Scenario-Based Flexible Modeling and Scalable Falsification for Reconfigurable CPSs

Jiawan Wang, Wenxia Liu, Muzimiao Zhang, Jiaqi Wei, Yuhui Shi, Lei Bu(✉), and Xuandong Li

State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing, China
`wangjw@smail.nju.edu.cn`, `bulei@nju.edu.cn`

Abstract. Cyber-physical systems (CPSs) are used in many safety-critical areas, making it crucial to ensure their safety. However, with CPSs increasingly dynamically deployed and reconfigured during runtime, their safety analysis becomes challenging. For one thing, reconfigurable CPSs usually consist of multiple agents dynamically connected during runtime. Their highly dynamic system topologies are too intricate for traditional modeling languages, which, in turn, hinders formal analysis. For another, due to the growing size and uncertainty of reconfigurable CPSs, their system models can be huge and even unavailable at design time. This calls for runtime analysis approaches with better scalability and efficiency. To address these challenges, we propose a scenario-based hierarchical modeling language for reconfigurable CPS. It provides template models for agent inherent features, together with an instantiation mechanism to activate single agent's runtime behavior, communication configurations for multiple agents' connected behaviors, and scenario task configurations for their dynamic topologies. We also present a path-oriented falsification approach to falsify system requirements. It employs classification-model-based optimization to explore search space effectively and cut unnecessary system simulations and robustness calculations for efficiency. Our modeling and falsification are implemented in a tool called `SNIFF`. Experiments have shown that it can largely reduce modeling time and improve modeling accuracy, and perform scalable CPS falsification with high success rates in seconds.

1 Introduction

A cyber-physical system (CPS) consists of multiple computing devices that communicate with each other and interact with the physical world in a feedback loop.

The authors are supported in part by the Leading-edge Technology Program of Jiangsu Natural Science Foundation (No. BK20202001), the National Natural Science Foundation of China (No. 62232008, 62172200), the Fundamental Research Funds for the Central Universities (No. 2023300180), and the Program A for Outstanding PhD Candidates of Nanjing University.

A. Gurfinkel and V. Ganesh (Eds.): CAV 2024, LNCS 14683, pp. 329–355, 2024.
https://doi.org/10.1007/978-3-031-65633-0_15

It is vital to ensure that CPSs work correctly as intended, especially in safety-critical applications like aircrafts, automobiles, and medical devices [4,41,42].

Model checking is a powerful, computer-assisted technique to analyze CPS correctness [11,19]. It involves formalizing system behavior in mathematical models, describing system requirements in logic specifications, and analyzing whether specifications are satisfied in models through complete verification algorithms or incomplete falsification algorithms. Verification algorithms prove system correctness automatically but are typically limited to CPS with few dimensions and simple dynamics [16,18,30,35,36,54,55]. Falsification algorithms, on the other hand, satisfy weaker forms of completeness but can handle complex system dynamics and disprove system correctness via counterexamples [3,8,27,33,53,63,66]. As a result, they are of practical interest and are the main focus of model checking in industrial applications [19,20,53].

However, with real-world CPSs increasingly deployed in uncertain environments and dynamically reconfigured during runtime [1,2,23,32,37,59], their formal modeling and falsification face great challenges.

Challenge 1: CPS modeling is complicated due to changing agent behaviors and dynamic system topologies. For one thing, in a reconfigurable CPS, each agent's role, function, and behavior may vary across scenarios to meet changing requirements. Therefore, it needs to be frequently modeled, especially when there are many different scenarios. For another, agents are usually dynamically wired due to changing demands or limitations. Some agents may even be part of the system in one scenario but leave in another. Traditional modeling languages, however, lack flexible mechanisms to formalize dynamic topology shifts.

Challenge 2: CPS falsification requires better scalability and efficiency due to the growing size and complexity of reconfigurable CPSs. Models of reconfigurable CPSs become huge and may even unavailable at design time. This calls for more scalable falsification approaches with improved runtime performance.

For challenge 1, we designed a hierarchical scenario-based modeling language for reconfigurable CPSs. To simplify agent modeling, we introduced a template model to formalize agent inherent features, including potential dynamics and communication capabilities. Based on template models, an instantiation mechanism is provided so that users can model and update agent runtime behaviors easily. Further, to capture the dynamically wired behaviors of multiple agents, we introduced communication configurations and scenario task configurations for instantiated agent models. Users can formalize agents' intricately connected behaviors through brief communication configurations, and abstract their dynamic topologies flexibly through intuitive scenario task configurations.

For challenge 2, we developed a path-oriented falsification approach for reconfigurable CPSs, searching for counterexample system behavior along generated paths. For the vast behavior space along each path, we designed a two-layered, classification-model-based optimization method to explore it effectively. Besides, since system simulation and robustness calculation are most time-consuming during falsification, we introduced concepts of time-context and lifespan for temporal specifications, to cut unnecessary system simulations and robustness calculations that do not affect the satisfaction of specifications.

The main contributions of this work are as follows:

- We designed a scenario-based hierarchical modeling language to formalize reconfigurable CPSs. We also extended a topology-aware temporal logic to specify their system requirements. (See Sect.3)
- We developed a path-oriented approach to falsify temporal logic specifications for CPSs. It employs classification-model-based optimization and lifespan-guided robustness calculation to improve efficiency. (See Sect.4)
- We implemented a tool called `SNIFF`, which supports scenario-based graphical modeling and falsification for reconfigurable CPSs. Studies show that it can greatly reduce modeling time and improve modeling accuracy. Experiments demonstrated its scalability and efficiency in falsification. (See Sect.5)

2 Background

2.1 Preliminaries

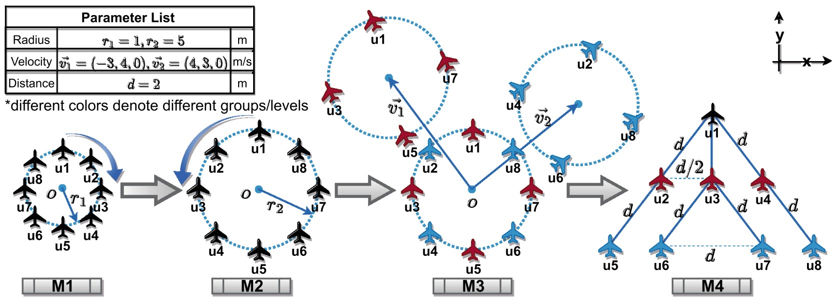
Hybrid Automaton (HA) is a popular formal language for modeling tightly coupled discrete and continuous CPS behaviors. Dense-time formalisms like Signal Temporal Logic (STL) serve as the basis for specifying CPS requirements.

Definition 1 *(***Hybrid Automaton** *[5]).* *An HA is a tuple $H = (X, U, Q, F, Inv, E, G, R)$, where X are continuous variables; U are external inputs*[1]*; Q are discrete operating modes; $F = \{F_q | \ q \in Q\}$ are flow functions, defining the flow of variables in mode q by differential equation $\dot{X} = F_q(X, U)$; $Inv = \{Inv_q | \ q \in Q\}$ are invariant conditions, constraining conditions of mode q by constraints $Inv_q(X, U)$; $E \subseteq Q \times Q$ are discrete transitions, denoting jumps of modes; $G = \{G_e | \ e \in E\}$ are guard conditions on transitions; $R = \{R_e | \ e \in E\}$ are reset functions on transitions, resetting variables by $X := R_e(X, U)$.*

Semantics: The state space of H is $S = Q \times X$. Its state, denoted by the pair $s = (q, x) \in S$, evolves in the following two ways. It can stay in the current mode q and evolve continuously based on the flow function F_q, within the space constrained by the invariant condition Inv_q. It can also jump from the mode q to q' discretely and instantaneously through transition $e = (q, q') \in E$ when its guard condition G_e is met, and be reset by the function R_e.

Note that under nondeterministic semantics, HA can choose arbitrarily between staying in the current mode and jumping to another mode, as long as conditions of current mode and target transition are both satisfied. It can also choose among multiple transitions when their guard conditions are all met [44].

[1] External inputs are often assumed to be time-varying, smooth, and parameterized.



Parameter List		
Radius	$r_1 = 1, r_2 = 5$	m
Velocity	$\vec{v_1} = (-3, 4, 0), \vec{v_2} = (4, 3, 0)$	m/s
Distance	$d = 2$	m

*different colors denote different groups/levels

Fig. 1. The Formation Target of the Multi-UAV System in Each Flight Mission

Definition 2 ***(Signal Temporal Logic** [46,47]). Let $\mathcal{X}$ denotes variables defined over domain $\overline{\mathbb{R}}^n$, where $\overline{\mathbb{R}}$=$\mathbb{R} \cup \{\top, \bot\}$ is the totally ordered set of real numbers with the smallest and greatest boolean elements $\bot$ and $\top$, $\top = -\bot$, $-\top = \bot$. Let $w : \mathbb{T} \to \overline{\mathbb{R}}^n$ denotes a multi-dimensional signal of $\mathcal{X}$, with $\mathbb{T} = [0, t) \subseteq \mathbb{R}$. The syntax of an STL formula φ interpreted over $\mathcal{X}$ is defined as:*

$$\varphi := true \mid \theta(\mathcal{X}) \geq d \mid \neg\varphi_1 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathbf{U}_\mathbf{I} \varphi_2.$$

Here, $\theta : \overline{\mathbb{R}}^n \to \mathbb{R}$ denotes a function that maps $\mathcal{X}$'s valuations into a real, and d is a constant. $\mathbf{U}$ denotes the until operator and $\mathbf{I} \subseteq \mathbb{R}^+$ denotes a timing interval, which can be omitted when it is $[0, \infty)$. In a formula φ, other standard operators can be derived as follows: $false \equiv \neg true$, $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $\varphi_1 \to \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$, $\Diamond_I \varphi \equiv true \mathbf{U}_\mathbf{I} \varphi$, and $\Box_I \varphi \equiv \neg\Diamond_I(\neg\varphi)$.

