Falsification of Cyber-Physical Systems Using PDDL+ Planning

Diego Aineto[1], **Enrico Scala**[1], **Eva Onaindia**[2], **Ivan Serina**[1]

[1] Università degli Studi di Brescia
[2] VRAIN, Universitat Politècnica de València
diego.ainetogarcia@unibs.it, enrico.scala@unibs.it, onaindia@upv.es, ivan.serina@unibs.it

Abstract

This work explores the capabilities of current planning technologies to tackle the falsification of safety requirements for cyber-physical systems. Cyber-physical systems are systems where software and physical processes interact over time, and their requirements are commonly specified in temporal logic with time bounds. Roughly, falsification is the process of finding a trajectory of the cyber-physical system that violates the safety requirements, and it is a task typically tackled with black-box algorithms. We analyse the challenges posed by industry-driven falsification benchmarks taken from the ARCH-COMP competition, and propose a first attempt to deal with these problems through PDDL+ planning instead. Our experimental analysis on a selection of these problems provides empirical evidence on the feasibility and effectiveness of planning-based approaches, whilst also identifying the main areas of improvement.

Introduction

This work explores the capabilities of current planning technologies for tackling safety validation of cyber-physical systems (CPS). A CPS is a system where software and physical processes interact over time. We focus on the task of safety validation through *falsification* of *safety properties*. A safety property specifies that some undesired event will never happen, and can be shown to be violated with a *counterexample*, also known as *falsifying trajectory*. Finding such counterexamples is the primary goal of the falsification task. While the connection between planning and model checking is well-established (Cimatti et al. 1997; Bogomolov et al. 2014), little work has been done to understand how planning can be employed for falsification and it is somewhat limited to simple models and requirements (Plaku, Kavraki, and Vardi 2013), most likely due to a scarce presence of efficient planning systems that can deal with CPS. With this paper we aim to provide a first step toward filling this gap.

Falsification benchmarks are heavily influenced by industry standards. Consequently, CPS are modelled in the MATLAB[1] environment which is widely used for prototyping. As an example, consider the automatic transmission system (Hoxha, Abbas, and Fainekos 2014) shown in Figure 1.

[1]https://www.mathworks.com/products/matlab.html

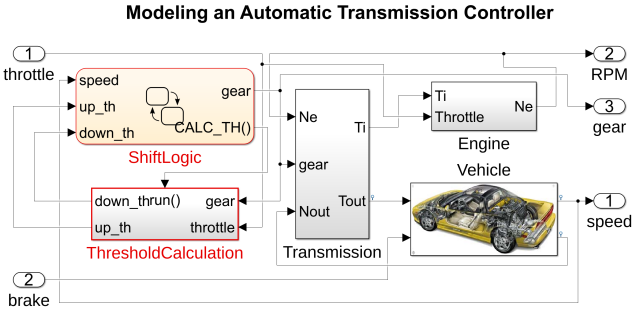

Figure 1: Matlab model of an automatic transmission.

The model for this system encompasses the dynamics of the engine, transmission and wheels of a vehicle, as well as the logic for the automatic transmission which selects gears based on throttle and brake inputs and other state information. A typical way to express safety requirements in CPS is to use temporal logic with time bounds such as Signal Temporal Logic (STL) (Donzé and Maler 2010). For instance, the STL requirement $\varphi_1 : \Box_{[0,15]} v < 50$ specifies that the system is safe if and only if the velocity is always less than 50mph over the closed interval that goes from 0s to 15s. Figure 2 shows an input signal (top) that steers the system into a trajectory (bottom) that falsifies requirement φ_1, which simply requires maintaining enough throttle.

Falsification algorithms leverage temporal logic robustness (Fainekos and Pappas 2009), a metric of the degree to which a system trajectory satisfies a safety requirement, to pose the falsification task as an optimization problem guided by this metric. This problem is then solved using standard optimization techniques (Annpureddy et al. 2011; Donzé 2010), or reformulated into reinforcement learning (Zhang et al. 2018; Akazaki et al. 2018) and path planning problems (Dreossi et al. 2015). As there is a widespread belief that CPS are too complex to explicitly reason about (Corso et al. 2021), falsification algorithms follow the *black-box assumption* by considering the CPS as a mapping from inputs to outputs (e.g. embedding the model in a simulator).

