Leveraging System Dynamics in Runtime Verification of Cyber-Physical Systems*

Houssam Abbas[1] and Borzoo Bonakdarpour[2]

[1] Oregon State University, Corvalis, OR, USA, houssam.abbas@oregonstate.edu
[2] Michigan State University, East Lansing, USA, borzoo@msu.edu

Abstract. Cyber-physical systems consist of control software systems that interact with physical components that obey the fundamental laws of physics. It has been long known that exhaustive verification of these systems is a computationally challenging problem and distribution makes the problem significantly harder. In this paper, we advocate for runtime verification of cyber-physical systems and layout a road map for enhancing its effectiveness and efficiency by exploiting the knowledge of dynamics of physical processes.

1 Introduction

Cyber-physical systems (CPS), the Internet of Things (IoT), and edge applications appear in many different applications in our daily lives. CPS generally have safety-critical nature and as in other such systems, *runtime verification* (RV) is a complementary approach to static verification and testing in order to gain assurance about the health of the system. CPS is particularly becoming *distributed* over networks of *agents*, e.g., in sensors in infrastructures, health-monitoring wearables, networks of medical devices, and autonomous vehicles. Moreover, CPS and IoT often have to deal with resource constraints and any improvement in minimizing resource utilization and consumption is in pressing need.

While there have been proposals for monitoring CPS [2,4,5], predictive monitoring [3], monitoring using worst-case bounds [6], mitigating the effects of timing inaccuracies [12,1], and for minimally intrusive CPS monitoring [9], to our knowledge, the work on distributed RV and exploiting system dynamics to reduce the overhead of RV is limited to [11]. System dynamics is especially an interesting feature of CPS, as physical processes are assumed to obey the laws of physics. For example, a car cannot accelerate from $1 - 150 km/h$ in only one second. Knowing this could significantly assist in designing runtime monitors that monitor the state of the system while ignoring scenarios that cannot be reached.

Our goal in this paper is to explore the idea of combining RV with a precomputed knowledge of system dynamics to enhance the effectiveness of RV and reduce its runtime overhead. To this end, we introduce a set of ideas on the types

* This work is sponsored by the NSF CCF Award 2118356.

of dynamics knowledge that can be available and how it might be leveraged by a runtime monitor, in the centralized and distributed setups. Specifically, we look at:

- Types of dynamics knowledge: from simple bounds on the state changes to full knowledge of the dynamics but only over a short amount of time.
- Different system models: we consider continuous-time and discrete-time models in a unified manner, to maintain maximum applicability of the monitors.
- Different ways to leverage the dynamics: from skipping parts of the signal where nothing interesting can happen, to early termination of the monitoring process.

We also discuss that the impact of the knowledge of the dynamics when the monitor is distributed over the agents, with each agent knowing only its own dynamics, and show how access to information about offline model-checking runs on the model can be leveraged.

The Model-Based Design Cycle and Runtime Verification Knowledge of the dynamics, and offline formal verification, are possible in the model-based design cycle, in which a model of the system-under-development is created, refined and ultimately synthesized. Thus, by construction, the system can predict its own future, up to a certain horizon, and up to a certain accuracy. This knowledge, available by construction, can accelerate or improve the accuracy of online RV.

Organization. The rest of the paper is organized as follows. In Section 2, we present the preliminary concepts. Section 3 discusses the role of system dynamics in improving the runtime overhead. We discuss the distributed setting and its challenges in Section 4 and the impact having access to verified dynamics in Section 5. Finally, we conclude in Section 6.

2 Preliminary Concepts

First, we set some notations. The set of reals is $\mathbb{R}$, the set of non-negative reals is $\mathbb{R}_+$, and the set of positive reals is $\mathbb{R}_+^*$. The set of integers $\{1, \ldots, N\}$ is abbreviated as $[N]$.

2.1 Signal Model

In this section, we review our signal model, i.e., our model of the output signal of an agent from [11].

Definition 1. *An* output signal *is a function $x : I \to \mathbb{R}^d$, which is right-continuous, left-limited, and is not Zeno. Here, I is an interval in $\mathbb{R}_+$ (which could be all of $\mathbb{R}_+$), and will be referred to as the* timeline *of the signal.* ■

Right-continuity means that at all t in its support, $\lim_{s\to t_+} x(s) = x(t)$. *Left-limitedness* means the function has a finite left-limit at every t in its support: $\lim_{s\to t_-} x(s) < \infty$. *Not being Zeno* means that x has a finite number of discontinuities in any bounded interval in its support. This ensures that the signal cannot jump infinitely often in a finite amount of time. A *discontinuity* in a signal $x(\cdot)$ can be due to a discrete event internal to agent A (like a variable updated by software), or to a message sent to or received from another agent A'.