Semantics: For a given signal, an STL formula φ has both boolean and quantitative semantics [29,46]. Its boolean semantics indicates φ's validity on this signal at some given time t via a binary satisfaction relation, while its quantitative semantics indicates how well this signal satisfies φ at time t via a real-valued robustness degree. Due to space limitations, we only provide quantitative semantics here. Given an STL formula φ, a signal w of variables $\mathcal{X}$, and time t, the quantitative semantics of its robustness degree $\rho(\varphi, w, t)$ is defined by induction:

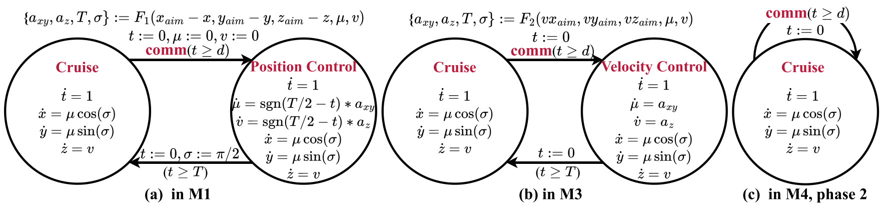
$$\begin{aligned}
\rho(true, w, t) &= \top \\
\rho(\theta(\mathcal{X}) \geq d, w, t) &= \theta(w(t)) - d \\
\rho(\neg\varphi_1, w, t) &= -\rho(\varphi_1, w, t) \\
\rho(\varphi_1 \vee \varphi_2, w, t) &= \max\{\rho(\varphi_1, w, t), \rho(\varphi_2, w, t)\} \\
\rho(\varphi_1 \mathbf{U}_\mathbf{I} \varphi_2, w, t) &= \sup_{t' \in t \oplus \mathbf{I}} \min\{\rho(\varphi_2, w, t'), \inf_{t'' \in [t, t')} \rho(\varphi_1, w, t'')\}
\end{aligned} \tag{1}$$

Here, $\oplus$ denotes the Minkowski sum, $t \oplus \mathbf{I} = \{t + i | i \in \mathbf{I}\}$. This quantitative semantics is used in most STL falsification. The degree is positive only if the signal satisfies the formula, and the higher the better.

2.2 Motivating Example: A Multi-UAV System

Consider a reconfigurable CPS with eight UAV (Unmanned Aerial Vehicle) agents (u_1-u_8). Each agent's position, speed, and acceleration are continuous variables, with communication delay as an external input. The system undertakes various flight missions based on runtime demands. To execute missions, agents collaborate through different communication strategies and composition architectures (e.g., classical virtual structures, leader-follower, behavior-based, and consensus-based ones [26]). Below is a example of its runtime mission sequence (M1-M4), with formation details in Fig. 1.

M1 All UAVs communicate with each other to locate their 3-D central point O. They then calculate and move to their target positions, forming a clockwise ring around the center O with a radius of r_1, at the same height of O.
M2 It is similar to M1, but forms a counter-clockwise ring with a radius of r_2.
M3 Upon receiving commands from the ground control center, odd-numbered and even-numbered UAVs adjust their speed to v_1 and v_2, respectively.
M4 It employs a hierarchical leader-follower architecture for tree formation. Phase 1: according to the leader u_1, followers u_{2-4} change their heading angles and move to desired relative positions. Phase 2: u_{2-4} act as leaders for u_5, u_{6-7}, and u_8 respectively, forming the third row of the tree similarly.



$\{a_{xy}, a_z, T, \sigma\} := F_1(x_{aim} - x, y_{aim} - y, z_{aim} - z, \mu, v)$
$t := 0, \mu := 0, v := 0$
comm$(t \geq d)$
Cruise
$\dot{t} = 1$
$\dot{x} = \mu\cos(\sigma)$
$\dot{y} = \mu\sin(\sigma)$
$\dot{z} = v$
Position Control
$\dot{t} = 1$
$\dot{\mu} = \mathrm{sgn}(T/2 - t) * a_{xy}$
$\dot{v} = \mathrm{sgn}(T/2 - t) * a_z$
$\dot{x} = \mu\cos(\sigma)$
$\dot{y} = \mu\sin(\sigma)$
$\dot{z} = v$
$t := 0, \sigma := \pi/2$
$(t \geq T)$
(a) in M1
$\{a_{xy}, a_z, T, \sigma\} := F_2(vx_{aim}, vy_{aim}, vz_{aim}, \mu, v)$
$t := 0$
comm$(t \geq d)$
Cruise
$\dot{t} = 1$
$\dot{x} = \mu\cos(\sigma)$
$\dot{y} = \mu\sin(\sigma)$
$\dot{z} = v$
Velocity Control
$\dot{t} = 1$
$\dot{\mu} = a_{xy}$
$\dot{v} = a_z$
$\dot{x} = \mu\cos(\sigma)$
$\dot{y} = \mu\sin(\sigma)$
$\dot{z} = v$
$t := 0$
$(t \geq T)$
(b) in M3
comm$(t \geq d)$
$t := 0$
Cruise
$\dot{t} = 1$
$\dot{x} = \mu\cos(\sigma)$
$\dot{y} = \mu\sin(\sigma)$
$\dot{z} = v$
(c) in M4, phase 2

Fig. 2. HA Models of u_3 in Scenarios with Differently Activated Functions

Although all agents are homogeneous in this case, each plays a different role. Moreover, even the same agent behaves differently across scenarios, and therefore, has different models. For example, in the models of u_3 shown in Fig. 2(a)-(c),

- in M1, it cruises while waiting for others' states. Once received, it calculates control parameters to adjust its position to the target location accordingly.
- in M3, it cruises while waiting for ground commands, and then adjusts velocity.
- in the phase 2 of M4, it cruises while broadcasting its state to followers, u_{6-7}.

These functions are embedded in UAV codes, activated dynamically at runtime.

Furthermore, the topology of the entire multi-UAV system changes dynamically, resulting in a complex and vast system model. In M1 and M2, vehicle-to-vehicle communication exists among all agents for central point calculation.

While in M3, agents are divided into two groups, equipped with ground-to-vehicle communication. In M4, system topology even shifts inside the mission and becomes hierarchical. Across scenarios, system models should be structured differently, and their shifts should also be formally modeled as a whole. Besides, specifying system requirements, especially topology-related ones, is also difficult. Many extra intermediate variables are required to describe system topology. We will address these modeling and specifying challenges in the next section.

3 Scenario-Based Formalism for Reconfigurable Systems

In this section, we present our formalism for reconfigurable CPSs with dynamic behaviors, compositions, and system topologies. It includes scenario-based system behavior modeling and topology-aware system property specifying.

3.1 Scenario-Based System Modeling

Our modeling language A) provides static template models for agents, along with an instantiation mechanism to customize each agent's actual internal behavior in each scenario; B) offers communication configurations to formalize multiple agents' interconnected behaviors in each single scenario; C) introduces scenario task configurations to describe system dynamic topologies in multiple scenarios. Finally, in D), we conclude the hierarchy of our modeling language.

A. Modeling Agent Runtime Behavior by Template Instantiation.

Definition 3 ***(Template Hybrid Automaton).*** *A template-HA is a tuple* $H^T = (X^T, U, Q, F, Inv, E^T, G, R, \alpha, \beta)$*, where*

- X^T *are variables classified by the function* α*:* $X^T \rightarrow \{X_{inp}, X_{out}, X_{local}\}$*.* X_{inp} *denotes global input variables, whose values can be updated by other agents during communication.* X_{out} *denotes global output variables that others can read during communication.* X_{local} *refers to private local variables.*
- E^T *are transitions classified by the function* β*:* $E \rightarrow \{E_{global}, E_{local}\}$*. Global transitions are public to other agents, therefore, communicating with others on global transitions is allowed. Local transitions are fixed in the agent's implementation, and are private to the agent itself.*
- U, Q, F, Inv, G, R *are parameters defined identically as HA in Definition 1.*

Template-HA formalizes the inherent features implemented in an agent. It constrains the agent's potential modes, transitions, and communication capabilities, including available communication ports, content, and channels, denoted as X_{inp}, X_{out}, and E_{global} respectively. To further constrain the agent's actual runtime behaviors in a specific scenario, we define Instance Hybrid Automaton.

Definition 4 ***(Instance Hybrid Automaton).*** *An instance-HA is a tuple* $H^I = (X, U, Q, F, Inv, E, G, R, \alpha, \beta, \gamma)$*, which can be* ***instantiated*** *from the template-HA* $H^T = (X, U, Q, F, Inv, E, G, R, \alpha, \beta)$ *through a transition activation function* γ*:* $E_{global} \rightarrow \{E_{act}, E_{deact}\}$*.*

Semantics: The state of H^I, denoted as $s = (q, x) \in Q \times X$, evolves either continuously in mode q or discretely by jumping to mode q' as defined in Definition 1. However, when it jumps via a global transition $e = (q, q')$, there is an extra constraint: the transition e must be activated by the function γ as $\gamma(e) = E_{act}$.

This instantiation mechanism improves template-HAs' flexibility and reusability, enabling adaptation to changing demands and reuse across scenarios by agents of the same type. For instance, in the multi-UAV system, agents are homogeneous and share the same template-HA, shown in brief in Fig. 3. It has three modes and five transitions. Variables are categorized by function α. For example, agent positions are global output variables, serving as permissible communication content among agents, and *inps* are global input variables as legal communication ports. Transitions are classified by function β, with global ones marked by dotted arrows, including `comm`, `commP`, and `commV`. Agents can be registered with ID and template-HA as shown in the purple box (line 2) in Fig. 4. During mission M4, when u_3 is leading u_{6-7}, the `comm` transition should be activated in u_3's instance-HA to inform its followers of its position, while u_{6-7}'s `commP` transitions should be activated for target calculation and position control. It can be concisely configured, as shown in the green box (line 15) in Fig. 4.

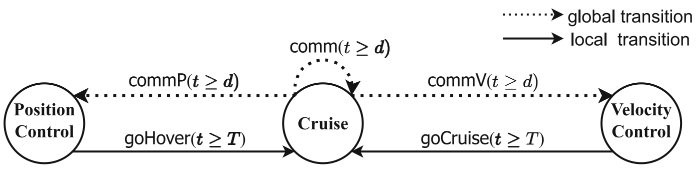
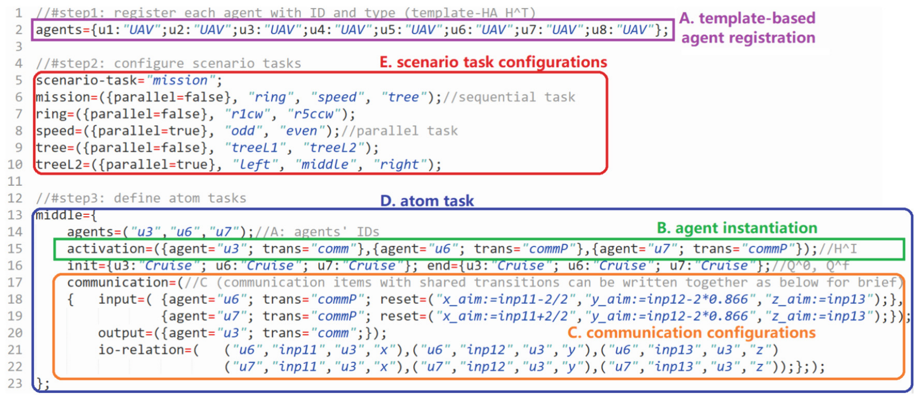


