EPiC Series in Computing
Volume 80, 2021, Pages 133–152
8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)

ARCH-COMP 2021 Category Report: Falsification with Validation of Results*

Gidon Ernst[1], Paolo Arcaini[2], Ismail Bennani[3], Aniruddh Chandratre[4], Alexandre Donze[5], Georgios Fainekos[4], Goran Frehse[6], Khouloud Gaaloul[7], Jun Inoue[8], Tanmay Khandait[4], Logan Mathesen[4], Claudio Menghi[9], Giulia Pedrielli[4], Marc Pouzet[3], Masaki Waga[10], Shakiba Yaghoubi[4], Yoriyuki Yamagata[8], and Zhenya Zhang[11]

[1] Ludwig-Maximilians-University (LMU), Munich, Germany gidon.ernst@lmu.de
[2] National Institute of Informatics (NII), Tokyo, Japan arcaini@nii.ac.jp
[3] École normale supérieure (ENS), Paris, France {ismail.lahkim.bennani,marc.pouzet}@ens.fr
[4] Arizona State University (ASU), Tempe, USA
{achand75,fainekos,tkhandai,lmathese,gpedriel,syaghoub}@asu.edu
[5] Decyphir SAS, Moirans, France alex@decyphir.com
[6] ENSTA Paris, Palaiseau, France goran.frehse@ensta-paris.fr
[7] University of Luxembourg, Luxembourg khouloud.gaaloul@uni.lu
[8] National Institute of Advanced Industrial Science and Technology (AIST), Osaka, Japan
{jun.inoue,yoriyuki.yamagata}@aist.go.jp
[9] McMaster University, Canada menghic@mcmaster.ca
[10] Kyoto University, Japan mwaga@fos.kuis.kyoto-u.ac.jp
[11] Nanyang Technological University, Singapore zhenya.zhang@ntu.edu.sg

Abstract

This report presents the results from the 2021 friendly competition in the ARCH workshop for the falsification of temporal logic specifications over Cyber-Physical Systems. We briefly describe the competition settings, which have been inherited from the previous years, give background on the participating teams and tools and discuss the selected benchmarks. Apart from new requirements and participants, the major novelty in this instalment is that falsifying inputs have been validated independently. During this process, we uncovered several issues like configuration errors and computational discrepancies, stressing the importance of this kind of validation.

Data: https://gitlab.com/goranf/ARCH-COMP, https://dx.doi.org/10.5281/zenodo.5651631

1 Introduction

We report on the friendly competition associated with the ARCH workshop of the year 2021. The goal of the competition is to compare the state-of-the-art of tools for testing and verification

*The falsification category was coordinated by the first author. The remaining authors represent all participants who have contributed results and/or text to this report and they are listed alphabetically.

of hybrid systems. The competition is organized in several categories, with different specifications (computing reachable regions, checking temporal properties) and varying dynamics in the system models (such as linear/non-linear and hybrid).

This report concerns the *falsification category*, which targets the black-/greybox analysis of executable models with respect to requirements expressed in temporal logic with time bounds, encoded in MTL [28] or STL [29]. The task is to find initial conditions and time-varying inputs subject to given constraints that steer the system into a violation of the respective requirement. Typical approaches are simulation-based and employ quantitative metrics [20, 21] of how close a given input is to violating a requirement ("robustness semantics"). Research in this area has produced a variety of techniques, mature tools, and practical applications; these are described in overview survey articles [4, 9]. For past instalments of this competition 2017–2020 see [13, 11, 18, 17].

The competition of 2021 followed the structure of previous years, with three notable changes:

- We have included some new conjunctive requirements (section 2), which intend to test whether the approaches can deal with multiple—possibly competing—ways in which a particular requirement can be falsified.
- There are two new participating tools (section 3), `FalCAuN` [38] and ForeSee [41]., which means that now we have results of an overall of eight competitors resp. approaches (section 4)
- For the first time, we do not only collect statictics on the success rate, but the actual falsifying inputs as produced by the tools, and we validate them independently (section 5).

The benchmark set developed by this competition series can be seen as a baseline for research in the area (cf. [15]), and we encourage authors to compare to the results presented here. To that end, the models and validation results produced by this competition are available through the shared GitLab repository at `https://gitlab.com/goranf/ARCH-COMP`, notably in the subfolders `models/FALS` and `2021/FALS`. An archive containing the traces submitted for validation is available at `https://dx.doi.org/10.5281/zenodo.5651631`.

2 Benchmark Definitions

2.1 Input Parameterization

Arbitrary piece-wise continuous input signals (Instance 1). This option leaves the input specification up to the participants. The search space is, in principle, the entire set of piece-wise continuous input signals (i.e., which permit discontinuities), where the values for each individual dimensions are from a given range. Additional constraints that were suggested are finite-number of discontinuity and finite variability for all continuous parts of inputs. Further, each benchmark may impose further constraints. Participants may instruct their tools to search a subset of the entire search space, notably to achieve finite parametrization, and then to apply an interpolation scheme to synthesize the input signal.

However, the participants agreed that such a choice must be "reasonable" and should be justified from the problem's specification without introducing external knowledge about potential solutions. Moreover, more general parametrizations that are shared across requirements and benchmark models were preferable. Due to the diversity of benchmarks, it was decided to evaluate the proposed solutions using common sense.

Constrained input signals (Instance 2). This option precisely fixes the format of the input signal, potentially allowing discontinuities. An example input signal would be piecewise constant with k equally spaced control points, with ranges for each dimension of the input, disabling interpolation at Simulink input ports so that tools don't need to up-sample their inputs. The arguments in favor of that are increased comparability of results. As possible downside was mentioned that optimization-based tools (S-TaLiRo and Breach) are just compared with respect to their optimization algorithm. Nevertheless such a comparison is still meaningful, in particular, as FalStar and falsify implement other approaches to falsification.

2.2 Models and Requirements

A brief description of the benchmark models follows, the formal requirements are shown in Table 1 as STL/MTL formulas. The conjunctive requirements introduced in this year are AT6abc, NNx, and CCx, as marked in the table.

Automatic Transmission (AT). This model of an automatic transmission encompasses a controller that selects a gear 1 to 4 depending on two inputs (throttle, brake) and the current engine load, rotations per minute ω, and car speed v. It is a standard falsification benchmark derived from a model by Mathworks and has been proposed for falsification in [25].

Input specification: $0 \leq \mathit{throttle} \leq 100$ and $0 \leq \mathit{brake} \leq 325$ (both can be active at the same time). Constrained input signals (instance 2) permit discontinuities at most every 5 time units. Requirements are specific versions of those in [25] where the parameters have been chosen to be somewhat difficult.

Fuel Control of an Automotive Powertrain (AFC). The model is described in [27] and has been used in two previous instalments of this competition [10, 12]. The specific limits used in the requirements are chosen such that falsification is possible but reasonably hard.

The constrained input signal (instance 2) fixes the throttle θ to be piecewise constant with 10 uniform segments over a time horizon of 50 with two modes (normal and power corresponding to feedback and feedforward control), and the engine speed ω to be constant with $900 \leq \omega < 1100$ to capture the input profile outlined in [27] and to match the previous competitions. For this reason, we do not consider the unconstrained (instance 1) input specification. Faults are disabled (e.g. by setting `fault_time` > 50).

Neural-network Controller (NN). This benchmark is based on MathWork's neural network controller for a system that levitates a magnet above an electromagnet at a reference position.[1] It has been used previously as a falsification demonstration in the distribution of Breach. The model has one input, a reference value Ref for the position, where $1 \leq \mathit{Ref}$ and $\mathit{Ref} \leq 3$. It outputs the current position of the levitating magnet Pos. The input specification for instance 1 requires discontinuities to be at least 3 time units apart, whereas instance 2 specifies an input signal with exactly three constant segments. The time horizon for the problem is 40. The requirement ensures that after changes to the reference, the actual position eventually stabilizes around that value with small error.

