Falsification of Cyber-Physical Systems Using Deep Reinforcement Learning

Yoriyuki Yamagata, Shuang Liu, Takumi Akazaki, Yihai Duan, Jianye Hao

Abstract—A *Cyber-Physical System* (CPS) is a system which consists of software components and physical components. Traditional system verification techniques such as model checking or theorem proving are difficult to apply to CPS because the physical components have infinite number of states. To solve this problem, robustness guided falsification of CPS is introduced. Robustness measures how robustly the given specification is satisfied. Robustness guided falsification tries to minimize the robustness by changing inputs and parameters of the system. The input with a minimal robustness (counterexample) is a good candidate to violate the specification. Existing methods use several optimization techniques to minimize robustness. However, those methods do not use temporal structures in a system input and often require a large number of simulation runs to the minimize robustness. In this paper, we explore state-of-the-art *Deep Reinforcement Learning* (DRL) techniques, i.e., *Asynchronous Advantage Actor-Critic* (A3C) and *Double Deep Q Network* (DDQN), to reduce the number of simulation runs required to find such counterexamples. We theoretically show how robustness guided falsification of a safety property is formatted as a reinforcement learning problem. Then, we experimentally compare the effectiveness of our methods with three baseline methods, i.e., random sampling, cross entropy and simulated annealing, on three well known CPS systems. We thoroughly analyse the experiment results and identify two factors of CPS which make DRL based methods better than existing methods. The most important factor is the availability of the system internal dynamics to the reinforcement learning algorithm. The other factor is the existence of learnable structure in the counterexample.

Index Terms—Robustness guided falsification, CPS, Reinforcement Learning

1 Introduction

A *Cyber-Physical Systems* (CPS) is a system which consists of software components and physical components. Many modern systems, such as automotive vehicles, railway systems and automated factories are CPS. As more and more CPSs are deployed to safety critical domains, detecting defects in a CPS becomes more and more important. However, traditional engineering focuses either on a software component or a physical component, and cannot handle interactions between software and physical components. Testing methods have been explored to guarantee the correctness of CPS models. However, it is hard for testing to achieve a high coverage, and thus provide guarantee with high confidence, due to the infinite state space involved. For software systems, formal methods such as model checking or theorem proving are often explored to thoroughly verify the system. However, model checking is undecidable [1] for majority of CPSs due to the infinite state space of CPS models, while theorem proving is extremely expensive. Therefore, robustness guided falsification [2], [3] methods are introduced to detect defects in CPS efficiently.

In robustness guided falsification, *Metric Temporal Logic* (MTL) [4] or *Signal Temporal Logic* (STL) [5] formulas are usually used to specify properties which must be satisfied by a CPS model. Robustness of an MTL formula, a numeric measure of how "robust" a property holds in the given CPS model, is defined. The state space of the CPS model is explored and system parameters which minimize the robustness value are identified as a good candidate for testing. In this way, robustness guided falsification aids to generate defect-leading system parameters (i.e., counterexamples), which enable more efficient, yet automatic detection of defects. Although the non-termination of robustness guided falsification does not mean the absence of counterexamples, it suggests the correctness of the CPS model to some extent.

There are existing approaches, which adopt various kinds of stochastic global optimization algorithms, e.g., simulated annealing [1] and cross-entropy method [6], to minimize the robustness value of a temporal logic formula. Many CPS systems, such as automotive, air plane and electric grid, are reactive. To test such CPS, we need to generate an input sequence which potentially leads to a failure. In existing approaches of robustness guided falsification, the input generation problem are framed as parameter generation problem. The input is divided into several control points and inputs at those points are treated as system parameters. However, the temporal relations between inputs and system responses are lost. Moreover, control points which represent an input form a high dimensional space, in which conducting optimization is very difficult.

With the rapid development of deep learning research, especially the capability that Deep Reinforcement Learning (DRL) [7] showed in the Go and various other computer games, the power of Reinforcement Learning has attracted more and more attentions. Reinforcement learning trains an agent from information learnt from reactions with the environment. The learning process is guided with a reward function, that maximizes a pre-defined global goal.

This work is partially supported by the National Natural Science Foundation of China (Grant Nos.: U1836214, 61802275, 61702362), Special Program of Artificial Intelligence, Tianjin Research Program of Application Foundation and Advanced Technology (No.: 16JCQNJC00100), and Special Program of Artificial Intelligence of Tianjin Municipal Science and Technology Commission (No.: 569 17ZXRGGX00150). This paper is partially based on results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO).
Shuang Liu is the corresponding author.

Inspired by the successful application of DRL on reactive, non-linear systems (such as games), in this work, we propose to frame the falsification problem as a *reinforcement learning* problem and explore the interactive learning power of DRL to find counterexamples. We focus on the CPS which is *reactive*, i.e., CPS which takes inputs from the environment in real time. We adopt two state-of-the-art *Deep Reinforcement Learning* (DRL) [8] techniques, i.e., *Asynchronous Advanced Actor Critic* (A3C) and *Double Deep-Q Network* (DDQN) to solve the robustness guided falsification problem on CPS.

We make the following contribution in this work:

- We propose to format the problem of falsifying CPS models into a reinforcement learning problem.
- We implement our proposed method and conduct experiments with three widely used CPS models.
- We compare our methods with state-of-the-art methods and provide thorough analysis on the results.
- We make our falsification tool and the three models public available[1].

The remaining of the paper is organized as follows. We discuss related works and present preliminaries in section 2 and section 3, respectively. We then introduce the details of our method and the evaluation methodology in section 4 and section 5. From section 6 to 8, we present the evaluation results using 3 widely adopted CPS models. In section 9, we provide a thorough analysis on the experiment results and discuss the performance-affecting factors of our methods. Finally, we conclude our approach and discuss potential future works in section 10.

2 Related Work

Formal verification of CPS Statistical model checking has been actively employed to conduct verification on CPS [9], [10], [11], [12]. Clarke et al. [11] propose to use statistical model checking techniques to verify CPS, and cross entropy is adopted to conduct rare event sampling. Zuliani et al. [12] propose a Bayesian statistical model checking technique to verify stochastic discrete-time hybrid systems. They show that their technique enables faster verification on hybrid systems through a case study. Grosu and Smolka [13] adopt the Monte-Carlo techniques to guide random walks over the state space of the system in model checking.

Model checking based approaches suffer from scalability problems, especially for the infinite state space of CPS. Therefore, it is hard for these kind of approaches to be applied to large-scale systems. Therefore, testing based approaches [14] gain popularity during the past decade. Researchers have been exploring good strategies to guide the test case selection process, and one of the most popular directions is robustness guided flasification [15], [6].

Robustness guided falsification of CPS In robustness guided falsification methods, quantitative semantics over *Metric Temporal Logic* (MTL) [4] and its variant, Signal Temporal Logic (STL) [16], [17], are employed. Robustness of an MTL formula, which is a numeric measurement of how "robust" a property holds in the given CPS model, is defined. Then the fault detection problem is formulated into the numerical minimization problem and system parameters which minimize the robustness value are identified as a good candidate for testing.

We can classify robustness guided methods into *black-box approaches* and *grey-box approaches*. Black-box approaches can further be classified into methods which convert the robustness falsification problem into a global optimization problem and the methods based on statistical modelling. The methods used a global optimization techniques include methods adopting simulated annealing [15], [2], [18], [19], methods adopting genetic algorithms [20], Tabu search [21], gradient decent [22] and Nelder-Mead [23]. The methods based on statistical modeling include cross-entropy method [6] and Gaussian regression [24], [25], [26], [27]. Cross Entropy (CE) [6] method samples a probability distribution (of input) which approximate the distribution induced by the robustness values of trajectories. The approaches which employ Gaussian regression cast the falsification process into the domain estimation problem, and then adopt the Gaussian Process to construct approximate probabilistic semantics of the temporal formulas to give high probabilities to regions where the formulas are falsified.

However, Gray-box approaches are most related to our work. Plaku et al. [28] propose an RRT-based approach, which incrementally generates the next action step by step by sampling the valid inputs. Dressosi et al. [29] propose a method in which RRT-search is guided by robustness. Constant interpolation is adopted to predict the final robustness. Our previous approach [30] adopts Deep Reinforcement Learning (DRL) techniques to solve the falsification problem. Our motivation is to explore more information, such as the temporal structure of the input and/or the output of the system, to guide the falsification. Unlike Dressosi et al., DRL integrates the guided search process by prediction of robustness, which is learnt from the past behaviours of the system. In this work, we provide a thorough extension, both in the theory part and the evaluation part to our previous work [30]. We also provide analysis and report our observations.

Controller Synthesis There are also approaches employing Reinforcement Learning (RL) to satisfy MTL properties in the field of controller synthesis. One of the challenges in this task is how to design the rewards in RL from the given formula. One strategy is exploring *automata-based reward shaping* [31], [32], [33]. These approaches convert the MTL formula into the corresponding automaton, and decide the reward based on the current state of the automaton. Another kind of approaches use the robustness of MTL [34], [35], [36]. Controller synthesis is a problem which tries to keep the system in safe states, while falsification is a problem which tries to find the critical errors in the system, and the different purposes lead to different reward functions.

3 Preliminary

3.1 Discrete time deterministic input-output systems

In this work, we focus on systems which consume inputs and deterministically produce outputs in real time. We assume that inputs and outputs occur at discrete time instances. Continuous time systems can be adapted to discrete

1. https://github.com/yoriyuki-aist/Falsify/

time systems through sampling. We use $\mathbf{x} = \mathbf{x}_1, \dots, \mathbf{x}_n, \dots$ to denote a finite (resp. infinite) sequence of inputs, $\mathbf{y} = \mathbf{y}_1, \dots, \mathbf{y}_n, \dots$ to denote a finite (resp. infinite) sequence of outputs, and $\mathbf{t} = t_1, \dots, t_n, \dots$ to denote a finite (resp. infinite) sequence of time instances at which inputs and outputs occur. We assume that inputs belong to a metric space X and that outputs belong to a metric space Y. A metric space is a set of points, in which the distance between two points that follows several axioms (e.g., triangle inequality) is defined. In this work, we focus on finite dimensional Euclid spaces with ordinary Euclidean distance.

Let X^* be the set of finite sequences of elements of X. The system $\mathbf{f}\colon X^* \to Y$ is a function from a finite sequence $\mathbf{x} = \mathbf{x}_1, \dots, \mathbf{x}_n \in X^*$ to an output $\mathbf{y}_{n+1} \in Y$. If $n = 0$, $\mathbf{f}$ returns the initial state $\mathbf{y}_1$ of the system. For simplification, we abuse notations and denote $\mathbf{f}(\mathbf{x}_1, \dots, \mathbf{x}_n) = \mathbf{y}_1, \dots, \mathbf{y}_{n+1}$, given that $\mathbf{f}(\mathbf{x}_1, \dots, \mathbf{x}_i) = \mathbf{y}_{i+1}$ for $i = 0, \dots, n$. Further, we apply $\mathbf{f}$ to an infinite sequence of inputs $\mathbf{x}_1, \dots, \mathbf{x}_n, \dots$ and obtain the infinite sequence of outputs $\mathbf{y}_1, \dots, \mathbf{y}_n, \dots$, given that $\mathbf{f}(\mathbf{x}_1, \dots, \mathbf{x}_i) = \mathbf{y}_{i+1}$ for $i = 0, 1, \dots$.

3.2 Metric temporal logic

To describe the property of the system $\mathbf{f}$, we employ a variant of *Metric tempral logic (MTL)* defined in [4]. The syntax is defined in the equation (1),

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2 \mid \varphi_1 \mathcal{S}_I \varphi_2 \quad (1)$$

p is either True, False or any closed set of a metric space. I is an interval over non-negative real numbers. If I is $[0, \infty)$, I is omitted. Intuitively, $\varphi_1 \mathcal{U}_I \varphi_2$ holds if φ_2 will become true at a time instance $t \in I$ after the current time and until then, φ_1 holds. Similarly, $\varphi_1 \mathcal{S}_I \varphi_2$ holds if φ_2 was true at a time instance $t \in I$ before the current time and after that time, φ_1 holds. We also use other common abbreviations, e.g., $\Box_I \varphi \equiv \neg(\mathsf{True}\, \mathcal{U}_I \neg\varphi)$, which means φ continues to hold for any time $t \in I$ units after the current time and $\boxminus_I \varphi \equiv \neg(\mathsf{True}\, \mathcal{S}_I \neg\varphi)$, which means φ holds at any time $t \in I$ units before the current time.

Definition 3.1. Let ϕ be a MTL-formula. n represents the number of time slots of which time is t_n. Let $\mathbf{t} = t_0, t_1, \dots, t_n, \dots$ be an infinite sequence of sampling time of the system states. Let $\mathbf{y} = \mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_n, \dots$ be system states of time instants $\mathbf{t_n}$ respectively. The relation $\mathbf{y}, n \models \phi$ (read that ϕ holds at n on the trace $\mathbf{y}$) is defined recursively on ϕ as follows.

$$\mathbf{y}, n \models p \iff \mathbf{y}_n \in p \quad (2)$$

$$\mathbf{y}, n \models \phi_1 \wedge \phi_2 \iff \mathbf{y}, n \models \phi_1 \text{ and } \mathbf{y}, n \models \phi_2 \quad (3)$$

$$\mathbf{y}, n \models \phi_1 \vee \phi_2 \iff \mathbf{y}, n \models \phi_1 \text{ or } \mathbf{y}, n \models \phi_2 \quad (4)$$

$$\mathbf{y}, n \models \neg\phi \iff \neg(\mathbf{y}, n \models \phi) \quad (5)$$

$$\mathbf{y}, n \models \phi\, \mathcal{U}_I \psi \iff \exists n' \begin{pmatrix} t_{n'} - t_n \in I, \mathbf{y}, n' \models \psi \text{ and} \\ n \leq \forall n'' < n', \mathbf{y}, n'' \models \phi \end{pmatrix} \quad (6)$$

$$\mathbf{y}, n \models \phi\, \mathcal{S}_I \psi \iff \exists n' \begin{pmatrix} t_n - t_{n'} \in I, \mathbf{y}, n' \models \psi \text{ and} \\ n' < \forall n'' \leq n, \mathbf{y}, n'' \models \phi \end{pmatrix} \quad (7)$$

We adopt the notion of *future-reach* $\mathsf{fr}(\varphi)$, which is the maximal time in future which is required to determine the truth value of formula φ, following [37].