Fig. 3. An Overview of the Template-HA Model for UAV Agents


A. template-based agent registration
E. scenario task configurations
D. atom task
B. agent instantiation
C. communication configurations

```
 1  //#step1: register each agent with ID and type (template-HA H^T)
 2  agents={u1:"UAV";u2:"UAV";u3:"UAV";u4:"UAV";u5:"UAV";u6:"UAV";u7:"UAV";u8:"UAV"};
 3
 4  //#step2: configure scenario tasks
 5  scenario-task="mission";
 6  mission=({parallel=false}, "ring", "speed", "tree");//sequential task
 7  ring=({parallel=false}, "r1cw", "r5ccw");
 8  speed=({parallel=true}, "odd", "even");//parallel task
 9  tree=({parallel=false}, "treeL1", "treeL2");
10  treeL2=({parallel=true}, "left", "middle", "right");
11
12  //#step3: define atom tasks
13  middle={
14      agents=("u3","u6","u7");//A: agents' IDs
15      activation=({agent="u3"; trans="comm"},{agent="u6"; trans="commP"},{agent="u7"; trans="commP"});//H^I
16      init={u3:"Cruise"; u6:"Cruise"; u7:"Cruise"}; end={u3:"Cruise"; u6:"Cruise"; u7:"Cruise"};//Q^0, Q^f
17      communication=(//C (communication items with shared transitions can be written together as below for brief)
18      {   input=( {agent="u6"; trans="commP"; reset=("x_aim:=inp11-2/2","y_aim:=inp12-2*0.866","z_aim:=inp13");},
19                  {agent="u7"; trans="commP"; reset=("x_aim:=inp11+2/2","y_aim:=inp12-2*0.866","z_aim:=inp13");});
20          output=({agent="u3"; trans="comm";});
21          io-relation=(   ("u6","inp11","u3","x"),("u6","inp12","u3","y"),("u6","inp13","u3","z")
22                          ("u7","inp11","u3","x"),("u7","inp12","u3","y"),("u7","inp13","u3","z"));};);
23  };
```

Fig. 4. Multi-UAV System Modeling Snippet in a Structured Configuration File (Color figure online)

B. Modeling Agent Composition by Communication Configurations

Agents are usually complexly coupled rather than isolated in a CPS. To formalize agents' composition, we define communication items and configurations for them.

Definition 5 *(**Communication Item**). Given two runtime agents modeled as H_1^I=$(X_1, U_1, Q_1, F_1, Inv_1, E_1, G_1, R_1, \alpha_1, \beta_1, \gamma_1)$ and H_2^I=$(X_2, U_2, Q_2, F_2, Inv_2, E_2, G_2, R_2, \alpha_2, \beta_2, \gamma_2)$, where $X_1 \cap X_2 = \emptyset$, $U_1 \cap U_2 = \emptyset$, $E_1 \cap E_2 = \emptyset$, $Q_1 \cap Q_2 = \emptyset$, a communication item for them is a tuple $c = (H_1^I, e_1, I, H_2^I, e_2, O, R'_{e_1})$, where*

- *$e_1 \in E_1$, $e_2 \in E_2$, $\beta_1(e_1) = \beta_2(e_2) = E_{global}$, $\gamma_1(e_1) = \gamma_2(e_2) = E_{act}$;*
- *I, O are ordered sets, $|I| = |O|$,[2] elements in I must be global input variables in H_1^I, while elements in O must be global output variables in H_2^I, i.e., $\forall x \in I,\ x \in X_1 \bigwedge \alpha_1(x) = X_{inp}$; $\forall x \in O,\ x \in X_2 \bigwedge \alpha_2(x) = X_{out}$;*
- *R'_{e_1} is an additional reset function for variables in X_1.*

A communication item formalizes a composition relationship between two agents. Under the item c, global transitions e_1 and e_2 are bound together as a communication channel, expected to happen simultaneously under their guard conditions. Once they happen, communication contents in O will be received by ports in I in sequence. The additional reset function R'_{e_1} will be executed next, followed by agents' original reset functions.

Definition 6 *(**Communication Configuration**). Given two agents modeled as H_1^I and H_2^I as above, a communication configuration C=$()|(c)|(c_1, \cdots, c_n)$, where c_i=$(H_{1\,i}^I, e_{1i}, I_i, H_{2\,i}^I, e_{2i}, O_i, R'_{e_{1\,i}})$ is a communication item for H_1^I and H_2^I.*

- *$C = ()$: H_1^I and H_2^I are independent;*
- *$C = (c)$: H_1^I and H_2^I are connected by the item c as Definition 5;*
- *$C = (c_1, \cdots, c_n)$: for any c_i and c_j $(1 \le i < j \le n)$, if $e_{1i} \cap e_{2i} \cap e_{1j} \cap e_{2j} \neq \emptyset$, then all these transitions are expected to happen simultaneously, together with the communication between I_i and O_i, I_j and O_j. Otherwise, transitions e_{1i}, e_{2i} are bound together as Definition 5, so are transitions e_{1j} and e_{2j}.*

Semantics: Communication configuration, consisting of none, one, or multiple items, formalizes two agents' composition. Their composition under a communication configuration can be formalized as another larger instance-HA as follows.

Definition 7 *(**Instance-HA Composition**). Given two instance-HA $H_1^I = (X_1, U_1, Q_1, F_1, Inv_1, E_1, G_1, R_1, \alpha_1, \beta_1, \gamma_1)$ and $H_2^I = (X_2, U_2, Q_2, F_2, Inv_2, E_2, G_2, R_2, \alpha_2, \beta_2, \gamma_2)$, where $X_1 \cap X_2 = \emptyset$, $U_1 \cap U_2 = \emptyset$, $E_1 \cap E_2 = \emptyset$, $Q_1 \cap Q_2 = \emptyset$, the composition of H_1^I and H_2^I under communication configuration C can be modeled as instance-HA $H^I = (X, U, Q, F, Inv, E, G, R, \alpha, \beta, \gamma)$, where*

- *$X = X_1 \cup X_2 = \{X_1, X_2\}$ are variables, classified by $\alpha(x) = \begin{cases} \alpha_1(x), x \in X_1 \\ \alpha_2(x), x \in X_2 \end{cases}$.*

[2] For any set A, |A| denotes the number of elements in A.

– $U = U_1 \cup U_2 = \{U_1, U_2\}$ *are external inputs;* $Q = Q_1 \times Q_2$ *are discrete modes;*
– $F = \{F_{(q_i,q_j)} | \ (q_i, q_j) \in Q\}$ *are flow functions, where* $\begin{cases} \dot{X}_1 = F_{1q_i}(X_1, U_1) \\ \dot{X}_2 = F_{2q_j}(X_2, U_2) \end{cases}$.
– $Inv = \{Inv_{(q_i,q_j)} = Inv_{1(q_i)} \cap Inv_{2(q_j)} | \ (q_i, q_j) \in Q\}$ *are invariant conditions;*
– $E \subseteq Q \times Q$ *are discrete transitions constructed in the following two ways: for* $\forall e_a = (q_a, q'_a) \in E_1, \forall e_b = (q_b, q'_b) \in E_2$,
  1) if $\exists c = (H^I_1, e_1, I, H^I_2, e_2, O, R'_{e_1}) \in C$, $\{e_a, e_b\}=\{e_1, e_2\}$ *or* $\{e_a, e_b\}=\{e_2, e_1\}$*:*
  - $e_c=((q_a, q_b), (q'_a, q'_b)) \in E$*, classified by* $\beta(e_c)=E_{global}$*, activated by* $\gamma(e_c)=E_{act}$*;*
  - $G_{e_c}=G_{e_a} \cap G_{e_b}$ *is the guard condition, and* R_{e_c} *resets* $X := R_{e_c}(X, U)$ *as:*
    * $X := \{R_{e_1}(R'_{e_1}(R_{I/O}(X_1), U_1), U_1), R_{e_2}(X_2, U_2)\}$ *when* $\{e_a, e_b\} = \{e_1, e_2\}$*;*
    * $X := \{R_{e_2}(X_1, U_1), R_{e_1}(R'_{e_1}(R_{I/O}(X_2), U_2), U_2)\}$ *when* $\{e_a, e_b\} = \{e_2, e_1\}$*;*

    where function $R_{I/O}(x) = \begin{cases} O(i) & , I(i) = x \\ x & , x \in X - I \end{cases}$.

  2) otherwise:
  - $e_c = ((q_a, q_b), (q'_a, q_b)) \in E$, $e_d = ((q_a, q_b), (q_a, q'_b)) \in E$*;*
  - $\beta(e_c) = \beta(e_a)$, $\gamma(e_c) = \gamma(e_a)$, $G_{e_c} = G_{e_a}$, $R_{e_c}(X) := \{R_{e_a}(X1, U_1), X_2\}$*;*
  - $\beta(e_d) = \beta(e_b)$, $\gamma(e_d) = \gamma(e_b)$, $G_{e_d} = G_{e_b}$, $R_{e_d}(X) := \{X_1, R_{e_b}(X2, U_2)\}$.

The composition of multiple agents can be formalized by applying Definition 7 recursively. With communication configurations, agents' static compositions are efficiently configured, and always adhere to their inherent communication capabilities. For example, as shown in the orange box (line 17-22) in Fig. 4, communication configuration defines the composition of u_3 and its followers u_{6-7} in mission M4, where u_{6-7} gets u_3's position to calculate their own target positions.

C. Modeling Dynamic Typology by Scenario Task Configurations

In a reconfigurable CPS, agents usually interconnect dynamically, and the system topology shifts across scenarios during runtime. To capture this dynamic feature, we introduce atom tasks and scenario tasks below.

Definition 8 ***(Atom Task)****. Given a system with multiple agents indexed by unique IDs, an atom task of the system is a tuple* $AT=(A, \mathbb{H}^I, Q^0, Q^f, \mathbb{C})$*, where*

– $A = \{A_i\}_{i=1}^n$ *are the IDs of all involvezd agents in the current scenario;*
– $\mathbb{H}^I=\{H^I_i\}_{i=1}^n$ *are involved agents' instance-HAs,* H^I_i *instantiated from* $H^T_{A_i}$*;*
– $Q^0=\{Q^0_i\}_{i=1}^n$, $Q^f=\{Q^f_i\}_{i=1}^n$ *are agents' initial and final modes,* $Q^0_i, Q^f_i \in Q_{A_i}$*;*
– $\mathbb{C}=\{C_i\}_{i=1}^m$ *are communication configurations defined on* $\mathbb{H}^I$*.*

An atom task describes a scenario, detailing active agents, behaviors, and compositions. It begins with agents evolving simultaneously from initial modes. It ends once all agents have reached final modes. For example, the scenario where u_3 leads u_{6-7} in mission M4 is described in the blue box (line 13-23) in Fig. 4.

