Monitoring Cyber-Physical Systems

Ezio Bartocci[1], Jyotirmoy Deshmukh[2], Alexandre Donzé[3], Georgios Fainekos[4], Oded Maler[5], Dejan Ničković[6], and Sriram Sankaranarayanan[7]

[1] Vienna University of Technology
[2] Toyota Technical Center
[3] University of California at Berkeley
[4] Arizona State University
[5] VERIMAG, CNRS and University of Grenoble-Alpes (UGA)
[6] AIT Austrian Institute of Technology GmbH
[7] Univesity of Colorado, Boulder

Abstract. The term Cyber-Physical Systems (CPS) typically refers to to engineering, physical and biological systems monitored and/or controlled by an embedded computational core. The behavior of a CPS over time is generally characterized by the evolution of physical quantities and discrete software and hardware states. In general, these can be mathematically modeled by the evolution of continuous state variables for the physical components interleaved with discrete events. Despite large effort and progress in the exhaustive verification of such hybrid systems, the complexity of CPS models limits formal verification of their safe behavior only to small instances. An alternative approach, closer to the practice of simulation and testing, is to monitor and to predict CPS behaviors at simulation-time or at runtime. In this chapter we attempt to summarize the state-of-the-art techniques for qualitative and quantitative monitoring of CPS behaviors. We present an overview of some of the important applications and, finally, we describe the tools supporting CPS monitoring and compare their main features.

1 Introduction

The world around us is in a constant flux with "things" changing dynamically. Planets move, temperatures rise and fall, rivers flow, rocks break down. In addition to these physical dynamics, a large part of the changing world is due to the activities of living systems and in particular humans, their social constructs and the artifacts they build. Houses are illuminated and air-conditioned, power is generated, distributed and consumed, cars drive on roads and highways, plants manufacture materials and objects, commercial transactions are made and recorded in information systems. Airplanes fly, continuously changing location and velocity while their controllers deal, via sensors and actuators, with various state variables in the engine and wings. Conceptually those processes can be viewed as temporal behaviors, waveforms or signals or time series or sequences, where continuous and discrete variables change their values over time and various types of events occur along the time axis.

The systems that generate these behaviors are evaluated to some extent as good or bad, efficient or worthless, excellent or catastrophic. Such an evaluation can apply to the system in question as a whole, to some of its components or to a particular period of operation. We use *monitoring* to denote the act of observing and evaluating such temporal behaviors. Behaviors can be very long, spanning over a large stretches of time, densely populated with observations. They can also be very wide, recording many variables and event types. As such they carry too much information by themselves to be easily and directly evaluated. What should be distilled out of these behaviors should somehow be expressed and specified. The mathematical objects that do this job are functions that map complex and information-rich behaviors into low dimensional vectors of bits and/or numbers that indicate satisfaction of logical requirements and the values of various performance indices, see Figure 1.

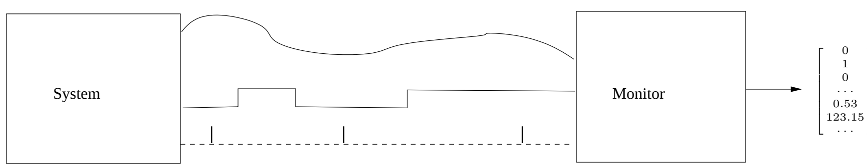

Fig. 1. Monitoring as reducing complex temporal behaviors into low-dimensional vectors of bits and numbers.

The evaluation can be based on a gross abstraction of the behavior, for example the event of an airplane crash corresponds to a zero location on the z dimension and a large downward velocity at some point in time. Likewise the death of a patient can be specified by the stabilization of his or her heart beat signal to a constant value. More often than not, those global catastrophic events may be related to (and preceded by) more detailed temporal behaviors that involve intermediate steps and variables, for example, some rise in the engine temperature which is not followed by certain actions such as lowering speed or turning on a cooling system. In less safety-critical contexts, systems are evaluated for performance, for example the time a client spends in a queue between requesting a service and being granted, or the energy consumption of a computer or a chemical plant along some segment of time.

Before going further, let us distinguish between two major contexts in which the monitoring of dynamic behaviors can take place (see a more elaborate discussion in [87]). The first is the monitoring of real systems during their execution via online measurements. Here the role of monitoring is to alert in *real time* in order to trigger corrective actions, either by a human operator or by a supervisory layer of control. A primitive form of this type of monitoring exists in many domains: indicators on the control panel of a car, airplane or electronic device, monitors for physiological conditions of patients in a hospital and SCADA (Supervisory Control and Data Acquisition) systems for controlling complex large-scale systems such as airports, railways or industrial plants. In fact, any information system can be viewed as performing some kind of a monitoring activity.

The other context is during model-based system design and development where all or some of the system components do not exist yet in flesh and blood and their models, as well as the model of the environment they are supposed to interact with, exist as virtual objects of mathematical and computational nature. The design process of such systems is typically accompanied by an extensive simulation and verification campaign where the response of the system to numerous scenarios is simulated and evaluated. This chapter is focused mostly on this design-time monitoring context, where those simulation traces constitute the input of the monitoring process. Some of the techniques and considerations are shared, nevertheless, with the monitoring of real systems.

The activity of simulating a system and checking its behavior is part of the verification and validation process whose goal is to ensure, as much as possible, that the system behaves as expected and to avoid unpleasant surprises after its deployment. In some restricted contexts of simple programs or digital circuits, this process can be made exhaustive and "formal" in the sense that all possible classes of scenarios are covered. When dealing with cyber-physical systems, whose existence and interaction scope are not confined to the world inside a computer for which practically exact models exist, complete formal verification is impossible. In this domain, simulation-based lightweight verification is the common practice, accompanied by the hope of providing a good finite coverage of the infinite space of behaviors.

Part of the runtime verification movement is coming from formal verification circles, attempting to export to the simulation-based verification domain another ingredient of formal verification, namely, the formal specification of the system requirements. In the context of discrete systems, software or digital hardware, formalisms such as *temporal logic* or *regular expressions* are commonly used. They can specify in a declarative manner which system behaviors, that is, sequences of states and events, conform with the intention of the designer in terms of system functionality, and which of these behaviors do not. Such specifications can be effectively translated into monitoring programs that read behaviors and check whether the requirements are satisfied. As such they can replace or complement tedious manual inspection of simulation traces or ad hoc programming of property testers.

Let us give some intuitive illustrations of the nature of these formalisms. Linear-time temporal logic (LTL) provides a compact language for speaking of sequences and the relations between their values at different points in time. The semantics of an LTL formula φ is time dependent with $(w, t) \models \varphi$ indicating that formula φ holds for sequence w at position t. The simplest formulas are state formulas which are satisfied at t according to the value of the sequence at t. That is, writing p for the truth value of a logical variable, or $x > 0$ for a numerical variable, is interpreted at each t as $p[t]$ and $x[t] > 0$, respectively.

More complex formulas are built using Boolean and temporal operators. The latter are divided into two types, future and past operators. The satisfaction of a future operator at position t depends on the values of the sequence at some or all the positions from t onward, that is, the suffix of w from t to $|w|$. For example $\Box\, p$ (*always* p) is true at any t such that p holds at every $t' \geq t$. The analogous past formula $\boxminus\, p$ (*historically* p) holds at t if p holds at any position $t' \leq t$, in other words, along the prefix of w from 0 to t. The satisfaction of a future formula by the whole sequence w is defined as its satisfaction at position 0 while that of a past formula, by its satisfaction at $|w|$. Past formulas have some advantages such as causality, while future LTL is more commonly used and is considered by some to be more intuitive.

The formula $\Box\, p$ quantifies universally over all time instances. The dual formula $\Diamond\, p$ (*eventually* p) quantifies existentially. It holds at t if p holds at *some* t' in the future. The weakness of such a property from a practical standpoint is that there is no bound on the distance between t and t', a fact that may upset some impatient clients waiting for a response during their lifetime. We should note that in verification, formulas are often interpreted over *infinite* sequences generated by automata, while in monitoring we deal anyway with finite sequences and if p never becomes true until the end of w, formula $\Diamond\, p$ is falsified.

A more quantitative alternative to $\Diamond$ can be expressed in discrete time using the *next* operator. Formula $\bigcirc\, p$ (*next* p) holds at t if p holds at $t+1$. Thus, the requirement that each p is followed by q within 2 to 3 time steps is captured by the formula

$$\Box(p \rightarrow (\bigcirc(\bigcirc\, q)) \vee (\bigcirc(\bigcirc(\bigcirc\, q)))).$$

This formulation may become cumbersome for large delay constants and by extending the syntax we can write this formula as

$$\Box(p \rightarrow \Diamond_{[2,3]}\, q)$$

with $\Diamond_{[a,b]}\, p$ being satisfied at t if p is satisfied at some $t' \in [t+a, t+b]$. In discrete time, this can be viewed as a syntactic sugar, but in dense time where *next* is anyway meaningless, this construct allows events to occur anywhere in an interval, not necessarily at sampling points or clock ticks.

Sequential composition is realized in future LTL using the *until* operator. The formula $p \mathcal{U} q$ (p *until* q) is satisfied at t if q occurs at some later point in time while p holds continuously until then. Using this operator, for which $\Diamond$ and $\Box$ are degenerate cases, one can require that some process should not start as long as another process has not terminated. The semantics of *until* is defined below.[1]

$$(w,t) \models \varphi_1 \mathcal{U} \varphi_2 \quad \text{iff} \quad \exists t' \geq t \; ((w,t') \models \varphi_2 \wedge \forall t'' \in [t,t') \; (w,t'') \models \varphi_1) \tag{1}$$

The past counter-part of *until* is the *since* operator with $q \mathcal{S} p$ (q *since* p) meaning that p occured in the past and q has been holding continuously since then. The semantics of *since* is given below.

$$(w,t) \models \varphi_2 \mathcal{S} \varphi_1 \quad \text{iff} \quad \exists t' \leq t \; ((w,t') \models \varphi_1 \wedge \forall t'' \in (t',t] \; (w,t'') \models \varphi_2) \tag{2}$$

[1] Variants of *until* may differ on whether φ_2 is required to occur or whether φ_1 can cease to hold at the moment φ_2 starts or only after that.

It is interesting to compare these operators with the *concatenation* operation used in regular expressions.

Regular expressions constitute a fundamental and popular formalism in computer science, conceived initially to express the dynamic behavior of neural networks, and later applied to lexical and grammatical analysis. Traditionally, such expressions are defined over a monolithic alphabet of symbols but in order to present them in the same style as LTL, we will use product alphabets such as $\{0,1\}^n$, defined and accessed via variables. Thus an expression p in our approach would be interpreted in the traditional approach as the set of all Boolean vectors in a global alphabet in which the entry corresponding to p is 1.

In discrete time, p is satisfied by any sequence of length one in which p holds. Sequential composition is realized by the concatenation operation where $\varphi_1 \cdot \varphi_2$ is satisfied by any sequence w that admits a factorization $w = w_1 \cdot w_2$ such that w_1 satisfies φ_1 and w_2 satisfies φ_2. This is best illustrated by defining the semantics using the satisfaction relation $(w,t,t') \models \varphi$ which holds whenever the subsequence of w starting at t and ending in t' satisfies expression φ:

$$(w,t,t') \models \varphi_1 \cdot \varphi_2 \quad \text{iff} \quad \exists t'' \in [t,t'] \; (w,t,t'') \models \varphi_1 \wedge (w,t'',t') \models \varphi_2 \qquad (3)$$

The Kleene star allows to repeat concatenation for an indefinite but finite number times, with φ^* being satisfied by any sequence that admits a finite factorization in which all factors satisfy φ. As an example, expression $(\neg p)^+ \cdot q \cdot p$ specifies sequences in which a finite period where p does not hold is followed by the occurrence of q followed by p.

