Clairvoyant Monitoring for Signal Temporal Logic

Xin Qin(✉) and Jyotirmoy V. Deshmukh(✉)

University of Southern California, Los Angeles, USA
{xinqin,jyotirmoy.deshmukh}@usc.edu

Abstract. In this paper, we consider the problem of monitoring temporal patterns expressed in Signal Temporal Logic (STL) over time-series data in a *clairvoyant* fashion. Existing offline or online monitoring algorithms can only compute the satisfaction of a given STL formula on the time-series data that is available. We use off-the-shelf statistical time-series analysis techniques to fit available data to a model and use this model to forecast future signal values. We derive the joint probability distribution of predicted signal values and use this to compute the satisfaction probability of a given signal pattern over the prediction horizon. There are numerous potential applications of such prescient detection of temporal patterns. We demonstrate practicality of our approach on case studies in automated insulin delivery, unmanned aerial vehicles, and household power consumption data.

1 Introduction

Safety-critical cyber-physical systems (CPS) such as autonomous ground vehicles, unmanned aerial vehicles and medical devices often operate in highly uncertain and noisy environments. It is often impossible to anticipate all possible exogenous inputs to such systems at design-time; and most designers typically test their applications in only a finite number of scenarios. An alternative approach is to perform runtime monitoring to ensure that such systems do not have catastrophic failures of safety. A key aspect of runtime monitoring is the ability to raise alarms when the violation of a safety property is detected.

There has been considerable amount of recent work on the use of real-time temporal logics such as Signal Temporal Logic (STL) to specify correctness properties of safety-critical CPS applications [1,4,5,15,16,18,19,22]. Essentially, STL allows specification of properties of real-valued signals defined over dense time. A basic building block of an STL formula is a signal predicate (i.e. some condition over signal values for a given time), and general STL formulas can be constructed by combining signal predicates using Boolean ($\wedge, \vee, \neg$) or temporal (such as always, eventually, etc.) operators. In addition to Boolean satisfaction of a formula φ by a signal trace $x(t)$, *quantitative semantics* for STL allow us to define a *robust satisfaction value* or *robustness* which can be viewed as a signed distance between the signal $x(t)$ and the set of signals satisfying (or violating) φ, where a positive (resp. negative) sign indicates that φ is satisfied (resp. violated).

N. Bertrand and N. Jansen (Eds.): FORMATS 2020, LNCS 12288, pp. 178–195, 2020.
https://doi.org/10.1007/978-3-030-57628-8_11

Existing algorithms for monitoring STL specifications are either *offline* or *online*, and either compute the Boolean satisfaction or the robustness value. Offline algorithms assume that the entire signal trace is available, while online algorithms can estimate satisfaction or violation when only a prefix of a given signal is available. Online algorithms potentially provide early detection of safety violations of the system; however, by their nature, online algorithms are limited in identifying violations "as they happen." In this paper, we propose a new class of algorithms for *clairvoyant monitoring* that go beyond existing online algorithms by predicting future signal values and give probabilistic bounds on the satisfaction or violation of the STL formula in the future. Our notion of clairvoyance is derived from literature on statistical techniques for forecasting signals. In this paper, we focus on *signal patterns* specified by STL formulas, i.e., instead of the traditional use of STL to express formulas that are satisfied or violated true over an entire trace, we focus on bounded horizon STL formulas that are evaluated over the given prediction horizon[1].

To understand the motivation for clairvoyant monitoring using patterns specified in STL, consider a weather forecasting system that decides the advent of winter by checking if the forecasted temperature is lower than a certain threshold for a certain number of days. A single day of forecasted low temperature can easily be an outlier, and hence we want to estimate the probability of a certain event repeating for a number of days. Such a specification can be easily expressed in STL: $\mathbf{F}_{[0,10]}\mathbf{G}_{[0,5]}(\theta < 40)$. This specification says that in the next 10 days, there is some 5 day period where the temperature is consistently lower than 40 °F.

Our clairvoyant monitoring framework consists of three main components: (1) a *predictor* that uses past values of a signal to produce n predictions of the signal value at future time-points, (2) an algorithm to *enumerate possible scenarios* in which the given STL formula may be satisfied, and, (3) a *probability estimator*, that, given a target robustness value of the STL specification, computes the probability of exceeding that value.

We demonstrate clairvoyant monitoring on three applications: (1) monitoring hypo- and hyper-glycemia conditions in an automated insulin delivery system model, (2) monitoring safety of an unmanned aerial vehicle, and (3) monitoring power consumption.

The main technical contributions of our paper are:

1. For a given STL formula, a technique to automatically enumerate each distinct conjunction of signal predicates that lead to formula satisfaction.
2. We use statistical time-series analysis techniques to forecast future signal values, and we derive the joint probability distribution across the predicted time-points. We assume that the data can be modeled as a realization of an ARMA or ARIMA process. In case these models are not a good fit, the methods in this paper are not applicable.

[1] We can easily extend clairvoyant monitoring of unbounded horizon STL formulas over entire traces by considering the notion of nominal robustness [10]. This would also require us to track the robustness over the signal prefix.

3. We use basic laws of probability to compute the probability of a signal satisfying or violating a given STL formula φ utilizing the above two results. We can also compute the probability of the robust satisfaction value of the predicted signal (w.r.t. φ) exceeding or falling below a given threshold.

1.1 Illustrative Example

We use scenario depicted in Fig. 1 to illustrate the clairvoyant monitoring technique presented in this paper.