2.2 Signal Temporal Logic (STL) [8]

Let AP be a set of *atomic propositions*. The syntax for *signal temporal logic* (STL) is defined for infinite traces using the following grammar:

$$\varphi := \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \,\mathcal{U}_{[a,b]}\, \varphi$$

where $\top$ is the logical constant standing for `true`, $p \in \mathsf{AP}$ and $\mathcal{U}$ is the 'until' temporal operator. We view other propositional and temporal operators as abbreviations, that is, $\bot = \neg\top$ (`false`), $\Diamond_{[a,b]}\,\varphi = \top\,\mathcal{U}_{\,[a,b]}\varphi$ (*eventually*), $\Box_{[a,b]}\,\varphi = \neg(\Diamond_{[a,b]}\,\neg\varphi)$ (*always*). We denote the set of all STL formulas by Φ_{STL}.

Let a *trace* $\sigma = (x_1, \ldots, x_N)$ be a set of N signals with common support $\mathbb{R}_+$. Thus, a trace is, by definition, of infinite duration. This models that all agents are executing concurrently, and that time does not stop. Even if the agent does nothing over some interval $[a, b]$, its physical output signal still has a value. This models the output of a set of N agents operating concurrently. Each signal might be d-dimensional. For simplicity of presentation and without loss of generality in this paper, we assume $d = 1$.

To every atom p in AP is associated an N-ary function $f_p : \mathbb{R}^N \to \mathbb{R}$.

Let $\models$ denote the *satisfaction* relation. We write $\sigma, t \models \varphi$ to signify 'the infinite trace σ satisfies formula φ starting at time t'. Since the signals share a support, time t is in that shared support. Satisfaction is formally defined as follows.

$$\begin{array}{lll}
(\sigma,t) \models \top & & \text{(no condition, } \top \text{ is satisfied by all traces at all times)} \\
(\sigma,t) \models p & \text{iff} & f_p(x_1(t), \ldots, x_n(t)) > 0 \\
(\sigma,t) \models \varphi \wedge \psi & \text{iff} & (\sigma,t) \models \varphi \wedge (\sigma,t) \models \psi \\
(\sigma,t) \models \neg\varphi & \text{iff} & \neg((\sigma,t) \models \varphi) \\
(\sigma,t) \models \varphi\mathcal{U}_{\,[a,b]}\psi & \text{iff} & \exists t' \in [t+a, t+b] : (\sigma,t') \models \psi \wedge \\
 & & \forall t'' \in [t,t') : (\sigma,t'') \models \varphi
\end{array}$$

Note that for the Until operator, formula φ is required to hold starting at the evaluation moment t up to and excluding the 'hand-over' moment t'. When $t = 0$ we write $\sigma \models \varphi$ instead of $(\sigma, 0) \models \varphi$. For example, given the trace σ shown in Fig. 1, the STL formula $\varphi = p\mathcal{U}_{\,[4,6.5]}q$ holds at time 0, that is, $\sigma \models \varphi$. However, φ does not hold after time 2, as in that case, q must be observed after time $2+4$ and before $2 + 6.5$, which does not happen.

A subtlety of the online monitoring setup is that only a finite-duration segment of a trace is available at any given time, not an infinite trace. This is handled

in the classical manner: given a finite duration signal x, we say it satisfies/violates φ iff every extension $(x.y)$, where y is an infinite signal, satisfies/violates φ. Otherwise, the monitor returns Unknown. Here, the dot '.' denotes concatenation in time.

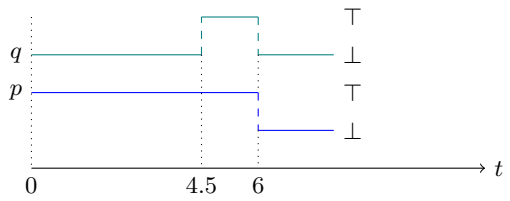

Fig. 1: A trace σ generated by a system.

3 Exploiting Knowledge of System Dynamics

RV first emerged as a set of techniques for monitoring a signal against a specification, without paying any attention to the dynamics that generates the signal. This was deliberate: the setup imagined by RV pioneers was one in which a model of the systems was not available, or was available but too complicated for exhaustive verification (i.e., model checking). The advantage of ignoring the signal-generating dynamics is that the resulting monitor is maximally applicable: all it needs is the signal to monitor. It might also be thought that this creates a faster monitor, since taking signal dynamics into account might add a computational burden.

