Generating Automated and Online Test Oracles for Simulink Models with Continuous and Uncertain Behaviors

Claudio Menghi
University of Luxembourg
Luxembourg
claudio.menghi@uni.lu

Shiva Nejati
University of Luxembourg
Luxembourg
shiva.nejati@uni.lu

Khouloud Gaaloul
University of Luxembourg
Luxembourg
khouloud.gaaloul@uni.lu

Lionel C. Briand
University of Luxembourg
Luxembourg
lionel.briand@uni.lu

ABSTRACT

Test automation requires automated oracles to assess test outputs. For cyber physical systems (CPS), oracles, in addition to be automated, should ensure some key objectives: (i) they should check test outputs in an online manner to stop expensive test executions as soon as a failure is detected; (ii) they should handle time- and magnitude-continuous CPS behaviors; (iii) they should provide a quantitative degree of satisfaction or failure measure instead of binary pass/fail outputs; and (iv) they should be able to handle uncertainties due to CPS interactions with the environment. We propose an automated approach to translate CPS requirements specified in a logic-based language into test oracles specified in Simulink – a widely-used development and simulation language for CPS. Our approach achieves the objectives noted above through the identification of a fragment of Signal First Order logic (SFOL) to specify requirements, the definition of a quantitative semantics for this fragment and a sound translation of the fragment into Simulink. The results from applying our approach on 11 industrial case studies show that: (i) our requirements language can express all the 98 requirements of our case studies; (ii) the time and effort required by our approach are acceptable, showing potentials for the adoption of our work in practice, and (iii) for large models, our approach can dramatically reduce the test execution time compared to when test outputs are checked in an offline manner.

CCS CONCEPTS

• **Software and its engineering** → **Software verification and validation**; **Formal language definitions**.

KEYWORDS

Cyber Physical Systems, Test Oracle, Signal Logic, Monitoring

ACM Reference Format:
Claudio Menghi, Shiva Nejati, Khouloud Gaaloul, and Lionel C. Briand. 2019. Generating Automated and Online Test Oracles for Simulink Models with Continuous and Uncertain Behaviors. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3338906.3338920

1 INTRODUCTION

The development of Cyber Physical Systems (CPSs) starts by specifying CPS behaviors as executable models described in languages such as Matlab/Simulink [3]. These models are complex and subject to extensive testing before they can be used as a basis for software code development. Existing research on automated testing of CPS models has largely focused on automated generation of test suites [39, 55, 58]. However, in addition to test input generation, test automation requires *automated oracles* [26], i.e., a mechanism to automatically determine whether a test has passed or failed.

To automate oracles, engineers often rely on runtime crashes (a.k.a. implicit oracles [59]) to detect failures. However, implicit oracles often cannot effectively reveal violations of functional requirements as most of such violations do not lead to crashes. As mandated by safety certification standards [42], for CPS, functional requirements must be specified, and be used as the main authoritative reference to derive test cases and to demonstrate system behavior correctness. To achieve this goal, we need to develop oracles that can automatically check the correctness of system behaviors with respect to requirements. In this paper, we propose an approach to generating oracles that *automatically* determine whether outputs of CPS models satisfy or violate their requirements. For CPS, oracles, in addition to be automated, need to contend with a number of considerations that we discuss and illustrate below.

Motivating Example. We motivate our work using SatEx, a real-world satellite model [2]. SatEx is modeled in Matlab/Simulink and developed by our partner LuxSpace. The main functional requirements of SatEx are presented in the middle column of Table 1, and the variables used in the requirements are described in Table 2.

Before software coding or generating code from Simulink models (a common practice when Simulink/Matlab models are used), engineers need to ensure that their models satisfy the requirements of interest (e.g., those in Table 1). Although there are a few automated verification tools for Simulink, in practice, verification of CPS Simulink models largely relies on simulation and testing. This is because existing tools for verifying Simulink models [6, 7, 10, 63] are not amenable to verification of large Simulink models like SatEx that contain continuous physical computations and third-party library code [15, 52]. Further, CPS Simulink models often capture dynamic and hybrid systems [17]. It is well-known that model checking such systems is in general undecidable [16, 18, 40].

To effectively test CPS models, engineers need to have automated test oracles that can check the correctness of simulation outputs with respect to the requirements. To be effective in the context of CPS testing, oracles should further ensure the following objectives:

Table 1: Requirements for the satellite control system (SatEx) developed by LuxSpace.

ID	Requirement	Restricted Signal First-Order logic formula*
R1	The angular velocity of the satellite shall always be lower than $1.5m/s$.	$\forall t \in [0, 86\,400)\colon \|\vec{w}_{sat}(t)\| < 1.5$
R2	The estimated attitude of the satellite shall be always equal to 1.	$\forall t \in [0, 86\,400)\colon \|\vec{q}_{estimate}(t)\| = 1$
R3	The maximum reaction torque must be equal to $0.015Nm$.	$\forall t \in [0, 86\,400)\colon \|\vec{trq}(t)\| \leq 0.015$
R4	The satellite attitude shall reach close to its target value within 2 000 sec (with a deviation not more than 2 degrees) and remain close to its target value.	$\forall t \in [2\,000, 86\,400)\colon \|\vec{q}_{real}(t) - \vec{q}_{target}(t)\| \leq 2$
R5	The satellite target attitude shall not change abruptly: for every t, the difference between the current target attitude and the one at two seconds later shall not be more than $\alpha°$.	$\forall t \in [0, 86\,400)\colon \|\vec{q}_{target}(t) - \vec{q}_{target}(t+2)\| \leq 2 \times sin(\frac{\alpha}{2})$
R6	The satellite shall reach close to its desired attitude (with a deviation not more than %2) 2000 sec after it enters its normal mode (i.e., $sm(t) = 1$) and it has stayed in that mode for at least 1 sec.	$\forall t \in [0, 86\,400)\colon (sm(t) = 0 \wedge (\forall t_1 \in (t, t+1]\colon sm(t_1) = 1) \rightarrow \|\vec{q}_{real}(t + 2000) - \vec{q}_{estimate}(t + 2000)\| \leq 0.02)$

* The notation $\vec{a}$ indicates that a is a vector; $\|\vec{a}\|$ indicates the norm of the vector.

Table 2: Signals variables of the SatEx model.

Var.	Description	Var.	Description
sm	Satellite mode status.	$\vec{trq}$	Satellite torque.
$\vec{w}_{sat}$	Satellite angular velocity.	$\vec{q}_{real}$	Current satellite attitude.
$\vec{q}_{estimate}$	Estimated satellite attitude.	$\vec{q}_{target}$	Target satellite attitude.

