53rd IEEE Conference on Decision and Control
December 15-17, 2014. Los Angeles, California, USA

Anomaly Detection in Cyber-Physical Systems: A Formal Methods Approach

Austin Jones, Zhaodan Kong, Calin Belta

***Abstract*— As the complexity of cyber-physical systems increases, so does the number of ways an adversary can disrupt them. This necessitates automated anomaly detection methods to detect possible threats. In this paper, we extend our recent results in the field of inference via formal methods to develop an unsupervised learning algorithm. Our procedure constructs from data a signal temporal logic (STL) formula that describes normal system behavior. Trajectories that do not satisfy the learned formula are flagged as anomalous. STL can be used to formulate properties such as "If the train brakes within 500 m of the platform at a speed of 50 km/hr, then it will stop in at least 30 s and at most 50 s." STL gives a more human-readable representation of behavior than classifiers represented as surfaces in high-dimensional feature spaces. STL formulae can also be used for early detection via online monitoring and for anomaly mitigation via formal synthesis. We demonstrate the power of our method with a physical model of a train's brake system. To our knowledge, this paper is the first instance of formal methods being applied to anomaly detection.**

I. Introduction

Cyber-physical systems (CPSs) integrate physical processes with computational resources via communication networks. In light of high-profile attacks, such as the Maroochy water breach [1], there has been a surge of interest in understanding how an adversary can disrupt a cyber-physical system and how such attacks can be identified and potentially mitigated [2], [3]. In all the cited works, the designer of the controllers and/or estimators is assumed to have perfect knowledge of the physical systems under consideration, which are assumed to be described by linear models. These assumptions are not consistent with the growing complexity of modern CPSs and the involvement of agents, such as humans, whose behavior is generally quite hard to predict.

In this paper, we apply formal methods to an anomaly detection framework to identify whether or not a given CPS is under attack. Anomaly detection is the problem of detecting patterns from data that do not conform to expected behavior [4]. In our case, we are looking for patterns in the output of a CPS that lead us to believe that the underlying dynamics of the system have changed due to attack. Tools from machine learning, such as Gaussian processes, have been adapted to anomaly detection [5]. In general, existing techniques infer a surface embedded in a high-dimensional feature space that separates normal and anomalous data. However, it is hard to interpret the meanings of the surfaces, especially in the context of prediction, knowledge base construction and on-line monitoring (i.e. determining on-line whether a behavior is anomalous).

Our approach to the problem circumvents the over-specificity of model-based CPS security methods and the low usability of existing anomaly detection techniques. We present a model-free unsupervised learning algorithm for inferring a signal temporal logic (STL) formula from system output data that can be used to classify data as normal or anomalous. STL can express system properties that include time bounds and bounds on physical system parameters, e.g. "If the boat remains in region A while maintaining its speed below 10 kph for 10 min., it is guaranteed to reach the port within 15 min." STL formulae are easy to formulate in natural language and have a rigorous mathematical definition, meaning they can be used for both human-in-the-loop and automated on-line monitoring.

Our procedure is an extension of our previous work [6] (which, in turn, was inspired by [7]), in which we developed a supervised learning algorithm for inferring formulae to distinguish between desirable (e.g. "the car successfully stops before hitting an obstruction") and undesirable (e.g. "The car strikes the obstruction") behavior. We defined a fragment of STL, called reactive STL (rSTL), whose formulae can also indicate possible causes for each set of behaviors (e.g. "If the speed of the car is greater than 15 m/s within 0.5s of brake application, the obstruction will be struck"). By using the concept of a robustness degree [8], [9], we showed how to perform a directed search over this set. In contrast, in this paper, we address the problem of finding such a formula when the system output data is not labeled. We use many of the same concepts and theoretical results from [6], but due to the different problem formulations, we use a fragment of rSTL that does not require a causal structure.

We use two case studies, a simple academic example and a more realistic model of an electronically controlled pneumatic train brake system (adapted from [10]), to demonstrate that our algorithm is able to correctly identify anomalies. Although the focus of our paper is on detecting anomalies, we use the case studies to also demonstrate how the inferred formula can be used for on-line monitoring. Further research will approach this integration in a more formal and rigorous manner.