In this work, we set out to evaluate the performance of declarative, white-box planning algorithms for falsification problems. We start by analysing the industry-driven benchmarks proposed in the falsification category of the ARCH-

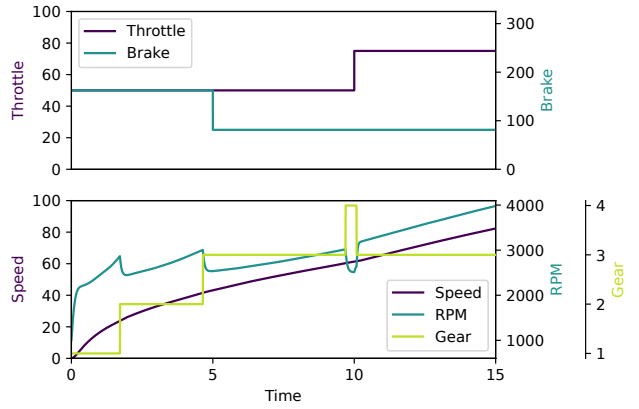

Figure 2: Input (top) and output (bottom) signals of an automatic transmission system.

COMP competition and identifying the main challenges of the reformulation into planning. Afterwards, we evaluate our proposal against a state-of-the-art falsification tool, providing empirical evidence for the feasibility of planning-based approaches and discussing the main areas of improvement. Our work incentivizes cross-fertilization between the falsification and planning communities by establishing a connection between these two problems, and seeks to spur research on PDDL+ Planning by extending the pool of benchmarks and identifying new challenges.

Finding Falsifying Input Signals for STL Requirements Using PDDL+ Planning

A falsification problem consists of a system model $\mathcal{M}$ and a safety requirement φ that the system should satisfy. Safety properties are defined over state trajectories $\mathbf{s} = [s_0, \ldots, s_t]$ where s_t is the state of the system at time t. Falsification is about finding an input signal $\mathbf{u}$ that causes the outputs of the system to violate the safety requirement φ: $\mathbf{u}$ s.t. $\mathcal{M}(\mathbf{u}) \not\models \varphi$

Falsification of CPS in the ARCH-COMP

The general definition of falsification takes a more concrete form in the ARCH-COMP competition (Ernst et al. 2021), where the system model $\mathcal{M}$ is defined as a Simulink/Stateflow model, the safety requirement φ is specified using STL, and the input signals have continuous domains and shape constraints (e.g. piece-wise constant).

Simulink/Stateflow models: Following industry standards, falsification benchmarks are defined in the MATLAB environment as Simulink/Stateflow models. Simulink uses a block diagram representation where each block has different semantics, from basic arithmetic operators to complex subsystems. Stateflow extends Simulink with an automata-like representation based on states connected by transitions; this is commonly used to represent decision logic and hybrid systems that combine symbolic and numeric reasoning. Overall, a Simulink/Stateflow model describes a CPS as a set of differential equations modelling the physical aspect of the system and a set of hybrid automata (Henzinger 2000) modelling the software component. Further, ARCH-COMP benchmarks are all deterministic, so under the same inputs they always generate the same output trajectory.

STL safety requirements: STL is a temporal logic with time bounds for real-valued signals over a continuous timeline. STL predicates are arbitrary functions of the form $\mu(s_i) \geq 0$ mapping a signal s_i from $\mathbb{R}^n$ to $\mathbb{R}$. STL requirements are specified by combining predicates with Boolean operators (*conjunction* $\wedge$, *disjunction* $\vee$, and *negation* $\neg$) and temporal operators (*always* $\Box$, *eventually* $\Diamond$, *next* $\circ$, and *until* $\mathcal{U}$). STL temporal operators can be indexed with a time interval over which the property is expected to hold. For example, the STL requirement $\Box_{[0,30]}((\neg g1 \wedge \circ g1) \rightarrow \circ \Box_{[0,2.5]} g1)$ specifies that *within 30s, after changing to first gear, there should be no shift to any other gear before 2.5s*.

