This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3056360, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

PDF: Path-Oriented, Derivative-Free Approach for Safety Falsification of Nonlinear and Nondeterministic CPS

Jiawan Wang, Lei Bu, *Member, IEEE,* Shaopeng Xing, and Xuandong Li

***Abstract*—Cyber-physical systems (CPS) integrate discrete computations with continuous physical processes and can be highly nonlinear and nondeterministic. Unlike the verification of CPS, which is difficult to handle, the falsification of CPS fulfills certain requirements from testing by seeking witness behavior of these systems and is easier to conduct. However, existing falsification techniques may fail to support the general complex CPS in practice because they usually focus on certain restricted classes of systems.**

In this paper, we present a path-oriented, derivative-free approach to falsify safety properties in nonlinear and nondeterministic CPS. In our approach, we model the behavior of CPS by hybrid automata. Then, we enumerate candidate paths of hybrid automata, transform the feasibility of candidate paths into optimization problems and solve these optimization problems by our newly proposed classification-model-based, derivative-free optimization algorithm. We also provide two novel pruning techniques to further improve the efficiency and efficacy of our approach: a nested optimization structure with better model refinements for continuous search space pruning and a hardly feasible path prefixes guided backtracking for discrete search space pruning. We implement our approach into a tool called PDF. Our experiments showed that PDF supported the safety falsification of CPS in all of our benchmarks, and it achieved success rates no lower than 95% in only seconds on 22/28 of the benchmarks.

***Index Terms*—Cyber-physical systems (CPS), falsification, reachability analysis, path-oriented, derivative-free optimization.**

I. Introduction

CYBER-PHYSICAL systems (CPS) are integrations of computations and physical processes, where physical processes and computations affect each other through feedback loops [1]. Most CPS combine both discrete and continuous dynamics, and they can be highly nonlinear and nondeterministic. To formalize the hybrid dynamics of CPS, hybrid automata (HA) [2] is naturally used as one of their modeling languages.

CPS play an important role in today's world and are widely applied in many areas, including safety-critical areas such as digital medical instruments, aerospace control, and autonomous systems. It is important to keep such complex and dynamic systems working safely. Thus, the critical issue of the reachability verification of HA has attracted massive attention over the past two decades [3]–[7]. However, due to the high complexity of HA's behavior, the verification of HA is tough, and the reachability verification problem of HA is indeed proven to be undecidable [8]. At the current stage, the verification of nonlinear HA is still a huge obstacle, and the class of HA that could be verified efficiently is generally limited to HA with simple dynamics and few or no external inputs.

As the reachability verification of nonlinear complex HA is hard to apply in most cases, recent studies have proposed the falsification of HA as a complementary approach, and appreciable progress has been achieved. Instead of providing a rigorous proof, falsification of HA attempts to seek a concrete counterexample—a simulated witness trajectory that violates some given properties in systems—by testing. Compared with verification techniques, testing-based falsification techniques can be better applied to nonlinear and industrial-scale CPS. The falsification of HA is conducted in the following two main directions: motion-planning-based falsification [9]–[12] and optimization-based falsification [13]–[17]. Besides, other methods, such as gradient-based ones [18], [19] and the learning-based one [20], have also been proposed recently.

Motion-planning-based HA falsification explores the systems' state space by Rapidly Random Tree (RRT) techniques [21]. Different metrics, such as the coverage degree of the state space [11] and the robustness of the given safety property [22], [23], have been utilized to guide the random simulations during RRT construction [12]. Although RRT techniques have shown their abilities in robotics and motion planning, their performance in HA falsification is limited.

Optimization-based HA falsification performs a random walk by sampling from a parameterized search space. Most of these optimization-based works assume that the systems under investigation are with deterministic semantics, which means that given certain external inputs and initial state, the behavior of the system is unique and certain. Thus, the search space of these works typically consists of the external inputs and the initial state of systems. Optimization-based HA falsification simulates HA's behavior with an initial sample generated from the search space. Various heuristic search methods—such as Simulated Annealing [15], Ant-Colony [16], and Cross-Entropy [24]—would be deployed to mutate the current sample iteratively until a feasible sample is located in the parame-

Manuscript received January 8, 2020; revised May 30, 2020, September 29, 2020, and November 28, 2020; accepted January 16, 2021. This work is supported in part by the Leading-edge Technology Program of Jiangsu Natural Science Foundation (No. BK20202001) and the National Natural Science Foundation of China (No.61632015, No.61690204). *(Corresponding author: Lei Bu.)*

The authors are with the State Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing, 210023 China (e-mail: wangjw@smail.nju.edu.cn; bulei@nju.edu.cn).

0278-0070 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Nanjing University. Downloaded on July 18,2021 at 12:55:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3056360, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

terized search space. In this line of HA falsification, mature tools such as S-TaLiRo [14], Breach [13], and FalStar [17] have been developed. However, the dynamic of many complex CPS in the real world is nondeterministic, and works in this line may fail to support the general nondeterministic CPS. In addition, heuristic search methods used in these works usually lack sufficient theoretical guarantees and have many parameters that can greatly affect their performance.

In a nondeterministic hybrid automaton [25] with specific external inputs and initial state, the behavior of the automaton can be nondeterministic because of the uncertain choice between its continuous and discrete dynamics and the uncertain choice among its multiple enabled discrete dynamics. Thus, to find a witness behavior in a nondeterministic HA, the search space of this falsification problem would contain not only the external inputs and the initial state, but also the answers to these uncertain choices during the automaton's evolution.

To handle this large search space, we applied a two-layered path-oriented framework. Candidate paths of the HA were enumerated on the bottom layer, and the corresponding continuous space—consisting of the external inputs, the initial state, and the exact jumping time of locations in candidate paths—were searched on the top layer. To search the continuous space, we encoded the feasibility of each candidate path into an optimization problem, which was then solved by a classification-model-based derivative-free optimization (DFO) algorithm rather than by any heuristic search method. Recently, classification-model-based derivative-free methods have been proposed for nonlinear optimization [26], [27]. Such methods scale well to high-dimension tasks. They also have a grounded theory about the query complexity of the objective function and the rate of convergence. Thus, a specific classification-model-based DFO algorithm was designed in this work to solve the optimization problem on the top layer.

Furthermore, to enhance the efficiency of our approach, we proposed two light-weight techniques to prune the search space on the continuous and discrete levels respectively. First, to prune the continuous search space on the top layer, we proposed a nested classification-model-based DFO structure. Under this nested structure, the lower and the upper bounds of the jumping time of locations along the candidate paths can be computed, and these jumping time bounds can then guide better classification-model refinements during the DFO. Second, to prune the discrete search space on the bottom layer, we proposed the idea of 'hardly feasible path prefixes'. Guided by the hardly feasible path prefixes, backtracking can be performed during the generation of candidate paths.

All of these techniques have been implemented in a tool called PDF (***P***ath-Oriented, ***D***erivative-Free ***F***alsification) for the safety falsification of nonlinear and nondeterministic CPS. PDF has been evaluated thoroughly by well-known benchmarks. The current experiments showed that PDF can handle arbitrary nonlinear HA effectively and efficiently.

The main contributions of this work are:

1. We provided the first path-oriented framework to falsify the safety properties in nonlinear and nondeterministic HA. To the best of our knowledge, our falsification framework is also the first optimization-based one to handle the general nondeterminism in HA.
2. Instead of using heuristic search methods, we designed a learning-based DFO algorithm to check the feasibility of the continuous state space of candidate paths. We also designed two novel pruning algorithms under our framework to handle the large search space induced by the nondeterminism. Experiments showed that our learning-based DFO algorithm outperformed existing mature heuristic methods significantly.
3. We implemented a tool called PDF to falsify safety properties in HA. Experiments showed the effectiveness and efficiency of PDF, solving all our benchmarks, and achieving success rates no lower than 95% in only seconds on $22/28$ of them .

The rest of this paper is organized as follows. Backgrounds are given in section II. Descriptions of our two-layered path-oriented approach and our classification-model-based DFO algorithm are presented in section III. Two search space pruning techniques are presented in section IV. Implementation details and experiments are given in section V. We discuss related works in section VI and conclude in section VII.

II. Backgrounds

A. Problem Formulation

Definition 2.1: (Hybrid Automata) A hybrid automaton consists of a tuple $H = (L, X, U, E, F, Inv, G, R, S_0, S_f)$, where

L	discrete locations;
$X \subseteq \mathbb{R}^n$	space of continuous states;
$U \subseteq \mathbb{R}^m$	space of external inputs;
$E \subseteq L \times L$	discrete transitions;
$F = \{F_l : X \times U \to \mathbb{R}^n \mid l \in L\}$	flow functions;
$Inv = \{Inv_l \in 2^{X \times U} \mid l \in L\}$	invariant conditions;
$G = \{G_e \in 2^{X \times U} \mid e \in E\}$	guard conditions;
$R = \{R_e : X \times U \to X \mid e \in E\}$	reset functions;
$S_0 \in 2^{L \times X}$	initial conditions;
$S_f \in 2^{L \times X}$	unsafe conditions.

Given a hybrid automaton (HA) H, each discrete location $l \in L$ models a system operating mode, and its flow function F_l defines the differential equation of the system continuous states by $\dot{x} = F_l(x, \mu)$, where the continuous states $x \in X$ and the external inputs $\mu \in U$. If any of the functions or conditions in the definition of H is not linear, we say H is a nonlinear HA.

In this work, we assume that the external inputs in H are time-varying and piecewise smooth. Control points in a simulation time horizon are given to the external inputs, and the interpolation (such as piecewise constant, piecewise linear and splines) across these control points is performed to parameterize the external inputs.

Definition 2.2: (Semantics) Given a HA $H = (L, X, U, E, F, Inv, G, R, S_0, S_f)$, the state space of H is defined as $S = L \times X$, and the state of H is denoted by the pair $s = (l, x) \in S$. The state of H evolves in the two following ways:

- Continuous Evolution: by continuous dynamic and a time delay Δt, state $s = (l, x)$ can evolve to $s' = (l, x')$ ac-

0278-0070 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Nanjing University. Downloaded on July 18,2021 at 12:55:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3056360, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

cording to the flow function F_l, if the invariant condition Inv_l is always satisfied during Δt.

- Discrete Evolution: by a discrete and instantaneous transition, state $s = (l, x)$ can evolve to $s' = (l', x')$, if $(l, l') \in E$ and it is enabled. We say a discrete transition (l, l') in H is enabled when H is at location l and the guard condition $G_{(l,l')}$ is satisfied.

The nondeterminism of the dynamic in the HA is captured under this semantic. Given certain external inputs and the state s of a HA H, the evolvement of s may be uncertain. For one thing, if multiple discrete transitions are enabled, H can evolve through any one of them. For another, if both the continuous and the discrete evolution are allowed, H can either continue staying at the current location or evolve to another one. In fact, this nondeterminism is a common case for systems in practice.

In this work, given the initial state and external inputs of a HA, if the behavior of the HA is certain, we call it a deterministic HA, otherwise, it is a nondeterministic HA. To define a certain behavior of a nondeterministic HA, we give the definition of paths, jumping time sequences along paths, and control instances.

Definition 2.3: (Path) A path of a nondeterministic HA H is a location sequence $\rho = \{l_i\}_0^N$, such that

- $\exists x \in X,\ (l_0, x) \in S_0$;
- $\forall i \in [0, N),\ (l_i, l_{i+1}) \in E$.

Definition 2.4: (Jumping Time Sequence) A jumping time sequence of a path $\rho = \{l_i\}_0^N$ in a HA H is a collection $\tau = \{t_i\}_0^{N+1}$, such that

- $t_0 = 0$;
- $\forall i \in [0, N],\ t_i < t_{i+1}$.