II. Mathematical Preliminaries

A signal x is a map $x : \mathbb{R}^+ \to X \subseteq \mathbb{R}^n$. We denote the value of x at time t as $x(t)$ and the suffix of signal x from time t as $x[t]$. In this paper, given a system $\mathcal{S}$ (e.g. a

Austin Jones and Calin Belta are with the Division of Systems Engineering. Zhaodan Kong and Calin Belta are with the Department of Mechanical Engineering at Boston University, Boston, MA 02115. Email: {austinmj, zhaodan, cbelta}@bu.edu

978-1-4673-6090-6/14/$31.00 ©2014 IEEE

set of ODEs or a hybrid automaton), we call the set of its trajectories the *language* of $\mathcal{S}$, denoted $L(\mathcal{S})$.

Signal temporal logic (STL) [11] is a predicate temporal logic defined over signals. In this work, we focus on the fragment called inference STL (iSTL), which is a generalization of reactive STL (rSTL), a fragment we previously defined in [6]. iSTL differs from rSTL in that the syntax does not require every formula to contain Boolean implication ($\Rightarrow$). The *syntax* of iSTL is defined as

$$\phi ::= F_{[0,T)}(\phi_c) \tag{1a}$$

$$\phi_c ::= F_{[a,b)}\ell | \ G_{[a,b)}\ell | \phi_c \wedge \phi_c | \phi_c \vee \phi_c \tag{1b}$$

where T is the maximum duration of x, $[a,b)$ is a time interval, ℓ is a rectangular predicate of the form $(x_{(i)} \sim c), \sim \in \{<, \geq\}$, $c \in \mathbb{R}$, and $x_{(i)}$ is a one-dimensional element of the signal x. $\vee$ and $\wedge$ are conjunction and disjunction, respectively. $F_{[a,b)}$ and $G_{[a,b)}$ are the temporal operators "finally" ("eventually") and "globally" ("always"), respectively. The external operator $F_{[0,T]}$ means that we are searching for properties that can occur at any point in a signal.

The *semantics* of iSTL is defined recursively as

$$\begin{array}{rll} x[t] \models (x_{(i)} \sim c) & \text{iff} & x_{(i)}(t) \sim c \\ x[t] \models \phi_1 \wedge \phi_2 & \text{iff} & x[t] \models \phi_1 \text{ and } x[t] \models \phi_2 \\ x[t] \models \phi_1 \vee \phi_2 & \text{iff} & x[t] \models \phi_1 \text{ or } x[t] \models \phi_2 \\ x[t] \models F_{[a,b)}\phi & \text{iff} & \exists t' \in [t+a, t+b) \text{ s.t. } x[t'] \models \phi \\ x[t] \models G_{[a,b)}\phi & \text{iff} & x[t'] \models \phi \ \forall t' \in [t+a, t+b) \end{array} \tag{2}$$

A signal x satisfies an iSTL formula ϕ if $x[0] \models \phi$. The *language* of an STL formula ϕ, $L(\phi)$, is the set of all signals that satisfy ϕ.

The *robustness degree* of a signal x with respect to an iSTL formula ϕ at time t is given as $r(x, \phi, t)$, where r can be calculated recursively via the *quantitative semantics* [8], [9]

$$\begin{array}{rl} r(x, (x_{(i)} \geq c), t) & = x_{(i)}(t) - c \\ r(x, (x_{(i)} < c), t) & = c - x_{(i)}(t) \\ r(x, \phi_1 \wedge \phi_2, t) & = \min(r(x, \phi_1, t), r(x, \phi_2, t)) \\ r(x, \phi_1 \vee \phi_2, t) & = \max(r(x, \phi_1, t), r(x, \phi_2, t)) \\ r(x, \ F_{[a,b)}\phi, t) & = \max\limits_{t' \in [t+a, t+b)} r(x, \phi, t') \\ r(x, \ G_{[a,b)}\phi, t) & = \min\limits_{t' \in [t+a, t+b)} r(x, \phi, t') \end{array}$$

The robustness degree of the entire signal is denoted as $r(x, \phi) = r(x, \phi, 0)$. If $r(x, \phi)$ is large and positive (negative), then x satisfies (violates) ϕ and a large perturbation to x would be required in order for the resulting signal x' to violate (satisfy) ϕ. If $r(x, \phi) \approx 0$, then if even a small perturbation is applied to x, whether or not x' satisfies ϕ is unpredictable.