Definition 3.2 (Future reach). For each MTL-formula, we define the *future reach* $\mathsf{fr}(\phi)$ as follows.

$$\mathsf{fr}(p) = 0 \quad (8)$$

$$\mathsf{fr}(\phi \wedge \psi) = \mathsf{fr}(\phi \vee \psi) = \max(\mathsf{fr}(\phi), \mathsf{fr}(\psi)) \quad (9)$$

$$\mathsf{fr}(\neg\phi) = \mathsf{fr}(\phi) \quad (10)$$

$$\mathsf{fr}(\phi\, \mathcal{U}_I \psi) = \max(\mathsf{fr}(\phi) + \sup I, \mathsf{fr}(\psi) + \sup I) \quad (11)$$

$$\mathsf{fr}(\phi\, \mathcal{S}_I \psi) = \max(\mathsf{fr}(\phi), \mathsf{fr}(\psi)) \quad (12)$$

If $\mathsf{fr}(\phi) = 0$, ϕ is called *pure past dependendt* [38]. For example, $\mathsf{fr}(p) = 0$, $\mathsf{fr}(\Box_{[0,3]} p) = 3$, $\mathsf{fr}(\boxminus_{[0,3]} p) = 0$ and $\mathsf{fr}(\Box_{[0,t_1]} p \to \Box_{[t_1,t_2]} p) = t_2$.

3.3 Robustness

Section 3.2 defines the relation $\models$, which determines whether an MTL-formula holds or not for the given system output at the given time. In this section, we define *robustness degree*, which is a numerical measure of how robustly the MTL-formula holds for the given system output at the given time.

First, we define the *distance* $\mathsf{Dist}(x, D)$ between a point x in a metric space X and its closed subset D.

$$\mathsf{Dist}(x, D) = \begin{cases} \inf\{\mathsf{dist}(x, z) \mid z \in X \backslash D\} & \text{if } x \in D \\ -\inf\{\mathsf{dist}(x, z) \mid z \in D\} & \text{if } x \notin D, \end{cases} \quad (13)$$

where $\mathsf{dist}(x, z)$ is the Euclidean distance between x and y.

For a given formula φ, an output signal $\mathbf{y}$ and time t, we adopt the notation of work [39] and define the *robustness degree* $[\![\varphi]\!](\mathbf{y}, t)$ of output signal $\mathbf{y}$ against φ at time t.

Definition 3.3 (Robustness). Let ϕ be an MTL-formula. The robustness function $[\![\phi]\!](\mathbf{y}, n)$ over infinite traces $\mathbf{y} = \mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_n, \dots$ is defined as follows.

$$[\![p]\!](\mathbf{y}, n) = \mathsf{Dist}(\mathbf{y}_n, p) \quad (14)$$

$$[\![\phi_1 \wedge \phi_2]\!](\mathbf{y}, n) = \min([\![\phi_1]\!](\mathbf{y}, n), [\![\phi_2]\!](\mathbf{y}, n)) \quad (15)$$

$$[\![\neg\phi]\!](\mathbf{y}, n) = -[\![\phi]\!](\mathbf{y}, n) \quad (16)$$

$$[\![\phi_1 \vee \phi_2]\!](\mathbf{y}, n) = \max([\![\phi_1]\!](\mathbf{y}, n), [\![\phi_2]\!](\mathbf{y}, n)) \quad (17)$$

$$[\![\phi\, \mathcal{U}_I \psi]\!](\mathbf{y}, n) = \max_{n', t_{n'} - t_n \in I} \min \left\{ \begin{array}{c} [\![\psi]\!](\mathbf{y}, n'), \\ \min\limits_{n''=n}^{n'-1} [\![\phi]\!](\mathbf{y}, n'') \end{array} \right\} \quad (18)$$

$$[\![\phi\, \mathcal{S}_I \psi]\!](\mathbf{y}, n) = \max_{n', t_n - t'_n \in I} \min \left\{ \begin{array}{c} [\![\psi]\!](\mathbf{y}, n'), \\ \min\limits_{n''=n'+1}^{n} [\![\phi]\!](\mathbf{y}, n'')) \end{array} \right\} \quad (19)$$

$$(20)$$

For the empty set $\emptyset$, $\min \emptyset = \infty$ and $\max \emptyset = -\infty$.

We can show that $[\![\phi]\!](\mathbf{y}, n) > 0$ if and only if $\mathbf{y}, n \models \phi$. For derived operators $\Box$ and $\boxminus$, we have the following equations.

$$[\![\Box_I \phi]\!](\mathbf{y}, n) = \min\{[\![\phi]\!](\mathbf{y}, n') \mid t_{n'} - t_n \in I\} \quad (21)$$

$$[\![\boxminus_I \phi]\!](\mathbf{y}, n) = \min\{[\![\phi]\!](\mathbf{y}, n') \mid t_n - t_{n'} \in I\} \quad (22)$$

If the robustness of $[\![\phi]\!](\mathbf{y}, n)$ becomes negative, $\mathbf{y}$ falsifies the property ϕ. Therefore, robustness guided falsification aims at finding a falsifying $\mathbf{y}$ through minimizing $[\![\phi]\!](\mathbf{y}, n)$.

3.4 Reinforcement Learning

Reinforcement learning is a machine learning technique in which an agent learns the structure of the environment based on observations, and maximizes the rewards by acting according to the learnt knowledge. The standard setting of a reinforcement learning problem consists of an agent and an environment. The agent observes the current state and reward from the environment, and returns the next action to the environment. Reinforcement learning is often formulated as a *Markov decision process (MDP)* [40]. A MDP is a triple $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{P}_0)$. $\mathcal{X}$ is a set of states, $\mathcal{A}$ is a set of actions and $\mathcal{P}_0$ is the transition probability kernel. A transition probability kernel $\mathcal{P}_0$ assigns a probability distribution over $\mathcal{X} \times \mathbb{R}$, which is a distribution over the next states and the reward, to each state-action pair $(x, a) \in \mathcal{X} \times \mathcal{A}$. Intuitively, $\mathcal{P}_0$ specifies the probability density of $(x', r) \in \mathcal{X} \times \mathbb{R}$ where x' is the next state and r is the reward which the agent obtains if it takes action a in system state x. The goal of reinforcement learning is for each step n, given the sequence of previous states $x_0, \ldots, x_{n-1}$, rewards $r_1, \ldots, r_n$ and actions $a_0, \ldots, a_{n-1}$, generating an action a_n, which maximizes expected value of the sum of rewards:

$$r = \sum_{i=n}^{\infty} \gamma^i r_{i+1}, \tag{23}$$

where $0 < \gamma \leq 1$ is a discount factor.

There are different kinds of reinforcement learning algorithms proposed in the literature. These approaches mainly falls into 2 different categories, i.e., value based and policy based. Deep reinforcement learning explores *deep neural networks* to represent a Q-function and/or a policy π.

Q-learning [41], [42] is the representative method for value-based reinforcement learning algorithm. For each action-state pair (x, a), let *optimal action-value function* $Q^*(x, a)$ be the highest achievable expected value of r when $x_0 = x$ and $a_0 = a$. Once the value of Q^* is known, the optimal strategy is to choose action a which maximizes $Q^*(x, a)$ for the current state x (following the greedy policy). One approach of reinforcement learning is to directly estimate Q^* and use this estimated value to determine best actions.

Deep Q Network (DQN) [43] is the first method introducing deep learning to reinforcement learning successfully. DQN uses a deep neural network to approximate optimal action-value function $Q^*(x, a)$ in Q-learning. Double DQN (DDQN) [44] is an improvement of DQN, it has two networks which conduct action selection and Q-value evaluation separately. DDQN learns faster and can avoid the over-estimation problem of DQN. Originally, DQN and DDQN can only output discrete actions. Later, a new representation of Q-function called NAF [44] is proposed, which makes learning Q-function for a continuous action domain possible.

Actor-critic [41], [42] is the representative method for policy based approaches. A *stochastic stationary policy* (or just *policy*) π maps states in $\mathcal{X}$ to probability distributions over actions in $\mathcal{A}$. Each policy π gives rise to a *Markov reward process (MRP)* $\mathcal{M} = (\mathcal{X}, \mathcal{P}_0)$. In a MRP, the state makes transitions as a Markovian process and generates a sequence of rewards $r_1, r_2, \ldots$. The action-value function Q^π is defined by

$$Q^\pi(x, a) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| x_0 = x, a_0 = a\right] \tag{24}$$

where $\mathbf{E}$ signifies the expected value. An actor-critic method works as follows. First, it starts with a random policy π_0 and the "actor" follows π_0 for some duration of time. Then, the "critic" estimates Q^{π_0} based on the results of the run. The greedy policy π_1 determined by estimated Q^{π_0} is generated and the actor follows π_1 in the next phase. The actor-critic method repeats this process.

Asynchronous Advantage Actor-Critic (A3C) [45] utilizes multiple processes to accelerate the training process. All processes run the same training algorithm and the information is collected by a central process. In this way, the algorithm can train models much faster.

In this work, we particularly adopted two state-of-the-art deep reinforcement learning algorithms, i.e., *Asynchronous Advantage Actor-Critic* (A3C) [45] and *Double Deep Q Network* (DDQN) [44].

4 Our Approach

4.1 Target properties

Our method focuses on safety properties of CPS, in particular, finite future reach safety properties.

Definition 4.1 (Finite future reach safety properties). If the MTL formula ψ has a form $\psi \equiv \Box\varphi$, in which $\mathsf{fr}(\varphi) < \infty$, we call ψ a *finite future reach safety property*. If $\mathsf{fr}(\varphi) = 0$, we call ψ a *pure past dependent safety property*.

Safety property is an important class of temporal property [38]. The future reach of properties that we focus on are often finite, mainly because we are usually interested in the system reactions in a finite period of time. Note that if the formula is past dependent, the robustness of it at a certain time is determined only by the trace until that time.

Lemma 4.1 (Robustness of a pure past dependent formula). Let $\mathbf{y}$ be a finite trace of system states $\mathbf{y}_0, \ldots, \mathbf{y}_n$. Let $\bar{\mathbf{y}}^{(1)}$ and $\bar{\mathbf{y}}^{(2)}$ be two infinite extensions of $\mathbf{y}$. If ϕ is pure past dependent,

$$[\![\phi]\!](\bar{\mathbf{y}}^{(1)}, n') = [\![\phi]\!](\bar{\mathbf{y}}^{(2)}, n') \tag{25}$$

for all $0 \leq n' \leq n$.

We define the robustness $[\![\phi]\!](\mathbf{y}, n')$ of the finite trace $\mathbf{y}$ as $[\![\phi]\!](\bar{\mathbf{y}}, n')$ where $\bar{\mathbf{y}}$ is an infinite extension of $\mathbf{y}$.

In general, we are interested in finite future reach safety properties, which cannot be represented by pure past dependent formula. Therefore, we approximate a finite future reach safety property using *monitoring formula*.

Definition 4.2 (monitoring formula). Let $\psi = \Box\varphi_0$ be a finite future reach safety property. The *monitoring formula* φ for ψ is defined by $\boxminus_{[\mathsf{fr}(\varphi_0), \mathsf{fr}(\varphi_0)]}\varphi_0$.

Because

$$\mathbf{y}, n \models \boxminus_{[\mathsf{fr}(\varphi_0), \mathsf{fr}(\varphi_0)]}\varphi_0 \iff \mathbf{y}, n - \mathsf{fr}(\varphi_0) \models \varphi_0 \tag{26}$$

Algorithm 1 Falsification for ψ by reinforcement learning

```
input: A finite future reach safety property $\psi$, its monitoring formula $\varphi$, a system $\mathbf{f}$ and an agent $\mathbf{a}$
output: A counterexample input signal $\mathbf{x}$ if exists
parameters: The end time $L$, the maximum number of the episode $N$
1: for numEpisode $\leftarrow$ 1 to $N$ do
2:     $i \leftarrow 0$, $\mathbf{y_0}$ be the initial (output) state of f and $r \leftarrow \mathsf{reward}(\mathbf{y}, \varphi, 0)$, $\mathbf{y} \leftarrow \mathsf{append}(\mathbf{y}, \mathbf{y_0})$
3:     $\mathbf{x}, \mathbf{y}$ be the empty input/output signal sequence
4:     while $i < L$ do
5:         $\mathbf{x}_i \leftarrow \mathbf{a}.\mathsf{step}(\mathbf{y}_i, r)$, $\mathbf{a}.\mathsf{update}()$, $\mathbf{x} \leftarrow \mathsf{append}(\mathbf{x}, \mathbf{x}_i)$    ▷ choose the next input by the agent
6:         $\mathbf{y}_{i+1} \leftarrow \mathbf{f}(\mathbf{x})$, $\mathbf{y} \leftarrow \mathsf{append}(\mathbf{y}, \mathbf{y}_{i+1})$    ▷ simulate, observe the new output state
7:         $r \leftarrow \mathsf{reward}(\mathbf{y}, \varphi, i+1)$    ▷ calculate reward based on Definition (4.3)
8:         $i \leftarrow i + 1$
9:     end while
10:    if $\mathbf{y} \not\models \psi$ then return $\mathbf{x}$ as a falsifying input
11:    end if
12:    $\mathbf{a}.\mathsf{reset}(\mathbf{y}_L, r)$
13: end for
```

The monitoring formula indicates whether φ_0 holds or not $\mathsf{fr}(\varphi_0)$ time-unit before. Note we assume that if $n - \mathsf{fr}(\varphi_0) < 0$, then $\mathbf{y}, n{-}\mathsf{fr}(\varphi_0) \models \varphi_0$ does not hold. The monitoring formula is a pure past dependent formula, therefore Lemma 4.1 holds. Our method requires φ_0 to be pure past-dependent. Therefore, we use robustness of the monitoring formula (estimating the robustness of φ_0) to falsify $\Box\varphi_0$. If the time interval is between $-\infty$ and ∞, $\Box \boxminus_{[\mathsf{fr}(\varphi_0),\mathsf{fr}(\varphi_0)]} \varphi_0$ holds if and only if $\Box\varphi_0$ holds. Therefore, we can use the monitoring formula to convert a finite future reach safety property to a pure past dependent safety property. In the setting of our paper, the time interval is cut at 0 and some finite T. Therefore, $\Box \boxminus_{[\mathsf{fr}(\varphi_0),\mathsf{fr}(\varphi_0)]} \varphi_0$ is not equivalent to $\Box\varphi_0$. However, we use the monitoring formula to approximate the original formula, and the experiment shows that this approximation still leads to reasonably good results.