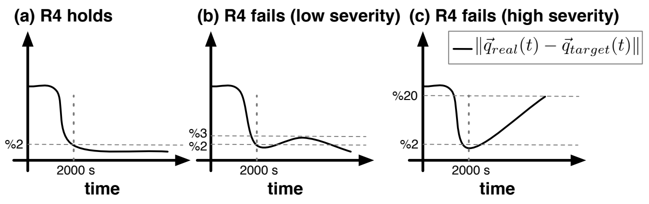


Figure 1: Three simulation outputs of our SatEx case study model indicating the error signal $\|\vec{q}_{real}(t) - \vec{q}_{target}(t)\|$. The signal in (a) passes R4 in Table 1, but those in (b) and (c) violate R4 with low and high severity, respectively.

O1. *Test oracles should check outputs in an online mode.* An online oracle (a.k.a as a monitor in the literature [23]) checks output signals as they are generated by the model under test. Provided with an online oracle, engineers can stop model simulations as soon as failures are identified. Note that CPS Simulink models are often computationally expensive because they have to capture physical systems and processes using high-fidelity mathematical models with continuous behaviors. Further, CPS models have to be executed for a large number of test cases. Also, due to the reactive and dynamic nature of CPS models, individual test executions (i.e., simulations) have to run for long durations to exercise interactions between the system and the environment over time. For example, to simulate the satellite behavior for $24h$ (i.e., 86 400s), the SatEx model has to be executed for 84 minutes (~1.5 hours) on 12-core Intel Core $i7$ 3.20GHz 32GB of RAM. Further, the $24h$-length simulation of SatEx has to be (re)run for tens or hundreds of test cases. Therefore, online test oracles are instrumental to reduce the total test execution time and to increase the number of executed test cases within a given test budget time.

O2. *Test oracles should be able to evaluate time and magnitude-continuous signals.* CPS model inputs and outputs are signals, i.e., functions over time. Signals are classified based on their time-domain into time-discrete and time-continuous, and based on their value-range into magnitude-discrete and magnitude-continuous. The type of input and output signals depends on the modeling formalisms. For example, differential equations [57] often used in physical modeling yield continuous signals, while finite state automata [44] used to specify discrete-event systems generate discrete signals. Figure 1 shows three magnitude- and time-continuous signal outputs of SatEx indicating the error in the satellite attitude, i.e., the difference between the real and the target satellite attitudes ($\|\vec{q}_{real}(t) - \vec{q}_{target}(t)\|$). An effective CPS testing framework should be able to handle the input and output signals of different CPS formalisms including the most generic and expressive signal type, i.e., time-continuous and magnitude-continuous. Such testing frameworks are then able to handle any discrete signal as well.

O3. *Test oracles for CPS should provide a quantitative measure of the degree of satisfaction or violation of a requirement.* Test oracles typically classify test results as failing and passing. The boolean partition into "pass" and "fail", however, falls short of the practical needs. For CPS, test oracles should assess test results in a more nuanced way to identify among all the passing test cases, those that are more acceptable, and among all the failing test cases, those that reveal more severe failures. Therefore, an effective test oracle for CPS should assess test results using a *quantitative fitness measure*. For example, the satellite attitude error signal in Figure 1(a) satisfies the requirement **R4** in Table 1. But, signals in Figures 1(b) and (c) violate **R4** since the error signal does not remain below the %2 threshold after 2000s. However, the failure in Figure 1(c) is more severe than that in Figure 1(b) since the former deviates from the threshold with a larger margin. A quantitative oracle can differentiate between these failures.

O4. *Test oracles should be able to handle uncertainties in CPS function models.* We consider two main recurring and common sources of uncertainties in CPS [30, 36]: (1) Uncertainty due to unknown hardware choices which results in model parameters whose values are only known approximately at early design stages. For example, in SatEx, there are uncertainties in the type of the magnetometer and in the accuracy of the sun sensors mounted on the satellite (see Table 3). (2) Uncertainty due to the noise in the inputs received from the environment, particularly in the sensor readings. This is typically captured by white noise signals applied to the model inputs (e.g., Table 3 shows the signal-to-noise (S2N) ratios for the magnetometer and sun sensor inputs of SatEx). Oracles for CPS models should be able to assess outputs of models that contain parameters with uncertain values and signal inputs with noises.

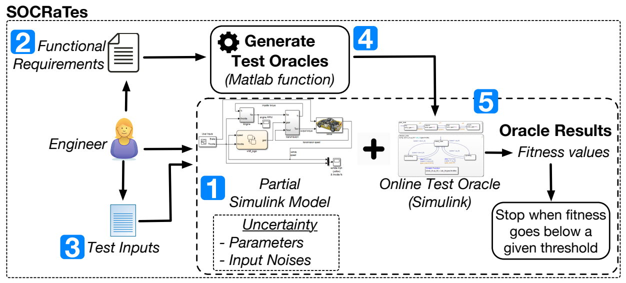

SOCRaTes
2 Functional Requirements
Generate Test Oracles (Matlab function)
4
5
Engineer
Partial Simulink Model
Online Test Oracle (Simulink)
Oracle Results
Fitness values
Stop when fitness goes below a given threshold
1
Uncertainty
- Parameters
- Input Noises
3 Test Inputs

Figure 2: Overview of SOCRaTes, our automated oracle generation approach.

Table 3: Uncertainty in SatEx: The values of the magnetometer type and the sun sensor accuracy parameters are given as ranges (middle column). The noise values for the magnetometer and sun sensor inputs are given in the right column.

Component	Parameter Values	Noises (S2N)
Magnetometer	[60000, 140000] nT	$100 \cdot e^{-12}$ T/$\sqrt{\text{Hz}}$
Sun sensor	$2.9 \cdot 10^{-3} \pm 10\%$	$2.688 \cdot e^{-6}$ A

Contributions. We propose *Simulink Oracles for CPS RequiremenTs with uncErtainty (SOCRaTEs)*, an approach for generating online oracles in the form of Simulink blocks based on CPS functional requirements (Section 2). Our oracle generation approach achieves the four objectives discussed above through the following novel elements:

- We propose Restricted Signals First-Order Logic (RFOL), a signal-based logic language to specify CPS requirements (Section 3). RFOL is a restriction of Signal First Order logic [22] (SFOL) that can capture properties of time- and magnitude-continuous signals while enabling the generation of efficient, online test oracles. We define a quantitative semantics for RFOL to compute a measure of fitness for test results as oracle outputs.

- We develop a procedure to translate RFOL requirements into automated oracles modeled in the Simulink language (Section 4). We prove the soundness of our translation with respect to the quantitative semantics of RFOL. Further, we demonstrate that: (1) the generated oracles are able to identify failures as soon as they are revealed (i.e., our oracles are online); and (2) our oracles can handle models containing parameters with uncertain values and signal inputs with noises by exploiting existing Simulink features. We have implemented our automated oracle generation procedure in a tool which is available online [1].

