Technical Report No. NJU-SEG-2019-IJ-002

2019-IJ-002

Scenario-based Online Reachability Validation For CPS Fault Prediction

Lei Bu, Qixin Wang, Xinyue Ren, Shaopeng Xing, Xuandong Li

IEEE Transactions on Computer-Aided Design of Integrated Circuits and System 2019

Scenario-based Online Reachability Validation For CPS Fault Prediction

Lei Bu, *Member, IEEE,* Qixin Wang, *Member, IEEE,* Xinyue Ren, Shaopeng Xing, and Xuandong Li

***Abstract*—Unlike standalone embedded devices, behaviors of a Cyber-Physical System (CPS) are highly dynamic. Many parameter values (e.g. those related to nature environment and third party black box functions) are unknown offline. Furthermore, distributed sub-CPSs may exchange data online. In this paper, we first propose the concept of *parametric hybrid automata* (PHA) to describe such complex CPSs. As some PHA parameter values are unknown until runtime, conventional offline model checking is infeasible. Instead, we propose to carry out PHA model checking online, as a fault prediction mechanism. However, this usage is challenged by the high time cost of state reachability verification, which is the conventional focus of model checking. To address this challenge, we propose that the model checking shall focus on *online scenario reachability validation* instead. Furthermore, we propose a mechanism to compose/decompose scenarios. Our scenario reachability validation can exploit linear programming to achieve polynomial time cost. Evaluations on a state-of-the-art train control system show that our approach can cut online model checking time cost from over 1 hour to within 200 milliseconds.**

***Index Terms*—Linear Hybrid Automata, Scenario Reachability Validation, Online Modeling and Verification, CBTC based Train Control CPS**

I. Introduction

The inevitable convergence of computers and physical world results in the booming of *Cyber-Physical Systems* (CPSs) [41]. Many CPSs are mission/life critical (e.g. modern train control systems, see Section II-A), hence safety is a top concern [26] [50]. To increase safety, state reachability verification model checking [25] before deployment is widely adopted [26]. Conventional state reachability verification model checking builds an *offline* formal model of a system during development stage and verify if a set of predefined unsafe-states are reachable from initial state(s). If the model is fully accurate and unsafe-states are verified unreachable, then the system is considered safe.

Though state reachability model checking is proven successful for many applications (e.g. hardware design), its application to CPSs still faces major challenges [41] [26].

Manuscript received September 20, 2018. The authors in Nanjing University are supported in part by the National Key Research and Development Plan (No. 2017YFA0700604), and the National Natural Science Foundation of China (No.61632015, No.61572249, 61561146394). The author in Hong Kong Polytechnic University is supported in part by the Hong Kong RGC GRF PolyU 152002/18E, PolyU 152164/14E, RGC Germany/HK Joint Research Scheme G-PolyU503/16, The Hong Kong Polytechnic Univ. fund P0009706 (G-YBXW), P0009367 (G-YBMW), P0010792 (G-YN37), P0000157 (1-BBWC), P0013879 (1-BBWH), P0001888 (4-ZZHD), and P0008501 (G-UA7L).

L. Bu, X. Ren, S. Xing and X. Li are are with State Key Laboratory of Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing, China. E-mail: `bulei@nju.edu.cn`

Q. Wang is with Department of Computing, The Hong Kong Polytechnic University, Hong Kong. E-mail: `csqwang@comp.polyu.edu.hk`

Challenge 1: Due to the coexistence of discrete and continuous dynamics in CPSs, hybrid automaton [34] becomes the de facto standard (and often inevitable) modeling tool for CPSs. However, as CPS behavior is highly dynamic, some parameters of the hybrid automaton model cannot be accurately predicted offline [19]. This makes offline system modeling difficult, even impractical. Take the train control system of Section II-A for example, many key parameter values (e.g. wind velocity, rail conditions etc.) have to be sensed online. Some other parameter values are configured by third party black box functions online. There is no offline model to accurately predict these parameter values; even narrow bounds for these parameter values are hard to find. In addition, dynamics of a CPS can be further complicated by online data exchange between its distributed sub-CPSs, which are hard to be modeled as conventional automata events with no data payloads.

To deal with **Challenge 1**, this paper proposes *parametric hybrid automata* (PHA) to model CPSs with unknown parameter values offline and complex online data exchanges (see Section II). PHAs can be composed to describe large complex CPSs. *Offline* model checking of PHAs is infeasible due to the value-unknown parameters. To address this problem, a natural strategy is to carry out *online* model checking [10], [19], [21], [42], [46] instead, i.e. to use model checking as an online fault prediction mechanism. The basic idea is as follows: during runtime, we periodically sense/collect the values of all related CPS parameters (the period is thus called the "*online model checking period*"), and *concretize* the PHAs into conventional hybrid automata [34]. We then carry out model checking of the updated model to predict if the CPS can reach any unsafe states in future (i.e. carry out state reachability verification model checking). If so, an alarm is raised to trigger an application-dependent fall-back plan. If not, we run the CPS till the next online model checking period starts. In each online model checking period, if the model checking cannot finish within a short deadline, to play safe, we always raise an (possibly false) alarm to trigger the fall-back plan anyways [42]. In this way, no major system failure can happen in case of a deadline miss. That said, we still need to speed up the online model checking. This is due to two reasons:

D1 Frequent deadline misses cause frequent false alarms and triggering of the fall-back plan. This harms equipment and user experience in the long run. Furthermore, this fosters negligence among human operators, which can lead to critical failures. In order to reduce deadline misses, we need faster online model checking.

D2 Even if deadline misses are eliminated, faster online model checking is always preferred, as it can raise a (positive) alarm sooner, hence win more time to carry out the fall-back plan.

However, speeding up the online model checking faces another challenge.

Challenge 2: It is well known that hybrid automata state reachability verification (i.e. given hybrid automata and initial state(s), check if certain state(s) is/are reachable) is very difficult and time-costly [34]. In fact, although given a rectangular hybrid automaton with special restrictions, the state reachability problem is decidable with PSPACE, if we relax any restriction, the problem will be undecidable [31], [32], [35], [36]. One main reason for this is *state explosion* [51]: hybrid automata state space can easily explode due to the tangling of discrete and continuous behavior, and the increasing of number of CPS subsystems. Again take the train control systems of Section II-A for example, a metropolitan subway can often run a dozen of trains on a single track in parallel. To model such a composed CPS with conventional approach, a Cartesian product of at least a dozen of hybrid automata is needed, which implies at least 6^{12} work modes of the holistic system model (see Fig. 2 (A), each train has at least 6 work modes): an astronomical scale that is computationally inhibiting.

To address **Challenge 2**, we propose *online scenario reachability validation* model checking of PHA to replace state reachability verification model checking. This proposal is based on the empirical needs of industry. In practice, industry knows **Challenge 2** for a long time, hence does not insist on discovering all reachable unsafe states (i.e. completeness of model checking) in a CPS. Instead, very often, industry only demands to know if the CPS can reach certain unsafe states in certain work mode via certain sequence of actions. That is, if certain "*scenario*" can happen [45]. Our proposal formalizes this demand as the online scenario reachability validation model checking. Furthermore, we propose a mechanism to compose/decompose scenarios. When the PHA is linear, online scenario reachability model checking can finish within polynomial time using *linear programming* (LP).

We evaluated our proposal upon a state-of-the-art *communication based train control* (CBTC) system, a typical life/mission critical CPS. The results show that our proposal can shrink online model checking time cost from over 1 hour to within 200 milliseconds.

In summary, the contributions of this paper include:

1) We propose a novel approach based on the notion of "PHA execution trace" to rigorously define the PHA modeling language, which supports modeling offline value-unknown parameters, and online data exchange.
2) Also, using the "PHA execution trace" notion, we rigorously redefine the scenario reachability verification.
3) We propose for online verification, we should conduct scenario reachability instead of state reachability due to the computational efficiency.
4) We conduct comprehensive study on a real-case CBTC system to show the expressability of PHA, and the applicability of the scenario reachability online verification.

The rest of the paper is organized as follows. Section II proposes our parametric hybrid automata design together with our motivating application. Section III proposes our scenario reachability validation framework. Section IV evaluates our methods. Section V discusses related work. Section VI concludes the paper.

II. Parametric Hybrid Automata

In this section, we first introduce a motivating example: a typical CPS, where **Challenge 1** exists. We then propose the formal definition of parametric hybrid automata to model such CPSs.

A. Motivating Application: CBTC

Modern train control systems are typical CPSs and they play key roles to the safety and efficiency of railway systems. Several train control systems exist nowdays [33] [47]. Among these, *communication based train control system* (CBTC) is arguably the most advanced, and is still evolving. It is widely adopted by many subway systems and train systems, and is a candidate solution for China's latest high speed railway systems [47].

Without loss of generality, in the following we describe a CBTC system developed for an urban railway system in China. The CBTC uses data communications between trains and various control facilities to guarantee the safety and efficiency of operations. It consists of two main parts: *ground systems* and *onboard systems*. The ground systems' *radio block centers* (RBCs) periodically poll the runtime state/context parameter values of all running trains in the CBTC. The RBCs then send control parameters, particularly, *movement authorities* (MAs), to the onboard systems of respective trains. The MA specifies an *end-of-authority* (EOA) point on the track ahead of the train [49] [27], along with other parameter values. Based on the received MA, EOA, and other current state/context parameters, each onboard system plans future train movements. Particularly, the onboard system plans/updates *legal operating velocity ranges* for various work modes and a *safe braking point* (SBP) ahead of the train on the track. The train is free to move within the planned legal operation velocity ranges under corresponding work modes before reaching the SBP. Once the train reaches the SBP, it shall immediately apply *standard braking* (SBraking), and is supposed to reach a full stop in 50 seconds before hitting the EOA point. A train must communicate with its RBC every 0.5 seconds to update the MA (includig the EOA), and then update the legal operating velocity ranges and SBP. If the train runs for 5 consecutive seconds without receiving any signals from the RBC, then it shall assume communication failure and start *emergency braking* (EBraking). The emergency braking is supposed to reach a full stop in 20 seconds. Fig. 1 illustrates the above CBTC concepts.

However, the software modules to compute the MAs, EOAs, legal operation velocity ranges, and SBPs are third party black box modules. Their correctness is not fully dependable. To add redundancy for better dependability, regulations require our CBTC system to monitor two key safety rules.

R1 During EBraking, a train must not reach the train ahead of it on the track.

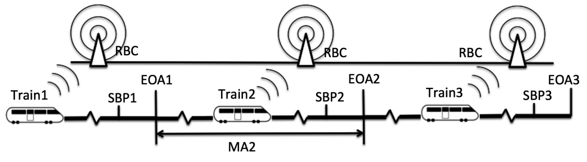


Fig. 1. Illustration of a CBTC System

R2 During SBraking, a train must not exceed its EOA.

To check if the CBTC system ensures **R1** and **R2**, we need a formal model of the CBTC. However, many context parameter values of the CBTC are unavailable offline: this includes the black box third party software modules' outputs, and context parameters such as wind speed and railway conditions (e.g. the railway condition may change due to rain, snow, or ice). In other words, we face **Challenge 1** described in Section I.

B. Parametric Hybrid Automata Formal Definition

To address **Challenge 1** in CBTC modeling, we propose the concept of *parametric hybrid automata* (PHA), where offline value-unknown context parameters are modeled as variables. Specifically, given a global set of real value *state parameter* variables $\mathcal{X}$, a global set of real value *context parameter* variables $\mathcal{P}$, where

$$\mathcal{P} \cap \mathcal{X} = \varnothing, \tag{1}$$

a global set of *event labels* $\mathcal{L}$, and a function $S : \mathcal{L} \mapsto 2^{\mathcal{X} \cup \mathcal{P}}$, where $S(l) \subseteq \mathcal{X} \cup \mathcal{P}$ ($\forall l \in \mathcal{L}$) specifies the set of state and context parameter variables *shared* (e.g. by data communications) by an event labeled l, we define the following.