4.2 Overview of our algorithm

Let us consider the falsification problem which finds a counterexample that falsifies the finite future reach safety property ψ. Let φ be the monitoring formula for ψ. Our mission is to generate an input signal $\mathbf{x}$ for system $\mathbf{f}$, such that the corresponding output signal $\mathbf{f}(\mathbf{x})$ does not satisfy ψ. We assume that a discretization of time is given (common for industry systems) and $\mathbf{x}, \mathbf{y}$ are discrete signals.

Our algorithm of falsifying a finite future reach safety property is shown in Algorithm 1. Our approach adopts deep reinforcement learning, which uses deep neural networks. Deep Neural network is known to be a universal function approximator for non-linear functions, and has shown impressive performance on tasks of non-linear systems. Therefore, Deep RL is suitable for non-linear systems [42]. In our algorithm, we fix the simulation time to be L and call one simulation until time L an *episode* in conformance with the reinforcement learning terminology. The agent $\mathbf{a}$ generates the input signal sequence $\mathbf{x} = \mathbf{x}_1, \ldots, \mathbf{x}_L$ by repeating the following steps:

1) At time i ($i = 0, 1, \ldots, L$), the agent $\mathbf{a}$ chooses the next input value $\mathbf{x}_i$. The generated input signal is extended to $\mathbf{x} \leftarrow \mathsf{append}(\mathbf{x}, \mathbf{x}_i)$ (line 5). The method call $\mathbf{a}.\mathsf{step}(\mathbf{y_i}, r)$ represents that the agent $\mathbf{a}$ takes the current state and reward pair ($\mathbf{y}_i$, r) and simultaneously returns the next action $\mathbf{x}_i$ (the input signal in the next step). Then the agent updates policy weights and append the newly generated action into the action sequence.
2) Our method obtains the corresponding output signal $\mathbf{y_{i+1}} = \mathbf{f}(\mathbf{x})$ by stepping forward one simulation on the model $\mathbf{f}$ from time i to $i+1$ on input $\mathbf{x}_i$ (line 6).
3) Let $\mathbf{y}_{i+1}$ be the new state (i.e., output) of the system. We compute reward r by $\mathsf{reward}(\mathbf{y}, \varphi, i+1)$ (line 7). Function $\mathsf{reward}(\mathbf{y}, \psi, i+1)$ calculates the reward based on Definition 4.3 and the details are discussed in Section 4.3.

At the end of each episode, we obtain the output signal trajectory $\mathbf{y}$ and check whether it satisfies the property ψ or not. If it is falsified, our algorithm returns the current input signal $\mathbf{x}$ as a counterexample (line 10). Otherwise, we discard the current generated signal input but retain the memory in the agent (line 12) and restart the episode from the beginning. The method call $\mathbf{a}.\mathsf{reset}(\mathbf{y}_i, r)$ notifies the agent that the current episode is completed with the state $\mathbf{y}_i$ and the reward r.

4.3 Reward definition for finite future reach safety property falsification

For a given future reach safety property ψ and a system $\mathbf{f}$, our goal is to find an input signal $\mathbf{x} = \mathbf{x}_0, \ldots, \mathbf{x}_{L-1}$ which minimizes the robustness $[\![\psi]\!](\mathbf{f}(\mathbf{x}), 0)$. For simplicity we assume that φ is pure past dependent.

At each time step $i = 0, \ldots, L$, we choose $\mathbf{x}_i$ in a greedy way. Let us assume that $\mathbf{x}_0 \ldots, \mathbf{x}_{i-1}$ are already determined. Then we choose the next input $\mathbf{x}_i$ as follows.

$$\mathbf{x}_i = \arg\min_{\mathbf{x}_i} \min_{\mathbf{x}_{i+1},\ldots,\mathbf{x}_{L-1}} [\![\Box\varphi]\!](\mathbf{f}(\mathbf{x}), 0) \tag{27}$$

$$= \arg\min_{\mathbf{x}_i} \min_{\mathbf{x}_{i+1},\ldots,\mathbf{x}_{L-1}} \min_{j=0}^{L} [\![\varphi]\!](\mathbf{f}(\mathbf{x}_0, \ldots, \mathbf{x}_{j-1}), j) \tag{28}$$

$$\sim \arg\min_{\mathbf{x}_i} \min_{\mathbf{x}_{i+1},\ldots,\mathbf{x}_{L-1}} -\log \sum_{j=0}^{L} e^{-[\![\varphi]\!](\mathbf{f}(\mathbf{x}_0,\ldots,\mathbf{x}_{j-1}),j)} \tag{29}$$

$$= \arg\max_{\mathbf{x}_i} \max_{\mathbf{x}_{i+1},\ldots,\mathbf{x}_{L-1}} \sum_{j=0}^{L} e^{-[\![\varphi]\!](\mathbf{f}(\mathbf{x}_0,\ldots,\mathbf{x}_{j-1}),j)} \tag{30}$$

$$= \arg\max_{\mathbf{x}_i} \max_{\mathbf{x}_{i+1},\ldots,\mathbf{x}_{L-1}} \sum_{j=i+1}^{L} e^{-[\![\varphi]\!](\mathbf{f}(\mathbf{x}_0,\ldots,\mathbf{x}_{j-1}),j)} \tag{31}$$

Let $r_j = e^{-[\![\varphi]\!](\mathbf{f}(\mathbf{x}_0,\ldots,\mathbf{x}_{j-1}),j)}$. Then, (31) can be written

$$= \arg\max_{\mathbf{x}_i} \max_{\mathbf{x}_{i+1},\ldots,\mathbf{x}_{L-1}} \sum_{j=i+1}^{L} r_j. \tag{32}$$

Recall that φ is past dependent, therefore, $[\![\varphi]\!](\mathbf{f}(\mathbf{x}), j) = [\![\varphi]\!](\mathbf{f}(\mathbf{x}_0, \ldots, \mathbf{x}_{j-1}), j)$ for all j. Therefore, we have the derivation from (27) to (28). (29) uses a widely adopted approximation of maximum by the log-sum-exp function [46], [47].

$$\max(x_1, \ldots, x_n) \sim \log(e^{x_1} + \ldots + e^{x_n}) \tag{33}$$

The approximation by log-sum-exp has a more general form

$$\max(x_1, \ldots, x_n) \sim \frac{1}{\alpha} \log(e^{\alpha x_1} + \ldots + e^{\alpha x_n}). \tag{34}$$

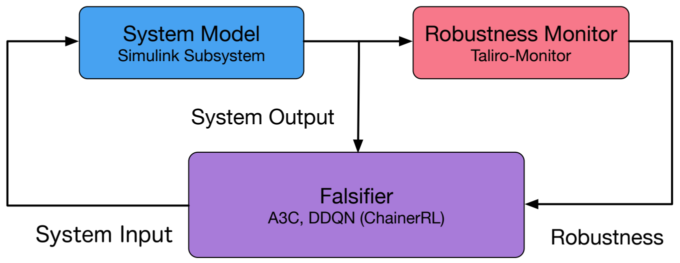


Fig. 1: Architecture of our system

We empirically explore the impact of α to the performance of our approach in Section 9.1. The derivation from (29) to (30) uses monotonicity of $\log$ function. Since $\mathbf{x}_0, \dots, \mathbf{x}_{j-1}$ are determined in previous steps, $\sum_{j=0}^{i} e^{-[\![\varphi]\!](\mathbf{f}(\mathbf{x}_0,\dots,\mathbf{x}_{j-1}),j)}$ is regarded as a constant. Therefore, we only maximize $\sum_{j=i+1}^{L} e^{-[\![\varphi]\!](\mathbf{f}(\mathbf{x}_0,\dots,\mathbf{x}_{j-1}),j)}$. This yields (31).

By comparing (23) with (32), we find that (32) defines exactly the reward of a reinforcement learning problem with $\gamma = 1$. The argument above motivates the following reward definition.

Definition 4.3 (reward). Let ψ be a finite future reach safety formula and φ the monitoring formula for ψ. and $\mathbf{y}$ be a finite sequence of signals. We define $\mathsf{reward}(\varphi, \mathbf{y}, i)$ as

$$\mathsf{reward}(\varphi, \mathbf{y}, i) = \exp(-[\![\varphi]\!](\mathbf{y}, i)) - 1 \qquad (35)$$

1 is subtracted from $\exp(-[\![\varphi]\!](\mathbf{y}, i))$ for scaling purposes.

Note that the deep reinforcement learning models we adopted, i.e., A3C and DDQN, follows the Markov assumption. We define the reward function based on past dependent formula to approximate the Markov assumption. Our evaluation results show that our approximation achieves good performance.

5 Evaluation Methodology

In this section, we first discuss the implementation of our approach, then we describe our experiment design and analysis methodology of the experiment results.

5.1 Implementation

The overall architecture of our system is shown in Fig. 1. We implement our system[2] in Matlab/Simulink. The entire system is driven by a Matlab function `falsify` which starts a Simulink module consisting three components, i.e., the Falsifier, the System Model and the Robustness Monitor. Falsifier and Robustness Monitor are Simulink blocks and System Model is a Simulink subsystem which implements the system under falsification. The simulation is conducted with the Matlab/Simulink models. The robustness monitor, which is implemented using the `TaliRo` monitor of S-Taliro [48], computes the robustness of the monitoring formula. The falsifier computes the reward from the robustness and call the reinforcement learning component, which implements A3C and DDQN using a Python library called ChainerRL [49] (version 0.3.0). Finally, the generated inputs and outputs by the Simulink module are sent back to a Matlab function `falsify` which calculate the (overall) robustness of the target formula using the `dp-taliro` component of S-Taliro. If the robustness is negative, falsification succeeds. Otherwise, `falsify` notifies the end of an episode to the reinforcement learning component and starts another episode.

The A3C is implemented by Gaussian policies with diagonal covariance using the *Long Short Term Memory (LSTM)* model. The network structure and most of the parameters are the same to the default model settings except `t_max` and γ. `t_max` is the time duration that the policy is updated. In our problem, the agent must find the falsifying input quickly. Therefore, we reduce `t_max` from the default 10 to 5. We also set $\gamma = 1$ according to our reward definition (refer to Section 4.3). Our version of DDQN is implemented by a fully connected neural network for Q-function using quadratic action value. We adopt the default parameter settings. We also define RAND, a random agent which outputs uniform random outputs regardless of the input the agent receives RAND is used as a baseline method to measure the impact of different falsifying methods.

We do not do parameter tuning for two reasons. Firstly, it is difficult to tune and obtain "optimal" parameters in practice, since parameter tuning requires a large number of experiment runs. Secondly, and more importantly, we want to make our approach generally applicable and avoid over-fitting to some particular system. Even if we find that a model is falsified faster with fine-tuned parameters, these parameters may not work for other models. Therefore, our evaluation aims to find whether reinforcement learning works well or not without fine tuning of parameters. However, we also observe that some poor performance instances of reinforcement leaning might be caused by the choice of parameters.

Our experiment is conducted on Ubuntu 16.04 with 18 core 2.3GHz Intel Xeon W processor and 128G byte RAM for $\varphi_1, \dots, \varphi_4$ of the CARS model in Section 6, and iMac Pro using 10 core 3.0GHz Intel Xeon W processor, and 128G byte RAM for other models.

5.2 Experiment design

We evaluate our proposed method as well as the baseline methods (which will be discussed in section 5.2.1) with 3 models. The results of the evaluations are shown from section 6 to section 8, respectively.

5.2.1 *Baseline methods*

The baseline methods we use for comparison are uniform random sampling (RAND), Cross Entropy (CE) [6] and Simulated Annealing (SA) [50]. RAND conducts uniform random sampling and is used as a baseline method for comparison. The cross entropy method is a representative method to learn the distributions which have good chance of falsifying the models. Simulated annealing is a representative method using genetic optimization techniques. Both methods are state-of-the-art methods for robustness-guided falsification of CPS systems.

2. The source code and models are available on https://github.com/yoriyuki-aist/Falsify/.

5.2.2 *Measurements*

We use the number of simulations required to falsify the property as a measurement to evaluate the performance of each method. The reasons are two folds. Firstly, the simulation process dominates the execution time among all steps of our method, therefore the number of simulations is good indication of the execution time. Secondly, considering that the execution time is affected by implementation details (e.g., programming language) as well as the execution environment, it is not a good performance measurement since different approaches are implemented in different environment and programming languages. To substantiate these two claims, we conduct an analysis of execution time in Section 9.2.

5.2.3 *Analysis methods*

Since all methods involve randomness, the results are highly non-deterministic. Therefore, we use statistical methods to compare the performance of methods we consider.

First, we present the summary statistics of the number of simulations required to falsify each property. We present two statistics, i.e., the rate in which the properties are successfully falsified by 200 episodes (we set 200 as the upper bound of episodes in our experiment), and the median of the number of episodes required when falsification succeeds. The reason of using median instead of average number, is that the distribution of the number of episodes required for falsification is highly non-normal, and thus average is not meaningful.