We apply our approach to 11 industry Simulink models from two companies in the CPS domain [2, 5]. Our results show that our proposed logic-based requirements language (RFOL) is sufficiently expressive to specify all the 98 CPS requirements in our industrial case studies. Further, our automated translation can generate online test oracles in Simulink efficiently, and the effort of developing RFOL requirements is acceptable, showing potentials for the practical adoption of our approach. Finally, for large and computationally intensive industry models, our online oracles can bring about dramatic time savings by stopping test executions long before their completion when they find a failure, without imposing a large time overhead when they run together with the model.

Structure. Section 2 outlines SOCRaTEs and its underlying assumptions. Section 3 presents the Restricted Signals First-Order Logic and its semantics. Section 4 describes our automated oracle generation procedure. Section 5 evaluates SOCRaTEs. Section 6 presents the related work and Section 7 concludes the paper.

2 SOCRATES

Figure 2 shows an overview of SOCRaTeS (*Simulink Oracles for CPS RequiremenTs with uncErtainty*), our approach to generate automated test oracles for CPS models. SOCRaTeS takes three inputs: (1) a CPS model with parameters or inputs involving uncertainties, (2) a set of functional requirements for the CPS model and (3) a set of test inputs that are developed by engineers to test the CPS model with respects to its requirements. SOCRaTeS makes the following assumptions about its inputs:

A1. *The CPS model is described in Simulink* (1). Simulink is used by more than 60% of engineers for simulation of CPS [25, 72], and is the prevalent modeling language in the automotive domain [53, 71]. It is particularly suitable for specifying dynamic systems, is executable and allows engineers to test their models as early as possible.

A2. *Functional requirements are described in a signal logic-based language* (2). We present our requirements language in Section 3 and compare it with existing signal logic languages [22, 49]. We evaluate expressiveness of our language in Section 5.

A3. *A set of test inputs exercising requirements are provided* (3). We assume engineers have a set of test inputs for their CPS model. The test inputs may be generated manually, randomly or based on any test generation framework proposed in the literature [53, 71]. Our approach is agnostic to the selected test generation method.

SOCRaTeS automatically converts functional requirements into oracles specified in Simulink (4). The oracles evaluate test outputs of the CPS model in an automated and online manner and generate fitness values that provide engineers with a degree of satisfaction or failure for each test input (5). Engineers can stop running a test in the middle when SOCRaTeS concludes that the test fitness is going to remain below a given threshold for the rest of its execution.

3 CONTEXT FORMALIZATION

In Section 3.1, we describe CPS Simulink models (1) without and with uncertainty and their inputs (3). In Section 3.2, we present *Restricted Signals First-Order Logic (RFOL)*, the logic we propose to specify CPS functional requirements (2). In Section 3.3, we describe how oracles compute fitness values of test inputs (5).

3.1 Simulink Models

Simulink is a data-flow-based visual language that can be executed using Matlab and consists of blocks, ports and connections. Blocks typically represent operations and constants, and are tagged with ports that specify how data flow in and out of the blocks. Connections establish data-flows between ports.

To simulate a Simulink model M, the simulation engine receives signal inputs defined over a time domain and computes signal outputs at successive time steps over the same time domain used for the inputs. A *time domain* $\mathbb{T} = [0, b]$ is a non-singular bounded interval of $\mathbb{R}$. A *signal* is a function $f : \mathbb{T} \to \mathbb{R}$. A *simulation*, denoted by $H(I, M) = O$, receives a set $I = \{i_1, i_2 \dots i_m\}$ of input

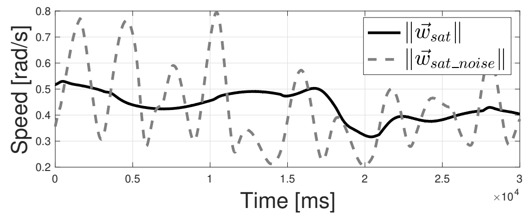


Figure 3: Signals $\|\vec{w}_{sat}\|$ for the w_{sat} output of SatEx. The solid-line signal is generated by SatEx with no uncertainty, and the dashed-line signal is generated when the S2N ratios in Table 3 are applied to the SatEx inputs.

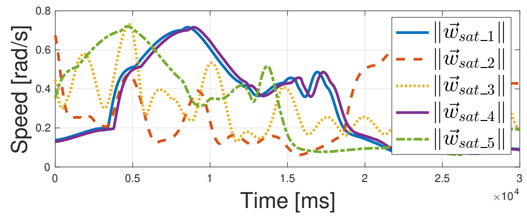


Figure 4: A set of signals $\|\vec{w}_{sat}\|$ for the output w_{sat} of SatEx with uncertain parameters (i.e., when the sun sensor and magnetometer parameters are specified as in Table 3).

signals and produces a set $O = \{o_1, o_2 \dots o_n\}$ of output signals such that each $o_i \in O$ corresponds to one model output. For example, Figure 3 shows a signal (black solid line) for the w_{sat} output of SatEx computed over the time domain $[0, 3 \times 10^4]$.

Simulink uses numerical algorithms [11] referred to as *solvers* to compute simulations. There are two main types of solvers: fixed-step and variable-step. Fixed-step solvers generate signals over discretized time domains with equal-size time-steps, whereas variable-step solvers (e.g., Euler, Runge-Kutta [21]) generate signals over continuous time domains. While the underlying techniques and details of numerical solvers are outside the scope of this paper, we note that our oracles rely on Simulink solvers to properly handle signals based on their time domains, whether discrete or continuous. As a result, our work, in contrast to existing techniques, is able to seamlessly handle the verification of logical properties over not just discrete but also continuous CPS models.

Simulink has built-in support to specify and simulate some forms of uncertainty. We refer to Simulink models that contain uncertain elements as *partial* models (denoted M_p), while we use the term *definitive* to indicate models with no uncertainty. Simulink can specifically capture the following two kinds of uncertainty that are common for CPS and also discussed in Section 1 (objective **O4**) [61]:

(i) Uncertainty due to the noise in inputs. In Simulink, uncertainty due to the noise is implemented by augmenting model inputs with continuous-time random signals known as *White Noise (WN)* [38]. The degree of WN for each input is controlled by a *signal-to-noise ratio (S2N)* value which is the ratio of a desired signal over the background WN [9]. Table 3 shows the S2N ratios for two inputs of SatEx. Fig. 3 shows the signal $\|\vec{w}_{sat}(t)\|$ (gray dashed line) after adding some noise to the original w_{sat} signal (black solid line).