In this section, we argue that this is not always the case – indeed, we argue that exploiting available knowledge of signal dynamics can reduce the monitor's run time. In a sense, this parallels earlier developments in cyber-physical systems which brought together computer scientists and control theorists to develop model-checking and verification algorithms for hybrid dynamical systems. The new model checking algorithms had to account for the dynamics to verify properties expressed in terms of the state variables.

3.1 Motivating Example

To motivate the benefit of using systems dynamics, we repeat one of our examples from [11] (see Fig. 2). From knowing the rate bound $|\dot{x}| \leq 1$ (shown by a dashed line), the monitor concludes that the earliest x can satisfy the atom $x \geq 3$ is τ_1. Similarly for y. Given that $\tau_1 > \tau_2$, the monitor discards, roughly speaking, the fragment $[0, \tau_2]$ from each signal and monitors the remaining pieces.

In [11] we demonstrated that such dynamics knowledge can significantly reduce the runtime of the monitor. This paper explores other possibilities for exploiting other forms of knowledge.

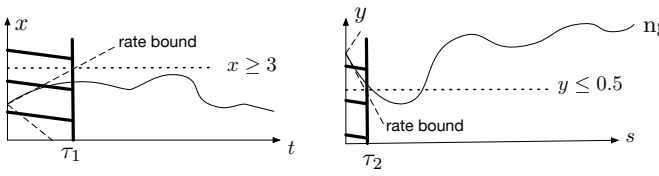

Fig. 2: Leveraging dynamics.

3.2 System Model

We consider signals x generated by a system under observation. A system is modeled as a tuple $\Sigma = (X, I, U, f, H)$, where $X \subseteq \mathbb{R}^d$ is the state space, $I \subset X$ is the set of initial states, $U \subset \mathbb{R}^m$ is the input space, $f : X \times U \to X$ is the transition function, and $H : X \to \mathbb{R}^o$ is the output or observation function.

The model is either an Ordinary Differential Equation (ODE) in continuous time

$$\dot{x}(t) = f(x(t), u(t)),\ x(0) \in I, u(t) \in U, t \in [0, \infty)$$

or a Difference Equation (DEs) in discrete time

$$x(t+1) = f(x(t), u(t)),\ x(0) \in I, u(t) \in U, t \in \mathbb{N}$$

The DE can model, as a special case, finite state automata: in this case f models the transition relation of the automaton. In both cases, u is the input signal, provided by the system's controller; in the discrete-time case, u is of course a (discrete-time) sequence. We write $x \xrightarrow{\overline{u}} x'$ to indicate a transition from x to x' under input signal or sequence $\overline{u}$ of unspecified length.

3.3 Knowledge of Signal Dynamics

In this section, by 'signal dynamics', we mean the dynamics of the model that is generating the monitored signal. E.g., the signal x might be the position of a car which obeys an ODE

$$\dot{x}(t) = v(t)cos(\theta(t)),$$

where $\dot{v}(t) = a(t)$, a is the acceleration input, and $\theta(t)$ is the steering input. Thus the input signal is 2-dimensional: $u = [a, \theta]$.

The monitor, which runs on-board the system, may have different kinds of knowledge about the dynamics:

1. **Bounds on the signal's derivative/difference**: $|\dot{x}(t)| \leq L$ or $|x(t+1) - x(t)| \leq L$. This is the simplest form of knowledge that we consider.
2. **Local model**: To each value of the state, the monitor associates a local approximation of the dynamics. That is, if the measured state at time t is x, the monitor considers that the signal obeys $\dot{x}(t) = f_x(x(t), u(t))$, where f_x is the local model. We do not commit to how f_x and f are related: the former may be a linearization of the latter, or something else entirely. For f_x to be useful, we must assume that its error to f is bounded, namely

$\|f_x(y,u) - f(y,u)\| \leq e(x)$ for all x, y and u. (Note that sometimes f_x does not take the same input as f, in which case we must also assume some mapping from f-inputs to f_x-inputs). Presumably f_x is also faster to simulate than f. Similar equations apply to the DE case.

3. **Simulating model**: In this case, the monitor has access to a dynamical model $\Sigma' = (X_g, I_g, U_g, g, H)$, with state space X_g and control space U_g, which *simulates* f. This means that there is relation R between X_f and X_g, called the *simulation relation*, such that:
   (a) Any two related states have the same output: $H(x_f) = H(x_g)$ for all $(x_f, x_g) \in R$. Thus, an observer cannot tell whether they are observing a state of f or g.
   (b) For every x in I_f there exists a point y in I_g s.t. $(x, y) \in R$ (initial conditions are related).
   (c) For every $(x, y) \in R$ it holds that: $x \xrightarrow{\overline{u}} x'$ in the f-system implies the existence of a transition $y \xrightarrow{\overline{u}'} y'$ in the g-system s.t. $(x', y') \in R$. That is, related states evolve into related states.