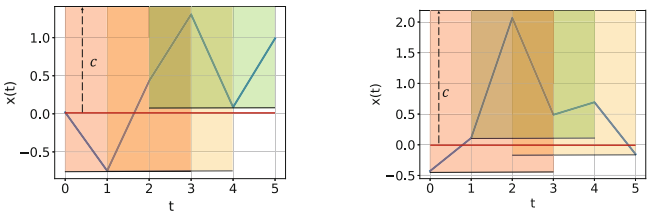


Fig. 1. A signal behavior satisfies an STL pattern – expressed as a disjunction over conjunctions of signal predicates – as long as there exists one satisfying disjunct. Figure (1a)(1b) shows two different possible scenarios that the signal in future satisfies STL formula φ_1 in Eq. (1.1). Each green block indicates one satisfying conjunction of signal predicates, where four consecutive time steps all have signal values greater than zero. (Color figure online)

Suppose we are observing a series of data generated by a system and at time t, we want to know if the formula φ_1 in Eq. (1.1) is true over a prediction horizon of length 6.

$$\varphi_1 \equiv \mathbf{F}_{[0,2]}\mathbf{G}_{[0,3]}(x(t) \geq 0) \tag{1.1}$$

Figure 1 shows a subset of possible predicate conjunctions that make the inner formula $\mathbf{G}$ true, and in this case φ_1 also. If we know the joint probability distribution of signal values within the prediction horizon, we can use marginal distributions and inclusion-exclusion principle to calculate the probability of the STL formula to be satisfied.

In order to predict future signal values and also to compute joint probability distribution over these predictions, we rely on statistical time-series models such as auto-regressive and moving-average models.

2 Background on Signal Temporal Logic

Definition 2.1 (Univariate Signal, Time Horizon). *A* time domain $\mathbb{T}$ *is a finite set of uniform time instants* $\{t_0, t_1, \ldots, t_N\}$ *where* $t_0 = 0$*, and* $t_i \in \mathbf{R}^{\geq 0}$*, and* $t_{i+1} - t_i = \Delta$*, for some* $\Delta \in \mathbf{R}^{>0}$*.Let* $\mathcal{D}$ *be a bounded subset of* $\mathbf{R}$*. A* signal

x (also called a trace *or* time-series *is a function*[2] *from* $\mathbb{T}$ *to* $\mathcal{D}$*. The set* $\mathcal{D}$ *is also called the* value domain*. The quantity* $\mathsf{horizon} = \max(\mathbb{T})$ *is known as the time horizon of the signal.*

Signal Temporal Logic (STL). STL is a real-time logic, typically interpreted over a dense-time domain for signals that take values in a continuous metric space (such as $\mathbf{R}^m$). The basic primitive in STL is a *signal predicate* μ that is a formula of the form $f(x(t)) > 0$, where $x(t)$ is the value of the signal x at time t, and f is a function from the signal domain $\mathcal{D}$ to $\mathbf{R}$. STL formulas are then defined recursively using Boolean combinations of subformulas, or by applying an interval-restricted temporal operator to a subformula. The syntax of STL is formally defined as follows:

$$\varphi ::= \mu \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{G}_I\varphi \mid \mathbf{F}_I\varphi \mid \varphi\mathbf{U}_I\varphi \mid \varphi\mathbf{R}_I\varphi \tag{2.1}$$

Here, I is an interval over $\mathbf{R}^{\geq 0}$. The precise Boolean semantics of STL can be defined in recursive fashion (we omit the formal semantics for brevity). The semantics of Boolean combinations of subformulas define the obvious meaning. A temporal subformula, for example, $\varphi\mathbf{U}_{[a,b]}\psi$ holds at time t if there exists a time t' in $[t+a, t+b]$ where ψ is satisfied, and for all times t'' in $[t, t')$, φ must be satisfied. For some interval I, the formula $\mathbf{F}_I\varphi$ is an abbreviation for $true\mathbf{U}_I\varphi$, and $\mathbf{G}_I\varphi$ is equivalent to $\neg\mathbf{F}_I\neg\varphi$. Next, we introduce the notion of quantitative semantics for STL:

The quantitative semantics for STL defines the notion of a degree to which a given signal satisfies an STL formula φ. This is technically done by defining a function ρ that maps the signal and φ to a real value at each time t. This is defined recursively on the formula structure of STL as follows:

Definition 2.2 (Robust Satisfaction Value or Robustness Value)

$$\begin{aligned}
\rho(f(x) > c, x, t) &= f(x(t)) - c \\
\rho(\neg\varphi, x, t) &= -\rho(\varphi, x, t) \\
\rho(\varphi_1 \wedge \varphi_2, x, t) &= \min(\rho(\varphi_1, x, t), \rho(\varphi_2, x, t)) \\
\rho(\mathbf{G}_I\varphi, x, t) &= \inf_{t' \in t \oplus I} \rho(\varphi, x, t') \\
\rho(\mathbf{F}_I\varphi, x, t) &= \sup_{t' \in t \oplus I} \rho(\varphi, x, t') \\
\rho(\varphi_1\mathbf{U}_I\varphi_2, x, t) &= \sup_{t' \in t \oplus I} \left(\min \left(\rho(\varphi_2, x, t'), \inf_{t'' \in [t, t')} \rho(\varphi_1, x, t'') \right) \right) \\
\rho(\varphi_1\mathbf{R}_I\varphi_2, x, t'') &= \inf_{t' \in t \oplus I} \left(\max \left(\rho(\varphi_2, x, t'), \sup_{t'' \in [t, t']} \rho(\varphi_1, x, t'') \right) \right)
\end{aligned} \tag{2.2}$$

[2] When signals are evaluated w.r.t. Signal Temporal Logic formulas, we assume that the signal is defined at each time point in the interval $[0, t_N]$. We can do this using piecewise constant interpolation, i.e. $\forall i \in [0, N-1] : (t_i \leq t < t_{i+1}) \implies x(t) = x(t_i)$.

The robustness of a signal x w.r.t. a formula φ is then defined as $\rho(\varphi, x, 0)$ by convention. Note that if the robustness value is positive, the signal satisfies the STL formula, and if it is negative, it violates the STL formula. The convention is to treat the robustness value of 0 as the signal satisfying the formula.

We remark that in this paper, we focus on *signal patterns* expressed using STL. A signal pattern is essentially a bounded horizon STL formula, i.e. the scope of any temporal operator is upper bounded by some (small) finite time constant. A signal pattern is evaluated in the future of a given time t, and the robustness of signal pattern φ at time t is simply $\rho(\varphi, x, t)$. Examples of signal patterns include: $\mathbf{F}_{[0,3]}(x < 0)$, $\mathbf{F}_{[0,2]}\mathbf{G}_{[0,3]}(x > 0)$, $\mathbf{F}_{[0,1]}(x > 0 \wedge \mathbf{F}_{[0,1]}(x < -1 \wedge \mathbf{F}_{[0,1]}(x > 0)))$.