Note that unlike LTL, regular expressions are more symmetric with respect to the arrow of time, as can be seen by the difference between their respective semantics definitions. The definitions of $\models$ in (1) and (2) go recursively from t to the future or the past, respectively. When they come up from the recursion they do it in the opposite direction: for future LTL, satisfaction is computed backwards and that of past LTL is computed forward. For concatenation, in contrast, the semantics of $\models$ in (3) is defined by a double recursion which takes the whole sequence and splits it into two parts which are the arguments for the two recursive calls. The semantics is collected from both ends while coming up from the recursion.

The exportation of these formalisms and their monitoring algorithms to the cyber-physical world has to cope with the hybrid nature of such systems. The dynamics of digital systems is captured by discrete event systems such as automata, generating discrete sequences of logical states and events. Physical systems are modeled using formalisms such as differential equations, producing behaviors viewed as continuous signals and trajectories. Specification formalisms and monitoring algorithms should then be extended so as to express and check temporal properties of such behaviors, exhibiting dense time and numerical values. This topic is the focus of the present chapter.

In order to be relevant to real applications, we should keep in mind that continuous dynamical systems are the object of study of various branches of mathematics and engineering, in particular, control and signal processing. These domains have developed over the years a variety of ways to measure and evaluate such systems and their behaviors, which are appropriate to their physical and mathematical nature. There is a variety of mathematical norms that reduce such behaviors into single numbers. There are transformations like Fourier's that extract the spectral properties of signals for the purpose of classification or noise removal. There are many statistical ways to assess signals and time series and detect occurring patterns. The challenge in monitoring cyber-physical systems is to integrate these traditional performance measures with those provided by the newly developed verification-inspired formalisms which are more suitable for capturing sequential aspects of behaviors.

2 Specification Languages

In this section, we present Signal Temporal Logic (STL) as the specification language that we use in this document for expressing properties of CPS. We introduce the syntax of the formalism, together with its qualitative and quantitative semantics.

2.1 Signal Temporal Logic

In this section, we introduce Signal Temporal Logic (STL) as the specification language for expressing requirements of cyber-physical systems. It extends the continuous-time Metric Temporal Logic (MTL) [79] with numerical predicates over real-valued variables. In particular, STL enables reasoning about real-time properties at the interface between components that exhibit both discrete and continuous dynamics.

We denote by X and P finite sets of *real* and *propositional* variables. We let $w : \mathbb{T} \to \mathbb{R}^m \times \mathbb{B}^n$ be a multi-dimensional *signal*, where $\mathbb{T} = [0, d) \subseteq \mathbb{R}$, $m = |X|$ and $n = |P|$. Given a variable $v \in X \cup P$ we denote by $\pi_v(w)$ the *projection* of w on its component v.

We now define the variant of STL that contains both *past* and *future* temporal operators. The syntax of an STL formula φ over $X \cup P$ is defined by the grammar

$$\varphi := p \mid x \sim c \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2 \mid \varphi_1 \mathcal{S}_I \varphi_2$$

where $p \in P$, $x \in X$, $\sim \in \{<, \leq\}$, $c \in \mathbb{Q}$, and $I \subseteq \mathbb{R}^+$ is an interval. We define the semantics of STL as the *satisfiability relation* $(w, t) \models \varphi$, indicating that the signal w satisfies φ at the time point t, according to the following definition. Given that we interpret the logic only over the finite traces, we let the satisfaction relation to be defined only for $t \in \mathbb{T}$.

$$\begin{array}{ll}
(w,t) \models p & \leftrightarrow \pi_p(w)[t] = \mathsf{true} \\
(w,t) \models x \sim c & \leftrightarrow \pi_x(w)[t] \sim c \\
(w,t) \models \neg\varphi & \leftrightarrow (w,t) \not\models \varphi \\
(w,t) \models \varphi_1 \vee \varphi_2 & \leftrightarrow (w,t) \models \varphi_1 \text{ or } (w,t) \models \varphi_2 \\
(w,t) \models \varphi_1 \mathcal{U}_I \varphi_2 & \leftrightarrow \exists t' \in (t+I) \cap \mathbb{T} : (w,t') \models \varphi_2 \text{ and } \forall t'' \in (t,t')(w,t''), \models \varphi_1 \\
(w,t) \models \varphi_1 \mathcal{S}_I \varphi_2 & \leftrightarrow \exists t' \in (t-I) \cap \mathbb{T} : (w,t') \models \varphi_2 \text{ and } \forall t'' \in (t',t), (w,t'') \models \varphi_1
\end{array}$$

We say that a signal w *satisfies* a STL formula φ, denoted by $w \models \varphi$, iff $(w, 0) \models \varphi$. In the remainder of this section, we discuss several specific aspects of the STL syntax and semantics.

Finitary interpretation Specification formalisms with future temporal operators are typically defined over *infinite* behaviors. In particular, we can have a specification that is satisfied at time t iff a future obligation is fulfilled at some future time instant $t' > t$. Consequently, observing a finite prefix of a behavior may not be sufficient to determine the satisfaction or the violation of a temporal specification according to its standard semantics. The finitary interpretation of future temporal logics is a well-studied problem in the monitoring research field. In order to tackle this issue, we adapt the semantics of $\mathcal{U}$ and restrict the existential quantification of time to the (possibly bounded) signal domain. This altered semantics provides a natural interpretation of STL over finite signals. In particular, the *eventually* operator has a so-called *strong* interpretation – $\Diamond \varphi$ is satisfied iff φ holds at any time before the signals ends. Similarly, the *always* operator has a *weak* interpretation under this semantics – $\Box \varphi$ is satisfied iff φ is not violated during the signal duration.

The problem of interpreting temporal logic over finite or truncated behaviors was extensively studied in [49], where *weak*, *strong* and *neutral* views of the finitary semantics for LTL are proposed. In [48], the authors provide a topological characterization of weak and strong temporal operators. Finally, we also mention the real-time monitoring framework from [24], where 3-valued $\{\mathsf{true}, \mathsf{false}, \mathsf{inconclusive}\}$ semantics are used to provide a finitary interpretation of a real-time temporal logic.

Strict interpretation of temporal operators We adopt the semantics of $\mathcal{U}_I$ and $\mathcal{S}_I$ that are *strict* in both arguments, as originally proposed in [7]. This is in contrast to the non-strict semantics of $\mathcal{U}$ and $\mathcal{S}$ in the discrete-time LTL [100]. It turns out that the strict interpretation of the temporal operators is needed for continuous-time temporal logics, a fact that we shortly discuss in this paragraph. Let us first denote by $\bar{\mathcal{U}}$ the non-strict until operator, and by $\mathcal{U}$ its strict counterpart [2]. We also recall that LTL contains the *next* operator $\bigcirc$ in addition to $\bar{\mathcal{U}}$. We can show that in discrete time, a temporal logic with $\bar{\mathcal{U}}$ and $\bigcirc$ such as LTL is equivalent to the logic that has $\mathcal{U}$ only, by using the following rules.

$$\begin{aligned} \varphi_1 \mathcal{U} \varphi_2 &\equiv \bigcirc(\varphi_1 \bar{\mathcal{U}} \varphi_2) \\ \bigcirc \varphi &\equiv \mathsf{false}\ \mathcal{U} \varphi \\ \varphi_1 \bar{\mathcal{U}} \varphi_2 &\equiv \varphi_2 \vee (\varphi_1 \wedge (\varphi_1 \mathcal{U} \varphi_2)) \end{aligned}$$

In contrast to LTL, continuous-time temporal logics such as STL do not have the next operator. It turns out that the strict interpretation of the temporal operators strictly increases the expressiveness of the underlying logic in dense time, as it enables "forcing" the time to advance. The main practical consequence

[2] We restrict our argument to the future operators for the sake of simplicity – the same reasoning can be applied to the past operators.

of strict interpretation of $\mathcal{U}$ and $\mathcal{S}$ is that it allows specification of instantaneous events in continuous time.

Derived operators The syntactic definition of STL is minimal and includes only basic operators. We can derive other standard operators as follows:

$$
\begin{array}{lll}
\text{True constant:} & \mathsf{true} & \equiv p \vee \neg p \\
\text{False constant:} & \mathsf{false} & \equiv \neg \mathsf{true} \\
\text{Conjunction:} & \varphi_1 \wedge \varphi_2 & \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2) \\
\text{Implication:} & \varphi_1 \rightarrow \varphi_2 & \equiv \neg\varphi_1 \vee \varphi_2 \\
\text{Eventually:} & \Diamond_I \varphi & \equiv \mathsf{true}\ \mathcal{U}_I\ \varphi \\
\text{Once:} & \diamondminus_I \varphi & \equiv \mathsf{true}\ \mathcal{S}_I\ \varphi \\
\text{Always:} & \Box_I \varphi & \equiv \neg \Diamond_I \neg\varphi \\
\text{Historically:} & \boxminus_I \varphi & \equiv \neg \diamondminus_I \neg\varphi
\end{array}
$$

In addition to these derived operators, we can also define instantaneous *events* that have zero duration. Such events enable specification of *rising* and *falling edges* in boolean signals.

$$
\begin{array}{l}
\text{Rising edge: } \uparrow \varphi \equiv (\varphi \wedge (\neg\varphi\ \mathcal{S}\ \mathsf{true})) \vee (\neg\varphi \wedge (\varphi\ \mathcal{U}\ \mathsf{true})) \\
\text{Falling edge: } \downarrow \varphi \equiv (\neg\varphi \wedge (\varphi\ \mathcal{S}\ \mathsf{true})) \vee (\varphi \wedge (\neg\varphi\ \mathcal{U}\ \mathsf{true}))
\end{array}
$$

2.2 Signal Temporal Logic with Quantitative Semantics

In Section 2.1, we introduced STL with *qualitative* semantics. This classical definition of STL enables to determine the *correctness* of a signal with respect to a specification. Specifically, it gives a binary pass/fail answer to the monitoring problem. When reasoning about hybrid systems that involve both discrete and continuous dynamics, the qualitative verdict may not be informative enough. After all, systems with continuous dynamics are usually expected to admit some degree of tolerance with respect to initial conditions, system parameters and environmental perturbations. Consequently, a quantitative degree of satisfaction/violation would be preferable to a simple yes/no output given by the qualitative interpretation of STL.

Fages and Rizk [103] and Fainekos and Pappas [53] proposed to tackle this issue by equipping the temporal logic with *quantitative* semantics. This extension replaces the binary satisfaction relation with the quantitative *robustness degree* function, while preserving the original syntax of the specification language. In essence, the robustness degree function gives a real value that indicates how far is a signal from satisfying or violating a specification. We illustrate the concept of the robustness degree function with a simple example on numerical predicates. Let $x < c$ be a numerical predicate. This predicate partitions the $\mathbb{R}$ domain into the set of all real values that are strictly smaller than c and those that are greater or equal to c. Picking a concrete value for x, the robustness degree gives the relative position of x to c, instead of only indicating whether x is above or below the threshold. This idea is naturally extended to the logical and temporal operators that we now formalize.

