	−142
$AFC1_1^{3}$	12	425.7	30	138	85.7	−102.1	30	77.1	85.7	−138.7
$AFC1_2^{0}$	16	421.5	23	331.7	35.9	−23.8	23	346.8	35.9	−19.4
$AFC1_2^{1}$	25	345.9	12	456.8	−70.3	27.6	27	227.9	7.7	−41.1
$AFC1_2^{2}$	8	497.2	15	446.6	60.9	−10.7	25	320.5	103	−43.2
$AFC1_2^{3}$	5	518.1	16	438.9	104.8	−16.5	21	364	123.1	−34.9
$AFC1_3^{0}$	11	457.7	16	531.3	37	14.9	15	442	30.8	−3.5
$AFC1_3^{1}$	13	479.2	7	514.9	−60	7.2	14	455.5	7.4	−5.1
$AFC1_3^{2}$	2	590.7	6	554.3	100	−6.4	15	453.2	152.9	−26.3
$AFC1_3^{3}$	5	545.6	16	472.1	104.8	−14.4	8	510.6	46.2	−6.6
$AFC1_4^{0}$	9	498.2	4	559.9	−76.9	11.7	9	502.1	0	0.8
$AFC1_4^{1}$	8	494	3	571.7	−90.9	14.6	12	455	40	−8.2
$AFC1_4^{2}$	4	556.8	6	555.9	40	−0.2	11	468.7	93.3	−17.2
$AFC1_4^{3}$	1	587.4	7	547.8	150	−7	9	513.4	160	−13.4
$AFC1_5^{0}$	6	565.6	10	517.5	50	−8.9	5	559.8	−18.2	−1
$AFC1_5^{1}$	7	548.2	3	587.6	−80	6.9	7	527.3	0	−3.9
$AFC1_5^{2}$	0	600	3	577.6	200	−3.8	4	556.3	200	−7.6
$AFC1_5^{3}$	1	586	3	585.5	100	−0.1	7	553.4	150	−5.7

Index