The interpretation is that if a HA H evolves along a path $\rho = \{l_i\}_0^N$ with a jumping time sequence $\tau = \{t_i\}_0^{N+1}$ of ρ, H should stay in location l_i when $t \in [t_i, t_{i+1})$ $(i \in [0, N])$; in other words, the discrete transition (l_i, l_{i+1}) should take place at time instance t_{i+1} $(i \in [0, N))$. If multiple discrete transitions are enabled, the path guides H which transition to take. If both the discrete and the continuous evolution are allowed, the jumping time sequence provides H with exact dwell time in locations.

Definition 2.5: (Control Instance) A control instance of a HA H is a collection $\delta = (\rho, \tau, \mu, x)$, where ρ is a path, τ is a jumping time sequence of ρ, μ is the external inputs, and x is the initial values of the system continuous states.

Given a control instance of a nondeterministic HA H, the behavior of H is certain and unique.

Definition 2.6: (Simulated Trajectory) A simulated trajectory of a control instance $\delta = (\rho, \tau, \mu, x)$ in a HA H is a collection of differentiable maps $\mathcal{X}_\delta = \{\mathcal{X}_\delta^i\}_0^N$, where $\mathcal{X}_\delta^i : [t_i, t_{i+1}] \to X$, such that $\forall i \in [0, N)$,

- Flow: $\forall t \in [t_i, t_{i+1}),\ \dot{\mathcal{X}}_\delta^i(t) = F_{l_i}(\mathcal{X}_\delta^i(t), \mu(t))$;
- Reset: $\mathcal{X}_\delta^{i+1}(t_{i+1}) = R_{(l_i,l_{i+1})}(\mathcal{X}_\delta^i(t_{i+1}), \mu(t_{i+1}))$.

Given a control instance δ, its simulated trajectory $\mathcal{X}_\delta$ in H can be achieved according to the flow functions and reset functions by numerical integration of the ordinary differential equations. However, a simulated trajectory can be infeasible for HA H because neither invariant conditions nor guard conditions are considered in simulated trajectories.

Definition 2.7: (Feasible Trajectory) A simulated trajectory $\mathcal{X}_\delta$ of a control instance $\delta = (\rho, \tau, \mu, x)$ is a feasible trajectory if and only if:

- Invariant: $\forall i \in [0, N],\ \forall t \in [t_i, t_{i+1}),\ (\mathcal{X}_\delta^i(t), \mu(t)) \in Inv_{l_i}$;
- Guard: $\forall i \in [0, N),\ (\mathcal{X}_\delta^i(t_{i+1}), \mu(t_{i+1})) \in G_{(l_i,l_{i+1})}$.

If there is a feasible trajectory $\mathcal{X}_\delta$ of a control instance $\delta = (\rho, \tau, \mu, x)$ in H, we say the path ρ is **feasible** and all locations in ρ are reachable.

Definition 2.8: (Witness Trajectory) A feasible trajectory $\mathcal{X}_\delta$ of a control instance $\delta = (\rho, \tau, \mu, x)$ is a witness trajectory of H if and only if:

- $(l_0, \mathcal{X}_\delta^0(t_0)) \in S_0$;
- $(l_N, \mathcal{X}_\delta^N(t_{N+1})) \in S_f$.

We use $(l_0, \mathcal{X}_\delta^0(t_0))$ and $(l_N, \mathcal{X}_\delta^N(t_{N+1}))$ to denote the initial state and the final state, respectively, in a simulated trajectory $\mathcal{X}_\delta$. A witness trajectory of H represents the behavior of H that starts from an initial state in S_0 and reaches an unsafe state in S_f. We say a control instance that leads to a witness trajectory is a **witness control instance**. We say a path ρ is **falsification-related feasible** in H if H can evolve to an unsafe state through ρ. That is, there exists τ, μ and x so that the simulated trajectory $\mathcal{X}_\delta$ of the control instance $\delta = (\rho, \tau, \mu, x)$ is a witness trajectory in H.

An example of a nondeterministic HA is shown in Fig. 1. Its system continuous states are x and y and external input $\mu = \emptyset$. There are 6 discrete locations (from v0 to v5) in it, with their flow functions and invariant conditions in the form of equations and inequalities respectively. For example, the flow functions in v0 are $\dot{x} = \ln(x + 2)$ and $\dot{y} = 1$; the invariant condition in v0 is $y \leq 5 - \frac{\sin x}{x}$ and the location v5 has no invariant condition. The HA has 7 edges, with their reset functions and guard conditions in the form of equations and inequalities respectively. The initial state is in $S_0 = \{(l, x, y) \mid l = \text{v}_0 \bigwedge -0.01 \leq x \leq 0.01 \bigwedge y == 0\}$, and the target unsafe state is in $S_f = \{(l, x, y) \mid l = \text{v}_3 \bigwedge x \geq y - 1.5\}$. To falsify this automaton, we need to find a control instance δ, under which its simulated trajectory $\mathcal{X}_\delta$ is a witness trajectory. This example is used to illustrate our approach in this work.

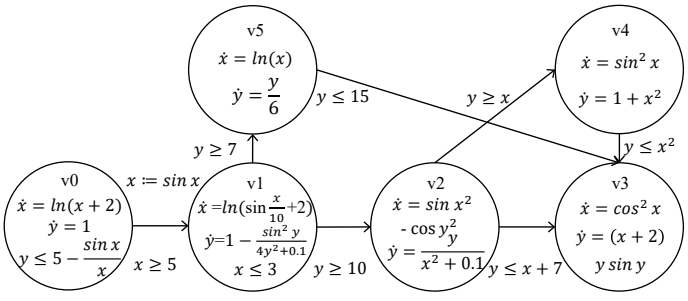

Fig. 1. An Illustrative Hybrid Automaton

B. Classification-Model-Based Derivative-Free Optimization

Derivative-free optimization (DFO) can tackle complex constrained optimization problems. Given a domain X and an objective function $\mathcal{F}$ to evaluate solutions in X, DFO outputs an optimal solution $\arg\min_{x \in X} \mathcal{F}(x)$. No derivative

0278-0070 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Nanjing University. Downloaded on July 18,2021 at 12:55:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3056360, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

information is required in DFO, so the objective function $\mathcal{F}$ can be non-convex, non-smooth, non-differentiable, or even in some way noisy. Many DFO methods are heuristic-based, using distribution or joint distribution as their model with distribution estimation algorithms. Typical DFO methods include genetic algorithms [28], cross-entropy methods [29], Bayesian optimization methods [30], and optimistic optimization methods [31]. However, due to the heuristics in many DFO methods and the variety of optimization problems, most DFO methods either have little theoretical support or suffer from poor scalability.

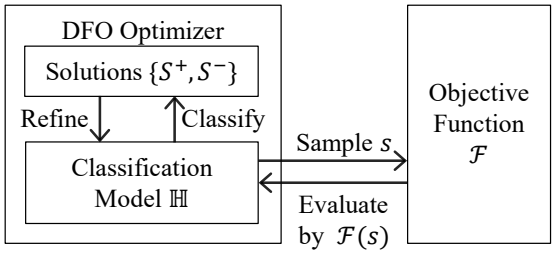

Fig. 2. Classification-Model-Based DFO

Recently, efficient classification-model-based DFO methods with better theoretical support have been proposed. A classification-model-based DFO method with advanced model learning techniques has been empirically verified as having efficacy and high efficiency. It has also been theoretically proven to be able to solve problems with local-Lipschitz continuity in polynomial time. See [26] for analysis of general optimization performance bound, including proofs of convergence rate and query complexity of the objective function $\mathcal{F}$.

Classification-model-based DFO methods learn a classification model $\mathbb{H}$ to discriminate bad solutions from good ones. As depicted in Fig. 2, these classification-model-based DFO methods share a framework of cycles of solution sampling and model refinement. In each iteration, the classification-model-based DFO samples a batch of new solutions s from the domain defined by $\mathbb{H}$. For each sampled solution s, an objective function $\mathcal{F}$ is called to evaluate s. Following this, evaluations are sent back to $\mathbb{H}$ to classify each of the newly sampled solutions as either positive or negative. Finally, these positive and negative solutions refine the classification model $\mathbb{H}$ to achieve a better area of the search space for sampling new solutions in the next iteration. After all iterations are completed, the classification-model-based DFO returns the optimal solution to the caller. The sampling and the model refinement cycle iterates to enhance the accuracy of the classification model $\mathbb{H}$, so as to improve the quality of the solutions generated from the model $\mathbb{H}$.

III. Path-Oriented DFO-Based Falsification

A. Approach Framework

In this section, we present our two-layered, path-oriented and DFO-based falsification approach to falsify safety properties in nonlinear and nondeterministic HA. Since the behavior of a nondeterministic HA is decided by a control instance $\delta = (\rho, \tau, \mu, x)$, the search space of the HA falsification problem is the domain of control instances in the HA.

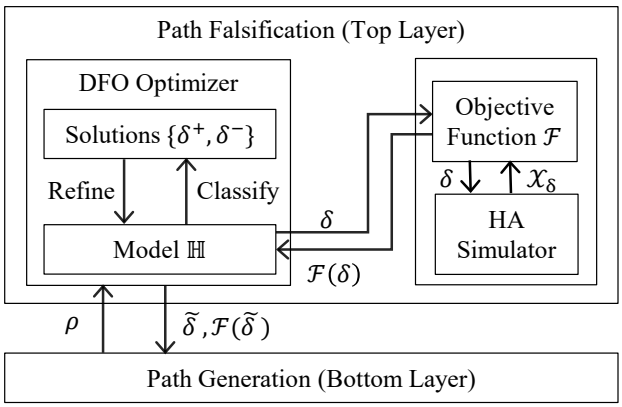

Fig. 3. Path-Oriented DFO-Based Framework

The main framework of our approach is shown in Fig. 3. Our framework is two-layered. We generate candidate paths on the bottom layer. Each time a candidate path ρ is generated, we send it to the top layer to perform path falsification. To generate candidate paths, we first obtain all initial locations and target locations by initial conditions S_0 and unsafe conditions S_f respectively. Then we generate candidate paths from initial locations to target locations. As for the path generation, we can obtain the graph structure of the HA by its discrete locations and transitions, and we then perform a target-oriented DFS for this graph structure. We can also perform the path generation by encoding the graph structure of the HA as a SAT formula [32].

To perform path falsification on the top layer for each candidate path ρ, we search for a control instance δ of the path ρ so that the simulated trajectory $\mathcal{X}_\delta$ is a witness trajectory. First, we encode the falsification-related feasibility of each candidate path ρ into an optimization problem and define the objective function $\mathcal{F}$ for this path ρ. Optimization problem encoding and the definition of paths' objective function are given in Sec. III-B. Then, we perform a classification-model-based DFO algorithm for this optimization problem, sampling control instances along ρ, simulating and evaluating them iteratively until the iteration limit is reached or a witness is found. In each iteration of the DFO, control instances δ are sampled from the model $\mathbb{H}$, evaluated by the objective function $\mathcal{F}(\delta)$ and classified as either positive or negative by the classification model $\mathbb{H}$ according to its evaluation. The classification-model $\mathbb{H}$ is then refined by positive and negative control instance sets $\{\delta^+, \delta^-\}$. The classification-model-based DFO algorithm including detailed model refinement process is given in Sec. III-C.

For example, for the HA in Fig. 1, v_0 is the only initial location in S_0 and v_3 is the only target location in S_f. We generate candidate paths from v_0 to v_3 on the bottom layer. The candidate path $\rho_0 = \mathrm{v}_0 \to \mathrm{v}_1 \to \mathrm{v}_2 \to \mathrm{v}_3$ is firstly generated, and we perform path falsification for ρ_0 on the top layer. If we fail to find a witness control instance along ρ_0 on the top layer, the candidate path $\rho_1 = \mathrm{v}_0 \to \mathrm{v}_1 \to \mathrm{v}_2 \to \mathrm{v}_4 \to \mathrm{v}_3$ would be generated on the bottom layer.