3 Background on Signal Forecasting

In this section, we give basic background on stochastic processes, and some key results that help us derive some guarantees on monitoring STL formulas in a predictive fashion in Sect. 4. Most of the definitions in this section have been adapted from the following reference: [8].

Definition 3.1 (Probability Space, Random Variables). *A* probability space *is a triple $(\Omega, \mathcal{F}, \mathcal{P})$, where Ω is a finite or infinite set describing possible outcomes, $\mathcal{F}$ is the σ-algebra over Ω (i.e. a collection of subsets of Ω including the empty set, that is closed under complement, countable unions and intersections), and $\mathcal{P}$ is a probability measure. Given a measurable state-space E, a random variable x is measurable function $x : \Omega \to E$.*

Definition 3.2 (Stochastic Process, Realizations, and Purely Random Process). *A stochastic process x is a finite or infinite collection of random variables ordered in (discrete or continuous) time. We denote the random variable at time t by $x(t)$ if time is continuous, and by x_t if time is discrete. A realization of a stochastic process is a signal that assigns concrete values from the signal range to each of the random variables $x(t)$. A discrete-time process consisting of a sequence of random variables z_t that are mutually independent and identically distributed is called a purely random process.*

Example 1 (Random Walk). Suppose z_t is a discrete, purely random process with mean μ and variance σ_z^2. A process x_t is said to be a random walk if $x_t = x_{t-1} + z_t$

Definition 3.3 (Stationary Processes). *A stochastic process x is called strictly stationary if the joint distribution of $x(t_1), \ldots, x(t_n)$ is the same as the joint distribution of $x(t_1+h), \ldots, x(t_n+h)$, for all $t_1, \ldots, t_n, h$. A stochastic process is called weakly stationary if its expected value is constant, and its covariance function only depends on the lag, formally,*

$$E[x(t)] = \mu \qquad Cov(x(t), x(t+h)) = \gamma(h) \tag{3.1}$$

Definition 3.4 (Autocovariance Function and Autocorrelation Function). *Let x_t be a stationary time series. The autocovariance function (ACVF) denoted $\gamma(h)$ is defined in Eq. (3.2) and autocorrelation function (ACF) $\rho(h)$ is defined as $\frac{\gamma(h)}{\gamma(0)}$.*

$$\gamma(h) = E[x(t) - \mu][(x(t+h) - \mu] = Cov[x(t), x(t+h)] \tag{3.2}$$

3.1 Linear Process and ARMA (ARIMA) Process

We first define the notion of a linear process.

Definition 3.5 (Linear Process). *A stochastic process x is called a linear process if it can be represented as: $x_t = \sum_{j=-\infty}^{\infty} \psi_j z_{t-j}$, where for all t, $\{z_t\} \sim \mathcal{N}(0, \sigma_z^2)$ and ψ_j is a constant series with $\sum_{j=-\infty}^{\infty} |\psi_j| < \infty$.*

Linear processes include all of the autoregressive (AR) processes, moving-average processes (MA), AR with MA (ARMA) processes and AR with integrated moving-average (ARIMA) models. Linear process models provide basic properties for studying ARMA, ARIMA, SARIMA and any other linear models. In what follows, it is convenient to define a new operator called the backward operator B, essentially, $Bx(t) = x(t+1)$, $B^h x(t) = x(t+h)$, etc. Using this operator, we can define an AR process with moving average.

Definition 3.6 (ARMA). *An ARMA process represents a combination of an autoregressive process (a process that can be represented as $\phi(B)x_t$), and a moving average process (a process that can be represented as $\theta(B)z_t$). Here, $\phi(B)$ and $\theta(B)$ are polynomials in the operator B, i.e. $\phi(B) = 1 - \sum_{i=1}^{p} \phi_i B^i$, and $\theta(B) = 1 + \sum_{j=1}^{q} \theta_j B^j$. An ARMA process thus has the following form:*

$$\phi(B)x_t = \theta(B)z_t \tag{3.3}$$

ARMA models are one of the most popular models used for forecasting values of a time-series. ARMA models are used for time-series data that can be viewed as a realization of a stationary stochastic process. However, ARMA models may not be adequate when the underlying process is not stationary, i.e. has trends. In such a case, an ARIMA (AR with integrated moving average) process model can be used.

Definition 3.7 (ARIMA). *An ARIMA model can be described by Eq. (3.4).*

$$\phi(B)(1-B)^d x_t = \theta(B)z_t \tag{3.4}$$

Here, when $d = 1$, an ARIMA model is suitable to model linear trends in the data, and for higher values of d, the model can be used to handle higher order trends (quadratic, cubic, etc.).

3.2 Forecasting Procedure

Given a signal-prefix up to time t_n, a forecasting procedure predict h future values of the signal, i.e. $x(t_{n+1}), \ldots, x(t_{n+h})$. Here, h is called the *prediction horizon*. Forecasting signal values involves several steps. The first step is to assume that the signal is the realization of a particular stochastic process, and then estimate the parameters of the stochastic process model from the signal values. This usually involves estimating the autocovariance and autocorrelation of the signal, and then using these to do *model fitting*. Model fitting attempts to identify the parameters of the chosen model (say ARMA), by solving certain optimization problems. A popular technique to do model fitting is based on Yule Walker equations [8].

After fitting the model, we have to forecast a time series. Since either ARMA or ARIMA model are all linear process, the *best linear predictor* is the optimal predictor for forecasting future values for the signal [8]. We now define the best linear predictor, and explain how it is computed.