(ii) Uncertainty related to parameters with unknown values. In Simulink, parameters whose values are uncertain are typically defined using variables of type *uncertain real* (ureal), which is a built-in type in Matlab/Simulink that specifies a range of values for a variable [13]. Table 3 shows two parameters of SatEx whose exact values are unknown, and hence, value ranges are assigned to them.

Let M_p be a partial Simulink model with n outputs, and let k be the number of different value assignments to uncertain parameters of M_p. A *simulation* of a partial Simulink model M_p, denoted by $H_p(I, M_p) = \{O_1, O_2 \dots O_k\}$, receives a set $I = \{i_1, i_2 \dots i_m\}$ of input signals defined over the same time domain, and produces a set of simulation outputs $\{O_1, O_2 \dots O_k\}$ such that each O_i is generated by one value assignment to uncertain parameters of M_p. Specifically, for each $O_i \in \{O_1, O_2 \dots O_k\}$, we have $O_i = \{o_1, o_2 \dots o_n\}$ such that $o_1, \dots o_n$ are signals for outputs of M_p, i.e., each O_i contains a signal for each output of M_p. The function H_p generates the simulation outputs consecutively and is provided in the Robust Control Toolbox of Simulink [8] which is the uncertainty modeling and simulation tool of Simulink models with dynamic behavior. The value of k indicating the number of value assignments to uncertain parameters can either be specified by the user or selected based on the recommended settings of H_p. For example, Figure 4 plots five simulation outputs for the output w_{sat} of SatEx. The uncertainty in this figure is due to the sun sensor accuracy parameter that takes values form the range $2.9 \cdot 10^{-3} \pm 10\%$ as indicated in Table 3.

3.2 Our Requirements Language

Our choice of a language for CPS requirements is mainly driven by the objectives **O1** and **O2** described in Section 1. These two objectives, however, are in conflict. According to **O2**, the language should capture complex properties involving magnitude- and time-continuous signals. Such language is expected to have a high runtime computational complexity [23]. This, however, makes the language unsuitable for the description of online oracles that should typically have low runtime computational complexity, thus contradicting **O1**. For example, Signals First Order (SFO) logic [23] is an extension of first order logic with continuous signal variables. SFO, however, is not amenable to online checking in its entirety due to its high expressive power that leads to high computational complexity of monitoring SFO properties [23]. Thus, the procedure for monitoring SFO properties is tailored to offline checking. In order to achieve both **O1** and **O2**, we define Restricted Signals First-Order Logic (RFOL), a fragment of SFO. RFOL can be effectively mapped to Simulink to generate online oracles that run together with the model under test by the same solvers applied to the model, which can handle any signal type (i.e., discrete or continuous), hence addressing both **O1** and **O2**. Note that even though RFOL is less expressive than SFO, as we will discuss in Section 5, all CPS requirements in our case studies can be captured by RFOL.

RFOL Syntax. Let $T = \{t_1, t_2, \dots t_d\}$ be a set of *time variables*. Let $F = \{f_1, f_2, \dots, f_l\}$ be a set of signals defined over the same time domain $\mathbb{T}$, i.e., $f_i : \mathbb{T} \rightarrow \mathbb{R}$ for every $1 \leq i \leq l$.

Let us consider the grammar $\mathcal{G}$ defined as follows:

$$\begin{aligned}
\tau &::= \quad t + n \mid t - n \mid t \mid n \\
\rho &::= \quad f(\tau) \mid \mathbf{g}(\rho) \mid \mathbf{h}(\rho_1, \rho_2) \\
\phi &::= \quad \rho \sim r \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \forall t \in \langle \tau_1, \tau_2 \rangle : \phi \mid \exists t \in \langle \tau_1, \tau_2 \rangle : \phi
\end{aligned}$$

where $n \in \mathbb{R}_0^+$, $t \in T$, $f \in F$, $r \in \mathbb{R}$, and $\mathbf{g}$ and $\mathbf{h}$ are, respectively, arbitrary unary and binary arithmetic operators, $\sim$ is a relational operator in $\{<, \leq, >, \geq, =, \neq\}$, and $\langle \tau_1, \tau_2 \rangle$ is a *time interval* of $\mathbb{T}$

(i.e., $\langle \tau_1, \tau_2 \rangle \subseteq \mathbb{T}$) with lower bound τ_1 and upper bound τ_2. The symbols $\langle$ and $\rangle$ are equal to [or (, respectively to] or), depending on whether τ_1, respectively τ_2, are included or excluded from the interval. We refer to τ, ρ and ϕ as *time term*, *signal term* and *formula term*, respectively. A *predicate* is a formula term in the form $\rho \sim r$.

Definition 3.1. A *Restricted Signals First-Order Logic* (RFOL) formula φ is a formula term defined according to the grammar $\mathcal{G}$ that also satisfies the following conditions: (1) φ is closed, i.e., it does not have any free variable; and (2) every sub-formula of φ has at most one free time variable.

In RFOL, boolean operations ($\wedge$, $\vee$) combine predicates of the form $\rho \sim r$, which compare signal terms with real valuesNote that in our logic, negation $\neg$ is applied at the level of predicates. The formulas further quantify over time variables of signal terms ρ in $\rho \sim r$ and bound them in time intervals $\langle \tau_1, \tau_2 \rangle$. Table 1 shows the formalization of the SatEx requirements in RFOL. For example, the predicate $\|\vec{w}_{sat}\| < 1.5$ of formula $R1$ states that the angular velocity of the satellite should be less than 1.5m/s, and $\forall t \in [0, 86\,400)$ forces the predicate to hold for a duration of 86 400s $\simeq$ 24h, the estimated time required for the satellite to finish an orbit.

RFOL expressiveness. Here, we discuss what types of SFO properties are eliminated from RFOL due to the conditions in Definition 3.1. Condition 1 in Definition 3.1 requires closed formulas. RFOL properties must not include free variables (i.e., they should be formulas and not queries) so that they generate definitive results when checking test outputs. Condition 2 in Definition 3.1 is needed to ensure that the formulas can be translated into online oracles specified in Simulink. This condition eliminates formulas containing predicates $\rho \sim r$ where ρ includes an arithmetic operator applied to signal segments over different time intervals (i.e., signal segments with different time scopes). For example, the formula $\forall t \in [1, 5] : \forall t' \in [7, 9] : f(t) + f(t') < 4$ is not in RFOL since $f(t) + f(t') < 4$ has two free time variables t and t' (i.e., it violates condition 2 in Definition 3.1). The predicate $f(t) + f(t') < 4$ in this formula computes the sum of two segments of signal f related to time intervals $[1, 5]$ and $[7, 9]$. Such formulas are excluded from RFOL since during online checking, the operands $f(t)$ and $f(t')$ cannot be simultaneously accessed to compute $f(t)+f(t')$. We note that formulas with arithmetic operators applied to signal segments over the *same* time interval (e.g., R4 and R5 in Table 1), or formulas involving different predicates over different time intervals, but connected with *logical* operators (e.g., R6) are included in RFOL.