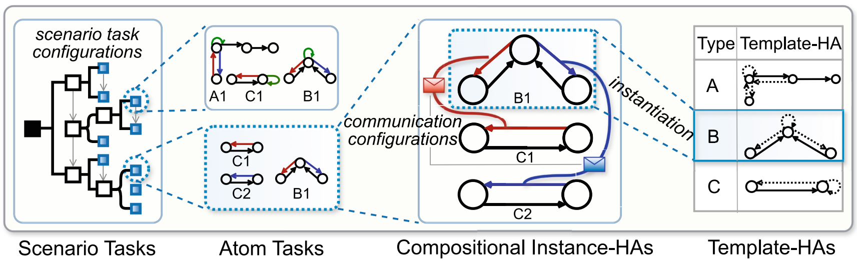


Fig. 5. The Hierarchy of Our Scenario-based Modeling Language

Besides, we offer a template mechanism for atom tasks to enhance task reusability. In a task template, involved agents' types, behaviors, and compositions are defined, leaving their IDs and some const variables for customization. For example, missions M1 and M2 can be customized by the same task template.

Definition 9 ***(Scenario Task)***. *A scenario task* $ST = AT|(ST;ST)|(ST||ST)$, *where* $(ST;ST)$ *denotes the sequential execution of two scenario tasks, while* $(ST||ST)$ *denotes the parallel execution of two scenario tasks. Therefore, it can either be a single atom task or a sequential/parallel compositional task.*

Agents involved in the two subtasks of a parallel compositional task should be disjoint. Besides, if an agent is involved first in atom task AT_1 *and then in* AT_2, *its initial mode in* AT_2 *should be consistent with its final mode in* AT_1.

Scenario task specifies the flow of scenarios recursively, which in turn, configures the dynamic shifts of system topologies. Therefore, it is also called **scenario task configuration** in this work. For example, as shown in the red box (line 5-10) in Fig. 4, the dynamic topology of the multi-UAV system in mission M1-4 is defined by scenario task configurations. The scenario task $T_{mission} = ((T_{ring}; T_{speed}); T_{tree})$ represents the sequential execution of three scenario tasks: T_{ring} for M1-2, T_{speed} for M3, and T_{tree} for M4. See Fig. 8a for a graphical illustration.

D. Hierarchical Modeling for Reconfigurable CPSs

The hierarchy of our modeling language is summarized in Fig. 5. The design idea is to abstract agents' inherent features into template-HA models, so that reconfigurable CPSs can be modeled through simple instantiations and configurations.

Agent internal runtime behavior is modeled by an instance-HA, which is instantiated from its inherent template-HA model. Multiple agents' interconnected behaviors are modeled by the composition of their instance-HAs, which are connected by communication configurations. Then, the behavior of a reconfigurable CPS in a specific scenario is modeled by an atom task, with its involved agents' compositional instance-HAs. Finally, the CPS's runtime behaviors in changing scenarios with dynamic topologies are structured by scenario task configurations.

3.2 Specifying System Requirements in Topology-Aware STL

Topology-aware STL is a domain-specific extension of STL, designed to specify requirements for multi-agent systems with dynamic topologies. Although it shares the same expressiveness as STL, it facilitates the construction and evaluation of specifications related to system topologies.

Definition 10 ***(Topology-Aware STL).*** *A topology-aware STL formula φ interpreted over boolean, integer, and real variables $\mathcal{X}=\{\mathcal{X}_b, \mathcal{X}_i, \mathcal{X}_r\}$ is defined as:*

$\varphi := true \mid \psi \mid \neg\varphi_1 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathbf{U_I} \varphi_2$, with the atomic predicate ψ as
$\psi := End[T] \mid Mode[A] = d_q \mid \theta(\mathcal{X}_r) \geq d$, where

$End[T] \in \mathcal{X}_b$ indicates whether a scenario task T has been completed. $Mode[A] \in \mathcal{X}_i$ denotes the mode index of an agent A, while d_q is an integer mode index constant. Real variables $\mathcal{X}_r$ consist of agents' variables, external inputs, and scenario tasks' run time, denoted as $Time[T]$ for the scenario task T. Function θ: $\overline{\mathbb{R}}^n \to \mathbb{R}$ maps valuations of $\mathcal{X}_r$ into real numbers, and d is a constant.

Semantics: Given a topology-aware STL formula φ, a signal w of $\mathcal{X}$, and time t, the quantitative semantics of its robustness degree $\rho(\varphi, w, t)$ is defined as Definition 2, except the robustness of atomic predicates is defined as below:

$$\rho\ (End[T], w, t) = \begin{cases} 0 & , T \text{ is a completed atom task} \\ -1 & , T \text{ is an uncompleted atom task} \\ (\sum_{i=1}^{2} \rho(End[T_i], w, t))/2 & , T = (T_1; T_2) \text{ is a sequential task} \\ \min_{i\in[1,2]} \rho(End[T_i], w, t) & , T = (T_1 || T_2) \text{ is a parallel task} \end{cases} \tag{2}$$

$$\rho\ (Mode[A] = d_q, w, t) = -\mathbf{Dist}_A(Mode[A](t), d_q) \tag{3}$$

$$\rho\ (\theta(\mathcal{X}_r) \geq d, w, t) = \theta(w(t)) - d \tag{4}$$

The robustness of $End(T)$ indicates scenario task T's completion extent. The robustness of the predicate $q(A) = d_q$ indicates agent A's distance to the target mode d_q. The function $\mathbf{Dist}_A$: $Q_A \times Q_A \to \mathbb{Z}^+$ maps two modes in agent A to the minimum number of hops required to jump from the first mode to the second by transitions in A. The robustness degree is negative only if the signal of variables violates the formula, and it decreases as the severity of the violation increases. In this work, signals are interpreted according to system trajectories. When the trajectory is clear, we use a simpler notation $\rho(\varphi)$, instead of $\rho(\varphi, w, 0)$, to denote the robustness of formula φ on trajectory signal w at time zero.

Topology-aware STL is augmented with customized variables, predicates, and semantics. Predefined elements for task duration, completion, and mode identification help to describe topology-related requirements while avoiding introducing loads of extra variables, constraints, and functions to system models. For example, Table 1 lists some topology-aware STL specifications for the multi-UAV system modeled in the scenario task $T_{mission}$. Furthermore, topology-aware STL provides more meaningful quantitative semantics with better robustness degrees. This can facilitate CPS falsification by offering more effective guidance.

Table 1. Examples of Topology-Aware STL Specifications

Topology-Aware STL Specification	Description
$\Box(\neg(Time[T_{tree}] \geq 10))$	tree formation takes less than 10 s
$\Diamond(End[T_{ring}] \wedge \frac{Time[AT_{r5ccw}]}{Time[AT_{r1cw}]} \geq 1)$	ring formations are completed eventually and the second ring takes no less time than the first
$\neg\Diamond(\Box_{[0,4]}(Mode[u_2] = 1 \wedge u_2.\mu \geq 6.6))$	u_2 never stays in the Cruise mode (index 1) with dangerous horizontal speeds for 4 s

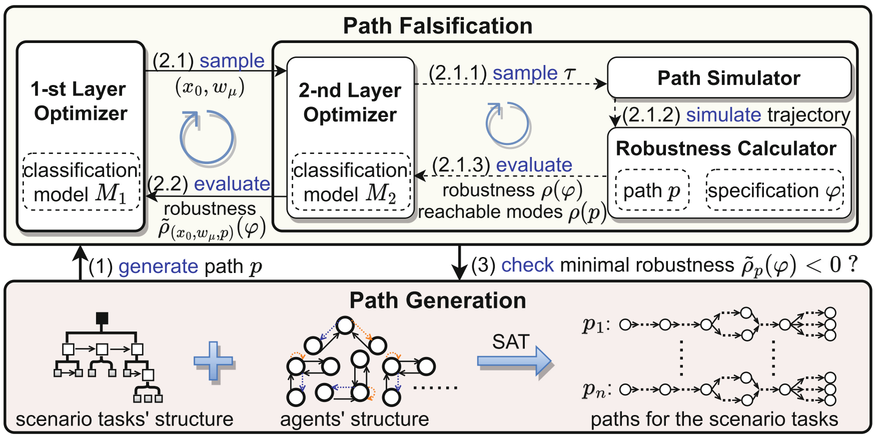

Path Falsification
1-st Layer Optimizer
classification model M_1
(2.1) sample (x_0, w_μ)
(2.2) evaluate robustness $\tilde{\rho}_{(x_0,w_\mu,p)}(\varphi)$
2-nd Layer Optimizer
classification model M_2
(2.1.1) sample τ
Path Simulator
(2.1.2) simulate trajectory
Robustness Calculator
path p
specification φ
(2.1.3) evaluate robustness $\rho(\varphi)$ reachable modes $\rho(p)$
(1) generate path p
(3) check minimal robustness $\tilde{\rho}_p(\varphi) < 0$?
Path Generation
scenario tasks' structure
agents' structure
SAT
p_1:
p_n:
paths for the scenario tasks

Fig. 6. The Path-oriented Optimization-based Falsification Framework

4 Path-Oriented Optimization-Based System Falsification

4.1 Falsification Framework

For a reconfigurable CPS modeled in a scenario task T and its requirement specified as a topology-aware STL φ, falsification aims to find a witness trajectory that violates φ, i.e. specification φ's robustness is negative on this trajectory. We denote a counterexample pair as (x_0, w_μ, p, τ), under which the trajectory is a witness. Here, x_0 represents the initial value of agents' variables, w_μ represents the signal of external inputs, p represents the path of agents' mode and transition sequence, and τ represents agents' dwelling time in modes along p.

To fully explore both discrete and continuous dimensions of the vast search space, our falsification approach uses a path-oriented strategy, as shown in Fig. 6. At the bottom, it explores discrete space, generating paths based on the discrete structure of agents and scenario tasks. On the top, it explores continuous space for each generated path p, performing path falsification to find an optimal pair (x_0, w_μ, τ) that minimizes the specification φ's robustness on trajectories along p. The falsification problem is solved when the minimal robustness is negative.

Our SAT-based path generation and optimization-based path falsification are given next. For brevity, we will only consider models with nondeterministic semantics, since deterministic models can be viewed as a simplified case.

4.2 Path Generation for Hierarchical Scenario Tasks

For a scenario task, its path determines how its involved agents change their modes and make their transitions during it. Below is the definition of path.