[1] https://au.mathworks.com/help/deeplearning/ug/design-narma-l2-neural-controller-in-simulink.html

Table 1: Requirement formulas for the benchmarks

Key	STL formula	Remarks/Constraints
AT1	$\square_{[0,20]} v < 120$	
AT2	$\square_{[0,10]} \omega < 4750$	
AT51	$\square_{[0,30]}((\neg g1 \wedge \circ\, g1) \rightarrow \circ\, \square_{[0,2.5]} g1)$	where $\circ\, \phi \equiv \diamond_{[0.001,0.1]}\ \phi$
AT52	$\square_{[0,30]}((\neg g2 \wedge \circ\, g2) \rightarrow \circ\, \square_{[0,2.5]} g2)$	
AT53	$\square_{[0,30]}((\neg g3 \wedge \circ\, g3) \rightarrow \circ\, \square_{[0,2.5]} g3)$	
AT54	$\square_{[0,30]}((\neg g4 \wedge \circ\, g4) \rightarrow \circ\, \square_{[0,2.5]} g4)$	
AT6a	$(\square_{[0,30]} \omega < 3000) \rightarrow (\square_{[0,4]} v < 35)$	
AT6b	$(\square_{[0,30]} \omega < 3000) \rightarrow (\square_{[0,8]} v < 50)$	
AT6c	$(\square_{[0,30]} \omega < 3000) \rightarrow (\square_{[0,20]} v < 65)$	
AT6abc	AT6a $\wedge$ AT6b $\wedge$ AT6c	cojunctive requirement
AFC27	$\square_{[11,50]}((\mathit{rise} \vee \mathit{fall}) \rightarrow (\square_{[1,5]} \lvert\mu\rvert < \beta))$	$0 \leq \theta < 61.2$ (normal mode)
AFC29	$\square_{[11,50]} \lvert\mu\rvert < \gamma$	$0 \leq \theta < 61.2$ (normal mode)
AFC33	$\square_{[11,50]} \lvert\mu\rvert < \gamma$	$61.2 \leq \theta \leq 81.2$ (power mode)
	where $\beta = 0.008$, $\gamma = 0.007$	
	$\mathit{rise} = (\theta < 8.8) \wedge (\diamond_{[0,0.05]}(\theta > 40.0))$	
	$\mathit{fall} = (\theta > 40.0) \wedge (\diamond_{[0,0.05]}(\theta < 8.8))$	
NN	$\square_{[1,37]}\big(\lvert Pos - Ref\rvert > \alpha + \beta\lvert Ref\rvert \rightarrow \diamond_{[0,2]}\square_{[0,1]}\neg(\alpha + \beta\lvert Ref\rvert \leq \lvert Pos - Ref\rvert)\big)$	
	where $\alpha = 0.005$ and $\beta = 0.03$	
NNx	$\diamond_{[0,1]}(Pos > 3.2) \wedge \diamond_{[1,1.5]}(\square_{[0,0.5]}(1.75 < Pos < 2.25)) \wedge \square_{[2,3]}(1.825 < Pos < 2.175)$	conjunctive requiremet $1.95 \leq Ref \leq 2.05$
WT1	$\square_{[30,630]} \theta \leq 14.2$	
WT2	$\square_{[30,630]} 21000 \leq M_{g,d} \leq 47500$	
WT3	$\square_{[30,630]} \Omega \leq 14.3$	
WT4	$\square_{[30,630]} \diamond_{[0,5]} \lvert\theta - \theta_d\rvert \leq 1.6$	
CC1	$\square_{[0,100]} y_5 - y_4 \leq 40$	
CC2	$\square_{[0,70]} \diamond_{[0,30]} y_5 - y_4 \geq 15$	
CC3	$\square_{[0,80]}((\square_{[0,20]} y_2 - y_1 \leq 20) \vee (\diamond_{[0,20]} y_5 - y_4 \geq 40))$	
CC4	$\square_{[0,65]} \diamond_{[0,30]} \square_{[0,20]} y_5 - y_4 \geq 8$	
CC5	$\square_{[0,72]} \diamond_{[0,8]}((\square_{[0,5]} y_2 - y_1 \geq 9) \rightarrow (\square_{[5,20]} y_5 - y_4 \geq 9))$	
CCx	$\bigwedge_{i=1..4} \square_{[0,50]}(y_{i+1} - y_i > 7.5)$	conjunctive requirement
F16	$\square_{[0,15]} \mathit{altitude} > 0$	
SC	$\square_{[30,35]}(87 \leq \mathit{pressure} \wedge \mathit{pressure} \leq 87.5)$	

Wind Turbine (WT). The model is a simplified wind turbine model proposed in [37]. The input[2] of the system is wind speed v and the outputs are blade pitch angle θ, generator torque $M_{g,d}$, rotor speed Ω and demanded blade pitch angle θ_d. The wind speed is constrained by $8.0 \leq v \leq 16.0$. Instance 1 allows any piece-wise continuous inputs, while instance 2 constrains inputs to piece-wise constant signals whose control points which are evenly spaced each 5 seconds. The model is relatively large. Further, the time horizon is long (630) compared to other benchmarks.

Chasing cars (CC). The model is derived from Hu et al. [26] which presents a simple model of an automatic chasing car. Chasing cars (CC) model consists of five cars, in which the first car is driven by inputs (*throttle* and *brake*), and other four are driven by Hu et al.'s algorithm. The output of the system is the location of five cars y_1, y_2, y_3, y_4, y_5. The properties to be falsified are constructed artificially, to investigate the impact of complexity of the formulas to falsification. The input specifications for instance 1 allows any piecewise continuous signals while the input specification for instance 2 constraints inputs to piecewise constant signals with control points for each 5 seconds, i.e., 20 segments.

Aircraft Ground Collision Avoidance System (F16). The model has been derived from the one presented in [24]. The F16 aircraft and its inner-loop controller for Ground Collision avoidance have been modeled using 16 continuous variables with piece-wise nonlinear differential equations. Autonomous maneuvers are performed in an outer-loop controller that uses a finite-state machine with guards involving the continuous variables. The system is required to always avoid hitting the ground during its maneuver starting from all the initial conditions for roll, pitch, and yaw in the range $[0.2\pi, 0.2833\pi] \times [-0.4\pi, -0.35\pi] \times [-0.375\pi, -0.125\pi]$.[3] Since the benchmark has no time-varying input, there is no distinction between instance 1 and instance 2. The requirement is checked for a time horizon equal to 15.

Steam condenser with Recurrent Neural Network Controller (SC). The model is presented in [39]. It is a dynamic model of an steam condenser based on energy balance and cooling water mass balance controlled with a Recurrent Neural network in feedback. The time horizon for the problem is 35 seconds. The input to the system can vary in the range $[3.99, 4.01]$. For instance 2, the input signal should be piecewise constant with 20 evenly spaced segments.

3 Participants

We briefly describe in alphabetical order all participating tools, the respective main ideas of the underlying approaches, followed by details on how each tool was set up for the competition.

3.1 ARIsTEO

Description. ARIsTEO [33] is a Matlab toolbox for test case generation against system specifications presented in STL and it is developed on the top of S-TaLiRo. ARIsTEO is

[2]Organizer's comment: The status of the wind turbine benchmark is currently unresolved. The model contains an alternative setup, where the wind speed is not a time-varying input provided by the falsification tool, but taken from one of 13 preset inputs. It remains unclear how this mechanism interacts with the input port for v of the Simulink model. Until this issue is resolved, the model is exempt from validation.