Let φ be an STL formula, w a signal and t a time instant in $\mathbb{T}$. We then define the robustness degree function $\rho(\varphi, w, t)$ as follows.

$$
\begin{aligned}
\rho(p, w, t) &= \begin{cases} \infty & \text{if } \pi_p(w)[t] = \mathsf{true} \\ -\infty & \text{otherwise} \end{cases} \\
\rho(x \sim c, w, t) &= c - \pi_x(w)[t] \\
\rho(\neg\varphi, w, t) &= -\rho(\varphi, w, t) \\
\rho(\varphi_1 \vee \varphi_2, w, t) &= \max\{\rho(\varphi_1, w, t), \rho(\varphi_2, w, t)\} \\
\rho(\varphi_1 \mathcal{U}_I \varphi_2, w, t) &= \sup_{t' \in (t+I) \cap \mathbb{T}} \min\{\rho(\varphi_2, w, t'), \inf_{t'' \in (t,t')} \rho(\varphi_1, w, t'')\} \\
\rho(\varphi_1 \mathcal{S}_I \varphi_2, w, t) &= \sup_{t' \in (t-I) \cap \mathbb{T}} \min\{\rho(\varphi_2, w, t'), \inf_{t'' \in (t',t)} \rho(\varphi_1, w, t'')\}
\end{aligned}
$$

There is a couple of fundamental properties that relate the STL quantitative semantics to its qualitative counterpart. Consider an arbitrary STL formula φ, a signal w and time $t \in \mathbb{T}$. The first property says that for any $\rho(\varphi, w, t) \neq 0$, its sign determines whether $(w, t) \models \varphi$. The second property states that if $(w, t) \models \varphi$, then for any signal w' whose pointwise distance from w is smaller than $\rho(\varphi, w, t)$ we also have $(w', t) \models \varphi$

Alternative quantitative semantics In this section, we presented a quantitative semantics that allows measuring *spatial* robustness of STL specifications. This definition takes into account the spatial variations of signals when compared to STL specifications. Developing alternative notions of robustness degree for STL has been an active area of research in the recent years. In [43], the authors extend the quantitative semantics of STL by combining spatial with *time* robustness, thus also allowing to quantify temporal perturbations in signals. The idea of the combined space-time robustness for STL is further enhanced in [6] with *averaged* temporal operators. In [104], the authors identify that the bounded eventually $\diamondsuit_{[a,b]}$ operator behaves like the *convolution* operator commonly used in filtering and digital signal processing. Following this surprising observation, one can develop various quantitative semantics for temporal logic, by defining the appropriate kernel window used for evaluating the formula. These additional operators enable reasoning not only about the worst-case but also the average-case behaviors. The Skorokhod metric provides an altertnative way to measure mismatches between continuous signals in both space and time. An effective procedure for computing the Skorokhod distance between two behaviors is developed and presented in [85, 35]. This method is extended to estimate the Skorokhod distance between *reachpipes* in [86]. Nevertheless, there are no available methods yet to compute the Skorokhod distance between a signal and an STL formula.

3 Monitoring Algorithms

In this section, we present algorithms answering the following *monitoring* question: what is the qualitative and/or quantitative satisfaction of a formula φ by a signal w? This problem is much easier than *model-checking*, i.e., proving that a system satisfies a formula, which is undecidable for STL even for simple classes of systems. Yet, it is desirable that efficient algorithms exist for monitoring, as this task can typically be repeated on large numbers of instances, or on signals of long durations. We consider two different settings: offline and online. In the offline setting, we assume that w is known before computing the satisfaction of φ. In the online setting, we assume only a partial knowledge of w, and compute successive estimates of φ satisfaction as new samples of w become available.

3.1 Offline Monitoring

For simplicity we restrict the presentation to the case of STL with future operators, and piecewise constant signals, i.e., we assume that w is completely defined by a sequence of time instants $t_0 < t_1 < \ldots < t_i < \ldots$ and values $w_0, w_1, \ldots, w_i, \ldots$ such that

$$\forall t \in [t_i, t_{i+1}), w[t] = w_i[t].$$

Here, we assume that the sequence of (time, value) pairs is finite, i.e., $i \leq N$. Moreover, we assume that the signal w holds its final value indefinitely, i.e., for all $t > t_N$, $w[t] = w[t_N]$. We present briefly an algorithm computing quantitative satisfaction of φ by w, adapted in a simpler form from [42]. For a purely Boolean monitoring algorithm, see [97]. It turns out that computing the quantitative satisfaction is not more complex than computing the Boolean, as both can be achieved in linear complexity in the size of signals.

The algorithm work by induction on the structure of the formula following the generic scheme presented in Algorithm 1.

Algorithm 1 Monitor(φ, w)

```
switch (φ)
case p:
  return ComputeSatisfaction(p, w)
case x ∼ c:
  return ComputeSatisfaction(x ∼ c, w)
case ∗ φ:
  w′ := Monitor(φ1, w)
  return ComputeSatisfaction(∗, w′)
case φ ∗ ψ:
  w′ := Monitor(φ, w)
  w′′ := Monitor(ψ, w)
  return ComputeSatisfaction(∗, w′, w′′)
end switch
```

To implement Algorithm 1, we need to provide an implementation of the function ComputeSatisfaction for each instance of the switch statement. Instances which do not involve any temporal operator are straightforward and presented in Table 1. As can be expected, the only non-trivial case is with the temporal until operator.

	Boolean	Quantitative
(p, w)	$y[t] = p$	if $p ==$ true, $y[t] = +\infty$, else $y[t] = -\infty$
$(x > c, w)$	$y[t] = (\pi_x(w)[t] > c)$	$y[t] = \pi_x(w)[t] - c$
$(\neg, w')$	$y[t] = \neg w[t]$	$y[t] = -w[t]$
$(\vee, w', w'')$	$y[t] = w'[t] \vee w''[t]$	$y[t] = \max(w'[t], w''[t])$

Table 1. ComputeSatisfaction for various operators. y is the signal returned, either Boolean or Real valued.

To compute satisfaction signals for formulas involving timed until operators, we make use of the following result:

Lemma 1. *For two STL formulas* φ, ψ,

$$\varphi \mathcal{U}_{[a,b]} \psi \Leftrightarrow \Diamond_{[a,b]} \psi \wedge \varphi \mathcal{U}_{[a,+\infty)} \psi \tag{4}$$

$$\varphi \mathcal{U}_{[a,+\infty)} \psi \Leftrightarrow \Box_{[0,a]} (\varphi \mathcal{U} \psi) \tag{5}$$

This result reduces the computation of satisfaction signal to the cases of *untimed until* ($\mathcal{U}$) and *bounded globally* ($\Box_{[0,a]}$). Note that the bounded eventually case can be easily deduced from bounded globally by time shifting of signals and the equivalence $\Box_I \varphi \Leftrightarrow \neg \Diamond_I \neg\varphi$.

Untimed Until To compute the satisfaction of untimed until, we are given two signals: w' and w'' and need to compute y. Since w' and w'' are of finite length N, y is also of finite length N. The computation goes backward starting from (y_N, t_N). The time sequence $t_0, t_1, \ldots, t_{N-1}$ is obtained by merging and sorting the sequences t'_i and t''_i so that to simplify the notations, we assume that the sequences w'_i and w''_i are defined on the same time sequence, i.e., $w'_i = w'[t_i]$ and $w''_i = w''[t_i]$ for all i. Using standard $\min - \max$ manipulations, we can show that the following recurrence is true:

$$\begin{cases} y_{N-1} = \min(w'_{N-1}, w''_{N-1}) \\ y_k \quad\;\;\; = \max(\min(w'_k, w''_k), \min(w'_k, y_{k+1})), k \in \{0, \ldots, N-2\} \end{cases}$$

Implementing this recurrence yields an algorithm with complexity in $O(2N)$.

Bounded Eventually Recall that

$$\rho(\Diamond_{[0,a]}(\varphi), w, t) = \sup_{[0,a]} \rho(\varphi, w, t)$$

so that

$$y[t] = \max_{t_i \in [t, t+a]} \{w[t_i]\}$$

In other words, the signal $t \to y[t]$ stores the sequence of maximums of signal w over a sliding finite window of size a. In [42], the authors observed that such a sliding window could be computed using an algorithm due to Daniel Lemire with linear complexity in the size of signal w. We refer the reader to [82] for details about this algorithm.

3.2 Online Monitoring

Offline algorithms assume that the entire trace is available to the monitoring procedure, and then run on the trace to produce either a Boolean satisfaction value or a quantitative (robust) satisfaction value. There are a number of situations where offline monitoring is unsuitable. Consider the case where the monitor is to be deployed in an actual system to detect erroneous behavior. As embedded software is typically resource constrained, offline monitoring – which requires storing the entire observed trace – is impractical. Also, when a monitor is used in a simulation-based validation tool, a single simulation may run for several minutes or even hours. If we wish to monitor a safety property over the simulation, a better use of resources is to abort the simulation whenever a violation (or satisfaction) is conclusive from the observed trace prefix. Such situations demand an *online monitoring algorithm*, which has markedly different requirements. In particular, a good online monitoring algorithm must: (1) be able to generate intermediate estimates of property satisfaction based on *partial signals*, (2) use minimal amount of data storage, and (3) be able to run fast enough in a real-time setting.

A basic online algorithm returns a *true* or *false* satisfaction value when the satisfaction or violation of the property being monitored can be concluded by observing the finite trace prefix. In many cases, the information in the trace prefix is insufficient to produce a conclusive answer. Thus, several kinds of semantics have been proposed to interpret MTL or STL formulae over truncated traces. These semantics typically extend the satisfaction in a Boolean sense as used by offline monitoring algorithms to a richer satisfaction domain, typically having three or four values.

In what follows, we first discuss qualitative monitoring algorithms; these algorithms, given an MTL or STL property, decide from a given prefix of a signal, if the entire signal would satisfy or violate the given property. An orthogonal, but relevant issue for online monitoring is the semantics of the MTL/STL property to be monitored on partial signal traces. We recall that following the tradition of the semantics for LTL on truncated traces [49], there are different notions of satisfaction that can be used to reason over prefixes of signals. We say that a signal-prefix *strongly* satisfies a given property if for *any* suffix the resulting signal would satisfy the property. In other words, the signal-prefix is sufficient to decide the satisfaction of the given property. We say that a signal-prefix *weakly* satisfies a given property if there is *some* suffix such that the resulting signal would satisfy the property. A third notion of satisfaction is *neutral satisfaction*, which is if the given signal-prefix satisfies the property where the temporal operators are restricted to quantify only over the length of the signal-prefix. Some algorithms for qualitative monitoring make use of such richer notions of satisfaction.

Qualitative Online Monitoring There are two main flavors of qualitative online monitoring. The first, based on work in [88], uses a modification of an algorithm similar to Algorithm 1. This procedure called *incremental marking*, essentially treats the signal as being available in chunks. The algorithm computes the robustness signal in a bottom-up fashion, starting from the leaves (i.e. atomic formulas) appearing in the STL formula and then for each super-formula, combining the robustness signals for its subformulae. The algorithm maintains the robustness signal partitioned as the concatenation of two signals: the first segment containing values that have already been propagated to the super-formula (by virtue of having sufficient information to allow deciding the satisfaction of the super-formula), and the second segment containing values that have not been propagated, as they may influence the satisfaction of the super-formula because of a part of the signal not being available.

The second flavor of qualitative online monitoring makes use of automata-based monitors [63] and the richer notions of strong and weak satisfaction of MTL properties by signal-prefixes. In this approach,the given MTL formula (with future and past modalities) is first rewritten so that all temporal operators bound by finite time intervals appear in the scope of zero or more temporal operators that are unbounded. Essentially, each bounded temporal formula defines a finite time-window into the signal, where signal values would have to be available to evaluate the subformula. The algorithm maintains a time-window for each such bounded temporal subformula as a tableau and updates it using dynamic programming methods. This effectively allows treating bounded temporal subformulae as atomic propositions. The outer unbounded operators are monitored using two-way alternating Büchi automata that accept *informative prefixes* of the signal. An informative prefix is a signal-prefix that allows deciding the satisfaction or violation of the given unbounded temporal formula. The procedure has space-bounds that are linear in the variability of the given signal, and length of the formula, and requires constructing automata that are doubly exponential in the size of the given MTL formula.