Comparison with STL. In addition to SFO, Signal Temporal Logic (STL) [49] is another logic proposed in the literature that can capture CPS continuous behaviors. We compare RFOL with STL, and in particular, with *bounded* STL since test oracles can only check signals generated up to a given bound. Hence, for our purpose, bounded STL temporal operators have to be applied (e.g., $\mathcal{U}_{[a,b]}$). RFOL subsumes bounded STL since boolean operators of STL can be trivially expressed in RFOL, and any temporal STL formula in the form of $\varphi_1 \,\mathcal{U}_{[a,b]}\, \varphi_2$ can also be specified in RFOL using time terms and time intervals. The detailed translation is available online [1].

RFOL Semantics. We propose a (quantitative) semantics for RFOL to help engineers distinguish between different degrees of satisfaction and failure (objective **O3**). As shown in Table 1 and also based on RFOL syntax, CPS requirements essentially check predicates $\rho \sim r$ over time. To define a quantitative semantics for RFOL, we need to first define the semantics of these predicates in a quantitative way. We define a (domain-specific) *diff* function to assign a fitness value to $\rho \sim r$. We require *diff* to have these characteristics: (1) The range of *diff* is $[-1, 1]$. (2) A value in $[0, 1]$ indicates that $\rho \sim r$ holds, and a value in $[-1, 0)$ indicates that $\rho \sim r$ is violated.

Definition 3.2. Let *diff* be a domain-specific semantics function for predicates $\rho \sim r$. Let $F = \{f_1, \ldots, f_l\}$ be a set of signals with the same time domain $\mathbb{T}$. The semantics of an RFOL formula ϕ for the signal set F is denoted by $[\![\phi]\!]_F$ and is defined as follows:

$$
\begin{aligned}
[\![f(n)]\!]_F &= \begin{cases} f(n) & \text{if } f \in F \text{ and } n \in \mathbb{T} \\ \text{undefined} & \text{otherwise} \end{cases} \\
[\![\mathbf{g}(\rho)]\!]_F &= \mathbf{g}([\![\rho]\!]_F) \\
[\![\mathbf{h}(\rho_1, \rho_2)]\!]_F &= \mathbf{h}([\![\rho_1]\!]_F, [\![\rho_2]\!]_F) \\
[\![\rho \sim r]\!]_F &= \mathit{diff}([\![\rho]\!]_F \sim r) \\
[\![\phi_1 \wedge \phi_2]\!]_F &= \mathit{min}([\![\phi_1]\!]_F, [\![\phi_2]\!]_F) \\
[\![\phi_1 \vee \phi_2]\!]_F &= \mathit{max}([\![\phi_1]\!]_F, [\![\phi_2]\!]_F) \\
[\![\forall t \in \langle n_1, n_2 \rangle : \phi]\!]_F &= \underset{\forall t' \in \langle n_1, n_2 \rangle}{\mathit{min}} ([\![\phi[t \leftarrow t']]\!]_F) \\
[\![\exists t \in \langle n_1, n_2 \rangle : \phi]\!]_F &= \underset{\forall t' \in \langle n_1, n_2 \rangle}{\mathit{max}} ([\![\phi[t \leftarrow t']]\!]_F)
\end{aligned}
$$

The choice of the *max* and *min* operators for defining the semantics of $\exists$ and $\forall$ is standard [46]: the minimum has the same behavior as $\wedge$ and evaluates whether a predicate holds over the entire time interval. Dually, the max operator captures $\vee$. The semantics of signal terms $f(n)$ depends on whether the signal is included in F and whether n is in the time domain $\mathbb{T}$, otherwise $f(n)$ is undefined. We say $\varphi \in$ *RFOL* is *well-defined with respect to a signal set F* iff no signal term in φ is undefined. To avoid undefined RFOL formulas, signal time domains $\mathbb{T}$ should be selected such that signal indices are included in $\mathbb{T}$, and further, the formula should not have negative signal indices. For example, for properties in Table 1, we need a time domain $\mathbb{T} = [0, 86\,400]$ for **R1** to **R4**, a time domain $\mathbb{T} = [0, 86\,402]$ for **R5**, and a time domain $\mathbb{T} = [0, 88\,400]$ for **R6**. Finally, we can infer the boolean semantics of RFOL from its quantitative semantics: For every formula term φ, we have $F \models \varphi$ iff $[\![\varphi]\!]_F \geq 0$. In other words, φ holds over the signal set F iff $[\![\varphi]\!]_F \geq 0$.

Let $\mu = [\![\rho]\!]_F - r$. In our work, we define *diff* as follows:

$$
\mathit{diff}([\![\rho]\!]_F = r) = \frac{-|\mu|}{|\mu| + 1} \qquad \mathit{diff}([\![\rho]\!]_F \neq r) = \begin{cases} \frac{|\mu|}{|\mu|+1} & \text{if } \mu \neq 0 \\ -\epsilon & \text{else} \end{cases}
$$

$$
\mathit{diff}([\![\rho]\!]_F \geq r) = \frac{\mu}{|\mu| + 1} \qquad \mathit{diff}([\![\rho]\!]_F > r) = \begin{cases} \frac{\mu}{|\mu|+1} & \text{if } \mu \neq 0 \\ -\epsilon & \text{else} \end{cases}
$$

$$
\mathit{diff}([\![\rho]\!]_F \leq r) = \frac{-\mu}{|\mu| + 1} \qquad \mathit{diff}([\![\rho]\!]_F < r) = \begin{cases} \frac{-\mu}{|\mu|+1} & \text{if } \mu \neq 0 \\ -\epsilon & \text{else} \end{cases}
$$

In the above, ϵ is an infinitesimal positive value that ensures *diff* < 0 when $\mu = 0$ and either $<$, $>$ or $\neq$ is used.

Our *diff* function satisfies the two conditions described earlier and is closed under logical $\wedge$ and $\vee$. For example, $(\rho \leq r) \wedge (\rho \geq r)$ is equal to $(\rho = r)$. Our *diff* function, further, provides a quantitative fitness measure distinguishing between different levels of satisfaction and refutation. Specifically, a higher value of *diff* indicates that $\rho \sim r$ is fitter (i.e., it better satisfies or less severely violates the requirement under analysis). For example, the *diff* value of the predicate $\|\vec{w}_{sat}\| < 1.5$ for the signals shown in Figure 4 is above zero implying that the signals satisfy the predicate. In contrast, the *diff* values for signals $\|\vec{q}_{real} - \vec{q}_{target}\|$ in Figures 1(b) and (c) are -0.5 and -0.95, respectively. This shows that the violation in Figure 1(c) is more severe than that in Figure 1(b).