B. From Path Falsification to Control Instance Optimization

In this part, we transform the falsification of a candidate path to the optimization for an optimal control instance along

0278-0070 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Nanjing University. Downloaded on July 18,2021 at 12:55:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3056360, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

this path. Firstly, we encode the falsification-related feasibility problem of a candidate path into a satisfiability problem of a set of constraints. Secondly, we transform the satisfiability problem of constraints into an optimization problem for an optimal control instance.

1) Falsification-Related Feasibility Encoding: In a HA H, given a path ρ and a simulated trajectory $\mathcal{X}_\delta$ along ρ, we encode all constraints that $\mathcal{X}_\delta$ should satisfy if it is a witness trajectory in H. We consider the constraints in the discrete and the continuous level respectively.

Considering the constraints for a witness trajectory $\mathcal{X}_\delta$ in the discrete level. First, the initial state and the final state in $\mathcal{X}_\delta$ should satisfy conditions in S_0 and S_f respectively. Second, the guard condition of each discrete transition along the path ρ should be satisfied when the HA H take this transition. Therefore, constraints in the discrete level are denoted as

$$\begin{aligned}\mathbb{C}_{dis}(\delta) =& S_0(l_0, \mathcal{X}_\delta^0(t_0)) \bigwedge S_f(l_N, \mathcal{X}_\delta^N(t_{N+1})) \\ & \bigwedge_{i\in[0,N),i\in\mathbb{Z}} G_{(l_i,l_{i+1})}(\mathcal{X}_\delta^i(t_{i+1}), \mu(t_{i+1}))\end{aligned}$$

Considering the constraints for a witness trajectory $\mathcal{X}_\delta$ in the continuous level. The invariant condition of each location along the path ρ should be satisfied when the HA H dwells in this location. Therefore, constraints in the continuous level are denoted as

$$\mathbb{C}_{con}(\delta) = \bigwedge_{i\in[0,N],i\in\mathbb{Z}} \bigwedge_{\forall t\in[t_i,t_{i+1})} Inv_{l_i}(\mathcal{X}_\delta^i(t), \mu(t))$$

Combining the constraints in both levels, we denote constraints for a witness trajectory $\mathcal{X}_\delta$ as

$$\mathbb{C}(\delta) = \mathbb{C}_{dis}(\delta) \bigwedge \mathbb{C}_{con}(\delta)$$

For example, for the path $\rho_0 = \mathrm{v}_0 \to \mathrm{v}_1 \to \mathrm{v}_2 \to \mathrm{v}_3$ in the HA in Fig. 1, its falsification-related feasibility is encoded into the satisfiability of constraints $\mathbb{C}(\delta)$, where δ is a control instance along ρ_0 and

$$\begin{aligned}\mathbb{C}(\delta) = -0.01 \le & \mathcal{X}_\delta^0(0)(x) \le 0.01 \bigwedge \mathcal{X}_\delta^0(0)(y) = 0 \\ & \bigwedge \mathcal{X}_\delta^3(t_4)(x) \ge \mathcal{X}_\delta^3(t_4)(y) - 1.5 \\ & \bigwedge \mathcal{X}_\delta^0(t_1)(x) \ge 5 \bigwedge \mathcal{X}_\delta^1(t_2)(y) \ge 10 \\ & \bigwedge \mathcal{X}_\delta^2(t_3)(y) \le \mathcal{X}_\delta^2(t_3)(x) + 7 \\ & \bigwedge_{\forall t\in[t_0,t_1)} \mathcal{X}_\delta^0(t)(y) \le 5 - \frac{sin(\mathcal{X}_\delta^0(t)(x))}{\mathcal{X}_\delta^0(t)(x)} \\ & \bigwedge_{\forall t\in[t_1,t_2)} \mathcal{X}_\delta^1(t)(x) \le 3\end{aligned} \tag{1}$$

A candidate path ρ is falsification-related feasible if and only if constraints $\mathbb{C}(\delta)$ are satisfied by a control instance δ along the path ρ.

2) From Constraint Solving to Optimization: After encoding the candidate path's falsification-related feasibility into the satisfiability of constraints in $\mathbb{C}$, we need to solve this satisfiability problem. However, the constraints in $\mathbb{C}$, such as those shown in Eq.(1), are usually full of nonlinear arithmetics and even formulas with '$\forall$', which cannot be clearly solved by existing SMT solvers. In this work, we transform the satisfiability problem into an optimization problem.

Instead of direct constraint solving, we sample potential solutions of $\mathbb{C}$ (in the form of the control instance δ) to verify whether they can make the whole constraint problem satisfiable. We define the **dissatisfactory degree** of constraints w.r.t the solution δ to measure whether a sampled potential solution can fulfill the constraints, and if not, how bad it is. The dissatisfactory degree of constraints is defined below.

A simple constraint is in the form of 'Expr $\bowtie$ 0', where 'Expr' is an expression about continuous states and $\bowtie \in \{\ge, >, =, \ne\}$. Function '**val**': Expr$\to \mathbb{R}$ maps each expression to its value. The dissatisfactory degree of such a simple constraint $\mathbb{C}$ is defined as:

$$\mathcal{D}(\mathbb{C})^1 = \begin{cases} \textbf{val}(\text{Expr}) \ge 0 ?\ 0 : -\textbf{value}(\text{Expr}) & \text{if } \mathbb{C} \text{ is Expr} \ge 0 \\ \textbf{val}(\text{Expr}) > 0 ?\ 0 : 1 - \textbf{value}(\text{Expr}) & \text{if } \mathbb{C} \text{ is Expr} > 0 \\ \textbf{val}(\text{Expr}) = 0 ?\ 0 : |\textbf{value}(\text{Expr})| & \text{if } \mathbb{C} \text{ is Expr} = 0 \\ \textbf{val}(\text{Expr}) \ne 0 ?\ 0 : 1 & \text{if } \mathbb{C} \text{ is Expr} \ne 0 \end{cases}$$

A complex constraint is in the form of the conjunction or the disjunction of simple or complex constraints. The dissatisfactory degree of such a complex constraint $\mathbb{C}$ is defined as:

$$\mathcal{D}(\mathbb{C}) = \begin{cases} \min(\mathcal{D}(\mathbb{C}_1), \mathcal{D}(\mathbb{C}_2)), & \text{if } \mathbb{C} = \mathbb{C}_1 \bigvee \mathbb{C}_2 \\ \mathcal{D}(\mathbb{C}_1) + \mathcal{D}(\mathbb{C}_2), & \text{if } \mathbb{C} = \mathbb{C}_1 \bigwedge \mathbb{C}_2 \end{cases}$$

For example, the first element in the $\mathbb{C}(\delta)$ in Eq.(1) can be written as a complex constraint $(\mathcal{X}_\delta^0(0)(x) + 0.01 \ge 0) \wedge (\mathcal{X}_\delta^0(0)(x) - 0.01 \le 0)$. Given a control instance δ together with its simulated trajectory $\mathcal{X}_\delta$, we can compute the dissatisfactory degree of this complex constraint.

However, the universal quantification '$\forall$' exists in the continuous-level falsification-related feasibility constraints $\mathbb{C}_{con}$. It is impossible for us to compute and accumulate dissatisfactory degrees of constraints in the invariant conditions at all moments infinitely. So we approximate the dissatisfactory degree of constraints $\mathbb{C}_{con}$ by focusing on the dissatisfactory degrees of constraints in the invariant conditions at regular intervals and critical moments. Thus, the approximation of $\mathbb{C}_{con}$ and $\mathbb{C}$ are defined as $\mathbb{C}'_{con}$ and $\mathbb{C}'$ respectively, where

$$\mathbb{C}'_{con}(\delta) = \bigwedge_{i\in[0,N],i\in\mathbb{Z}} \bigwedge_{\substack{\alpha_j\in[t_i,t_{i+1}),\\ \alpha_j\in(\alpha_{j-1},\alpha_{j-1}+\triangle t],\\ \alpha_j=\alpha_{j-1}+\triangle t||\alpha_j \text{ is critical}^2}} Inv_{l_i}(\mathcal{X}_\delta^i(\alpha_j), \mu(\alpha_j))$$

$$\mathbb{C}'(\delta) = \mathbb{C}_{dis}(\delta) \bigwedge \mathbb{C}'_{con}(\delta)$$

Given a potential solution δ of a constraint $\mathbb{C}(\delta)$, we evaluate it by the dissatisfactory degree of $\mathbb{C}'(\delta)$ as:

$$\mathcal{F}(\delta) = \mathcal{D}(\mathbb{C}'(\delta)) \tag{2}$$

The dissatisfactory degree of constraints is always non-negative and a smaller dissatisfactory degree implies better satisfaction of constraints. The dissatisfactory degree would be 0 if and only if the constraints are satisfied. Thus, the falsification-related feasibility of the path ρ is transformed into the optimization problem of $\arg\min_{\delta \text{ along } \rho} \mathcal{F}(\delta)$.

[1] '?' is a conditional ternary operator here. Given an expression 'A? B:C', if A is true, the entire expression evaluates to B, and otherwise to C.

[2] $\triangle$t is a small and fixed time interval. The time instance α_j is critical if Inv_{l_i} is violated at α_j, for the first time since α_{j-1}. Critical time instances are obtained by the ODE solver used in our work.

0278-0070 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Nanjing University. Downloaded on July 18,2021 at 12:55:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3056360, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

C. Classification-Model-Based DFO for Optimal Control Instance

Algorithm 1 Classification-Model-based DFO for the Optimal Control Instance along a Candidate Path

Input:
X : Domain of Control Instances along the path ρ;
$\mathcal{F}$: Dissatisfactory Degree Function of the path ρ;
N : Number of Iterations;
m : Number of Solutions Sampled in Each Iteration;
n : Number of Mutations When Sampling New Solutions.
Output: $\arg\min_{\delta\in X}\mathcal{F}(\delta),\ \min_{\delta\in X}\mathcal{F}(\delta)$.

1: **function Optimize**(X,$\mathcal{F}, N, m, n$)
2: $S, Y = \{\}$ ▷ Solution set, evaluation set
3: $S^+ = X, S^- = \{\}$ ▷ Positive, negative solutions
4: $\tilde{\delta} = \{\}$ ▷ Optimal solution
5: **for** $n = 1$ to N **do**
6: **for** $i = 1$ to m **do**
7: $\delta^+ =$ **UniformRandom**(S^+)
8: $\mathbb{H} =$ **ModelRefine**(X, δ^+, S^-)
9: $\delta_i =$ **Sample**$(\mathbb{H}, \delta^+, n, |X|)$
10: $y_i = \mathcal{F}(\delta_i)$ ▷ Evaluate solutions
11: **end for**
12: $S = \{\delta_1, ..., \delta_m\}, Y = \{y_1, ..., y_m\}$
13: $\{S^+, S^-\} = \mathbf{C_k}(S, Y)$ ▷ Classify solutions
14: $\tilde{\delta} = \arg\min_{\delta\in S\cup\{\tilde{\delta}\}} \mathcal{F}(\delta)$ ▷ Update optimal solution
15: **end for**
16: **return** $\tilde{\delta}, \mathcal{F}(\tilde{\delta})$ ▷ Optimal solution and evluation
17: **end function**

18: **function ModelRefine**(X, δ^+, S^-)
19: $\mathbb{H} = X$ ▷ Init new model
20: **while** $\exists\delta^- \in S^-$, s.t. $\delta^- \in \mathbb{H}$ **do**
21: $d =$ **UniformRandom**$(1, |X|)$ ▷ Select dimension
22: $S_l^- = \{\delta^- \in S^- \mid \delta^-[d] > \delta^+[d]\}$
23: $S_s^- = \{\delta^- \in S^- \mid \delta^-[d] < \delta^+[d]\}$
24: **if** $|S_l^-| \geq |S_s^-|$ **then** ▷ Shrink upper bound
25: $r =$ **UniformRandom**$(\delta^+[d], \min_{\delta\in S_l^-} s[d])$
26: $\mathbb{H} = \mathbb{H} \cap \mathbb{H}[d] \leq r$
27: **end if**
28: **if** $|S_l^-| \leq |S_s^-|$ **then** ▷ Shrink lower bound
29: $r =$ **UniformRandom**$(\max_{\delta\in S_s^-} s[d], \delta^+[d])$
30: $\mathbb{H} = \mathbb{H} \cap \mathbb{H}[d] \geq r$
31: **end if**
32: **end while**
33: **return** $\mathbb{H}$ ▷ Return learnt model
34: **end function**

35: **function Sample**$(\mathbb{H}, \delta^+, n, D)$
36: $\delta = \delta^+$ ▷ Init new solution
37: **for** $i = 1$ to n **do** ▷ Mutate n times
38: $d =$ **UniformRandom**$(1, D)$ ▷ Select dimension
39: $\delta[d] =$ **UniformRandom**$(\mathbb{H}[d].low, \mathbb{H}[d].up)$
40: **end for**
41: **return** δ ▷ Return new solution
42: **end function**

For the current optimization problem transformed from the falsification-related feasibility of a candidate path ρ, its optimization domain consists of all control instances along ρ. The objective function to evaluate each control instance δ is the dissatisfactory degree function $\mathcal{F}(\delta)$ in Eq.(2). The optimal solution is the control instance $\tilde{\delta}$ with the minimum dissatisfactory degree $\mathcal{F}(\tilde{\delta})$. Since the objective function $\mathcal{F}(\delta)$ is non-convex, non-differential and sometimes even non-continuous, we cannot solve the optimization problem by gradient-based methods. Therefore, we solve this optimization problem in a derivative-free manner and design a new classification-model-based DFO algorithm based on the framework of [26].