Definition 1: A *parametric hybrid automaton* (PHA) is a tuple $H = (X,\, P,\, L,\, V,\, V^0,\, E,\, \alpha,\, \beta,\, \gamma)$, where

1) $X \subseteq \mathcal{X}$ and $P \subseteq \mathcal{P}$ are two finite set of variables respectively representing the *state parameters* and *context parameters* of H. Context parameters' dynamics are controlled by external entities (such as nature environment, or third party black box functions). Their values can be sampled online but are unknown offline. Given context parameters values, state parameters' dynamics are determined.
2) V is a finite set of *locations*; $V^0 \subseteq V$ is the set of *initial locations*.
3) E is a finite set of *events*, whose elements (i.e. events) are of the form (v, l, ϕ, ψ, v'), where
   a) $v, v' \in V$ are respectively the *source* and *destination* locations for this event.
   b) $l \in \mathcal{L}$ is the label for the event. If l is used by this PHA alone, the event is called a *local* event. Otherwise, the event is a *shared* event.
   c) ϕ is a finite set of *guards*. In case $\phi \neq \varnothing$, the ith ($i = 1$, 2, …) element of ϕ is of the form $f_{\phi,i}(Y_{\phi,i}) \leqslant a_{\phi,i}$, where $Y_{\phi,i} \subseteq X \cup P$ is the set of state or context parameter variables involved in the guard, $a_{\phi,i} \in \mathbb{R}$ is a constant[1]. ϕ are conditions that must all sustain to trigger the event.
   d) If the event is a shared event and $\phi = \varnothing$, i.e. the event cannot be triggered locally, then the event is called a *received* event. A received event must have $S(l) \cap (X \cup P) = \varnothing$. If the event is a shared event but not a received event, the it is called a *sent* event (when a sent event happens, $S(l) \cap (X \cup P)$ are the data sent to other PHAs in the system via matching received event(s), see Def. 1.3e and Def. 2).
   e) ψ is a finite set of *resets*. In case $\psi \neq \varnothing$, the ith ($i = 1,\ 2,\ \ldots$) element of ψ is of the form $x_{\psi,i} := f_{\psi,i}(Y_{\psi,i})$, where $x_{\psi,i} \in X \cup P$, and $\begin{cases} Y_{\psi,i} \subseteq X \cup P \cup S(l) & \text{for a received event} \\ Y_{\psi,i} \subseteq X \cup P & \text{otherwise} \end{cases}$.
4) $L \overset{\text{def}}{=} \{l | \exists (v, l, \phi, \psi, v') \in E\}$.
5) α is a labeling function, which maps each location $v \in V$ to a *location invariant*, which is a set of *parameter constraints*. In case $\alpha(v) \neq \varnothing$, the ith ($i = 1,\ 2,\ \ldots$) element of $\alpha(v)$ is of the form $f_{\alpha(v),i}(Y_{\alpha(v),i}) \leqslant a_{\alpha(v),i}$, where $Y_{\alpha(v),i} \subseteq X \cup P$, and $a_{\alpha(v),i} \in \mathbb{R}$.
6) β is a labeling function, which maps each location $v \in V$ to a set of *flow conditions*. In case $\beta(v) \neq \varnothing$, the ith ($i = 1,\ 2,\ \ldots$) element of $\beta(v)$ is of the form $\dot{x}_{\beta(v),i} = f_{\beta(v),i}(Y_{\beta(v),i})$, where $x_{\beta(v),i} \in X$, $Y_{\beta(v),i} \subseteq X \cup P$, and $f_{\beta(v),i}$ is a formula involving element(s) of $Y_{\beta(v),i}$. When $Y_{\beta(v),i} = \varnothing$, $f_{\beta(v),i}(Y_{\beta(v),i})$ can be either a constant in $\mathbb{R}$, or a range in $\mathbb{R}$. In the latter case (without loss of generality, suppose $f_{\beta(v),i}(\varnothing) = [a_{\beta(v),i}, b_{\beta(v),i}]$), we define $\dot{x}_{\beta(v),i} = [a_{\beta(v),i}, b_{\beta(v),i}]$ meaning $x_{\beta(v),i}$ is varying in a rate between $a_{\beta(v),i}$ and $b_{\beta(v),i}$. $\forall v \in V$, $\forall x \in X$, there is one and only one flow condition.
7) γ is a labeling function, which maps each location $v \in V^0$ to a set of *initial conditions*. In case $\gamma(v) \neq \varnothing$, the ith ($i = 1,\ 2,\ \ldots$) element of $\gamma(v)$ is of the form $x_{\gamma(v),i} := a_{\gamma(v),i}$, where $x_{\gamma(v),i} \in X$, $a_{\gamma(v),i}$ is either a real constant or a context parameter in P. $\forall v \in V^0$, $\forall x \in X$, there is at the most one initial condition. □

A group of PHAs can be composed to a more complex system. The *composition* is a PHA generated by synchronizing all the component PHAs based on shared events.

Definition 2: Let $H_1 = (X_1,\, P_1,\, L_1,\, V_1,\, V_1^0,\, E_1,\, \alpha_1,\, \beta_1,\, \gamma_1)$ and $H_2 = (X_2,\, P_2,\, L_2,\, V_2,\, V_2^0,\, E_2,\, \alpha_2,\, \beta_2,\, \gamma_2)$ be two PHAs, where

$$\begin{cases} X_1 \cap X_2 = \varnothing \\ P_1 \cap P_2 = \varnothing \\ X_1 \cap P_2 = \varnothing \\ X_2 \cap P_1 = \varnothing \end{cases}. \tag{2}$$

The *composition* of H_1 and H_2, denoted as $H_1 || H_2$, is a PHA $H = (X,\, P,\, L,\, V,\, V^0,\, E,\, \alpha,\, \beta,\, \gamma)$, where

1) $X = X_1 \cup X_2$ and $P = P_1 \cup P_2$.
2) $L = L_1 \cup L_2$.
3) $V = V_1 \times V_2$; $V^0 = V_1^0 \times V_2^0$.

[1]Unless otherwise denoted, in this paper, $a_\bullet$ and $b_\bullet$ represent real valued constants, and $c_\bullet$ represents real valued constant coefficients, where $\bullet$ is a subscript for identification purposes.

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.2019.2935062, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

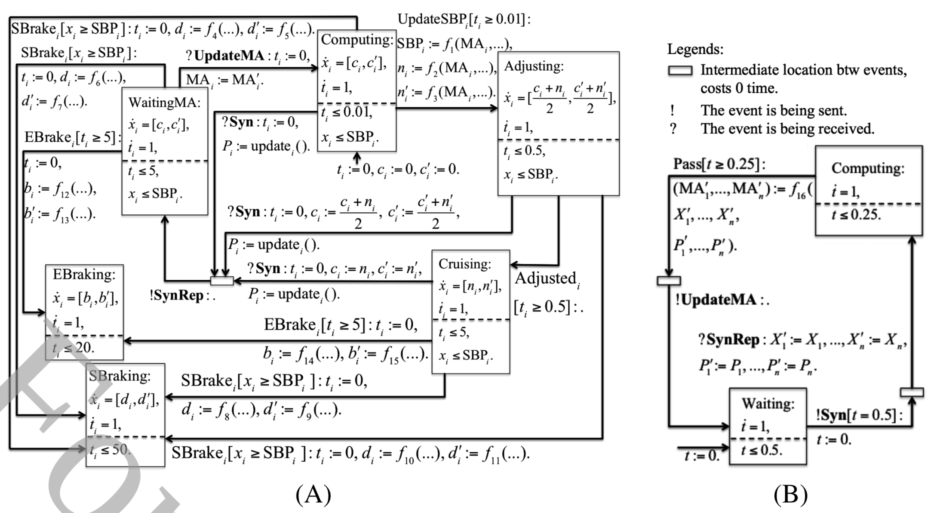

$\text{SBrake}_i[x_i \geq \text{SBP}_i]: t_i := 0, d_i := f_4(\ldots), d'_i := f_5(\ldots).$
$\text{SBrake}_i[x_i \geq \text{SBP}_i]:$ $t_i := 0, d_i := f_6(\ldots),$ $d'_i := f_7(\ldots).$
$?\textbf{UpdateMA}: t_i := 0,$ $\text{MA}_i := \text{MA}'_i.$
$\text{UpdateSBP}_i[t_i \geq 0.01]:$ $\text{SBP}_i := f_1(\text{MA}_i,\ldots),$ $n_i := f_2(\text{MA}_i,\ldots),$ $n'_i := f_3(\text{MA}_i,\ldots).$
Computing: $\dot{x}_i = [c_i, c'_i],$ $\dot{t}_i = 1,$ $t_i \leq 0.01,$ $x_i \leq \text{SBP}_i.$
Adjusting: $\dot{x}_i = [\frac{c_i+n_i}{2}, \frac{c'_i+n'_i}{2}],$ $\dot{t}_i = 1,$ $t_i \leq 0.5,$ $x_i \leq \text{SBP}_i.$
WaitingMA: $\dot{x}_i = [c_i, c'_i],$ $\dot{t}_i = 1,$ $t_i \leq 5,$ $x_i \leq \text{SBP}_i.$
$\text{EBrake}_i[t_i \geq 5]:$ $t_i := 0,$ $b_i := f_{12}(\ldots),$ $b'_i := f_{13}(\ldots).$
$?\textbf{Syn}: t_i := 0,$ $P_i := \text{update}_i().$
$t_i := 0, c_i := 0, c'_i := 0.$
$?\textbf{Syn}: t_i := 0, c_i := \frac{c_i+n_i}{2}, c'_i := \frac{c'_i+n'_i}{2},$ $P_i := \text{update}_i().$
$?\textbf{Syn}: t_i := 0, c_i := n_i, c'_i := n'_i,$ $P_i := \text{update}_i().$
$!\textbf{SynRep}: .$
Cruising: $\dot{x}_i = [n_i, n'_i],$ $\dot{t}_i = 1,$ $t_i \leq 5,$ $x_i \leq \text{SBP}_i.$
Adjusted_i $[t_i \geq 0.5]: .$
EBraking: $\dot{x}_i = [b_i, b'_i],$ $\dot{t}_i = 1,$ $t_i \leq 20.$
$\text{EBrake}_i[t_i \geq 5]: t_i := 0,$ $b_i := f_{14}(\ldots), b'_i := f_{15}(\ldots).$
SBraking: $\dot{x}_i = [d_i, d'_i],$ $\dot{t}_i = 1,$ $t_i \leq 50.$
$\text{SBrake}_i[x_i \geq \text{SBP}_i]: t_i := 0,$ $d_i := f_8(\ldots), d'_i := f_9(\ldots).$
$\text{SBrake}_i[x_i \geq \text{SBP}_i]: t_i := 0, d_i := f_{10}(\ldots), d'_i := f_{11}(\ldots).$
(A)
Legends:
Intermediate location btw events, costs 0 time.
! The event is being sent.
? The event is being received.
$\text{Pass}[t \geq 0.25]:$ $(\text{MA}'_1,\ldots,\text{MA}'_n) := f_{16}($ $X'_1,\ldots,X'_n,$ $P'_1,\ldots,P'_n).$
Computing: $\dot{t} = 1,$ $t \leq 0.25.$
$!\textbf{UpdateMA}: .$
$?\textbf{SynRep}: X'_1 := X_1, \ldots, X'_n := X_n,$ $P'_1 := P_1, \ldots, P'_n := P_n.$
Waiting: $\dot{t} = 1,$ $t \leq 0.5.$
$!\textbf{Syn}[t = 0.5]:$ $t := 0.$
$t := 0.$
(B)

Fig. 2. Parametric Hybrid Automata (PHA) Model of CBTC: (A) PHA for Train_i ($i = 1, \ldots, m$); (B) PHA for RBC. The state and context parameters (see Section II-B) of Train_i are respectively X_i and P_i. Each node (a.k.a. "location") represents a work mode. Inside of each work mode, above the dashed line are flow conditions, i.e. dynamics on how state parameters change; below the dashed line are location invariants, i.e. constraints that state parameters must satisfy. $x_i \in X_i$ represents the position of Train_i on track. "$\dot{x}_i = [a, b]$" means although the specific controller that drives the train is a third party black box, the velocity is controlled within range $[a, b]$. Each edge represents an event. Following each event's label, a "[●]" represents the event's guards, i.e. local conditions that trigger the event. Note the guards of event "!UpdateMA" and "!SynRep" are omitted. The two events indeed have guards, which are the stay duration limit of their source locations. ":=" means value assignment. $f_1(\bullet), \ldots, f_{16}(\bullet)$ are all third party black box functions. $\text{update}_i()$ updates all context parameter values in P_i (except MA_i, SBP_i, n_i, n'_i, b_i, b'_i, d_i, d'_i).

4) $\alpha((v_1, v_2)) = \alpha(v_1) \cup \alpha(v_2)$; $\beta((v_1, v_2)) = \beta(v_1) \cup \beta(v_2)$; $\gamma((v_1, v_2)) = \gamma(v_1) \cup \gamma(v_2)$.
5) E is defined as follows:
   a) for each $l \in L_1 \cap L_2$, for each $(v_1, l, \phi_1, \psi_1, v'_1) \in E_1$ and $(v_2, l, \phi_2, \psi_2, v'_2) \in E_2$, E contains $((v_1, v_2), l, \phi_1 \cup \phi_2, \psi_1 \cup \psi_2, (v'_1, v'_2))$;
   b) for each $l \in L_1 \setminus L_2$, for each $(v_1, l, \phi, \psi, v'_1) \in E_1$ and each $v_2 \in V_2$, E contains $((v_1, v_2), l, \phi, \psi, (v'_1, v_2))$;
   c) for each $l \in L_2 \setminus L_1$, for each $(v_2, l, \phi, \psi, v'_2) \in E_2$ and each $v_1 \in V_1$, E contains $((v_1, v_2), l, \phi, \psi, (v_1, v'_2))$.

For all $m > 2$, the *composition* of PHAs H_1, H_2, …, H_m, denoted as $H_1||H_2|| \ldots ||H_m$, is a PHA recursively defined as $H_1||H_2||\ldots||H_m = H_1||H'$, where $H' = H_2||H_3|| \ldots ||H_m$. □

With the above PHA definitions, we can build offline PHA models of the CBTC, which is shown in Fig. 2.