Inference parametric signal temporal logic (iPSTL) [12] is an extension of iSTL where the bound c and the endpoints of the time intervals $[a, b)$ are parameters instead of constants. We denote them as *scale* parameters $\pi = [\pi_1, ..., \pi_{n_\pi}]$ and *time* parameters $\tau = [\tau_1, ..., \tau_{n_\tau}]$, respectively. A full parameterization is given as $[\pi, \tau]$. The syntax and semantics of iPSTL are the same as those for iSTL. To avoid confusion, we will use ϕ to denote an iSTL formula and φ to refer to an iPSTL formula. A *valuation* θ is a mapping that assigns real values to the parameters appearing in an iPSTL formula. A valuation θ of an iPSTL formula φ induces an STL formula ϕ_θ. For example, if $\varphi = F_{[\tau_1, \tau_2)}(x_1 \geq \pi_1)$ and $\theta([\pi_1, \tau_1, \tau_2]) = [0, 0, 3]$, then $\phi_\theta = F_{[0,3)}(x_1 \geq 0)$.

In [6], we showed that the set of iSTL and the set of iPSTL formulae admit partial orders. Further, the set of all iPSTL formulae can be organized into a directed acyclic graph (DAG) where there exists a path from φ_1 to φ_2 if $\forall x, \theta$, $r(x, \phi_{1,\theta}) \leq r(x, \phi_{2,\theta})$. Therefor, if we use the robustness degree as a fitness measure, we can search for an iSTL formula that best fits a given set of data by iteratively using the DAG to perform a search over the set of iPSTL formula and using continuous optimization methods in order to find its optimal valuation.

III. Problem formulation

A. Cyber-Physical Systems Under Normal Operation

We denote a cyber-physical system under normal operation (i.e. operating as intended in the absence of an attack) as a system $\mathcal{S}_N$. A trajectory of $\mathcal{S}_N$ is a signal $x : \mathbb{R}^+ \to X$, where $X \subseteq \mathbb{R}_n$ is the (possibly) high-dimensional physical state space of the CPS. The operation of the system is observed via an output signal

$$y(t) = g(x(t), t, w), \tag{3}$$

where w is a noise process. This concept is illustrated in the following scenario which will serve as a running example throughout this paper.

Example 1 (Normal system)**.** Consider a train using an electronically-controlled pneumatic (ECP) braking system. The train has 3 cars, each of which has its own braking system. The state of the train system is defined by the velocity v of the train. Our model of the train system is modified from [10]. See Section V-B for more details.

In this model, the braking system is automated to regulate the velocity below unsafe speeds and above low speeds to ensure that the train reaches its destination. If $v(t)$ exceeds a threshold v_{max} (28.5 m/s), the velocity of the un-braked system increases (shown in green in Fig. 1). Each of the brakes responds to the threshold crossing after a random time delay by engaging. The brakes decrease the velocity of the train (shown in black in Fig. 1) until it passes a second threshold v_{min} (20 m/s). After random delays, the brakes disengage.

The system is observed via the output signal

$$y(t) = v(t) + w, \tag{4}$$

where w is a white noise process with variance 0.3. Some traces of the output signal are shown in Fig. 1. Note that there is quite a bit of variability in the signals due to noisy inputs and delays between brake engagement and disengagement, but they all maintain $v(t)$ below v_{max} and above v_{min} for most of the time. □

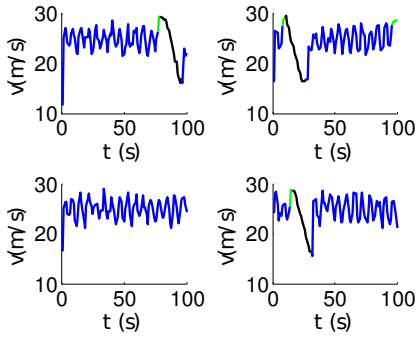


Fig. 1. Four output signals of the velocity of the train controlled by three ECP braking systems when all three brakes are functioning normally. The colors refer to modes of the hybrid automaton that describes the system (given in Section V-B): blue indicates the system oscillates between low and high velocities, green indicates the system is moving too fast, and black indicates the system is being braked.

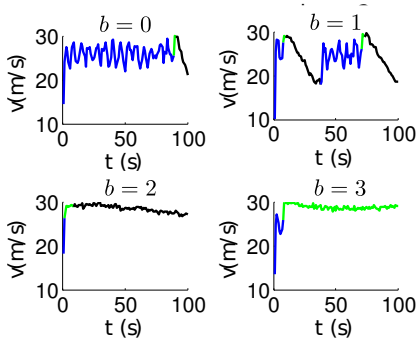


Fig. 2. Outputs of the train velocity system under different attack scenarios. An adversary has the ability to disable one, two, or three of the trains brakes in order to deregulate its velocity. The variable b is the number of brakes affected by attack.