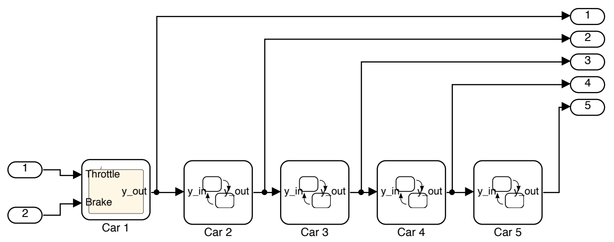
The summary statistics may not be enough since they may not reflect the differences of distributions. Statistical testing is required to provide confidence on the existence of differences between two methods. We choose *ordinal methods* for statistical testing, in particular, testing *relative effect size measures* between methods of interest to validate our claims. The *relative effect size measure* [51] between two random values X, Y is defined as

$$p = P(X < Y) + \frac{1}{2}P(X = Y). \tag{36}$$

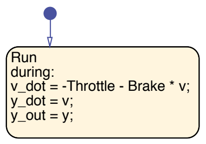
If $p < 0.5$, X is likely larger than Y and vice verso. p has close relationship to Cliff's d [52]

$$d = P(X < Y) - P(X > Y) \tag{37}$$

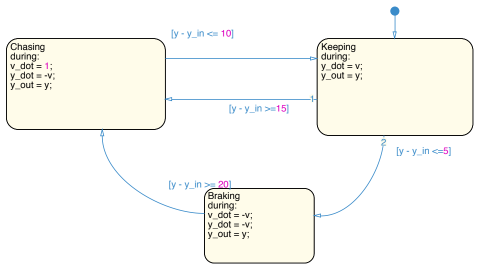
Both p and d are *ordinal* statistics solely determined by the relative order of data points. Cliff argues the advantages of ordinal statistics over traditional statistics using means or other summary values, which apply to our approach. First, our question can be directly formulated with the order between data. We want to know how often a method A outperforms a method B. This question is stated by referring the order between data points of A and B. Comparing means or medians of A and B only answers this question indirectly and under strong assumptions. Second, results of statistical testing using ordinal statistics is valid under weaker assumptions of the distribution of data than classical methods. Data from our experiments have unknown distribution and thus testing ordinal statistics are more favourable. In our case, the null hypothesis is $p = 0.5$ and there is no assumption on the distributions of data points being test [52].



(a) The **CARS** model



(b) Statechart of car 1



(c) Statechart of car 2 to car 5

Fig. 2: The **CARS** model

If we set the significance level to 0.05 for each testing, we obtain spurious results much higher probability than 0.05 in overall experiments, since the statistical testing is repeated multiple times. The easiest remedy of this situation is setting the significance level to be $0.05/N$ (known as Bonferroni correction), where N is the number of testing. However, Bonferroni correction is too conservative if the results of testing have correlations. Konietschke et al. [51] propose a method to test multiple hypothesis simultaneously for p and develop `nparcomp` package [3] in `R`. We set the significance level to 0.05 for each simulation models, i.e., **CARS**, **AT** and **PTC** models. For each formula, we apply Bonferroni correction and set the significance level to be $0.05/N$, where N is the number of formulas. We perform multiple comparisons between RAND and all the other methods, and comparisons between baseline methods and RL based methods using `nparcomp`.

6 Evaluation using CARS model

6.1 Model description

The **CARS** model contains five cars, as shown in Fig. 2a. The first, leading car drives by Throttle and Brake which are supplied as the system inputs (shown in Fig. 2b). The other cars drive autonomously according to the state-chart shown in Fig. 2c, which is derived from the work by Hu et al. [53].

To make **CARS** model discrete, we choose a sampling interval ΔT and discretize time into discrete intervals. Then, the inputs are represented by a sequence $\mathbf{x}$ of $\mathbf{x}_i$ which is a vector of Throttle and Brake values. Each $\mathbf{x}_i$ corresponds to the inputs of time instance $i\Delta T$. Similarly, the outputs

3. https://cran.r-project.org/web/packages/nparcomp/index.html

TABLE 1: The properties for the CARS model.

id	Formula
φ_1	$\Box y_5 - y_4 \leq 40.0$
φ_2	$\Box_{[0,70]} \Diamond_{[0,30]} y_5 - y_4 \geq 15$
φ_2	$\Box_{[0,80]}((\Box_{[0,20]} y_2 - y_1 \leq 20) \vee (\Diamond_{[0,20]} y_5 - y_4 \geq 40))$
φ_4	$\Box_{[0,65]} \Diamond_{[0,30]} \Box_{[0,5]} y_5 - y_4 \geq 8$
φ_5	$\Box_{[0,72]} \Diamond_{[0,8]} (\Box_{[0,5]} y_2 - y_1 \geq 9 \rightarrow \Box_{[5,20]} y_5 - y_4 \geq 9)$

TABLE 2: Success rate of falsifying the **CARS** model

Properties	A3C -WB	DDQN -WB	A3C -BB	DDQN -BB	RAND	CE	SA
φ_1	**100**	**100**	0	0	0	**100**	30
φ_2	**100**	**100**	**100**	**100**	**100**	34	11
φ_3	**100**	73	0	0	0	99	41
φ_4	24	**36**	0	0	0	0	0
φ_5	99	**100**	0	0	0	**100**	9

are represented by a sequence $\mathbf{y}$ of $\mathbf{y}_i$, which is a vector of $y_1, \ldots, y_5$ in this example.

6.2 Properties

We falsify properties listed in Table 1. All properties are safety properties and are designed to have increasing complexities. The outermost $\Box$ of φ_2–φ_5 has restricted time interval, but this is to exclude the influence of the endpoint. We choose properties in which the formulas inside the outermost $\Box$ is taken from the each hierarchy of properties proposed in Fig. 4, Chapter 2. of Clarke et al. [38]. The goal is to investigate the impact of each hierarchy of complexity of formulas to the performance. In φ_1, we use just an atomic predicate, thus the entire formula forms an invariant property. In φ_2, the inside of the outermost $\Box$ is a guarantee, thus the entire formula forms a recurrence. In φ_3, the inside of the outermost $\Box$ is an obligation. In φ_4, the inside of the outermost $\Box$ is a persistence. In φ_5, the inside of the outermost $\Box$ is a reactivity. The parameters are selected to make falsification possible yet challenging.

6.3 Experiment settings

For each given property, we test A3C, DDQN and baselines RAND, CE and SA. In particular, A3C and DDQN are tested on both *white-box* (which observes all locations $y_1, \ldots, y_5$ of the cars) and *black-box* (which can observe the robustness but cannot observe locations of cars at all) settings. The purpose of having two settings for the reinforcement learning algorithms is to measure the impact that system internal dynamics information has on the performance of RL algorithms. We consider CE and SA black-box methods since they only use robustness value and does not use the states $y_1, \ldots, y_n$ for optimization. For each combination of the given property and method, we run the falsification procedure 100 times. For each falsification procedure, we execute simulation episodes up to 200 times and measure the number of simulation episodes required to falsify the property. If the property cannot be falsified within 200 episodes, the procedure fails. We set $\Delta T = 5$. Since the Δ_T is much larger than the time step of simulation, we have to interpolate the inputs to the system. We use linear interpolation since RL generates the inputs to the system on the fly. Spline interpolation, which is the default method provided by S-Taliro, is difficult to apply in such situations. To make the comparison fair, we also use linear interpolation for RAND and S-Taliro based methods.

TABLE 3: Median number of episodes when falsification of **CARS** succeeds

Properties	A3C -WB	DDQN -WB	A3C -BB	DDQN -BB	RAND	CE	SA
φ_1	**3**	**3**	–	–	–	29	128.5
φ_2	4	5	**1**	**1**	**1**	7	19
φ_3	**5**	50	–	–	–	25	127
φ_4	**94.5**	108.5	–	–	–	–	–
φ_5	8	**5**	–	–	–	22	69

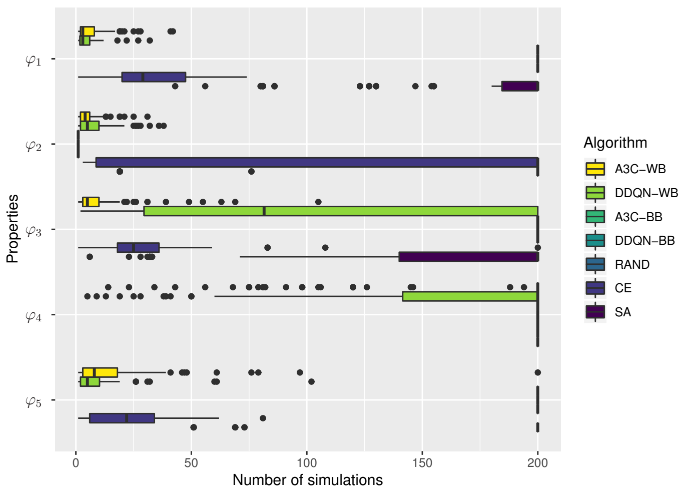


Fig. 3: number of episodes to falsify **CARS** model

6.4 Evaluation results

Table 2 shows the success rates of falsification within 200 simulations for each formula and method. The method A3C-WB/DDQN-WB indicates reinforcement learning methods A3C/DDQN in white-box settings. The method A3C-BB/DDQN-BB indicates reinforcement learning methods A3C/DDQN in black-box settings. Table 3 shows the median number (of episodes) required to falsify the model for each formula and method. If all falsification attempts fail, we mark the median umber of episodes as "-". Fig. 3 shows the distributions of number of episodes required to falsify the model for each formula and method in a box plot. We can observe that white-box A3C shows the most stable performance.

Table 4 and Table 5 show the relative effect size measure on the number of episodes. The number less than 0.5 means the probability of outcomes from the method presented in the row, is less likely to be smaller than outcome from the method presented in the column. The number in bold is smaller than 0.5 in the statistically significant manner. The number in italic is larger than 0.5 in the statistically significant manner.

From Table 4, we can observe that all black-box reinforcement learning methods, A3C-BB and DDQN-BB, has no differences to random inputs. Further, we can observe that except φ_2, white box RL methods outperform RAND with large margin. Note that, for φ_2, CE and SA also underperform RAND.

TABLE 4: Relative effect size measure comparing to random inputs on **CARS** model

Properties	A3C -WB	DDQN -WB	A3C -BB	DDQN -BB	CE	SA
φ_1	**0.001**	**0.001**	0.500	0.500	**0.001**	**0.350**
φ_2	*0.920*	*0.920*	0.500	0.500	*0.999*	*0.999*
φ_3	**0.001**	**0.135**	0.500	0.500	**0.005**	**0.295**
φ_4	**0.380**	**0.320**	0.500	0.500	0.500	0.500
φ_5	**0.005**	**0.001**	0.500	0.500	**0.001**	0.455

TABLE 5: Relative effect size measure comparing to S-Taliro on **CARS** model

expName		A3C-WB	DDQN-WB	A3C-BB	DDQN-BB
φ_1	CE	**0.081**	**0.052**	*0.999*	*0.999*
	SA	**0.001**	**0.001**	*0.650*	*0.650*
φ_2	CE	**0.112**	**0.147**	**0.001**	**0.001**
	SA	**0.004**	**0.011**	**0.001**	**0.001**
φ_3	CE	**0.132**	*0.778*	*0.995*	*0.995*
	SA	**0.015**	**0.280**	*0.705*	*0.705*
φ_4	CE	**0.380**	**0.320**	0.500	0.500
	SA	**0.380**	**0.320**	0.500	0.500
φ_5	CE	**0.330**	**0.237**	*0.999*	*0.999*
	SA	**0.009**	**0.002**	0.545	0.545

From Table 5, we can observe that white box RL methods almost always outperform CE and SA. Only exception is φ_2, in which white box DDQN underperforms CE. One potential reason for RL to perform poorly on formula φ_2 is that, φ_2 is relatively easy for random sampling to find a counterexample. RL methods may need multiple simulations as well as "useful data" to learn such a rule.

7 Evaluation using AT model

7.1 Model description

We use a model of an automatic transmission controller (**AT** model), offered by Mathworks as a Matlab/Simulink example, as one of the benchmarks. **AT** model is a model of an automotive driving train. Fig. 4 shows the architecture of the **AT** model. The model takes, throttle and brake information as input, and outputs three values, i.e., Engine RPM, Gear and Vehicle Speed. All values except Gear are real values greater than or equal to 0. Gear takes four discrete values g_1, g_2, g_3, g_4 as input. The model consists of four components, i.e., Engine, Transmission, Vehicle and ShiftLogic. The first three components model the continuous dynamics of physical bodies, and thus are described by ordinary differential equations. ShiftLogic implements the logic of automatic transmission by a finite state automaton. Therefore, **AT** is a mixture of continuous and digital systems, which is a typical CPS. The detailed description as well as an implementation of the **AT** model are available from Mathworks web page [54].

7.2 Properties

We use the properties listed in Table 6 for evaluation. φ_1–φ_6 are variants of formulas adapted from [55]. We modified the original formula slightly to make them finite future reach safety properties. φ_7–φ_9 are constructed purposely to test behaviours of longer time than φ_1–φ_6. Therefore, for φ_1–φ_6, the behaviour of the system from the start to 30 seconds after is simulated, while for φ_7–φ_9, the behaviour of the system from the start to 100 seconds after is simulated.