Definition 11 ***(Path of Scenario Task)****. For a given scenario task ST, if*

- *$ST{=}AT{=}(A, \mathbb{H}^I, Q^0, Q^f, \mathbb{C})$ is an atom task: a path of it with length l is a tuple $p{=}(\{\mathcal{M}^i, \mathcal{T}^i\}_{i=0}^{l-1}, \mathcal{M}^l)$, where $\mathcal{M}^i{=}(\mathcal{M}^i_1, ..., \mathcal{M}^i_{|A|})$ denotes all agents' modes in the i-th step, and $\mathcal{T}^i{=}(\mathcal{T}^i_1, ..., \mathcal{T}^i_{|A|})$ denotes their next transitions after this step. For each agent $A_k \in A$, the tuple p should satisfy:*
  - ***agent discrete structure****: $\mathcal{M}^i_k$ should be a mode of agent A_k's instance-HA $H^I_k \in \mathbb{H}^I$; $\mathcal{T}^i_k$ can either be a transition that jumps from $\mathcal{M}^i_k$ to $\mathcal{M}^{i+1}_k$ in H^I_k or a stutter transition* `S` *that indicates no mode shift.*
  - ***atom task requirement****: agent A_k should start from its initial mode Q^0_k and end in its final mode Q^f_k. If A_k is connected to another agent by a communication item in $\mathbb{C}$, their synchronized transitions should occur simultaneously.*
- *ST is a sequential task: its path is the concatenation of its subtasks' paths.*
- *ST is a parallel task: its path is the combination of its subtasks' paths.*

A composition scenario task's paths are produced by recursively generating its subtasks' paths and then concatenating or combining them. An atom task's paths are generated by SAT encoding, solving, and decoding.

We transform the length-bounded path generation for atom tasks into a SAT problem. Constraints for paths with lengths no longer than m are encoded as the propositional formula set $\mathcal{F}^m$ in Eq.(8). Specifically, constraints on each agent's discrete structure are encoded as Eq.(5)-(6), and requirements on the atom task are encoded as Eq.(7). To solve this SAT problem, advanced SAT solvers like Glucose [9] and CryptoMiniSat [60] can be used, and then the solutions can be decoded into paths. For instance, ((`Cruise`, `Cruise`, `Cruise`), (`comm`, `commP`, `commP`), (`Cruise`, `Position Control`, `Position Control`), (`S`, `goHover`, `S`), (`Cruise`, `Cruise`, `Position Control`), (`S`, `S`, `goHover`), (`Cruise`, `Cruise`,

`Cruise`)) is a path of the atom task AT_{middle} in the running example.

$$NEXT_k^i := \bigwedge_{q \in Q_{A_k}} ((\mathcal{M}_k^i = q) \rightarrow ((\bigvee_{e=(q,q') \in E_{A_k} \wedge \gamma_{A_k}(e) = E_{act}} (\mathcal{T}_k^i = e \wedge \mathcal{M}_k^{i+1} = q')) \vee (\mathcal{T}_k^i = S \wedge \mathcal{M}_k^{i+1} = q))) \tag{5}$$

$$EXCLD_k^i := (\bigwedge_{q,q' \in Q_{A_k} \wedge q \neq q'} (\mathcal{M}_k^i = q \rightarrow \mathcal{M}_k^i \neq q')) \wedge (\bigwedge_{e,e' \in (E_{A_k} \cup S) \wedge e \neq e'} (\mathcal{T}_k^i = e \rightarrow \mathcal{T}_k^i \neq e')) \tag{6}$$

$$INIT_k := (\mathcal{M}_k^0 = Q_k^0),\ END_k^l := (\mathcal{M}_k^l = Q_k^f),\ SYNC_k^i := \bigwedge_{(H_k^I, e_1, I, H_{k'}^I, e_2, O, R'_{e_1}) \in C_j \in \mathbb{C}} (\mathcal{T}_k^i = e_1 \leftrightarrow \mathcal{T}_{k'}^i = e_2) \tag{7}$$

$$\mathcal{F}^m := \bigwedge_{A_k \in \mathbb{A}} (INIT_k \wedge (\bigvee_{0 < l \leq m} (END_k^l \wedge \bigwedge_{0 \leq i < l} SYNC_k^i \wedge \bigwedge_{0 \leq i < l} NEXT_k^i \wedge \bigwedge_{0 \leq i \leq l} EXCLD_k^i))) \tag{8}$$

4.3 Optimization-Based Falsification for Paths

Path falsification is achieved by optimizing for a trajectory that minimizes the target specification's robustness along the path. However, this optimization problem can be high-dimensional, nonlinear, nonconvex, and challenging, due to the complexity of agents, topologies, and scenarios involved. Therefore, we developed a two-layered classification-model-based optimization method for it, along with an efficient robustness calculation algorithm.

Two-Layered Classification-Model-Based Optimization. In Fig. 6, the first layer optimizes for the best pair of initial values and external inputs that minimizes robustness. For each pair (x_0, w_μ), the second layer performs multi-objective optimization for the best dwelling time τ that minimizes robustness while maximizes the number of reachable modes along the path. τ is evaluated by the two objective functions' weighted sum, i.e. $\min f = w_1 \rho(\varphi) + w_2 \rho(p)$ $(|w_1| + |w_2| = 1, w_1 > 0, w_2 < 0)$, with weighting coefficients updated dynamically during optimization.

Both optimization layers are solved with classification models. Unlike classical heuristic optimization methods with weak theoretical guarantees or poor scalability, classification-model-based optimization has a grounded theory about complexity and convergence [39,64]. It has been theoretically proven to solve problems with local-Lipschitz continuity in polynomial time, and empirically proven to be scalable to high-dimensional problems. Its basic idea is to learn a classification model iteratively to discriminate bad solutions from good ones.

We employ a hyper-rectangle M_1 in the first layer as the classification model to represent the positive region, and a hyper-rectangle M_2 in the second layer as Fig. 6. In both layers, solutions are sampled from corresponding models, evaluated accordingly, and classified into positive/negative ones to refine models. The sampling, evaluating, and model refinement cycles iterate to enhance classification models' accuracy and optimize solutions. Our two-layered optimization enables accurate model refinement and alleviates the scalability issue.

Efficient Robustness Calculation. During falsification, system simulation and robustness calculation are compute-intensive and time-consuming. We introduce time-context and lifespan in temporal logic to reduce them.

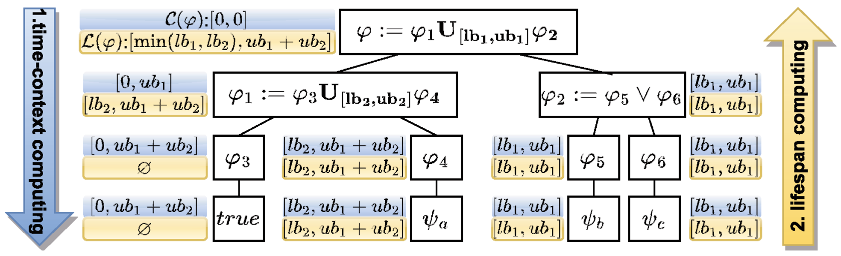


Fig. 7. An Example of Lifespan Calculation for the Target Specification φ

Definition 12 *(**Time-Context, Lifespan**). Given a target temporal logic specification φ, for any formula φ' in it (including φ itself), its **time-context** $\mathcal{C}(\varphi'|\varphi)$ is the period during which its satisfaction affects φ's satisfaction, its **lifespan** $\mathcal{L}(\varphi'|\varphi)$ is the period during which the signals affect its satisfaction in its time-context. We omit the target specification φ when it is clear. Relations between φ' and its subformulas, in terms of time-context and lifespan are:*

$$\begin{array}{rll}
\varphi' := true \rightarrow \mathcal{C}(true) = \mathcal{C}(\varphi') & \mathcal{L}(\varphi') = \varnothing & (9)\\
\varphi' := \psi \rightarrow \mathcal{C}(\psi) = \mathcal{C}(\varphi') & \mathcal{L}(\varphi') = \mathcal{L}(\psi) = \mathcal{C}(\psi) & (10)\\
\varphi' := \neg\varphi_1 \rightarrow \mathcal{C}(\varphi_1) = \mathcal{C}(\varphi') & \mathcal{L}(\varphi') = \mathcal{L}(\varphi_1) & \\
\varphi' := \varphi_1 \vee \varphi_2 \rightarrow \mathcal{C}(\varphi_1) = \mathcal{C}(\varphi_2) = \mathcal{C}(\varphi') & \mathcal{L}(\varphi') = \mathcal{L}(\varphi_1) \cup \mathcal{L}(\varphi_2) & \\
\varphi' := \varphi_1 \mathbf{U}_{[\mathbf{lb},\mathbf{ub}]} \varphi_2 \rightarrow \mathcal{C}(\varphi_1) = \mathcal{C}(\varphi') \cup (ub \oplus \mathcal{C}(\varphi')) & \mathcal{L}(\varphi') = \mathcal{L}(\varphi_1) \cup \mathcal{L}(\varphi_2) & \\
\mathcal{C}(\varphi_2) = (lb \oplus \mathcal{C}(\varphi')) \cup (ub \oplus \mathcal{C}(\varphi')) & &
\end{array}$$

To compute the target specification φ's robustness, we only need to simulate and record system trajectory within φ's lifespan. This helps to avoid unnecessary trajectory simulations, saving computation time and memory. The lifespan of the target specification φ is computed as defined in Definition 12. We demonstrate an example of it in Fig. 7. Note that the time-context of φ is $[0, 0]$, as our goal is to evaluate $\rho(\varphi, w, 0)$, the target specification's quantitative satisfaction on the trajectory signal starting at time zero. As shown in blue boxes in Fig. 7, starting with φ's time-context $[0, 0]$, the time-context of all subformulas is computed from top to bottom along the parse tree. Next, we get the lifespan of all atomic predicates according to Eq.(9)–(10). Finally, the lifespan of all formulas, including the root φ, is computed from bottom to top as yellow boxes.

During the robustness calculation of the target specification, its subformulas' time-context also can be utilized, avoiding unnecessary robustness calculations. Efficient dynamic programming algorithms have been developed for temporal logic robustness calculation [34,56]. Our algorithm follows the dynamic programming principle and further improves efficiency by considering the time-context of formulas in the target specification. Pseudocode appears in Algorithm 1.

This algorithm calculates the robustness of all formulas in φ's parse tree at each timestamp in the simulated trajectory, constructing a dynamic programming table $\mathcal{R}$ to store the robustness of formula $\phi[i]$ at time $\boldsymbol{t}[j]$ in $\mathcal{R}[i, j]$.