   We say g *simulates* f. Note that every f-behavior has a corresponding g-behavior, but not necessarily the other way around.
   The notion of ϵ-simulation generalizes the above definition by requiring bounded error between related states rather than equality. That is, $\|H(x_f) - H(x_g)\| \leq \epsilon$ for all $(x_f, x_g) \in R$
4. **Short model**: the monitor has access to the 'true' model f but can only execute it for a short amount of time. E.g., it can only solve the ODE up to time 5, or it can only see 5 hops ahead in the automaton.

We do not consider the case where the monitor has access to the true model f and can execute it to any length since then, the monitor can predict exactly the future trajectory and monitor it in a dynamics-agnostic manner.

Of course, there may be combinations of the above types of knowledge, e.g., a short local model.

3.4 Using Partial Knowledge

How might the monitor use these kinds of partial knowledge? And why would that reduce the monitoring overhead? By default, we present all examples for continuous-time signals. Obvious modifications can be made for the DE case, unless otherwise indicated. Our approach in this section will be to start from simple concrete examples which clarify the basic idea and technique to leverage dynamics in runtime monitoring. We then raise some of the challenges that will arise when we try to generalize, and suggest ways forward.

Early Termination: the most concrete idea, and one which can be readily implemented on top of existing software, is to use the dynamics knowledge to potentially terminate monitoring early. For instance, consider the atom $p := (x \geq 0)$ and suppose the formula is $\Diamond p$. So far no p has been observed, and at

the present moment t, $x(t) = -5$. If the monitor knows that $\dot{x}(s) \leq -1$ from here on out ($s \geq t$) then it can immediately declare a violation since x will never go positive.

The same principle applies if the monitor has a local model with a known deviation e to the true dynamics. The local bound provides a more refined bound, one that is not uniform, thus reducing the conservatism of the answer, but at the cost of solving the local model's time-varying ODE.

A simulating model is also useful for achieving the same task if the simulation relation meets a certain property. Namely, the state space of g, X_g, is obtained by partitioning X, where the states in each part all satisfy a particular combination of atoms. Formally, X_g is the quotient set of X by the equivalence relation

$$x \equiv y \text{ iff for all atomic propositions } p,\ x \models p \text{ iff } y \models p.$$

We say that the simulation *respects* AP. If the simulation relation respects AP, then it is enough to execute g to get a peek at the future. This is because g contains all the information necessary to monitor the formula: In such a case, the monitor executes g however long is feasible and tries to predict a violation or satisfaction, if they occur within the prediction horizon.

The above reasoning might be generalized somewhat if we find a notion of approximate satisfaction of an atom, which we then lift to the notion of a simulation relation that approximately respects AP. This might coincide with the notion of *approximate simulation* introduced in 2. Approximate simulations are well-known in the hybrid systems literature and have been successfully used to verify the *safety* of complex dynamics by reasoning over the simulating system. For general properties expressed in a temporal logic, this is more complicated as it compounds the errors in complex ways. One possible approach is to study the formula's language directly, and lift the approximate simulation to the languages. It is less clear whether a general simulating model, which does not (approximately) respect AP, can enable early termination.

It is fairly obvious that a short model can help in early termination: just execute the model as long as feasible, say for H time units, and see if the predicted trajectory already satisfies or violates the spec. However, since we assume the execution horizon H to be very short, this will probably waste more energy in trajectory prediction than it saves in early termination, since most predictions won't yield a decisive verdict. An interesting research direction arises if we assume that the monitor can *query* the model f at particular states: meaning that it can simulate f for H units *from a state q of its choosing*, not necessarily the current state $x(t)$. If this is possible, then the monitor can pre-emptively explore the model's behavior around potentially troublesome states from which violations of the formula can arise. Research is needed to define"troublesome states", and to do so in a way that takes into account the current on-going execution and the ways in which the formula can still be violated.

If f is an ODE or DE, the model can be queried in a straightforward fashion by setting the initial condition to q – but the difficulty immediately arises of what input signal to apply to solve the ODE $\dot{x} = f(x, u)$. If the controller is a

state-feedback controller, the answer is trivial, but in general, this is an open question. If f is an automaton, querying it requires communication between the monitor and the system itself (or perhaps the controller), in which the monitor asks to obtain that piece of the automaton which is centered on q; i.e., q and its out-neighbors up to H hops away. Whether this is realistic or not depends on the system architecture: for instance, many systems do not carry a model of their own dynamics at all, or that model is not all available in memory at any given point in time due to memory constraints.