[3]The report from 2019 [18] erroneously specifies: $[0.2\pi, 0.2833\pi] \times [-0.5\pi, -0.54\pi] \times [0.25\pi, 0.375\pi]$, however, previous results were in fact obtained with the correct range.

designed to targeting a large and practically-important category of CPS models, known as *compute-intensive* CPS (CI-CPS) models, where a single simulation of the model may take hours to complete. ARIsTEO embeds black-box testing into an iterative approximation-refinement loop. At the start, some sampled inputs and outputs of the model under test are used to generate a surrogate model that is faster to execute and can be subjected to black-box testing. Any failure-revealing test identified for the surrogate model is checked on the original model. If spurious, the test results are used to refine the surrogate model to be tested again. Otherwise, the test reveals a valid failure. ARIsTEO is publicly available under the General Public License (GPL).[4]

Setup. ARIsTEO provides the same interface and parameters as S-TaLiRo, while providing additional configuration options. We had used an ARX model (ARX-2) with order $na = 2$, $nb = 2$, and $nk = 2$[5] as structure for the surrogate model used in the approximation-refinement loop of ARIsTEO. For models with multiple inputs and outputs the dimension of the matrix na, nb and nk is changed depending on the number of inputs and outputs. We used the default configuration of S-TaLiRo for searching failure-revealing revealing tests on the surrogate model. We considered the same parametrization of S-TaLiRo for the input signals. The original Simulink model was executed once to learn the initial surrogate model. The cut-off values for the number of simulations of the original model and for the number of simulations of the surrogate model (per trial) were set to 300. The results of ARIsTEO can further improve by (i) using configurations for the surrogate model that provide more accurate approximations of the original models and more effectively guide the search toward faulty inputs; and (ii) using the SOAR option of S-TaLiRo that significantly improved the results of S-TaLiRo compared with the last edition of this competition.

3.2 Breach

Description. Breach [14] is a Matlab toolbox for test case generation, formal specification monitoring and optimization-based falsification and mining of requirements for hybrid dynamical systems. A particular emphasis is put on modularity and flexibility of inputs generation, requirement evaluation and optimization strategy. For this work, the approach has been to ensure that each benchmark was properly implemented and a default, relatively basic falsification strategy has been applied. The idea was to perform a first systematic investigation of the proposed problems, and then to provide a base to work on for future editions of the competition to test a larger variety on approaches on the most challenging instances. Breach is available under BSD license[6].

Setup. For most benchmarks (exceptions detailed below), a piecewise constant signal generation was used with fixed step size. For all instances, the optimization strategy used is the default global Nelder Mead (GNM) approach with a custom configuration for the competition, resulting in the following three phases behavior:

- Phase 1.: at most $n_{\text{corners}} = 64$ corner samples are tested, i.e., inputs for which control points take only extreme values;
- Phase 2.: $n_{\text{quasi-rand}} = 100 - n_{\text{corners}}$ quasi-random samples from the Halton sequence with varying start points determined by a random seed are tested;

[4]https://github.com/SNTSVV/ARIsTEO
[5]https://nl.mathworks.com/help/ident/ref/arx.html
[6]https://github.com/decyphir/breach

- Phase 3.: the robustness results from phase 1 and 2 are sorted and Nelder Mead optimization is run from the most promising samples.

Note that as a result of this approach, whenever a falsifying input is consistently found with less than 100 simulations, it indicates that the problem is likely falsifiable with extreme inputs or a quick stochastic exploration of the search space.

For instance 2 in several problems, the pulse generation approach investigated in [36] was used this year. This made it possible to obtain slightly better results than previous editions of the competitions. Specifically, CC4 could be falsified 3 times out of 50 using two independent saturated pulse generators for the 2 inputs with varying periods, delays and width, and SCa could be falsified 48 out of 50 times using a pulse generator with varying period and delay.

3.3 `FalCAuN`

Description. `FalCAuN` [38] is an experimental tool for testing a Simulink model using black-box checking [35], an automated testing method based on active automata learning and model checking. In `FalCAuN`, the input and the output signals of the Simulink model are discretized in time and values, and the model is abstracted into a black-box Mealy machine. `FalCAuN` learns the Mealy machine and conducts model checking to find a counterexample. `FalCAuN` is designed to efficiently falsify a Simulink model against multiple specifications by reusing the learned Mealy machine. `FalCAuN` is publicly available under General Public License (GPL) v3[7].

We utilize the *discrete-time* semantics of STL, which is essentially the same as the semantics of LTL. Because of such discretization, the control points must be fine enough to capture the timing constraints in the STL formula. For example, in order to capture the timing constraint $\Diamond_{[0,0.05]}$, the duration between the control points must be at most 0.05. We note that due to the use of the discrete-time semantics, the signal reported as a counterexample by `FalCAuN` may not falsify the model in terms of the continuous-time semantics.

Setup. For the signal discretization, we have the following hyperparameters: the (constant) duration of the intervals between samples, the input signal values at the control points, and the thresholds of the output signal values for the discretization. We used the shortest duration between the control points such that the LTL encoding of the STL formula is small enough for the back-end model checker LTSMin. The duration ranges from 1.0 to 10.0 time units. In most benchmarks, we let the input signal values be the maximum and the minimum of the range. The exceptions are as follows.

- In AT6a, AT6b, and AT6c, the throttle can be 50 in addition to 0 and 100.
- In AT2 instance 2, the throttle and the brake can be 5 and 6 evenly spaced values, respectively, i.e., the value of the throttle can be one of 0, 25, 50, 75, and 100.
- In SCa, the input can be 4.00 in addition to 3.99 and 4.01.

In most benchmarks, we let the threshold of the output signal values be the thresholds in the STL formula. The exceptions are as follows.

- In AT1 and AT2 instance 2, we have the common thresholds: 120 for the speed and 4750 for the RPM.
- In AT6a, AT6b, and AT6c, we have the common thresholds: 35, 50, and 65 for the speed and 3000 for the RPM.
- In CC1, CC2, and CC3, we have the common thresholds: 15 and 40 for $y_5 - y_4$ and 20 for $y_2 - y_1$.

[7] https://github.com/MasWag/FalCAuN

3.4 falsify

Description. falsify is an experimental program which solves falsification problems of safety properties by reinforcement learning [40]. falsify uses a *grey-box* method, that is, it learns system behavior by observing system outputs during simulation. falsify is currently implemented by a deep reinforcement learning algorithm *Asynchronous Advantage Actor-Critic* (A3C) [34].

Setup. The input specification uses piecewise constant function with discontinuities spaced in even intervals ΔT. $\Delta T = 1$ for all models except for SC in which $\Delta T = 0.1$ is used. The choice for the SC model was $\Delta T = 0.1$ model because Instance 2 uses $\Delta T = 1.75$, which is near to $\Delta T = 1$.

3.5 FalStar

Description. FalStar [19] is an a falsification tool that explores the idea to construct falsifying inputs incrementally in time, thereby exploiting potential time-causal dependencies in the problem. The algorithm used in this competition is called adaptive Las-Vegas tree search (aLVTS). The main idea is to try "simple" inputs first, i.e., to scale gracefully with the intuitive combinatorial hardness of the benchmark problem [16]. This is achieved by sampling from input domains with gradually finer temporal and spatial resolution, starting with extreme values. The approach can be regarded as a more reasonable baseline over random sampling, which takes the structure of the problem into account, but which neither relies on sophisticated optimization methods nor insight into the models themselves. The code is publicly available under the BSD license.[8]

Setup. The search space for instance 1 included piecewise constant inputs (the only parameterization currently supported), ranging from 2 upto 4 control points at which discontinuities are allowed (resp. upto 3 for NN). In this configuration FalStar benefits from a low number of control points and is more likely to try inputs with fewer control points first. For the AT benchmarks it was clear beforehand that this choice suffices to falsify all benchmarks, and the setting was then kept for the remaining experiments. Instance 2 input signals are parameterized with the number of control points as specified by the respective benchmarks. The F16 benchmark does not have a time-varying input and therefore the strategy proposed in [19] is not applicable. Instead, global optimization with Nelder/Mead is used on this benchmark, which is effective.