Algorithm 1: Time-Context Guided Robustness Calculation

```
Input: $\varphi$: target specification, $w = (\boldsymbol{t}, \boldsymbol{\mathcal{X}})$: trajectory signal.
Output: $\rho$: Robustness of the target specification.
1  Function computeRobustness($\varphi, w$)
2      $\phi$=parse($\varphi$);          ▷ formulas in $\varphi$'s parse tree in partial order (top-down)
3      for $i = |\phi|$ to 1 do          ▷ from bottom atoms up to the root $\varphi$
4          for $j = |\boldsymbol{t}|$ to 1 do          ▷ from last timestamp to time zero
5              if $\boldsymbol{t}[j] < \phi[i].\mathcal{C}[1]$ then break;    ▷ skip this robustness calculation
6              else if $\boldsymbol{t}[j] > \phi[i].\mathcal{C}[2]$ && $\phi[i] \neq \varphi_1 \mathbf{U}_{[c,\infty)} \varphi_2$ then continue;    ▷ skip
7              else    ▷ calculate robustness at timestamps within time-context
8                  if $\phi[i] := true$ then $\mathcal{R}[i,j] = \top$;
9                  else if $\phi[i] := \psi$ then $\mathcal{R}[i,j] = \rho(\psi, w, \boldsymbol{t}[j])$;    ▷ as Eq.(2)-(4)
10                 else if $\phi[i] := \neg\phi[k]$ then $\mathcal{R}[i,j] = -\mathcal{R}[k,j]$;    ▷ as Eq.(1)
11                 else if $\phi[i] := \phi[k_1] \vee \phi[k_2]$ then $\mathcal{R}[i,j] = \max(\mathcal{R}[k_1,j], \mathcal{R}[k_2,j])$;
12                 else if $\phi[i] := \phi[k_1]\mathbf{U}_I\phi[k_2]$ then    ▷ as Eq.(1)
13                     $m = j$;   $\mathcal{R}[i,j] = \bot$;   $r_1 = \top$;
14                     while $(t[m] - \boldsymbol{t}[j]) \notin I$ do $r_1 = \min(r_1, \mathcal{R}[k_1, m++])$;
15                     while $(\boldsymbol{t}[m] - \boldsymbol{t}[j]) \in I$ do
16                         if $I = [c, \infty)$ && $j \neq |t|$ && $(\boldsymbol{t}[m] - \boldsymbol{t}[j+1]) \in I$ then break;
17                         $\mathcal{R}[i,j] = \max(\mathcal{R}[i,j], min(r_1, \mathcal{R}[k_2, m]))$;
18                         $r_1 = \min(r_1, R[k_1, m++])$;
19                     if $I$=$[c,\infty)$&&$j \neq |t|$ then
                        $\mathcal{R}[i,j] = \max(\mathcal{R}[i,j], \min(\mathcal{R}[k_1,j], \mathcal{R}[i,j+1]))$;
20     return $\rho = \mathcal{R}[1,1]$          ▷ output $\varphi$'s robustness at time zero
```

It calculates robustness for atomic predicates, higher-level subformulas, and the root specification sequentially (line 3-19). Each is calculated from the last timestamp backwards to the initial timestamp (line 4-22). It skips unnecessary robustness calculations for timestamps outside the formula's time-context (line 5-6), i.e., the formula's satisfaction at that point does not affect the target specification's satisfaction. For timestamps inside the time-context, the formula's robustness is calculated by reusing its robustness at future timestamps and its subformulas' robustness (line 8-19), according to its robustness definitions in Eq.(1)–(4).

5 Implementation and Evaluation

5.1 Implementation and Research Questions

We have implemented our **S**ce**N**ar**I**o-based reconfigurable CPS modeling and **F**alsi**F**ication approach in a tool called SNIFF[3] in C++. It provides a graphical user interface (GUI) for modeling, supporting the construction of agent template models and agents' runtime instantiations, communication configurations and

[3] SNIFF is available at https://github.com/njuwjw/SNIFF.

scenario task configurations. It also provides path-oriented CPS falsification, displaying witness paths and signals.
Below are research questions to evaluate our modeling and falsification approach:

RQ1: Can `SNIFF` help users reduce modeling time for reconfigurable CPS?
RQ2: Can `SNIFF` help users improve CPS modeling accuracy?
RQ3: Is `SNIFF` effective and efficient at CPS falsification? In particular, how is its scalability and runtime performance?

Next, we introduce the reconfigurable CPSs used in our evaluations in detail.

Example 1: Multi-UAV System. The motivating example system is used for our evaluations. Initially, eight UAVs are scattered in a closed space, $x, y \in [-10, 10]$, $z \in [1, 10]$. During runtime, they are dynamically connected with uncertain communication delays of $d \in [0.05, 0.2]$. Its behavior is modeled by the scenario task $T_{mission}$, as shown in Fig. 8a. Atom tasks are in grey. Sequential and parallel compositional tasks are drawn in white-filled rectangles with square and rounded corners, respectively. Besides, for fair modeling evaluations, the flight mission was slightly modified as shown in Fig. 8b. Four UAVs are involved and its scenario task includes three compositional tasks and six atom tasks.

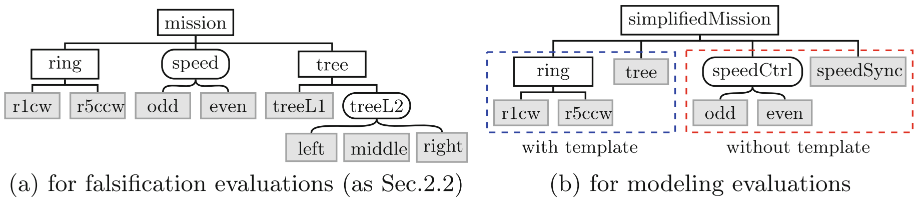


Fig. 8. Scenario Tasks for the Multi-UAV System

Example 2: Automatic Assembly System. It consists of four conveyors, three robots with grippers, and six robots with welding guns for car production. Conveyors transport floor, door, and roof panels along the assembly line, starting at $x \in [-2, 0]$. Grabbing robots communicate with conveyors with a delay of $t \in [0, 0.1]$, and wheel around to grasp, move, and place doors and roofs. Welding robots work together to spot weld joints in $x \in [8, 12], y \in [8, 12]$.

Its behavior is modeled by the scenario task $T_{assembly}$ as Fig. 9. Initially, in $T_{conveyGrasp}$, conveyors and grabbing robots collaborate to convey and grasp four panels in parallel. Then, in task T_{move}, grabbing robots first move doors and then slot the roof. Finally, in $T_{spotWeld}$, welding robots simultaneously weld joints while grabbing robots return home once relevant joints are welded, ready for the next assembly.

Experimental Setup. To evaluate `SNIFF`'s modeling time and accuracy, we conducted a user study comparing it to another tool called `SpaceEx` [36],

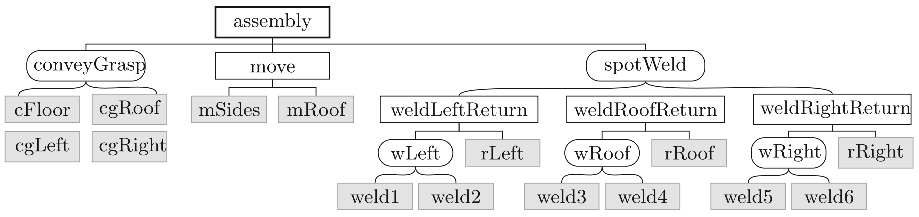


Fig. 9. Scenario Tasks for the Automatic Assembly System

which is a well-established platform for hybrid system modeling and verification. `SpaceEx`'s modeling language, called SX [21,28], is based on compositional hybrid automata, presented as base or network components. Both tools provide a graphical modeling editor with automatic user action logging, which allows us to evaluate modeling time accurately. Also, graphical models are saved automatically in their respective modeling languages, enabling our checks for modeling accuracy.

This user study was conducted with 111 computer science department students, including undergraduates, graduates, and PhDs. They received a 50-minute training session on CPS modeling and were required to model a multi-UAV system, described as Fig. 8b, using both tools in a week. Participants were given with a UAV template model and the templates for half atom tasks (in the blue box in Fig. 8b) in SNIFF format. They are also provided with base components in SpaceEx formats, containing the same information as the SNIFF templates we provided. Specifically, it contains information on all potential dynamics of a UAV agent and agent compositional information in three atom tasks.

To evaluate `SNIFF`'s falsification performance, we conducted a experiment comparing it with `S-TaLiRo` [8], an advanced optimization-based CPS falsification tool. We ran `S-TaLiRo` with its SOAR [48] option, which outperformed all of its other optimization options in our benchmarks. Systems are modeled in 'hautomaton' class as its input. We falsified various temporal specifications for the above two example systems on both tools. We conducted all experiments 100 times on the same PC (Intel Core i5-12500, 16GB RAM) with a time limit of 1800 s. The system simulation limit is 1000 iterations for deterministic benchmarks and 25000 for non-deterministic ones. We did not use `SpaceEx` for nonlinear systems' temporal logic falsification comparison since it mainly focuses on reachability verification for hybrid systems with piecewise affine dynamics.

5.2 Experimental Evaluation and Analysis

Time Cost of Modeling. According to participants' modeling logs, models were constructed in an average of 46.9 min in `SNIFF`, while it took 308.1 min, 6.6 times longer, in `SpaceEx`. To further analyze how time was saved in `SNIFF`, Table 2 compares their time cost in different modeling elements.

Table 2. Time Costs to Model Different System Elements in SNIFF and SpaceEx

Type of Elements	Number	Time (min)		Average Time (min)	
		SNIFF	SpaceEx	SNIFF	SpaceEx
Agent	4	2.3	31.4	0.6	7.9
Atom Task (without Template)	3	8.4 3.1 11.4	41.8 16.4 29.4	7.6	29.2
Atom Task (with Template)	3	5.1 2.4 3.0	40.0 15.4 25.0	3.5	26.8
Compositional Scenario Task	3	1.4 2.3 1.3	21.8 18.8 31.9	1.6	24.2
Others (Other GUI Actions)	—	6.3	36.4	—	—
Total		**46.9**	**308.1**		

- Agent internal behavior was modeled 13 times faster by `SNIFF` than by `SpaceEx`. `SNIFF`'s agent-instantiation mechanism allows quick updates to agent internal behavior, while `SpaceEx` requires additional components to be built.
- For both tools, modeling system behaviors in atom tasks took the longest. On average, it took 7.6 min with `SNIFF` when no task template was given, while nearly half an hour with `SpaceEx`. It indicates that communication configurations in `SNIFF` were helpful in defining agent composition in atom tasks.
- With task templates, atom tasks' modeling time was reduced by half in `SNIFF`, from 7.6 min to 3.5 min. `SNIFF`'s support for task templates makes agent composition more convenient.
- To describe topologies in compositional scenario tasks, it averagely took 1.6 min in `SNIFF`, but 24.2 min in `SpaceEx`. Dynamic topologies can be set quickly by `SNIFF`'s scenario task configurations, but challenging in `SpaceEx`.

The Answer to RQ1: `SNIFF`'s template models, together with its flexible instantiation and configuration mechanisms, simplify the modeling of reconfigurable CPS, saving considerable modeling time and effort for users.

Accuracy of Modeling. Among the models submitted by participants, 70.27% of `SNIFF` models were accurate, while only 7.21% of `SpaceEx` models were accurate. We analyzed all submitted modeling errors and classified them into four major types in Table 3. It lists their sources and probabilities, which are counted per source to eliminate the influence of the number of sources.

- Compared to `SpaceEx`, `SNIFF` reduces all types of modeling errors significantly.
- In `SpaceEx`, more than half (59.46%) of system behaviors with parallel topology shifts were incorrectly modeled, whereas `SNIFF`'s scenario task configurations simplify this modeling process and eliminate such errors completely.
- In `SNIFF`, modeling agents' compositions has the highest error rate (6.91%). Our additional statistics found that only 13% of these errors occur when task templates are used, indicating that task templates can help reduce such errors.