Our CBTC system includes m trains on track and many RBCs on ground. As all RBCs are connected by reliable backbone network and share information in real-time, we simplify them as one single RBC entity. For Train_i ($i = 1, 2, \ldots, m$) and the RBC, the PHAs are respectively shown in Fig. 2(A) and (B). According to the figure, the RBC polls all trains via a "Syn" event every 0.5 second, and Train_i replies its runtime state parameter values X_i (such as position, velocity, etc.) and context parameter values P_i (such as wind velocity, rail condition etc.) via a "SynRep" event. On receiving "SynRep", the RBC enters the "Computing" mode, where a third party black box function $f_{16}(\bullet)$ calculates a new MA for the train. The new MAs are sent to the trains by an "UpdateMA" event. Once a new MA is received, a train enters the "Computing" mode to compute a new SBP and a new legal operation velocity range $[n_i, n'_i]$ using third party black box functions $f_1(\bullet)$, $f_2(\bullet)$, and $f_3(\bullet)$. After that, the train enters the "Adjusting" mode to adjust its velocity from the current range $[c_i, c'_i]$ to the new range $[n_i, n'_i]$. After the velocity adjustment, the train enters the "Cruising" mode to cruise within the new legal operation velocity range. In "Computing", "Adjusting", "Cruising", and "WaitingMA" mode, if the train reaches its current SBP, it enters "SBraking" mode to apply standard braking, aiming to stop completely in 50 seconds, and to stop before the EOA. In "WaitingMA" or "Cruising" mode, if the train has not communicated with the RBC for 5 consecutive seconds, the train enters the "EBraking" mode to apply emergency braking, aiming to stop completely in 20 seconds.

In the above PHA model, many runtime context parameter values, such as those assigned by third party black box functions ($f_1(\bullet), \ldots, f_{16}(\bullet)$) and by nature environment (elements in P_i representing wind velocity etc.), are unavailable offline.

C. Limitations of State Reachability Model Checking

Due to the existence of value-unknown context parameters in PHA, offline model checking of PHA is impractical. However, during runtime, the values of PHA context parameters can be known. By replacing all context parameters with their concrete real values, the resulted PHA can be regarded as having empty context parameter set. We call such PHA a *concrete hybrid automaton* (HA), as defined in the following.

Definition 3: (Concrete PHA, LHA) Given a PHA $H = (X, P, L, V, V^0, E, \alpha, \beta, \gamma)$, if $P = \varnothing$, we say H is a *concrete hybrid automaton* (HA), and simplify it as $H = (X, L, V, V^0, E, \alpha, \beta, \gamma)$. Furthermore, for the above HA H, if

1) all its guards, location invariants, and initial conditions are linear inequalities of X;
2) each reset is of the form $x := f_\psi(Y)$ (where $Y \subseteq \mathcal{X}$; $f_\psi(Y)$ is a linear expression of Y);

3) and each flow condition is of the form $\dot{x} = [a, b]$ (a, b are constants in $\mathbb{R}$ and $a \leqslant b$),

then H is a *linear hybrid automaton* (LHA). [2] □

Therefore, instead of model checking offline PHAs, we shall model check concretized PHAs (i.e. HAs or LHAs) online. As pointed out by Section I, online model checking is indeed an online fault prediction mechanism, and is proposed by several prior works [19] [42] [10]. Conventional online model checking focuses on *state reachability verification*, i.e. given initial conditions, whether the state parameter values of the online model can reach certain unwanted regions in the state space, a.k.a. *unsafe-states*. Fig. 3 summarizes this idea. The algorithm is called periodically every T seconds, where T is the *online model checking period*. At the beginning of each online model checking period (denoted as t_0), all context parameter values are determined, so that a concrete hybrid automata model A of the CPS is built (see line 3 of Fig. 3). We then model check if A can reach unsafe-states in the next F seconds, where $[t_0, t_0 + F]$ is the *online model checking time horizon*. According to [42], the setting of F is application dependent, and is given by domain experts. Also according to [42], we shall set *online model checking relative deadline* (see line 1 and 5 of Fig. 3) $D = T/2$. Thus, we can use two parallel online fault prediction systems phased at $T/2$ to fully cover the future time horizon. Ideally, we shall set T and D as small as possible, to make the online model A more up-to-date, and to predict fault faster. However, the choice of T and D are lower bounded by the context parameter value updating time cost and the online model checking time cost.

```
   // online model checking called every $T$ seconds.
1. OnlineFaultPrediction(deadline $D$) **begin**
2.     $t_0$ set to the current time;
3.     update the online system model $A$;
4.     **if** (($A$ can reach unsafe-states in $[t_0, t_0 + F]$)
5.         **or** (current time $t \geqslant t_0 + D$)) **then**
6.         trigger the fall-back plan; //*non-blocking call.*
7. **end**;
```

Fig. 3. Pseudo code of online state reachability verification model checking.

In our CBTC example, we can call the online checking algorithm of Fig. 3 every time the RBC enters the "Computing" work mode, and set the deadline right before the broadcast of "UpdateMA" event (see Fig. 2). That is, online checking period $T = 0.5$ seconds, and $D = 0.25$ seconds. For online checking of safety rule **R1**, our domain experts set online checking time horizon length F to 25 seconds; while for **R2**, F is set to 50 seconds. If the online model checking confirms reachability to unsafe-states, or a deadline is missed, the RBC shall cancel the broadcast of "UpdateMA", and trigger a fall-back plan.

[2]Note the HA and LHA concepts defined here are indeed special cases of the more generic hybrid automata and LHA concepts proposed in [34]. Particularly, compared to generic HA and LHA, in our HA and LHA definition, external state parameters (i.e. those in $\mathcal{X} \setminus X$) can only be used in resets of receiver side shared events, also Formula (1)(2) sets several isolation constraints. We need these constraints to enforce balanced coupling and encapsulations between PHAs. In the following, unless otherwise denoted, HA and LHA refer to the concepts given in Def. 3.

According to Fig. 3, a deadline miss only results in inconvenience, i.e. triggering of the fall-back plan, instead of a disaster. Nevertheless, as pointed out by **D1** and **D2** in Section I, we still want to speed up the online model checking based fault prediction, so as to improve user experience by reducing unnecessary triggerings of the fall-back plan; and to win more time to carry out necessary fall-back plans.

However, this demand faces **Challenge 2** described in Section I. Basically, state reachability checking of complex hybrid automata is undecidable [4], [36], and is known to be time costly. Even for LHAs, state reachability checking is computationally time expensive and undecidable [4], [34], [36]. This is mainly due to combinatorial explosion caused by automata composition. Take CBTC for example, a normal scale CBTC of 12 trains results in an astronomical 6^{12} work modes (locations) in the composed PHA model.

Our quantitative pilot study results (see Section IV-B) also evidence **Challenge 2**. According to Section IV-B, the state reachability verification model checking time cost for merely 10 trains already exceeds 1 hour. To address **Challenge 2**, we need a method to speed up online model checking.

III. Scenario Reachability Validation based Online Model Checking for PHAs

To address **Challenge 2**, we decide to speed up the online checking based fault prediction of CPS. Inspired by the true industry demand on *scenario reachability validation* [45], we propose that online checking should switch focus from state reachability verification to *scenario reachability validation*. The latter turns the undecidable problem into a polynomial time problem solvable by *linear programming* (LP).

A. Scenario Reachability Validation

We observe that industry knows **Challenge 2** (see Section I) for long time via practice, hence does not insist on finding all reachable unsafe-states of a CPS (i.e. carrying out state reachability verification model checking or fully coverage testing on the CPS's hybrid automaton). Instead, very often, industry only demands to know if the CPS can *reach certain unsafe-states in certain work mode via certain sequence of actions*, i.e. if certain *scenario* can happen. In other words, considering the current mechanism used in industry is only scenario based simulation, industry demands *scenario reachability validation* instead of state reachability verification [45].

In the following, let us first formally define scenario reachability validation following our previous definitions on PHAs.

Definition 4: (State and Execution Trace of PHA) The *state* of a PHA $H = (X, P, L, V, V^0, E, \alpha, \beta, \gamma)$ is a tuple (ν, χ), where $\nu \in V$, and χ is an evaluation of X. An *execution trace* of H is a set of state of H, denoted as $\{(\nu(t), \chi(t))\}_{t \in [0, T_{\text{fin}}]}$, where continuous time $t \in [0, T_{\text{fin}}]$, $\nu(0) \in V^0$; and for any $t \in [0, T_{\text{fin}}]$, $\chi(t)$ complies with all location invariants, flow conditions, initial conditions (if any), guards (if any), and resets (if any) of location $\nu(t)$ in H. □

Note for consistency, we assume that if an event (v, l, ϕ, ψ, v') happens at t, then at t the state of H resides at v'. That is, execution traces are right continuous.

Definition 5: (Path of PHA) Given a PHA $H = (X, P, L, V, V^0, E, \alpha, \beta, \gamma)$, a *path* ρ for H is a finite sequence of the form $\left\langle \begin{smallmatrix} v_0 \\ \delta_0 \end{smallmatrix} \right\rangle \xrightarrow[l_0]{(\phi_0,\psi_0)} \left\langle \begin{smallmatrix} v_1 \\ \delta_1 \end{smallmatrix} \right\rangle \xrightarrow[l_1]{(\phi_1,\psi_1)} \dots \xrightarrow[l_{n-1}]{(\phi_{n-1},\psi_{n-1})} \left\langle \begin{smallmatrix} v_n \\ \delta_n \end{smallmatrix} \right\rangle$, where location $v_0 \in V^0$, $v_j \in V$ and duration $\delta_j \in \mathbb{R}_{>0}$ ($j = 0, \dots, n$), and $(v_j, l_j, \phi_j, \psi_j, v_{j+1}) \in E$ ($j = 0, \dots, n-1$). We call $\left\langle \begin{smallmatrix} v_j \\ \delta_j \end{smallmatrix} \right\rangle$ ($j = 0, \dots, n$) the jth *stage* of ρ, where v_j and δ_j are respectively the *location* and *duration* of the stage. We call $(v_j, l_j, \phi_j, \psi_j, v_{j+1})$ ($j = 0, \dots, n-1$) the jth *event* of ρ, a.k.a. the event for the jth stage of ρ. We call $\Delta_j \overset{\text{def}}{=} \sum_{k=0}^{j} \delta_k$ ($j = 0, \dots, n-1$) the *happening time* of the jth event of ρ, and $T_{\text{fin}} \overset{\text{def}}{=} \sum_{j=0}^{n} \delta_j$ the *finish time* of ρ. Define $\Delta_{-1} \overset{\text{def}}{=} 0$, we say ρ *resides* in location v_j during $[\Delta_{j-1}, \Delta_j)$ ($j = 0, \dots, n-1$); and ρ *resides* in location v_n during $[\Delta_{n-1}, T_{\text{fin}}]$. We say an execution trace $\{(\nu(t), \chi(t))\}_{t\in[0,T_{\text{fin}}]}$ of H *matches* ρ iff

$$\begin{cases} \nu(t) = v_j & \text{(when } \exists j \in \{0, \dots, n-1\} \text{ s.t.} \\ & \Delta_{j-1} \leqslant t < \Delta_j) \\ \nu(t) = v_n & \text{(when } \Delta_{n-1} \leqslant t \leqslant T_{\text{fin}}) \end{cases}.$$ □

As PHAs can be composed, we also extend the above concepts to composed PHAs.

Definition 6: (Projection of Execution Trace) Given composed PHA $H = H_1 || \dots || H_m$, where $H = (X, P, L, V, V^0, E, \alpha, \beta, \gamma)$ and PHAs $H_i = (X_i, P_i, L_i, V_i, V_i^0, E_i, \alpha_i, \beta_i, \gamma_i)$ ($i = 1, \dots, m$), given an execution trace $\theta = \{(\nu(t), \chi(t))\}_{t\in[0,T_{\text{fin}}]}$ of H, as per Def. 2, there must be $\nu(t) = (\nu_1(t), \dots, \nu_m(t))$ (where $\nu_i(t) \in V_i$ for $i = 1, \dots m$) for each $t \in [0, T_{\text{fin}}]$. Meanwhile, as per Def. 2, $X_i \subseteq X$ (for $i = 1, \dots m$). Thus we can use evaluation $\chi(t)$ to evaluate every element of X_i. Suppose the resulted evaluation is $\chi_i(t)$. We call the set $\{(\nu_i(t), \chi_i(t))\}_{t\in[0,T_{\text{fin}}]}$ the *projection of θ on H_i*, denoted as $\theta \downarrow H_i$. We also denote $\nu_i(t) \overset{\text{def}}{=} \nu(t) \downarrow H_i$ and $\chi_i(t) \overset{\text{def}}{=} \chi(t) \downarrow H_i$. □

Clearly, $\theta \downarrow H_i$ in Def. 6 is an execution trace in H_i.