The above *diff* function is only one alternative where we assume the fitness is proportional to the difference between ρ and r. We can define the *diff* function differently as long as the two properties described earlier are respected and the proposed semantics for *diff* respects logical conjunction and disjunction operators.

3.3 Test Oracles

In this section, we formally define our notion of test oracle. We specifically discuss test oracles for partial Simulink models since a definitive model is a specialization of a partial model. Recall that by simulating a partial Simulink model M_p for a given test input I, we obtain a set of k alternative signals for each output of M_p, while for a definitive model M, the simulation output contains only one signal for each model output.

Definition 3.3. Let M_p be a Simulink model under test, and let I be a test input for M_p defined over the time domain $\mathbb{T}$. Let φ be an RFOL formula formalizing a requirement of M_p. Suppose $\{O_1, O_2 \dots O_k\} = H_p(I, M_p)$ are the simulation results generated for the time domain $\mathbb{T}$. We denote the *oracle* value of φ for test input I over model M_p by oracle(M_p, I, φ) and compute it as follows:

$$\text{oracle}(M_p, I, \varphi) = \min_{O \in \{O_1, O_2 \dots O_k\}} [\![\varphi]\!]_O$$

Specifically, oracle(M_p, I, φ) indicates the fitness value of the test input I over model M_p and evaluated against requirement φ.

Recall that based on Definition 3.2, the oracle output is a value in $[-1, 1]$. For definitive models, the test yields a single set $O = \{o_1, \dots, o_n\}$ of simulation outputs, and hence, the oracle computes $[\![\varphi]\!]_O$, i.e., it evaluates φ over the set O of test outputs. As defined above, for a partial model, the oracle computes the minimum value of φ over every test output set. Hence, for a partial model, the fitness value for a test I is determined by the model output yielding the lowest fitness (i.e., the model output revealing the most severe failure or the model output yielding the lowest passable fitness).

4 ORACLE GENERATION

In this section, we present the oracle generation component of SOCRaTes (4 in Figure 2). This component automatically translates RFOL formulas into *online* test oracles specified in Simulink that can handle time and magnitude-continuous signals and conform to our notion of oracle described in Definition 3.3. Note that an RFOL formula may not be *directly* translatable into an online test oracle if it contains sub-formulas referring to future time instants or to signal values that are not yet generated at the current simulation time. For example, consider the predicate $\|\vec{q}_{real}(t + 2000) - \vec{q}_{estimate}(t + 2000)\| \leq 0.02$ in the **R6** property of Table 1. The fitness value of this predicate at t (i.e., the oracle output in Definition 3.3) can only be evaluated after generating signals $\vec{q}_{real}$ and $\vec{q}_{estimate}$ up to the time instant $t + 2000$. This requires extending the time domain $\mathbb{T}$ by 2000 seconds. Instead of forcing a longer simulation time, we propose a procedure that rewrites the RFOL formulas into a form that allows a direct translation into online test oracles. This procedure, called *time and interval shifting*, is presented in Section 4.1. Having applied the procedure to RFOL formulas, in Section 4.2, we describe our translation to convert RFOL formulas into *Simulink oracles*. We further present a proof of soundness and completeness of our translation in that section. All the proofs of the Theorems are provided in our online Appendix [1].

4.1 Time and Interval Shifting

Below, we present the *time-* and *interval-shifting* steps separately:

Time-shifting. Any signal term that refers to a signal value generated in the future should be rewritten as a signal term that does not refer to the future. For example, the formula $\vec{q}_{real}(t + 2000) < 5$ that refers to the value of $\vec{q}_{real}$ in the future cannot be checked online. Therefore, our time-shifting procedure replaces any signal term $f(t + n)$ with a signal term $f(t - n)$ as follows: Let ψ be an RFOL formula. We traverse ψ from its leaves to its root and replace every sub-formula $\forall t \in \langle n_1, n_2\rangle : \phi(t)$ (resp. $\exists t \in \langle n_1, n_2\rangle : \phi(t)$) of ψ with $\forall t \in \langle n_1 + d_t, n_2 + d_t\rangle : \phi(t - d_t)$ (resp. $\exists t \in \langle n_1 + d_t, n_2 + d_t\rangle : \phi(t - d_t)$), where d_t is the maximum value of constant n in time terms $t + n$ appearing as signal indices in $\phi(t)$. For example, the requirement **R5** in Table 1 is rewritten as:
$\forall t \in [2, 86\,402) : \|\vec{q}_{target}(t - 2) - \vec{q}_{target}(t)\| \leq 2 \times sin(\frac{\alpha}{2})$

Interval-Shifting. To ensure that ψ can be translated into an online test oracle, for any $\forall t \in \langle \tau_1, \tau_2\rangle : \phi$ in ψ, the interval $\langle \tau_1, \tau_2\rangle$ should end after all the intervals $\langle \tau_1', \tau_2'\rangle$ such that $\forall t' \in \langle \tau_1', \tau_2'\rangle : \phi'$ is a sub-formula of ϕ (i.e., $\tau_2 \geq \tau_2'$), and further, it should begin after all the intervals $\langle \tau_1', \tau_2'\rangle$ such that $\exists t' \in \langle \tau_1', \tau_2'\rangle : \phi'$ is a sub-formula of ϕ (i.e., $\tau_1 \geq \tau_2'$). Similarly, for any $\exists t \in \langle \tau_1, \tau_2\rangle : \phi$ in ψ, the dual of the above two conditions must hold. These conditions will ensure that the evaluation of the sub-formulas in the scope of t can be fully contained and completed within the evaluation of their outer formula. For example, $\forall t \in [0, 3] : (f(t) = 0 \wedge \forall t' \in [0, 5] : f(t') = 1)$ cannot be checked in an online way since the time interval of the inner sub-formula (i.e., $[0, 5]$) does not end before the time interval of the outer formula (i.e., $[0, 3]$). Therefore, our interval-shifting procedure shifts each time interval $\langle \tau_1, \tau_2\rangle$ to ensure that it terminates after all its related inner time intervals.