3.6 ForeSee

Description. In falsification, the *scale problem* can occur when the signals used in the specification have different scales (e.g., rpm and speed): namely, the contribution of a signal could be *masked* by another one when computing robustness. ForeSee [41] (`FOR`mula `E`xploitation by `S`equence tr`EE`) tackles this problem by introducing a new robustness definition, called *QB-Robustness*, which combines quantitative robustness and classical Boolean satisfaction. QB-Robustness does not require comparing (i.e., by minimum or maximum) robustness values of different sub-formulas, so possibly avoiding the scale problem. However, in order to be computed, QB-Robustness requires the selection of a sequence of sub-formulas along the syntax tree of the specification for which to compute the quantitative robustness. Different sub-formulas sequences can be more or less effective in mitigating the scale problem.

[8] https://github.com/ERATOMMSD/falstar

ForeSee implements a falsification strategy based on a Monte Carlo Tree Search over the structure of the formal specification: first, by tree traversal, it identifies the sub-formulas sequence; then, on the leaves, it performs numerical hill-climbing optimization, with the aim of falsifying the selected sub-formulas. ForeSee is the spiritual successor of FalStar/MCTS from [18, 17]. It is publicly available under GNU General Public License (GPL) v3.[9]

Setup. Since ForeSee is implemented on the basis of Breach, it provides the same interface of Breach, namely, users can characterize the shape of input signals with a number of options, including piecewise constant, piecewise linear, pulse, etc. In this report, we regulate the shape of input signals with piecewise constant, parametrized by the number of *control points*.

In the current implementation of ForeSee, only CMA-ES [2] is provided as the optimizer; this is due to our insight in the performances of different optimizers, in which CMA-ES outperforms other optimizers. However, involving other optimizers is not difficult for ForeSee, and will be considered in the future releases.

Since ForeSee technically relies on Monte Carlo Tree Search (MCTS), the hyperparameters in MCTS need to be properly selected. As a default setting, we use 0.2 as the scalar in the UCB1 algorithm, that takes a balance of *exploration* and *exploitation*; and we set 10 generations as the budget for the playout phase of MCTS.

3.7 S-TaLiRo

Description. S-TaLiRo [1] is a Matlab toolbox for monitoring and test case generation against system specifications presented in STL (or MTL). The test cases are automatically generated using optimization techniques guided by formal requirements in STL in order to find falsifying systems behaviors. The tool supports different optimization algorithms. In past competitions, the Stochastic Optimization with Adaptive Restarts (SOAR) [32] framework was used for all the benchmarks except for choosing instance 1 type inputs in Steam Condenser model. In that benchmark, Simulated annealing global search was combined with a local optimal control based search [39]. For the 2021 competition, we have used the minSOAR framework [31, 30] for the newly proposed conjunctive benchmarks. The minSOAR framework can be thought of as an extension to SOAR which can handle multiple conjunctive requirements by modeling information from each resulting robustness component. S-TaLiRo is publicly available on-line under General Public License (GPL) [10].

Setup. In S-TaLiRo, input signals are parameterized in two ways: the number of control points for the input signal, and the time location of those control points during simulation. The number of control points for each input signal is given by the user forming an optimization problem with search space dimension the same as the number of control points. An option is provided to the user to add to the search space the timing of the control points, but this option is not used in the competition. For this competition, the control point time locations are evenly spaced over the duration of the simulation for all the benchmarks except for the SC problem instance 1.

For the transmission model the [*throttle*, *brake*] control points are interpolated with the *pchip* function, with $[7, 3]$ as the number of control points in specifications 1-6 and $[4, 2]$ for 7-9 to reduce the dimensionality of the search space. For the Neural model, we use 13 control points to yield piecewise constant signals of 3.33 seconds apart. The Wind Turbine used the default

[9] https://github.com/choshina/ForeSee

[10] https://sites.google.com/a/asu.edu/s-taliro

model input of 126 control points interpolated linearly. For the SC model, Simulated Annealing (SA) global search was utilized in combination with an optimal control based local search on the infinite dimensional input space. The SA global search utilizes piecewise constant inputs with 12 possibly uneven time durations.

3.8 `zlscheck`

Description. `zlscheck` is a tool for test case generation of programs written in Zélus[11] [7], a language reminiscent of the synchronous languages Lustre [22] and Scade [8] extended in order to express ODEs. For now `zlscheck` applies to the discrete-time subset of Zélus.
Properties are expressed as synchronous observers [23] with a quantitative semantics to solve the falsification problem as an optimization problem. `zlscheck` uses automatic differentiation to compute gradients of the robustness of a model w.r.t. some input parameters and uses gradient-based techniques to find a falsifying input. Additionally, it uses FADBADml[12], a tool for automatic differentiation which we ported from C++ to OCaml.
All the models of this competition have been rewritten manually in Zélus and are available along with the tool on github[13]. The Simulink's Integrator block is programmed in Zélus as a fixed-step forward euler scheme.
The counter-examples found by `zlscheck` were systematically validated on their corresponding Simulink models, the ones that did not pass validation were discarded from the final results.

Setup. The inputs of the systems are bounded piecewise constant streams. A bounded piecewise constant stream x of size N and period k is such that $\forall n \in [0, N], x(n) = x(\lfloor \frac{n}{k} \rfloor \cdot k)$. It is totally defined by $(\lfloor \frac{N}{k} \rfloor + 1)$ values (parameters).
Two different strategies were used in these benchmarks:

- classic: the optimization is done offline: the parameters are generated at the beginning of the simulation, then the corresponding robustness is computed. The optimization algorithm then uses the robustness and its gradient w.r.t. the parameters to compute the next parameters.
  This strategy has been used for properties AT1, AT2, AT6, AFC29, AFC33, NN, WT, F16 and SC.
- mode switch: the optimization is performed online: for each input, the parameter number $\lfloor \frac{n}{k} \rfloor$ is generated at step $\lfloor \frac{n}{k} \rfloor \cdot k$ of the simulation.
  In order to achieve coverage of the different modes of the system (a mode is a state in a hierarchical automaton), `zlscheck` chooses randomly an available transition and drives the system towards triggering it: it computes a quantitative interpretation of the complement of its guard (transition robustness) and the gradients of that robustness w.r.t. the current values of the inputs of the system (at step n this would be the parameters number $\lfloor \frac{n}{k} \rfloor$). The value of the next parameter is then computed by the optimization algorithm using its precedent value and the gradient.
  Once the transition has been triggered, the tool chooses a new one randomly and so on, until the simulation ends. In this mode, the inputs are generated independently of the property being falsified.
  This strategy has been used for properties AT5, AFC27 and CC.

[11] http://zelus.di.ens.fr/
[12] https://fadbadml-dev.github.io/FADBADml/
[13] https://github.com/ismailbennani/zlscheck

The gradient-based algorithm used by `zlscheck` is a simple gradient descent with decreasing step-size: at step l of the optimization, the step-size is $a(l) = a(0)/\sqrt{l}$ where $a(0)$ is a meta-parameter. The first input is sampled uniformly in the input space, and if the same input is generated twice in a row, the algorithm restarts. Other gradient descent algorithms (ADAM, ADAGRAD, AMSGRAD) are implemented in `zlscheck` and have been tested but did not give significantly better results.
Additionally, given that the integration scheme is fixed-step, we can express the period k of the inputs as times instead of number of simulation steps. For instance 1, those periods are (in seconds): $\Delta T_{AT} = 0.5$ except $\Delta T_{AT2} = 2.5$, $\Delta T_{AT6a} = 5$, $\Delta T_{AT6b} = 10$, $\Delta T_{AT6c} = 15$ and $\Delta T_{AFC} = 5$ and $\Delta T_{NN} = 3$ and $\Delta T_{WT} = 5$ and $\Delta T_{CC} = 5$ and $\Delta T_{F16} = +\infty$, $\Delta T_{SC} = 0.2$.

4 Evaluation

Falsification tools were instructed to run each individual requirement 50 times, to account for the stochastic nature of most algorithms. We report the falsification rate, i.e., the number of trials where a falsifying input was found, as well as the median and mean of the number of simulations required to find such input (not including the unsuccessful runs in the aggregate). The cut-off for the number of simulations per trial was 300.