Definition 7: (Composable Path Set) Let $H = (X, P, L, V, V^0, E, \alpha, \beta, \gamma) = H_1 || \dots || H_m$ be a composed PHA of PHAs $H_i = (X_i, P_i, L_i, V_i, V_i^0, E_i, \alpha_i, \beta_i, \gamma_i)$ ($i = 1, \dots, m$). A set $\{\rho_i\}$ ($i = 1, \dots, m$) is a *composable path set* for $(H_1, \dots, H_m)$ iff each ρ_i ($i = 1, \dots, m$) is a path for H_i of the form $\left\langle \begin{smallmatrix} v_{i,0} \\ \delta_{i,0} \end{smallmatrix} \right\rangle \xrightarrow[l_{i,0}]{(\phi_{i,0},\psi_{i,0})} \left\langle \begin{smallmatrix} v_{i,1} \\ \delta_{i,1} \end{smallmatrix} \right\rangle \xrightarrow[l_{i,1}]{(\phi_{i,1},\psi_{i,1})} \dots \xrightarrow[l_{i,n_i-1}]{(\phi_{i,n_i-1},\psi_{i,n_i-1})} \left\langle \begin{smallmatrix} v_{i,n_i} \\ \delta_{i,n_i} \end{smallmatrix} \right\rangle$, and ρ_is are synchronized to each other by shared events, i.e. $\forall j \in \{1, \dots, m\}$, we have:

C1 $\forall l \in L_j$, we have $S(l) \subseteq X \cup P$ and $\forall k \in \{1, \dots, m\}$, $(S(l) \cap (X_k \cup P_k) \neq \varnothing) \Rightarrow (l \in L_k)$.

C2 $\forall k \in \{1, \dots, m\} \setminus \{j\}$, $\forall l \in L_j \cap L_k$, for any occurrence in ρ_j of l, without loss of generality, suppose it is the dth occurrence, and the occurring element is $l_{j,\ell}$ ($\ell \in \{0, \dots, n_j - 1\}$), then the dth occurrence of l in ρ_k must also exist, suppose the occurring element is $l_{k,h}$ ($h \in \{0, \dots, n_k - 1\}$), we have $\sum_{\imath=0}^{\ell} \delta_{j,\imath} = \sum_{\imath=0}^{h} \delta_{k,\imath}$.

C3 $\sum_{\ell=0}^{n_j} \delta_{j,\ell} = \sum_{\ell=0}^{n_k} \delta_{k,\ell}$ (i.e. all finish times are the same). □

Let $H = H_1 || \dots || H_m$ be a composed PHA of PHAs $H_i = (X_i, P_i, L_i, V_i, V_i^0, E_i, \alpha_i, \beta_i, \gamma_i)$ ($i = 1, \dots, m$). Given a composable path set $\{\rho_i\}_{i=1}^m$ for $(H_1, \dots, H_m)$, where path ρ_i for H_i is of the form $\left\langle \begin{smallmatrix} v_{i,0} \\ \delta_{i,0} \end{smallmatrix} \right\rangle \xrightarrow[l_{i,0}]{(\phi_{i,0},\psi_{i,0})} \left\langle \begin{smallmatrix} v_{i,1} \\ \delta_{i,1} \end{smallmatrix} \right\rangle \xrightarrow[l_{i,1}]{(\phi_{i,1},\psi_{i,1})} \dots \xrightarrow[l_{i,n_i-1}]{(\phi_{i,n_i-1},\psi_{i,n_i-1})} \left\langle \begin{smallmatrix} v_{i,n_i} \\ \delta_{i,n_i} \end{smallmatrix} \right\rangle$, denote $T_{\text{fin}} \overset{\text{def}}{=} \sum_{j=0}^{n_1} \delta_{1,j}$, then due to **C3**, $\forall i \in \{1, \dots, m\}$, $\sum_{j=0}^{n_i} \delta_{i,j} \equiv T_{\text{fin}}$. Meanwhile, we create an empty list $\mathcal{E} := ()$, and then for each $\left\langle \begin{matrix} v_{i,j} \\ \delta_{i,j} \end{matrix} \right\rangle \xrightarrow[l_{i,j}]{(\phi_{i,j},\psi_{i,j})}$ in ρ_i ($i = 1, \dots, m$; $j = 0, \dots, n_{i-1}$), let $\Delta_{i,j} \overset{\text{def}}{=} \sum_{k=0}^{j} \delta_{i,k}$ and insert tuple $(\Delta_{i,j}, l_{i,j})$ into $\mathcal{E}$ as per ascending order of $\Delta_{i,j}$ value, breaking ties by

1) replacing the existing tuple with the new tuple (if the event labels are the same, i.e. shared event).
2) ascending order of i value (if the event labels are different) [3]

With the above T_{fin} and sorted list $\mathcal{E}$, we can compose ρ_is according to the algorithm of Fig. 4.

```
1.  PathComposition($\{\rho_i\}_{i=1}^m$, $\mathcal{E}$, $T_{\text{fin}}$) begin
2.      $\varrho := ()$; // () means empty list
3.      $v := (v_{1,0}, \dots, v_{m,0})$, $v' := ()$;
4.      $\phi := \varnothing$, $\psi := \varnothing$, $t := 0$;
5.      foreach $i$ in $\{1, \dots, m\}$, let $j_i := 0$ and $j'_i := 1$;
6.      while ($\mathcal{E}$ is not empty) begin
7.          deque the head element $(\Delta, l)$ of $\mathcal{E}$;
8.          foreach $i$ in $\{1, \dots, m\}$
9.              if $(l_{i,j_i} = l)$ then begin
10.                 $v' := (v', v_{i,j'_i})$; //list concatenation
11.                 $\phi := \phi \cup \phi_{i,j_i}$, $\psi := \psi \cup \psi_{i,j_i}$;
12.                 $j_i := j'_i$, $j'_i := j'_i + 1$;
13.             end else
14.                 $v' := (v', v_{i,j_i})$;
15.         $\varrho := (\varrho, \left\langle \begin{smallmatrix} v \\ \Delta - t \end{smallmatrix} \right\rangle \xrightarrow[l]{(\phi,\psi)})$; //list concatenation
16.         $v := v'$, $v' := ()$, $\phi := \varnothing$, $\psi := \varnothing$, $t := \Delta$;
17.     end;
18.     $\varrho := (\varrho, \left\langle \begin{smallmatrix} v \\ T_{\text{fin}} - t \end{smallmatrix} \right\rangle)$; //list concatenation
19.     return $\varrho$;
20. end;
```

Fig. 4. Pseudo code of PHA path composition. $\{\rho_i\}_{i=1}^m$ is the composable path set, $\mathcal{E}$ is the sorted linked list of all (event happening time, event label) tuples of $\{\rho_i\}_{i=1}^m$, T_{fin} is the finish time of all paths in $\{\rho_i\}_{i=1}^m$.

We call the above ϱ a *composed path* for H from $\{\rho_i\}_{i=1}^m$, and denote it as $\varrho = \rho_1 || \dots || \rho_m$. Furthermore, we call ρ_i ($i \in \{1, \dots, m\}$) the *projection of ϱ on H_i*, denoted as $\rho_i = \varrho \downarrow H_i$. We have the following.

Lemma 1: (Path Matching between Execution Trace and Its Projections) Let $H = (X, P, L, V, V^0, E, \alpha, \beta, \gamma) = H_1 || \dots || H_m$ be a composed PHA of PHAs $H_i = (X_i, P_i, L_i, V_i, V^0, E_i, \alpha_i, \beta_i\ \gamma_i)$ ($i = 1, \dots, m$). Set $\{\rho_i\}_{i=1}^m$ is a composable path set for $(H_1, \dots, H_m)$, and $\varrho = \rho_1 || \dots || \rho_m$. Suppose $\theta = \{(\nu(t), \chi(t))\}_{t\in[0,T_{\text{fin}}]}$ is an execution trace of H matching ϱ, then $\theta \downarrow H_i$ is an execution trace of H_i matching ρ_i ($\forall i \in \{1, \dots, m\}$). ■

[3] This means we are ignoring the different interleaving orders of the events of different PHAs that happened simultaneously.

Proof: If θ matches ϱ, then θ changes and only changes location at ϱ's event happening times. Suppose at any event happening time Δ of ϱ, the happening event is $e = (v, l, \phi, \psi, v')$, then we have $\nu(\Delta^-) = v$ and $\nu(\Delta) = v'$. Meanwhile, we have *Case 1*) an event labeled l also happens in ρ_i, which changes ρ_i's residing location from v_i to v_i'; or *Case 2*) no event labeled l is happening in ρ_i, which changes ρ_i's residing location from v_i to $v_i' = v_i$. Either way, v_i and v_i' are respectively the ith element of vector v and v'. This means $\nu(\Delta^-) \downarrow H_i = v_i$ and $\nu(\Delta) \downarrow H_i = v_i'$.

Also, between any two consecutive event happening time of ϱ, $\nu(t)$ does not change; and as no event happens in ρ_i, ρ_i does not change residing location,

Above all means $\theta \downarrow H_i$ matches ρ_i. ■

With the above notions, we can define scenario reachability related concepts.

Definition 8: (Reachability Specification) Given a PHA $H = (X, P, L, V, V^0, E, \alpha, \beta, \gamma)$, suppose each $x \in X$ is given a unique index label $k \in \{1, \ldots, |X|\}$ and denoted as x_k. A *reachability specification*, denoted as $\mathcal{R}(v, \varphi)$, is a tuple consisting of a location $v \in V$ and a set φ of parameter constraints of the form $a \leqslant \sum_{k=1}^{|X|} c_k x_k \leqslant b$, where a, b, and c_k $(k = 1, \ldots, |X|)$ are real constants.

Furthermore, given a path ρ of H, we say H *reaches* $\mathcal{R}$ *along* ρ, denoted as $H \overset{\rho}{\rightsquigarrow} \mathcal{R}$, iff there exists an execution trace $\theta = \{(\nu(t), \chi(t))\}_{t \in [0, T_{\text{fin}}]}$ of H matching ρ, such that $\nu(T_{\text{fin}}) = v$ and $\chi(T_{\text{fin}})$ satisfies all constraints in φ. □

With the above concepts, we can formally define *scenario reachability validation*.

Definition 9: (Scenario Reachability Validation) Given a PHA $H = (X, P, L, V, V^0, E, \alpha, \beta, \gamma)$, a path ρ of H of the form $\left\langle \begin{smallmatrix} v_0 \\ \delta_0 \end{smallmatrix} \right\rangle \xrightarrow[l_0]{(\phi_0,\psi_0)} \left\langle \begin{smallmatrix} v_1 \\ \delta_1 \end{smallmatrix} \right\rangle \xrightarrow[l_1]{(\phi_1,\psi_1)} \cdots \xrightarrow[l_{n-1}]{(\phi_{n-1},\psi_{n-1})} \left\langle \begin{smallmatrix} v_n \\ \delta_n \end{smallmatrix} \right\rangle$ is called a *scenario* of H iff all δ_js $(j = 0, \ldots, n;\ \delta_j \in \mathbb{R}_{>0})$ are configurable variables. Given a reachability specification $\mathcal{R}(v, \varphi)$ of H, a *scenario reachability validation on* $(H, \rho, \mathcal{R})$ checks if there is an evaluation of all the δ_js s.t. $H \overset{\rho}{\rightsquigarrow} \mathcal{R}$. If the answer is yes, we denote $(H, \rho, \mathcal{R}) \models H \overset{\rho}{\rightsquigarrow} \mathcal{R}$; otherwise, we denote $(H, \rho, \mathcal{R}) \not\models H \overset{\rho}{\rightsquigarrow} \mathcal{R}$. □

Definition 10: (Composed Scenario Reachability Validation) Let PHA $H = (X, P, L, V, V^0, E, \alpha, \beta, \gamma) = H_1 || \ldots || H_m$ be a composed PHA of PHAs $H_i = (X_i, P_i, L_i, V_i, V_i^0, E_i, \alpha_i, \beta_i, \gamma_i)$ $(i = 1, \ldots, m)$. Let ρ_i $(i = 1, \ldots, m)$ of the form $\left\langle \begin{smallmatrix} v_{i,0} \\ \delta_{i,0} \end{smallmatrix} \right\rangle \xrightarrow[l_{i,0}]{(\phi_{i,0},\psi_{i,0})} \left\langle \begin{smallmatrix} v_{i,1} \\ \delta_{i,1} \end{smallmatrix} \right\rangle \xrightarrow[l_{i,1}]{(\phi_{i,1},\psi_{i,1})} \cdots \xrightarrow[l_{i,n_i-1}]{(\phi_{i,n_i-1},\psi_{i,n_i-1})} \left\langle \begin{smallmatrix} v_{i,n_i} \\ \delta_{i,n_i} \end{smallmatrix} \right\rangle$ be a path for H_i, where all $\delta_{i,j}$s $(i = 1, \ldots, m;\ j = 0, \ldots, n_i;\ \delta_{i,j} \in \mathbb{R}_{>0})$ are configurable variables (i.e. ρ_i is a scenario of H_i). Furthermore, given a reachability specification $\mathcal{R}(v, \varphi)$ of H, a *composed scenario reachability validation on* $(H, \{\rho_i\}_{i=1}^m, \mathcal{R})$ checks if there is an evaluation of all the above $\delta_{i,j}$s, s.t. $\{\rho_i\}_{i=1}^m$ is a composable path set and $H \overset{\varrho}{\rightsquigarrow} \mathcal{R}$ (where $\varrho \overset{\text{def}}{=} \rho_1 || \ldots || \rho_m$, we call ρ_is the "*component scenarios* of H" and ϱ the "*composed scenario* of H"). If the answer is yes, we denote $(H, \{\rho_i\}_{i=1}^m, \mathcal{R}) \models H \overset{\varrho}{\rightsquigarrow} \mathcal{R}$; otherwise we denote $(H, \{\rho_i\}_{i=1}^m, \mathcal{R}) \not\models H \overset{\varrho}{\rightsquigarrow} \mathcal{R}$. □

B. Composed Scenario Reachability Validation as a Linear Programming (LP) Problem

Composed scenario reachability validation time cost can still go exponential. For example, when the m component paths (i.e. ρ_is) of Def. 10 do not have shared events, the total possibilities of ϱ's topology alone is already $\Omega(\Pi_{i=1}^m n_i)$, depending on the configurations of $\delta_{i,j}$s.