Input signal: In principle, a solution to the falsification problem can be any arbitrary piece-wise continuous input signal. However, the ARCH-COMP competition recommends fixing the format of the solutions to make comparisons fairer. The proposed format is piece-wise constant signals with a fixed number of equally spaced *control points*. A control point is a point in time where falsification algorithms decide new values for the input variables, possibly causing a discontinuity in the piece-wise constant signal. As an example, the input signal shown in Figure 2 (top) follows this format with control points allowed every 5s that cause the discontinuities observed at the 5s and 10s marks.

Falsification of CPS as PDDL+ Planning

A standard methodology to verify that a system S modeled by an automaton $\mathcal{A}_s$ satisfies a requirement φ is to derive an automaton $\mathcal{A}_{\neg\varphi}$ accepting exactly trajectories that violate φ and check that the language of the product automaton $\mathcal{A}_S \times \mathcal{A}_{\neg\varphi}$ is empty. Our reformulation into planning follows this methodology to build a planning problem $P(\mathcal{M}, \neg\varphi)$ whose solutions are plans inducing counterexample trajectories.

Due to the features of Simulink/Stateflow models, we look at PDDL+ (Planning Domain Definition Language) as the language of our reformulation. PDDL+ is a high-level declarative language to formulate planning problems that require reasoning over a mixture of discrete and continuous variables over a continuous timeline (Fox and Long 2006). PDDL+ explicitly differentiates between Boolean and numeric variables, and supports three types of state transitions, namely, *actions*, *events* and *processes*. Actions and events model instantaneous transitions causing discrete changes and numeric discontinuities. The difference between the two is that the execution of an action is decided by the planner while events happen the instant their conditions are met. Processes model continuous transitions through time-dependent effects of the form $\dot{x} = \xi$ (ξ represents the first derivative of x w.r.t. time) and last for as long as their conditions are met; as events, processes are not in control of the agent, but represent the natural evolution of the environment. A PDDL+ plan is a set of timed actions plus the ending time $t_e \in \mathbb{R}^{\geq 0}$ at which the goal is satisfied. Intuitively, we reformulate the falsification problem as a planning problem $P(M, \neg\varphi)$, where the CPS and the negated STL requirement, i.e., $\neg\varphi$,

are modeled through events and processes, and actions are the responsible for picking the values of the input signals such that a solution plan to $P(M, \neg\varphi)$ induces a falsifying trajectory that satisfies $\neg\varphi$, that is, it violates φ.

The general translation scheme we followed to transform Simulink/Stateflow models into PDDL+ associates one process to each differential equation and leverages known results on the relation between PDDL+ and hybrid automata (Fox and Long 2006; Bryce et al. 2015; Bogomolov et al. 2015). Broadly, automata locations and variables map into PDDL+ Boolean and numeric variables, respectively, discrete transitions are represented using events and the differential equations associated to each location using processes. Moreover, since ARCH-COMP benchmarks also feature piece-wise linear functions through look-up tables, we captured these dynamics through the use of (bi-)linear interpolation inside the time-dependent effects of the associated processes. Overall, our experience suggests that it is possible to build a general translation algorithm from Simulink/Stateflow models to PDDL+ accounting for a large fragment of the supported dynamics. Such an algorithm would immediately amplify the reach of planning.

Regarding the STL requirements, translations for temporal logic with time bounds target variants of timed automata that make use of clock variables to enforce duration constraints. Timed automata are a subclass of hybrid automata and can be encoded in PDDL+ following the same mapping. In principle, the resulting timed automaton can be nondeterministic requiring the use of actions for its encoding in PDDL+. In a dense time setting, this can lead to a blow up of the branching factor so we restrict our attention to safety requirements that can be translated to deterministic timed automata. As a future work to overcome this limitation, we identify a recent result by Ferrere et al. (2019) that presents a novel compositional translation.