Let ψ be an RFOL formula. We traverse ψ from its leaves to its root and we perform the following operations: (i) replace every sub-formula $\forall t \in \langle \tau_1, \tau_2\rangle : \phi(t)$ of ψ with $\forall t \in \langle \tau_1 + d_u, \tau_2 + d_u\rangle : \phi(t - d_u)$, where d_u is the maximum value of constant n in the upper bounds τ_2 of time intervals $\langle \tau_1, \tau_2\rangle$ associated with $\forall$ operators and the lower bounds τ_1 of time intervals $\langle \tau_1, \tau_2\rangle$ associated with $\exists$ operators in $\phi(t)$; (ii) execute a dual procedure to update the time intervals of existential sub-formulae. For example, the interval-shifting procedure rewrites the formula previously introduced as $\forall t \in [2, 5] : f(t - 2) = 0 \wedge \forall t' \in [0, 5] : f(t') = 1$.

To ensure interval-shifting is applied to signal variables with constant indices, we replace every $f(n)$ in ψ where n is a constant with $\forall t^* \in [n, n] : f(t^*)$ where t^* is a new time variable that has not been used in ψ. We refer to the RFOL formula obtained by sequentially applying time-shifting and interval-shifting to an RFOL formula φ as *shifted-formula* and denote it by $\varphi_{\Uparrow}$.

THEOREM 4.1. *Let φ be an RFOL formula and let $\varphi_{\Uparrow}$ be its* shifted-formula. *For any signal set F, we have:* $[\![\varphi]\!]_F = [\![\varphi_{\Uparrow}]\!]_F$

The time complexity of generating a *shifted-formula* $\varphi_{\Uparrow}$ is $|\varphi|$ where $|\varphi|$ is the size of the formula φ, i.e., the sum of the number of its temporal and arithmetic operators. Both time and the interval shiftings scan the syntax tree of φ from its leaves to the root twice: one for computing the shifting values d_t and d_u for every subformula of φ; and the other to apply the shifting, i.e., replacing the variable t with $t - d_t$ or $t - d_u$.

4.2 From RFOL to Simulink

In this section, we translate RFOL formulas written in their shifted-forms (as described in Section 4.1) into Simulink. Table 4 presents the rules for translating each syntactic construct of RFOL defined in Definition 3.1 into Simulink blocks. Note that **h** and **g** in Table 4, respectively, refer to binary arithmetic operators (e.g., +) or unary functions (e.g., `sin`) and map to their corresponding Simulink operations. Below, we discuss the rules for t, $f(t-n)$, $\forall t \in \langle \tau_1, \tau_2 \rangle : \phi$, and $\rho \sim r$ since the other rules in Table 4 directly follow from the RFOL semantics. Note that signal variables in shifted formulas are all written as $f(t-n)$ s.t. $n \geq 0$. Hence, we give a translation rule for signal variables in the form of $f(t-n)$ only.

• Rule1: To compute the value of t, we use an integrator Simulink block to compute the formula $\int_0^t dt$ which yields t.

• Rule4: To encode $f(t-n)$, we first obtain the delay n applied to the signal f. To obtain the value of n from $t-n$, we compute $t-(t-n)$. We then use the *transport delay* block of Simulink to obtain the value of f at n time instants before t.

• Rule8: The formula $\forall t \in \langle \tau_1, \tau_2 \rangle : \phi$ is mapped into a Simulink model that initially generates the value 1 until the start of the time interval $\langle \tau_1, \tau_2 \rangle$. When $t \in \langle \tau_1, \tau_2 \rangle$ holds, the multiplexer of Rule8 selects the value of ϕ instead of 1. Note that we use symbol $<$ for $\langle =$ "(", symbol $\leq$ for $\langle =$ "[", symbol $>$ for $\rangle =$ ")", and symbol $\geq$ for $\rangle =$ "]". The feedback loop in the model combined with a delay block (i.e., z^{-1}) computes the minimum of ϕ over the time interval $\langle \tau_1, \tau_2 \rangle$. Once the time interval $\langle \tau_1, \tau_2 \rangle$ expires, the multiplexer chooses constant 1 again. This, however, has no side-effect on the value v already computed for the formula $\forall t \in \langle \tau_1, \tau_2 \rangle : \phi$ because $v \leq 1$ and the minimum of v and 1 remains v until the end of the simulation. Note that the rule for translating $\exists t \in \langle \tau_1, \tau_2 \rangle : \phi$ into Simulink is simply obtained by replacing in Rule8 MIN with MAX and constant 1 with constant -1.

• Rule9: Recall that the semantics of $\rho \sim r$ depends on a domain specific fitness function. In our work, we implement the diff block in Rule9 based on the functions given in Section 3.2 for function *diff*.

Let φ be an RFOL formula and $\varphi_{\Uparrow}$ be its corresponding shifted formula. We denote by M_φ the Simulink model obtained by translating $\varphi_{\Uparrow}$ using the rules in Table 4. The model M_φ is a definitive Simulink model and has one and only one output because every model fragment in Table 4 has one single output. This output will be indicated in the following with the symbol e. Below, we argue that M_φ conforms to our notion of test oracle given in Definition 3.3, and is an online oracle that can handle continuous signals. In order to use M_φ to check outputs of model M_p with respect to a property φ, it suffices to connect the outputs of M_p to the inputs of M_φ. We denote the model obtained by connecting the output ports of M_p to the input ports of M_φ by $M_p + M_\varphi$. Clearly, $M_p + M_\varphi$ has only one output signal e (i.e., the output of M_φ).

Theorem 4.2. *Let M_p be a (partial) Simulink model, and let I be a test input for M_p defined over the time domain $\mathbb{T} = [0, t_u]$. Let φ be a requirement of M_p in RFOL. Suppose $\{O_1, O_2 \dots O_k\} = H_p(I, M_p)$ and $\{\{e_1\}, \{e_2\}, \dots, \{e_k\}\} = H_p(I, M_p + M_\varphi)$ are simulation results generated for the time domain $\mathbb{T}$. Then, the value of φ over every signal set $O_i \in \{O_1, O_2 \dots O_k\}$ is equal to the value of the signal e_i generated by $M_p + M_\varphi$ at time t_u. That is, $[\![\varphi]\!]_{O_i} = e_i(t_u)$. Further, we have:*

$$oracle(M_p, I, \varphi) = \underset{e \in \{e_1, \dots, e_k\}}{min} e(t_u)$$

That is, the minimum value of the outputs of $M_p + M_\varphi$ at t_u is equal to the oracle value as defined by Definition 3.3.

Theorem 4.2 states that our translation of RFOL formulas into Simulink is *sound* and *complete* with respect to our notion of oracle in Definition 3.3. Note that in the case of a definite Simulink model M, the output of $M + M_\varphi$ is a single signal e. In summary, according to Theorem 4.2, $M_p + M_\varphi$ (or $M + M_\varphi$) is able to correctly compute the fitness value of φ for test input I.