To apply our proposed validation for online usage, we must reduce the time cost. We find that in case PHAs can be concretized to LHAs (see Def. 3), we can decompose the *composed* scenario reachability validation to *component* scenario reachability validation, and use LP to solve the problem by the "shallow synchronization" encoding proposed in [17], reducing time cost to polynomial.

Now, let us reformulate the "shallow synchronization" encoding in the formalism system of this paper as follows:

Theorem 1: (LHA Composed Scenario Reachability Validation) Let LHA $H = (X, L, V, V^0, E, \alpha, \beta, \gamma) = H_1 || \ldots || H_m$ be a composed LHA of LHAs $H_i = (X_i, L_i, V_i, V_i^0, E_i, \alpha_i, \beta_i, \gamma_i)$ $(i = 1, \ldots, m)$. Suppose each $x \in X_i$ $(i \in \{1, \ldots, m\})$ is given a unique index label (i, k) $(k \in \{1, \ldots, |X_i|\})$, hence is denoted as $x_{i,k}$. Note according to Formula (1)(2) and Def. 2, the (i, k) index label therefore also uniquely identifies each $x \in X$. Let ρ_i $(i = 1, \ldots, m)$ of the form $\left\langle \begin{smallmatrix} v_{i,0} \\ \delta_{i,0} \end{smallmatrix} \right\rangle \xrightarrow[l_{i,0}]{(\phi_{i,0},\psi_{i,0})} \left\langle \begin{smallmatrix} v_{i,1} \\ \delta_{i,1} \end{smallmatrix} \right\rangle \xrightarrow[l_{i,1}]{(\phi_{i,1},\psi_{i,1})} \cdots \xrightarrow[l_{i,n_i-1}]{(\phi_{i,n_i-1},\psi_{i,n_i-1})} \left\langle \begin{smallmatrix} v_{i,n_i} \\ \delta_{i,n_i} \end{smallmatrix} \right\rangle$ be a path for H_i, where all $\delta_{i,j}$s $(i = 1, \ldots, m;\ j = 0, \ldots, n_i;\ \delta_{i,j} \in \mathbb{R}_{>0})$ are configurable variables (i.e. ρ_i is a scenario of H_i). Given reachability specification $\mathcal{R}(v, \varphi)$ of H, where $v = (v_{1,n_1}, \ldots, v_{m,n_m})$. Then $(H, \{\rho_i\}_{i=1}^m, \mathcal{R}) \models H \overset{\varrho}{\rightsquigarrow} \mathcal{R}$ iff the following set of constraints on $\delta_{i,j} \in \mathbb{R}_{>0}$, $\lambda_{i,j,k} \in \mathbb{R}$, and $\zeta_{i,j,k} \in \mathbb{R}$ $(i = 1, \ldots, m;\ j = 0, \ldots, n_i;\ k = 1, \ldots, |X_i|)$ is feasible.

C1, **C2**, **C3** are respectively the same as **C1**, **C2**, **C3** of Def. 7.

C4 For each $x_{i,k} \in X_i$ $(i = 1, \ldots, m;\ k = 1, \ldots, |X_i|)$, $x_{i,k} := \lambda_{i,0,k}$ is a valid initial condition in $v_{i,0}$, i.e. $x_{i,k} := \lambda_{i,0,k} \in \gamma(v_{i,0})$.

C5 For each flow condition $\dot{x}_{i,k} = [u_{i,j,k}, u'_{i,j,k}] \in \beta_i(v_{i,j})$ $(i = 1, \ldots, m;\ j = 0, \ldots, n_i;\ k = 1, \ldots, |X_i|)$, there is $u_{i,j,k}\delta_{i,j} \leqslant \zeta_{i,j,k} - \lambda_{i,j,k} \leqslant u'_{i,j,k}\delta_{i,j}$, where $\zeta_{i,j,k}$ and $\lambda_{i,j,k}$ represents the valuation of $x_{i,k}$ leaves and enters location $v_{i,j}$ respectively.

C6 For each location invariant constraint $a \leqslant \sum_{k=1}^{|X_i|} c_{i,k} x_{i,k} \leqslant b$ in $\alpha_i(v_{i,j})$ $(i = 1, \ldots, m;\ j = 0, \ldots, n_i)$, there is $\begin{cases} a \leqslant \sum_{k=1}^{|X_i|} c_{i,k}\lambda_{i,j,k} \leqslant b \\ a \leqslant \sum_{k=1}^{|X_i|} c_{i,k}\zeta_{i,j,k} \leqslant b \end{cases}$.

C7 For each guard $a \leqslant \sum_{k=1}^{|X_i|} c_{i,k} x_{i,k} \leqslant b$ in $\phi_{i,j}$ $(i = 1, \ldots, m;\ j = 0, \ldots, n_i - 1)$, there is $a \leqslant \sum_{k=1}^{|X_i|} c_{i,k}\zeta_{i,j,k} \leqslant b$.

C8 $\forall i \in \{1, \ldots, m\}, \forall j \in \{0, \ldots, n_i - 1\}$, we have

C8.1 if the event of $l_{i,j}$ is not a received event, then according to Def. 1.3e and Def. 3, resets in $\psi_{i,j}$ on $x_{i,k}$ $(k = 1, \ldots, |X_i|)$ must have the form $x_{i,k} := \sum_{\ell=1}^{|X_i|} c_{i,j,\ell} x_{i,\ell}$. Correspondingly, we have the constraint $\lambda_{i,j+1,k} = \sum_{\ell=1}^{|X_i|} c_{i,\ell}\zeta_{i,j,\ell}$.

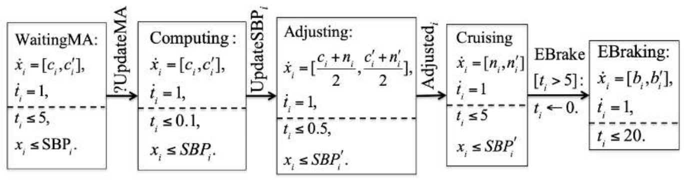


Fig. 5. Scenario for **R1**

C8.2 if the event of $l_{i,j}$ is a received event, denote $l \overset{\text{def}}{=} l_{i,j}$, and suppose $l_{i,j}$ is the dth occurrence of label l in ρ_i. Denote $K \overset{\text{def}}{=} \{\kappa | \kappa \in \{1, \ldots, m\} \setminus \{i\}, l \in L_\kappa, \text{ and } S(l) \cap X_\kappa \neq \varnothing\}$. Then for each ρ_κ (where $\kappa \in K$), according to **C2**, the dth occurrence of l exists. Suppose that is l_{κ,h_κ} ($h_\kappa \in \{0, \ldots, n_\kappa - 1\}$). Also, according to Def. 1.3e, Def. 3, **C1**, and **C2**, resets in $\psi_{i,j}$ on $x_{i,k}$ ($k = 1, \ldots, |X_i|$) must have the form $x_{i,k} := (\sum_{\ell=1}^{|X_i|} c_{i,j,\ell} x_{i,\ell}) + \sum_{\kappa \in K} \sum_{\ell \in I_\kappa} c_{\kappa,h_\kappa,\ell} x_{\kappa,\ell}$ (where $I_\kappa \overset{\text{def}}{=} \{\ell | \ell \in \{1, \ldots, |X_\kappa|\} \text{ and } x_{\kappa,\ell} \in S(l)\}$). Correspondingly, we have the constraint $\lambda_{i,j+1,k} = (\sum_{\ell=1}^{|X_i|} c_{i,j,\ell} \zeta_{i,j,\ell}) + \sum_{\kappa \in K} \sum_{\ell \in I_\kappa} c_{\kappa,h_\kappa,\ell} \zeta_{\kappa,h_\kappa,\ell}$.

C9 For each constraint $a \leqslant \sum_{i=1}^{m} \sum_{k=1}^{|X_i|} c_{i,k} x_{i,k} \leqslant b$ in φ, we have $a \leqslant \sum_{i=1}^{m} \sum_{k=1}^{|X_i|} c_{i,k} \zeta_{i,n_i,k} \leqslant b$. ■

The proof of the above theorem is referred to the appendix of [17]. Here, we use the CBTC motivating example to show a real-case scenario reachability validation problem and how can it be solved by the related method.

Recall safety rule **R1** (see Section II-A), the state reachability verification specification of **R1** is whether the position of Train_i ($i = 1, \ldots, m-1$) on track can reach that of Train_{i+1} during the EBraking (see Fig. 2(A)) work mode within time horizon length $F = 25$ seconds.

Our CBTC CPS industry collaborators are only concerned with some special scenario reachabilities for **R1** actually. The concerned scenario is: after sync with RBC, all the trains will receive the "UpdateMA" event, and start to run under the new control parameters, whether Train_i ($i = 1, \ldots, m-1$) will hit the train ahead of it (i.e. Train_{i+1}) during their "EBraking" work modes within the online model checking time horizon (see Fig. 2(A)).

This is formalized as $(m-1)$ composed scenario reachability problems. The concretized LHA H_i of Train_i ($i = 1, \ldots, m$) is shown in Fig. 5, which can be derived from Fig. 2(A). The component scenario ρ_i from H_i ($i = 1, \ldots, m$) is $\left\langle \begin{array}{c} \text{WaitingMA} \\ \delta_{i,0} \end{array} \right\rangle \xrightarrow{\text{UpdateMA}} \left\langle \begin{array}{c} \text{Computing} \\ \delta_{i,1} \end{array} \right\rangle \xrightarrow{\text{UpdateSBP}_i} \left\langle \begin{array}{c} \text{Adjusting} \\ \delta_{i,2} \end{array} \right\rangle \xrightarrow{\text{Adjusted}_i} \left\langle \begin{array}{c} \text{Cruising} \\ \delta_{i,3} \end{array} \right\rangle \xrightarrow{\text{EBrake}_i} \left\langle \begin{array}{c} \text{EBraking} \\ \delta_{i,4} \end{array} \right\rangle$. Note the event guards and resets are omitted due to space limit. Interested readers can refer to the corresponding edges in Fig. 2(A).

In the ith ($i = 1, \ldots, m-1$) composed scenario reachability problem, scenario ρ_i is composed with scenario ρ_{i+1}; and the reachability constraint φ_i is $0 \leqslant x_i - x_{i+1} \leqslant 0$ and $t \leqslant 25$, where x_i and x_{i+1} are respectively Train_i and Train_{i+1}'s position on track, t is the time, and 25 is the concerned online model checking time horizon length F.

Now, take a scenario of only 2 trains for example, the scenario we have is:

- ρ_1 : $\left\langle \begin{array}{c} \text{WaitingMA} \\ \delta_{1,0} \end{array} \right\rangle \xrightarrow{\text{UpdateMA}} \left\langle \begin{array}{c} \text{Computing} \\ \delta_{1,1} \end{array} \right\rangle \xrightarrow{\text{UpdateSBP}_1} \left\langle \begin{array}{c} \text{Adjusting} \\ \delta_{1,2} \end{array} \right\rangle \xrightarrow{\text{Adjusted}_1} \left\langle \begin{array}{c} \text{Cruising} \\ \delta_{1,3} \end{array} \right\rangle \xrightarrow{\text{EBrake}_1} \left\langle \begin{array}{c} \text{EBraking} \\ \delta_{1,4} \end{array} \right\rangle$.
- ρ_2 : $\left\langle \begin{array}{c} \text{WaitingMA} \\ \delta_{2,0} \end{array} \right\rangle \xrightarrow{\text{UpdateMA}} \left\langle \begin{array}{c} \text{Computing} \\ \delta_{2,1} \end{array} \right\rangle \xrightarrow{\text{UpdateSBP}_2} \left\langle \begin{array}{c} \text{Adjusting} \\ \delta_{2,2} \end{array} \right\rangle \xrightarrow{\text{Adjusted}_2} \left\langle \begin{array}{c} \text{Cruising} \\ \delta_{2,3} \end{array} \right\rangle \xrightarrow{\text{EBrake}_2} \left\langle \begin{array}{c} \text{EBraking} \\ \delta_{2,4} \end{array} \right\rangle$.