Quantitative Online Monitoring Qualitative monitoring algorithms by their nature are unable to quantify the degree to which the signal corresponding to the given signal-prefix may satisfy or violate the property of interest. Online algorithms for computing robust satisfaction semantics seek to address this gap.

The first algorithm we discuss is for online monitoring of MTL formulas with bounded future and unbounded past formulas [37]. For a given MTL formula φ, the algorithm computes the horizon, or the number of look-ahead steps that would be required to evaluate the bounded future component of a formula, and the history or the number of samples in the past that would be needed to evaluate a given bounded past formula. The algorithm then maintains a tableau that is updated every time a new signal value becomes available using a dynamic programming based step. For unbounded past operators, the algorithm exploits the fact that an unbounded past formula can be rewritten such that the computation over an unbounded history can be stored as a *summary* in a variable that is updated each time a new signal value becomes available. For the unobserved parts of the signal that would be required to compute the satisfaction of a bounded future temporal subformula, the algorithm uses a predictor and thus computes a robustness estimate.

The second algorithm computes *robust interval semantics* for STL formulas with bounded and unbounded future formulas [34]. A robust satisfaction interval (ℓ, u) for a given signal-prefix is defined such that ℓ (respectively, u) is the infimum (respectively, supremum) over the robust satisfaction values of the given property for all signals that have the given partial signal as the prefix. For example, consider a constant signal-prefix with magnitude 1, defined over time $t \in [0, 5)$. The robust satisfaction interval for the formula $\square_{[0,10]}(x > 0)$ over this signal-prefix is $(-\infty, 1]$, while the robust satisfaction interval for the formula $\diamondsuit_{[0,10]}(x > 0)$ is $[1, \infty)$. The interval semantics generalizes the notion of strong and weak semantics. A signal-prefix strongly satisfies (resp. violates) a property if the lower-bound of the robust satisfaction interval is positive (resp. upper-bound of the robust satisfaction interval is negative). A signal-prefix weakly satisfies a property if the upper bound of the robust satisfaction interval is positive.

The algorithm to compute the robust satisfaction interval uses ideas similar to ideas for the online monitoring algorithm for MTL and the qualitative algorithm based on *incremental marking*. The algorithm decorates each formula-node in the the syntax tree of the given STL formula with lists of values that are required to compute the robust satisfaction value of that subformula. For bounded future temporal formulas, assuming bounded variability for the signal, each list is of finite length. As new signal values become available, the lists for each subformula are updated recursively (possibly inserting new nodes in the list). For unbounded horizon temporal formulas, such lists would be infinite in size, but for a large class of unbounded horizon temporal formulas, a finite summarization procedure can be used to keep the list sizes bounded.

4 Extensions

It is out of doubt that STL has gained in the last decade an increased popularity among engineers for its conciseness and expressive power enabling to specify complex behavioral properties related to the order and the temporal distance among discrete *events* such as the satisfaction of predicates (e.g., threshold crossing) over the real variables.

However, the pure time-domain nature of this specification language sometimes has revealed to be a technical impediment to overcome for an immediate applicability to cyber-physical systems. In particular, abnormal signal behaviours such as undesirable oscillations and complex topological requirements (i.e. the spatial distribution of the entities generating signals) are very challenging to capture using only the time-domain.

For this reason, in the last decade STL has inspired a number of extensions that have been successfully applied in many applications ranging from identifying oscillatory behaviours in analog circuits [94], biological systems [26] and music melodies [44] to specifying spatiotemporal requirements in reaction-diffusion systems [11, 13, 92] and smart grids [61]. In the following we aim to provide an overview of some of the STL extensions recently proposed.

4.1 Monitoring Complex Oscillatory Properties

The dynamical behaviour of a physical system often exhibits complex oscillatory patterns representing an infinite periodically behaviour. Real-life analog signals are characterised by omnipresent noise, i.e., random perturbations of the desired signal. Damped oscillations or oscillations with increasing amplitude are peculiar aspects in many biological systems [26, 18]. Abnormal oscillations due to the presence of *spikes* or *hunting* oscillations [94] are considered undesired behaviors in analog circuit design. Signal-processing tasks such as *peak* detection [16] is common in many medical cyber-physical systems whose correctness impacts the performance and the sensitivity of the computational devices involved. Providing a concise formal specification language expressive enough to characterise such patterns and to efficiently monitor them is a very challenging task.

The classic STL is not expressive enough to distinguish classes of oscillatory patterns such as damped oscillations or oscillations with increasing amplitude, because it is not able to globally reference and compare local properties (i.e., local minima/maxima) of a signal. Motivated by this necessity the authors in [26] have proposed an extension of STL (named STL-$*$), augmenting STL with a freezing operator that allows to record the signal values during the evaluation of a sub-property, and to reuse it for comparison in the other parts of the formula. This operator increases the expressive power of STL and for instance it enables to express and to capture various dynamic aspects of oscillations. A quantitative semantics for STL-$*$ is then proposed in [27].

Although STL-$*$ is more expressive than STL in its discriminating power of oscillatory patterns its analysis is still limited to the time-domain representation of a signal. However, oscillatory patterns (i.e., chirp signals, hunting behaviours, noise filtering) are in general very challenging and tedious to investigate using only time and a time-frequency analysis is essential sometime to efficiently detect them. Time-frequency analysis is an important branch of *signal processing* and it is based on the study of the *spectrogram*, a representation of the frequencies' magnitude in a signal as they vary with time. Spectrograms can be calculated from digitally sampled data in the time-domain representation of the signal using extensions of the classic Fourier transforms such as the Short Time Fourier Transform (STFT) or Wavelet Transforms (WTs). Although a preliminary work on combining time and frequency domain specifications for periodic signals is reported in [31], the first attempt to provide a unified formalism to express time-frequency properties of a signal is the *time-frequency logic* (TFL) introduced in [44]. TFL extends STL with predicates evaluating the magnitude of a particular frequency range in a point in time. The semantics of TFL operates over a spectrogram generated using STFT. In [44] TFL was applied to detect musical patterns, but it can be easily used in other application domains. More recently, TFL was extended in [94] to operate over spectrograms generated using WTs that generally provide spectrograms with a better trade-off between the resolution in the time domain and the resolution in the frequency domain w.r.t. STFT.

4.2 Monitoring Spatio-Temporal Behaviors

The components in CPS are generally distributed across space and connected via a communication infrastructure. The complex behaviour of each individual component due to a fully-integrated hybridisation of computational (logical) and physical action and the interactions between these components via the network enable them to produce very rich and complicated emergent spatiotemporal behaviours, often impossible to predict at design time. Examples include smart grids, robotics teams or collections of genetically engineered living cells. In such examples, temporal logics may be not sufficient to capture also topological spatial requirements. For example, the notion of being *surrounded* or *spatial superpositioning* (averaging resources in a space) are not available in the standard STL and encoding them with specific functions may result cumbersome.

Recently, two different spatiotemporal extensions of STL, Spatial-Temporal Logic (SpaTeL) [61] and the Signal Spatio-Temporal Logic (SSTL) [13, 92] have been proposed to accommodate the growing need of expressing not just temporal but also spatiotemporal requirements in CPS.

SpaTeL [61] is the unification of STL and Tree-Spatial-Superposition-Logic (TSSL) introduced in [11] to classify and detect spatial patterns. TSSL reasons over quadtrees, spatial data structures that are constructed by recursively partitioning the space into uniform quadrants. TSSL uses the notion of spatial superposition (introduced in [60]) that provides a way to describe statistically the distribution of discrete states in a particular partition of the space and that enable to specify self-similar and fractal-like structures that generally characterise the patterns emerging in nature. SpaTeL is equipped with a qualitative (yes/no answer) and a quantitative semantics that provide a measure or robustness of how much the property is satisfied or violated. In [61] this measure of robustness is used as a fitness function to guide the parameter synthesis process for the neighborhood prices in a demand-side management system model of a smart grid using particle swarm optimisation (PSO) algorithms.

SSTL [13, 92] instead extends STL with three spatial modalities *somewhere*, *everywhere* and *surround*, which can be nested arbitrarily with the original STL temporal operator. SSTL is interpreted over a discrete model of space represented as a finite undirected graph. Each edge of the graph is labeled with a positive weight that can be used to represent the distance between two nodes. This provides a metric structure to the space, in terms of shortest path distances. However, the weight can be used to encode also other kind of information (i.e., the average travelling time between two cities). In [92], the authors provide a qualitative and quantitative semantics of SSTL and efficient monitoring algorithms for both semantics.

4.3 Matching and Measuring Temporal Patterns over CPS Behaviors

In the introduction of this document, we mentioned that declarative specification languages are typically based on temporal logics or regular expressions. Both formalisms have their merits and weaknesses. Temporal logic are often good for describing global behaviors and the expected relations between the events and the states that evolve over time. In contrast, regular expressions are convenient for expressing local temporal patterns (consecutive sequences of events at states) that happen in a behavior. In the digital hardware community, it has been observed that the necessary expressiveness and succinctness of the specification language is truly achieved when temporal logic is combined with regular expressions. In fact, both IEEE formal specification language standards, SystemVerilog Assertions (SVA) [116] and Property Specification Language (PSL) [47] adopt this combined approach.

In the context of continuous-time applications, Timed Regular Expressions (TRE) were proposed in [10] as a real-time extension of regular expressions. For a long time, this formalism was subject to theoretical studies, but without any real practical relevance. More recently, a novel algorithm for matching and extracting TRE patterns from hybrid behaviors was developed in [114]. The original offline pattern matching procedure was extended with an online version in [115]. These results on TRE pattern matching enabled the combination of STL with regular expressions also in the continuous-time setting, as it was shown in [56]. Finally, automated extractions of quantitative measurements from CPS behaviors based on TRE patterns was proposed in [57].

5 Applications to Cyber-Physical Systems (CPS)

In this section, we provide an overview of the important applications of temporal logic-based monitoring techniques presented thus far and conclude with a presentation of the practical challenges that need to be addressed through future research.

5.1 Practical Considerations for CPS Monitoring

CPS integrate computation and control of physical processes to enable safety critical applications in many domains including medical devices, automotive systems, avionics and power systems [80]. The problem of monitoring CPS has been a productive area for the runtime verification community as a whole, leading to many important considerations such as monitoring timed properties, quantitative semantics, simulation-guided falsification and online monitoring challenges.

Both offline and online monitoring setups present unique challenges for CPS applications. As described in Section 3, the offline monitoring setup analyzes trace data collected from running a system, after the execution has terminated. A key challenge includes that of monitoring large volumes of data efficiently [21]. At the same time, richer specification languages with higher computational worst-case complexities can be accommodated in an offline monitoring setup, exacerbating the challenge of efficient offline monitoring.

Online monitoring, on the other hand, is constrained by the limited ability to store the trace as the system being monitored executes and the hard real time demands on the computation time. Furthermore, in practice, monitoring is often restricted to perform a single pass through the trace in the forward direction. This naturally restricts us to specification languages that can be monitored efficiently in an online fashion. For instance, the presence of unbounded until operators in the specification can potentially require a large lookahead to resolve the truth of the formula appropriately [62, 58, 33]. Finally, if the monitoring shares the same platform as the deployed system, it should be *non-intrusive* as much as possible: in other words, its consumption of resources such as CPU time, memory and I/O must not interfere with that of the running application [117]. The latter concern is especially acute for CPS, wherein instrumentation can potentially interfere with critical timing properties of the system being observed.