The results were provided by the participants, and reflect experiments from multiple years, on multiple platforms with varying resources, and different MATLAB/Simulink versions. The results of ForeSee and `FalCAuN` (new participants), as well as those of falsify and ARIsTEO (tool changes), have been updated from last year; similarly for the results of the new conjunctive benchmarks where given.

The results for unconstrained piecewise-continuous input signals (instance 1) are shown in table 2. For a better comparison of the performance of the tools, a common ground is piecewise constant input signals (instance 2) with a concrete specification of the number of discontinuities allowed. The corresponding results are shown in table 3. Empty cells indicate lack of data for various reasons, such as missing tool support for a particular benchmark feature, or simply that the respective participants did not take the time to set up and/or run these experiments.

The results depend on the choices for the search space, which we briefly discuss for each participating tool as described in section 3.

We do not report falsification times, as this information is not meaningful and incomparable due to the way the results were produced. Instead, the comparison is based on how often the simulation engine is queried, which typically dominates falsification time by a large factor.

For a more detailed discussion of the outcomes, see [18, 17].

5 Validation.

The 2021 competition took the first steps towards validating the results produced by the tools. The overarching goal was to ensure that the comparison reported here is meaningful, and the approach taken accounts for several potential sources of error, both for technical reasons or because of human error. The hypothetical case of cheating participants was not regarded likely, and we emphasize upfront that no indication whatsoever for dishonest behavior was found. Rather, the goal is to establish a higher standard of quality of evaluation results, that can ultimately benefit any future work in simulation-based falsification: Just like the benchmark set established by this community gets adopted by experiments in the literature, validation of results using an independent reference checker should become standard, too.

Table 2: Results for piecewise continuous input signals (instance 1). *FR*: falsification rate wrt. number of 50 trials, $\overline{S}$ and $\widetilde{S}$: mean resp. median (rounded down) number of simulations over successful trials ("–" if *FR* is zero).

Tool:	**ARIsTEO**			**Breach**			**`FalCAuN`**			**falsify**			**FalStar**			**ForeSee**			**S-TaLiRo**			**`zlscheck`**		
Approach:	ARX-2			GNM						A3C			aLVTS						SOAR			GD		
Benchmark	*FR*	$\overline{S}$	$\widetilde{S}$	*FR*	$\overline{S}$	$\widetilde{S}$	*FR*	$\overline{S}$	$FR^\star$	*FR*	$\overline{S}$	$\widetilde{S}$	*FR*	$\overline{S}$	$\widetilde{S}$	*FR*	$\overline{S}$	$\widetilde{S}$	*FR*	$\overline{S}$	$\widetilde{S}$	*FR*	$\overline{S}$	$\widetilde{S}$
AT1	0	-	-	50	11.0	11	0	–	48	17	224.5	223	50	33.0	30	29	252.3	270	50	118.8	116	50	3.4	2
AT2	50	4.1	3	50	2.0	2	5	180	29	50	22.1	10	50	4.3	3	48	31.9	16	50	23.9	19	50	15.5	2
AT51	0	–	–	41	74.6	67							50	69.5	56	50	16.4	11	50	26.7	22	50	3.5	5
AT52	50	3.2	2	49	72.0	67							26	125.4	137	50	46.2	29	50	4.1	3	50	1.6	1
AT53	50	2.6	2	49	74.5	73							50	70.8	68	50	2.6	2	50	3.4	3	50	1.3	1
AT54	3	295.1	300	21	84.9	85							50	71.1	52	49	65.2	43	50	10.5	2	50	3.5	2
AT6a	50	39.0	23	50	97.9	97	1	290	50				50	76.1	70	49	117.0	108	49	78.4	40	50	48.3	29
AT6b	49	101.7	83	49	112.9	118	0	–	50				50	82.4	75	39	180.2	175	33	132.6	128	35	77.7	102
AT6c	50	48.2	34	50	94.1	89	0	–	50				8	221.4	234	47	124.6	122	47	61.3	38	32	18.9	23
AT6abc	50	27.2	16										10	140.9	134				48	73.0	42			
NN	50	62.8	46	48	96.3	101				50	1.0	1	36	122.8	106	50	96.8	64	50	26.7	22	50	1.0	1
$NN_{\beta = 0.04}$																			4	193.0	222	50	1.1	1
NNx													35	131.6	112	0	–	–	37	153.9	143			
WT1	50	13.0	10	50	3.0	3				37	47.7	7							50	91.0	91	50	1.4	1
WT2	50	1.3	1	50	3.0	3				46	8.0	2							50	32.6	30	48	1.0	1
WT3	50	2.4	2	50	3.0	3				50	2.5	1							50	44.1	60	50	1.1	1
WT4	50	1.0	1	50	30.0	30				50	4.9	4							50	3.3	2	0	–	–
CC1	50	27.0	15	50	3.0	3	48	161	50	47	51.3	17	50	4.1	2	50	26.8	23	50	9.5	7	50	8.8	12
CC2	50	9.2	6	50	1.0	1	50	43	50	37	24.2	4	50	4.0	2	14	25.4	20	50	6.0	4	50	4.7	3
CC3	50	56.3	46	50	3.0	3	48	155.5	50	46	35.4	8	50	6.9	5	50	28.6	9	50	19.9	5	50	23.4	16
CC4	0	–	–	3	287	287				1	26.0	26	2	52.0	60	13	311.1	315	20	188.0	179	32	124.6	164
CC5	50	25.9	17	49	26.1	19				31	29.7	26	46	91.2	79	50	63.7	22	50	42.9	36	50	2.0	3
CCx	15	250.0	300										12	123.5	106	45	137.5	122	45	232.5	227			
F16	0	–	–	1	297.0	297				(no support)			†	69.0	69	(no support)			7	127.6	94	0	-	-
SC	0	–	–	48	128.8	109.5	0	–	0	0	–	–	0	–	–	–	–	–	‡50	62.2	55	50	2.1	2

F16 †: FalStar uses Nelder/Mead optimization initialized by extreme values; since the approach is deterministic it was executed once only, with successfully finding a violation after 69 simulations.
SC ‡: The S-TaLiRo results for this benchmark are yielded by Simulated Annealing assisted with gradient based search (see [39]).
`FalCAuN` ⋆: Column $FR^\star$ are results obtained when the simulations required for automata learning are not counted, only the simulations required for equivalence testing are capped at 300 (see [38]).

One obvious source of error are mistakes in the translation of formal requirements into the format of the respective tool, and similar problems with the experimental setup, like wrong input parameterization.

More subtly, there might be mismatches in the numeric computations underpinning the simulations, due different versions of the models as well as MATLAB, or due to different model configurations, such as the ODE solver/integrator settings, and the interpolation setting on the input ports. Rounding errors introduced during pretty printing and parsing of the result format are expected, too, and MATLAB seems to implement some effort to work around limitations of the underlying floating point data-type, which mismatches the way such values are handled inside the Java virtual machine on which the validator runs.

Similarly, there may be differences between computation of the robustness score, from which falsification success is typically derived: Four implementations are used in the competition—from S-TaLiRo (also used by ARIsTEO), Breach (also used by ForeSee), FalStar, and

Table 3: Results for constrained input signals/instance 2. FR: falsification rate (of 50), $\overline{S}$: mean number of simulations, $\widetilde{S}$: median (rounded down) number of simulations.