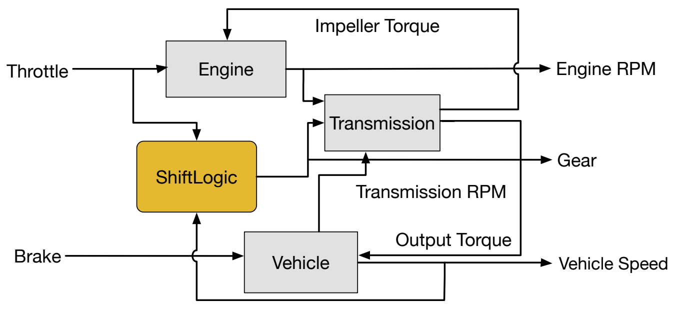


Fig. 4: The **AT** model

TABLE 6: The list of the evaluated properties on **AT**.

id	Formula
φ_1	$\square \omega \leq \overline{\omega}$
φ_2	$\square(v \leq \overline{v} \wedge \omega \leq \overline{\omega})$
φ_3	$\square((g_2 \wedge \diamond_{[0,0.1]} g_1) \rightarrow \square_{[0.1,1.0]} \neg g_2)$
φ_4	$\square((\neg g_1 \wedge \diamond_{[0,0.1]} g_1) \rightarrow \square_{[0.1,1.0]} g_1)$
φ_5	$\square \bigwedge_{i=1}^{4}((\neg g_i \wedge \diamond_{[0,0.1]g_i}) \rightarrow \square_{[0.1,1.0]} g_i)$
φ_6	$\square(\square_{[0,t_1]} \omega \leq \overline{\omega} \rightarrow \square_{[t_1,t_2]} v \leq \overline{v})$
φ_7	$\square v \leq \overline{v}$
φ_8	$\square \diamond_{[0,25]} \neg(\underline{v} \leq v \leq \overline{v})$
φ_9	$\square \neg \square_{[0,20]}(\neg g_4 \wedge \omega \geq \overline{\omega})$

In formulas φ_1–φ_9, v denotes the vehicle speed, ω denotes the engine speed and $g_1, \ldots, g_4$ denote the gear states. $\overline{v}$ and $\underline{v}$ denote the upper and lower bound of the vehicle speed, respectively. Similarly, $\overline{\omega}$ denotes the upper bound of the engine speed. We use different $\overline{v}$ and $\overline{\omega}$ for each property. These parameters are set based on experiments, to make falsification possible yet challenging.

φ_1 means that we want to limit the engine speed to the limit $\overline{\omega}$. φ_2 limits the vehicle speed and the engine speed simultaneously. We use $\overline{v} = 170.0$ and $\overline{\omega} = 4770.0$. φ_3–φ_5 are properties for gear movement. We changed the properties from the original paper by replacing the next operator Xp to $\diamond_{[0,0.1]}$ and the open time interval $(0, 2.5]$ to the closed interval $[0.1, 1.0]$. The reason of this change is to remove the dependency to the time interval used to calculate robustness from the properties. We use the time interval $[0.1, 1.0]$ instead $[0.1, 2.5]$ to increase the difficulties of falsifying the formulas. φ_3 states that once the gear is moved from g_2 to g_1, the gear never returns to g_2 until 1 second passes. φ_4 states that the gear is changed to g_1, then it stays at g_1 until 1 second passes. The formula φ_5 is the conjunction of φ_3-like properties for all gear states $g_1, \ldots, g_4$.

We rewrite φ_6 to a finite future reach safety property and change the statement to "if the engine speed does not exceed $\overline{\omega} = 4550.0$ continuously until $t_1 = 10$ seconds, the vehicle speed never exceeds $\overline{v} = 160.0$ from $t_1 = 10$ second to $t_2 = 20$ second.

φ_7 states that the vehicle speed never exceeds the upper limit of the speed $\overline{v} = 160$. φ_8 states that the vehicle speed v cannot satisfy the constraint $\underline{v} = 70 \leq v \leq \overline{v} = 80$ in 25 second continuously. φ_9 states that it is impossible to continuously maintain faster than the limit engine speed $\overline{\omega} = 3100$ without simultaneously going to gear state g_4.

7.3 Experiment settings

For each given property, we run the falsification procedure 100 times. For each falsification procedure, we execute sim-

TABLE 7: Success rates of falsifying **AT** model

	A3C			DDQN			RAND			CE			SA		
	1	5	10	1	5	10	1	5	10	1	5	10	1	5	10
φ_1	62	55	73	**100**	56	4	0	5	26	5	22	13	2	14	11
φ_2	60	47	74	**97**	61	6	0	3	27	0	21	6	0	20	20
φ_3	70	0	0	47	0	0	**100**	0	0	75	0	0	26	0	0
φ_4	74	0	0	51	0	0	**100**	0	0	86	0	0	22	0	0
φ_5	**100**	2	0	**100**	0	0	**100**	0	0	**100**	0	0	**100**	0	0
φ_6	66	63	76	**100**	**100**	**100**	0	39	**100**	0	95	**100**	0	21	84
φ_7	37	41	42	41	96	**100**	0	0	0	0	0	0	0	0	0
φ_8	27	25	34	94	**100**	**100**	**100**	**100**	**100**	**100**	99	97	32	79	87
φ_9	37	51	55	72	**100**	92	0	2	23	0	1	11	0	0	8

TABLE 8: Median number of episodes when falsification succeeds on **AT**

	A3C			DDQN			RAND			CE			SA		
	1	5	10	1	5	10	1	5	10	1	5	10	1	5	10
φ_1	27	**23**	35	26	94.5	107	–	120	84	67	68	88	109	137	85
φ_2	**18.5**	31	30	27	93	115	–	148	85	–	74	58	–	119	122
φ_3	**8.5**	–	–	62	–	–	22	–	–	9	–	–	16	–	–
φ_4	**6**	–	–	40	–	–	25	–	–	8	–	–	52	–	–
φ_5	**1**	128	–	2	–	–	**1**	–	–	**1**	–	–	**1**	–	–
φ_6	**3**	4	**3**	7.5	3.5	**3**	–	92	25	–	36	13	–	117	78
φ_7	31	17	**16**	89	57.5	82	–	–	–	–	–	–	–	–	–
φ_8	68	58	21	55.5	60.5	26	**10**	26	28	**10**	21	15	67	55	42
φ_9	12	**11**	16	25	20.5	26	–	85	101	–	51	22	–	–	124

TABLE 9: Relative effect size measure comparing to random inputs on **AT**

		A3C			DDQN			CE			SA		
		1	5	10	1	5	10	1	5	10	1	5	10
φ_1	1	**0.190**	**0.225**	**0.135**	**0.001**	**0.220**	0.480	0.475	**0.390**	0.435	0.490	**0.430**	0.445
	5	**0.205**	**0.241**	**0.148**	**0.003**	**0.244**	0.505	0.500	0.412	0.459	0.515	0.456	0.470
	10	**0.285**	**0.324**	**0.225**	**0.048**	**0.356**	*0.611*	*0.605*	0.514	0.565	*0.620*	0.566	0.576
φ_2	1	**0.200**	**0.265**	**0.130**	**0.015**	**0.195**	0.470	0.500	**0.395**	0.470	0.500	**0.400**	**0.400**
	5	**0.207**	**0.273**	**0.136**	**0.016**	**0.206**	0.485	0.515	**0.407**	0.484	0.515	**0.414**	**0.414**
	10	**0.294**	0.370	**0.222**	**0.072**	**0.336**	*0.606*	*0.635*	0.524	*0.604*	*0.635*	0.540	0.542
φ_3	1	0.556	*0.999*	*0.999*	*0.878*	*0.999*	*0.999*	0.471	*0.999*	*0.999*	*0.863*	*0.999*	*0.999*
	5	**0.150**	0.500	0.500	**0.265**	0.500	0.500	**0.125**	0.500	0.500	**0.370**	0.500	0.500
	10	**0.150**	0.500	0.500	**0.265**	0.500	0.500	**0.125**	0.500	0.500	**0.370**	0.500	0.500
φ_4	1	0.507	*0.999*	*0.999*	*0.824*	*0.999*	*0.999*	0.369	*0.999*	*0.999*	*0.910*	*0.999*	*0.999*
	5	**0.130**	0.500	0.500	**0.245**	0.500	0.500	**0.070**	0.500	0.500	**0.390**	0.500	0.500
	10	**0.130**	0.500	0.500	**0.245**	0.500	0.500	**0.070**	0.500	0.500	**0.390**	0.500	0.500
φ_5	1	0.542	*0.999*	*0.999*	*0.837*	*0.999*	*0.999*	0.520	*0.999*	*0.999*	0.561	*0.999*	*0.999*
	5	**0.001**	0.490	0.500	**0.001**	0.500	0.500	**0.001**	0.500	0.500	**0.001**	0.500	0.500
	10	**0.001**	0.490	0.500	**0.001**	0.500	0.500	**0.001**	0.500	0.500	**0.001**	0.500	0.500
φ_6	1	**0.170**	**0.185**	**0.120**	**0.001**	**0.001**	**0.001**	0.500	**0.025**	**0.001**	0.500	**0.395**	**0.080**
	5	**0.284**	**0.287**	**0.191**	**0.028**	**0.016**	**0.017**	*0.695*	**0.115**	**0.038**	*0.695*	0.599	**0.258**
	10	0.535	0.512	0.369	**0.163**	**0.102**	**0.116**	*0.999*	0.633	**0.292**	*0.999*	*0.968*	*0.808*
φ_7	1	**0.315**	**0.295**	**0.290**	**0.295**	**0.020**	**0.001**	0.500	0.500	0.500	0.500	0.500	0.500
	5	**0.315**	**0.295**	**0.290**	**0.295**	**0.020**	**0.001**	0.500	0.500	0.500	0.500	0.500	0.500
	10	**0.315**	**0.295**	**0.290**	**0.295**	**0.020**	**0.001**	0.500	0.500	0.500	0.500	0.500	0.500
φ_8	1	*0.960*	*0.948*	*0.879*	*0.931*	*0.882*	*0.688*	0.433	0.657	0.604	*0.899*	*0.872*	*0.852*
	5	*0.905*	*0.903*	*0.805*	*0.690*	0.650	0.429	**0.194**	0.379	**0.317**	*0.865*	*0.713*	0.625
	10	*0.923*	*0.916*	*0.823*	*0.748*	*0.725*	0.486	**0.245**	0.432	0.372	*0.875*	*0.759*	*0.678*
φ_9	1	**0.315**	**0.245**	**0.225**	**0.140**	**0.001**	**0.040**	0.500	0.495	0.445	0.500	0.500	0.460
	5	**0.323**	**0.253**	**0.234**	**0.150**	**0.008**	**0.048**	0.510	0.505	0.455	0.510	0.510	0.470
	10	0.401	**0.314**	**0.293**	**0.200**	**0.024**	**0.080**	*0.615*	*0.609*	0.550	*0.615*	*0.615*	0.576

ulation episodes up to 200 times and measure the number of simulation episodes required to falsify the property. If the property cannot be falsified within 200 episodes, the procedure fails. To analyze the effect of Δ_T has on the performance of each algorithm, we vary Δ_T among {1, 5, 10}. Similar to the CARS-model, we use linear interpolation for all methods.

7.4 Evaluation results

The summary statistics for success rate and the median number of episodes required for falsification are shown in Table 7 and Table 8, respectively. Fig. 5 shows the distribution of number of episodes required to falsify the model for $\Delta_T = 1$ in a box plot. In Table 8, "–" indicates that all falsification attempts fail. Table 8 shows that DDQN with $\Delta T = 1$ is relatively the most stable, though it may not always be the best, method.

Table 9 shows relative effect size measures of each combination of method (A3C, DDQN, CE, SA) and sampling rate $\Delta T = 1, 5, 10$, shown in columns, against RAND with $\Delta T = 1, 5, 10$, shown in rows. We further compare the DRL-based methods, i.e., A3C and DDQN, with methods implemented in S-Taliro, i.e., CE and SA, and the relative effect size measure results are shown in Table 10.

By statistical analysis, we can observe from Table 9 that A3C and DDQN outperform RAND for $\varphi_1, \varphi_2, \varphi_6, \varphi_7, \varphi_9$ for most cases. While RAND shows better performance for for property $\varphi_3, \varphi_4, \varphi_5, \varphi_8$ than all the other methods, i.e., A3C and DDQN, CE and SA. This may suggest that $\varphi_3, \varphi_4, \varphi_5, \varphi_8$ do not have an easy structure to be exploit for

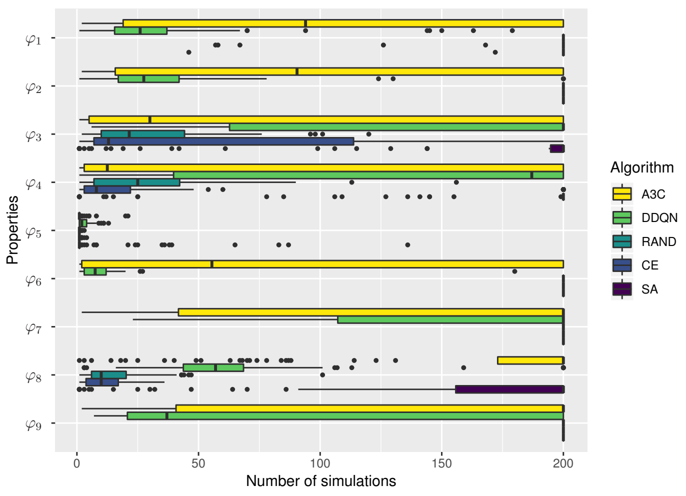


Fig. 5: number of episodes to falsify **AT** model ($\Delta_T = 1$)