Table 3. Probabilities of Different Modeling Errors in SNIFF and SpaceEx

Type of Errors	Source of Errors	Probability (per Source)	
		SNIFF	SpaceEx
Agent Internal Behavior Error	Agent	1.13%	15.32%
Agents' Composition Error	Atom Task	**6.91**%	38.89%
Sequential Topology Shift Error	Sequential Compositional Task	0.90%	17.12%
Parallel Topology Shift Error	Parallel Compositional Task	0.00%	**59.46**%
Others[1]	—	1.80%	29.73%

[1] This includes misnamed, incomplete and undefined parameters, agents, tasks, etc.

The Answer to RQ2: SNIFF significantly improves the accuracy of CPS models constructed by users, reducing all types of modeling errors.

Falsification Performance. Two example systems are falsified under various scenarios, ranging from the simplest to the most complex. Table 4 lists the size of each scenario task, including the number of atom tasks. Scenarios within 20 s are highlighted in green, others in red. We listed the falsification performance of SNIFF and S-TaLiRo for each specification, including success rates, and costs of time and simulation iterations to find a witness. We also calculated the average falsification time for each atom task, presented in grey.

Generally, SNIFF supports falsification for models with richer semantics and can handle a wider range of specifications. The multi-UAV system follows deterministic semantics, but the assembly system follows nondeterministic semantics, which is not specifically supported in S-TaLiRo. Out of the total thirty specifications, six are unsupported in S-TaLiRo and are underlined in the table.

- Effectiveness: On average, SNIFF's success rate is 99.7%, whereas S-TaLiRo is below 45%, indicating that our path-oriented falsification approach helps explore search space effectively. Especially, SNIFF's success rate is at least 97% in all cases, while S-TaLiRo achieved this in only two cases.
- Efficiency: SNIFF is more efficient than S-TaLiRo. It takes 0.1–39.2 s to falsify these specifications, while S-TaLiRo takes an average of 730.9 s, more than 12 min. Also, in the 16 cases that S-TaLiRo supports, SNIFF averagely requires only one-fifth as many simulation iterations as S-TaLiRo.
- Scalability: As the scenario grows in size and complexity, system behaviors become more intricate, and falsification inevitably becomes challenging and time-consuming. The table shows that S-TaLiRo's success rate decreases greatly in complex scenarios, whereas SNIFF's success rate barely drops. The rise in SNIFF's time-cost and simulation iterations is also reasonable.
- Runtime Performance: CPSs may need online modeling and analysis due to runtime reconfiguration [13]. To evaluate our performance for runtime atom task scenarios, we roughly divide the falsification time by the number of atom tasks, as highlighted in grey. It indicates that SNIFF can handle such scenarios in around 1.7 s on average. Since automatic modeling in our language

Table 4. Falsification Results of Example Systems in SNIFF and S-TaLiRo

System	Scenario Task				Spec[2]	SNIFF				S-TaLiRo[3]			
	name	size[1]				success rate	time (s)	time per AT (s)	sim-iter	success rate	time (s)	time per AT (s)	sim-iter
		s_1	s_2	s_3									
Multi -UAV System	treeL1	1	4	16	s1	100%	0.7	0.7	31	100%	20.1	20.1	179
					s2	100%	2.2	2.2	110	100%	33.9	33.9	179
					s3*	100%	0.5	0.5	25	91%	211.1	211.1	653
					s4*	100%	0.9	0.9	47	78%	207.1	207.1	588
	tree	4	8	32	s5	100%	1.8	0.5	42	40%	776.0	194.0	373
					s6	100%	2.3	0.6	51	62%	492.3	123.1	276
					s7*	100%	2.1	0.5	22	45%	905.7	226.4	308
					s8*	100%	2.1	0.5	22	79%	728.7	182.2	348
	mission	8	8	32	s9	99%	25.3	3.2	55	0%	#N/A	#N/A	#N/A
					s10	97%	39.2	4.9	120	12%	1330.3	166.3	341
					s11	98%	38.2	4.8	87	3%	989.1	123.6	333
					s12	100%	34.6	4.3	129	43%	1154.9	144.4	336
					s13*	100%	23.1	2.9	130	1%	1532.5	191.6	346
					s14*	100%	17.7	2.2	65	34%	1269.3	158.7	350
					s15*	100%	30.2	3.8	111	4%	1220.9	152.6	339
					s16*	100%	37.9	4.7	83	24%	91.8	11.5	202
					s17	100%	10.1	1.3	73	/	/	/	/
					s18	100%	15.3	1.9	68	/	/	/	/
					s19	100%	13.6	1.7	62	/	/	/	/
					s20	100%	28.9	3.6	108	/	/	/	/
Automatic Assyembly System	weld2	1	1	2	s21*	100%	0.1	0.1	1414	/	/	/	/
					s22*	100%	0.2	0.2	2123	/	/	/	/
	wRoof	2	2	4	s23	100%	0.8	0.4	3865	/	/	/	/
					s24*	100%	0.6	0.3	2799	/	/	/	/
	convey -Grasp	4	7	14	s25*	99%	1.9	0.5	2335	/	/	/	/
					s26*	99%	2.7	0.7	2172	/	/	/	/
	assembly	15	13	26	s27	100%	14.2	0.9	1260	/	/	/	/
					s28*	100%	13.9	0.9	1219	/	/	/	/
					s29*	100%	18.3	1.2	1635	/	/	/	/
					s30*	98%	16.3	1.1	1803	/	/	/	/
Average						99.7%	13.2	1.7	736	44.8%	730.9	143.1	343

1 This column indicates the complexity of scenarios. s_1, s_2, and s_3 respectively denote the number of involved atom tasks, involved agents, and uncertain variables/external inputs.
2 Asterisked specifications are topology-related and are indirectly supported by S-TaLiRo. Underlined specifications involve nonlinear arithmetic that are unsupported by S-TaLiRo.
3 '#N/A' denotes unknown costs to falsify when a case hasn't been successfully falsified. '/' denotes unavailable falsification performance due to an unsupported model or specification.

barely costs time, we can handle the entire online configuring, modeling, and falsification process in seconds, improving system runtime safety.

The Answer to RQ3: SNIFF is highly effective, efficient, and scalable for temporal logic falsification in CPSs. It also demonstrates satisfactory runtime performance, handling runtime scenarios with a single atom task in seconds.

5.3 Threats to Validity

In this section, we discuss possible threats to the validity of our user study.

Internal Validity. is the assessment of modeling accuracy. In our user study, to evaluate modeling accuracy accurately, we restricted names of agents and tasks in `SNIFF` and components in `SpaceEx`. In this way, `SNIFF`/`SpaceEx` models could be automatically checked by examining their scripts. Another threat is that participants' familiarity with the target CPS could affect modeling. After collecting participants' modeling logs (automatically logged by both tools), we examined the order in which participants used the tools. It shows that 60% of participants used `SNIFF` first, and 84% of participants completed modeling in `SNIFF` first. This may introduce bias into our modeling evaluation.

External Validity. The main external threat arises from the representative of subjects of our user study. Formal modeling is typically conducted by engineers with relevant expertise. To conduct the user study with representative subjects, we recruited 111 participants from a course called Formal Languages and Automata, ensuring that they had a basic understanding of modeling and were close to real-world industry target users. Another threat is that our conclusions might not generalize to all reconfigurable CPS. We alleviate the threat by deriving the Multi-UAV system from a real-world case. Results from real-world CPS will help us evaluate our approach better. Additionally, we plan to conduct larger-scale experiments on more real-world reconfigurable CPSs in the future.

6 Related Work

Formalizing CPS is complex due to its hybrid and compositional nature. Formal models, such as hybrid automata [38,45] and hybrid process algebras [12,22], are used to formalize the tangled continuous and discrete behaviors in CPS. Further, compositional hybrid models formalize the behavior of multiple agents, by composing hybrid automata [6,45], processes [15], etc. Based on these formalisms, various modeling languages have been developed for multi-agent systems, including Unified Modeling Language (UML) based notations such as AUML [40], Constraint Logic Programming (CLP) notations [50,51], and Verse [43], a Python library for systems with multiple agents moving on a map, etc. However, for reconfigurable multi-agent CPS whose system topologies change dynamically at runtime [37,59,62], these models and notations do not provide sufficient mechanisms to support such flexibility. Although there have been works on modeling reconfigurable systems, they focus mainly on general architectural modeling [10,17,58]. In contrast, this work aims to provide a behavior modeling formalism that can effectively capture the reconfigurability of these systems.

CPS falsification is an effective way to detect system behaviors that violate specifications. Many falsification approaches search for witness behaviors by applying optimization algorithms to minimize robustness, including optimistic optimization [65,66], Bayesian optimization [25,61], classical heuristic-based methods [7,49,52,57], etc. Our falsification approach also falls under this category. Its most closely related work is [63], which also applies classification-model-based optimization [39,64]. However, it is limited to the safety falsification of a single agent. In contrast, we support broader temporal logic falsification for CPSs with multiple dynamically wired compositional agents. Apart from optimization-based ones, there are also other falsification approaches, such as motion-planning-based falsification [24,31,53], gradient-based falsification [14], and learning-based falsification [3].

7 Conclusion and Future Work

We present a hierarchical formal modeling language for reconfigurable CPSs. It provides templates for agent inherent features and allows for the formalization of reconfigurable CPS through agent instantiations, agents' communication configurations, and scenario task configurations. While it doesn't add expressivity beyond HA, its flexible hierarchy can simplify the complex modeling process, saving time and improving accuracy. Our future plans involve providing support for more CPS architectures and system topology variations.

We also propose a path-oriented falsification approach for reconfigurable CPSs. It employs a two-layered optimization approach to explore the search space effectively and cuts unnecessary calculations to improve efficiency. Experiments have shown that it can perform scalable falsification with high success rates. In the future, we plan to further enhance its runtime performance by leveraging offline and past online analysis results.