The last important bit of this reformulation is handling input variables with continuous domains. Inspired by the practical ideas used in falsification (Ernst et al. 2019), we parameterize the reformulated problem $P_k(\mathcal{M}, \neg\varphi)$ with a *discretization level* $k \in \mathbb{N}$, a constant used to transform a continuous domain $[a, b]$ into a discrete domain $\{i(b-a)/k+a \mid 0 \leq i \leq k\}$ that can be sampled using actions. This means that, for a variable with domain $[0, 100]$, P_1 would only consider values in $\{0, 100\}$ while a P_4 would consider $\{0, 25, 50, 75, 100\}$. While this approach in isolation is incomplete, it can be coupled with a meta-algorithm that iteratively increases k, resulting in a systematic sampling of the inputs that is asymptotically complete.

Experimental Analysis

We evaluated the performance of the proposed planning-based approach to solve falsification benchmarks (https://github.com/daineto/falsification_benchmarks). In order to establish a baseline for comparisons, we took Falstar (https://github.com/ERATOMMSD/falstar), one of the best performers of the ARCH-COMP. Falstar works by incrementally exploring alternative falsifying input signals using an adaptive search that samples the input domain with increasingly finer discretizations. This systematic exploration naturally scales with the hardness of the problem and makes for a better baseline than random sampling.

Req.	STL formula
AT1	$\Box_{[0,20]} v < 120$
AT2	$\Box_{[0,10]} w < 4750$
AT51	$\Box_{[0,30]}((\neg g1 \wedge \circ g1) \rightarrow \circ \Box_{[0,2.5]} g1)$
AT52	$\Box_{[0,30]}((\neg g2 \wedge \circ g2) \rightarrow \circ \Box_{[0,2.5]} g2)$
AT53	$\Box_{[0,30]}((\neg g3 \wedge \circ g3) \rightarrow \circ \Box_{[0,2.5]} g3)$
AT54	$\Box_{[0,30]}((\neg g4 \wedge \circ g4) \rightarrow \circ \Box_{[0,2.5]} g4)$
AT6a	$(\Box_{[0,30]} w < 3000) \rightarrow (\Box_{[0,4]} v < 35$
AT6b	$(\Box_{[0,30]} w < 3000) \rightarrow (\Box_{[0,8]} v < 50$
AT6c	$(\Box_{[0,30]} w < 3000) \rightarrow (\Box_{[0,20]} v < 65$
AT6$\wedge$	AT6a $\wedge$ AT6b $\wedge$ AT6c
AT5$\vee$	$\bigvee_{i=1..4} \Box_{[0,50]}((\neg g_i \wedge \circ g_i) \rightarrow \circ \Box_{[0,2.5]} g_i)$
AT6$\vee$	AT6a $\vee$ AT6b $\vee$ AT6c
CC1	$\Box_{[0,100]} y_5 - y_4 \leq 40$
CC2	$\Box_{[0,70]} \Diamond_{[0,30]} y_5 - y_4 \geq 15$
CC3	$\Box_{[0,80]}((\Box_{[0,20]} y_2 - y_1 \leq 20) \vee (\Diamond_{[0,20]} y_5 - y_4 \geq 40))$
CC4	$\Box_{[0,65]} \Diamond_{[0,30]} \Box_{[0,5]} y_5 - y_4 \geq 8$
CC5	$\Box_{[0,72]} \Diamond_{[0,8]} (\Box_{[0,5]} y_2 - y_1 \geq 9 \rightarrow \Box_{[5,20]} y_5 - y_4 \geq 9)$
CCx	$\bigwedge_{i=1..4} \Box_{0,50}(y_{i+1} - y_i > 7.5)$
CCy	$\bigvee_{i=1..4} \Box_{0,50}(y_{i+1} - y_i \leq 100)$
CCz	$\Box_{0,50}(\bigvee_{i=1..4} y_{i+1} - y_i \leq 100)$

Table 1: STL safety requirements for the benchmarks.

Models and Requirements

We used two benchmark models from the ARCH-COMP competition with very different features. The *automatic transmission* benchmark presents complex dynamics with large differential equations but relatively simple STL requirements. The *chasing cars* benchmark has simple dynamics but complex STL requirements with more levels of nesting and longer time intervals.