Definition 3.8 (Best linear predictor). *The best linear predictor based on observation* $\{x_1, x_2, \ldots, x_n\}$ *of an ARMA or ARIMA process* $\{x_t\}$ *is given by Eq.* (3.5)*. Let* $\overline{x_t}$ *denote the predicted value of* x_t*.*

$$\begin{array}{l} \overline{x_{n+h}} = a_0^h + \sum_{i=1}^{n} a_i^h x_{n+1-i} \\ \textit{where, } \arg\min_{a_0^h, a_1^h, \ldots, a_n^h} \quad E[\overline{x_{n+h}} - x_{n+h}]^2. \end{array} \tag{3.5}$$

The optimized a_i is determined by two variables: since $\{x_t\}$ is a stationary process, denoting $\gamma_x(h) = Cov(x_{t+h}, x_t)$.

$$\Gamma_n = [Cov(x_{n-i+1}, x_{n-j+1})]_{i,j=1}^{n} = [\gamma_x(|i-j|)]_{i,j=1}^{n} \tag{3.6}$$

$$\gamma_n(h) = [\gamma_x(h+i-1)]_{i=1}^{n} \tag{3.7}$$

With these two variables we can define a_i as $(a_1, a_2, \ldots, a_n)^\top = \Gamma_n^{-1}\gamma_n(h)$ and $a_0 = \mu_x(1 - \sum_{i=1}^{n} a_i)$.

Now we see the prediction value for a single time step, in Sect. 4.1 we introduce how to derive a joint distribution for multiple time steps.

4 Clairvoyant Monitoring Procedure

To perform clairvoyant monitoring, we essentially need to forecast signal values using an appropriate stochastic process model, and more importantly compute the probability that a signal pattern is satisfied by the predicted signal values. To do the latter, we need to compute the joint distribution of the predicted values for the given stochastic process.

4.1 Deriving the Joint Distribution of Predictions

We first consider this computation for ARMA processes. By Definition 3.6 and Eq. (3.5), we can see the signal prediction values $\overline{x_{n+h}}$ produced by the best linear predictor are linear combinations of x_t and $\mathbf{a}$. We show that we can construct the joint distribution of multiple prediction values $\{\overline{x_{n+1}}, \ldots, \overline{x_{n+h}}\}$ by a sequence of linear transformations. First, we recall a standard result on linear combinations of normally distributed variables in Lemma 1.

Lemma 1. *If a random variable $X \sim \mathcal{N}(\mu_x, \Sigma_x)$, and $Y = CX + D$ is some linear transformation of X using matrices C and D, then $Y \sim \mathcal{N}(C\mu_x + D, C\Sigma_x C^\top)$.*

Theorem 1. *The joint distribution of h predicted values using the best linear predictor for an ARMA process x has a multivariate normal distribution $\mathcal{N}(\mathbf{a}, \Sigma)$, where $\mathbf{a}$ is vector $[a_0^1, \ldots, a_0^h]^\top$, Σ is given by the following equation:*

$$\Sigma = A\,\Phi^{+}\,\Theta\;\Sigma_z\;\Theta^{\top}\,(\Phi^{+})^{\top}\,A^{\top}. \tag{4.1}$$

Here, A, Φ and Θ are matrices of coefficients used in the ARMA model Definition 3.6 and the best linear predictor (3.5). (Precise definitions of each follow in the proof).

Proof. After using a standard technique to fit an ARMA model with order p and q [8], we obtain the set of equations in (4.2). This is simply the repeated application of Definition 3.6.

$$\begin{aligned}
x_{p+1} + \phi_1 x_p + \phi_2 x_{p-1} + \cdots + \phi_{1p} x_1 &= z_{p+1} + \theta_1 z_p + \theta_2 z_{p-1} + \cdots + \theta_q z_{1+p-q} \\
x_{p+2} + \phi_1 x_{p+1} + \phi_2 x_p + \cdots + \phi_p x_2 &= z_{p+2} + \theta_1 z_{p+1} + \theta_2 z_p + \cdots + \theta_q z_{2+p-q} \\
&\vdots \\
x_n + \phi_1 x_{n-1} + \phi_2 x_{n-2} + \cdots + \phi_p x_{n-p} &= z_n + \theta_1 z_{n-1} + \theta_2 z_{n-2} + \cdots + \theta_q z_{n-q}
\end{aligned} \tag{4.2}$$

We can write (4.2) as a matrix with appropriate zero padding to get:

$$\Phi \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \Theta \begin{pmatrix} z_{1+p-q} \\ z_{2+p-q} \\ \vdots \\ z_n \end{pmatrix} \tag{4.3}$$

Here, Θ is $(n-p) \times (n-p+q)$ matrix, and Φ is a $(n-p) \times n$ matrix. For a matrix M, let M^+ denote its Moore-Penrose inverse or its pseduo-inverse. We can multiply both sides of Eq. (4.3) Φ^+ to get Eq. (4.4).

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \Phi^{+}\Theta \begin{pmatrix} z_{1+p-q} \\ z_{2+p-q} \\ \vdots \\ z_n \end{pmatrix} \tag{4.4}$$

Recall the definition of best linear predictor in Definition 3.5:

$$\overline{x}_{n+h} = a_0^h + \sum_{i=1}^{n} a_i^h x_{n-i+1} \tag{4.5}$$

Writing this equation for each of the prediction steps from $n+1$ to $n+h$, we get Eq. (4.6).

$$\begin{cases} \overline{x}_{n+1} = a_n^1 x_1 + a_{n-1}^1 x_2 + \cdots + a_1^1 x_n + a_0^1 \\ \overline{x}_{n+2} = a_n^2 x_1 + a_{n-1}^2 x_2 + \cdots + a_1^2 x_n + a_0^2 \\ \quad \vdots \\ \overline{x}_{n+h} = a_n^h x_1 + a_{n-1}^h x_2 + \cdots + a_1^h x_n + a_0^h \end{cases} \tag{4.6}$$

This can be further written compactly as follows:

$$\begin{pmatrix} \overline{x}_{n+1} \\ \overline{x}_{n+2} \\ \vdots \\ \overline{x}_{n+h} \end{pmatrix} = A \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} a_0^1 \\ a_0^2 \\ \vdots \\ a_0^h \end{pmatrix}, \tag{4.7}$$

where A denotes the following coefficient matrix.

$$A = \begin{pmatrix} a_n^1 & a_{n-1}^1 & \ldots & a_1^1 \\ a_n^2 & a_{n-1}^2 & \ldots & a_1^2 \\ \vdots & \vdots & \ddots & \vdots \\ a_n^h & a_{n-1}^h & \ldots & a_1^h \end{pmatrix}. \tag{4.8}$$