B. Cyber-Physical Systems Under Attack

In the previous subsection, we described cyber-physical systems under normal operation. However, we are interested in the case in which an adversary can affect the sensors or actuators of the system in order to disrupt its normal operation. Given a system $\mathcal{S}_N$ under normal operation, we define a system with attack vulnerabilities as a system $\mathcal{S}_T$ such that $L(\mathcal{S}_N) \subset L(\mathcal{S}_T)$. That is, $\mathcal{S}_T$ behaves normally (produces the same outputs as $\mathcal{S}_N$) when no attacks take place and behaves qualitatively differently if an adversary disrupts it.

Example 1 (Attacked system)**.** An adversary has the possibility to disable each of the brakes of the train. A few sample outputs from the system $\mathcal{S}_T$ are shown in Fig. 2.

The behavior of the system depends on how much access the adversary has, i.e. the number of brakes b that can be disabled. All three attacked outputs behave qualitatively differently from each other, but they all clearly violate the desired invariant behavior demonstrated in Fig. 1. Although the difference between normal and attacked outputs is visually obvious, it is difficult to quantify the difference between the two sets of behaviors due to a large variability in each output set. Our procedure is able to separate these two sets automatically with no *a priori* knowledge of system dynamics or attack models. □

C. Problem Definition

We are interested in determining whether a cyber-physical system is under attack or not. As illustrated in Example 1, a CPS may exhibit a wide range of behaviors. Thus we must compare individual system executions to some global property that all normal executions of the system need to satisfy. The logic iSTL (defined in Section II) is well-suited for compactly and precisely describing CPS behavior. Logical operators describe how different components of the output signal interact. Temporal operators describe how the system changes over time. The bounds given by rectangular predicates and the time bounds on the temporal operators incorporate physical parameters into the description. Further, the set of iSTL formulae may be searched in an efficient and principled manner as demonstrated in Section IV.

Example 1 (iSTL properties)**.** From examining the outputs shown in Fig. 1, it is clear that each output satisfies the iSTL formula

$$F_{[0,100)}(F_{[0,10)}(y > 25) \wedge G_{[0,30)}(y < 30)). \tag{5}$$

In plain English, this means "At some point, the output exceeds 25 m/s within the next 10s while remaining below 30 m/s for the next 30 s." □

Given a model $\mathcal{S}_T$ of a cyber-physical system, it is in general difficult to determine analytically an iSTL formula that describes only the normal behaviors of the system. Further, in many applications, explicit models of the CPS are unavailable for analysis. Therefore, we propose to find such a formula directly from system output data.

In the ideal case, we would be able to construct a formula that can perfectly distinguish normal and attacked behaviors. However, since the CPS under consideration may involve process and observation noise, and only a finite number of traces are available, we focus our attention on the more realistic goal given by Problem 1.

Problem 1. A CPS with attack vulnerabilities $\mathcal{S}_T$ produces a set of trajectories $\{x_i\}_{i=1}^N$. Given the set $\{y_i\}_{i=1}^N$ with y_i being the observed output associated with x_i, find an iSTL formula ϕ such that the misclassification rate, given by

$$\frac{|\{y_i | y_i \models \phi, x_i \notin L(\mathcal{S}_N)\}| + |\{y_i | y_i \not\models \phi, x_i \in L(\mathcal{S}_N)\}|}{N}, \tag{6}$$

is minimized.

That is, we want to find a formula that has a high correct detection rate (correctly flags outputs from systems under attack as anomalous) and a low false alarm rate (rarely flags outputs from the system under normal operation as anomalous). In order to simplify the problem, we make the following two key assumptions. First, attacks happen infrequently, i.e., given an output y_i from a CPS $\mathcal{S}_T$, the *a priori* probability that it was produced by $\mathcal{S}_T$ under attack is low. Second, the outputs of a system under attack differs

qualitatively from the outputs of a system under normal operation. Otherwise, it is impossible to infer any classifier to separate the two sets of outputs. These assumptions are plausible for real-world scenarios and are commonly made in other anomaly detection problems [4].