As shown in Alg. 1, classification-model-based DFO is performed in the function '**Optimize**'. It employs a hyper-rectangle $\mathbb{H}$ as its classification model. For each solution δ in the optimization domain X, it is viewed as a good one if it is in the hyper-rectangle $\mathbb{H}$, otherwise, viewed as a bad one. The main workflow of '**Optimize**' performs N times of iterations (line 5-15). In each iteration, it first performs m cycles of classification-model refinement by the function '**ModelRefine**' (line 8), solution sampling by the function '**Sample**' (line 9), and solution evaluation by the input objective function $\mathcal{F}$ (line 10). In the same cycle, model refinement (line 8) and solution sampling (line 9) are based on the same randomly sampled positive solution δ^+ (line 7). Then, this batch of m new solutions are divided into positive and negative ones by the function '$\mathbf{C_k}$' (line 13). Solutions with k smallest evaluations are classified into the positive set S^+, and the rest into the negative set S^-. At the end of each iteration, we update the optimal solution $\tilde{\delta}$ by the one with the smallest dissatisfactory degree (line 14).

Model refinement and solution sampling techniques are key to a classification-model-based DFO method. We introduce our model refinement process (line 18-34) and solution sampling process (line 35-42) below in detail.

During the model refinement, we first initialize the model $\mathbb{H}$ by the optimization domain X (line 19). Then, we refine the model $\mathbb{H}$ iteratively until no negative solutions in S^- exist inside (line 20-32). In each iteration, we refine $\mathbb{H}$ in a randomly selected dimension d (line 21). By counting the number of negative solutions whose value is larger than the input positive solution δ^+ in dimension d (line 22) and the number of negative solutions whose value is smaller (line 23), we decide how to shrink the bounds of $\mathbb{H}$ in dimension d (line 24-31). For example, in dimension d, if the majority of the negative solutions are larger than δ^+, (which means S_l^- is the majority), we shrink the upper bound of $\mathbb{H}[d]$ to a random value larger than $\delta^+[d]$ and smaller than the value of any solution in S_l^- in dimension d (line 24-27). As such, in each iteration, we shrink the model $\mathbb{H}$ to exclude the majority of S^- while keeping the positive solution δ^+ still inside. Like most classification model learning techniques, our model refinement technique is also equipped with various high-level randomness (omitted in the pseudocode) to keep reducing the positive search space with a small error-target dependence and a small shrinking rate.

During the solution sampling, we generate new solutions from $\mathbb{H}$ by mutating the input positive solution δ^+ for n times (line 37-40). In each iteration, we select a random dimension d (line 38) and adjust the solution's value in dimension d to a randomly selected value in $\mathbb{H}[d]$ (line 39). Since δ^+ is inside the newly refined model $\mathbb{H}$, the mutated solution is also inside $\mathbb{H}$.

Based on our classification-model-based DFO, we accomplish the falsification of the illustrative HA in Fig. 1. For the candidate path $\rho_0 = \mathrm{v}_0 \to \mathrm{v}_1 \to \mathrm{v}_2 \to \mathrm{v}_3$, control instances in

0278-0070 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Nanjing University. Downloaded on July 18,2021 at 12:55:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3056360, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

the optimization domain $X = \{\rho_0, \tau = \{t_i\}_0^4, \mu = \emptyset, x \in \mathbb{R}^2\}$ are sampled and evaluated iteratively. After all iterations are over, we achieve the optimal control instance along ρ_0 with a minimal dissatisfactory degree 3.814, which is larger than 0. Thus, the candidate path ρ_0 is viewed as not falsification-related feasible and we continue to generate more candidate paths on the bottom layer. Path $\rho_1 = \mathrm{v}_0 \to \mathrm{v}_1 \to \mathrm{v}_2 \to \mathrm{v}_4 \to \mathrm{v}_3$ is then generated. The dissatisfactory degree of the optimal control instance along ρ_1 is 5.591, which is still larger than 0. At last, the third candidate path $\rho_2 = \mathrm{v}_0 \to \mathrm{v}_1 \to \mathrm{v}_5 \to \mathrm{v}_3$ is generated. The dissatisfactory degree of the optimal control instance along ρ_2 is 0, and the optimal control instance is $\tilde{\delta} = (\rho_2, \tilde{\tau}, \emptyset, \tilde{x})$, where $\tilde{\tau} = \{4.86022, 2.23614, 3.40948, 2.77142\}$ and $\tilde{x} = \{-0.00760778, 0\}$. We complete the falsification of the illustrative HA with this witness control instance $\tilde{\delta}$.

IV. Search Space Pruning

In this section, two search space pruning techniques under our path-oriented, DFO-based framework are provided. The first technique introduces a nested DFO structure with better model refinements during the path falsification. It reduces the continuous search space of the jumping time sequences along candidate paths. The second technique reduces the discrete search space by introducing the idea of 'hardly feasible path prefix' and using it to guide backtracking during the path generation.

A. Nested DFO Structure with Better Model Refinements

Given a HA H with its current state (l, x) and the external inputs, we can simulate the evolution of H during a simulation time horizon to figure out when a discrete transition (l, l') can be enabled in H and when H can not continue staying in the current location l. Therefore, when H evolves under a control instance δ in the first i locations and arrives at the $(i+1)$th location l_i at the moment t_i, we can define its smallest and largest possible jumping time to the location l_{i+1}.

Definition 4.1: (Jumping Time Bounds) Given a HA H having evolved to the location l_i under a control instance $\delta = \{\rho = \{l_i\}_0^N, \tau = \{t_i\}_0^{N+1}, \mu, x\}$, the upper jumping time bound to l_{i+1} is t_{i+1}^u, where

- $t_i < t_{i+1}^u$
- $(\mathcal{X}_\delta^i(t_{i+1}^u), \mu(t_{i+1}^u)) \notin Inv_{l_i}$
- $\forall t \in [t_i, t_{i+1}^u),\ (\mathcal{X}_\delta^i(t), \mu(t)) \in Inv_{l_i}$

the lower jumping time bound to l_{i+1} is t_{i+1}^u, where

- $t_i < t_{i+1}^l \le t_{i+1}^u$
- $(\mathcal{X}_\delta^i(t_{i+1}^l), \mu(t_{i+1}^l)) \in G_{(l_i, l_{i+1})}$
- $\forall t \in [t_i, t_{i+1}^l),\ (\mathcal{X}_\delta^i(t), \mu(t)) \notin G_{(l_i, l_{i+1})}$

When simulating a HA under a control instance, the jumping time bounds for all reachable locations can be obtained with a small amount of extra effort. When the HA has evolved to the location l_i, we simulate its continuous dynamics in some simulation time horizon. The upper jumping time bound of the location l_{i+1} is the first moment that the invariant condition of l_i is not satisfied during the simulation. The lower jumping time bound of the location l_{i+1} is the first moment that the guard condition of the discrete transition (l_i, l_{i+1}) is satisfied and the invariant condition of l_i has not been violated. When the upper jumping time bound of l_{i+1} doesn't exist, we can set it to the simulation time horizon. When the lower jumping time bound of l_{i+1} doesn't exist, it suggests that the location l_{i+1} is not reachable under this control instance.

However, under our original single layer DFO structure, it is hard to refine the classification model by the jumping time bounds directly. For one thing, the jumping time bound $[t_{i+1}^l, t_{i+1}^u]$ under a control instance δ is valid only when the HA is given the same initial values, external inputs and dwells in all locations before l_i as δ requires. For another, the space required to store all jumping time bounds under all sampled control instances can be giant. Therefore, to utilize the jumping time bounds efficiently, we propose a nested optimization structure during the path falsification. The workflow under the nested optimization structure is shown in Fig. 4.

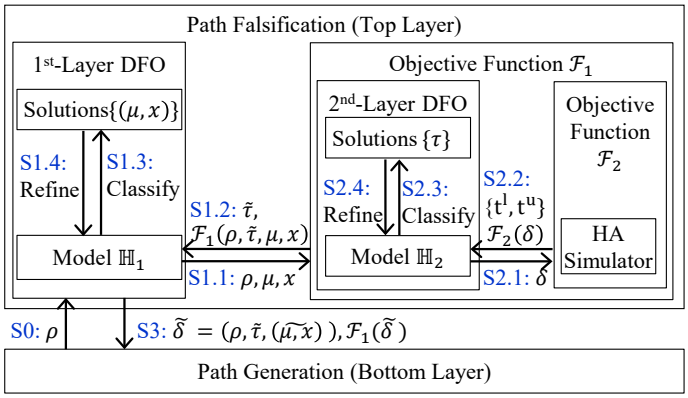

Fig. 4. Workflow under the Nested Optimization Structure

In the first layer of the nested DFO, it samples and evaluates pairs of external inputs and initial values iteratively. It performs S1.1-1.4 in cycles and returns an optimal control instance $\tilde{\delta}$ together with its dissatisfactory degree $\mathcal{F}_1\{\tilde{\delta}\}$ in S3. The objective function $\mathcal{F}_1$ is defined as before in Eq.(2).

S1.1 A pair of external inputs and initial values (μ, x) is sampled from the model $\mathbb{H}_1$ as before in Alg. 1 and is sent with the candidate path ρ to the second layer DFO to be evaluated.

S1.2 Evaluations of a pair of (ρ, μ, x), including an optimal jumping time sequence $\tilde{\tau}$ of it and the dissatisfactory degree $\mathcal{F}_1(\rho, \tilde{\tau}, \mu, x)$, are returned from the second layer DFO to the first layer DFO.

S1.3 Each pair of (μ, x) is classified into a positive or a negative one by its evaluation $\mathcal{F}_1(\rho, \tilde{\tau}, \mu, x)$.

S1.4 The model $\mathbb{H}_1$ is refined by positive and negative solutions sampled before as in Alg. 1.

In the second layer of the nested DFO, for each input (ρ, μ, x), it samples and evaluates time sequences τ along ρ iteratively. It performs S2.1-2.4 in cycles and returns evaluations of (ρ, μ, x) to the first layer in S1.2. The objective function $\mathcal{F}_2$ in the second layer DFO is defined to return the number of unreachable locations, the smaller the better.

S2.1 A jumping time sequence τ of ρ is sampled from the model $\mathbb{H}_2$ as before and then the combined control instance $\delta = (\rho, \tau, \mu, x)$ is sent to be evaluated by $\mathcal{F}_2$.