Finally, substituting vector $[x_1, x_2, \ldots, x_n]^\top$ from Eq. (4.4), we achieved in rewriting the vector of predicted values into a linear transformation of white noise:

$$\begin{pmatrix} \overline{x}_{n+1} \\ \overline{x}_{n+2} \\ \vdots \\ \overline{x}_{n+h} \end{pmatrix} = A\Phi^+\Theta \begin{pmatrix} z_{1+p-q} \\ z_{2+p-q} \\ \vdots \\ z_n \end{pmatrix} + \begin{pmatrix} a_0^1 \\ a_0^2 \\ \vdots \\ a_0^h \end{pmatrix} \tag{4.9}$$

As white noise is normally distributed, from Lemma 1 we have the joint probability distribution of predictions of h steps for an ARMA process. ■

Theorem 2. *The normalized prediction value in an ARIMA process x_h has a multivariate normal distribution $\mathcal{N}(0, \Sigma)$, where Σ is given by the following equation:*

$$\Sigma = T_2 T_1 \Sigma_z T_1^\top T_2^\top. \tag{4.10}$$

Here, T_2, T_1 are matrices representing terms appearing in the best linear predictor expression (3.5) across h predictions and the ARIMA model Definition (3.4).

Proof. (Sketch) We omit the proof due to lack of space, but it follows a very similar recipe as the proof for the ARMA model. See [21] for details.

4.2 Enumerating Distinct Conjunctions of Signal Predicates

This is essentially a combinatorial problem. First, we assume that the formula is in Negation Normal Form (NNF), i.e. all negations are pushed to the signal predicates. This can always be done, cf. [14]. Further, note that we can replace negated atomic predicates by new atomic predicates, e.g. $\neg(x > 0) \equiv (x \leq 0)$. The basic idea of the algorithm is to expand the evaluation of the satisfaction probability of the STL formula (over the prediction horizon) into a disjunctive formula, where each disjunct is a conjunction of atomic predicates. This is an expensive step because of the complexity of a CNF to DNF conversion, but for small prediction horizons, this does not become prohibitive. The exact procedure to do this is through Algorithm 1, which essentially computes an expanded DNF representation for an STL formula. The above algorithm is invoked with the value $i = n + 1$. Each value of i is a time instant for predictions, so i ranges over $[n + 1, n + h]$. It essentially recursively travels the STL formula building the desired expression. We omit the case for the release operator for brevity, but the expansion follows the definition of the release operator. The following lemma can be easily proved using properties of Boolean operators $\vee$ and $\wedge$.

Algorithm 1: Expand$_h(\varphi,i)$

```
switch φ do
    case f(x̄(t_i)) > c
        return f(x̄(t_i)) > c
    case φ_1 ∧ φ_2
        A ←Expand_h(φ,i)
        B ←Expand_h(φ,i)
        Res ←{ };
        foreach C ∈ A do
            foreach D ∈ B do
                Res ←Res ∪ {C ∧ D}
    case φ_1 U_[a,b] φ_2
        Res ←{}
        foreach j ∈ [i, h − horizon(φ_2)] do
            Res_j ←{}
            A_j ←Expand_h(φ_1,j)
            foreach k ∈ [i, j] do
                B_k ←Expand_h(φ_2, k)
            Res_j ←Expand_h(A_j ∧ ⋀_k B_k, j)
        Res ←Expand_h(⋀_j Res_j, i)
```

Lemma 2. *The result of calling Expand$_h(\varphi, n{+}1)$ on an STL formula results in a disjunction over terms, where each term is a conjunction of atomic predicates at some times in* $[n + 1, n + h]$.

4.3 Calculating Probabilistic Guarantees for Monitoring

Now we have the joint distribution (across multiple time steps) for predicted signal values using Theorem 1 (for ARMA) and Theorem 2 (for ARIMA), and the disjunction over conjunctions of signal predicates corresponding to the STL-based signal pattern. The next step is to accumulate probabilities of these conjunctions of signal predicates. Direct addition will result in parts of joint distribution be integrated more than once. We use the inclusion-exclusion principle for computing probability of unions shown in (4.11) to solve the problem of calculating $P(\cup_{i=1}^{n} A_i)$.

$$\sum_{i=1}^{n} P(A_i) - \sum_{i<j} P(A_i \cap A_j) + \sum_{i<j<k} P(A_i \cap A_j \cap A_k) - \cdots + (-1)^{n-1} P(\cap_{i=1}^{n} A_i) \tag{4.11}$$

Probabilities of each conjunctions of atomic predicates in (4.11) can easily be done by marginalizing all other variables. Formally, let K indicate the set of times over which we want to compute the joint PDF. Then, we can compute the marginal probability using Eq. (4.12).

$$P(\bigwedge_{k \in K} A_k) = \int \dots \int P(A_1; \dots; A_h) dA_{j_1} \dots dA_{j_\ell} \tag{4.12}$$

Here, $\{j_1, \dots, j_\ell\} = \{1, \dots, h\} \setminus K$.

4.4 Complexity of the Algorithm

The complexity of our clairvoyant monitoring procedure depends on the exact form of STL formula involved. In this section, we list the upper bound of complexity for some formula forms. Assume that querying the marginal probability from the joint distribution is of complexity $O(1)$.

Consider the signal pattern $\mathbf{F}_{I_1}\mathbf{G}_{I_2}$. For this pattern, the complexity of computing the probability bounds is $O(2^{\lceil \frac{|I_1|}{\Delta} \rceil})$, as there are $|I_1|$ disjunctive marginal probability terms, and applying Eq. (4.11) will cost $O(2^n)$, where n equals to $\lceil \frac{|I_1|}{\Delta} \rceil$. Similarly, for a formula of the form $\mathbf{G}_{I1}\mathbf{F}_{I2}$, the complexity will be $O(2^{\lceil \frac{|I_2|}{\Delta} \rceil \lceil \frac{|I_1|}{\Delta} \rceil})$. For a general signal pattern, the worst-case complexity will depend on the number of conjunctions that will be enumerated.