IV. Solution

Since Problem 1 is an unsupervised learning problem, we use some notions from classical unsupervised learning to aid in our approach. In particular, we consider one-class support vector machines (SVMs). A one-class SVM is an optimization that, given a set of data, lifts the data to a higher-dimensional feature space and constructs a surface in this space that separates normal data from anomalous data [13]. We adapt the objective function used in one-class SVM and map Problem 1 to the following optimization.

$$\min_{\phi_\theta, \epsilon} d(\phi_\theta) + \frac{1}{\nu N} \sum_{i=1}^{N} \mu_i - \epsilon \tag{7}$$

such that

$$\mu_i = \begin{cases} 0 & r(y_i, \phi_\theta) > \epsilon/2 \\ \epsilon/2 - r(y_i, \phi_\theta) & \text{else} \end{cases} \quad \forall i, \tag{8}$$

where ϕ_θ is an iSTL formula, ϵ is the "gap" in signal space between outputs identified as normal and outputs defined as anomalous, ν is the upper bound of the *a priori* probability that the CPS is under attack, μ is a slack variable, which is positive if y_i does not satisfy ϕ_θ with minimum robustness $\epsilon/2$, and the function d is a "tightness" function that penalizes the size of $L(\phi_\theta)$.

Minimizing the sum of the μ_i minimizes the number of traces the learned formula ϕ_θ classifies as anomalous. This is consistent with a low prior attack probability. Maximizing the gap ϵ maximizes the separation between normal and anomalous outputs. Minimizing the function $d(\phi_\theta)$ prevents the learned formula from trivially describing all observed signals, i.e. finding a formula s.t. $y_i \models \phi_\theta$, $\forall x_i \in L(\mathcal{S}_T)$.

Solving (7) requires searching over the set of continuous variables (θ and ϵ) as well as over the discrete set of iPSTL formula structures (the structure φ of ϕ_θ). We showed in our previous work [6] how the set of reactive PSTL (rPSTL) formulae may be efficiently searched and developed an algorithm to solve the supervised learning problem by iterating between the discrete structure search and continuous variable optimization via simulated annealing. Algorithm 1 is an adaptation of this procedure to solve (7).

Algorithm 1 begins by organizing all of the iPSTL formulae of length 1, e.g., the set of all formulae $O_{[\tau_1,\tau_2)}(x_i \sim \pi)$ where $O \in \{F, G\}, x_i \in V$, and $\sim \in \{\leq, \geq\}$, into a DAG (DAGInitialization). The set of formulae in the graph is then organized into a ranked list (ListInitialization). The ranking of the formulae in the list is generated randomly during initialization, as no *a priori* knowledge of the fitness of the formulae exists.

After the graph and list are initialized, the iterative learning procedure begins. The parameter estimation loop (lines 9-13) iterates over formulae structures φ in the list $List$ from lowest rank to highest rank (Line 10). ParameterEstimation uses simulated annealing to solve (7) to find the optimal values of θ and ϵ for each formula structure φ.

Algorithm 1 Anomaly detection algorithm. L_{max} is the limit of the length of the mined formula. V is the set of variables that can appear in rectangular predicates. d is a tightness function. δ is an acceptable performance threshold.

1: **function** FindFormula($L_{max}, V, \{y_i\}_{i=1}^N, d, \delta$)
2: **for** $i = 1$ to L_{max} **do**
3: **if** $i = 1$ **then**
4: $\mathcal{G}_1 \leftarrow$ DAGInitialization(V);
5: $List \leftarrow$ ListInitialization($\mathcal{G}_1$);
6: **else**
7: $\mathcal{G}_i \leftarrow$ PruningAndGrowing($\mathcal{G}_{i-1}$);
8: $List \leftarrow$ Ranking($\mathcal{G}_i \setminus \mathcal{G}_{i-1}$,);
9: **while** $List \neq \emptyset$ **do**
10: $\varphi \leftarrow List$.pop();
11: $(\theta, cost, \epsilon) \leftarrow$ ParameterEstimation($\{y_i\}_{i=1}^N, \varphi, d$);
12: **if** $cost \leq \delta$ **then**
13: **return** (φ, θ).
14: **return** MinimumCostNode($\mathcal{G}_{L_{max}}$);

The ParameterEstimation procedure uses the heuristic tightness function d when calculating the objective function in (7). In this paper, we use the heuristic given by Algorithm 2. The subroutine Normalize normalizes the value of a parameter to $[0, 1]$. For each linear predicate in φ, Tightness penalizes the size of the parameter τ_1, as for monitoring purposes we prefer formulae that describe behaviors of the early parts of the system's outputs. If $\sim = <$, the size of π is penalized, as $L(x_i < \pi)$ grows with π. If $\sim = \geq$, then small values of π are penalized. The sum of the penalized quantities is normalized over the interval $[0, 1]$ so that the magnitude of the output remains invariant with the length of $||\varphi||$. This value is then multiplied by a constant λ that takes into account the total number of trajectories and the ranges of θ.