0278-0070 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Nanjing University. Downloaded on July 18,2021 at 12:55:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3056360, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

S2.2 For each control instance δ, its simulated-trajectory and all jumping time bounds $\{t^l, t^u\}$ are obtained by the HA simulator and the number of unreachable locations is obtained by $\mathcal{F}_2(\delta)$.

S2.3 Each jumping time sequence τ is classified into a positive or a negative one by its evaluation $\mathcal{F}_2(\delta)$.

S2.4 The model $\mathbb{H}_2$ is refined by selecting a random positive solution τ^+ and adding constraints of all jumping time bounds under $\delta^+ = (\rho, \tau^+, \mu, x)$—constraints are $\mathbb{H}_2[i] \in [t_i^l, t_i^u]$ for each reachable location l_i in ρ.

The validity of the jumping time bounds is ensured from two aspects. First, the initial values and external inputs are certain inside the second layer. Second, model $\mathbb{H}_2$ is refined by the jumping time bounds of a random selecting positive solution τ^+, and then new solutions are sampled from $\mathbb{H}_2$ by mutating the same τ^+.

The jumping time bounds refine the classification model $\mathbb{H}_2$ in the second layer DFO more accurately and efficiently, compared to balancing from positive and negative solutions of jumping time sequences sampled before. Actually, we only need to compute jumping time bounds when needed. Also, under this nested DFO structure, our approach is compatible with deterministic HA. For a deterministic HA, given (ρ, μ, x), we can obtain the unique legal jumping time sequence quickly through the model refinements in the second layer DFO by jumping time bounds.

B. Hardly Feasible Path Prefixes Guided Backtracking

The discrete search space in HA falsification can be large. Problems such as "path explosion" limit the performance of "path-oriented" methods. To sufficiently leverage our path-oriented framework and enhance the efficiency of our approach, we introduce the infeasible location sets and infeasible path prefixes. We then prune the discrete search space by an approximation of the shortest infeasible path prefix, which we call "the shortest hardly feasible path prefix". To be specific, we perform backtracking guided by this path prefix, during our path generation process on the bottom layer. When we run out of the simulation limit and find no witness in the current candidate path, instead of generating and checking the next candidate path in the DFS order, we can turn to check a more appropriate one directly.

An infeasible location set of a path ρ is a set of locations in ρ that the constraints of these locations can not be satisfied simultaneously. We give a formal definition of the infeasible location set as follows.

Definition 4.2: (Infeasible Location Set) Given a path $\rho = \{l_i\}_0^N$ in a HA H, the location set λ is an infeasible location set of ρ, if and only if $\lambda \subseteq \rho$ and it is impossible for all constraints (including invariant conditions and guard conditions) of the locations in λ be satisfied when H evolves along ρ.
While some locations in an infeasible location set $\lambda = \{l_i\}_{j_0}^{j_n}$ of the path ρ can be reachable along ρ, the last location l_{j_n} in λ is always not reachable when H evolves along ρ.

Definition 4.3: (Shortest Infeasible Path Prefix) Given a path $\rho = \{l_i\}_0^N$ in a HA H, $\rho' = \{l_i\}_0^M$ is the shortest infeasible path prefix of ρ, if and only if

- $\exists \lambda = \{l_i\}_{j_0}^{j_n}$, λ is an infeasible location set of ρ, $M = j_n$
- $\nexists \lambda = \{l_i\}_{j_0}^{j_n}$, λ is an infeasible location set of ρ, $M > j_n$

The shortest infeasible path prefix of the last candidate path can guide us to perform backtracking during path generation. To find the shortest infeasible path prefix of a path, we need to find all infeasible location sets of it. However, the infeasible location sets are theoretically unobtainable in our approach. Thus, we search for some **hardly feasible location sets** as an alternative.

In our approach, a hardly feasible location set of the candidate path ρ is a set of locations in ρ that the constraints of these locations are never satisfied simultaneously when we simulate the HA along ρ during the path falsification. Usually, hundreds or thousands of simulations are performed for each candidate path during the path falsification on the top layer. Therefore, the hardly feasible location sets can be viewed as an alternative to the infeasible location sets. Then, we could obtain the shortest hardly feasible path prefix of ρ through the hardly feasible location sets of ρ. We only need a little extra effort to record the satisfiability of each location's constraints during the simulations on the top layer. See Alg. 2 for the pseudocode of generating the shortest hardly feasible path prefix.

Algorithm 2 Shortest Hardly Feasible Prefix Generation

Input:
ρ : the Candidate Path;
N: the Number of Simulations.
Output:
$\{l_i\}_0^{\text{index}}$: the Shortest Hardly Feasible Path Prefix of ρ.

```
1: function GenPrefix(ρ, N)
2:     index = |ρ| − 1                    ▷ Init it to the index of last location
3:     for i = 1 to 3 do                  ▷ Check location sets with size 1-3
4:         for each set λ, |λ| = i, λ ⊆ ρ do
5:             sat=FALSE                  ▷ Init satisfiability
6:             for j = 0 to N-1 do        ▷ Check all simulations
7:                 if all the constraints w.r.t. λ is TRUE then
8:                     sat=TRUE, goto line 11
9:                 end if
10:            end for
11:            if ¬ sat then              ▷ Update to the smallest one
12:                index = min(index, λ[i − 1])
13:            end if
14:        end for
15:    end for
16:    return {l_i}_0^index               ▷ The shortest hardly feasible prefix
17: end function
```

Only hardly feasible location sets of size smaller than 4 are considered in our work. In our experience, searching for and utilizing hardly feasible location sets of larger size are more time-consuming and less effective. Therefore, we only search for hardly feasible location sets of size 1-3 (line 3-15). We check each location set by the satisfiability records of each location's constraints in each simulation (line 6-10). For each hardly feasible location set, we check the index of its last location and record the smallest one in the variable 'index' (line 11-13). Finally, we return the shortest hardly feasible path prefix of ρ (line 16).

Take the illustrative HA in Fig. 1 as an example. Firstly, the candidate path $\rho_0 = \text{v}_0 \rightarrow \text{v}_1 \rightarrow \text{v}_2 \rightarrow \text{v}_3$ is generated and we search for its shortest hardly feasible path prefix during

0278-0070 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Nanjing University. Downloaded on July 18,2021 at 12:55:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3056360, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

the path falsification of it. According to Alg. 2, hardly feasible location sets of ρ_0, such as $\lambda_0 = \{v_0, v_1, v_2\}$ and $\lambda_1 = \{v_2, v_3\}$, are found. Among the last locations of all hardly feasible location sets, the location v_2 is the one with the smallest index. Thus, $\rho_0' = v_0 \rightarrow v_1 \rightarrow v_2$ is the shortest hardly feasible path prefix of ρ_0. We could backtrack from the location v_3 to v_1 on the bottom layer, and then, the next generated candidate path is $\rho_2 = v_0 \rightarrow v_1 \rightarrow v_5 \rightarrow v_3$. Finally, the falsification-related feasible path ρ_2 is falsified. Compared to the falsification process described in Sec. III, the generation and falsification of $\rho_1 = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_3$ is not needed any more.

V. Implementation & Experimental Evaluation

A. Implementation Details

We have implemented our approach in a tool called PDF, (***P***ath-Oriented, ***D***erivative-Free ***F***alsification), in C++. Given a HA, PDF would return a witness trajectory if it finds any within the given simulation iterations. PDF supports the general HA with external inputs, nondeterministic semantics, and arbitrary nonlinear dynamics. Details of our implementation are presented below.

System parameters: We implemented our basic approach (Sec. III) and our two pruning techniques (Sec. IV) in PDF. System parameters of PDF include the interpolation method and the number of control points for external inputs, the determinism of HA, the error tolerance, the simulation time horizon, the simulation iteration bound for each path, the total simulation iteration bound, etc. The default error tolerance is set to a small number ($1e^{-8}$) for practical purposes.

Simulation: We have used Sundials' CVODE [33] to handle the numerical integration of the ordinary differential equations associated with the flow functions in the HA. The interpolation of the external inputs is obtained by GNU Scientific Library (GSL) [34], which offers various interpolation types and methods to the users.

B. Benchmarks & Experiments Setup

PDF was tested over a set of widely used benchmarks for HA falsification. The benchmarks used in our experiments included HA with different features such as non-linearity and nondeterminism, different numbers of locations, continuous states and external inputs, and different difficulty levels of unsafe specifications to be falsified. All benchmarks and their features are given in table I.

More specifically, Air1-3 [35] model and capture the essential features of the aircraft flight. They contain a fairly simple deterministic HA with only one discrete location, but with a large number of external inputs. IG1-3 [36] model the dynamics of the interaction between glucose and insulin in the blood system, and IG-4 is modified from IG1-3 to introduce possible errors in the measurement by medical instruments. IG1-3 and IG-4 are benchmarks of deterministic HA without and with external inputs respectively. Nav1-15 [15], [37] model the movement of a vehicle on a $\mathbb{R}^2$ map, and the vehicle's desired velocity is determined by its position on an $n \times m$ grid of the map. In the HA of Nav benchmarks, fixing the cells' size of the grid, the number of locations and the number discrete transitions both increase with the size of the map. Nav1-2, Nav3-4, and Nav-5 are benchmarks of deterministic HA with 16, 25, and 225 locations respectively. Nav6-15 are benchmarks of nondeterministic HA modified from Nav1-5 to introduce blurred borders of the cells on practical maps.

Besides, it's worth mentioning that AFC1 and AFC2 contain a frequently used automatic control system in ARCH-COMP, whose model is reported as "a high complexity model" with "complex constraint input spaces" [38]–[41]. This benchmark was first introduced in [42], consisting of a powertrain engine model and an air-fuel controller with 4 operation modes, 2 external inputs, and 8 continuous states. Representative system requirements were also given in [42]. In our experiment, we falsified the two requirements that can be written in the form of safety property, with the input settings given in [42]. To be specific, we fixed the input throttle to be piecewise constant with 10 uniform segments over [0,61.2) and the input speed to be constant over [900,1100). AFC1a-c falsify the controlled signal overshoot/undershoot requirement presented in Eq.(26) in [42], and AFC2a-c falsify the accumulated error requirement presented in Eq.(29) there. For the requirements in AFC1a-c, their specific limits increase in value, such that their falsification difficulties increase in order. So do AFC2a-c.

Our experiments are organized in the following aspects.

1. We evaluated the performance of our approach and the effectiveness of the two pruning techniques proposed under our path-oriented, DFO-based framework.
2. We evaluated the performance of PDF by comparing it with S-TaLiRo [14] and Breach [13], two state-of-the-art optimization-based falsification tools in the falsification track of ARCH-COMP 20 [41], designed to falsify temporal logic specifications in deterministic HA. Therefore, we compared PDF with them merely in deterministic benchmarks. We converted the unsafe conditions to safety properties expressed by temporal logic formulas for S-TaLiRo and Breach, and applied HyST [43], [44] to translate all models into Simulink models so that Breach can handle. For the AFC benchmark, we used the hybrid automaton version of the AFC system presented in [42]. We also slightly modified its Simulink version for other tools, such that its dynamics are consistent with our hybrid automaton version.
3. We implemented the Simulated Annealing (SA) algorithm in PDF according to S-TaLiRo's setting. Then, we compared the performance of our classification-model-based DFO algorithm with this classical heuristic method.

All experiments were conducted on the same PC (Intel Core i7 2.20GHz, 16GB RAM, UBUNTU 15.5.2 Virtual Machine). The simulation iteration bound is 10000 for deterministic benchmarks and 30000 for nondeterministic cases. The falsification time bound is 1800s for all cases. For all benchmarks except AFC, we ran PDF, S-TaLiRo$_{SA}$, and S-TaLiRo$_{CE}$ 100 times and ran S-TaLiRo$_{SOAR}$ and Breach 20 times. For the AFC benchmark, whose model complexity leads to its time-consuming simulation process, we ran PDF 100 times and other tools 20 times.

