CHALMERS
UNIVERSITY OF TECHNOLOGY

Evaluating Optimization Solvers and Robust Semantics for Simulation-Based Falsification

Evaluating Optimization Solvers and Robust Semantics for Simulation-Based Falsification*

Johan Lidén Eddeland[1,2], Sajed Miremadi[1], and Knut Åkesson[2]

[1] Volvo Car Corporation, Gothenburg, Sweden
{johan.eddeland, sajed.miremadi}@volvocars.com
[2] Chalmers University of Technology, Gothenburg, Sweden
{johan.eddeland, knut}@chalmers.se

Abstract

Temporal-logic based falsification of Cyber-Physical Systems is a testing technique used to verify certain behaviours in simulation models, however the problem statement typically requires some model-specific tuning of parameters to achieve optimal results. In this experience report, we investigate how different optimization solvers and objective functions affect the falsification outcome for a benchmark set of models and specifications. With data from the four different solvers and three different objective functions for the falsification problem, we see that choice of solver and objective function depends both on the model and the specification that are to be falsified. We also note that using a robust semantics of Signal Temporal Logic typically increases falsification performance compared to using Boolean semantics.

1 Introduction

In this experience report, we present our observation of running simulation-based falsification for a subset[1] of benchmarks from previous year's ARCH workshop [8]. We run all falsification attempts using the same input parameters, but we vary both the *optimization solvers* and the *robust semantics* of *Signal Temporal Logic* (STL) used for falsification of the specifications.

2 Preliminaries

In this section, we describe the underlying theory needed to understand the results.

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

[1]The reason for not including all specifications is due to time constraints.

2.1 Discrete-time signals

Signal Temporal Logic is commonly defined for continuous time [6]. In this work, however, we compare the results of using different robust semantics for STL formulas, and since one of them is the additive semantics for Valued Booleans [4], we define STL for discrete-time signals to be consistent with the Valued Boolean definition. We note that generalization to continuous time is possible [9, 10].

Definition 1. *A* discrete-time signal *is a function $x[k]$ from a finite subset of $I \subset \mathbb{N}$ to $\mathbb{R}$, where $k \in I$. The set I labels the time instants of the signals, and the signal takes on continuous values at each of those time instants.*

2.2 Signal Temporal Logic

The grammar of STL formulas is defined as

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

Here, π^{μ} is an atomic predicate, and φ and ψ are STL formulas. The truth value of π^{μ} is determined by the sign of a real-valued function μ. $\wedge$ denotes logical *and*, $\Box_{[a,b]}$ is the timed *globally* (or *always*) operator, and $\mathcal{U}_{[a,b]}$ is the timed *until* operator. We define logical *or* $\varphi \vee \psi$ as $\neg(\neg\varphi \wedge \neg\psi)$, and the timed *eventually* operator $\Diamond_{[a,b]}\varphi$ as $\neg(\Box_{[a,b]}\neg\varphi)$. Like in previous works [14], we define the validity of a formula φ with respect to the discrete-time signal x at time instant k as

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

2.3 Robust semantics for STL

We present different variations of robust semantics for STL, which indicate not just whether a specification is fulfilled or not, but how *robustly* it is fulfilled. For STL robustness, a positive value indicates that the specification is fulfilled, and a negative value indicates that the specification is not fulfilled. The robust semantics are presented as in previous works [14] as a real-valued function ρ^{φ} of a signal x and time index k such that $(x,k) \models \varphi \equiv \rho^{\varphi}(x,k) > 0$.

We refer to the standard semantics defined in earlier works [10, 6] as the *max* semantics (similar to the definition of Valued Booleans [4]). Apart from this, we also present additive and constant robust semantics.

2.3.1 Max semantics

The max semantics are defined as

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

2.3.2 Additive semantics

The additive semantics are originally defined for Valued Booleans [4]. Note that while a Valued Boolean contains both a Boolean value and a non-negative robustness, and STL robustness is just a real-valued number, the two are in practice equivalent and their only difference is technical.

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

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

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

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

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

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

which is equivalent to the previous definition for discrete time.

2.3.3 Constant semantics

The constant semantics only utilize Boolean information about the satisfaction of the specification, and is thus reasonable to use as a baseline comparison to see if using a robust semantics allows for more successful falsification than just using Boolean semantics.

$$
\rho_c^\varphi(x,k) = \begin{cases} M & \text{if } (x,k) \models \varphi \\ -M & \text{otherwise} \end{cases}
$$

Here, M is an arbitrary positive real-valued number.