Finally, physical measurements are well known to be noisy and they require specialized sensors. As a result, the problem of monitoring under incomplete and noisy measurements is especially relevant for CPS applications [112, 75, 19].

Another important classification, especially for CPS applications, involves the problem of modeling the physical environment surrounding the closed loop being monitored [71]. Herein, different monitoring setups are distinguished by the complexity and fidelity of the physical plant models and the software runtime setup used [76]. Software-in-the-loop monitors properties using the control system being designed and a mathematical model of the plant. Hardware-in-the-Loop monitoring (also known as Processor in the Loop) executes the controller on the runtime platform used during deployment, while using a mathematical model of the plant [109]. The monitoring challenges include the problem of mapping the "simulation time" of the plant model to the real-time elapsed for the real-time software. Model-based development environments such as Matlab (tm) Simulink/Stateflow(tm), Modelica (tm), Scade(tm) and DSpace(tm) support software/hardware in the loop testing and runtime monitoring for complex CPS [68].

Real-time Monitoring of CPS Monitoring of real systems in real time during their execution requires adapting the techniques and the algorithms presented so far in this document. In the case of real-time monitoring, the property observers are implemented on a physical device that is connected to the system-under-test (SUT) [108, 95]. Several considerations must be taken into account - the real-time sensing of the SUT signals and the environment, the frequency of the monitor operation that must be at least as high as the frequency at which the SUT works, the limited availability of resources that are available on the monitor device, etc. When considering real-time monitors implemented in embedded software or hardware, it is often the case that both the computations are done at the periodic intervals over the quantized input signals. In that case, STL is interpreted over signals that are defined in discrete time and over finite domains. While in this setting, STL is not more expressive than LTL, keeping explicit real-time operators and numerical inputs allows efficient implementation of monitors and allows giving them both qualitative and quantitative semantics. A translation of STL specifications into real-time monitors implemented in FPGA is proposed in [69]. A quantitative semantics for such STL properties based on the *weighted edit distance*, together with the algorithms for computing the robustness degree of a trace with respect to a property are developed in [70].

5.2 Property Falsification and Parameter Synthesis

In this section, we summarize work on the falsification of properties of CPS using robustness of traces (Sec. 2.2). Falsification techniques attempt to find a counterexample to a given property and a model of a system. Falsification is rather valuable approach, generalizing from testing to the automatic search for violating counterexamples. However, the core challenge in falsification is the question of where to search for violations. This is very challenging for CPS due to the continuous nature of the input space which consists of initial conditions and input signals to the model.

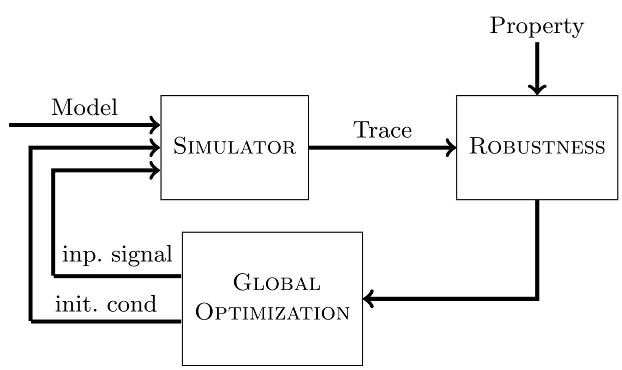

Fig. 2. Setup for the falsification of properties using robustness metrics.

To automate the search, we may use robustness to provide guidance on what to search for. Figure 2 illustrates the overall setup for falsification. As pointed out in [93] and in [2], robustness is a natural measure of a distance between a signal and property. A tool that tries to minimize robustness is in essence a tool for finding counterexamples. To this end, a global optimization engine is used to systematically guide the search for inputs that minimize the overall robustness. The approach does not need to find the globally minimum robustness. Rather it stops whenever the robustness is smaller than a given threshold (usually zero). Also, even if a falsification is not obtained, the minimum robustness obtained can serve as useful information to provide the engineer.

The idea of minimizing robustness to search for falsification has been implemented in tools such as S-Taliro [9] and Breach [39]. However, the challenges lie in the choice of a global optimization solver that can effectively find inputs of lower robustness, leading to a violation. In theory, any such solver will have to fundamentally grapple with the underlying undecidability of finding falsifying inputs for programs, in general. On the other hand, approaches such as Nelder-Meade algorithm [91], Simulated Annealing [93], Ant-Colony Optimization [8], Gaussian Process Optimization with Upper Confidence Bound [15], and the Cross-Entropy method [105] have been used to report success on large examples. Path-planning based methods like RRTs (rapidly exploring random trees) combined with online monitoring of STL robustness have shown promise in systems with hybrid dynamics [45].

As the model fidelity and complexity increases, so does the model simulation time. Long computation times are acceptable in optimal design applications, e.g., [50], since the system must be designed once; however, they can be problematic in system testing applications. Typically, the developers may be willing to wait overnight for test results, but most probably they will not be willing to wait for a week, for example. To improve the performance of stochastic optimization methods, e.g., [93, 8, 15, 105], in [5, 1] they proposed hybrid techniques where robustness descent directions are analytically computed and interleaved with stochastic optimization methods. Such a process guarantees fast convergence to local minimizers. On the other hand, it requires a white-box model, i.e., the mathematical model must be known to the falsification algorithm. Recently, the aforementioned restriction was relaxed in [120] where it was shown that the descent directions can be approximated as long as the system simulator can provide linearizations of the model along the simulation trace of the system.

Automotive Systems Automotive systems present an important application for many aspects of runtime monitoring and property falsification, discussed thus far. As automobiles become ever more autonomous, it is important to check the functional correctness of their core components.

The work in [54] describes the application of S-Taliro to automotive system models. Therein, they show the presence of unexpected behaviors in an automatic transmission model that were not revealed by previous testing approaches. Falsification methods for stochastic systems were applied to stochastic models of automotive systems in [3]. The presented framework for robustness guided falsification in this section is also used as an intermediate step in specification mining methods [119, 66]. The methods presented in [119, 66] are primarily applied to automotive applications. In [121], the authors applied Breach as part of a compositional verification scheme for complex automotive system with many sub-modules. The Breach requirement mining feature was used at the system level to induce pre-conditions for sub-modules, making it easier to apply successfully model-checking analysis at the module level. The contribution in [77] formulated a library of control-theoretic specifications that can be expressed in STL and showed its application to an automotive powertrain control benchmark. Both STL and TRE have been recently used to specify, monitor and measure hybrid properties of the DSI3 standard [96, 57].

CPS Engineering Education STL monitoring was used in the context of a MOOC[3] (Massively Open Online Class) teaching basic concepts of CPS design [74]. A key assignment was for the student to design, simulate and execute on real hardware a control algorithm driving a robot in an environment with obstacles. In order to evaluate hundreds of students contributions, a simulator was designed and equipped with STL monitoring capabilities. Grading was then done by evaluating a set of test cases and STL properties implementing fault monitors, i.e., each STL property evaluated to true would indicate a specific type of fault. The system would then return either some feedback if the user were a student or a partial grade if the user were an instructor.

Systems and Synthetic Biology The growing need of computational models and methods [20] to investigate and to design complex biological systems with a predictable behavior has also benefited greatly from the use of the aforementioned monitoring techniques. STL has become popular also among bioscientists to specify in a concise and unambiguous way the behavior of several cellular and molecular mechanisms. The quantitative semantics of STL and its extensions has triggered the development of several *parameter synthesis* techniques and invaluable tools [15, 40, 11, 14, 61] to automatically characterize the parameter region of a biological model responsible for a behavior of interest. Similarly to falsification analysis, parameter synthesis leverages an optimization process using a particular heuristic. The only difference is that the objective function is to maximize (instead of minimizing) the robustness with respect to an STL requirement. This approach has been successfully employed to study several biological case studies. Examples include the study of the onset of new blood vessel sprouting [41], the programmed cell death (apoptosis) [113], the effect of iron metabolism on blood cell specialization [90] and the logical characterization of an oscillator of the circadian clock in the Ostreococcus Tauri [16].

Medical Devices The growing area of closed-loop medical devices has led to devices such as implantable pacemakers and artificial pancreas that provide life sustaining treatments in real-time. As a result, the problem of monitoring and verifying their operation takes on great significance. The broader area of closed loop medical devices has received a lot of recent interest from the formal verification community. This started with work on pacemakers and implantable cardiac defibrillators (ICDs) that includes hybrid automata models for excitable cells in the heart [99, 17, 59], leading to approaches that employ these models to test closed loop systems [98, 73, 72].

Other examples of safety critical medical devices include the ones used in intensive care. In [28], the authors propose a method to automatically detect ineffective breathing efforts in patients in intensive care subject to assisted ventilation. Their approach is based on learning and monitoring STL specification discriminating between normal and ineffective breaths.

[3] https://www.edx.org/course/cyber-physical-systems-uc-berkeleyx-eecs149-1x

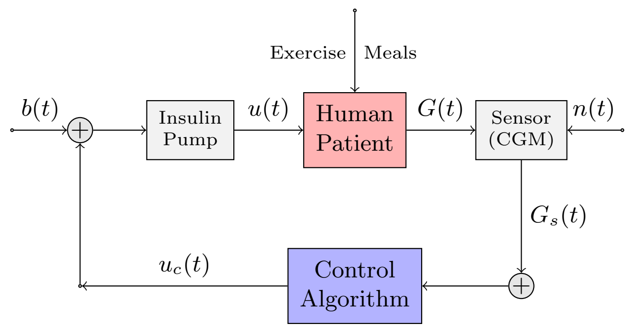

Fig. 3. Overview of the key components of an artificial pancreas control system. $b(t)$: external user commanded insulin, $u(t)$: insulin infused to patient, $G(t)$: blood glucose level of the patient, $n(t)$: sensor measurement error (noise), $G_s(t)$: glucose level estimated/reported by sensor, $u_c(t)$: insulin infusion commanded by the algorithm.

More recently, also the artificial pancreas concept has emerged as an important approach to treat type-1 diabetes, approaching a *de facto* cure [78].

Artificial Pancreas Control Systems The artificial pancreas concept refers to a series of increasingly sophisticated devices that automate the delivery of insulin to patients with type-1 diabetes in a closed loop, automatically responding to changes in the patient's blood glucose levels and activities such as meals and exercise [64, 32, 78]. However, such systems can pose risks to the patient arising from defects and malfunctions. Short-term risks include extremely low blood glucose levels called *hypoglycemia*, that can lead to seizures, loss of consciousness, coma or even death in extreme cases.

Cameron et al use robustness-guided falsification techniques for checking properties of closed loop control systems for the artificial pancreas [30]. Their work investigates a PID controller proposed in [118, 111, 110] based on published descriptions of the control system available. The simulation environment incorporates this controller in a closed loop with models of the patient [89], the sensors and actuators. Their work formulated nearly six different temporal properties of the closed loop and obtained falsification for three of them. However, they could not falsify the remaining three properties that governed the absence of prolonged hypoglycemia and hyperglycemia in the patient.

Another recent study [106] was performed to test a predictive pump shutoff controller designed in [29] that has undergone outpatient clinical trials [84]. This study involved the entire controller software *as is*, without any modifications. At the same time, the closed loop simulation permits us to pose a rich set of questions that compare the closed loop performance with a corresponding open loop under the same meal inputs and physiological model conditions. The falsification discovered adverse noise patterns in the CGM sensor that could trick the Kalman filter into predicting inaccurate forecasts for the future glucose value, and thus prevent appropriate pump shutoff/resumption. At the same time, critical properties such as not commanding excess insulin when the patient is in hypoglycemia could not be violated. The study concluded the need to investigate these violations under more realistic patterns of CGM noise.