5 Experimental Evaluation

In this section we experimentally demonstrate the power of predictive monitoring on interesting examples from the cyber-physical systems domain.

Table 1. The probability of predicted traces satisfying the specified STL pattern and runtime for computing the probabilities for the case studies on blood glucose prediction in insulin delivery and velocity prediction for a UAV.

Case Study	ψ	φ_1	φ_2	φ_3	φ_4
I	$P(BG(t) \models \psi)$ $(\sigma_1 = 50, \sigma_2 = \infty)$	1	1	1	1
	Time (s)	0.1068	0.1107	0.0944	2.9037
	$P(BG(t) \models \psi)$ $(\sigma_1 = 50, \sigma_2 = 150)$	0.3836	0.3838	0.3846	0.8462
	Time (s)	0.1067	0.3551	0.0705	26.9337
II	$P(v(t) \models \psi)(\sigma_1 = -0.5, \sigma_2 = 0.5)$	0.1452	0.3446	0.4160	0.2723
	Time (s)	0.0615	0.0792	0.0634	2.3367

Case Study I: Automated Insulin Delivery. Monitoring blood glucose levels is a crucial task for diabetes patients. In certain kinds of severe diabetes (e.g., type I diabetes), patients use automated insulin delivery systems (such as infusion pumps) to give a basal dose of insulin, and to optionally provide a *bolus* if the patient thinks that they are exceeding their usual intake of food (e.g. rich in carbohydrates). The tricky aspect of such devices is that while the response of the blood glucose to insulin is very slow, the response to carbohydrates is relatively fast. Thus, a patient upon seeing a high blood glucose level may exceed their required insulin dose. This can lead to a life-threatening condition called hypoglycemia. Thus, it is crucially important to monitor the blood glucose level in a *predictive fashion*. Similarly, if the blood glucose remains too high for a prolonged period of time (also known as prolonged hyperglycemia), then the patient can suffer long term consequences that can also eventually lead to death.

We have developed a simple linear Simulink® model representing the blood-glucose dynamics in a patient. For this experiment, we obtained the blood glucose (BG) signal by simulating the model with a fixed eating pattern by the patient. We simulated the patient behavior for one week, where BG was monitored at 15 min intervals. We fit an ARIMA process with order $p = 5, d = 2, q = 1$ to the BG signal, and used that for prediction. We checked various requirements (Eqs. (5.1)-(5.4)) on the blood glucose signal. For brevity, we write the formulas in a way that 1 time unit in the formula refers to 15 min of time.

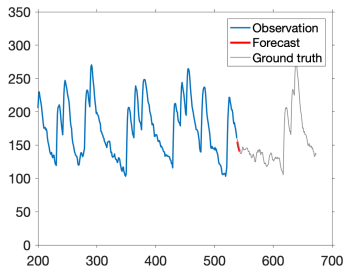


Fig. 2. BG signal with 5 prediction steps (Color figure online)

$$\varphi_1 \equiv \mathbf{G}(\mathbf{F}_{[0,1]}\mathbf{G}_{[0,2]}(\sigma_1 < x(t) < \sigma_2)) \tag{5.1}$$

$$\varphi_2 \equiv \mathbf{G}(x(t) \leq \sigma_1 \implies \mathbf{F}_{[0,1]}\mathbf{G}_{[0,2]}(\sigma_1 < x(t) < \sigma_2)) \tag{5.2}$$

$$\varphi_3 \equiv \mathbf{G}(x(t) \leq \sigma_1 \implies \mathbf{F}_{[0,2]}(\sigma_1 < x(t) < \sigma_2)) \tag{5.3}$$

$$\varphi_4 \equiv \mathbf{G}(\mathbf{F}_{[0,1]}(\sigma_1 < x(t) < \sigma_2)) \tag{5.4}$$

In formulas (5.1)–(5.4), we assume that x is BG. Formula (5.1) is an artificial requirement that says that any time, within the next 15 min the BG signal should remain within the given bounds for 30 min. Formula (5.2) says that if the BG signal is ever lower than the safe threshold, it should return to the safe threshold within 15 min, and stay in the threshold for 30 min. Formula (5.3) is a weaker requirement demanding the BG signal to simply return to the safe region. Finally, formula (5.4) says that eventually always within 15 min the BG signal should return to the safe region. We picked these formulas more to highlight that our algorithm works with different STL formulas with different temporal operator alternations. Some of these formulas are similar to the ones found in [9]. We picked a prediction horizon of 5 time steps (i.e. 75 min). In Fig. 2, we show the predicted blood glucose traces. We conducted two experiments, one where we picked σ_2 to be ∞ and the other where σ_2 was 150 mg/dL. σ_1 was fixed to 50 mg/dL. The results are shown in Table 1.

The first row of Table 1 shows that the patient can never become hypoglycemic, with very high probability[3]. From the third row, we can see that the controller that we implemented for automated insulin delivery does not do a good job with the hyperglycemia requirements (except for the last formula). From Fig. 2, we can see that for a small prediction horizon, the decreasing value of the BG signal (shown in red) gives enough confidence that in the next 15 min, the patient will not be hyperglycemic.

Case Study II: UAV Vertical Velocity. Now we look into the case of Unmanned Aerial Vehicle (UAV). We apply our technique to monitor vertical velocity, which is a crucial component affecting how vehicle control system adjust its rudder angle.