Tool:	**ARIsTEO**			**Breach**			`FalCAuN`			**falsify**			**FalStar**			**ForeSee**			**S-TaLiRo**			`zlscheck`		
Approach:	ARX-2			GNM						A3C			aLVTS						SOAR			GD		
Benchmark	*FR*	$\overline{S}$	$\widetilde{S}$	*FR*	$\overline{S}$	$\widetilde{S}$	*FR*	$\overline{S}$	*FR**	*FR*	$\overline{S}$	$\widetilde{S}$	*FR*	$\overline{S}$	$\widetilde{S}$	*FR*	$\overline{S}$	$\widetilde{S}$	*FR*	$\overline{S}$	$\widetilde{S}$	*FR*	$\overline{S}$	$\widetilde{S}$
AT1	0	–	–	0	–	–	46	171.2	50	39	125.8	110	50	33.0	30	29	252.3	270	50	170.3	171	50	2.3	2
AT2	50	5.1	4	50	2.0	2	0	–	20	48	19.0	7	50	4.3	3	48	31.9	16	50	16.8	9	50	9.2	2
AT51	50	8.4	6	50	7.0	7							50	69.5	56	50	16.4	11	50	12.6	11	50	8.7	12
AT52	50	3.5	3	50	3.0	3							26	125.4	137	50	46.2	29	49	17.6	4	50	35.1	31
AT53	50	2.6	2	50	3.0	3							50	70.8	68	50	2.6	2	50	3.4	3	50	22.7	26
AT54	50	29.4	17	50	3.0	3							50	71.1	52	49	65.2	43	50	24.2	16	47	56.3	43
AT6a	47	103.1	80	0	–	–							50	76.1	70	49	117.0	108	44	130.4	149	50	42.7	26
AT6b	45	164.7	174	0	–	–							50	82.4	75	39	180.2	175	39	207.2	236	5	129.5	102
AT6c	49	89.1	60	0	–	–	43	50	214.1				0	–	–	47	124.6	122	42	197.5	208	2	261.7	288
AT6abc	50	79.1	70													50	105.2	88						
AFC27	50	2.3	1	50	3.0	3				50	1.6	1	50	3.9	3	50	2.8	2	50	70.3	78	50	1.0	1
AFC29	50	1.0	1	50	3.0	3				50	1.0	1	50	1.2	1	50	1.0	1	50	13.0	10	0	–	–
AFC33	50	1.0	1	0	–	–				50	1.0	1	0	–	–	50	1.0	1	0	–	–	24	2.1	2
NN	50	62.8	46	50	6.0	6				50	1.0	1	26	177.0	197	50	44.2	34	49	68.0	48	50	1.3	1
NN$_{\beta = 0.04}$																			3	127.0	74	50	1.4	1
NNx	50	1.0	1										23	175.7	166	0	–	–						
WT1	50	1.4	1	50	3.0	3				49	8.6	2							50	7.1	5	50	1.4	1
WT2	50	1.0	1	50	3.0	3				50	2.8	2							50	1.0	1	48	1.0	1
WT3	50	1.1	1	50	3.0	3				50	2.0	1							50	1.0	1	50	1.1	1
WT4	50	1.0	1	50	30.0	30				50	3.7	3							50	12.0	9	0	–	–
CC1	50	9.1	6	50	5.0	5				50	23.5	7	50	7.3	6	50	26.8	23	50	67.8	91	50	8.8	12
CC2	50	6.7	4	50	1.0	1				46	14.4	4	50	15.9	9	14	25.4	20	48	114.5	105	50	4.7	3
CC3	50	18.9	15	50	28.6	9				44	13.5	2	4	207.5	229	50	14.4	17	50	22.4	13	50	23.4	16
CC4	0	–	–	0	–	–				9	120.4	168	0	–	–	13	311.1	315	0	–	–	32	124.6	164
CC5	50	29.1	12	16	84.7	79				32	37.2	8	39	117.9	103	50	63.7	22	50	74.9	48	50	2.0	3
CCx	20	229.4	300													45	137.5	122						
SC	0	–	–	0	–	–				0	–	–	0	–	–	0	–	–	0	–	–	0	–	–

`zlscheck`—whereas `FalCAuN` uses boolean semantics of formulas.

To that end, for each (successful) of the 50 respective trials per requirement, we collected the following information, from which the above questions can be answered satisfactorily:

- information about which benchmark (model + requirement identifier)
- the initial conditions and time-series input signal resulting from that trial
- whether the signal is expected to falsify the requirement
- if available, a robustness value derived from running the input through the model
- optionally, the corresponding output signal, and further information such as time stamps or wall-clock times

In the following, we will refer to the this information as the "reported" result. Multiple questions were identified as concerns for validation, albeit not all points were achieved or fully taken into account:

1. does the reported input signal adhere to the valid ranges of input for that particular model, as described in section 2?
2. does the reported input signal adhere to the constraints of the respective instance?
3. is the reported verdict correct, i.e., whether running this signal through the model produces a falsifying trace?

4. is the reported robustness value consistent with the verdict (values strictly < 0 indicate a falsification), and how accurately can the robustness be reproduced?
5. how accurately can the reported output be reproduced, if given?
6. are the values reported in tables 2 and 3 correct?

In the end, we checked for 1., 3., and 4. Regarding 2., it is not quite straight-forward to check the shape of the signal, so we did not attempt it. Regarding 5., not all participants did report output signals, so we omitted this aspect from further investigation. For 6., there was simply not sufficient consistent data, as many participants chose to run a fraction only of the experiments represent in the tables.

Technical Details. A simple CSV-based format was established,[14] which includes one row for each result, and one column per result for the individual entries. Compound data, specifically initial conditions and time-series signals, are formatted as Matlab matrices that are included as quoted atomic table entries. Reference to benchmarks was done via mnemonic keys for the models (like `AT`), as well as for the requirements (like `AT1`), which avoids tight coupling with a particular model file or requirement syntax.

Validation is implemented inside FalStar. It is based on the reference models in the folder `models/FALS` in the shared Git repository, in conjunction with a configuration file that sets up the parameter and input ranges, and which defines the STL formulas associated with the requirements in FalStar's internal syntax. The configuration file as well as the scripts used to interface with the simulation engine are public and transparent.

The process to converge to a reasonable set of results was iterative, where the validator, the format, and the data reported was refined multiple times, uncovering a number of problems that were fixed one by one, with some remaining ones left open.

Once it became clear that numeric inaccuracies are prevalent, it is likewise not reasonable to require that the robustness value as computed during the validation run is *strictly* negative, and for the results presented here, we granted a small, benchmark-dependent tolerance, such that positive values below this tolerance are recognized as falsifications proper, but only if the tool reported a falsification in the first place. The numeric tolerances have been chosen by the organizer as follows, and likely need to be refined for the next competition: AT1: 1.2 (1% of max speed); AT2: 10.0 (ok wrt. the large RPM scale); AT6*: 1.0; AFC27: 0.0008 (10% of β); AFC29, AFC33: 0.0007 (10% of γ); NN, NNx: 0.1 (somewhat loose in comparison to the precision of the magnet) CC1: 1.0; SC: 0.05 (1% of valid range).

Results. The results of the validation runs are reported in Table 4. Entries where the falsification rate *FR* as reported by the participants coincides with the number of validated experiments (column ✓), could be confirmed within the above stated tolerance. Mismatches are highlighted in **bold**. The raw data underlying this evaluation can be downloaded at `https://dx.doi.org/10.5281/zenodo.5651631`, the validator and its configuration is available on GitLab as mentioned.

As explained in Sec. 3, `FalCAuN` evaluates requirements over learned discrete abstractions of the actual system. While this permits for efficient simulations, it also introduces an error such that a falsifying trace for the discrete system is close to a falsifying one on the original model, but misses the mark by a small margin. On the experiments for AT6, the overall mean robustness is approximately 18, which is double the tolerance imposed above, expressing that `FalCAuN` works

[14]`https://gitlab.com/gernst/ARCH-COMP/-/blob/FALS/2021/FALS/Validation.md`

Table 4: Results of validation, falsification rate: *FR*(as reported by participants), ✓ (successfully validated by FalStar), wrt. a given # number of trials (sometimes not 50). Results except for AFC are with instance 1, for instance 2 the falsification rates are different but the validation rates are very similar (not shown since the latter are available for few tools only).