3 Experimental setup and results

We evaluate how temporal logic-based falsification is affected by different robust semantics and optimization by attempting to falsify specifications from last year's ARCH workshop [8]. We refer to the benchmark proposal for further details on the models.

The optimization solvers used are discussed in Section 3.1. The table with results is shown in Section 3.3, and we have a brief discussion about the results in Section 3.4.

3.1 Optimization solvers

We briefly present the different optimization solvers used in this experience report. The solvers are mainly chosen by availability to the average user, meaning that there are existing MATLAB implementations of each solver that require little to no setup before using them for falsification. Note that the goal of using different solvers is not mainly to compare the performance of different solvers or to declare which solver is the best for the benchmark, but rather to include solvers with different characteristics and heuristics and see if the effects of varying the specifications' robust semantics are consistent among the different solvers.

The first solver used is a variant [2] of the Simulated Annealing solver [1] which is included in S-TaLiRo [3]. The second solver is *SNOBFIT* [12] (Stable Noisy Optimization by Branch and Fit). The third solver is a CMA-ES solver [11] (Covariance Matrix Adaptation Evolution Strategy) which is included in Breach [5]. The fourth and final solver is an implementation of the Nelder-Mead method [13], which is also included in Breach, however we start with a set of pseudo-random samples (unlike in Breach and in [8], where the algorithm starts with a set of "corner" samples). We also include a comparison with uniform random sampling (UR) – the idea is that if uniform random sampling outperforms a certain solver/semantics combination, it would be better to just perform random testing for the given specification.

We note that from a user's perspective, the first three solvers start from a single random starting point, while Nelder-Mead first evaluates 100 random points before starting the simplex iterations based on a sorted list of the most promising robustness values from the first 100 points.

3.2 Models and input generation

The subset of models used in this experience report, along with the input parametrization for each model, is shown below.

- Automatic Transmission (AT): Throttle and brake inputs are configured with control points at variable times. The throttle has possible values in $[0, 100]$ and has 3 control points, while the brake has possible values in $[0, 325]$ and has 2 control points;
- Fuel Control (AFC): The throttle has possible values in $[0, 61.2]$ and has 10 control points evenly spread over the simulation, while the engine speed has possible values in $[900, 1000]$ and is constant over the simulation;
- Neural Network (NN): The input reference has possible values in $[1, 3]$ and is piecewise constant with 3 control points evenly spread over the simulation;
- Chasing Cars (CC): Both the throttle and the brake have possible values in $[0, 1]$ and are piecewise constant with 4 control points evenly spread over the simulation;
- Aircraft Collision Avoidance (F16): The initial condition for roll, pitch, and yaw have possible values in $[0.2\pi, 0.2833\pi] \times [-0.4\pi, -0.35\pi] \times [-0.375\pi, -0.125\pi]$;

- Steam Condenser (SC): The input has possible values in $[3.99, 4.01]$ and is piecewise constant with 12 control points evenly spread over the simulation.

3.3 Results

Table 1 shows all results from running falsification on a subset of the ARCH workshop benchmarks with varying optimization solver and robust semantics. For each combination of specification, solver, and semantics, we show the average falsification rate, *i.e.*, the number of falsifications divided by the number of simulations. We also show, in parentheses, the average number of simulations per successful falsification.

There are a total of 50 falsifications run for each specification, solver, and robust semantics. Each falsification is set to have a maximum of 300 simulations performed.

We also have additional highlighting that is intended to make the table easier to read:

- For each specification, the solver/semantics combination with the highest (or tied highest) falsification rate has the falsification rate displayed in **bold** characters;
- For each specification, out of the solver/semantics combinations with the highest (or tied highest) falsification rate, the solver/semantics combination with the lowest average number of simulations per successful falsification has that number displayed in **bold** characters (inside the parentheses); and
- For each specification and solver, the robust semantics with the highest falsification rate has grey background. When there are several semantics with the highest falsification rates, only the ones with both highest falsification rate and lowest numbers of average simulations per successful falsification have grey background.

3.4 Discussion

One of the main goals of this paper is to verify that the results of varying robust semantics (which has been evaluated before [7]) also apply when considering several different solvers. In other words, we want to verify that the choice of robust semantics for a specific falsification problem should not be entirely dependent on the optimization solver used.