Automatic Transmission (AT): This is the benchmark model we used throughout the paper. The system has two inputs: throttle and brake with ranges from 0 to 100 and 0 to 325, respectively. The output is two continuous-time variables representing the physical state of the system, the speed of the engine w given as revolutions per minute (RPM) and the speed of the vehicle v (mph), as well as the transmission gear g. The vehicle is initially at rest ($v = 0$ and $w = 0$) and on first gear ($g = 1$). The Simulink/Stateflow model for the automatic transmission domain consists of 69 blocks including 2 integrators (one for each continuous-time variable), 6 look-up tables defining piece-wise linear functions and a Stateflow chart with two concurrent automata modelling the logic for changing gears.

Chasing Cars (CC): This model describes a platooning scenario (Hu, Lygeros, and Sastry 2000) consisting of five cars where the lead car is driven by throttle and brake inputs, and the other four are driven by a deterministic controller that decides among different modes: maintain speed, accelerate or brake, based on the distance to the car ahead. The system's outputs are the positions y_1, y_2, y_3, y_4 and y_5 of the five cars. Safety requirements specify maximum and minimum distance thresholds between cars. Initially, all vehicles are at rest and separated by 10 distance units. The

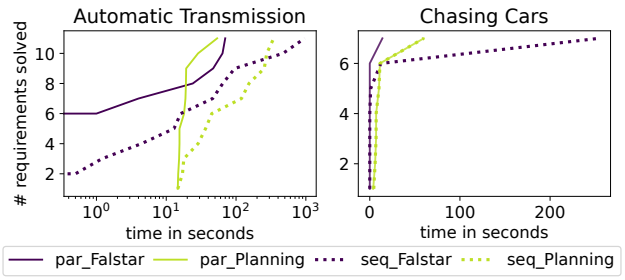

Figure 3: Coverage with respect to runtime.

Simulink/Stateflow model consists of 5 concurrent hybrid automata, one for each car. The lead car automaton has only one single location while the controller for the chasing cars has three locations, one for each mode.

Safety requirements associated to each of these two models are listed in Table 1. We consider all requirements proposed by the ARCH-COMP and add four new disjunctive instances, namely, AT5$\vee$. AT6$\vee$, CCy and CCz.

Results

Falsification tools such as Falstar are stochastic so they are evaluated over several trials. In contrast, our planning-based approach is deterministic but we must solve a different planning instance for each input discretization level. We run Falstar with the ARCH-COMP settings and a limit of 10 trials, while we run our approach with 6 discretization levels $\{1, 2, 4, 8, 16, 32\}$. Our PDDL+ planning approach is WA* search guided by the h^{MRP}_{max} heuristic (Scala et al. 2020), obtained from ENHSP-20 (https://gitlab.com/enricos83/ENHSP-Public). In order to comply with the ARCH-COMP specifications, we set the planning delta to 5s and the execution delta to 0.001s.

We evaluated coverage of Falstar and our PDDL+ solution. We run both approaches using two configurations, sequential and parallel, with a time budget of 100s to solve each Falstar trial or planning instance. The sequential configuration launches a new trial or instance (with increased discretization level) when the time budget is consumed until a solution is found. For the parallel configuration, we run all trials and instances in parallel and report the minimum time across solved ones. We account for the stochastic nature of Falstar by using the expected time to the first successful trial E_s/fr where E_s is the expected time of a single trial and fr is the *falsification rate* (ratio of successful trials).