TABLE 10: Relative effect size measure comparing to S-Taliro on **AT** model

			A3C			DDQN		
			1	5	10	1	5	10
φ_1	CE	1	**0.206**	**0.242**	**0.149**	**0.004**	**0.245**	0.505
		5	**0.270**	**0.307**	**0.212**	**0.032**	**0.341**	*0.592*
		10	**0.234**	**0.272**	**0.176**	**0.017**	**0.287**	0.545
	SA	1	**0.196**	**0.231**	**0.141**	**0.002**	**0.229**	0.490
		5	**0.233**	**0.268**	**0.171**	**0.011**	**0.284**	0.549
		10	**0.226**	**0.262**	**0.167**	**0.012**	**0.275**	0.535
φ_2	CE	1	**0.200**	**0.265**	**0.130**	**0.015**	**0.195**	0.470
		5	**0.271**	**0.344**	**0.199**	**0.043**	**0.312**	0.578
		10	**0.220**	**0.287**	**0.147**	**0.021**	**0.227**	0.501
	SA	1	**0.200**	**0.265**	**0.130**	**0.015**	**0.195**	0.470
		5	**0.259**	**0.331**	**0.182**	**0.034**	**0.284**	0.570
		10	**0.255**	**0.327**	**0.175**	**0.023**	**0.281**	0.569
φ_3	CE	1	0.524	*0.875*	*0.875*	*0.772*	*0.875*	*0.875*
		5	**0.150**	0.500	0.500	**0.265**	0.500	0.500
		10	**0.150**	0.500	0.500	**0.265**	0.500	0.500
	SA	1	**0.275**	*0.630*	*0.630*	0.414	*0.630*	*0.630*
		5	**0.150**	0.500	0.500	**0.265**	0.500	0.500
		10	**0.150**	0.500	0.500	**0.265**	0.500	0.500
φ_4	CE	1	0.561	*0.930*	*0.930*	*0.823*	*0.930*	*0.930*
		5	**0.130**	0.500	0.500	**0.245**	0.500	0.500
		10	**0.130**	0.500	0.500	**0.245**	0.500	0.500
	SA	1	**0.215**	*0.610*	*0.610*	**0.355**	*0.610*	*0.610*
		5	**0.130**	0.500	0.500	**0.245**	0.500	0.500
		10	**0.130**	0.500	0.500	**0.245**	0.500	0.500
φ_5	CE	1	0.523	*0.999*	*0.999*	*0.823*	*0.999*	*0.999*
		5	**0.001**	0.490	0.500	**0.001**	0.500	0.500
		10	**0.001**	0.490	0.500	**0.001**	0.500	0.500
	SA	1	0.475	*1.000*	*0.999*	*0.723*	*0.999*	*0.999*
		5	**0.001**	0.490	0.500	**0.001**	0.500	0.500
		10	**0.001**	0.490	0.500	**0.001**	0.500	0.500
φ_6	CE	1	**0.170**	**0.185**	**0.120**	**0.001**	**0.001**	**0.001**
		5	0.503	0.461	**0.305**	**0.045**	**0.034**	**0.035**
		10	0.572	0.561	0.435	**0.277**	**0.168**	**0.198**
	SA	1	**0.170**	**0.185**	**0.120**	**0.001**	**0.001**	**0.001**
		5	**0.229**	**0.239**	**0.159**	**0.016**	**0.011**	**0.012**
		10	0.432	0.418	**0.286**	**0.069**	**0.052**	**0.054**
φ_7	CE	1	**0.315**	**0.295**	**0.290**	**0.295**	**0.020**	**0.001**
		5	**0.315**	**0.295**	**0.290**	**0.295**	**0.020**	**0.001**
		10	**0.315**	**0.295**	**0.290**	**0.295**	**0.020**	**0.001**
	SA	1	**0.315**	**0.295**	**0.290**	**0.295**	**0.020**	**0.001**
		5	**0.315**	**0.295**	**0.290**	**0.295**	**0.020**	**0.001**
		10	**0.315**	**0.295**	**0.290**	**0.295**	**0.020**	**0.001**
φ_8	CE	1	*0.968*	*0.956*	*0.893*	*0.960*	*0.914*	*0.746*
		5	*0.936*	*0.923*	*0.836*	*0.840*	*0.791*	0.548
		10	*0.936*	*0.923*	*0.844*	*0.890*	*0.827*	0.613
	SA	1	0.525	0.536	0.477	**0.180**	**0.155**	**0.123**
		5	*0.762*	*0.770*	*0.674*	0.406	0.394	**0.233**
		10	*0.825*	*0.825*	*0.723*	0.542	0.519	**0.308**
φ_9	CE	1	**0.315**	**0.245**	**0.225**	**0.140**	**0.001**	**0.040**
		5	**0.319**	**0.248**	**0.228**	**0.143**	**0.001**	**0.043**
		10	**0.367**	**0.292**	**0.274**	**0.203**	**0.049**	**0.096**
	SA	1	**0.315**	**0.245**	**0.225**	**0.140**	**0.001**	**0.040**
		5	**0.315**	**0.245**	**0.225**	**0.140**	**0.001**	**0.040**
		10	**0.344**	**0.268**	**0.248**	**0.160**	**0.008**	**0.054**

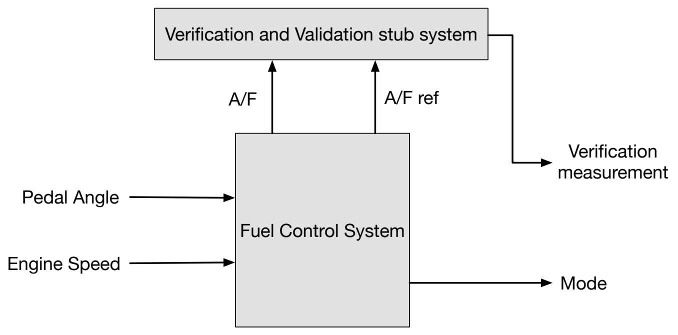


Fig. 6: Power train control (**PTC**) model

faster falsification than random input generation.

We can also observe from Table 10 that if $\Delta T = 1$, DDQN significantly outperforms CE and SA for $\varphi_1, \varphi_2, \varphi_6, \varphi_7, \varphi_9$. For these properties, other ΔT are also good except for the case of φ_1, in which $\Delta T = 10$ has no difference to or slightly underperforms CE and SA. A3C outperforms CE and SA for all ΔT to falsify $\varphi_1, \varphi_2, \varphi_7, \varphi_9$. However, to falsify φ_6 A3C has no statistically significant advantage over CE and SA. For $\varphi_3, \varphi_4, \varphi_5$, we cannot find statistically significant difference between A3C, DDQN and CE, SA. All difference for $\varphi_3, \varphi_4, \varphi_5$ seems to be caused by the differences of ΔT, rather than different methods. For φ_8, CE significantly outperforms DDQN and SA with all ΔT. We also find that φ_9 is falsified by keeping full throttle and full braking continuously. This means that φ_9 is easy to falsify yet only RL-based methods can falsify φ_9.

The results may suggest that if the property has a structure which can be exploit for fast falsification, reinforcement learning is advantageous to general purpose robustness guided falsification, and if there is no such structure, random inputs are sufficient for falsification.

8 Evaluation using PTC model

8.1 Model description

We also evaluate our method with a widely used model, the power train controller (**PTC**) model provided by Toyota Technical Center [56]. This model presents a controller for air-fuel (A/F) ratio for an internal combustion engine. Fig. 6 shows the architecture of **PTC**-model. The model takes the pedal angle and the engine speed as inputs. It outputs the verification measurement, which indicates the control error, and operational mode. The PTC system consists of two parts, i.e., the Fuel Control System (FCS) and the Verification and Validation stub system (VVSS). FCS takes pedal angle and the engine speed as inputs, computes the ideal A/F ratio and control the physical system to achieve this A/F ratio. It also models the engine dynamics to predict the real A/F ratio. FCS has two outputs, the A/F which indicates the real A/F ratio, and the A/F ref which indicates the goal A/F ratio set by the controller. VVSS takes A/F and A/F ref as input, and computes the control error and outputs as verification measurement.

There are 4 modes, i.e., startup, normal, power and fault, in the system. The system is in the startup mode within the first few seconds. Then the system enters the normal mode and stays at the normal mode if the pedal angle is within $(8.8^\circ, 70^\circ)$. If the pedal angle exceeds 70°, then the system

TABLE 11: The list of the evaluated properties on **PTC** model.

id	Formula
φ_{26}	$\Box_{[11,50]}\lvert\mu\rvert \leq 0.2$
φ_{27}	$\Box_{[11,50]}(\text{rise} \vee \text{fall} \implies \Box_{[1,5]}\lvert\mu\rvert \leq 0.15)$
φ_{30}	$\Box_{[11,50]}\mu \geq -0.25$
φ_{31}	$\Box_{[11,50]}\mu \leq 0.2$
φ_{32}	$\Box_{[11,50]}(\text{power} \wedge \Diamond_{[0,0.1]}\text{normal} \implies \Box_{[1,5]}\lvert\mu\rvert \leq 0.2)$
φ_{33}	$\Box_{[11,50]}(\text{power} \implies \lvert\mu\rvert \leq 0.2)$
φ_{34}	$\Box_{[0,50]}(\text{sensr}_{\text{f}}\text{ail} \implies \Box_{[1,5]}\lvert\mu\rvert \leq 0.15)$

enters the power mode. Finally, if a sensor fails, the system enters the fault mode.

8.2 Properties

We adopt formulas (26), (27), (30)–(34) defined in the original paper [56]. For simplicity, we use μ to represent Verification Measurement. The predicate "normal" means that the system operates under the normal mode. The predicate "power" means that the system operates under the power mode. The predicate "sensor_fail" means that a sensor failure event occurs. The predicate "rise" means the pedal angle jumps up to a high value from a low value. The predicate "fall" means the pedal angle falls down from a high value to a low value.

We make several modifications to the original formulas. First, we generally increase the tolerance for μ to make falsification harder. We also modify the definitions of rise and fall. In the original paper [56], the input to the system is limited to a pulse train signal. Our approach uses a piecewise constant function for the input, it is difficult to find the input which satisfies the constraints originally used. Therefore, we relax the condition and define

$$\text{rise} \equiv (\theta \leq 25^\circ) \wedge \Diamond_{[0,0.1]}(\theta \geq 45^\circ) \tag{38}$$

$$\text{fall} \equiv (\theta \geq 45^\circ) \wedge \Diamond_{[0,0.1]}(\theta \leq 25^\circ). \tag{39}$$

We also change open time intervals (e.g., $\Diamond_{(0,\epsilon)}$) in the original work to closed time intervals (e.g., $\Diamond_{[0,0.1]}$). Constants in the properties are set as $\tau_s = 11$, $T = 50$, $\eta = 1$, $\xi = 10$, $\epsilon = 0.1$. The other parameters remain the same.

φ_{26} states that the control error is within 0.2. φ_{27} states that even if the large change of the pedal angle occurs, the control error dissipates and becomes smaller than 0.15 within 1 second and remains until 5 seconds later. To falsify φ_{26} and φ_{27}, we force the pedal angle to be within $[8.8^\circ, 70^\circ]$ so that the system stays in the normal mode. φ_{30} and φ_{31} intend to ensure that the control error remains reasonable even under the low accuracy of sensors and actuators. These formulas are tested under the extreme tolerance $c_{23} = 1.05, c_{24} = 1.01, c_{25} = 1.05$ and $c_{23} = 0.95, c_{24} = 0.99, c_{25} = 0.95$ respectively. These parameters cause the steady state to real A/F ratio. We set the pedal angle within $[8.8^\circ, 70^\circ]$ as $\varphi_{26}, \varphi_{27}$ to ensure the normal mode of operation. φ_{32} states the requirement for the transition out of the power mode. Now we assume that the pedal angle is within $[8.8^\circ, 90^\circ]$ and when the system exits from the power mode, the control error dissipates fast and becomes smaller than 0.2 after 1 second and remains within 5 second. φ_{33} states the requirement for the control error when the system is under the power mode. Similarly, we assume that the pedal angle is within $[70^\circ, 90^\circ]$ to enforce the system into the power mode. Finally, φ_{34} states the requirement when the sensor failure occurs. We inject the sensor failure to the model 15 seconds after the beginning of the simulation.

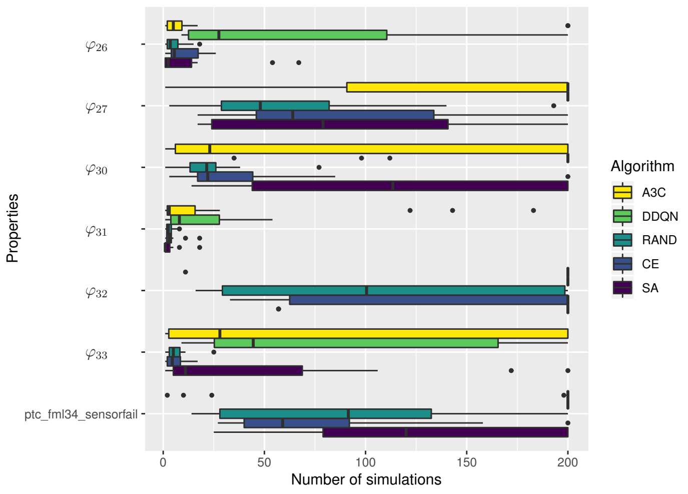


Fig. 7: number of episodes required to falsify **PTC** model

TABLE 12: Success rates to falsify PTC model

Properties	A3C	DDQN	RAND	CE	SA
φ_{26}	90	85	**100**	**100**	**100**
φ_{27}	45	0	**100**	80	80
φ_{30}	70	15	**100**	95	65
φ_{31}	**100**	**100**	**100**	**100**	**100**
φ_{32}	5	0	**75**	40	15
φ_{33}	65	75	**100**	**100**	95
φ_{34}	20	0	85	**95**	60

8.3 Experiment settings

For each given property, we run the falsification procedure 20 times due to the time consuming nature of the **PTC** model. For each falsification procedure, we execute simulation episodes up to 200 times and measure the number of simulation episodes required to falsify the property. If the property cannot be falsified within 200 episodes, the procedure fails. We choose $\Delta_T = 5$ because the original paper specifies that the input is the pulse period ζ, where $10 \leq \zeta \leq 30$. We do not use the periodic pulse for the input but we assume that the change of the input only occurs every $\zeta/2$ seconds. We generate a piecewise constant for inputs because an input specified by the original paper is a pulse. In the original paper, the engine speed is constant but we also change the engine speed during simulation since reinforcement learning methods cannot synthesize fixed parameters. Δ_T and the interpolation method for the engine speed is same as the pedal angle.

8.4 Evaluation results

Table 12 shows the success rate of falsification within 200 simulations for each formula and method. Table 13 shows the median number required to falsify the model for each formula and method. If all falsification attempts fail, the median number of episodes is indicated by –. Fig. 7 shows

TABLE 13: Median number of episodes when falsification succeeds

Properties	A3C	DDQN	RAND	CE	SA
φ_{26}	3	22	3.5	5.5	**2.5**
φ_{27}	**30**	–	48	53	72
φ_{30}	**7**	98	21.5	22	44
φ_{31}	3	8	2.5	3	**1**
φ_{32}	**11**	–	56	55.5	57
φ_{33}	**3**	40	5.0	4.5	11.0
φ_{34}	**17**	–	81.0	59.0	81.0

TABLE 14: Relative effect size measure comparing to random inputs on PTC model

Properties	A3C	DDQN	CE	SA
φ_{26}	0.543	*0.964*	0.624	0.456
φ_{27}	0.755	*0.999*	0.634	0.632
φ_{30}	0.543	*0.995*	0.595	0.840
φ_{31}	0.628	0.794	0.522	0.356
φ_{32}	*0.831*	*0.875*	0.680	*0.804*
φ_{33}	0.679	*0.979*	0.455	0.693
φ_{34}	0.790	*0.925*	0.434	0.657

the distributions of number of episodes required to falsify the model for each formula and method in a box plot. As we can see, random inputs show the most stable performance and for most properties, baselines outperform our methods. For RL-based methods, A3C is better than DDQN.