Now, let us show how to encode the scenario reachability problem

- **C1-C3** are about Synchronization Encoding:
  - According to **C1**, the shared label between these two components is UpdateMA only.
  - **C2** says the related component should fire the shared label at the same time, so we have $\delta_{1,0} = \delta_{2,0}$.
  - **C3** asks all the component should reach the target state by the same time, therefore we have $\sum_{\ell=0}^{4} \delta_{1,\ell} = \sum_{\ell=0}^{4} \delta_{2,\ell}$
- **C4-C9** are about Local Encoding:
  - Take location $Cruising$ of ρ_1 for example, we mark $Cruising$ as location $v_{1,4}$, we mark variable x_1 as $x_{1,1}$, t_1 as $x_{1,2}$. According to **C5** , for variable x_1, we have variables $\zeta_{1,4,1}$ and $\lambda_{1,4,1}$ encode the valuation of x_1 leaves and enters $Cruising$ ($v_{1,4}$) respectively. We also get constraint $n_i \delta_{1,4} \leq \zeta_{1,4,1} - \lambda_{1,4,1} \leq n_i' \delta_{1,4}$
  - For invariant $x_1 \leq SBP_1'$ in $Cruising$, we have $\zeta_{1,4,1} \leq SBP_1'$ and $\lambda_{1,4,1} \leq SBP_1'$ according to **C6**.
  - For guard $t_1 > 5$ on transition $Ebrake_i$, we have $\zeta_{1,4,2} > 5$, according to **C7**.
  - For reset $t_1 := 0$ on transition $Ebrake_i$, we have $\zeta_{1,5,2} := 0$, according to **C8.1**.
  - For the specification $0 \leqslant x_1 - x_2 \leqslant 0$ and $t \leqslant 25$ in φ, we have $0 \leqslant \zeta_{1,5,1} - \zeta_{2,5,1} \leqslant 0$ and $\zeta_{1,5,2} \leqslant 25$, according to **C9**.

C. Online Fault Prediction with Online Composed Scenario Reachability Validation Model Checking

Now, we show how our online scenario reachability verification is integrated into the classical control loop. The general idea of classical control loop for complex CPS systems is shown in the left part of Fig.6. Whenever a running system gets a new instruction/command or receives stimuli from the environment, it will compute/collect the numeric values of the control parameters, then the parameters will be deployed by the running system immediately. If we call the left part of Fig.6 as "classical control system", our online verification module, the right part of Fig.6, can be used as a "runtime monitor" [44] which works as a guardian of the "classical control system" and guarantee the safety of system operation.

1) After the control functions are called by the running system and a new set of parameter values are generated, these values will be deployed on the running system immediately as in the classical control loop. In other

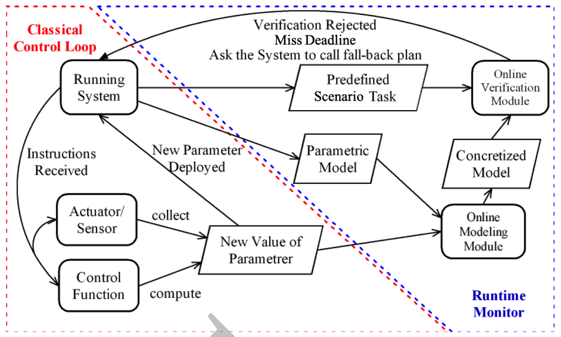


Fig. 6. Online Modeling and Verification Framework

words, the running system will not wait for the pass of the verification to deploy the control parameters. Because in such manner, during the verification period the system have to work under the old set of parameter values, which is out-of-date and could be dangerous.

2) Meanwhile, once the new set of parameter values is generated, the online modeling module will be called to build the online model for the time-bounded behavior of the system by concretizing the free parameters in the model according to their numeric values.
3) Then, the online verification module will verify the predefined scenario verification tasks on the concretized models. If the verification is rejected, or the verification module fails to get a result before the deadline, the online verification module will ask the system to stop the current parameters immediately and start a fall-back plan which is predefined by designers to guarantee the safety of system operation.

In such manner, the offline unverifiable parametric model can be verified online. The online modeling and verification module can be introduced as a runtime monitor into the classical control loop. Our monitor will not interrupt the behavior of the system unless the verification is rejected.

More specifcally, we modify the online state reachability verification model checking algorithm of Fig. 3 to the online composed scenario reachability validation model checking algorithm of Fig. 7.

```
   //online model checking called every $T$ seconds.
   //online validation deadline is $D$.
1. OnlineFaultPrediction($D$, $\{\rho_i\}_{i=1}^m$, $\mathcal{R}(v, \varphi)$) begin
2.    set $t_0$ to the current time; set $T_{\text{fin}}$ to $t_0 + F$;
3.    concretize each component PHA to an HA $\{H_i\}_{i=1}^m$;
      //Denote $H = H_1 || \ldots || H_m$
4.    if ((($H$, $\{\rho_i\}_{i=1}^m$, $\mathcal{R}$) $\models H \overset{\varrho}{\rightsquigarrow} \mathcal{R}$)
5.       or (current time $t \geqslant t_0 + D$)) then
6.       trigger the fall-back plan; //non-blocking call.
7. end;
```

Fig. 7. Pseudo code of online composed scenario reachability validation model checking. Note, unlike ρ_is, H_is are always composable due to Def. 2: no composability conditions are needed.

D. Complexity, Soundness, and Completeness

We have the following results on the time cost, *soundness* (i.e. no false alarm is triggerred; formally, if line 6 of Fig. 7 is executed, then there must be $(H, \{\rho_i\}_{i=1}^m, \mathcal{R}) \models H \overset{\varrho}{\rightsquigarrow} \mathcal{R}$), and *completeness* (i.e. no alarm is missed; formally, if $(H, \{\rho_i\}_{i=1}^m, \mathcal{R}) \models H \overset{\varrho}{\rightsquigarrow} \mathcal{R}$, then line 6 of Fig. 7 must be executed) of the algorithm of Fig. 7.

Theorem 2: (LHA Composed Scenario Reachability Validation Time Cost, Soundness, and Completeness) For the given H, $\{\rho_i\}_{i=1}^m$, and $\mathcal{R}$ in Theorem 1, the validation of $(H, \{\rho_i\}_{i=1}^m, \mathcal{R}) \models H \overset{\varrho}{\rightsquigarrow} \mathcal{R}$ can be done via linear programming (LP), hence takes polynomial time. Further more, given sufficient time (i.e. big enough D), the online validation algorithm of Fig. 7 is sound and complete. In case D is not big enough (but $D \geqslant 0$), the algorithm of Fig. 7 is complete. ∎

Proof: Due to Theorem 1, checking $(H, \{\rho_i\}_{i=1}^m, \mathcal{R}) \models H \overset{\varrho}{\rightsquigarrow} \mathcal{R}$ is equivalent to checking the feasibility of constraints **C1** $\sim$ **C9**. Checking **C1** takes polynomial time. Constructing all linear constraints in **C2** $\sim$ **C9** takes polynomial time. Finding a feasible solution to the constraints in **C2** $\sim$ **C9** is an LP problem, which also takes polynomial time. (*)

(*) implies line 4 of Fig. 7 costs polynomial time. Denote the exact time cost as T_{cost}. If $D \geqslant T_{\text{cost}}$, then the only reason that triggers the execution of line 6 of Fig. 7 is $(H, \{\rho_i\}_{i=1}^m, \mathcal{R}) \models H \overset{\varrho}{\rightsquigarrow} \mathcal{R}$. Meanwhile, when $D \geqslant T_{\text{cost}}$ and $(H, \{\rho_i\}_{i=1}^m, \mathcal{R}) \models H \overset{\varrho}{\rightsquigarrow} \mathcal{R}$, line 4 of Fig. 7 will return true to trigger the execution of line 6. (**)

When $0 \leqslant D < T_{\text{cost}}$ and $(H, \{\rho_i\}_{i=1}^m, \mathcal{R}) \models H \overset{\varrho}{\rightsquigarrow} \mathcal{R}$, line 5 of Fig. 7 will trigger the execution of line 6. (***)

Beause of (*), (**), and (***), we proved the theorem. ∎

Now, the only missing case is the soundness of algorithm in Fig. 7 when D is not big enough ($0 \leqslant D < T_{\text{cost}}$). In this case, false positive alarms may be triggered. Fortunately, as LP is well studied, mature high performance LP solvers are widely available. Hence, most of the time T_{cost} is very small, making $D < T_{\text{cost}}$ cases empirically rare. This is corroborated by our evaluation in Section IV. As shown by Table II and III, the time costs of model checking $(H, \{\rho_i\}_{i=1}^m, \mathcal{R}) \models H \overset{\varrho}{\rightsquigarrow} \mathcal{R}$ (i.e. T_{cost}) are all below 200ms, which is way smaller than the usual configurations of D at the granularity of seconds.

IV. Implementation and Evaluation

A. Tool Implementation & Setting

Tool Implementation. In order to support the online modeling and verification procedure, we implemented our proposed PHA scenario reachability validation mechanisms (see Section III) in BACH [18] model checker, and extended BACH [18] into a special version BACH$_{OL}$ which can support the online linear hybrid automata modeling and verification.

Inheriting all the functionalities of BACH, BACH$_{OL}$ is extended with the following new capabilities:

1) A graphical editor for modeling parametric LHA with free symbolic parameters.
2) A module which can generate concrete LHA model automatically by replacing the free parameters in the PHA

to the numeric values and performing the mathematical computation whenever needed.

Online Setting. In the experiments, BACH$_{OL}$ communicates with the onboard train control system by UDP. At the beginning of each cycle, the values of all the free parameters will be concretized. Then, the onboard system will pack up all the runtime values, including the current speed of the train, new MA value, new SBP value, new velocity ranges and so on. Besides these data, for the sake of security, the id of the sender (train), receiver (checker), the serial number and the size of the message package will be sent as well, and result in a package of 28 bytes in total. Then, the train control system sends this data package to the checker through the UDP socket.

After receiving the package, BACH$_{OL}$ confirms the package is sent to it from the correct train and confirms there is no package loss by checking the continuity of the serial number. Then, it retrieves the corresponding runtime data from the package and concretizes the parametric model correspondingly. Then, the predefined verification tasks will be conducted. The verification results will be written into a package with the size of 8 bytes, and send back to the corresponding train through the UDP socket again.

B. Online State Reachability Verification Model Checking

We conduct experiments on our CBTC CPS (see Section II-A) to compare online state reachability verification model checking with our proposed online scenario reachability validation model checking for online CPS fault-prediction. The experiments are conducted on a Think Center (UBUNTU 16.04, 64 bit, Intel Core i7-6700 CPU $3.40\mathrm{GHz}$, $16\mathrm{GB}$ RAM).

The CBTC CPS safety rules **R1** and **R2** (see Section II-A) can be translated into the following state reachability verification specifications.

- The state reachability verification specification of **R1** is whether the position of Train_i ($i = 1, \ldots, m-1$) on track can reach that of Train_{i+1} during the EBraking (see Fig. 2(A)) work mode within time horizon length $F = 25$ seconds, refer to Section III-B.
- The state reachability verification specification of **R2** is whether the position of Train_i ($i = 1, \ldots, m$) on track can exceed its EOA during the SBraking (see Fig. 2(A)) work mode within time horizon length $F = 50$ seconds.

We conduct the online state reachability verification model checking with the state-of-the-art LHA model checker SpaceEx [32]. The time costs are listed in Table I. Note in the table, the time cost for **R1** refers to the time cost to model check if one pair of neighboring trains (e.g. Train_1 and Train_2) will hit each other. Model checking other pairs of trains can be conducted in parallel independently. In the table, the time cost for **R2** refers to the time cost to model check if one train (e.g. Train_1) will exceed its EOA. Model checking of other trains can be conducted in parallel independently. The statistics (i.e. mean, max, std) are conducted upon 100 trials (for the cases of $2 \sim 8$ trains respectively), and upon 10 trials (for the cases of 9 and 10 trains respectively)[4].

[4]Note the 100 trials are executing the completely same data 100 times.

According to Table I, when total number of trains on track is small, online state reachability verification may reach the fixpoint of the state space quickly as the state space is over approximated in the computation. Therefore, we can see that when the number of trains is less than 6, the state reachabililty can catch fault-prediction deadline of $D = 250$ milliseconds (see Section II-C). However, due to the composition explosion problem, the search space explodes quickly which makes the verification of large system impractical. Therefore, when the total number of trains exceeds 6, the time costs quickly soar up. With 10 trains, even by exploiting parallel computing, the time cost exceeds 1 hour. This makes online model checking infeasible[5]. Note in reality, all trains in a railway system may affect each other directly or indirectly. This is why even when model checking two (or one) train(s), the total number of trains in the system affects model checking time cost.

We do notice there is series of reachability checkers for hybrid automata available, including SpaceEx, dReach [37], Flow* [20], CORA [3], Hylaa [9] and so on. The reason we use SpaceEx in the experiment is that among all these tools, SpaceEx supports the verification of LHA naturally as it integrates the mechanism of PHAVer. On the other hand, dReach, Flow* and CORA are targeting on nonlinear hybrid system, while Hylaa handles discrete time hybrid system, which are different with the class of system studied in this work. Meanwhile, we do not intent to compare with SpaceEx here. We list the runtime data of SpaceEx only to illustrate how difficult it is to conduct online state reachability verification.