The results of this experiment are shown in Figure 3 where each curve relates the number of solved problems with run-time. Falstar is able to falsify some requirements almost instantly, but then struggles with the harder ones. In contrast, our PDDL+ approach takes a few seconds to see the first solutions (partly due to preprocessing) but it converges faster. Overall, the PDDL+ approach proves to be very competitive, achieving the coverage of Falstar. Table 2 presents a detailed view of the performance of both approaches across all STL requirements; to have some indication on how the planner behaves, we show the results for three input discretizations (columns P_2, P_8, P_{32}). Interestingly, most problems can be falsified with a small discretization level, indicating that safety requirements are often falsified by extreme values. Besides CC5, for which we did not manage to build a deterministic automaton, we falsify all chasing cars requirements but are slower than Falstar. We attribute this to h^{MRP}_{max} being quite uninformed for this class of problems, in particular those having long horizon solutions. On the other hand, Falstar was only able to falsify 2 of the new disjunctive requirements (AT6$\vee$ and CCy) with a relatively large number of simulations. Via planning we solve AT5$\vee$ that requires a specific sequence of gear operations to falsify, something that seems difficult for black-box algorithms such as Falstar. Notably, the planner proves the system safe for some discretization levels (the unsolvable results), something beyond the capabilities of standard falsification techniques.

	P_2			P_8			P_{32}			Falstar		
Req.	FR	E	T	FR	E	T	FR	E	T	FR	S	T
AT1	1	343	91.3	0	-	-	0	-	-	7	177.0	53
AT2	1	19	17.9	1	344	80.5	0	-	-	10	3.7	0.5
AT51	1	303	104	1	1k	255	0	-	-	10	16.6	4.4
AT52	1	5	14.8	1	324	91.9	0	-	-	10	5.8	1.2
AT53	1	4	15.5	1	19	19.8	1	9k	1550	10	1.7	0
AT54	1	16	19.5	1	113	45.2	1	8k	1398	10	55.7	16.1
AT6a	1	12	20.8	1	33	27.5	1	11	18.9	10	43.8	12.9
AT6b	*	45	28.6	1	431	122	1	11	19.3	2	236.0	70.5
AT6c	*	720	193	1	18	21.4	1	11	19.1	1	212.0	64
AT6$\wedge$	1	12	16.5	1	18	16.9	1	11	15.5	9	112.7	35.1
AT5$\vee$	1	1k	165.3	0	-	-	0	-	-	0	-	-
AT6$\vee$	*	18	15.3	*	243	68.9	1	11	15.3	10	211.9	64.5
CC1	1	266	13.8	1	12k	108	0	-	-	10	1.5	0
CC2	1	285	13.8	1	9k	101	0	-	-	10	1.3	0
CC3	1	1k	28.9	1	56k	461	1	87k	686	10	79.7	12.3
CC4	1	43k	434	0	-	-	0	-	-	10	1.3	0
CC5										10	1.0	0
CCx	1	14	3.8	1	15	4.5	1	15	4.3	10	1.0	1
CCy	1	1k	24.8	1	151k	1330	1	11k	107	3	226.7	21
CCz	1	5k	53	0	-	-	0	-	-	0	-	-

Table 2: FR: Falsification rate (out of 1 for planning and out of 10 for Falstar); * denotes a problem proved unsolvable. E: Expanded nodes. S: Average simulations. T: Time in secs.

Discussion

This paper showed how industrially-driven benchmarks for CPS can be handled by a PDDL+ approach using an off-the-shelf planner. Although the planner proves the approach feasible, our study also reveals a number of avenues for future research. First, from our study it emerged quite clearly that current domain-independent heuristics are either too slow (e.g., the AIBR heuristic by Scala et al. (2016)) or little informed (e.g., the h^{MRP}_{max} is blind to non simple dynamics). In our experiments, the problems are mainly solved due to an efficient search rather than strong guidance but this shows its limits when many decisions must be taken over time. Secondly, we want to study principled ways to handle general STL formulas, and continuous inputs inside the planner. Finally, we would also like to explore the possibility of applying falsification algorithms to PDDL+ benchmarks.

Acknowledgements

We thank all the anonymous reviewers for their insightful comments. This work has been partially supported by EU-H2020 projects AIPlan4EU (No. 101016442), and by he Department of Excellence Fund 2018-2022 of the Department of Information Engineering of the University of Brescia.

References

Akazaki, T.; Liu, S.; Yamagata, Y.; Duan, Y.; and Hao, J. 2018. Falsification of cyber-physical systems using deep reinforcement learning. In *International Symposium on Formal Methods*, 456–465. Springer.