Algorithm 2 Tightness Function

1: **function** Tightness(θ,φ)
2: $k = 0$
3: **for all** $(\tau_1, \tau_2, \pi) \in \theta$ such that $O_{[\tau_1,\tau_2)} x_i \sim \pi)$ in φ **do**
4: tightness[k] = Normalize(τ_1); $k++$;
5: **if** $\sim$ is $<$ **then**
6: tightness[k] = Normalize(π); $k++$;
7: **else**
8: tightness[k] = 1-Normalize(π); $k++$;
9: **return** $\lambda \frac{\sum_k \text{tightness}[k]}{\text{length(tightness)}}$

If the optimal cost $cost$ from ParameterEstimation is small enough (less than δ), the current formula is returned. If no acceptable solution is found, the set of iPSTL formulae is searched (Lines 6-8). At the ith iteration, PruningandGrowing uses logical rules to grow the graph of searched formula to include iPSTL formulae of length $i+1$. This function also

prunes a constant proportion of formulae of length i whose optimal cost found by parameter estimation was too high. This prevents the algorithm from searching over formulae that we can assume have poor performance. After the graph is grown, the formulae of length $i+1$ are organized into a list ranked according to the performance of their parents in the DAG. When the formulae are iterated over in Lines 6-8, the formulae with lowest rank (those whose parents performed best) are considered first. This iterative process of parameter estimation and formula structure search continues until either a formula with low enough cost is found or the length L_{max} is reached.

V. Case Studies

A. Linear System

1) Model: We first test our implementation of Algorithm 1 on a system $\mathcal{S}_T$ whose dynamics evolve according to

$$\dot{x} = 0.03x + w \tag{9}$$

under normal operation or

$$\dot{x} = -0.03x + w \tag{10}$$

under attacked operation, where w is a white noise process with variance 0.025. For simplicity, $y(t) = x(t)$.

2) Formula inference: We generated 200 different trajectories of $\mathcal{S}_T$, shown in Fig. 3(a). We ran our inference algorithm on a training set of 100 of the trajectories, 4 of which represented an attack (red), and reserved the other 100 (with 7 attacks) for testing the output of Algorithm 1.

From the training set, our implementation of Algorithm 1 inferred the formula

$$F_{[0,3.0)}(G_{[0.5,2.0)}(y > 0.9634)). \tag{11}$$

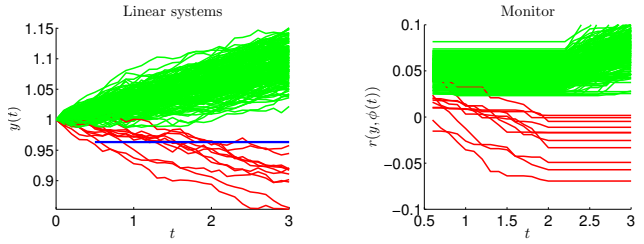
In plain English, this is "At every point within 0.5 and 2.0 s in the future, the output y exceeds 0.9634". We used 3 simulated annealing cycles with 15 samples per cycle. The computation time was 130 s on an 8 core PC with 2.1 GHz processors and 8 GB RAM.

The threshold 0.9634 from formula (11) is indicated with a blue line in Fig. 3(a). The formula (11) successfully separates the normal and attacked outputs, i.e. the misclassification rate for the training set was 0 and the training set had a misclassification rate of 0.01. The single missed attack had a robustness degree of 0.0018 with respect to the inferred formula, meaning it was "barely" missed.

3) Monitoring: Fig. 3(b) shows the robustness degree of each y_i at time t with respect to the parameterized formula

$$\phi(t) = F_{[0,t)}(G_{[0.5,\min(t,2.0))}(y > 0.9634)) \tag{12}$$

for $t > 0.1$. This serves as a rudimentary on-line monitor by quantifying how close an output signal y_i observed up to time t is to satisfying or violating the part of (11) that can be checked up to time t. It can be seen that all of the outputs initially have positive robustness, meaning they do not yet violate the formula. However, while the normal trajectories become more positively robust with respect to $\phi(t)$ over time, the robustness of the attack trajectories steadily declines.