C. Online Scenario Reachability Validation Model Checking

Indeed, via experience, industry knows the impracticality of state reachability verification for long time, hence does not insist on completeness of model checking (i.e. finding all reachable unsafe states). Therefore, in practice, industry often only demands to know if a CPS can reach certain unsafe states in certain work mode via certain sequence of actions. Using the formal terms proposed in Section III, this is a scenario reachability validation problem. As long as the concerned unsafe scenarios are validated to be unreachable, sufficient confidence is built to allow the CPS to run.

Specifically, instead of pursuing complete guarantees of safety rule **R1** and **R2** (see Section II-A), our CBTC CPS industry collaborators are only concerned with some special scenario reachabilities for **R1** and **R2**.

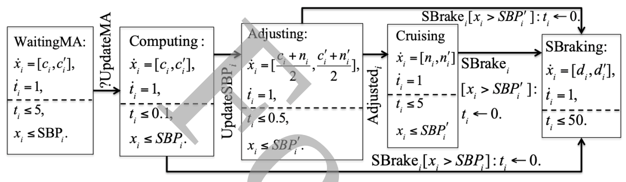
1) Online Composed Scenario Reachability Validation Model Checking for **R1***:* The concerned scenario is: after sync with RBC, all the trains will receive the "UpdateMA" event, and start to run under the new control parameters, whether Train_i ($i = 1, \ldots, m-1$) will hit the train ahead of it (i.e. Train_{i+1}) during their "EBraking" work modes within the online model checking time horizon (see Fig. 5).

The detail scenario is described in Section III-B. We use BACH$_{OL}$ to check the above **R1** related composed scenario reachability validations. The time costs are summarized by Table II. Note same as Table I, in Table II, the time cost refers

[5]The time costs are recorded by the Linux system command : time, with the 1ms granularity.

TABLE I
ONLINE STATE REACHABILITY VERIFICATION MODEL CHECKING TIME COST (DEFAULT UNIT: MILLISECOND)

total no. of trains	2	3	4	5	6	7	8	9	10
R1.mean	11.96	15.32	22.8	38.64	103.96	621.04	7077	285768.4	$>$ 1h
R1.max	20	20	28	44	108	680	7792	307636	$>$ 1h
R1.std	2	1.79	2.37	2.67	3.23	12.72	127.91	22089.34	N/A
R2.mean	11.68	15.32	23	38.72	104.16	621.08	7092.88	281737.6	$>$ 1h
R2.max	20	16	24	48	108	668	7480	313124	$>$ 1h
R2.std	1.93	1.6	1.73	2.32	3.53	12.18	103.88	24501.6	N/A



WaitingMA: $\dot{x}_i = [c_i, c'_i]$, $\dot{t}_i = 1$, $t_i \le 5$, $x_i \le \mathrm{SBP}_i$.
?UpdateMA
Computing : $\dot{x}_i = [c_i, c'_i]$, $\dot{t}_i = 1$, $t_i \le 0.1$, $x_i \le SBP_i$.
UpdateSBP$_i$
Adjusting: $\dot{x}_i = [\frac{c_i + n_i}{2}, \frac{c'_i + n'_i}{2}]$, $\dot{t}_i = 1$, $t_i \le 0.5$, $x_i \le SBP'_i$.
Adjusted$_i$
Cruising $\dot{x}_i = [n_i, n'_i]$ $\dot{t}_i = 1$ $t_i \le 5$ $x_i \le SBP'_i$
SBrake$_i$ $[x_i > SBP'_i]$: $t_i \leftarrow 0$.
SBraking: $\dot{x}_i = [d_i, d'_i]$, $\dot{t}_i = 1$, $t_i \le 50$.
SBrake$_i[x_i > SBP'_i] : t_i \leftarrow 0$.
SBrake$_i[x_i > SBP_i] : t_i \leftarrow 0$.

Fig. 8. Scenario for **R2**

to the time cost to check if one pair of neighboring trains (e.g. Train_1 and Train_2) will hit each other. Checking other pairs of trains can be conducted in parallel independently. The statistics (i.e. mean, max, std) are conducted upon 100 trials.

According to Table II, by exploiting parallel computing, even when a railway track has 20 simultaneously operating trains[6], all online composed scenario reachability validation model checking can finish within 250 milliseconds. This is far less than the time cost for online state reachability verification model checking (see Table I), and less than the online model checking deadline $D = 250$ms (see Section II-C). We also make an investigation to see how many checking missed the deadline. As the longest time used is 184ms for the case of 20 trains, which is still smaller than $D = 250$ms, there is no miss of deadline in the experiments. Clearly, we can further enhance the number of trains that can be solved in the 250 ms time limitation. The result confirms our technique can handle such system efficiently and this also evaluates the completeness of our approach in practice.

Note when model checking every pair of neighboring trains, Theorem 1 still requires us to list linear constraints **C1** $\sim$ **C8**, which are related to all trains. Hence time cost still increases as total number of trains increases. This reflects the reality that in a railway system, all trains on the track are affecting each other directly or indirectly.

2) Online Scenario Reachability Validation Model Checking for **R2***:* **R2** asks whether a train can exceed its EOA during SBraking. As the specification is about the position of every single train respectively, in our scenario-oriented reachability checking, we can simplify the problem to the verification of one automaton, instead of compositional verification. The concerned scenarios of **R2** are illustrated by Fig. 8, which is derived from Fig. 2(A) directly.

- Scenario 1: We are concerned that after synchronization with RBC and receiving an "UpdateMA" event, whether Train_i ($i = 1, \ldots, m$) can pass SBP_i in work mode "Computing" and then exceed EOA. The corresponding scenario $\rho_{1,i}$ is $\left\langle \begin{matrix}\text{WaitingMA}\\ \delta_{i,0}\end{matrix} \right\rangle \xrightarrow{\text{UpdateMA}} \left\langle \begin{matrix}\text{Computing}\\ \delta_{i,1}\end{matrix} \right\rangle \xrightarrow{\text{SBrake}_i} \left\langle \begin{matrix}\text{SBraking}\\ \delta_{i,2}\end{matrix} \right\rangle$.
- Scenario 2: We are concerned that after synchronization with RBC and receiving an "UpdateMA" event, whether Train_i ($i = 1, \ldots, m$) can pass SBP_i in work mode "Adjusting" and then exceed EOA. The corresponding scenario $\rho_{2,i}$ is $\left\langle \begin{matrix}\text{WaitingMA}\\ \delta_{i,0}\end{matrix} \right\rangle \xrightarrow{\text{UpdateMA}} \left\langle \begin{matrix}\text{Computing}\\ \delta_{i,1}\end{matrix} \right\rangle \xrightarrow{\text{UpdateSBP}_i} \left\langle \begin{matrix}\text{Adjusting}\\ \delta_{i,2}\end{matrix} \right\rangle \xrightarrow{\text{SBrake}_i} \left\langle \begin{matrix}\text{SBraking}\\ \delta_{i,3}\end{matrix} \right\rangle$.
- Scenario 3: We are concerned that after after synchronization with RBC and receiving an "UpdateMA" event, whether Train_i ($i = 1, \ldots, m$) can pass SBP_i in work mode "Cruising" and then exceed EOA. The corresponding scenario $\rho_{3,i}$ is $\left\langle \begin{matrix}\text{WaitingMA}\\ \delta_{i,0}\end{matrix} \right\rangle \xrightarrow{\text{UpdateMA}} \left\langle \begin{matrix}\text{Computing}\\ \delta_{i,1}\end{matrix} \right\rangle \xrightarrow{\text{UpdateSBP}_i} \left\langle \begin{matrix}\text{Adjusting}\\ \delta_{i,2}\end{matrix} \right\rangle \xrightarrow{\text{Adjusted}_i} \left\langle \begin{matrix}\text{Cruising}\\ \delta_{i,3}\end{matrix} \right\rangle \xrightarrow{\text{SBrake}_i} \left\langle \begin{matrix}\text{SBraking}\\ \delta_{i,4}\end{matrix} \right\rangle$.

Note the event guards and resets are omitted due to space limit. Interested readers can refer to the corresponding edges in Fig. 2(A).

There are no composed scenarios of interest of **R2**. Therefore the $3 \times m$ scenarios $\rho_{1,i}$, $\rho_{2,i}$, $\rho_{3,i}$ ($i = 1, \ldots, m$) respectively corresponds to $3 \times m$ independent scenario reachability validation problems that can be computed in parallel. For a scenario reachability validation problem related to $\rho_{1,i}$, $\rho_{2,i}$, or $\rho_{3,i}$ ($i = 1, \ldots, m$), the reachability specification constraint φ_i is $x_i > \mathrm{EOA}_i$ and $t \leqslant 50$, where x_i is Train_i's position on track, t is the time, and 50 is the concerned online model checking time horizon length F.

Again we use our BACH_{OL} model checker implementation to check the above **R2** related scenario reachability validations. The time costs are summarized by Table III. Note in the table, the time cost refers to the time cost to model check if one train (e.g. Train_1) will exceed its EOA in the corresponding scenario. Model checking of other scenarios, and of other trains can be conducted in parallel independently. The statistics (i.e. mean, max, std) are conducted upon 100 trials.

According to Table III, by exploiting parallel computing, all scenario reachability validations can be model checked with 40 milliseconds, which is way less than the online model checking deadline $D = 250$ milliseconds (see Section II-C), and of course there is no miss of deadline in the experiments.

V. RELATED WORK

Online Monitor and Verification. Similar to online model checking, "runtime verification" [29] is also performed while the system is running. However, the purpose of runtime

[6]This is a normal number of running trains that can be effectively monitored by an RBC simultaneously on a line for an urban subway system.

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.2019.2935062, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE II
ONLINE COMPOSED SCENARIO REACHABILITY VALIDATION MODEL CHECKING TIME COST FOR **R1** (DEFAULT UNIT: MILLISECOND)

total no. of trains	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
mean	34.84	41.12	48.24	55.68	63.36	71	75.72	83.44	90.4	98.4	104.44	118.12	125.68	133.2	141.48	150.04	156.76	163.84	172.92
max	40	48	52	60	72	76	84	88	96	112	112	124	132	144	152	160	168	172	184
std	3.41	3.63	3.74	4.26	4.82	4.05	4.6	4.42	4.93	5.22	4.52	4.51	5.13	5.67	5.76	5.32	5.68	5.85	5.45
miss deadline	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TABLE III
ONLINE SCENARIO REACHABILITY VALIDATION MODEL CHECKING TIME COST FOR **R2** (DEFAULT UNIT: MILLISECOND)

Scenario	$\rho_{1,i}$	$\rho_{2,i}$	$\rho_{3,i}$
mean	26.04	25.68	31.54
max	32	36	40
std	2.68	3.84	4.48
miss deadline	0	0	0

verification is to retrieve program execution traces for offline debugging/analysis, rather than online fault prediction.

Easwaran et al. [28] propose an online steering architecture for discrete systems. The architecture monitors the runtime behavior of the system and checks whether the event generated by the system will violate certain user-defined properties in bounded steps. However, this architecture is for discrete systems instead of CPS.

As a special subclass of the parametric hybrid automata, which is discussed in this paper, the scheduling analysis of parametric timed automata is studied in [23] and [39] to evaluate the parameters space that guarantees a feasible schedule. These works perform the analysis by concretizing every combination of parameter values repeatedly to obtain the scheduability region.

Several works [10], [19], [21], [42], [46] propose to use online state reachability model checking as a CPS fault prediction tool. However, how to deal with value-unknown parameters, online data exchange, and computational time cost scalability are problems yet to be explored. This paper makes an initial attempt to address these problems holistically. It generalizes the solution of [17] [18] [43] by adding support for modeling offline value-unknown parameters and online data exchange, and by proposing a more rigorous theoretical framework based on the PHA execution trace concept.

Verification of Hybrid System. The verification for hybrid systems is very difficult. Even for a relatively simple class of hybrid systems – linear hybrid automata – the reachability analysis problem is also undecidable [4], [34], [36]. The main technique under the classical reachability problem is trying to abstract the state space by certain mathematic methods like polyhedral [7], ellipsoidal [38], support functions [40], and then compute the transitive closure of the state space of the system behavior by techniques like polyhedral computation, which is very expensive and not guaranteed to terminate. Several model checking tools have been developed, for example PHAVer [31] and its improvement SpaceEx [32]. But they do not scale well to the size of practical problems.

In recent years, *Bounded Model Checking* [12] has been presented as a technique alternative to BDD-based symbolic model checking. There are several related works [8], [30] to check linear hybrid systems by BMC. Several tools were developed, such as MathSAT [8] and HySAT [30]. These tools are based on a SAT-solver that calls the solver on demand for conjunctions of the domain-specific constraints. Nevertheless, the experiment results show that it is difficult to apply those tools to analyze problems of practical size. The performance is even worse for reachability analysis of a composition of several linear hybrid automata.