0278-0070 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Nanjing University. Downloaded on July 18,2021 at 12:55:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3056360, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

C. Evaluations

1) Evaluate PDF and Its Pruning Techniques: In Table. I, we presented the falsification results of PDF on all 28 benchmarks. The data we observed have been presented as follows:

1. PDF supports the falsification of deterministic and nondeterministic HA with hundreds of locations and nonlinear time-varying dynamics. PDF successfully generated witness trajectories for all 28 benchmarks and achieved success rates no lower than 95% on $23/28$ of them.
2. The efficiency of PDF is satisfactory in that it obtained witness trajectories of all benchmarks in seconds and used more than 18 seconds in only 2 of all 28 cases.
3. Comparing the basic version of PDF with the optimized version, we concluded that the pruning techniques made substantial improvements in PDF:
   - Without pruning, the success rates of PDF were less than 2% in 9 cases. More specifically, the success rates were 0% in 7 cases and 1% in the other 2 cases. Equipped with pruning techniques, all these failed cases were then solved with high success rates (no lower than 89%).
   - While the success rates were improved in all cases, the simulation iteration usage was generally maintained or even reduced in many cases.

2) Compare PDF with Other Tools in Deterministic Cases: Here, we evaluate the performance of PDF by comparing it with existing works. However, we failed to find any publicly available tool that supports the falsification of nondeterministic and nonlinear HA. Therefore, we made a comparison with S-TaLiRo and Breach, two state-of-the-art optimization-based falsification tools focusing on deterministic hybrid systems. S-TaLiRo and Breach support various optimization methods. In the experiment, we ran Breach with Nelder-Meade (NM), its default optimization method. Since no default or dominating optimization method exists in S-TaLiRo, we chose to run S-TaLiRo with Simulated Annealing (SA), Stochastic Optimization with Adaptive Restart (SOAR), and Cross-Entropy (CE). As these tools only handle deterministic hybrid systems, the comparisons were conducted in the deterministic cases only.

The falsification results of PDF, S-TaLiRo, and Breach are shown in Table. II. From the data we observed the following:

1. For success rates:
   - PDF is the only tool that solved all 18 deterministic cases. Meanwhile, S-TaLiRo solved at most $17/18$ by S-TaLiRo$_{SOAR}$, and Breach solved $14/18$ of all cases.
   - PDF achieved or tied for the highest success rate in $17/18$ of all cases. In the remaining one case, Nav-2, the success rate of PDF is up to 97%.
   - Comparing all settings of S-TaLiRo and Breach, Breach achieved or tied for the highest success rate in $10/18$ of all cases. Among the different settings of S-TaLiRo, S-TaLiRo$_{SA}$ won by achieving or tying for the highest success rate in $6/18$ of all cases.
2. For iteration numbers:
   - In the strategy of Breach, it tries the corner samples first to look for a witness trajectory. In the experiment, this strategy worked, and Breach used very few iterations in $8/18$ of all cases. However, when this strategy did not work, Breach had unsatisfactory success rates or iteration numbers in $8/10$ of the rest cases, and therefore showed an unstable performance in our experiment.
   - S-TaLiRo$_{SOAR}$ consumed the fewest iterations in 7 cases. The SOAR optimizer captured the characteristics of the search space and exploited learned information sufficiently. However, the SOAR optimizer brought heavy time usage, which we will discuss later.
   - While the path of deterministic HA is certain under each input sample, for PDF which is designed to handle nondeterministic HA, all paths need to be checked and more simulated iterations are required naturally. However, even though PDF is not specially tuned for deterministic cases at all, the iteration number used by PDF still was the best three in $17/18$ of all cases, when comparing with other 4 settings which are all designed for deterministic HA.
3. For time usages:
   - Since the implementation of all these tools is different, it is not quite fair to compare PDF with S-TaLiRo and Breach in this aspect. PDF was implemented in C++, but Breach is a Matlab/C++ toolbox and S-TaLiRo is a Matlab toolbox. Thus, PDF used notably less time in most cases naturally.
   - Interestingly, while S-TaLiRo$_{SOAR}$ required fewer iterations than S-TaLiRo$_{SA}$ and S-TaLiRo$_{CE}$, it consumed remarkably more time in most cases. For example, S-TaLiRo$_{SOAR}$ consumed 2-20 times fewer iterations but 1.1-76.0 times more amount of time than S-TaLiRo$_{SA}$ in $8/10$ of the cases that they both solved. The SOAR optimizer explored and exploited learned information in a delicate but time-consuming way. Therefore, S-TaLiRo$_{SOAR}$ is especially suitable for the cases when the simulation number of the model is strictly restricted or when the simulation cost is extremely high.

3) Compare Classification-Model-Based DFO with Heuristic Methods: Different from S-TaLiRo and Breach, PDF performs in a path-oriented manner and handles large search space caused by the nondeterminism. Thus, it is inappropriate to compare our DFO method in PDF with the heuristic ones in other tools just by comparing PDF with them. Instead, we implemented a classical heuristic search method into PDF to make a fair comparison. Among all settings of S-TaLiRo and Breach, S-TaLiRo$_{SA}$ has satisfactory average success rate and balanced performance in the aspects of iteration numbers and execution time, as shown in Table. II. Therefore, we implemented the same SA method (as its implementation in S-TaLiRo) in PDF.

The falsification results of all benchmarks by PDF with SA, PDF with and without pruning are shown in Fig. 5.

1. Optimized PDF with DFO $\mathcal{VS}$. PDF with SA
   - The optimized version of PDF with classification-model-based DFO and pruning outperformed PDF with SA substantially in success rate in all cases.

0278-0070 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Nanjing University. Downloaded on July 18,2021 at 12:55:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3056360, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE I
EXPERIMENTAL RESULTS OF PDF WITH AND WITHOUT PRUNING TECHNIQUES IN ALL BENCHMARKS

Benchmark							Success Rate(%)		#Iter[2]		Time(s)[2]	
Name	Description	Det[1]	#Loc[1]	#Var[1]	#Mu[1]	Unsafe Conditions	Basic[3]	Opt[3]	Basic	Opt	Basic	Opt
Air-1	Aircraft [35]	True	1	3	2*10	v1, $x_1 \in [260, \infty), x_2 \in [-10, 10], t \in [3, 3.2]$	**100**	**100**	502	**399**	**0.2**	0.5
Air-2						v1, $(x_1 + x_3) \in [413.5, \infty), t \in [3, 4]$	13	**64**	**2075**	3643	**1.3**	3.6
Air-3						v1, $x_1 \in [256, \infty), x_3 \in [146, \infty), t \in [3, 4]$	20	**32**	6675	**6560**	**4.3**	10.2
IG-1	Insulin Glucose Control (IG) [36]	True	6	4	0	v3, $G \in [-\infty, 5.325], t \in [88, 92]$	0	**100**	-	**490**	-	**0.7**
IG-2						v5, $G \in [-\infty, 4], I \in [-\infty, -13.29], t \in [120, 360]$	0	**100**	-	**565**	-	**2.6**
IG-3						v3, $G \in [-\infty, 5.52], t \in [98, 102]$	1	**100**	4594	**2226**	4.9	**3.4**
IG-4	Modified IG [36]	True	6	4	1*5	v4, $G \in [6, 16], t \in [90, 100]$	0	**100**	-	**296**	-	**0.4**
Nav-1	Navigation [15]	True	16	4	0	v15,$x \in [2.47, 2.53], v_x \in [-\infty, 0.8], v_y \in [-0.5, 0.5], t \in [0, 10]$	0	**100**	-	**1013**	-	**0.5**
Nav-2						v12, $x \in [3.1, 3.9], y \in [2.1, 2.9], t \in [0, 25]$	0	**97**	-	**4486**	-	**3.4**
Nav-3	Navigation [37]	True	25	4	0	v6,$(x + y) \in [2.399, 2.401], t \in [0, 5]$	1	**100**	2186	**150**	0.2	**0.1**
Nav-4						v6, $x \in [0.99, 1], y \in [1.3, 1.6], v_x \in [0, 1], \lvert v_y \rvert \in [0, 0.2], t \in [0, 5]$	0	**100**	-	**897**	-	**0.2**
Nav-5	Navigation [37]	True	225	4	0	v125, $x \in [4, 4.96], t \in [0, 5]$	0	**89**	-	**2864**	-	**2.1**
Nav-6	Modified Navigation [15]	False	16	4	2*5	v11, $x \in [2.5, 3], t \in [0, 25]$	**100**	**100**	1338	**790**	0.6	**0.3**
Nav-7						v12, $t \in [0, 25]$	73	**95**	3428	**1903**	2.0	**1.0**
Nav-8						v8, $y \in [1.6, 2], t \in [0, 25]$	38	**95**	**5383**	16308	**3.9**	17.4
Nav-9						v3, $x \in [2.3, 2.7], t \in [0, 25]$	76	**95**	24267	**20739**	**17.3**	19.2
Nav-10	Modified Navigation [37]	False	25	4	2*5	v3, $x \in [2.3, 2.7], t \in [0, 25]$	**100**	**100**	1078	**63**	0.4	**0.1**
Nav-11						v5, $v_x, v_y \in [-1, 1], t \in [0, 25]$	87	**96**	3315	**2957**	**1.7**	2.1
Nav-12						v10, $x \in [4.2, 5], t \in [0, 25]$	76	**80**	6927	**6629**	**4.3**	5.2
Nav-13	Modified Navigation [37]	False	225	4	2*5	v125, $y - x \in [3, 4], t \in [0, 25]$	90	**99**	9992	**6960**	5.8	**5.3**
Nav-14						v70, $t \in [0, 25]$	49	**99**	**11489**	15938	**7.2**	11.4
Nav-15						v71, $t \in [0, 25]$	21	**50**	**13804**	16524	**10.2**	15.5
AFC-1a	Modified Automotive Powertrain [42]	True	4	8	1*10 +1*1	v2, $\lvert \mu \rvert \in [0.0075, \infty], t \in [11, 50]$	82	**100**	4668	**2**	165.9	**0.1**
AFC-1b						v2, $\lvert \mu \rvert \in [0.0076, \infty], t \in [11, 50]$	81	**100**	4122	**20**	151.6	**1.7**
AFC-1c						v2, $\lvert \mu \rvert \in [0.0077, \infty], t \in [11, 50]$	77	**100**	4680	**31**	165.4	**2.7**
AFC-2a						v2, $\mathbf{x}_{rms} \in [0.30, \infty], t \in [11, 50]$	9	**100**	5221	**99**	314.7	**11.4**
AFC-2b						v2, $\mathbf{x}_{rms} \in [0.31, \infty], t \in [11, 50]$	8	**100**	6223	**146**	385.5	**17.0**
AFC-2c						v2, $\mathbf{x}_{rms} \in [0.32, \infty], t \in [11, 50]$	5	**96**	6331	**1310**	396.3	**159.7**

[1] Det marks the determinism of systems, #Loc is the number of discrete locations, #Var is the number of continuous states, #Mu is the number of external inputs times the number of control points.
[2] #Iter is the average simulation iterations for successfully falsified runs. Time(s) is the average execution time for successfully falsified runs in seconds.
[3] PDF was run with 2 settings: Basic (without pruning technique) and Opt (with both pruning techniques).