Skipping Parts of the Signal: The same types of knowledge might allow skipping parts of the signal: they are not monitored, thus saving in monitoring energy consumption, or freeing the monitor for other tasks, like monitoring multiple signals in alternating fashion. This is true for instance if the monitor knows bounds $|x(t)| \leq L$. Let's reprise the above example: consider the atom $p := (x \geq 0)$ and suppose the formula is $\Diamond p$. So far no p has been observed, and at the present moment t, $x(t) = -5$. If the monitor knows that $|\dot{x}| \leq 1$ from here on out, then it knows x will remain negative at least for the next 5 seconds. So it can ignore the next 5 seconds of the signal, saving processing power, and start observing again at $t + 5$. The same reasoning applies if the monitor has a local model, the only difference being that we now have possibly less conservative bounds on the derivative and trajectory.

The use of a simulating model to skip signal segments is much less obvious. One of the difficulties is that the simulating model g does not have the same notion of time as the original model f. While every f-transition $x \xrightarrow{u} x'$ has a corresponding transition $y \xrightarrow{u'} y'$ in the g-system, the two transitions do not necessarily take the same amount of time, and in general, there is no known relation between the timelines. In fact, one of the reasons we develop a simulating model is that it can abstract away or accelerate time, allowing us to run faster simulations with it.

The use of a short model is also not obvious, because of the short horizon. It is not clear to us whether adding the ability to query the short model at will helps: questions of where and when to query it must be driven by the formula being monitored and the history of the system trajectory so far, and all of these are challenging research questions.

An additional challenge to all the above methods in the case of a general temporal logic formula is that the monitor must track all the ways the formula might be satisfied or violated, to avoid skipping signal segments that affect the truth value of the formula. While in principle possible (for a finite duration trajectory), it poses a heavy memory burden on the RV module, which needs to be lightweight. A partial workaround is to simply track each atom individually and avoid any truth value changes of the atoms. This is only a partial workaround: as shown in the example, the truth value can change simply due to time passing, without change in the values of any atoms.

4 Monitoring Distributed CPS

If the system under monitoring consists of multiple nodes or agents, each of which has a local clock, then new challenges arise for runtime monitoring. These challenges further complicate our attempts at leveraging system dynamics.

At the highest level, the biggest challenge is that local clocks can and do drift: thus when one agent reports $x_1(5) = 3$ and the other agent reports $x_2(5) = 2$, these are not necessarily truly synchronous readings. I.e., the first clock reads 5 at a different (physical) time than the second clock. Such drift can obviously affect the validity of the monitor's calculations.

The second high-level challenge, which is independent of clock drift, is that each agent only has access, a priori, to its own state. So formulas that couple agents' states, like $(x_1 > 0)\,\mathcal{U}\,(x_2 > 3)$, require communication between agents and exchange of information. Thus we must design a low-overhead communication protocol between the agents.

In this section, we first describe the formal setup of a distributed CPS. Then we describe the challenges that arise when trying to leverage system dynamics' knowledge in RV.

4.1 Distributed CPS Architecture

We assume a loosely coupled system. Specifically, the system consists of N reliable *agents* that do not fail, denoted by $\{A_1, A_2, \ldots, A_N\}$, without any shared memory or global clock. The output signal of agent A_n is denoted by x_n, for $1 \leq n \leq N$. We will need to refer to some global clock which acts as a '*real*' timekeeper. However, this global clock is a theoretical object used in definitions and theorems, and is *not* available to the agents. Messages that are passed between agents as part of their normal functioning (i.e., independently of the monitoring task) can be modeled as instantaneous changes in signal value, and will not be modeled separately here. We make two assumptions:

- *(A1) Partial synchrony.* The *local clock* (or time) of an agent A_n can be represented as an increasing function $c_n : \mathbb{R}_+ \to \mathbb{R}_+$, where $c_n(\chi)$ is the value of the local clock at global time χ. Then, for any two agents A_n and A_m, where $m, n \in [N]$, we have:

$$\forall \chi \in \mathbb{R}_+ . |c_n(\chi) - c_m(\chi)| < \varepsilon$$

  with $\varepsilon > 0$ being the maximum *clock skew*. The value ε is assumed fixed and known by the monitor in the rest of this paper. In the sequel, we make it explicit when we refer to 'local' or 'global' time.
- *(A2) Deadlock-freedom.* The agents being analyzed do not deadlock.

Assumption (A1) is met by using a clock synchronization algorithm, like NTP [10], to ensure bounded clock skew among all agents. Assumption (A2) is stronger, meaning that some systems do indeed deadlock, while others are explicitly guaranteed not to deadlock.