6 Tools

Due to relatively low computational complexity of the online and offline monitoring algorithms, many software tools have been developed over the last two decades. Among the first tools that were developed for monitoring (a subset or superset) of Boolean-valued temporal logic specifications were the Temporal Rover [46], MaC [81], Java PathExplorer [62] and LOLA [33]. Since then, there has been a wealth of research on on- and off-line monitoring of requirements expressed in some form of temporal logic (see for example the competition at the *Runtime Verification* conference series [12, 55, 102]) and several publicly available tools have resulted from this effort, for example:

1. RV-Monitor [83]: is available at
   `https://runtimeverification.com/monitor/`
2. MonPoly [22]: is available at
   `http://www.infsec.ethz.ch/research/projects/mon_enf.html`
3. LTLFO2Mon [23] is available at
   `https://github.com/jckuester/ltlfo2mon`

The review in this section focuses only on tools that can reason about real-time properties of traces (output signals) since this is a necessity for testing and monitoring for Embedded and Cyber-Physical Systems. In addition, the focus is on publicly available software tools for off- and on-line monitoring that can be readily downloaded and utilized in testing and monitoring applications. Some of these tools are open source with licenses that allow extensions and redistribution. In the following, we group the software tools survey into two main broader categories: Boolean semantics and multi-valued semantics.

6.1 Software Tools for Boolean Semantics

In the first category, i.e., Boolean semantics, the tool AMT [97] available at:

`http://www-verimag.imag.fr/DIST-TOOLS/TEMPO/AMT/content.html`

analyzes STL properties over analog system output signals. In particular, the properties analyzed by AMT are an extension of the industrial specification language PSL with STL requirements. The software tool AMT is a stand alone executable with a graphical interface where the user provides the STL/PSL properties, the signals and whether the analysis is going to be offline or incremental. In return, the tool plots the Boolean satisfaction of each property over time.

Table 2. Tools for reasoning with multi-valued temporal logics and their functionality.

Analysis	Functionality	Breach	S-Taliro	U-Check
Signal	Offline testing	[39, 42]	[52, 54, 53]	
	Online monitoring	[34]	[36]	
System (best effort)	Falsification	[39]	[2]	
	Coverage Testing		[38]	
	Specification Mining	[119]	[122, 66]	
	Parameter Synthesis	[39]	[107]	[25]
	Conformance		[4]	
Specification	Visual specifications		[67, 65]	
	Debugging		[37]	

6.2 Software Tools for Quantitative Semantics

When considering software tools for evaluating quantitative semantics of STL over signals, there are several options. The following tools are publicly available and they can monitor both real valued and Boolean signals:

1. Breach [39]: available at
   `https://github.com/decyphir/breach`
2. S-Taliro [9]: available at
   `https://sites.google.com/a/asu.edu/s-taliro/`
3. U-Check [25]: available at
   `https://github.com/dmilios/U-check`

Breach and S-Taliro are add-on toolboxes for the Matlab environment while U-Check is a stand-alone program written in Java. Breach and S-Taliro provide analysis tools for black box testing of models and hardware-in-the-loop systems while U-Check deals with stochastic models (Continuous-Time Markov Chains).

The efficient evaluation of STL requirements over real-valued and Boolean signals gave raise to a number of semi-formal verification methods from testing based verification [51] to parameter mining [122, 119] to falsification [2] to synthesis [101]. As reviewed in Section 5, the aforementioned methods have been applied to a wide range of practical applications. Table 2 provides an overview with references of the various analysis methods that each tool supports.

7 Conclusion

Cyber-Physical Systems (CPS) combine heterogeneous and networked computational entities with physical components interacting with them through sensors and actuators. Continuous and hybrid behaviors naturally arise from such dynamical systems. Here, we have provided an in-depth overview of the state-of-the-art techniques for CPS monitoring.

The common denominator of all these methods is the possibility to express in a very powerful, concise and unambiguous way the properties of interest using a formal specification language. In this work, we have mainly focused our attention on Signal Temporal Logic (STL), a formalism enabling the designer to reason about real-time properties over real-valued signals. In the recent years, there has been a great effort to provide efficient algorithms to support online and offline monitoring of STL formulas over (system output) signals.

The introduction of novel quantitative semantics has considerably widened the spectrum of applications from just monitoring qualitatively real-time signals to providing novel falsification analysis and parameter synthesis techniques in model-based testing as well as hardware-in-the-loop testing. As a consequence, the application domains have also grown dramatically, ranging now from automotive systems to synthetic biology and medical devices.

We believe that these techniques will play more and more a key role in industry in the design and engineering safe and resilient CPS and/or to equip them with real-time hardware-based monitors enabling CPS self-awareness and adaptation.

Acknowledgment Ezio Bartocci and Dejan Ničković acknowledge the support of the EU ICT COST Action IC1402 on Runtime Verification beyond Monitoring (ARVI) and of the HARMONIA (845631) project, funded by a national Austrian grant from Austrian FFG under the program IKT der Zukunft. Georgios Fainekos acknowledges the support of the NSF CAREER award 1350420.

References

1. Houssam Abbas and Georgios Fainekos. Computing descent direction of mtl robustness for non-linear systems. In *American Control Conference*, 2013.
2. Houssam Abbas, Georgios E. Fainekos, Sriram Sankaranarayanan, Franjo Ivancic, and Aarti Gupta. Probabilistic temporal logic falsification of cyber-physical systems. *ACM Transactions on Embedded Computing Systems*, 12(s2), May 2013.
3. Houssam Abbas, Bardh Hoxha, Georgios Fainekos, and Koichi Ueda. Robustness-guided temporal logic testing and verification for stochastic cyber-physical systems. In *Proc. of IEEE International Conference on CYBER Technology in Automation, Control, and Intelligent Systems*, 2014.
4. Houssam Abbas, Hans Mittelmann, and Georgios E. Fainekos. Formal property verification in a conformance testing framework. In *Proc. of MEMOCODE 2014: the 12th ACM-IEEE International Conference on Formal Methods and Models for System Design*, pages 155–164. IEEE, 2014.
5. Houssam Abbas, Andrew Winn, Georgios E. Fainekos, and A. Agung Julius. Functional gradient descent method for metric temporal logic specifications. In *Proc. of ACC 2014: the American Control Conference*, pages 2312–2317. IEEE, 2014.
6. Takumi Akazaki and Ichiro Hasuo. Time robustness in MTL and expressivity in hybrid system falsification. In *Proc. of CAV 2015: the 27th International Conference on Computer Aided Verification*, volume 9207 of *LNCS*, pages 356–374. Springer, 2015.
7. Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996.

8. Yashwanth Singh Rahul Annapureddy and Georgios E. Fainekos. Ant colonies for temporal logic falsification of hybrid systems. In *Proceedings of the 36th Annual Conference of IEEE Industrial Electronics*, pages 91 – 96, 2010.
9. Yashwanth Annpureddy, Che Liu, Georgios E. Fainekos, and Sriram Sankaranarayanan. S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In *Proc. of TACAS 2011: the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 6605 of *LNCS*, pages 254–257, 2011.
10. Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *Journal of ACM*, 49(2):172–206, 2002.
11. Ebru Aydin-Gol, Ezio Bartocci, and Calin Belta. A formal methods approach to pattern synthesis in reaction diffusion systems. In *Proc. of CDC 2014: the 53rd IEEE Conference on Decision and Control*, pages 108–113. IEEE, 2014.
12. Ezio Bartocci, Borzoo Bonakdarpour, and Ylies Falcone. First international competition on software for runtime verification. In *Runtime Verification*, volume 8734 of *LNCS*, pages 1–9. Springer, 2014.
13. Ezio Bartocci, Luca Bortolussi, Dimitrios Milios, Laura Nenzi, and Guido Sanguinetti. Studying emergent behaviours in morphogenesis using signal spatio-temporal logic. In *Proc. of HSB 2015: the Fourth International Workshop on Hybrid Systems and Biology*, volume 9271 of *LNCS*, pages 156–172. Springer, 2015.
14. Ezio Bartocci, Luca Bortolussi, and Laura Nenzi. A temporal logic approach to modular design of synthetic biological circuits. In *Proc. of CMSB 2013: the 11th International Conference on Computational Methods in Systems Biology*, volume 8130 of *LNBI*, pages 164–177. Springer, 2013.
15. Ezio Bartocci, Luca Bortolussi, Laura Nenzi, and Guido Sanguinetti. System design of stochastic models using robustness of temporal properties. *Theor. Comput. Sci.*, 587:3–25, 2015.
16. Ezio Bartocci, Luca Bortolussi, and Guido Sanguinetti. Data-driven statistical learning of temporal logic properties. In *Proc. of FORMATS 2014: the 12th International Conference on Formal Modeling and Analysis of Timed Systems*, volume 8711 of *LNCS*, pages 23–37. Springer, 2014.
17. Ezio Bartocci, Flavio Corradini, Maria Rita Di Berardini, Emilia Entcheva, Scott A. Smolka, and Radu Grosu. Modeling and simulation of cardiac tissue using hybrid I/O automata. *Theor. Comput. Sci.*, 410(33-34):3149–3165, 2009.
18. Ezio Bartocci, Flavio Corradini, Emanuela Merelli, and Luca Tesei. Detecting synchronisation of biological oscillators by model checking. *Theor. Comput. Sci.*, 411(20):1999–2018, 2010.
19. Ezio Bartocci, Radu Grosu, Atul Karmarkar, Scott A. Smolka, Scott D. Stoller, Erez Zadok, and Justin Seyster. Adaptive runtime verification. In *Proc. of RV 2012: the Third International Conference on Runtime Verification*, volume 7687 of *LNCS*, pages 168–182. Springer, 2013.
20. Ezio Bartocci and Pietro Liò. Computational modeling, formal analysis, and tools for systems biology. *PLoS Computational Biology*, 12(1), 2016.
21. David Basin, Germano Caronni, Sarah Ereth, Matúš Harvan, Felix Klaedtke, and Heiko Mantel. *Scalable Offline Monitoring*, volume 8734 of *Lecture Notes in Computer Science*, pages 31–47. Springer, 2014.
22. David A. Basin, Matús Harvan, Felix Klaedtke, and Eugen Zalinescu. MONPOLY: monitoring usage-control policies. In *Runtime Verification (RV)*, volume 7186 of *LNCS*, pages 360–364. Springer, 2011.