Study [17] presented a path-oriented method for composed verification of LHA which is similar to this work. However, this work provides a more rigorous and generic theoretical framework, exploiting the PHA execution trace concept. This work is also more generic than [17] in the sense that our framework supports modeling offline value-unknown parameters and online data exchange. On the other hand, dReach [37] is a hybrid automata model checker which also conduct path-oriented reachability verification. The main decision procedure under dReach is interval analysis which supports nonlinear constraint solving quite well. However, dReach only supports scenario reachability of a single automaton. It will be an interesting topic to see how to integrate the composed scenario encoding method proposed in this work with dReach to handle the verification of composed nonlinear hybrid systems.

Verification of Parametric Hybrid System. The language of parametric hybrid automata introduced in this paper is inspired by the slope parametric linear hybrid automata (SPLHA) proposed and analyzed in studies including [1], [2], [15]. In these works, the flow conditions, a.k.a. slope, of a continuous variable could be free parameters. However, in real case systems, all the control variables could be parametric, not necessarily only the slope variable. Meanwhile, composed systems may exchange data along with synchronization events, therefore, we support the compositional modeling and value exchanging. On the other hand, existing verification studies of SPLHA mainly focused on the synthesis of the slope variables to make the system satisfy the desired property. Differently, our work does not focus on parameter synthesis, we conduct online verification to see whether the runtime control parameters are safe or not in the short-run future.

Similar with our work, parametric timed automata (PTA) [5] and its variants [11], [16] were proposed to model the real time system working in open environment by introducing parameters in guards and invariants. Timed automata is a special class of linear hybrid automata. Therefore, the express ability of the PHA presented in this paper is more powerful than PTA. Most of the verification studies of PTA are also focusing on parameter synthesis. Meanwhile, there are also other properties of PTA studied, including reachability, unavoidability and so on. However, most of the non-trivial problems on PTA are proven to be undecidable [6].

Besides of these works about verification of parameterized hybrid system, there are many works conducted on the verification of parametrized system, especially in the area of distributed or concurrent systems. Study [13] gives a survey of literatures on this topic. The system considered in [13] assumes the model for each component is concrete with a finite state space, while the parameterized aspect mostly come from the number of active components of the system. Differently, the PHA considered in this paper is more complicate than the class of system surveyed in [13] . First of all, the state space of hybrid automata is not finite. Then, the model of each component has also free parameters included.

Verification of Train Control System. The verification of train control system has been intensively studied. Chiappini et al. [22] propose a method to generate high level requirements from a subset of specifications in an ETCS system, and use the method of [24] to verify the consistency between requirements. These two works belong to the category of requirement engineering, which do not touch real-time.

Bohn et al. [14] model the communication in train control systems with Live Sequence Chart (LSC), then validate the LSC by model checking and testing. Peleska et al. [48] model the behavior of train control systems by timed state transition systems, then verify given properties by BMC and compositional reasoning. These works focus on the high level discrete models instead of hybrid models.

Platzer et al. [49] model an ETCS system with differential dynamic logic and verify the system by logical deductive verification. Damm et al. [27] build different models for different layers of an ETCS system and verify these models using layer-specific technologies. These works focus on static instead of online models of the ETCS systems.

VI. Conclusion

We proposed the concept of PHA to model CPSs with offline value-unknown parameters and online data exchanges. Online model checking of PHAs can work as an online fault prediction tool for CPS. The corresponding time cost challenge is addressed by our proposed scenario reachability validation framework, which exploits the industry demand that incomplete model checking is acceptable. The framework supports scenario compose/decompose and the resulted online scenario reachability validation can be conducted by linear programming, hence incurs polynomial time cost. Evaluation upon a real-world train control system shows that our proposed approach cuts fault prediction time from over 1 hour to within 200 milliseconds. This makes online operation possible.

Acknowledgment

The authors would like to thank anonymous reviewers and Dr. Yao Chen for their valuable comments.

References

[1] M. Adélaïde and O. F. Roux. Using cylindrical algebraic decomposition for the analysis of slope parametric hybrid automata. In *Proceedings of 6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2000*, pages 252–263, 2000.

[2] M. Adélaïde and O. F. Roux. A class of decidable parametric hybrid systems. In *Proceedings of AMAST 2002*, pages 132–146, 2002.

[3] M. Althoff. An introduction to CORA 2015. In *Proceedings of 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems, ARCH@CPSWeek 2015.*, pages 120–151, 2015.

[4] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[5] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 592–601, 1993.

[6] É. André. What's decidable about parametric timed automata? In *Proceedings of Fourth International Workshop of Formal Techniques for Safety-Critical Systems, FTSCS 2015. Revised Selected Papers*, pages 52–68, 2015.

[7] E. Asarin, O. Bournez, T. Dang, and O. Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In *Proceedings of HSCC'00*, pages 20–31. Springer, 2000.

[8] G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani. Verifying industrial hybrid systems with mathsat. *Electronic Notes in Theoretical Computer Science*, 119(2):17–32, 2005.

[9] S. Bak and P. S. Duggirala. Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In *Proceedings of HSCC 2017*, pages 173–178, 2017.

[10] S. Bak, T. T. Johnson, M. Caccamo, and L. Sha. Real-time reachability for verified simplex design. In *Proceedings of Real-Time Systems Symposium, RTSS'14*, pages 138–148. IEEE, 2014.

[11] B. Bérard, S. Haddad, A. Jovanovic, and D. Lime. Parametric interrupt timed automata. In *Reachability Problems - 7th International Workshop, RP 2013, Uppsala, Sweden, September 24-26, 2013 Proceedings*, pages 59–69, 2013.

[12] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in computers*, 58:117–148, 2003.

[13] R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. Decidability in parameterized verification. *SIGACT News*, 47(2):53–64, 2016.

[14] J. Bohn, W. Damm, J. Klose, A. Moik, H. Wittke, H. Ehrig, B. Kramer, and A. Ertas. Modeling and validating train system applications using statemate and live sequence charts. In *Proc. IDPT*, 2002.

[15] F. Boniol, A. Burgueño, O. F. Roux, and V. Rusu. Analysis of slope-parametric hybrid automata. In *Proceedings of International Workshop of Hybrid and Real-Time Systems. HART'97*, pages 75–80, 1997.

[16] L. Bozzelli and S. La Torre. Decision problems for lower/upper bound parametric timed automata. *Formal Methods in System Design*, 35(2):121–151, 2009.

[17] L. Bu and X. Li. Path-oriented bounded reachability analysis of composed linear hybrid systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 13(4):307–317, 2011.

[18] L. Bu, Y. Li, L. Wang, and X. Li. Bach: Bounded reachability checker for linear hybrid automata. In *Proceedings of the 2008 International Conference on Formal Methods in Computer-Aided Design*, page 9. IEEE Press, 2008.

[19] L. Bu, Q. Wang, X. Chen, L. Wang, T. Zhang, J. Zhao, and X. Li. Toward online hybrid systems model checking of cyber-physical systems' time-bounded short-run behavior. *ACM SIGBED Review*, 8(2):7–10, 2011.

[20] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Proceedings of CAV 2013*, pages 258–263, 2013.

[21] X. Chen and S. Sankaranarayanan. Model predictive real-time monitoring of linear systems. In *Proceedings of IEEE Real-Time Systems Symposium, RTSS 2017*, pages 297–306, 2017.

[22] A. Chiappini, A. Cimatti, L. Macchi, O. Rebollo, M. Roveri, A. Susi, S. Tonetta, and B. Vittorini. Formalization and validation of a subset of the european train control system. In *Proceedings of ICSE'2010*, volume 2, pages 109–118. IEEE, 2010.

[23] A. Cimatti, L. Palopoli, and Y. Ramadian. Symbolic computation of schedulability regions using parametric timed automata. In *Real-Time Systems Symposium, 2008*, pages 80–89. IEEE, 2008.

[24] A. Cimatti, M. Roveri, and S. Tonetta. Requirements validation for hybrid systems. In *Proceedings of CAV'09*, volume 5643, pages 188–203. Springer, 2009.

[25] E. M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT press, 1999.

[26] E. M. Clarke, B. Krogh, A. Platzer, and R. Rajkumar. Analysis and verification challenges for cyber-physical transportation systems. 2008.

[27] W. Damm, A. Mikschl, J. Oehlerking, E.-R. Olderog, J. Pang, A. Platzer, M. Segelken, and B. Wirtz. Automating verification of cooperation, control, and design in traffic applications. In *Formal Methods and Hybrid Real-Time Systems*, pages 115–169. Springer, 2007.
[28] A. Easwaran, S. Kannan, and O. Sokolsky. Steering of discrete event systems: Control theory approach. *Electronic Notes in Theoretical Computer Science*, 144(4):21–39, 2006.
[29] B. Finkbeiner, S. Sankaranarayanan, and H. B. Sipma. Collecting statistics over runtime executions. *Formal Methods in System Design*, 27(3):253–274, 2005.
[30] M. Franzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:209–236, 2007.
[31] G. Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In *Proceedings of HSCC'05*, volume 5, pages 258–273. Springer, 2005.
[32] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. Spaceex: Scalable verification of hybrid systems. In *Proceeding os Computer Aided Verification, CAV'11*, pages 379–395. Springer, 2011.
[33] E. U. Group. Unisig: Ertms/etcs system requirements specification. http://www.era.europa.eu, 2002.
[34] T. A. Henzinger. The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*, pages 265–292. Springer, 2000.
[35] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. In *International Conference on Computer Aided Verification, CAV'97*, pages 460–463. Springer, 1997.
[36] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 373–382. ACM, 1995.
[37] S. Kong, S. Gao, W. Chen, and E. M. Clarke. dreach: δ-reachability analysis for hybrid systems. In *Proceeding of TACAS 2015*, pages 200–205, 2015.
[38] A. A. Kurzhanskiy and P. Varaiya. Ellipsoidal techniques for reachability analysis of discrete-time linear systems. *IEEE Transactions on Automatic Control*, 52(1):26–38, 2007.
[39] T. T. H. Le, L. Palopoli, R. Passerone, and Y. Ramadian. Timed-automata based schedulability analysis for distributed firm real-time systems: a case study. *International Journal on Software Tools for Technology Transfer*, pages 1–18, 2013.
[40] C. Le Guernic and A. Girard. Reachability analysis of hybrid systems using support functions. In *Proceedings of CAV'09*, volume 5643, pages 540–554. Springer, 2009.
[41] E. A. Lee. Cyber-physical systems-are computing foundations adequate. In *Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*, volume 2, 2006.
[42] T. Li, F. Tan, Q. Wang, L. Bu, J.-n. Cao, and X. Liu. From offline toward real time: A hybrid systems model checking and cps codesign approach for medical device plug-and-play collaborations. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):642–652, 2014.
[43] X. Li, S. J. Aanand, and L. Bu. Towards an efficient path-oriented tool for bounded reachability analysis of linear hybrid systems using linear programming. *Electronic Notes in Theoretical Computer Science*, 174(3):57–70, 2007.
[44] P. O. Meredith, D. Jin, D. Griffith, F. Chen, and G. Rosu. An overview of the MOP runtime verification framework. *STTT*, 14(3):249–289, 2012.
[45] W. G. Najm, J. D. Smith, and M. Yanagisawa. Pre-crash scenario typology for crash avoidance research. In *DOT HS*. Citeseer, 2007.
[46] L. V. Nguyen, C. Schilling, S. Bogomolov, and T. T. Johnson. Runtime verification for hybrid analysis tools. In *Proceedings of 6th International Conference on Runtime Verification, RV 2015*, pages 281–286, 2015.
[47] B. Ning, T. Tang, K. Qiu, C. Gao, and Q. Wang. Ctcschinese train control system. *WIT Transactions on The Built Environment*, 74, 2004.
[48] J. Peleska, D. Große, A. E. Haxthausen, and R. Drechsler. Automated verification for train control systems. *Proceedings of FORMS/FORMAT*, pages 252–265, 2004.
[49] A. Platzer and J.-D. Quesel. European train control system: A case study in formal verification. In *Proceedings of ICFEM'09*, volume 5885, pages 246–265. Springer, 2009.
[50] L. Sha, S. Gopalakrishnan, X. Liu, and Q. Wang. Cyber-physical systems: A new frontier. In *Proceedings of SUTC'08*, pages 1–9. IEEE, 2008.
[51] B. I. Silva, O. Stursberg, B. H. Krogh, and S. Engell. An assessment of the current status of algorithmic approaches to the verification of hybrid systems. In *Proceedings of the 40th IEEE Conference on Decision and Control, 2001*, volume 3, pages 2867–2874. IEEE, 2001.

Lei Bu is an associate 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.

Qixin Wang received the BE and ME degrees from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 1999 and 2001, respectively, and the PhD degree from the Department of Computer Science of the University of Illinois at Urbana-Champaign in 2008. He joined the Department of Computing of the Hong Kong Polytechnic University in 2009 as an assistant professor, and is currently an associate professor. His main research interests include cyber-physical systems, real-time and embedded systems, and computer networks. He is a member of the IEEE and the ACM.

Xinyue Ren received the BSc degree from Dalian University of Technology, in 2016. She received masters degree from Nanjing University in 2019. Her research interest mainly focus on verification of composed linear hybrid automata.

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 full 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.