The analysis of the relative effect size measure verifies this observations. Table 14 indicates that all methods under-perform or has no significant difference from RAND. Further, DDQN consistently under-performs RAND for all properties. Table 15 shows the comparison of RL methods with CE and SA. For all cases, RL methods under-perform or has no significant difference from CE and SA.

The reasons of bad performance of reinforcement learning methods could be the combination of the following facts.

1) Lack of a learnable structure in the model;
2) lack of output of internal states from the model; and
3) fast dynamics of the system compared to the sampling interval $\Delta_T = 5$.

The good performance of RAND suggests that the model behaviour may not have an easily learnable structure. Moreover, the **PTC** model only outputs the control error and the operational mode, and does not allow observing of internal dynamics of the system. This prevents DRL algorithms from learning knowledge. This could also explain the better performance of A3C compared to DDQN. Our implementation of A3C uses LSTM neural architecture which can learn a structure of sequential data. Using LSTM would help to learn the model behaviours based on the context information. On the contrary, DDQN only uses a feed-forward network, the input of which is the observation of the previous step. Therefore, DDQN cannot "guess" the current state of the system from the sequence of previous observations. Finally, the control error, which is the only observable state of the system, usually converges to 0 in much shorter time than $\Delta_T = 5$. Therefore, it is difficult to learn knowledge from the model observations.

TABLE 15: Relative effect size measure comparing to S-Taliro on PTC model

Properties	Base lines	A3C	DDQN
φ_{26}	CE	0.429	*0.850*
	SA	0.569	*0.874*
φ_{27}	CE	0.653	*0.900*
	SA	0.655	*0.900*
φ_{30}	CE	0.491	*0.956*
	SA	0.332	0.761
φ_{31}	CE	0.607	0.772
	SA	0.740	*0.843*
φ_{32}	CE	0.665	0.700
	SA	0.546	0.575
φ_{33}	CE	0.722	*0.983*
	SA	0.575	0.769
φ_{34}	CE	0.828	*0.975*
	SA	0.670	*0.800*

9 Analysis

In this section, we analyse the factors that may affect the evaluation results of our approach and discuss the observations in detail.

9.1 Analysis of impact of log-sum-exp approximation

The log-sum-exp approximation of max (29) has a more general form, i.e.,

$$\max(x_1, \ldots, x_n) \sim \frac{1}{\alpha} \log \left\{ \sum_{i=1}^{n} e^{\alpha x_i} \right\} \tag{40}$$

for $\alpha > 0$. Larger α provides better approximation, while the approximation is less smooth respect to $x_1, \ldots, x_n$ and thus the reward is harder to learn. To investigate the impact of different α on the experiment results, we evaluate the **CARS** model on $\alpha = 0.25, 0.5, 1, 2, 4$ and report the results. Fig. 8 shows the box diagram indicating number of episodes needed to falsify the formulas. Table 16 shows the success rate of falsification for each combination of a formula, RL method (A3C and DDQN) and α. Similarly, Table 17 shows the median number of episodes when the falsification is successful, on different combinations of formula, RL method (A3C and DDQN) and α.

As can be seen from Fig. 8, Table 16 and Table 17, there is no consistent tendency of performance for different α. Indeed, Tukey's method (all-pairs comparison) using `nparcomp` does not find statistically significant differences between any different α for all pairs of formulas and RL algorithms. Therefore, we conclude that there is not significant impact of different α to the performance of our method and we set α to 1 for simplicity.

9.2 Execution time

Our work utilizes the number of simulations, rather than execution time, as the performance measurement. We evaluate the effects that different methods, ΔT and properties, have on execution time. The evaluation results justify our decisions on using number of simulations as the performance measurement.

Fig. 9 shows the execution time per one simulation run for each combination of algorithms, ΔT and properties

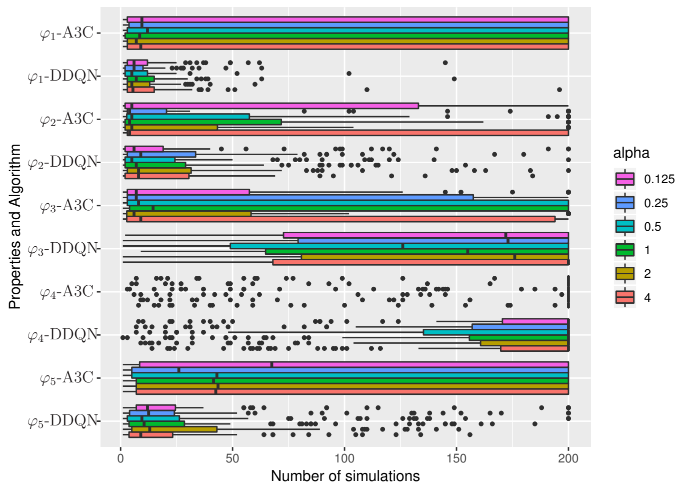


Fig. 8: Number of episodes required to falsify the model using different α on **CARS**

TABLE 16: Success rate of falsifying the model using different α on **CARS**

α	0.125	0.25	0.5	1	2	4
φ_1-A3C	71	73	70	66	73	70
φ_1-DDQN	100	100	100	100	100	100
φ_2-A3C	76	85	83	78	78	74
φ_2-DDQN	98	100	99	100	99	100
φ_3-A3C	82	76	67	73	78	75
φ_3-DDQN	57	56	61	59	54	48
φ_4-A3C	14	13	24	16	15	19
φ_4-DDQN	29	34	35	36	33	26
φ_5-A3C	65	68	68	64	57	67
φ_5-DDQN	95	97	97	100	100	100

$\varphi_1, \ldots, \varphi_9$ for the **AT** model. Execution time differs between RL based algorithms and S-TaLiRo based algorithms. This is expected since S-Taliro is implemented with Matlab and C, while RL based algorithms are implemented with Matlab, Python and C. According to Mathworks support center, calling Python prevents optimization. There is no significant difference between RL based algorithms and RAND. This suggests that the overhead of machine learning component contributes little to the total execution time, and simulation component dominates execution time. This is also supported by the fact that there is no difference on time consumptions between different ΔT. If machine learning component has large influence, small ΔT increases learning iterations and thus should result in slower execution. As for properties, there is no influence on performance by properties under falsification, except φ_5. The result of φ_5 is caused by initialization overhead to load a new simulation model, because φ_5 needs only a small number of simulations to falsify and the initialization overhead dominates the execution time. Interestingly, there is no significant difference between $\varphi_1 - \varphi_4$, φ_6 and $\varphi_7 - \varphi_9$. For $\varphi_7 - \varphi_9$, we use longer simulation time. This may suggest that execution time is further dominated by initialization of each simulation.

The evaluation results show that there is large variance of execution time among different combination of methods, ΔT and property under falsification. This can be explained by the fact that we are using a single multicore machine

TABLE 17: Median number of episodes when falsification succeeds using different α on **CARS**

α	0.125	0.25	0.5	1	2	4
φ_1-A3C	4	5	4	3	4	4
φ_1-DDQN	6	6	5	7	5	5.5
φ_2-A3C	3	4	4	3	3	3
φ_2-DDQN	6	9	5	7	8	8
φ_3-A3C	4	5	4	6	4	4
φ_3-DDQN	84	81.5	64	91	81.5	64.5
φ_4-A3C	57.5	49	104.5	58	49	59
φ_4-DDQN	117	115.5	102	107	108	68
φ_5-A3C	15	9.5	9.5	11	8	13
φ_5-DDQN	11	12	8	10.5	13	9

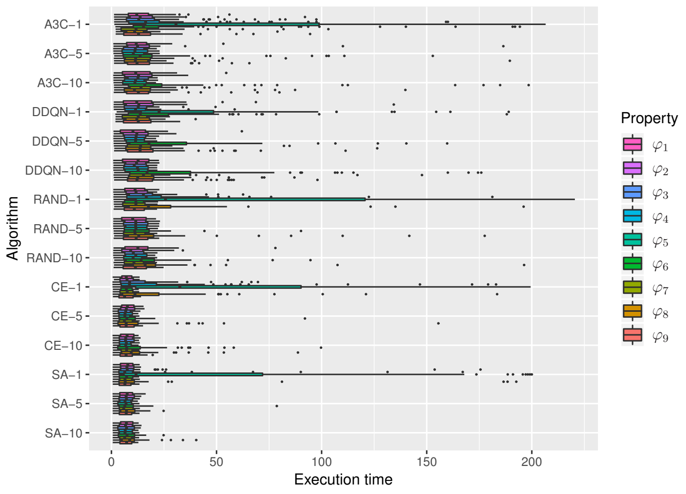


Fig. 9: Difference of execution time depending on algorithm, ΔT and properties on **AT**

for the experiment, running benchmarks simultaneously, therefore execution time becomes non-deterministic. Since we observe that the simulation time dominates the total execution time, we use the number of simulations to measure performance of each method to avoid the non-determinism caused by the experiment environment as well as programming languages, etc.

9.3 Observable states

Reinforcement learning algorithms often, although not always, assume that the states of the environment are completely observable from the agent. Therefore, it is expected that the availability of the state information of the system influences the performance of reinforcement learning based methods.

The experimental results support this expectation. This is most evidenced in the experiment of the **CARS** model, in which we measure the performance of two variants of reinforcement learning, i.e., the white-box RL approaches and the black-box RL approaches. A white-box RL can observe all system states while a black-box RL cannot observe system states at all, and only obtains the reward corresponding to the system states. As shown in Table 4, black-box RLs have the same performance as RAND and often under-perform CE and SA. White-box RLs almost always outperform CE, SA and RAND. The non-observability of system states also may cause bad performance of RL based methods for the **PTC** model, because the **PTC** model only allows observation of the control error and the operational mode. Similarly, for φ_3 and φ_4 of the **ATC** model, the bad performance is likely because φ_3 and φ_4 concern events which occur in a short period of time.

9.4 Property to be falsified

Similar to the observability of states, we are interested to understand whether there are properties, on which the reinforcement learning methods consistently outperforms other methods. However, from the experimental results, we cannot find any consistent description of such a property. Our main observation is, whenever SA or CE outperforms RL methods, RAND almost always outperforms RL methods, which is shown in the case of φ_3 and φ_4 of the **AT** model and all properties of the **PTC** model. Moreover, in such cases, RAND outperforms or has no statistically significant difference with SA and CE. This suggests that there are two kinds of properties, one can be efficiently falsified by RAND and the other requires a structured input to be falsified. RL based methods show better performance on the latter.

9.5 Model

Dynamics of a model seems to contribute to the difference of performance between A3C and DDQN. For the **CARS** and the **PTC** models, A3C generally has better performance compared to DDQN. For **AT** model, DDQN generally has better performance compared to A3C. A3C and DDQN are competing methods and has similar performance. However, we use them in different settings. We use A3C with a single thread. A3C is supposed to be used in a multi-threaded environment to stabilize learning. Therefore, our setting may adversely affect the performance of A3C. On the other-hand, our implementation of A3C uses LSTM, which can memorize past observations and rewards. Because our implementation of DDQN uses a simple feed forward network from the current state, if the system behaviour depends on the past states, A3C has an advantage. For example, the behaviour of the **CARS** model depends on velocity of each car, which can only be computed by using past observations.

9.6 The ΔT

We investigate the effect of sampling interval ΔT on performances using the **AT** model. The results show that there is no consistent effects of ΔT. It may be expected that small ΔT gives advantages to reinforcement learning, since small ΔT improves the opportunities to learn and control the system. However, it seems difficult to make an agent generate a constant system input for a long time. Therefore, small ΔT can adversely affect the performance if the system input must be kept constant for a long time in order to falsify the system. To conclude, Tables 2, 3, 7, 8, 12 and 13 suggest that reinforcement learning methods may not always be the best methods, but both A3C and DDQN show stable performance regardless of the property to be falsified for all models. For some cases, cross entropy method (CE) outperforms reinforcement learning based methods. However CE is not stable. Moreover, we observe that when reinforcement learning methods under-perform CE and SA, then RAND outperforms them for almost all cases. The results of relative effect size measure also support these analysis. We can claim (with our experiment results) that reinforcement learning based methods provides the most stable performance. For some situations in which RAND is good enough, RL based method may not show its advantages. Based on the above observations, we believe that an adaptive method, which combines the reinforcement learning with RAND may be a good direction to explore.

10 Conclusion and Future Work

In this paper, we propose a new method of falsification of CPS using reinforcement learning. Our method focuses on safety properties. We theoretically show how to format a CPS falsification problem into a reinforcement learning problem. We also experimentally investigate and compare the performance of the proposed methods with three baseline methods, i.e., uniformly random sampling, cross entropy and simulated annealing. To provide a result with good confidence, we employ a rigorous statistical method called relative effect size measure for experiment results analysis. Through a thorough analysis on the experiment results, we identify the factors which make reinforcement learning approaches outperform baselines. The first factor is the availability of the system states to the reinforcement learning agent. The other factor is the existence of a structure in the falsifying input.

As future works, there are three directions that we would like to explore. First, we would like to further explore the criteria/scenarios in which reinforcement learning works better than uniform random input. We are also interested to extend our method to non-safety properties. Second, technically finding practical ways to tune deep learning parameters may help improve the effectiveness of our approach. Deep learning tends to have large number of parameters, which must be tuned. Finally, we want to extend our experimental study to involve more models, properties and different methods.