(a) (b)

Fig. 3. (a) Outputs of system (9) (green) and (10) (red). The blue line indicate the threshold 0.9634 used in the inferred formula (11). (b) A simple monitor of the formula $\phi(t)$ given in (12).

B. Braked Train

1) Model: In this section, we apply our algorithm to the train braking scenario [10] used in Example 1. We model the train as a classical hybrid automaton [14]. A hybrid automaton produces continuous trajectories $x : \mathbb{R}^+ \to X \subseteq \mathbb{R}^n$. A trajectory x evolve according to dynamics which depend on the current discrete mode $q \in Q$ (denoted by a vertex of a graph) of the automaton. The mode of the system changes (denoted by edges of a graph) if a *guard condition* over the state of the system is satisfied. During a transition, the state of the system may change discontinuously according to a *reset relation*. Here we denote guards in black text and reset relations in red text.

The hybrid automaton H_T which describes the total model of the train consists of 3 identical braking subsystems with modes $Q_{b_k} = \{q_{b_k,j}\}_{j=1}^5$ that describe the state of each brake and a velocity subsystem with modes $Q_v = \{q_{v,j}\}_{j=1}^3$ which describes the dynamics of the train's velocity. The subsystem associated with brake 1 is shown in Fig. 4(a). The noise processes $n_1 \ldots n_5$ are all Gaussian processes with variance 1, 0.1, 0.3, 3, and 3, respectively. The brake remains in mode $q_{b_1,1}$ during acceleration until $v(t)$ exceeds a threshold v_{max}. At this point, the system transitions to the delay mode $q_{b_1,2}$ before moving to the braked mode $q_{b_1,3}$. After the velocity is decreased below v_{min}, the system transitions to a second delay state $q_{b_1,4}$ before returning to mode $q_{b_1,1}$. An adversary can disable the brakes (denoted by the exogenous event $attack_1$), which forces the system to transition from $q_{b_1,2}$ to a failure mode $q_{b_1,5}$. Brakes in the other two cars can similarly be disabled.

The velocity subsystem is shown in Fig. 4(b). The velocity of the train begins in mode $q_{v,1}$ (blue in Fig.s 1 and 2) and accelerates to v_{max}. The dynamics of the train shift to the higher velocity mode $q_{v,2}$ (green). Once at least one brake engages, the system transitions to a decelerating mode (black).

2) Formula inference: We used the model given in the previous section to generate 50 outputs. 43 of the trajectories were from normal operation and 7 were from an attacked operation. We only considered attacks in which all of the brakes were disabled ($b = 3$). Our algorithm inferred the formula

$$F_{[0,100)}(F_{[10,69)}(y < 24.9) \wedge F_{[13.9,44.2)}(y > 17.66)). \tag{13}$$

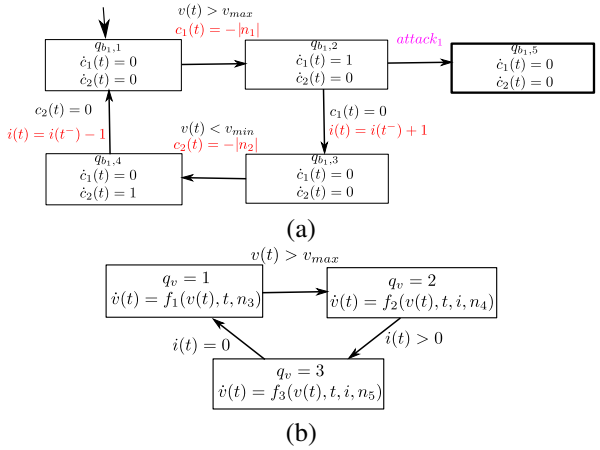


Fig. 4. (a) ECP braking subsystem of the first car in the train. (b) Velocity subsystem of the entire train.

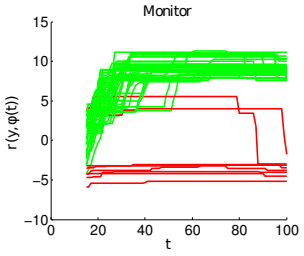


Fig. 5. On-line monitoring of train output with respect to (14).