Monitoring Vertical Velocity. Vertical velocity is hard to directly observe, we obtain it through the observed acceleration signal given by gyroscope. The vertical velocity is vital for UAVs, as if the vertical velocity exceeds some threshold it will cause the vehicle to be damaged. We observed the autocorrelation function for the velocity trajectories, and used that to set the parameters $p = 12, d = 4, q = 8$ for the ARIMA model then do a 5 step look-ahead prediction. Conducting 5 steps look-ahead prediction in our data is equivalent to predicting 1 s ahead. The transverse velocity of UAV can achieve over 130 m/s, which makes behaviors that may happen in future 5 steps meaningful. The prediction results are

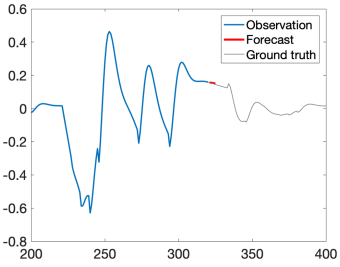


Fig. 3. UAV vertical velocity signal and predictions

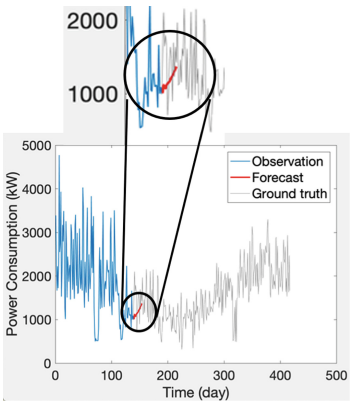
[3] Our implementation was done in Matlab, and Matlab has a certain precision when computing probabilities, and the number 1 is actually $1 - \delta$, where δ is smaller than the machine precision. This indicates that the probability is so high that it is practically 1.

shown in Fig. 3, and the probability guarantees of the predicted trace satisfy the STL requirement are shown in Table 1.

Case Study III: Monitoring Power Consumption Patterns. Household power consumption is an important factor that allows utility companies to estimate the overall power demand. In this case study, we study various STL formulas representing typical queries a utility company may find valuable. To perform this study, we use data from the UCI Machine Learning Repository [11]. The dataset is a multivariate time series dataset that describes the power consumption for a single household over four years. Each time step represents the average power consumption over a day. We fit an ARIMA model to this data with the model parameters $p = 5, d = 2, q = 1$. We are interested in computing the probabilistic guarantees on the STL formulas depicted in Eq. (5.5). In these formulas, $p(t)$ represents the power consumption at time t in KW, and c represents a threshold value.

$$\begin{aligned} \varphi_1 &= \mathbf{G}_{[0,n]}(p(t) > c) & \varphi_2 &= \mathbf{G}_{[0,n]}(p(t) < c) \\ \varphi_3 &= \mathbf{F}_{[0,n]}(p(t) > c) & \varphi_4 &= \mathbf{F}_{[0,2]}\mathbf{G}_{[0,5]}(p(t) > c) \end{aligned} \tag{5.5}$$

The formula φ_1 seeks to answer if there are n consecutive future days where the power consumption exceeds the threshold c. The formula φ_2 is true if the expected power consumption over the next n consecutive days is always below c. The formula φ_3 checks if it is always true if there is some day within the next two week period where the power consumption exceeds a threshold. Finally, φ_4 checks if there is some future time within two time steps where it is true that starting from that point, the power consumption always exceeds some threshold c. For each experiment, we assumed that the prediction horizon was 15. Our tool reads the first 139 samples to fit the ARIMA model, and then does its predictions on the next 15 time steps. We summarize the results in Fig. 4b.



(a) Power consumption signal and predictions

Formula	Parameters		Probability
	n	c	
φ_1	3	500	1
φ_1	3	1010	0.4139
φ_1	3	1120	0
φ_2	10	500	0
φ_2	10	1120	0.0531
φ_2	10	1200	0.3633
φ_2	10	1400	0.9976
φ_3	14	500	1
φ_3	14	1120	0.5468
φ_3	14	1500	0
φ_4	–	500	1
φ_4	–	1100	0.2968
φ_4	–	1500	0

(b) The probability guarantee of power consumption

Fig. 4. Power consumption case study

While the results generally agree with the ground truth data shown in Fig. 4a, in case of the formula φ_2, the experiments indicate that there is over 99% chance of never exceeding 1400 KW of power consumption. Clearly, the ground truth data shows otherwise. The problem here is that the ARIMA model has an effect of smoothing the data and focussing on the trends. If we were to fit the ARIMA model to data over a smaller granularity, it is possible that the model would track sharp local variations more faithfully. In general, if the time-step of the model is too coarse to capture all meaningful trends, the predictive monitoring algorithm can give misleading answers. For this case study, the runtime for fitting the ARIMA model and computing the probabilities was less than 3 s on a standard laptop machine with a 2.6 GHz processor.

6 Related Work and Conclusion

Related Work. Monitoring techniques for specifications in real-time temporal logics such as STL, Timed Propositional Temporal Logic (TPTL) and Metric Temporal Logic (MTL) have received considerable attention recently. See [6] for a recent survey. In [27], the authors define predictive semantics for LTL: these are similar to the three-valued semantics for LTL on incomplete traces and use a system model and model checking over trace suffixes to compute one of the three values (true, false or unknown). However, this approach does not compute violation probabilities. In [2,3], the authors define an interesting predictive monitoring approach. The key idea is to construct an Hidden Markov Model abstraction of a system and use that to predict satisfaction of a given temporal property. This is an alternate way of modeling probabilities in the system, and represents a different take on the same problem. In future work, we will consider extending our signal predictors to those based on Markovian assumptions on the underlying process. We note that these papers focus on LTL with Boolean predicates rather than STL (which has signal predicates).

Also of relevance is the work in the R2U2 monitoring framework [13,20,23,24]. The R2U2 framework uses efficient temporal observers for LTL coupled with dynamic Bayesian networks to probabilistically estimate the state and health of system components. The work proposed in [7,17,26] also addresses a similar problem. In many ways, these are also monitoring problems that are predictive in nature, but the prediction here is regarding hidden system states, rather than predictions in time. Seminal work on *monitorability* of various kinds of stochastic dynamical models (typically with Markovian assumptions) refers to this problem as internal monitoring [25], and we distinguish our work in its clairvoyant abilities.

7 Conclusion

In this paper, we present monitoring framework for signal patterns expressed using Signal Temporal Logic (STL). The main contribution of this paper is an algorithm for clairvoyant monitoring that computes the probability of a signal pattern being satisfied/violated by a set of future/unseen signal values. To

achieve clairvoyance, the algorithm utilizes using statistical time-series modeling techniques, assuming that observed data is the realization of a linear stochastic process (such as ARMA or ARIMA). The key technical result is a technique to compute the joint probability distribution of the predicted values and use it to compute the satisfaction probability of the given temporal pattern. In future work, we will consider techniques that help calibrate the prediction result, give expected value for robustness and also explore techniques based on reachability, such as those in [12], to compute forward reachable sets to estimate satisfaction probabilities of STL formulas.