Tool:	**ARIsTEO**			**Breach**			**FalCAuN**			**falsify**			**FalStar**			**ForeSee**			**S-TaLiRo**		
Approach:	ARX-2			GNM						A3C			aLVTS								
Benchmark	*FR*	✓	#	*FR*	✓	#	*FR*	✓	#	*FR*	✓	#	*FR*	✓	#	*FR*	✓	#	*FR*	✓	#
AT1				2	2	2	4	4	50	31	31	50	50	50	50	31	**14**	50			
AT2	50	50	50	2	2	2	8	8	50	50	50	50	50	50	50	46	46	50			
AT51	0	0	50	2	2	2				50	50	50	50	50	50	50	50	50			
AT52	50	50	50	2	2	2				50	50	50	26	26	50	50	50	50			
AT53	50	50	50	2	2	2				50	50	50	50	50	50	50	50	50			
AT54	3	3	50	2	2	2				50	50	50	50	50	50	50	50	50			
AT6a	50	50	50	1	1	2	46	**12**	50	50	50	50	50	50	50	49	**47**	50			
AT6b	49	49	50	1	1	2	49	**0**	50	11	11	50	50	50	50	47	**35**	50			
AT6c	50	50	50	1	1	2	77	**0**	100	0	0	22	8	8	50	50	**48**	50			
AT6abc	50	50	50	1	1	2						50	10	10	50				48	**47**	50
																			†47	47	50
AFC27										50	**33**	50	50	50	50	50	50	50			
AFC29	50	50	50							50	**22**	50	50	50	50	50	50	50			
AFC33	50	50	50							50	**0**	50	50	**0**	50	50	50	50			
NN				2	2	2				91	91	100	23	23	45	50	50	50			
$NN_{\beta\,=\,0.04}$										21	21	100									
NNx	50	50	50	2	2	2							50	50	50				37	37	50
																			†20	20	50
CC1	50	50	50	2	2	2	7	7	50	46	46	50	50	50	50	50	50	50			
CC2	50	50	50	2	2	2	1	1	50	29	29	50	50	50	50	10	10	50			
CC3	50	50	50	2	2	2	4	4	50	50	50	50	50	50	50	50	**47**	50			
CC4	0	0	50	1	1	2				0	0	50	2	2	50	17	**0**	50			
CC5	50	50	50	2	2	2				28	28	50	48	48	50	50	**34**	50			
CCx	50	50	50	2	**0**	2							40	40	50	50	50	50	45	45	50
																			†49	49	50
SC	0	0	50	1	1	1				0	0	50	0	0	50						

The S-TaLiRo results marked with † were obtained with minSOAR [31, 30], whereas the unmarked ones were obtained using SOAR like in the tables from Sec. 4.

well in principle. We remark that often such near-misses are of interest in practice, since they reflect insufficient error margin in the design.

falsify shows some discrepancy for AFC, with a mean of roughly 0.015, significantly exceeping the accepted tolerance accepted as well as the error margin of the requirements, for unclear reasons that we were not able to analyze in time. We remark that the AFC model's dynamics are difficult to integrate numerically with high accuracy (stiff differential equations), such that even small initial perturbations or minor model variations could cause such discrepancies.

The results of FalStar are self-validated very accurately, which is still a meaningful result, because it confirms that no errors are introduced by serializing and re-evaluating the experimental data. We report for most experiments, the error is exactly zero, however, a switch from generating the data with MATLAB 2021a to validating it version 2021b introduced some measurable inaccuracies in the order of 10^{-10} to 10^{-14}, confirming our suspicion that indeed the Matlab version might have an effect, or perhaps inadvertently some model parameters had been changed. For benchmark AFC33 the input signal is somehow considered invalid, the error was found too late during preparation of this report to be investigated and corrected.

According to the participants, the unvalidated results ForeSee for AT1 are due to a reporting error, where the data indicates an excessive number of simulations above the limit of 300 being reported, but the actual number of simulations used could not be determined during validation hence we chose to highlight this result. The other discrepancies should be attributed to a mismatch between the QB-robustness used by ForeSee [41], which balances the magnitudes of input signals against each other. Nevertheless, this requires further investigation as in principle the approach should be sound and therefore should not report wrong results.

S-TaLiRo and Breach both achieve very accurate validation results, with a single defect in the experimental data of S-TaLiRo due to a manual copy & paste error (the actual result is probably correct). The invalid result for Breach is due to a transcription error of the STL requirement.

Other Issues and Findings. A number of other issues was uncovered during validation:

The maximum value of the brake input of the AT model was stated correctly as 325 in previous reports, but incorrectly as 350 in a file `requirements.txt` in the GitLab repository, which was uncovered by checking the input ranges of reported experimental data. This finding is in line with some feedback of participants that more documentation on how to get started would be useful.

Another aspect was the interpretation of the robustness score for the gear value of the AT model, which is supposed to be discrete. Hence, the robustness value of atomic propositions $g1, \dots, g4$ of AT5* must reflect this discrete nature, such that $\rho(gi) < 0$ if the ith gear is currently selected but $\rho(gi) >$ otherwise, with strict inequalities in both cases. Indeed, some preliminary results were wrong due to an issue related to this interpretation. Moreover, some tools return $\pm\infty$ while others ± 1 for AT5*, which needs to be accounted for by the validator (FalStar's robustness is based on the former).

There were several instances of properties being specified wrong. For instance, FalStar's result for AT6c from previous years employed different bounds on the temporal operator and the input signals, which led to unsuccessful validation attempts for the (correct) results of other tools, as the validator inherited this misconfiguration.

Finally, the time-series input signals reported by the participants came with different sampling schemes (e.g. finely-sampled with values each <0.1 time-points, coarsely-sampled at each control point), which requires that interpolation settings inside the validator correctly match those of the tools. The conventions are now documented, alternatively participants resorted to reporting finely sampled inputs, which solved numerous issues in one tool.

6 Conclusion and Outlook

The benchmark set established by this competition appears to gain traction in the falsification literature, underpinning the results and experiments that are being published. Publishing such results as part of the competition report sets a reference and gold-standard against which such efforts can compare. It is therefore great to again have new participants this year.

Clearly, the approaches implemented in the participating tools have become even more diverse, and a comparison on even footing and for a full set of results becomes more difficult. Nevertheless, the results reported in tables 2 and 3 already give a good overview of the strengths and weaknesses of the respective approaches. It has become apparent that no "best" approach can be singled out.

The effort to validate the falsifying inputs generated by the tools proved to be more involved than initially thought. A non-negligible part of that effort was to establish the technical basis in terms of exchange formats and a validator tool. There are plenty of sources of error—by the human and computational—which each of its own warrants dedicated investigation. While many big and systematic issues could be resolved, some more subtle ones remain.

Overall, our findings stress the importance of validating experimental data, especially in a well-defined comparative setting. This experience is shared with other competitions like SV-COMP (which has validation since 2016 [5]), Test-Comp (which had independent coverage evaluation from the start in 2019 [6]), and many other competitions (for an overview, see [3]).

For the future, we want to address two issues. From previous years, we inherit the goal of a larger benchmark set with more complex and higher-dimensional models. Moreover, many steps in the evaluation that are now manual should be automated, including running the tools, taking measurements, producing the validation witness files, checking them, and formatting a summary of the experimental data. With this kind of automation, it will be much easier to obtain a consistent and complete set of results in a repeatable way.

Acknowledgments Many thanks to the organizers of the ARCH workshop 2021 for hosting this competition and for providing a supportive and friendly environment. The organizer thanks all participants for their time and patience during investigation of the discrepancies found during validation. P. Arcaini is supported by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST. Funding Reference number: 10.13039/501100009024 ERATO. The ASU team (S-TaLiRo + SOAR) was partially supported by DARPA FA8750-20-C-0507, NSF CNS 1932068, NSF CMMI 2046588, NSF CNS 2000792, NSF CMMI 1829238, NSF IIP-1361926 and the NSF I/UCRC Center for Embedded Systems. Z. Zhang is supported by JSPS KAKENHI Grant No. 20H04168, 19K24348, 19H04086, JST-Mirai Program Grant No. JPMJMI20B8, Japan. K. Gaaloul and C. Menghi are supported by the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant No 694277). M. Waga is partially supported by JST ACT-X Grant Number JPMJAX200U and JST CREST Grant Number JPMJCR2012, Japan. `zlscheck` was funded by the ModeliScale Inria Project Lab.