TABLE II
EXPERIMENTAL RESULTS OF PDF AND OTHER TOOLS IN DETERMINISTIC BENCHMARKS

Benchmark Name	Success Rate(%)					#Iter[1]					Time(s)[1]				
	PDF	S-TaLiRo			Breach	PDF	S-TaLiRo			Breach	PDF	S-TaLiRo			Breach
		SA	SOAR	CE	NM		SA	SOAR	CE	NM		SA	SOAR	CE	NM
Air-1	**100**	0	**100**	**100**	30	399	-	**236**	512	1528	**0.5**	-	127.5	8.5	757.6
Air-2	**64**	0	30	5	0	3643	-	**286**	2464	-	**3.6**	-	1346.3	51.8	-
Air-3	**32**	0	30	0	0	6560	-	**271**	-	-	**10.2**	-	961.9	-	-
IG-1	**100**	99	**100**	90	**100**	490	962	103	3626	**2**	0.7	4.0	304.0	13.9	**0.3**
IG-2	**100**	**100**	**100**	**100**	0	565	338	**52**	1004	-	**2.6**	53.7	358.8	414.6	-
IG-3	**100**	**100**	**100**	60	**100**	2226	2176	114	4207	**2**	3.4	8.0	331.1	14.2	**0.4**
IG-4	**100**	**100**	**100**	**100**	**100**	296	547	81	647	**3**	**0.4**	8.9	12.3	10.1	1.7
Nav-1	**100**	**100**	15	43	**100**	1013	1091	**207**	599	1870	**0.5**	93.8	399.7	44.1	466.1
Nav-2	97	29	10	0	**100**	4486	4938	1369	-	**2**	3.4	960.2	1096.2	-	**0.6**
Nav-3	**100**	**100**	90	38	**100**	150	1118	232	2403	**13**	**0.1**	111.2	232.3	168.6	4.5
Nav-4	**100**	**100**	85	**100**	85	897	1041	**453**	2210	2284	**0.2**	99.5	500.7	148.3	765.0
Nav-5	**89**	86	0	17	50	2864	2398	-	6927	**327**	**2.1**	590.7	-	1547.6	938.4
AFC-1a	**100**	20	95	55	**100**	**2**	250	159	247	3	**0.1**	1118.2	745.3	995.7	3.6
AFC-1b	**100**	10	75	20	**100**	**20**	208	146	177	401	**1.7**	931.4	706.1	681.9	581.8
AFC-1c	**100**	0	55	0	70	**31**	-	156	-	460	**2.7**	-	783.3	-	1037.4
AFC-2a	**100**	0	80	0	**100**	99	-	157	-	**78**	**11.4**	-	843.2	-	115.9
AFC-2b	**100**	0	60	0	**100**	146	-	139	-	**78**	**17.0**	-	746.6	-	117.9
AFC-2c	**96**	0	40	0	0	1310	-	**152**	-	-	**159.7**	-	948.3	-	-

[1] #Iter is the average simulation iterations for successfully falsified runs. Time(s) is the average execution time for successfully falsified runs in seconds.

- Meanwhile, compared to PDF with SA, both the iterations and the time consumed by the optimized PDF are sharply reduced in deterministic cases and maintained at the same level in nondeterministic cases.

2. Basic PDF with DFO $\mathcal{VS}$. PDF with SA
   - For success rates, our basic version solved 21 benchmarks, and the SA version solved 16 benchmarks. Besides, in the 14 simultaneously solved cases, the success rate of our basic version was on average 3 times higher than the SA version and 1-73 times higher in $11/14$ of them.
   - For iteration numbers, our basic version showed more efficiency than the SA version in 12 out of the 14 simultaneously solved cases. And in all 14 simultaneously solved cases, our basic version consumed 26% fewer iterations than the SA version on average.
   - In terms of time usage, our basic version and the SA version performed at the same level. In the 14 simultaneously solved cases, the time used by these two versions is close. Summing up all of the average time used in these 14 cases, the basic version used 0.2 second less than the SA version.

0278-0070 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Nanjing University. Downloaded on July 18,2021 at 12:55:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3056360, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

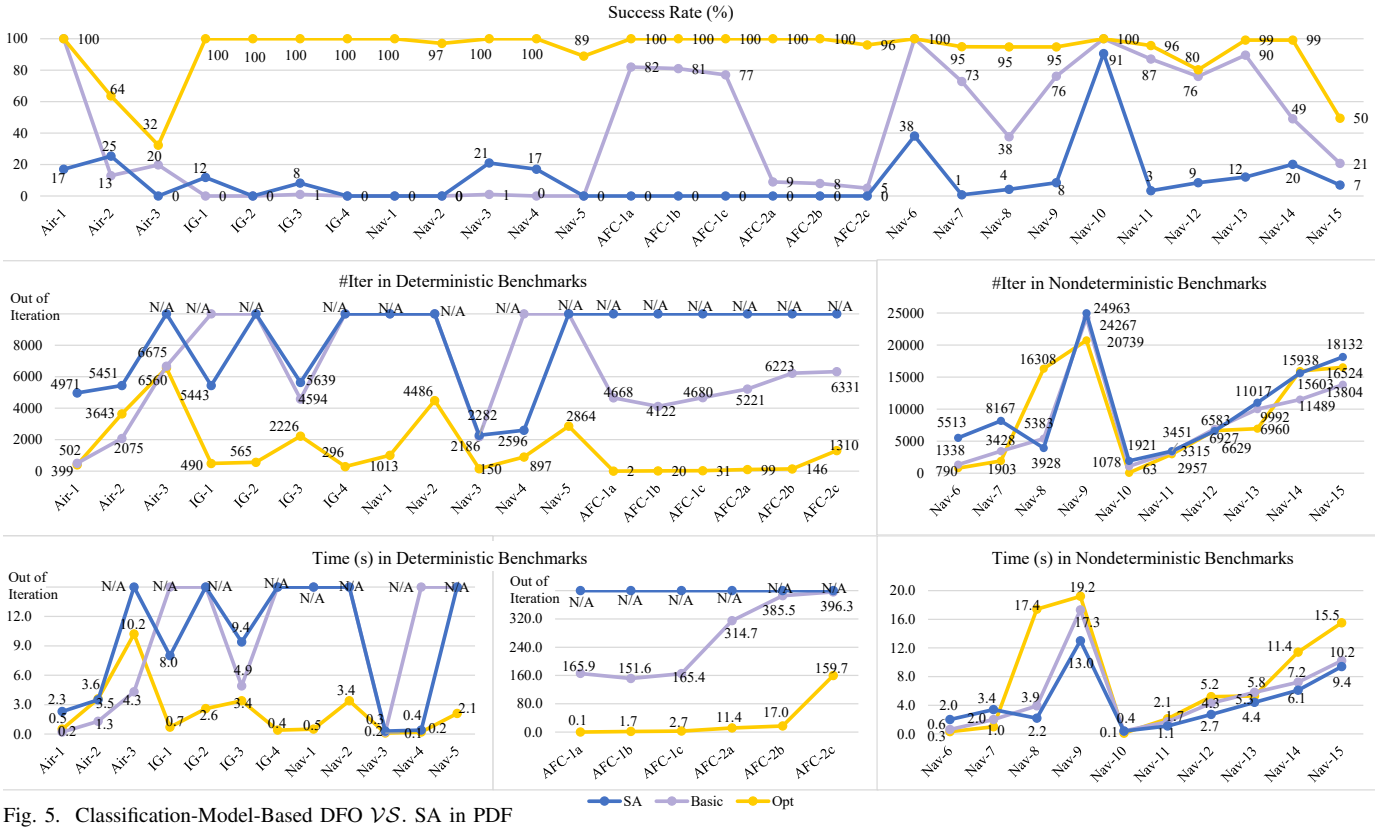

Fig. 5. Classification-Model-Based DFO $\mathcal{VS}$. SA in PDF

VI. Related Work

Verification proves the correctness of systems by technologies such as model checking and theorem proving. Over the last few decades, many verification techniques [3]–[7] have been developed and resulted in tools such as SpaceEx [5], Flow* [6] and C2E2 [7]. However, formal verification techniques have complexity and scalability issues, and can not handle arbitrary nonlinear and nondeterministic hybrid systems well. Since existing verification approaches suffer from various significant limitations and the general problem of the verification of HA is undecidable [8], the falsification of the continuous and hybrid systems becomes an important and practical topic that attracts lots of attention. Falsification of systems aims at finding counterexamples that violating system properties directly.

A. Inner-Approximation-Based Falsification

Inner approximations prove the reachability of the desired states in systems. These methods can be used in the falsification of safety properties in systems. Given safety properties, a system is unsafe if its inner approximations of reachable states sets intersect its unsafe state sets. Inner approximations for nonlinear systems have been computed by modal intervals together with some set-based methods [45]. They have also been computed by the general polytopes [46], reducing the conservativeness induced by the interval representations. Besides, study [47] and its extension [48] compute inner approximations for nonlinear systems via Taylor models, while study [49] computes inner approximations by optimal control theory through Pontryagin's principle, providing an automated tool-chain named UTOPIC. However, UTOPIC only handles continuous systems.

B. Simulation-Based Falsification

Apart from inner-approximation-based approaches, most existing falsification works are simulation-based, requiring a numerical system simulator and observing the target systems by simulated trajectories. Inevitably, the simulator might introduce discrepancies when solving ODEs numerically. As such, a simulated witness trajectory is an approximated system behavior and can be invalid due to simulation discrepancies. However, the simulation-based falsification results are still helpful in industry, where the scalability and the complexity of systems keep increasing, and the white-box models are not always available. See [50] for an overview of the simulation-based approaches.

Generally, the simulation-based falsification is conducted in either the motion-planning-based or the optimization-based directions. Recently, there are also emerging works that attack such a problem by various methods such as machine learning. While many simulation-based approaches aim to falsify MTL [51] or STL [52] properties, our work falsifies safety properties. We can falsify STL properties defined as $G\varphi$ or $G_{[a,b]}\varphi$, where φ is the system's safety property. Supporting the falsification of the general STL properties will be in our future work.

Motion-Planning-Based Falsification Motion-planning-based falsification steers algorithms such as rapidly growing random tree (RRT) [21] to explore the search space. The RRT can grow both forward [9] or backward [10] in time

0278-0070 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Nanjing University. Downloaded on July 18,2021 at 12:55:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3056360, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

and can be guided by different metrics. A coverage metrics called star discrepancy is utilized in [11], and a robustness metric is utilized in [12] to guide the growth of RRT. The motion-planing-based falsification approach does not require the parameterization of external inputs. External inputs are allowed to change along the systems' current trajectory while growing RRT.

Optimization-Based Falsification S-TaLiRo [14], Breach [13], and FalStar [17] are mature Matlab toolboxes using various optimizers to search for counterexamples of temporal logic specifications. These tools are designed for deterministic systems and support Simulink/Stateflow models. As PDF, these tools also parameterize the space of external inputs by control points.

Breach uses Nelder-Mead simplex-based methods [53] with multiple restarts as its default optimizer. S-TaLiRo supports various optimization algorithms, such as Simulated Annealing [15], Ant-Colony [16], and Cross-Entropy [24]. Recently, the SOAR optimizer [54], which conducts global search by a global Gaussian process (GP) model and trust-region-based local search by multiple local GP models, is also implemented in S-TaLiRo. Besides SOAR, multiple other optimization algorithms are also developed in recent falsification works by tightly combining global and local search, balancing exploration and exploitation. FalStar combines Monte Carlo tree searching and local hill-climbing optimization, and study [55] combines Simulated Annealing with local search using gradient descent directions.

Our falsification approach is also an optimization-based one. Unlike the aforementioned tools, which are mostly restricted to models with deterministic semantics, our work also supports nondeterministic models. Compared with existing approaches, we are solving optimization problems with larger optimization domains and different objective functions (associated with the different constraint encoding). Also, instead of the various heuristic searching algorithms used in these tools, we design and use an efficient classification-model-based DFO algorithm.

Gradient-Based Falsification and Learning-Based Falsification Instead of heuristic search, gradient-based solving has been applied to the falsification of HA in recent studies. An approach called multiple-shooting is proposed in [18]. Trajectory segments with gaps starting from the initial states are introduced. By narrowing the gaps between segments iteratively with system gradient, approximate trajectories can be achieved from promising segmented trajectories. Recently, the multiple-shooting-based approach is also combined with symbolic reachability analysis in [19]. By adding reachability constraints, it improves the efficiency of producing concrete counterexamples. While the class of models that can be handled is limited as these works rely on gradient-based solving, our work supports the analysis of arbitrary nonlinear systems. Our future research will further explore the idea of integrating path falsification into the CEGAR loop of verification.