23. Andreas Bauer, Jan-Christoph Küster, and Gil Vegliach. From propositional to first-order monitoring. In *Runtime Verification (RV)*, volume 8174 of *LNCS*, pages 59–75. Springer, 2013.
24. Andreas Bauer, Martin Leucker, and Christian Schallhart. Monitoring of real-time properties. In *Proc. of FSTTCS 2006: the 26th International Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 4337 of *LNCS*, pages 260–272. Springer, 2006.
25. Luca Bortolussi, Dimitrios Milios, and Guido Sanguinetti. U-Check: Model checking and parameter synthesis under uncertainty. In *12th International Conference Quantitative Evaluation of Systems*, pages 89–104. Springer International Publishing, 2015.
26. Lubos Brim, Petr Dluhos, David Safránek, and Tomas Vejpustek. Stl*: Extending signal temporal logic with signal-value freezing operator. *Inf. Comput.*, 236:52–67, 2014.
27. Lubos Brim, Tomas Vejpustek, David Safránek, and Jana Fabriková. Robustness analysis for value-freezing signal temporal logic. In *Proc. of HSB 2013: the Second International Workshop on Hybrid Systems and Biology*, volume 125 of *EPTCS*, pages 20–36, 2013.
28. Sara Bufo, Ezio Bartocci, Guido Sanguinetti, Massimo Borelli, Umberto Lucangelo, and Luca Bortolussi. Temporal logic based monitoring of assisted ventilation in intensive care patients. In *Proc. of ISoLA 2014: the 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Part II*, volume 8803 of *LNCS*, pages 391–403. Springer, 2014.
29. F. Cameron, Darrell M. Wilson, Bruce A. Buckingham, Hasmik Arzumanyan, Paula Clinton, H. Peter Chase, John Lum, David M. Maahs, Peter M. Calhoun, and B. Wayne Bequette. Inpatient studies of a kalman-filter-based predictive pump shutoff algorithm. *J. Diabetes Science and Technology*, 6(5):1142–1147, 2012.
30. Faye Cameron, Georgios Fainekos, David M. Maahs, and Sriram Sankaranarayanan. Towards a verified artificial pancreas: Challenges and solutions for runtime verification. In *Proceedings of Runtime Verification (RV'15)*, volume 9333 of *Lecture Notes in Computer Science*, pages 3–17, 2015.
31. Aleksandar Chakarov, Sriram Sankaranarayanan, and Georgios E. Fainekos. Combining time and frequency domain specifications for periodic signals. In *Proc. of RV 2011: the Second International Conference on Runtime Verification*, volume 7186 of *LNCS*, pages 294–309. Springer, 2011.
32. Claudio Cobelli, Chiara Dalla Man, Giovanni Sparacino, Lalo Magni, Giuseppe De Nicolao, and Boris P. Kovatchev. Diabetes: Models, signals and control (methodological review). *IEEE reviews in biomedical engineering*, 2:54–95, 2009.
33. B. D'Angelo, S. Sankaranarayanan, C. Sanchez, W. Robinson, B. Finkbeiner, H.B. Sipma, S. Mehrotra, and Z. Manna. LOLA: runtime monitoring of synchronous systems. In *12th International Symposium on Temporal Representation and Reasoning (TIME)*, 2005.
34. Jyotirmoy V. Deshmukh, Alexandre Donzé, Shromona Ghosh, Xiaoqing Jin, Juniwal. Garvit, and Sanjit A. Seshia. Robust online monitoring of signal temporal logic. In *Proc. of RV 2015: the 6th International Conference on Runtime Verification*, volume 9333 of *LNCS*, pages 55–70. Springer, 2015.
35. Jyotirmoy V. Deshmukh, Rupak Majumdar, and Vinyak S. Prabhu. Quantifying conformance using the Skorokhod metric. In *Proc. of CAV 2015: the 27th International Conference on Computer Aided Verification*, volume 9207 of *LNCS*, pages 234–250, 2015.

36. Adel Dokhanchi, Bardh Hoxha, and Georgios E. Fainekos. On-line monitoring for temporal logic robustness. In *Proc. of RV 2014: the 5th International Conference on Runtime Verification*, volume 8734, pages 231–246. Springer, 2014.
37. Adel Dokhanchi, Bardh Hoxha, and Georgios E. Fainekos. Metric interval temporal logic specification elicitation and debugging. In *Proc. of MEMOCODE 2015: the 13th ACM/IEEE International Conference on Formal Methods and Models for Codesign*, pages 70–79. IEEE, 2015.
38. Adel Dokhanchi, Aditya Zutshi, Rahul T. Sriniva, Sriram Sankaranarayanan, and Georgios Fainekos. Requirements driven falsification with coverage metrics. In *12th International Conference on Embedded Software (EMSOFT)*, 2015.
39. Alexandre Donzé. Breach, A toolbox for verification and parameter synthesis of hybrid systems. In *Proc. of CAV 2010: the 22nd International Conference on Computer Aided Verification*, volume 6174 of *LNCS*, pages 167–170. Springer, 2010.
40. Alexandre Donzé, Gilles Clermont, Axel Legay, and Christopher James Langmead. Parameter synthesis in nonlinear dynamical systems: Application to systems biology. In *Proc. of RECOMB 2009: the 13th Annual International Conference on Research in Computational Molecular Biology*, volume 5541 of *LNCS*, pages 155–169. Springer, 2009.
41. Alexandre Donzé, Eric Fanchon, Lucie Martine Gattepaille, Oded Maler, and Philippe Tracqui. Robustness analysis and behavior discrimination in enzymatic reaction networks. *PLoS ONE*, 6(9):e24246, 09 2011.
42. Alexandre Donzé, Thomas Ferrère, and Oded Maler. Efficient robust monitoring for STL. In *Proc. of CAV 2013: the 25th International Conference on Computer Aided Verification*, volume 8044 of *LNCS*, pages 264–279. Springer, 2013.
43. Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *Proc. of FORMATS'10: the 8th International Conference on Formal Modeling and Analysis of Timed Systems*, volume 6246 of *LNCS*, pages 92–106. Springer, 2010.
44. Alexandre Donzé, Oded Maler, Ezio Bartocci, Dejan Ničković, Radu Grosu, and Scott A. Smolka. On temporal logic and signal processing. In *Proc. of ATVA 2012: the 10th International Symposium on Automated Technology for Verification and Analysis*, volume 7561 of *LNCS*, pages 92–106. Springer, 2012.
45. Tommaso Dreossi, Thao Dang, Alexandre Donzé, James Kapinski, Xiaoqing Jin, and Jyotirmoy V. Deshmukh. Efficient guiding strategies for testing of temporal properties of hybrid systems. In *Proc. of NFM 2015: the 7th International Symposium on NASA Formal Methods*, volume 9058 of *LNCS*, pages 127–142. Springer, 2011.
46. Doron Drusinsky. Monitoring temporal rules combined with time series. In *Computer Aided Verification*, volume 2725 of *LNCS*, pages 114–117, 2003.
47. Cindy Eisner and Dana Fisman. *A practical introduction to PSL*. Springer, 2006.
48. Cindy Eisner, Dana Fisman, and John Havlicek. A topological characterization of weakness. In *Proc. of PODC 2005: the Twenty-Fourth Annual ACM Symposium on Principles of Distributed Computing*, pages 1–8. ACM, 2005.
49. Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. Reasoning with temporal logic on truncated paths. In *Proc. of CAV 2003: the 15th International Conference on Computer Aided Verification*, volume 2725 of *LNCS*, pages 27–39. Springer, 2003.
50. Georgios E. Fainekos and Kyriakos C. Giannakoglou. Inverse design of airfoils based on a novel formulation of the ant colony optimization method. *Inverse Problems in Engineering*, 11(1):21–38, 2003.

51. Georgios E. Fainekos, Antoine Girard, and George J. Pappas. Temporal logic verification using simulation. In *Proc. of FORMATS 2006: the 4th International Conference on Formal Modelling and Analysis of Timed Systems*, volume 4202 of *LNCS*, pages 171–186. Springer, 2006.
52. Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications. In *Proc. of FATES 2006: First Combined International Workshops on Formal Approaches to Testing and Runtime Verification*, volume 4262 of *LNCS*, pages 178–192. Springer, 2006.
53. Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.*, 410(42):4262–4291, 2009.
54. Georgios E. Fainekos, Sriram Sankaranarayanan, Koichi Ueda, and Hakan Yazarel. Verification of automotive control applications using S-TaLiRo. In *Proc. of ACC 2012: American Control Conference*, pages 3567–3572, 2012.
55. Ylies Falcone, Dejan Nickovic, Giles Reger, and Daniel Thoma. Second international competition on software for runtime verification. In *Runtime Verification*, volume 9333 of *LNCS*, pages 405–422. Springer, 2015.
56. Thomas Ferrère. Assertions et mesures pour la simulation en signaux mixtes. 2016.
57. Thomas Ferrère, Oded Maler, Dejan Ničković, and Dogan Ulus. Measuring with timed patterns. In *Proc. of CAV 2015: the 27th International Conference on Computer Aided Verification*, volume 9207 of *LNCS*, pages 322–337. Springer, 2015.
58. Bernd Finkbeiner and Henny B. Sipma. Checking finite traces using alternating automata. In Klaus Havelund and Grigore Rosu, editors, *Runtime Verification 2001*, volume 55 of *Electronic Notes in Theoretical Computer Science*, pages 44–60. Elsevier, July 2001. Extended version to appear in *Formal Methods in System Design*.
59. Radu Grosu, Grégory Batt, Flavio H. Fenton, James Glimm, Colas Le Guernic, Scott A. Smolka, and Ezio Bartocci. From cardiac cells to genetic regulatory networks. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *LNCS*, pages 396–411. Springer, 2011.
60. Radu Grosu, Scott A. Smolka, Flavio Corradini, Anita Wasilewska, Emilia Entcheva, and Ezio Bartocci. Learning and detecting emergent behavior in networks of cardiac myocytes. *Commun. ACM*, 52(3):97–105, 2009.
61. Iman Haghighi, Austin Jones, Zhaodan Kong, Ezio Bartocci, Radu Grosu, and Calin Belta. SpaTeL: a novel spatial-temporal logic and its applications to networked systems. In *Proc. of HSCC'15: the 18th International Conference on Hybrid Systems: Computation and Control*, pages 189–198. IEEE, 2015.
62. Klaus Havelund and Grigore Rosu. Monitoring java programs with java pathexplorer. *Electronic Notes in Theoretical Computer Science*, 55(2):200–217, 2001.
63. Hsi-Ming Ho, Joël Ouaknine, and James Worrell. Online monitoring of metric temporal logic. In *Proc. of RV 2014: the 5th International Conference on Runtime Verification*, volume 8734 of *LNCS*, pages 178–192. Springer, 2014.
64. Roman Hovorka. Continuous glucose monitoring and closed-loop systems. *Diabetic Medicine*, 23(1):1–12, 2005.
65. Bardh Hoxha, Hoang Bach, Houssam Abbas, Adel Dokhanci, Yoshihiro Kobayashi, and Georgios Fainekos. Towards formal specification visualization for testing and monitoring of cyber-physical systems. In *International Workshop on Design and Implementation of Formal Tools and Systems, DIFTS'14*, 2014.