References

[1] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 254–257. Springer, 2011.

[2] 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, 2005.

[3] Ezio Bartocci, Dirk Beyer, Paul E Black, Grigory Fedyukovich, Hubert Garavel, Arnd Hartmanns, Marieke Huisman, Fabrice Kordon, Julian Nagele, Mihaela Sighireanu, et al. Toolympics 2019: An overview of competitions in formal methods. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 3–24. Springer, 2019.

[4] Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donzé, Georgios Fainekos, Oded Maler, Dejan Ničković, and Sriram Sankaranarayanan. Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. In *Lectures on Runtime Verification*, pages 135–175. Springer, 2018.

[5] Dirk Beyer. Reliable and reproducible competition results with benchexec and witnesses (report on sv-comp 2016). In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 887–904. Springer, 2016.

[6] Dirk Beyer. International competition on software testing (test-comp). In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 167–175. Springer, 2019.

[7] Timothy Bourke and Marc Pouzet. Zélus: A synchronous language with ODEs. In *16th International Conference on Hybrid Systems: Computation and Control (HSCC'13)*, pages 113–118, Philadelphia, USA, March 2013.

[8] Jean-Louis Colaço, Bruno Pagano, and Marc Pouzet. Scade 6: A Formal Language for Embedded Critical Software Development. In *TASE 2017 - 11th International Symposium on Theoretical Aspects of Software Engineering*, pages 1–10, Nice, France, September 2017.

[9] Anthony Corso, Robert J Moss, Mark Koren, Ritchie Lee, and Mykel J Kochenderfer. A survey of algorithms for black-box safety validation. *arXiv preprint arXiv:2005.02979*, 2020.

[10] Adel Dokhanchi, Shakiba Yaghoubi, Bardh Hoxha, and Georgios Fainekos. ARCH-COMP17 category report: Preliminary results on the falsification benchmarks. In Goran Frehse and Matthias Althoff, editors, *ARCH17. 4th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 48 of *EPiC Series in Computing*, pages 170–174. EasyChair, 2017.

[11] 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 *ARCH@ ADHS*, pages 104–109, 2018.

[12] 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 Goran Frehse, editor, *ARCH18. 5th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 54 of *EPiC Series in Computing*, pages 104–109. EasyChair, 2018.

[13] Adel Dokhanchi, Shakiba Yaghoubi, Bardh Hoxha, and Georgios E Fainekos. Arch-comp17 category report: Preliminary results on the falsification benchmarks. In *ARCH@ CPSWeek*, pages 170–174, 2017.

[14] Alexandre Donzé. Breach, A toolbox for verification and parameter synthesis of hybrid systems. In *International Conference on Computer Aided Verification (CAV 2010)*, LNCS, pages 167–170. Springer, 2010.

[15] Johan Liden Eddeland, Alexandre Donze, Sajed Miremadi, and Knut Akesson. Industrial temporal logic specifications for falsification of cyber-physical systems. In *ARCH@CPSIoTWeek*, 2020.

[16] G. Ernst, S. Sedwards, Z. Zhang, and I. Hasuo. Falsification of hybrid systems using adaptive probabilistic search. *Transactions on Modeling and Computer Simulations (TOMACS)*, 2021. Accepted.

[17] Gidon Ernst, Paolo Arcaini, Ismail Bennani, Alexandre Donze, Georgios Fainekos, Goran Frehse, Logan Mathesen, Claudio Menghi, Giulia Pedrinelli, Marc Pouzet, et al. Arch-comp 2020 category report: Falsification. *EPiC Series in Computing*, 2020.

[18] 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. In *ARCH@CPSIoTWeek*, pages 129–140, 2019.

[19] Gidon Ernst, Sean Sedwards, Zhenya Zhang, and Ichiro Hasuo. Fast falsification of hybrid systems using probabilistically adaptive input. *arXiv preprint arXiv:1812.04159*, 2018.

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

[21] Martin Fränzle and Michael R Hansen. A robust interpretation of duration calculus. In *International Colloquium on Theoretical Aspects of Computing*, pages 257–271. Springer, 2005.

[22] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, 1991.

[23] Nicolas Halbwachs, Fabienne Lagnier, and Pascal Raymond. Synchronous observers and the verification of reactive systems. In *Proceedings of the Third International Conference on Methodology and Software Technology: Algebraic Methodology and Software Technology*, AMAST '93, pages 83–96, Berlin, Heidelberg, 1994. Springer-Verlag.

[24] Peter Heidlauf, Alexander Collins, Michael Bolender, and Stanley Bak. Verification challenges in f-16 ground collision avoidance and other automated maneuvers. In Goran Frehse, editor, *ARCH18. 5th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 54 of *EPiC Series in Computing*, pages 208–217. EasyChair, 2018.

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

[26] Jianghai Hu, John Lygeros, and Shankar Sastry. Towards a theory of stochastic hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 160–173. Springer, 2000.

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

[28] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, Nov 1990.

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

[30] Logan Mathesen. *Making Bayesian Optimization Practical in the Context of High Dimensional, Highly Expensive, Black-Box Functions.* PhD thesis, Arizona State University, 2021.

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

[32] Logan Mathesen, Giulia Pedrielli, Szu Hui Ng, and Zelda B Zabinsky. Stochastic optimization with adaptive restart: A framework for integrated local and global learning. *Journal of Global Optimization*, 79(1):87–110, 2021.

[33] Claudio Menghi, Shiva Nejati, Lionel Briand, and Yago Isasi Parache. Approximation-refinement

testing of compute-intensive cyber-physical models: An approach based on system identification. In *International Conference on Software Engineering (ICSE)*. IEEE / ACM, 2020.

[34] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1928–1937, 2016.

[35] Doron A. Peled, Moshe Y. Vardi, and Mihalis Yannakakis. Black box checking. In Jianping Wu, Samuel T. Chanson, and Qiang Gao, editors, *Formal Methods for Protocol Engineering and Distributed Systems, FORTE XII / PSTV XIX'99, IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX), October 5-8, 1999, Beijing, China*, volume 156 of *IFIP Conference Proceedings*, pages 225–240. Kluwer, 1999.

[36] Zahra Ramezani, Alexandre Donzé, Martin Fabian, and Knut Åkesson. Temporal logic falsification of cyber-physical systems using input pulse generators. In *ARCH21@IFAC*, 2021.

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

[38] Masaki Waga. Falsification of cyber-physical systems with robustness-guided black-box checking. In Aaron D. Ames, Sanjit A. Seshia, and Jyotirmoy Deshmukh, editors, *HSCC '20: 23rd ACM International Conference on Hybrid Systems: Computation and Control, Sydney, New South Wales, Australia, April 21-24, 2020*, pages 11:1–11:13. ACM, 2020.

[39] Shakiba Yaghoubi and Georgios Fainekos. Gray-box adversarial testing for control systems with machine learning components. In *International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2019.

[40] Yoriyuki Yamagata, Shuang Liu, Takumi Akazaki, Yihai Duan, and Jianye Hao. Falsification of cyber-physical systems using deep reinforcement learning. *IEEE Transactions on Software Engineering*, 2020.

[41] Zhenya Zhang, Deyun Lyu, Paolo Arcaini, Lei Ma, Ichiro Hasuo, and Jianjun Zhao. Effective hybrid system falsification using monte carlo tree search guided by QB-robustness. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification*, pages 595–618, Cham, 2021. Springer International Publishing.