In the discrete-time setting, a distributed signal is defined as a set of events (where an event is a value change in an agent's variables, due to a software update for example). We now update this definition for the continuous-time setting of this paper. Namely, we define a distributed signal to be a set of signals (one per agent), where each timeline is measured using the *local* clock c_n defined above, an event on the n^{th} agent is a pair $(t, x_n(t))$, and inter-agent events are partially ordered by a variation of the *happened-before* ($\rightsquigarrow$) relation [7], extended by our assumption (A1) on bounded clock skew among all agents. . The formal definition follows.

Definition 2. *A* distributed signal *on N agents is a pair $(E, \rightsquigarrow)$, where $E = (x_1, x_2, \ldots, x_N)$ is a vector of signals, one for each agent. All signals x_k share a support, modeling that all agents execute in parallel and that time (measured locally) does not stop. The relation $\rightsquigarrow$ is a relation between events in signals such that:*

(1) In every signal x_n, all events are totally ordered, that is,

$$\forall n \in [N].\ \forall t, t' \in I_n.\Big(t < t'\Big) \Rightarrow \Big((t, x_n(t)) \rightsquigarrow (t', x_n(t'))\Big),$$

where the set I_n is a bounded nonempty interval.

(2) If the time between any two events is more than the maximum clock skew ε, then the events are totally ordered, that is,

$$\forall m, n \in [N].\ \forall t, t' \in I_n.\Big(t + \varepsilon < t'\Big) \Rightarrow \Big((t, x_m(t)) \rightsquigarrow (t', x_n(t'))\Big).\ \blacksquare$$

We use a new symbol E for the trace in a distributed signal, rather than σ, to emphasize that signals in E are partially synchronous, and that their supports are bounded (which is a technical requirement - see [11]).

Example Figure 3 shows two timelines, generated by two agents executing concurrently. Every moment in each timeline corresponds to an event $(t, x_n(t))$, $n \in [2]$. Thus, we see that the following hold: $(1, x_1(1)) \rightsquigarrow (2.3, x_1(2.3))$, $(2.3, x_1(2.3)) \rightsquigarrow (2.94, x_2(2.94))$, $(1, x_2(1)) \rightsquigarrow (2.94, x_2(2.94))$, and $(2.94, x_2(2.94)) \not\rightsquigarrow (3, x_1(3))$.

The classic case of complete asynchrony is recovered by setting $\varepsilon = \infty$. Because the agents are only synchronized within an ε, it is not possible to actually evaluate all signals at the same moment in global time. The notion of *consistent cut* and its frontier, defined next, capture *possible* global states: that is, states that could be valid global states (see Fig. 3).

Definition 3. *Let $(E, \rightsquigarrow)$ be a distributed signal over N agents and S be the set of all events defined as follows:*

$$S = \Big\{(t, x_n(t)) \mid x_n \in E \wedge t \in I_n \wedge I_n \subseteq \mathbb{R}_+\Big\}.$$

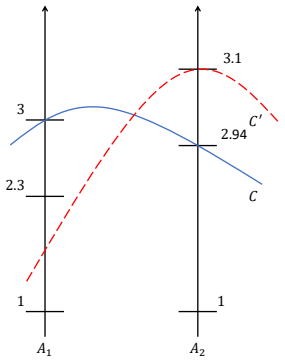

Fig. 3: Two partially synchronous continuous concurrent timelines with $\varepsilon = 0.1$. C is a consistent cut. C' is not a consistent cut: indeed $(1, x_1(1)) \rightsquigarrow (3.1, x_2(3.1))$ by definition of happened-before. Moreover, $(3.1, x_2(3.1)) \in C'$ but $(1, x_1(1))$ is not in C', which violates the definition of consistent cut.

A consistent cut C *is a subset of* S *if and only if when* C *contains an event* e*, then it contains all events that happened before* e*. Formally,*

$$\forall e, f \in S \;.\; (e \in C) \wedge (f \rightsquigarrow e) \Rightarrow (f \in C). \qquad \blacksquare$$

From this definition and Definition 2 it follows that if $(t', x_n(t'))$ is in C, then C also contains every event $(t, x_m(t))$ s.t. $t + \varepsilon < t'$. Observe that due to time asynchrony, at any global time $\chi \in \mathbb{R}_+$, there exists infinite number of consistent cuts denoted by $\mathcal{C}(\chi)$. This is attributed to the fact that between any two local time instances t_1 and t_2 on some signal x, there exists infinite number of time instances. Therefore, infinite number of consistent cuts can be constructed.