In plain English, (13) means "At some point, between 10s and 69s in the future the output dips below 24.9 m/s and the output exceeds 17.66 m/s. between 10 and 44.2 s in the future." This is consistent with the oscillatory nature of the velocity output under normal behavior, as the velocity must increase and decrease over a window.

The formula (13) perfectly separates the data, i.e. the misclassification rate is 0. The formula was inferred using 15 simulated annealing cycles with 15 sample points per cycle. The computation time was 154 s on the same PC as described in Section V-A.

3) Monitoring: Fig. 5 shows the robustness degree of the train's output signal with respect to

$$\phi(t) = \begin{array}{l} F_{[0,t)}(F_{[10,\min(t,69))}(y < 24.9) \\ \wedge F_{[13.9,\min(t,44.2))}(y > 17.66)). \end{array} \quad (14)$$

The robustness of the normal outputs are shown in green and the robustness of the attacked outputs are shown in red. As you can see from Fig. 5, many of the normal outputs initially have negative robustness. However, as time goes on, the robustness measure improves for all of the normal outputs. In contrast, the performance of (most) of the attacked outputs remains low and worsens over time. By time $t = 40$, all but two of the attacked outputs are clearly separated from the normal outputs.

VI. Conclusion

In this paper, we consider a general framework for anomaly detection for cyber-physical systems security. In place of using classical anomaly detection tools, we apply a formal methods approach to the problem. We designed and implemented an algorithm which is able to infer a data classifier in the form of a signal temporal logic formula from unlabeled data. The inferred formula can be interpreted in natural language and can be used in the future for on-line monitoring. We demonstrated our approach using two case studies, including a model of a train under attack. Our approach was able to classify the attacked and normal outputs for both case studies with low misclassification rates. Further, we used the formula to test a simple on-line monitor. Results indicate that the monitors provide early warning for systems under attack.

Acknowledgment

This work was partially supported by ONR under grants ONR MURI N00014-10-10952, ONR MURI N00014-09-1051 and ONR N00014-14-1-0554 ONR, and supported by NSF under grant NSF CNS-1035588.

References

[1] J. Slay and M. Miller, *Lessons learned from the maroochy water breach*. Springer, 2007.

[2] F. Pasqualetti, F. Dorfler, and F. Bullo, "Cyber-physical attacks in power networks: Models, fundamental limitations and monitor design," in *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*. IEEE, 2011, pp. 2195–2201.

[3] A. Teixeira, D. Pérez, H. Sandberg, and K. H. Johansson, "Attack models and scenarios for networked control systems," in *Proceedings of the 1st international conference on High Confidence Networked Systems*. ACM, 2012, pp. 55–64.

[4] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[5] K. Kowalska and L. Peel, "Maritime anomaly detection using gaussian process active learning," in *Information Fusion (FUSION), 2012 15th International Conference on*. IEEE, 2012, pp. 1164–1171.

[6] Z. Kong, A. Jones, A. Medina Ayala, E. Aydin Gol, and C. Belta, "Temporal logic inference for classification and prediction from data," in *The 17th International Conference on Hybrid Systems: Computation and Control (HSCC)*, Berline, Germany, 2014.

[7] X. Jin, A. Donze, J. Deshmukh, and S. Seshia, "Mining requirements from closed-loop control models," in *Hybrid Systems: Computation and Control (HSCC)*, 2013.

[8] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.

[9] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Formal Modeling and Analysis of Timed Systems*. Springer, 2010, pp. 92–106.

[10] A. P. Sistla, M. Žefran, and Y. Feng, "Monitorability of stochastic dynamical systems," in *Computer Aided Verification*. Springer, 2011, pp. 720–736.

[11] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pp. 71–76, 2004.

[12] E. Asarin, A. Donzé, O. Maler, and D. Nickovic, "Parametric identification of temporal properties," in *Runtime Verification*. Springer, 2012, pp. 147–160.

[13] H. J. Shin, D.-H. Eom, and S.-S. Kim, "One-class support vector machinesan application in machine fault detection and classification," *Computers & Industrial Engineering*, vol. 48, no. 2, pp. 395–408, 2005.

[14] J. Lygeros, K. Johansson, S. Sastry, and M. Egerstedt, "On the existence of executions of hybrid automata," in *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*, vol. 3, 1999, pp. 2249–2254 vol.3.