References

[1] H. Abbas and G. E. Fainekos, "Convergence proofs for simulated annealing falsification of safety properties," in *50th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2012, Allerton Park & Retreat Center, Monticello, IL, USA, October 1-5, 2012.* IEEE, 2012, pp. 1594–1601. [Online]. Available: https://doi.org/10.1109/Allerton.2012.6483411

[2] H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivančić, and A. Gupta, "Probabilistic temporal logic falsification of cyber-physical systems," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, pp. 95:1–95:30, May 2013. [Online]. Available: http://doi.acm.org/10.1145/2465787.2465797

[3] Y. Annpureddy, C. Liu, G. E. Fainekos, and S. Sankaranarayanan, "S-taliro: A tool for temporal logic falsification for hybrid systems," in *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, 2011, pp. 254–257.

[4] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-time systems*, vol. 2, no. 4, pp. 255–299, 1990.

[5] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems.* Springer, 2004, pp. 152–166.

[6] S. Sankaranarayanan and G. E. Fainekos, "Falsification of temporal properties of hybrid systems using the cross-entropy method," in *Hybrid Systems: Computation and Control (part of CPS Week 2012), HSCC'12, Beijing, China, April 17-19, 2012*, T. Dang and I. M. Mitchell, Eds. ACM, 2012, pp. 125–134. [Online]. Available: http://doi.acm.org/10.1145/2185632.2185653

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[9] H. L. S. Younes and R. G. Simmons, "Statistical probabilistic model checking with a focus on time-bounded properties," *Inf. Comput.*, vol. 204, no. 9, pp. 1368–1409, Sep. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.ic.2006.05.002

[10] E. Clarke, A. Donzé, and A. Legay, "Statistical model checking of mixed-analog circuits with an application to a third order $\Delta - \Sigma$ modulator," in *Proceedings of the 4th International Haifa Verification Conference on Hardware and Software: Verification and Testing*, ser. HVC '08. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 149–163. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01702-5_16

[11] E. M. Clarke and P. Zuliani, "Statistical model checking for cyber-physical systems." in *ATVA*, vol. 11. Springer, 2011, pp. 1–12.

[12] P. Zuliani, A. Platzer, and E. M. Clarke, "Bayesian statistical model checking with application to simulink/stateflow verification," in *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '10. New York, NY, USA: ACM, 2010, pp. 243–252. [Online]. Available: http://doi.acm.org/10.1145/1755952.1755987

[13] R. Grosu and S. A. Smolka, "Monte carlo model checking." in *TACAS*, vol. 3440. Springer, 2005, pp. 271–286.

[14] J. Kapinski, B. H. Krogh, O. Maler, and O. Stursberg, "On systematic simulation of open continuous systems," in *HSCC*, vol. 3. Springer, 2003, pp. 283–297.

[15] T. Nghiem, S. Sankaranarayanan, G. Fainekos, F. Ivancić, A. Gupta, and G. J. Pappas, "Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems," in *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*. ACM, 2010, pp. 211–220.

[16] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Grenoble, France, September 22-24, 2004, Proceedings*, ser. Lecture Notes in Computer Science, Y. Lakhnech and S. Yovine, Eds., vol. 3253. Springer, 2004, pp. 152–166. [Online]. Available: https://doi.org/10.1007/978-3-540-30206-3_12

[17] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010. Proceedings*, ser. Lecture Notes in Computer Science, K. Chatterjee and T. A. Henzinger, Eds., vol. 6246. Springer, 2010, pp. 92–106. [Online]. Available: https://doi.org/10.1007/978-3-642-15297-9_9

[18] Y. Annpureddy, C. Liu, G. Fainekos, and S. 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*. Springer, 2011, pp. 254–257.

[19] G. E. Fainekos, S. Sankaranarayanan, K. Ueda, and H. Yazarel, "Verification of automotive control applications using s-taliro," in *American Control Conference (ACC)*, 2012, pp. 3567–3572.

[20] Y. S. R. Annapureddy and G. E. Fainekos, "Ant colonies for temporal logic falsification of hybrid systems," in *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*, Nov 2010, pp. 91–96.

[21] J. V. Deshmukh, X. Jin, J. Kapinski, and O. Maler, "Stochastic local search for falsification of hybrid systems," in *Automated Technology for Verification and Analysis - 13th International Symposium, ATVA 2015, Shanghai, China, October 12-15, 2015, Proceedings*, ser. Lecture Notes in Computer Science, B. Finkbeiner, G. Pu, and L. Zhang, Eds., vol. 9364. Springer, 2015, pp. 500–517. [Online]. Available: https://doi.org/10.1007/978-3-319-24953-7_35

[22] S. Yaghoubi and G. Fainekos, "Falsification of temporal logic requirements using gradient based local search in space and time," in *6th IFAC Conference on Analysis and Design of Hybrid Systems, ADHS 2018, Oxford, UK, July 11-13, 2018*, ser. IFAC-PapersOnLine, A. Abate, A. Girard, and M. Heemels, Eds., vol. 51, no. 16. Elsevier, 2018, pp. 103–108. [Online]. Available: https://doi.org/10.1016/j.ifacol.2018.08.018

[23] A. Donzé, "Breach, A toolbox for verification and parameter synthesis of hybrid systems," in *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, ser. Lecture Notes in Computer Science, T. Touili, B. Cook, and P. B. Jackson, Eds., vol. 6174. Springer, 2010, pp. 167–170. [Online]. Available: https://doi.org/10.1007/978-3-642-14295-6_17

[24] T. Akazaki, "Falsification of conditional safety properties for cyber-physical systems with gaussian process regression," in *Runtime Verification - 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings*, ser. Lecture Notes in Computer Science, Y. Falcone and C. Sánchez, Eds., vol. 10012. Springer, 2016, pp. 439–446. [Online]. Available: https://doi.org/10.1007/978-3-319-46982-9_27

[25] N. Polikarpova and S. Schneider, Eds., *An Active Learning Approach to the Falsification of Black Box Cyber-Physical Systems*, ser. Lecture Notes in Computer Science, vol. 10510. Springer, 2017. [Online]. Available: https://doi.org/10.1007/978-3-319-66845-1_1

[26] S. Silvetti, A. Policriti, and L. Bortolussi, "An active learning approach to the falsification of black box cyber-physical systems," *CoRR*, vol. abs/1705.01879, 2017. [Online]. Available: http://arxiv.org/abs/1705.01879

[27] J. V. Deshmukh, M. Horvat, X. Jin, R. Majumdar, and V. S. Prabhu, "Testing cyber-physical systems through bayesian optimization," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5, pp. 170:1–170:18, 2017. [Online]. Available: https://doi.org/10.1145/3126521

[28] *Falsification of LTL safety properties in hybrid systems*. Springer, 2009.

[29] T. Dreossi, T. Dang, A. Donzé, J. Kapinski, X. Jin, and J. V. Deshmukh, "Efficient guiding strategies for testing of temporal properties of hybrid systems," in *NASA Formal Methods - 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings*, ser. Lecture Notes in Computer Science, K. Havelund, G. J. Holzmann, and R. Joshi, Eds., vol. 9058. Springer, 2015, pp. 127–142. [Online]. Available: https://doi.org/10.1007/978-3-319-17524-9_10

[30] T. Akazaki, S. Liu, Y. Yamagata, Y. Duan, and J. Hao, "Falsification of cyber-physical systems using deep reinforcement learning," in *Formal Methods - 22nd International Symposium, FM 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 15-17, 2018, Proceedings*, ser. Lecture Notes in Computer Science, K. Havelund, J. Peleska, B. Roscoe, and E. P. de Vink, Eds., vol. 10951. Springer, 2018, pp. 456–465. [Online]. Available: https://doi.org/10.1007/978-3-319-95582-7_27

[31] G. De Giacomo, L. Iocchi, M. Favorito, and F. Patrizi, "Reinforcement learning for ltlf/ldlf goals," *CoRR*, vol. abs/1807.06333, 2018. [Online]. Available: http://arxiv.org/abs/1807.06333

[32] A. Camacho, O. Chen, S. Sanner, and S. A. McIlraith, "Non-markovian rewards expressed in LTL: guiding search via reward shaping," in *Proceedings of the Tenth International Symposium on Combinatorial Search, SOCS 2017, 16-17 June 2017, Pittsburgh, Pennsylvania, USA.*, A. Fukunaga and A. Kishimoto, Eds. AAAI Press, 2017, pp. 159–160. [Online]. Available: https://aaai.org/ocs/index.php/SOCS/SOCS17/paper/view/15811

[33] *Decision-Making with Non-Markovian Rewards: From LTL to automata-based reward shaping*, 2017, see also University of Toronto Technical Report CSRG-632. [Online]. Available: http://www.cs.toronto.edu/~sheila/publications/cam-etal-rldm17.pdf

[34] X. Li, C. I. Vasile, and C. Belta, "Reinforcement learning with temporal logic rewards," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*. IEEE, 2017, pp. 3834–3839. [Online]. Available: https://doi.org/10.1109/IROS.2017.8206234

[35] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "Q-learning for robust satisfaction of signal temporal logic specifications," in *55th IEEE Conference on Decision and Control, CDC 2016, Las Vegas, NV, USA, December 12-14, 2016*. IEEE, 2016, pp. 6565–6570. [Online]. Available: https://doi.org/10.1109/CDC.2016.7799279

[36] A. Jones, D. Aksaray, Z. Kong, M. Schwager, and C. Belta, "Robust satisfaction of temporal logic specifications via reinforcement learning," *CoRR*, vol. abs/1510.06460, 2015. [Online]. Available: http://arxiv.org/abs/1510.06460

[37] H. Ho, J. Ouaknine, and J. Worrell, "Online monitoring of metric temporal logic," in *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September*

22-25, 2014. Proceedings, ser. Lecture Notes in Computer Science, B. Bonakdarpour and S. A. Smolka, Eds., vol. 8734. Springer, 2014, pp. 178–192. [Online]. Available: https://doi.org/10.1007/978-3-319-11164-3_15

[38] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, *Handbook of model checking*. Springer, 2018.

[39] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications," in *Formal Approaches to Software Testing and Runtime Verification, First Combined International Workshops, FATES 2006 and RV 2006, Seattle, WA, USA, August 15-16, 2006, Revised Selected Papers*, ser. Lecture Notes in Computer Science, K. Havelund, M. Núñez, G. Rosu, and B. Wolff, Eds., vol. 4262. Springer, 2006, pp. 178–192. [Online]. Available: https://doi.org/10.1007/11940197_12

[40] C. Szepesvári and G. Bartok, "Algorithms for Reinforcement Learning (Errata)," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 4, no. x, pp. 1–103, 2010.

[41] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," 2011.

[42] ——, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[43] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[44] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 2829–2838. [Online]. Available: http://proceedings.mlr.press/v48/gu16.html

[45] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," vol. 48, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[46] J. D. Cook, "Basic properties of the soft maximum," 2011.

[47] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[48] A. Dokhanchi, B. Hoxha, and G. Fainekos, "On-line monitoring for temporal logic robustness," in *International Conference on Runtime Verification*. Springer, 2014, pp. 231–246.

[49] *The ChainerRL Library*, https://github.com/chainer/chainerrl.

[50] H. Abbas, G. E. Fainekos, S. Sankaranarayanan, F. Ivancic, and A. Gupta, "Probabilistic temporal logic falsification of cyber-physical systems," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 2s, pp. 95:1–95:30, 2013. [Online]. Available: http://doi.acm.org/10.1145/2465787.2465797

[51] F. Konietschke, M. Placzek, F. Schaarschmidt, and L. A. Hothorn, "nparcomp : An R Software Package for Nonparametric Multiple Comparisons and Simultaneous Confidence Intervals," *Journal of Statistical Software*, vol. 64, no. 9, pp. 1–17, 2015. [Online]. Available: http://www.jstatsoft.org/index.php/jss/article/view/v064i09/v64i09.pdf

[52] N. Cliff, *Ordinal methods for behavioral data analysis*. Psychology Press, 2014.

[53] J. Hu, J. Lygeros, and S. Sastry, "Towards a Theory of Stochastic Hybrid Systems," pp. 160–173, 2000. [Online]. Available: http://link.springer.com/10.1007/3-540-46430-1{_}16

[54] *Modeling an Automatic Transmission Controller*, https://www.mathworks.com/help/simulink/examples/modeling-an-automatic-transmission-controller.html.

[55] H. A. Bardh Hoxha and G. Fainekos, "Benchmarks for temporal logic requirements for automotive systems," *Proc. of Applied Verification for Continuous and Hybrid Systems*, 2014.

[56] X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda, and K. Butts, "Powertrain control verification benchmark," in *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '14. New York, NY, USA: ACM, 2014, pp. 253–262. [Online]. Available: http://doi.acm.org/10.1145/2562059.2562140

Yamagata, Yoriyuki is currently a senior researcher at National Institute of Advanced Industrial Science and Technology (AIST). He received PhD degree in mathematical science from University of Tokyo in 2002. He worked as postdoctoral researcher in AIST from 2004 to 2005, and researcher in AIST from 2005. His research interests include theoretical computer science, software engineering and verification.

Shuang Liu is currently an associate professor at Tianjin University, China (TJU). She received Bachelor degree in Renmin University of China in 2010, PhD degree from National University of Singapore in 2015. She worked as a research fellow in SUTD during 2015-2016, and lecturer in SiT during 2016-2017. She has been a faculty member of TJU since 2018. Shuang's research interests include software engineering, formal methods and privacy protection.

Takumi Akazaki is currently a researcher at Fujitsu Laboratories, Japan. He received his bachelor and master degree at The University of Tokyo in 2013, 2015 and was a PhD student there in 2015-2018. His research interests include formal methods, numerical optimization and machine learning.

Yihai Duan is a master student as Tianjin University, China (TJU). He received bachelor degree in Tianjin University in 2018. Yihai's major is Computer Science and his research interests include deep reinforcement learning and software engineering.

Jianye Hao is currently an associate professor at Tianjin University, China. He received his bachelor degree at Harbin Institute of Technology in 2008 and PhD at The Chinese University of Hong Kong in 2013. His research interests include reinforcement learning and multi-agent systems in general