A consistent cut C can be represented by its *frontier*

$$\mathsf{front}(C) = \Big\{(t_1, x_1(t_1)), \ldots, (t_N, x_n(t_N))\Big\},$$

in which each $(t_n, x_n(t_n))$, where $1 \leq n \leq N$, is the last event of agent A_n appearing in C. Formally:

$$\forall n \in [N] \;.\; (t_n, x_n(t_n)) \in C \text{ and}$$
$$t_n = \max\Big\{t \in I_n \mid (t, x_n(t)) \in C\Big\}.$$

Example Assuming $\varepsilon = 0.1$ in Fig. 3, it comes that all events below (thus, before) the solid arc form a consistent cut C with frontier $\mathsf{front}(C) = \{(3, x_1(3)), (2.94, x_2(2.94))\}$. On the other hand, all events below the dashed arc do *not* form a consistent cut since $(2.3, x_1(2.3)) \rightsquigarrow (3.1, x_2(3.1))$ and $(3.1, x_2(3.1))$ is in the set C', but $(2.3, x_1(2.3))$ is not in C'.

4.2 Additional Challenges in the Distributed Setup

We can conceive of numerous additional challenges arising in the distributed setup.

The first class of challenges of distributed monitoring is that the knowledge of the dynamics is distributed over the agents, with each agent knowing only its own dynamics a priori. Already, even if we assume total synchrony between the agents, there is a need to design a protocol by which this information is shared, or some summary of it is exchanged. We will give an example of the complexity of the task: assume that each agent has bounds on its own dynamics. In the centralized setting, this can be used to skip parts of the signal, as described earlier. In the distributed setup, this cannot be done directly: indeed whether a formula will change truth value in the next second, say, depends on the bounds of two or more agents. The relevant segment of x_1 (signal of agent 1 that should be monitored) depends on the relevant segment of x_2 and vice versa. E.g., if the atom is $(x_1 + x_2 > 0)$ then, without knowing the bounds of x_2, the entire x_1 can be relevant for monitoring.

On the other hand, if agent 1 know that $-1 \leq x_2 \leq 1$ then it can conclude the only relevant portion of x_1 is $1 \leq x_1$. Agent 1 can then send only those segments where its signal is larger than 1. Thus there are at least two kinds of tokens (packets of information) that must be exchanged: tokens with bounds (from Agent 2 to Agent 1) and tokens with pieces of the signal (from Agent 1 to Agent 3, etc).

Secondly, the partial synchrony creates difficulties in trying to leverage the model knowledge. Indeed, even on a single agent whose clock is drifting relative to global physical time ($c_n(\chi) \neq \chi$), there is an error between the model predictions, which embody an error-free timeline, and the measurements, which integrate a drifting timeline. For example, assume the monitor has a short model, which it uses to predict the trajectory $x_1[0,1] \to \mathbb{R}$. Based on this it concludes it can ignore the segment $x_1[0:0.3]$, i.e., the first 0.3 seconds of the signal, *measured on a perfect clock*. But its local clock has an unknown, ε-bounded drift from physical time. So, measured on its local clock, the conservative thing is to skip $x_1[0:0.3-\varepsilon]$.

It might be that the general case is a tedious, but ultimately straightforward, generalization of this calculation: conservatively ignore shorter pieces. The technical problem is then to build an inductive argument for the general case, taking into account the nature of the dynamics knowledge that is available.

The case of a simulating model is particularly interesting: recall that Σ_g simulates Σ_f if for every $(x, y) \in R$ it holds that: $x \xrightarrow{\overline{u}} x'$ in the f-system implies the existence of a transition $y \xrightarrow{\overline{u}'} y'$ in the g-system s.t. $(x', y') \in R$. The key characteristic is that $\overline{u}$ and $\overline{u}'$ are not necessarily of the same duration - indeed, in general, it is difficult to get some kind of relation between the two durations. So far in the literature, the properties verified on the simulating g-system have not depended on time. It is not at all clear how we might apply a conservative ε-shortening to the time bounds when the relation between the g and f timelines is unknown.

5 Using Verified Dynamics

In this section, we explore a different kind of information: the monitor does not have access to any kind of model. Rather, it has access to information about offline model-checking runs on the model. That is, offline, the designer ran a series of model-checking runs on the model, and the verdicts of these runs are available to the monitor. Every piece of information is thus structured as a tuple (x, h, V, ϕ), where:

1. x is the state from which the verification ran
2. h is the verification horizon, if applicable. E.g., in bounded model checking, this would be the bound.
3. ϕ is the formula against which the model was checked. That is, the designer checked whether the model satisfies $M, x \models \phi$. (Here, M is the model).
4. $V \in \{\top, \bot\}$ is the verdict, or result, of the verification.