References

1. Acharya, S., Bharadwaj, A., Simmhan, Y., Gopalan, A., Parag, P., Tyagi, H.: Cornet: A co-simulation middleware for robot networks. In: 2020 International Conference on COMmunication Systems & NETworkS (COMSNETS), pp. 245–251. IEEE (2020)
2. Ahmadi, A., Moradi, M., Cherifi, C., Cheutet, V., Ouzrout, Y.: Wireless connectivity of cps for smart manufacturing: a survey. In: 2018 12th International Conference on Software, Knowledge, Information Management & Applications (SKIMA), pp. 1–8. IEEE (2018)
3. Akazaki, T., Liu, S., Yamagata, Y., Duan, Y., Hao, J.: Falsification of cyber-physical systems using deep reinforcement learning. In: Havelund, K., Peleska, J., Roscoe, B., de Vink, E. (eds.) FM 2018. LNCS, vol. 10951, pp. 456–465. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95582-7_27
4. Alur, R.: Principles of cyber-physical systems. MIT press (2015)
5. Alur, R., et al.: The algorithmic analysis of hybrid systems. Theoret. Comput. Sci. **138**(1), 3–34 (1995)
6. Alur, R., Grosu, R., Lee, I., Sokolsky, O.: Compositional modeling and refinement for hierarchical hybrid systems. J. Logic Algebraic Program. **68**(1–2), 105–128 (2006)
7. Annapureddy, Y.S.R., Fainekos, G.E.: Ant colonies for temporal logic falsification of hybrid systems. In: IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society, pp. 91–96. IEEE (2010)

8. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: a tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 254–257. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_21
9. Audemard, G., Simon, L.: Glucose: a solver that predicts learnt clauses quality. SAT Competition pp. 7–8 (2009)
10. Bazydło, G.: Designing reconfigurable cyber-physical systems using unified modeling language. Energies **16**(3), 1273 (2023)
11. Bérard, B., et al.: Systems and software verification: model-checking techniques and tools. Springer Science & Business Media (2013)
12. Bergstra, J.A., Middelburg, C.A.: Process algebra for hybrid systems. Theoret. Comput. Sci. **335**(2-3), 215–280 (2005)
13. Bersani, M.M., García-Valls, M.: Online verification in cyber-physical systems: Practical bounds for meaningful temporal costs. J. Softw. Evolution Proc. **30**(3) (2018)
14. Bogomolov, S., Frehse, G., Gurung, A., Li, D., Martius, G., Ray, R.: Falsification of hybrid systems using symbolic reachability and trajectory splicing. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, pp. 1–10 (2019)
15. Brinksma, E., Krilaviĉius, T., Usenko, Y.S.: Process algebraic approach to hybrid systems. IFAC Proc. Vol. **38**(1), 325–330 (2005)
16. Bu, L., Li, Y., Wang, L., Li, X.: Bach: bounded reachability checker for linear hybrid automata. In: 2008 Formal Methods in Computer-Aided Design, pp. 1–4. IEEE (2008)
17. Cavalcante, E., Batista, T., Oquendo, F.: Supporting dynamic software architectures: From architectural description to implementation. In: 2015 12th Working IEEE/IFIP Conference on Software Architecture, pp. 31–40. IEEE (2015)
18. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: an analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 258–263. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_18
19. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R., et al.: Handbook of model checking, vol. 10. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8
20. Copty, F., et al.: Benefits of bounded model checking at an industrial setting. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 436–453. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44585-4_43
21. Cotton, S., Frehse, G., Lebeltel, O.: The spaceex modeling language. SpaceEx tool (2010)
22. Cuijpers, P.J.L., Reniers, M.A.: Hybrid process algebra. J. Logic Algebraic Program. **62**(2), 191–245 (2005)
23. Dafflon, B., Moalla, N., Ouzrout, Y.: The challenges, approaches, and used techniques of cps for manufacturing in industry 4.0: a literature review. Inter. J. Adv. Manufact. Technol. **113**, 2395–2412 (2021)
24. Dang, T., Nahhal, T.: Coverage-guided test generation for continuous and hybrid systems. Formal Methods Syst. Design **34**, 183–213 (2009)
25. Deshmukh, J., Horvat, M., Jin, X., Majumdar, R., Prabhu, V.S.: Testing cyber-physical systems through bayesian optimization. ACM Trans. Embedded Comput. Syst. (TECS) **16**(5s), 1–18 (2017)
26. Do, H.T., et al.: Formation control algorithms for multiple-uavs: a comprehensive survey. EAI Endorsed Trans. Indus. Netw. Intell. Syst. **8**(27), e3–e3 (2021)

27. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 167–170. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_17
28. Donzé, A., Frehse, G.: Modular, hierarchical models of control systems in spaceex. In: 2013 European Control Conference (ECC), pp. 4244–4251. IEEE (2013)
29. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 92–106. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15297-9_9
30. Doyen, L., Frehse, G., Pappas, G.J., Platzer, A.: Verification of hybrid systems. In: Handbook of Model Checking, pp. 1047–1110 (2018)
31. Dreossi, T., Dang, T., Donzé, A., Kapinski, J., Jin, X., Deshmukh, J.V.: Efficient guiding strategies for testing of temporal properties of hybrid systems. In: Havelund, K., Holzmann, G., Joshi, R. (eds.) NFM 2015. LNCS, vol. 9058, pp. 127–142. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17524-9_10
32. Du, Y., Lu, X., Wang, J., Chen, B., Tu, H., Lukic, S.: Dynamic microgrids in resilient distribution systems with reconfigurable cyber-physical networks. IEEE J. Emerging Selected Topics Power Electr. **9**(5), 5192–5205 (2020)
33. Ernst, G., Sedwards, S., Zhang, Z., Hasuo, I.: Fast falsification of hybrid systems using probabilistically adaptive input. In: Parker, D., Wolf, V. (eds.) QEST 2019. LNCS, vol. 11785, pp. 165–181. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30281-8_10
34. Fainekos, G.E., Sankaranarayanan, S., Ueda, K., Yazarel, H.: Verification of automotive control applications using s-taliro. In: 2012 American Control Conference (ACC), pp. 3567–3572. IEEE (2012)
35. Frehse, G.: Phaver: algorithmic verification of hybrid systems past hytech. Int. J. Softw. Tools Technol. Transfer **10**, 263–279 (2008)
36. Frehse, G., et al.: SpaceEx: scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_30
37. Gray, J., Rumpe, B.: Modeling dynamic structures (2020)
38. Henzinger, T.A.: The theory of hybrid automata. In: Proceedings 11th Annual IEEE Symposium on Logic in Computer Science, pp. 278–292. IEEE (1996)
39. Hu, Y.Q., Qian, H., Yu, Y.: Sequential classification-based optimization for direct policy search. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31 (2017)
40. Huget, M.P.: Agent uml notation for multiagent system design. IEEE Internet Comput. **8**(4), 63–71 (2004)
41. Lee, E.A.: The past, present and future of cyber-physical systems: a focus on models. Sensors **15**(3), 4837–4869 (2015)
42. Lee, E.A., Seshia, S.A.: Introduction to embedded systems: A cyber-physical systems approach. MIT press (2016)
43. Li, Y., Zhu, H., Braught, K., Shen, K., Mitra, S.: Verse: a python library for reasoning about multi-agent hybrid system scenarios. In: International Conference on Computer Aided Verification, pp. 351–364. Springer (2023). https://doi.org/10.1007/978-3-031-37706-8_18
44. Lygeros, J., Johansson, K.H., Simic, S.N., Zhang, J., Sastry, S.S.: Dynamical properties of hybrid automata. IEEE Trans. Autom. Control **48**(1), 2–17 (2003)
45. Lynch, N., Segala, R., Vaandrager, F.: Hybrid i/o automata. Inform. Comput. **185**(1), 105–157 (2003)

46. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT -2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30206-3_12
47. Maler, O., Nickovic, D., Pnueli, A.: Checking temporal properties of discrete, timed and continuous behaviors. Pillars of Computer Science: Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday, pp. 475–505 (2008)
48. Mathesen, L., Pedrielli, G., Ng, S.H., Zabinsky, Z.B.: Stochastic optimization with adaptive restart: a framework for integrated local and global learning. J. Global Optim. **79**, 87–110 (2021)
49. Mathesen, L., Yaghoubi, S., Pedrielli, G., Fainekos, G.: Falsification of cyber-physical systems with robustness uncertainty quantification through stochastic optimization with adaptive restart. In: 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE), pp. 991–997. IEEE (2019)
50. Mohammed, A., Furbach, U.: Multi-agent systems: modeling and verification using hybrid automata. In: Braubach, L., Briot, J.-P., Thangarajah, J. (eds.) ProMAS 2009. LNCS (LNAI), vol. 5919, pp. 49–66. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14843-9_4
51. Mohammed, A., Stolzenburg, F.: Implementing hierarchical hybrid automata using constraint logic programming. In: Proceedings of 22nd Workshop on (Constraint) Logic Programming, Dresden, pp. 60–71 (2008)
52. Nghiem, T., Sankaranarayanan, S., Fainekos, G., Ivancić, F., Gupta, A., Pappas, G.J.: Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In: Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, pp. 211–220 (2010)
53. Plaku, E., Kavraki, L.E., Vardi, M.Y.: Hybrid systems: from verification to falsification by combining motion planning and discrete search. Formal Methods Syst. Design **34**(2), 157–182 (2009)
54. Prajna, S., Jadbabaie, A.: Safety verification of hybrid systems using barrier certificates. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 477–492. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24743-2_32
55. Prajna, S., Jadbabaie, A., Pappas, G.J.: A framework for worst-case and stochastic safety verification using barrier certificates. IEEE Trans. Autom. Control **52**(8), 1415–1428 (2007)
56. Rosu, G., Havelund, K.: Synthesizing dynamic programming algorithms from linear temporal logic formulae (2001)
57. Sankaranarayanan, S., Fainekos, G.: Falsification of temporal properties of hybrid systems using the cross-entropy method. In: Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control, pp. 125–134 (2012)
58. Selić, B.: Specifying dynamic software system architectures. Softw. Syst. Model. **20**(3), 595–605 (2021)
59. Sha, L., Gopalakrishnan, S., Liu, X., Wang, Q.: Cyber-physical systems: A new frontier. In: 2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (Sutc 2008), pp. 1–9. IEEE (2008)
60. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 244–257. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02777-2_24
61. Waga, M.: Falsification of cyber-physical systems with robustness-guided black-box checking. In: Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control, pp. 1–13 (2020)

62. Wan, K., Hughes, D., Man, K.L., Krilavicius, T., Zou, S.: Investigation on composition mechanisms for cyber physical systems. Inter. J. Design, Anal. Tools Integrated Circ. Syst. **2**(1), 30 (2011)
63. Wang, J., Bu, L., Xing, S., Li, X.: Path-oriented, derivative-free approach for safety falsification of nonlinear and nondeterministic cps. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **41**(2), 238–251 (2021)
64. Yu, Y., Qian, H., Hu, Y.Q.: Derivative-free optimization via classification. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30 (2016)
65. Zhang, Z., Ernst, G., Sedwards, S., Arcaini, P., Hasuo, I.: Two-layered falsification of hybrid systems guided by monte carlo tree search. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **37**(11), 2894–2905 (2018)
66. Zhang, Z., Lyu, D., Arcaini, P., Ma, L., Hasuo, I., Zhao, J.: Effective hybrid system falsification using monte carlo tree search guided by QB-robustness. In: Silva, A., Leino, K.R.M. (eds.) CAV 2021. LNCS, vol. 12759, pp. 595–618. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81685-8_29