66. Bardh Hoxha, Adel Dokhanchi, and Georgios Fainekos. Mining parametric temporal logic properties in model based design for cyber-physical systems. *International Journal on Software Tools for Technology Transfer (In press)*, 2017.
67. Bardh Hoxha, Nikolaos Mavridis, and Georgios E. Fainekos. VISPEC: A graphical tool for elicitation of MTL requirements. In *Proc. of IROS 2015: the 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3486–3492. IEEE, 2015.
68. Mathworks Inc. Test generated code with sil and pil simulations. Cf. `https://www.mathworks.com/help/ecoder/examples/software-and-processor-in-the-loop-sil-and-pil-simulation.html`.
69. Stefan Jaksic, Ezio Bartocci, Radu Grosu, Reinhard Kloibhofer, Thang Nguyen, and Dejan Nickovic. From signal temporal logic to FPGA monitors. In *Proc. of MEMOCODE 2015: the 13. ACM/IEEE International Conference on Formal Methods and Models for Codesign*, pages 218–227. IEEE, 2015.
70. Stefan Jaksic, Ezio Bartocci, Radu Grosu, and Dejan Nickovic. Quantitative monitoring of STL with edit distance. In *Runtime Verification - 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings*, volume 10012 of *LNCS*, pages 201–218, 2016.
71. Jeff C. Jensen, Danica H. Chang, and Edward A. Lee. A model-based design methodology for cyber-physical systems. In *Proc. of IEEE Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems (CyPhy)*, 2011.
72. Zhihao Jiang, Miroslav Pajic, Rajeev Alur, and Rahul Mangharam. Closed-loop verification of medical devices with model abstraction and refinement. *International Journal on Software Tools for Technology Transfer*, pages 1–23, 2013.
73. Zhihao Jiang, Miroslav Pajic, Salar Moarref, Rajeev Alur, and Rahul Mangharam. Modeling and verification of a dual chamber implantable pacemaker. In *Proc. of TACAS 2012: the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214 of *LNCS*, pages 188–203, 2012.
74. Garvit Juniwal, Alexandre Donzé, Jeff C. Jensen, and Sanjit A. Seshia. CPSGrader: Synthesizing temporal logic testers for auto-grading an embedded systems laboratory. In *Proc. of EMSOFT 2014: the 2014 International Conference on Embedded Software*, pages 24:1–24:10. IEEE, 2014.
75. Kenan Kalajdzic, Ezio Bartocci, Scott A. Smolka, Scott D. Stoller, and Radu Grosu. Runtime verification with particle filtering. In *Proc. of RV 2013: the 4th International Conference on Runtime Verification*, volume 8174 of *LNCS*, pages 149–166. Springer, 2013.
76. Aaron Kane. *Runtime Monitoring for Safety-Critical Embedded Systems*. PhD thesis, Carnegie Mellon University, College of Engineering, 2015.
77. James Kapinski, Xiaoqing Jin, Jyotirmoy Deshmukh, Alexandre Donzé, Tomoya Yamaguchi, Hisahiro Ito, Tomoyuki Kaga, Shunsuke Kobuna, and Sanjit Seshia. ST-Lib: A library for specifying and classifying model behaviors. In *SAE Technical Paper*. SAE International, 2016.
78. Aaron Kowalski. Pathway to artificial pancreas revisited: Moving downstream. *Diabetes Care*, 38:1036–1043, June 2015.
79. Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
80. E. A. Lee. Cyber physical systems: Design challenges. In *ISORC'11*, pages 363–369, May 2008.

81. I. Lee, S. Kannan, M. Kim, O. Sokolsky, and M. Viswanathan. Runtime assurance based on formal specifications. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, 1999.
82. Daniel Lemire. Streaming maximum-minimum filter using no more than three comparisons per element. *arXiv preprint cs/0610046*, 2006.
83. Qingzhou Luo, Yi Zhang, Choonghwan Lee, Dongyun Jin, Patrick O'Neil Meredith, Traian-Florin Serbanuta, and Grigore Rosu. RV-Monitor: Efficient parametric runtime verification with simultaneous properties. In *Runtime Verification (RV)*, volume 8734 of *LNCS*, pages 285–300. Springer, 2014.
84. David M. Maahs, Peter Calhoun, Bruce A. Buckingham, and Others. A randomized trial of a home system to reduce nocturnal hypoglycemia in type 1 diabetes. *Diabetes Care*, 37(7):1885–1891, July 2014.
85. Rupak Majumdar and Vinayak S. Prabhu. Computing the Skorokhod distance between polygonal traces. In *Proc. of HSCC'15: the 18th International Conference on Hybrid Systems: Computation and Control*, pages 199–208. ACM, 2015.
86. Rupak Majumdar and Vinayak S. Prabhu. Computing distances between reach flowpipes. In *Proc. HSCC 2016: the 19th International Conference on Hybrid Systems: Computation and Control*, pages 267–276. ACM, 2016.
87. Oded Maler. Some thoughts on runtime verification. In *Runtime Verification (RV)*, volume 10012 of *LNCS*, pages 3–14. Springer, 2016.
88. Oded Maler and Dejan Ničković. Monitoring properties of analog and mixed-signal circuits. *STTT*, 15(3):247–268, 2013.
89. Chiara Dalla Man, Davide M. Raimondo, Robert A. Rizza, and Claudio Cobelli. GIM, simulation software of meal glucose-insulin model. *J. Diabetes Sci. and Tech.*, 1(3), May 2007.
90. Nicolas Mobilia, Alexandre Donzé, Jean Marc Moulis, and Eric Fanchon. Producing a set of models for the iron homeostasis network. In *Proc, of HSB 2013: the second International Workshop on Hybrid Systems and Biology*, volume 125 of *EPTCS*, pages 92–98, 2013.
91. John A. Nelder and Roger Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.
92. Laura Nenzi, Luca Bortolussi, Vincenzo Ciancia, Michele Loreti, and Mieke Massink. Qualitative and quantitative monitoring of spatio-temporal properties. In *Proc. of RV 2015: the 6th International Conference on Runtime Verification*, volume 9333 of *LNCS*, pages 21–37. Springer, 2015.
93. Truong Nghiem, Sriram Sankaranarayanan, Georgios E. Fainekos, Franjo Ivancic, Aarti Gupta, and George J. Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, pages 211–220, 2010.
94. Luan Nguyen, James Kapinski, Xiaoqing Jin, Jyotirmoy Deshmukh, Ken Butts, , and Taylor Johnson. Abnormal data classification using time-frequency temporal logic. In *Proc. of 20th ACM International Conference on Hybrid Systems: Computation and Control*, page to appear. ACM, 2017.
95. Thang Nguyen, Ezio Bartocci, Dejan Nickovic, Radu Grosu, Stefan Jaksic, and Konstantin Selyunin. The HARMONIA project: Hardware monitoring for automotive systems-of-systems. In *Proc. of ISoLA 2016: the 7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications, Part II*, volume 9953 of *LNCS*, pages 371–379, 2016.

96. Thang Nguyen and Dejan Ničković. Assertion-based monitoring in practice - checking correctness of an automotive sensor interface. In *Proc. of FMICS 2014: the 19th International Conference on Formal Methods for Industrial Critical Systems*, LNCS, pages 16–32. Springer, 2014.
97. Dejan Nickovic and Oded Maler. AMT: A property-based monitoring tool for analog systems. In *FORMATS*, volume 4763 of *LNCS*, pages 304–319. Springer, 2007.
98. M. Pajic, R. Mangharam, O. Sokolsky, D. Arney, J. Goldman, and I Lee. Model-driven safety analysis of closed-loop medical systems. *Industrial Informatics, IEEE Transactions on*, 10(1):3–16, Feb 2014.
99. Y. Pei, E. Entcheva, R. Grosu, and S.A. Smolka. Efficient modeling of excitable cells using hybrid automata. In *Proc. Computational Methods in Systems Biology*, pages 216–227, 2005.
100. Amir Pnueli. The temporal logic of programs. In *Proc. of the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.
101. Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M. Murray, and Sanjit A. Seshia. Reactive synthesis from signal temporal logic specifications. In *Proc. HSCC'15: the 18th International Conference on Hybrid Systems: Computation and Control*, pages 239–248. ACM, 2015.
102. Giles Reger, Sylvain Halle, and Ylies Falcone. Third international competition on software for runtime verification. In *Runtime Verification*, volume 10012 of *LNCS*, pages 21–37. Springer, 2016.
103. Aurélien Rizk, Grégory Batt, François Fages, and Sylvain Soliman. On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In *Proc. of CMSB 2008: the 6th International Conference on Computational Methods in Systems Biology*, volume 5307, pages 251–268. Springer, 2008.
104. Alena Rodionova, Ezio Bartocci, Dejan Nickovic, and Radu Grosu. Temporal logic as filtering. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, HSCC 2016, Vienna, Austria, April 12-14, 2016*, pages 11–20. ACM, 2016.
105. Sriram Sankaranarayanan and Georgios Fainekos. Falsification of temporal properties of hybrid systems using the cross-entropy method. In *ACM International Conference on Hybrid Systems: Computation and Control*, 2012.
106. Sriram Sankaranarayanan, Suhas Akshar Kumar, Faye Cameron, B. Wayne Bequette, Georgios Fainekos, and David M. Maahs. Model-based falsification of an artificial pancreas control system. *ACM SIGBED Review (Special Issue on Medical Cyber Physical Systems)*, 2016. To Appear, 2017.
107. Sriram Sankaranarayanan, Christopher Miller, Rangarajan Raghunathan, Hadi Ravanbakhsh, and Georgios E. Fainekos. A model-based approach to synthesizing insulin infusion pump usage parameters for diabetic patients. In *Proc. of the 50th Annual Allerton Conference on Communication, Control, and Computing*, pages 1610–1617. IEEE, 2012.
108. Konstantin Selyunin, Thang Nguyen, Ezio Bartocci, and Radu Grosu. Applying runtime monitoring for automotive electronic development. In *Proc. of RV 2016: the 16th International Conference on Runtime Verification*, volume 10012 of *LNCS*, pages 462–469. Springer, 2016.
109. M. Short and M. J. Pont. Hardware in the loop simulation of embedded automotive control system. In *Proceedings. 2005 IEEE Intelligent Transportation Systems, 2005.*, pages 426–431, Sept 2005.

110. Garry M. Steil. Algorithms for a closed-loop artificial pancreas: The case for proportional-integral-derivative control. *J. Diabetes Sci. Technol.*, 7:1621–1631, November 2013.
111. G.M. Steil, A.E. Panteleon, and K. Rebrin. Closed-sloop insulin delivery - the path to physiological glucose control. *Advanced Drug Delivery Reviews*, 56(2):125–144, 2004.
112. Scott D. Stoller, Ezio Bartocci, Justin Seyster, Radu Grosu, Klaus Havelund, Scott A. Smolka, and Erez Zadok. *Runtime Verification with State Estimation*, volume 7186 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 2012.
113. Szymon Stoma, Alexandre Donzé, François Bertaux, Oded Maler, and Grégory Batt. STL-based analysis of TRAIL-induced apoptosis challenges the notion of type I/type II cell line classification. *PLoS Computational Biology*, 9(5), 2013.
114. Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler. Timed pattern matching. In *Proc. of FORMATS 2014: the 12th International Conference on Formal Modeling and Analysis of Timed Systems*, volume 8711 of *LNCS*, pages 222–236. Springer, 2014.
115. Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler. Online timed pattern matching using derivatives. In *Proc. of TACAS 2016: the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 9636 of *LNCS*, pages 736–751. Springer, 2016.
116. Srikanth Vijayaraghavan and Meyyappan Ramanathan. *A practical guide for SystemVerilog assertions*. Springer, 2006.
117. C. Watterson and D. Heffernan. Runtime verification and monitoring of embedded systems. *IET Software*, 1(5):172–179, October 2007.
118. S Weinzimer, G Steil, K Swan, J Dziura, N Kurtz, and W. Tamborlane. Fully automated closed-loop insulin delivery versus semiautomated hybrid control in pediatric patients with type 1 diabetes using an artificial pancreas. *Diabetes Care*, 31:934–939, 2008.
119. Jin Xiaoqing, Alexandre Donzé, Jyotirmoy V. Deshmukh, and Sanjit A. Seshia. Mining Requirements from Closed-loop Control Models. In *Proc. of HSCC 2013: the ACM International Conference on Hybrid Systems: Computation and Control*, pages 43–52. ACM, 2013.
120. Shakiba Yaghoubi and Georgios Fainekos. Hybrid approximate gradient and stochastic descent for falsification of nonlinear systems. In *American Control Conference*, 2017.
121. Tomoya Yamaguchi, Tomoyuki Kaga, Alexandre Donzé, and Sanjit A. Seshia. Combining requirement mining, software model checking, and simulation-based verification for industrial automotive systems. In *Proceedings of the IEEE International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, October 2016.
122. Hengyi Yang, Bardh Hoxha, and Georgios Fainekos. Querying Parametric Temporal Logic Properties on Embedded Systems. In *Proc. of ICTSS 2012: the 24th IFIP WG 6.1 International Conference on Testing Software and Systems*, volume 7641 of *LNCS*, pages 136–151. Springer, 2012.