Annpureddy, Y.; Liu, C.; Fainekos, G.; and Sankaranarayanan, S. 2011. 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*, 254–257. Springer.

Bogomolov, S.; Magazzeni, D.; Minopoli, S.; and Wehrle, M. 2015. PDDL+ Planning with Hybrid Automata: Foundations of Translating Must Behavior. In *ICAPS*, 42–46. AAAI Press.

Bogomolov, S.; Magazzeni, D.; Podelski, A.; and Wehrle, M. 2014. Planning as model checking in hybrid domains. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28.

Bryce, D.; Gao, S.; Musliner, D. J.; and Goldman, R. P. 2015. SMT-Based Nonlinear PDDL+ Planning. In *AAAI*, 3247–3253. AAAI Press.

Cimatti, A.; Giunchiglia, E.; Giunchiglia, F.; and Traverso, P. 1997. Planning via model checking: A decision procedure for AR. In *Recent Advances in AI Planning: 4th European Conference on Planning, ECP'97 Toulouse, France, September 24–26, 1997 Proceedings 4*, 130–142. Springer.

Corso, A.; Moss, R.; Koren, M.; Lee, R.; and Kochenderfer, M. 2021. A survey of algorithms for black-box safety validation of cyber-physical systems. *Journal of Artificial Intelligence Research*, 72: 377–428.

Donzé, A. 2010. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *International Conference on Computer Aided Verification*, 167–170. Springer.

Donzé, A.; and Maler, O. 2010. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*, 92–106. Springer.

Dreossi, T.; Dang, T.; Donzé, A.; Kapinski, J.; Jin, X.; and Deshmukh, J. V. 2015. Efficient guiding strategies for testing of temporal properties of hybrid systems. In *NASA Formal Methods Symposium*, 127–142. Springer.

Ernst, G.; Arcaini, P.; Bennani, I.; Chandratre, A.; Donzé, A.; Fainekos, G.; Frehse, G.; Gaaloul, K.; Inoue, J.; Khandait, T.; et al. 2021. ARCH-COMP 2021 Category Report: Falsification with Validation of Results. In *ARCH@ ADHS*, 133–152.

Ernst, G.; Sedwards, S.; Zhang, Z.; and Hasuo, I. 2019. Fast falsification of hybrid systems using probabilistically adaptive input. In *International Conference on Quantitative Evaluation of Systems*, 165–181. Springer.

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

Ferrere, T.; Maler, O.; Ničković, D.; and Pnueli, A. 2019. From real-time logic to timed automata. *Journal of the ACM (JACM)*, 66(3): 1–31.

Fox, M.; and Long, D. 2006. Modelling Mixed Discrete-Continuous Domains for Planning. *J. Artif. Intell. Res.*, 27: 235–297.

Henzinger, T. A. 2000. The Theory of Hybrid Automata. In *Verification of Digital and Hybrid Systems*, 265–292. Springer Berlin Heidelberg.

Hoxha, B.; Abbas, H.; and Fainekos, G. 2014. Benchmarks for Temporal Logic Requirements for Automotive Systems. *ARCH@ CPSWeek*, 34: 25–30.

Hu, J.; Lygeros, J.; and Sastry, S. 2000. Towards a theory of stochastic hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, 160–173. Springer.

Plaku, E.; Kavraki, L. E.; and Vardi, M. Y. 2013. Falsification of LTL safety properties in hybrid systems. *International Journal on Software Tools for Technology Transfer*, 15(4): 305–320.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramírez, M. 2016. Interval-Based Relaxation for General Numeric Planning. In *ECAI 2016 - 22nd European Conference on Artificial Intelligence*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, 655–663. IOS Press.

Scala, E.; Saetti, A.; Serina, I.; and Gerevini, A. E. 2020. Search-Guidance Mechanisms for Numeric Planning Through Subgoaling Relaxation. In *ICAPS*.

Zhang, Z.; Ernst, G.; Sedwards, S.; Arcaini, P.; and Hasuo, I. 2018. Two-layered falsification of hybrid systems guided by monte carlo tree search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11): 2894–2905.