From the results, we can see that there is no specific solver that always outperforms the other three, and there is also no robust semantics that always outperform the other two. The first conclusion we come to is thus that the performance of each optimization solver is dependent on both the specification, the system, and the robust semantics used. This confirms the results presented earlier, however there is more discussion to be had about the results presented here.

For a clear overview of how each combination of solver and semantics perform, we also include a cactus plot in Figure 1.

3.4.1 Optimization solver performance

Looking purely at success rate, SNOBFIT is the overall most successful solver of the ones presented here. SNOBFIT has the highest or tied highest success rate for 17/19 (89%) specifications (using any of the three robust semantics). If we also take into account that a lower average number of simulations for successful falsifications is preferential, SNOBFIT is the best solver for 8/19 (42%) specifications.

Table 1: All the results from running a subset of the ARCH workshop benchmarks with varying optimization solvers and robust semantics. For each combination of specification, solver, and semantics, we show the average falsification rate, *i.e.*, the number of falsifications divided by the number of simulations. We also show, in parentheses, the average number of simulations per successful falsification.

Solver:	Simulated Annealing			SNOBFIT			CMA-ES			Nelder-Mead			UR
Semantics:	max	add	const	max	add	const	max	add	const	max	add	const	
AT1	52 (163)	72 (168)	0 (-)	**100** (36)	**100 (30)**	0 (-)	**100** (46)	**100** (42)	6 (5)	86 (187)	84 (185)	0 (-)	0 (-)
AT2	98 (21)	**100** (18)	**100** (89)	**100 (5)**	**100 (5)**	**100** (6)	**100** (14)	**100** (15)	70 (9)	**100** (13)	**100** (11)	**100** (12)	**100** (11)
AT51	90 (79)	96 (97)	98 (83)	**100** (10)	**100** (18)	**100 (9)**	36 (11)	64 (22)	36 (11)	**100** (17)	**100** (14)	**100** (18)	**100** (20)
AT52	94 (76)	64 (68)	**100** (77)	**100** (14)	**100** (21)	**100** (14)	78 (15)	88 (61)	78 (15)	**100 (12)**	**100** (14)	**100** (16)	**100** (18)
AT53	**100** (56)	86 (40)	**100** (57)	**100** (12)	**100** (14)	**100 (9)**	96 (12)	92 (24)	96 (12)	**100** (10)	**100** (13)	**100** (12)	**100** (10)
AT54	88 (78)	58 (55)	88 (96)	96 (71)	98 (71)	**100** (74)	38 (15)	76 (62)	38 (15)	**100 (60)**	98 (72)	98 (75)	98 (47)
AT6a	98 (33)	98 (35)	80 (101)	**100** (60)	96 (72)	98 (124)	**100** (45)	**100 (43)**	42 (12)	**100** (58)	98 (51)	98 (80)	98 (51)
AT6b	98 (53)	96 (53)	78 (114)	98 (85)	**100** (83)	94 (123)	**100** (79)	**100** (82)	26 (20)	**100** (82)	**100 (67)**	88 (103)	94 (95)
AT6c	**100 (41)**	**100** (45)	92 (98)	**100** (44)	**100** (53)	**100** (96)	**100** (44)	**100** (45)	46 (18)	**100 (41)**	**100** (42)	98 (46)	**100 (41)**
AFC27	50 (138)	54 (157)	22 (158)	**100** (35)	**100** (33)	90 (108)	**100 (4)**	**100 (4)**	**100 (4)**	72 (110)	56 (118)	26 (100)	40 (115)
AFC29	**100** (50)	**100** (41)	**100** (52)	**100** (6)	**100** (8)	**100** (8)	**100 (2)**	**100 (2)**	**100 (2)**	**100** (11)	**100** (11)	**100** (12)	**100** (13)
NN	98 (80)	96 (110)	90 (100)	98 (73)	**100** (72)	**100 (64)**	96 (69)	**100** (87)	14 (13)	56 (116)	26 (145)	14 (153)	92 (99)
CC1	**0** (-)	**0** (-)	**0** (-)	**0** (-)	**0** (-)	**0** (-)	**0** (-)	**0** (-)	**0** (-)	**0** (-)	**0** (-)	**0** (-)	**0** (-)
CC2	**100** (3)	**100** (5)	**100** (5)	**100 (1)**	**100 (1)**	**100 (1)**	**100** (2)	**100** (2)	**100** (2)	**100 (1)**	**100 (1)**	**100 (1)**	**100 (1)**
CC3	92 (71)	48 (116)	68 (117)	**100 (8)**	**100** (10)	**100** (10)	**100** (30)	96 (54)	32 (11)	**100** (44)	**100** (37)	98 (45)	**100** (47)
CC4	8 (133)	2 (139)	0 (-)	6 (274)	4 (226)	**10 (171)**	0 (-)	0 (-)	0 (-)	4 (109)	8 (181)	2 (242)	2 (91)
CC5	10 (148)	12 (149)	4 (179)	10 (152)	28 (163)	**54 (99)**	8 (30)	10 (38)	10 (16)	32 (126)	14 (147)	18 (117)	36 (149)
F16	10 (168)	8 (184)	2 (299)	40 (136)	36 (151)	6 (22)	32 (147)	**44 (126)**	2 (24)	12 (143)	2 (214)	0 (-)	2 (267)
SC	**0** (-)	**0** (-)	**0** (-)	**0** (-)	**0** (-)	**0** (-)	**0** (-)	**0** (-)	**0** (-)	**0** (-)	**0** (-)	**0** (-)	**0** (-)