Unlike the aforementioned related works, 'falsify' [20] learns the behavior of systems and solves falsification problems by reinforcement learning. Future research can investigate how to combine reinforcement learning with our classification-model-based learning in the falsification of CPS.

VII. Conclusion

In this work, we proposed a path-oriented, derivative-free approach for the safety falsification of a large scope of CPS that could be modeled by general hybrid automata, allowing for nonlinear and nondeterministic dynamics as well as external inputs. In our two-layered, path-oriented, derivative-free approach, the enumeration of the candidate paths was designed to handle the large search space of the nondeterministic systems; the derivative-free feature of our optimization algorithm was designed to handle the nonlinearity of the systems' dynamics. Moreover, two novel and light-weight techniques were proposed to prune the search space of the falsification problem and to improve the efficiency of our approach. Firstly, we proposed a nested optimization structure with better model-refinements to prune the continuous search space of the candidate paths' jumping time sequences. Secondly, we leveraged hardly feasible path prefixes to prune the discrete search space by backtracking during the candidate path generation.

We implemented our approach in a prototype tool called PDF and applied it to a set of benchmarks, including both the well-known deterministic benchmarks and the nondeterministic benchmarks. Our experiments showed that our path-oriented, DFO-based approach, together with our two pruning techniques, supported the safety falsification of complex CPS with high success rates and satisfactory efficiency.

References

[1] E. A. Lee and S. A. Seshia, *Introduction to embedded systems: A cyber-physical systems approach*. Mit Press, 2016.
[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger *et al.*, "The algorithmic analysis of hybrid systems," *Theor. Comput. Sci.*, vol. 138, no. 1, pp. 3–34, 1995.
[3] E. Asarin, T. Dang, and O. Maler, "The d/dt tool for verification of hybrid systems," in *CAV*, 2002, pp. 365–370.
[4] L. Bu, Y. Li, L. Wang, and X. Li, "Bach: Bounded reachability checker for linear hybrid automata," in *FMCAD*, 2008, pp. 1–4.
[5] G. Frehse, C. L. Guernic, and A. Donzé, "Spaceex: Scalable verification of hybrid systems," in *CAV*, vol. 6806, 2011, pp. 379–395.
[6] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *CAV*, vol. 8044, 2013, pp. 258–263.
[7] P. S. Duggirala, S. Mitra, M. Viswanathan, and M. Potok, "C2E2: A verification tool for stateflow models," in *TACAS*, vol. 9035, 2015, pp. 68–82.
[8] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" *J. Comput. Syst. Sci.*, vol. 57, no. 1, pp. 94–124, 1998.
[9] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Hybrid systems: from verification to falsification by combining motion planning and discrete search," *Formal Methods Syst. Des.*, vol. 34, no. 2, pp. 157–182, 2009.
[10] S. Proch and P. Mishra, "Directed test generation for hybrid systems," in *ISQED*, 2014, pp. 156–162.
[11] T. Dang and T. Nahhal, "Coverage-guided test generation for continuous and hybrid systems," *Formal Methods Syst. Des.*, vol. 34, no. 2, pp. 183–213, 2009.
[12] T. Dreossi, T. Dang, A. Donzé, J. Kapinski *et al.*, "Efficient guiding strategies for testing of temporal properties of hybrid systems," in *NFM*, 2015, pp. 127–142.
[13] A. Donzé, "Breach, A toolbox for verification and parameter synthesis of hybrid systems," in *CAV*, vol. 6174, 2010, pp. 167–170.
[14] Y. Annpureddy, C. Liu, G. E. Fainekos, and S. Sankaranarayanan, "S-taliro: A tool for temporal logic falsification for hybrid systems," in *TACAS*, vol. 6605, 2011, pp. 254–257.
[15] T. Nghiem, S. Sankaranarayanan, G. E. Fainekos, F. Ivancic *et al.*, "Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems," in *HSCC*, 2010, pp. 211–220.

0278-0070 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Nanjing University. Downloaded on July 18,2021 at 12:55:30 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3056360, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

[16] Y. S. R. Annapureddy and G. E. Fainekos, "Ant colonies for temporal logic falsification of hybrid systems," in *IECON*, 2010, pp. 91–96.
[17] Z. Zhang, G. Ernst, S. Sedwards, P. Arcaini *et al.*, "Two-layered falsification of hybrid systems guided by monte carlo tree search," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2894–2905, 2018.
[18] A. Zutshi, S. Sankaranarayanan, J. V. Deshmukh, and J. Kapinski, "A trajectory splicing approach to concretizing counterexamples for hybrid systems," in *CDC*, 2013, pp. 3918–3925.
[19] S. Bogomolov, G. Frehse, A. Gurung, D. Li *et al.*, "Falsification of hybrid systems using symbolic reachability and trajectory splicing," in *HSCC*, 2019, pp. 1–10.
[20] T. Akazaki, S. Liu, Y. Yamagata, Y. Duan *et al.*, "Falsification of cyber-physical systems using deep reinforcement learning," in *FM*, 2018, pp. 456–465.
[21] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," *Algorithmic and computational robotics: new directions*, no. 5, pp. 293–308, 2001.
[22] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theor. Comput. Sci.*, vol. 410, no. 42, pp. 4262–4291, 2009.
[23] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *FORMATS*, vol. 6246, 2010, pp. 92–106.
[24] S. Sankaranarayanan and G. E. Fainekos, "Falsification of temporal properties of hybrid systems using the cross-entropy method," in *HSCC*, 2012, pp. 125–134.
[25] J. Lygeros, K. H. Johansson, and S. N. Simic, "Dynamical properties of hybrid automata," *IEEE Trans. Automat. Contr.*, vol. 48, no. 1, pp. 2–17, 2003.
[26] Y. Yu, H. Qian, and Y. Hu, "Derivative-free optimization via classification," in *AAAI*, 2016, pp. 2286–2292.
[27] Y. Hu, H. Qian, and Y. Yu, "Sequential classification-based optimization for direct policy search," in *AAAI*, 2017, pp. 2029–2035.
[28] D. E. Goldberg, *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley, 1989.
[29] P. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals OR*, vol. 134, no. 1, pp. 19–67, 2005.
[30] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv preprint arXiv:1012.2599*, 2010.
[31] R. Munos, "From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning," *Found. Trends Mach. Learn.*, vol. 7, no. 1, pp. 1–129, 2014.
[32] D. Xie, L. Bu, J. Zhao, and X. Li, "SAT-LP-IIS joint-directed path-oriented bounded reachability analysis of linear hybrid automata," *Formal Methods Syst. Des.*, vol. 45, no. 1, pp. 42–62, 2014.
[33] S. D. Cohen, A. C. Hindmarsh, and P. F. Dubois, "Cvode, a stiff/nonstiff ode solver in c," *Computers in physics*, vol. 10, no. 2, pp. 138–143, 1996.
[34] M. Galassi, J. Davies, J. Theiler, B. Gough *et al.*, "Gnu scientific library," 1996.
[35] J. Lygeros, "On reachability and minimum cost optimal control," *Autom.*, vol. 40, no. 6, pp. 917–927, 2004.
[36] M. E. Fisher, "A semiclosed-loop algorithm for the control of blood glucose levels in diabetics," *IEEE Trans. Biomed. Eng.*, vol. 38, no. 1, pp. 57–61, 1991.
[37] A. Fehnker and F. Ivancic, "Benchmarks for hybrid systems verification," in *HSCC*, 2004, pp. 326–341.
[38] A. Dokhanchi, S. Yaghoubi, B. Hoxha, and G. E. Fainekos, "ARCH-COMP17 category report: Preliminary results on the falsification benchmarks," in *ARCH@ CPSWeek*, 2017, pp. 170–174.
[39] A. Dokhanchi, S. Yaghoubi, B. Hoxha, G. Fainekos *et al.*, "ARCH-COMP18 category report: Results on the falsification benchmarks," in *ARCH@ ADHS*, 2018, pp. 104–109.
[40] G. Ernst, P. Arcaini, A. Donze, G. Fainekos *et al.*, "ARCH-COMP 2019 category report: Falsification," in *ARCH@ CPSIoTWeek*, vol. 61, 2019, pp. 129–140.
[41] G. Ernst, P. Arcaini, I. Bennani, A. Donze *et al.*, "ARCH-COMP 2020 category report: Falsification," *EPiC Series in Computing*, vol. 74, pp. 140–152, 2020.
[42] X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda *et al.*, "Powertrain control verification benchmark," in *HSCC*, 2014, pp. 253–262.
[43] S. Bak, S. Bogomolov, and T. T. Johnson, "Hyst: a source transformation and translation tool for hybrid automaton models," in *HSCC*, 2015, pp. 128–133.
[44] S. Bak, O. A. Beg, S. Bogomolov, T. T. Johnson *et al.*, "Hybrid automata: from verification to implementation," *Int. J. Softw. Tools Technol. Transf.*, vol. 21, no. 1, pp. 87–104, 2019.
[45] E. Goubault, O. Mullier, S. Putot, and M. Kieffer, "Inner approximated reachability analysis," in *HSCC*, 2014, pp. 163–172.
[46] B. Xue, Z. She, and A. Easwaran, "Under-approximating backward reachable sets by polytopes," in *CAV*, 2016, pp. 457–476.
[47] E. Goubault and S. Putot, "Forward inner-approximated reachability of non-linear continuous systems," in *HSCC*, 2017, pp. 1–10.
[48] ——, "Inner and outer reachability for the verification of control systems," in *HSCC*, 2019, pp. 11–22.
[49] J. Doncel, N. Gast, M. Tribastone, M. Tschaikowski *et al.*, "Utopic: Under-approximation through optimal control," in *QEST*, 2019, pp. 277–291.
[50] J. Kapinski, J. V. Deshmukh, X. Jin, H. Ito *et al.*, "Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques," *IEEE Control Syst. Mag.*, vol. 36, no. 6, pp. 45–64, 2016.
[51] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real Time Syst.*, vol. 2, no. 4, pp. 255–299, 1990.
[52] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *FORMATS/FTRTFT*. Springer, 2004, pp. 152–166.
[53] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Comput. J.*, vol. 7, no. 4, pp. 308–313, 1965.
[54] L. Mathesen, S. Yaghoubi, G. Pedrielli, and G. Fainekos, "Falsification of cyber-physical systems with robustness uncertainty quantification through stochastic optimization with adaptive restart," in *CASE*, 2019, pp. 991–997.
[55] S. Yaghoubi and G. Fainekos, "Falsification of temporal logic requirements using gradient based local search in space and time," *IFAC-PapersOnLine*, vol. 51, no. 16, pp. 103–108, 2018.

Jiawan Wang received the BSc degree from Nanjing University in 2018. She is currently pursuing the Ph.D. degree from Nanjing University. Her research interest mainly focuses on the verification of cyber-physical systems.

Lei Bu is a professor in the Department of Computer Science and Technology and State Key Laboratory of Novel Software Technology at Nanjing University. He received his B.S. and PH.D. degree in Computer Science from Nanjing University in 2004 and 2010 respectively. His main research interests include formal method, model checking, especially verification of hybrid system and cyber-physical system. He has published more than 50 research papers in major peer-reviewed international journals and conference proceedings. He is a member of the IEEE and the ACM.

Shaopeng Xing received the BSc degree from Nanjing University in 2018. He was admitted to study for a Msc degree in Nanjing University in the same year. His research interests mainly include verification and optimal control of cyber-physical systems.

Xuandong Li received his MS and PhD degrees from Nanjing University, China, in 1991 and 1994, respectively. He is a professor at the Computer Science and Technology Department of Nanjing University. His research interests include formal support for design and analysis of reactive, disturbed, real-time, hybrid, and cyber-physical systems; software testing and verification.

0278-0070 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Nanjing University. Downloaded on July 18,2021 at 12:55:30 UTC from IEEE Xplore. Restrictions apply.