Acknowledgments. We thank the anonymous reviewers for their careful reading of the paper and the constructive feedback. We gratefully acknowledge the support by the National Science Foundation under grant no. CCF/SHF-1910088, and a grant from Toyota Motors R&D North America. We thank Yue Wu and Weixin Cai for fruitful discussions that helped shape the proofs for Theorem 1.

References

1. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: a tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 254–257. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_21
2. Babaee, R., Gurfinkel, A., Fischmeister, S.: Predictive run-time verification of discrete-time reachability properties in black-box systems using trace-level abstraction and statistical learning. In: Colombo, C., Leucker, M. (eds.) RV 2018. LNCS, vol. 11237, pp. 187–204. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03769-7_11
3. Babaee, R., Gurfinkel, A., Fischmeister, S.: $\mathcal{P}revent$: a predictive run-time verification framework using statistical learning. In: Johnsen, E.B., Schaefer, I. (eds.) SEFM 2018. LNCS, vol. 10886, pp. 205–220. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92970-5_13
4. Bartocci, E., Bortolussi, L., Sanguinetti, G.: Learning temporal logical properties discriminating ECG models of cardiac arrhytmias. arXiv preprint arXiv:1312.7523 (2013)
5. Bartocci, E., Bortolussi, L., Sanguinetti, G.: Data-driven statistical learning of temporal logic properties. In: Legay, A., Bozga, M. (eds.) FORMATS 2014. LNCS, vol. 8711, pp. 23–37. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10512-3_3
6. Bartocci, E., et al.: Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification. LNCS, vol. 10457, pp. 135–175. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75632-5_5
7. Bartocci, E., et al.: Adaptive runtime verification. In: Qadeer, S., Tasiran, S. (eds.) RV 2012. LNCS, vol. 7687, pp. 168–182. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35632-2_18
8. Brockwell, P.J., Davis, R.A., Calder, M.V.: Introduction to Time Series and Forecasting, vol. 2. Springer, Heidelberg (2002). https://doi.org/10.1007/b97391

9. Cameron, F., Fainekos, G., Maahs, D.M., Sankaranarayanan, S.: Towards a verified artificial pancreas: challenges and solutions for runtime verification. In: Bartocci, E., Majumdar, R. (eds.) RV 2015. LNCS, vol. 9333, pp. 3–17. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23820-3_1
10. Deshmukh, J.V., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., Seshia, S.A.: Robust online monitoring of signal temporal logic. Formal Meth. Syst. Des. **51**(1), 5–30 (2017). https://doi.org/10.1007/s10703-017-0286-7
11. Dua, D., Graff, C.: UCI machine learning repository (2017). http://archive.ics.uci.edu/ml
12. Duggirala, P.S., Mitra, S., Viswanathan, M., Potok, M.: C2E2: a verification tool for stateflow models. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 68–82. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_5
13. Geist, J., Rozier, K.Y., Schumann, J.: Runtime observer pairs and Bayesian network reasoners on-board FPGAs: flight-certifiable system health management for embedded systems. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 215–230. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_18
14. Ho, H.-M., Ouaknine, J., Worrell, J.: Online monitoring of metric temporal logic. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 178–192. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_15
15. Hoxha, B., Abbas, H., Fainekos, G.: Benchmarks for temporal logic requirements for automotive systems. In: Frehse, G., Althoff, M. (eds.) ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems. EPiC Series in Computing, vol. 34, pp. 25–30. EasyChair (2015)
16. Jin, X., Deshmukh, J.V., Kapinski, J., Ueda, K., Butts, K.: Powertrain control verification benchmark. In: Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control, pp. 253–262. ACM (2014)
17. Kalajdzic, K., Bartocci, E., Smolka, S.A., Stoller, S.D., Grosu, R.: Runtime verification with particle filtering. In: Legay, A., Bensalem, S. (eds.) RV 2013. LNCS, vol. 8174, pp. 149–166. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40787-1_9
18. Kapinski, J., et al.: ST-Lib: a library for specifying and classifying model behaviors. In: SAE Technical Paper. SAE (2016)
19. Kong, Z., Jones, A., Medina Ayala, A., Aydin Gol, E., Belta, C.: Temporal logic inference for classification and prediction from data. In: Proceedings of HSCC, pp. 273–282 (2014)
20. Moosbrugger, P., Rozier, K.Y., Schumann, J.: R2U2: monitoring and diagnosis of security threats for unmanned aerial systems. Formal Methods Syst. Des. **51**(1), 31–61 (2017). https://doi.org/10.1007/s10703-017-0275-x
21. Qin, X., Deshmukh, J.V.: Joint probability distribution of prediction errors of ARIMA. CoRR abs/1811.04685 (2018), http://arxiv.org/abs/1811.04685
22. Roehm, H., Gmehlich, R., Heinz, T., Oehlerking, J., Woehrle, M.: Industrial examples of formal specifications for test case generation. In: Workshop on Applied veRification for Continuous and Hybrid Systems, ARCH. pp. 80–88 (2015)
23. Schumann, J., Moosbrugger, P., Rozier, K.Y.: R2U2: monitoring and diagnosis of security threats for unmanned aerial systems. In: Bartocci, E., Majumdar, R. (eds.) RV 2015. LNCS, vol. 9333, pp. 233–249. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23820-3_15

24. Schumann, J., Moosbrugger, P., Rozier, K.Y.: Runtime analysis with R2U2: a tool exhibition report. In: Falcone, Y., Sánchez, C. (eds.) RV 2016. LNCS, vol. 10012, pp. 504–509. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46982-9_35
25. Sistla, A.P., Žefran, M., Feng, Y.: Monitorability of stochastic dynamical systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 720–736. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_58
26. Stoller, S.D., et al.: Runtime verification with state estimation. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 193–207. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29860-8_15
27. Zhang, X., Leucker, M., Dong, W.: Runtime verification with predictive semantics. In: Goodloe, A.E., Person, S. (eds.) NFM 2012. LNCS, vol. 7226, pp. 418–432. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28891-3_37