For the Nelder-Mead solver with constant semantics, we can see a sharp increase in the number of simulations needed for successful falsifications at 100 simulations. This is because the 100 first simulations of each falsification attempt are pseudo-random – indeed, we can see that the plot of Nelder-Mead closely resembles the Uniform Random sampling up until more than 100 simulations are needed for successful falsifications. After the first 100 simulations, it is clear that the constant semantics do not work well with the Nelder-Mead algorithm. Similarly, it can be seen that the CMA-ES solver has bad performance when all function values are equal.

3.4.2 Robust semantics performance

For some specifications and systems, there is a clear tendency that a specific robust semantics outperforms the other two. Whenever the constant semantics is the most preferable for a given solver, it indicates that the solver has good heuristics with respect to that specification when it comes to choosing the next sample when no useful robustness information is obtained.

For many specifications in this benchmark, the difference in performance between max semantics and additive semantics is small. Specifications where the choice of robust semantics is of higher importance are *e.g.* NN and CC5. We note that since the Nelder-Mead algorithm used starts with 100 random samples, specifications which are falsified by Nelder-Mead in (on average) less than 100 samples will have smaller differences in performance between the different robust semantics.

The general statement to take away from the results is that constant semantics makes the falsification problem dependent on the sampling heuristics of the solver. Specifically in some trivial cases, if a solver has heuristics that will seek to maximize some notion of coverage of the input parameter space, the constant semantics can be preferential to use (see *e.g.* SNOBFIT for AT51 - AT54).