Such information is naturally available in a model-based design cycle where formal methods play a role. Note we allow the possibility that the model-checking runs were limited, both in terms of their bounds, and in terms of the states they are run from.

The runtime monitor is monitoring for some property(ies) ψ that, in general, is different from ϕ. The question then: can the monitor leverage the knowledge that $M, x \models \phi$ to monitor ψ more efficiently? Intuitively, that should be possible some of the time: when the current state is 'near' x, and the formula ψ is somehow related to ϕ.

The following trivial cases illustrates that intuition: if the current state is exactly x, and $\psi = \phi$, the obviously there is no need to monitor. Or, assume the model is a finite automaton. The current state is within $s \leq h$ hops of x, and $\psi \Rightarrow \Diamond_s \phi$. Here too, it is not necessary to monitor since the satisfaction of ϕ now is implied by that of ψ earlier.

A more involved example is provided by offline reachability: suppose the model has been verified to not enter a set of states S within h time units if it starts from some set J. (Reachability tools like SpaceEx and Flow* could be used for this). Such a guarantee holds under some assumptions on the system's environment, say the assumption that any disturbance w is bounded: $|w(t)| \leq 1$ for all time t. Now the atomic proposition p that '$x \in S$' might appear in formulas that are monitored online. The offline reachability result can be leveraged in the monitor in a predictive manner: whenever the state x is about to enter (or enters) set J, every appearance of p in the monitored formula can be replaced by `false`, as long as it appears within the scope of a temporal operators whose combined intervals add up to less than h time units.

The challenge is to formalize then generalize this intuition to more cases. We will also need to account for the fact that the monitor does not actually know the current state: in general, it only has access to observables, i.e., to $H(x(t))$ rather than $x(t)$.

6 Conclusion

In this paper, we explored the idea of combining runtime verification (RV) with a pre-computed knowledge of system dynamics to enhance the effectiveness of RV and improve its runtime overhead. To this end, we introduced a set of ideas on how different aspects of system dynamics can be potentially leveraged to equip the monitor with tools to decrease it involvement by ruling out certain scenarios.

Our next natural step is to materialize these ideas by developing the theory as well as conducting rigorous experiments to validate our results. *Stream-based* runtime verification techniques such as RTLola[3] allows real-time monitoring for cyber-physical systems and networks but they currently cannot handle data streams coming from components that do not share a common clock and do not take system dynamics into account. Thus, longer-term interesting problem is to integrate our ideas in this on using system dynamics as well as our results in [11] with RTLola.

References

1. Houssam Abbas, Hand Mittelmann, and Georgios Fainekos. Formal property verification in a conformance testing framework. In *ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, October 2014.
2. Yashwanth Singh Rahul Annapureddy, Che Liu, Georgios E. Fainekos, and Sriram Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *Tools and algorithms for the construction and analysis of systems*, volume 6605 of *LNCS*, pages 254–257. Springer, 2011.
3. A. Dokhanchi, B. Hoxha, and G. Fainekos. Online monitoring for temporal logic robustness. In *Proc. of Runtime Verification*, 2014.
4. A. Donzé, T. Ferrère, and O. Maler. Efficient robust monitoring for STL. In *Proceedings of the 25th International Conference on Computer Aided Verification (CAV)*, pages 264–279, 2013.
5. A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *Proceedings of the 8th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, pages 92–106, 2010.
6. V. Deshmukh J, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, and S. A. Seshia. Robust online monitoring of signal temporal logic. *Formal Methods System Design*, 51(1):5–30, 2017.
7. L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.
8. O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Proceedings of the Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems (FORMATS/FTRTFT)*, pages 152–166, 2004.
9. R. Medhat, B. Bonakdarpour, D. Kumar, and S. Fischmeister. Runtime monitoring of cyber-physical systems under timing and memory constraints. *ACM Transactions of Embedded Computing Systems*, 14(4):79:1–79:29, 2015.

[3] https://www.react.uni-saarland.de/tools/rtlola/

10. D. Mills. Network time protocol version 4: Protocol and algorithms specification. RFC 5905, RFC Editor, June 2010.
11. A. Momtaz, N. Basnet, H. Abbas, and B. Bonakdarpour. Predicate monitoring in distributed cyber-physical systems. In *Proceedings of the 21st International Conference on Runtime Verification (RV) 2021*, pages 3–22, 2021.
12. Jan-David Quesel. *Similarity, Logic, and Games: Bridging Modeling Layers of Hybrid Systems*. PhD thesis, Carl Von Ossietzky Universitat Oldenburg, July 2013.