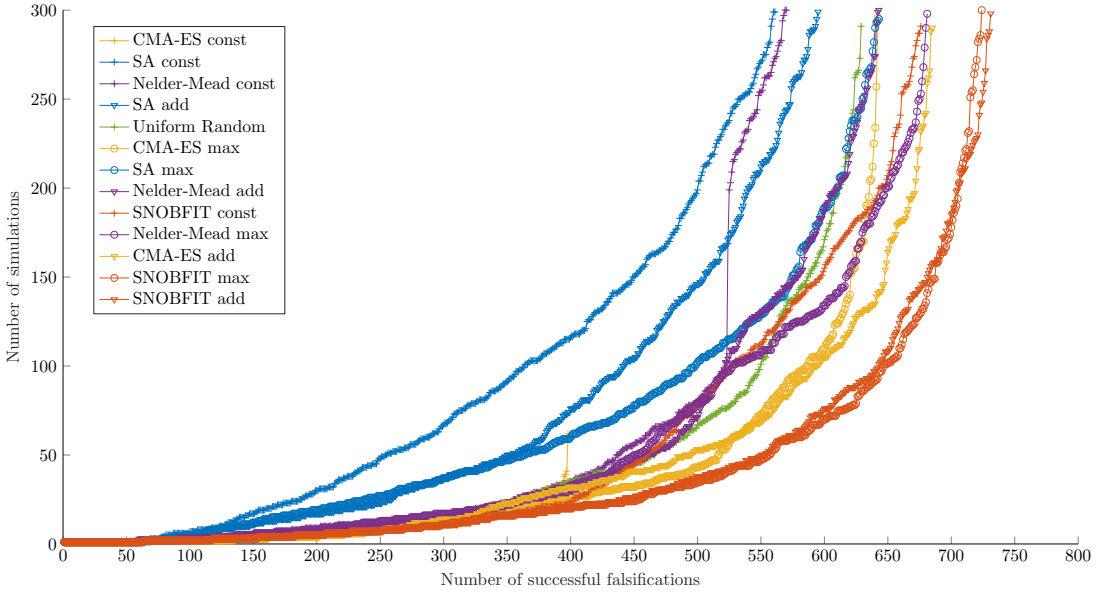

Figure 1: A cactus plot showing performance of combinations of solvers and semantics. For a specific solver/semantics combination, the value plotted shows how many successful falsifications (x-axis) were completed using less than the number of simulations on the y-axis. A curve further down and to the right indicates better performance than a curve upwards and to the left. The maximum number of simulations per falsification is 300. Note that we also include Uniform Random sampling as a baseline comparison of optimization-based testing and random testing.

For the specifications in the benchmark, the solvers in general have better falsification rate for max semantics than for additive semantics – the exception being the CMA-ES solver, however the difference between max and additive semantics is then quite small. Constant semantics perform overall worst for all four solvers, which reinforces the intuition that both max and additive semantics give useful robustness information when trying to falsify the specifications.

4 Conclusions

In this experience report, we investigated how different optimization solvers and robust semantics for STL affect the performance of temporal logic-based falsification for last year's ARCH benchmark specifications. We evaluated four implementations of optimization solvers: Simulated Annealing, SNOBFIT, CMA-ES, and Nelder-Mead. The results were also compared to uniform random sampling. The three different robust semantics evaluated were max semantics, additive semantics, and constant semantics (which is equivalent to using only Boolean semantics).

The main conclusion that can be drawn from the gathered data is that the performance of a constant semantics is dependent on specific solver heuristics – for the specifications evaluated in this report, constant semantics is the overall worst-performing semantics. Which solver performs best depends on the system, specification, and semantics used for falsification. The difference is mostly consistent over different solvers, but there are minor differences. For example, the CMA-

ES and SNOBFIT solvers have an overall better performance for additive semantics, while the other solvers perform best for max semantics. This result is not entirely in line with the result from [7], which showed that additive semantics outperformed max semantics in several cases. However, those results were gathered from a different set of specifications.

For practical purposes, it is clear that to get the best falsification performance, one should consider different semantics and solvers to use for falsification of a given specification. For future work, we aim to further investigate the difference between our results and those previously presented, and how different structures in the specifications affect which semantics are preferable to use for falsification.

References

[1] 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):95, 2013.

[2] 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 *14th Workshop on Advances in Model Based Testing (A-MOST)*, 2018.

[3] Yashwanth Annpureddy, Che Liu, Georgios E Fainekos, and Sriram Sankaranarayanan. S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In *TACAS*, volume 6605, pages 254–257. Springer, 2011.

[4] Koen Claessen, Nicholas Smallbone, Johan Eddeland, Zahra Ramezani, and Knut Åkesson. Using valued booleans to find simpler counterexamples in random testing of cyber-physical systems. *IFAC-PapersOnLine*, 51(7):408–415, 2018.

[5] Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *CAV*, volume 10, pages 167–170. Springer, 2010.

[6] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *FORMATS*, volume 6246, pages 92–106. Springer, 2010.

[7] J. L. Eddeland, K. Claessen, N. Smallbone, Z. Ramezani, S. Miremadi, and K. Åkesson. Enhancing temporal logic falsification with specification transformation and valued booleans. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020. Accepted for publication.

[8] Gidon Ernst, Paolo Arcaini, Alexandre Donze, Georgios Fainekos, Logan Mathesen, Giulia Pedrielli, Shakiba Yaghoubi, Yoriyuki Yamagata, and Zhenya Zhang. ARCH-COMP 2019 category report: Falsification. *EPiC Series in Computing*, 61:129–140, 2019.

[9] Georgios E Fainekos and George J Pappas. Robust sampling for MITL specifications. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 147–162. Springer, 2007.

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

[11] Nikolaus Hansen. The CMA evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pages 75–102. Springer, 2006.

[12] Waltraud Huyer and Arnold Neumaier. Snobfit-stable noisy optimization by branch and fit. *ACM Trans. Math. Softw.*, 35(2):9–1, 2008.

[13] John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[14] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M Murray, and Sanjit A Seshia. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th international conference on hybrid systems: Computation and control*, pages 239–248. ACM, 2015.