Theorem 4.3. *Let M_p be a (partial) Simulink model, and let I be a test input for M_p over the time domain $\mathbb{T} = [0, t_u]$. Let φ be a requirement of M_p in RFOL. Suppose $\{\{e_1\}, \{e_2\}, \dots, \{e_k\}\} = H_p(I, M_p + M_\varphi)$ are simulation results generated for $\mathbb{T}$. Let d be the maximum constant appearing in the upper bounds of the time intervals of $\varphi_{\Uparrow}$ for existential quantifiers (i.e., time intervals in the form of $\exists t \in [\tau_1, \tau_2] : \phi$ in $\varphi_{\Uparrow}$). Each $e_i \in \{\{e_1\}, \{e_2\}, \dots, \{e_k\}\}$ is decreasing over the time interval $(d, t_u]$.*

Note that d in Theorem 4.3 indicates the time instant when all the existentially quantified time intervals of φ are terminated, and hence all the sub-formulas within the existential quantifiers of φ are evaluated. According to Theorem 4.3, the oracle output for φ becomes monotonically decreasing after d. Therefore, after d, we can stop model simulations as soon as the output of $M_p + M_\varphi$ falls below some desired threshold level. More specifically, if the output of $M_p + M_\varphi$ falls below a threshold at time $t > d$ it will remain below that threshold for any $t' \geq t$. Hence, $M_p + M_\varphi$ is able to check test outputs in an online manner and stop simulations within the time interval $(d, t_u]$ as soon as some undesired results are detected. Note that $d = 0$ if φ does not have any existential quantifier.

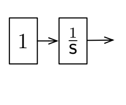
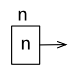
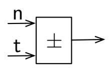
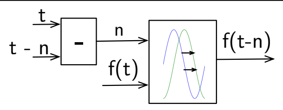
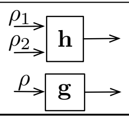
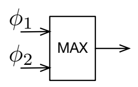
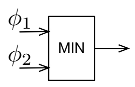
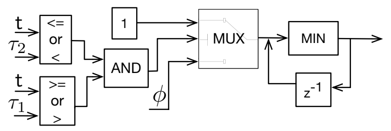
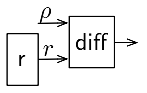
Our oracles can check Simulink models with time and magnitude-continuous signal outputs since all the blocks used in Table 4 can be executed by both fixed-step and variable-step solvers of Simulink, where the time step is decided by the same solver applied to the model under test. Finally, the running time of our oracle is linear in the size of the underlying time domain $\mathbb{T}$.

5 EVALUATION

In this section, we empirically evaluate SOCRaTEs using eleven realistic and industrial Simulink models from the CPS domain. Specifically, we aim to answer the following questions. **RQ1:** Is our requirements language (RFOL) able to capture CPS requirements in industrial settings? **RQ2:** Is the use of RFOL and our proposed translation into Simulink models likely to be practical and beneficial? **RQ3:** Is a significant amount of execution time saved when using online test oracles, as compared to offline checking?

Implementation. We implemented SOCRaTEs as an Eclipse plugin using Xtext [14] and Sirius [12] and we made it available online [1].

Table 4: Translating the SFFO formulae into Simulink Oracles.

Rule	Rule1	Rule2	Rule3	Rule4	Rule5
Formula	t	n	$t \pm n$	$f(t-n)$	$\mathbf{h}(\rho_1,\rho_2)/\mathbf{g}(\rho)$
Simulink					
Rule	Rule6	Rule7	Rule8	Rule9	
Formula	$\phi_1 \vee \phi_2$	$\phi_1 \wedge \phi_2$	$\forall t \in \langle \tau_1, \tau_2 \rangle : \phi$	$\rho \sim r$	
Simulink					

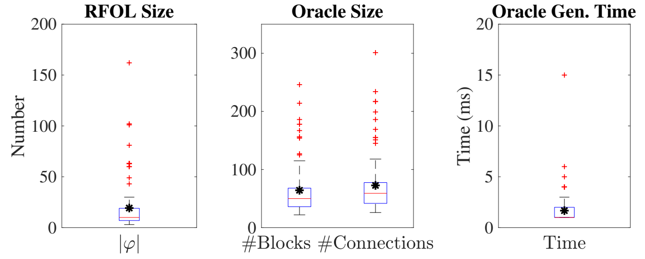


Figure 5: Plots reporting (a) the size of the RFOL formulas, (b) the number of blocks and connections of the oracle models and (c) the time took SOCRaTEs to generate the oracles.

Study Subjects. We evaluate our approach using eleven case studies listed in Table 5. We received the case studies from two industry partners: LuxSpace, a satellite system developer, and QRA Corp, a verification tool vendor to the aerospace, automotive and defense sectors. Each case study includes a Simulink model and a set of functional requirements in natural language that must be satisfied by the model. Two of our case studies, i.e., SatEx from LuxSpace and Autopilot from QRA Corp, are large-scale industrial models and respectively represent full behaviors of a satellite and an autopilot system and their environment. The other nine models capture smaller systems or sub-systems of some CPS. Our case study models implement diverse CPS functions and capture complex behaviors such as non-linear and differential equations, continuous behaviors and uncertainty. SatEx and Autopilot are continuous models. SatEx further has inputs with noise and some parameters with uncertain values. Table 5 also reports the number of blocks (#Blocks) of the Simulink models and the number of requirements (#Reqs) in our case studies. In total, our case studies include 98 requirements.

RQ1 (RFOL expressiveness). To answer this question, we manually formulated the 98 functional requirements in our case studies into the RFOL language. All of the 98 functional requirements of our eleven study subjects were expressible in RFOL without any need to alter or restrict the requirements descriptions. Further, all the syntactic constructs of RFOL described in Section 3.2 were needed to express the requirements in our study.

> The answer to RQ1 is that RFOL is sufficiently expressive to capture all the 98 CPS requirements of our industrial case studies.

RQ2 (Usefulness of the translation). Recall that engineers need to write requirements in RFOL before they can translate them into Simulink. To answer this question, we report the size of RFOL formulas used as input to our approach, the time it takes to generate online Simulink oracles and the size of the generated Simulink oracles. We measure the size of RFOL requirements as the sum of the number of quantifiers, and arithmetic and logical operators, and the size of Simulink oracles as their number of blocks and connections. Figure 5(a) shows the size of RFOL formulas ($|\varphi|$) for our case study requirements, and Figure 5(b) shows the number of blocks (#Blocks) and connections (#Connections) of the oracle Simulink models that are automatically generated by our approach. In addition, Figure 5(c) shows the time taken by our approach to generate oracle models from RFOL formulas. The total number of blocks necessary to encode all the requirements for each case study is reported in Table 5. As shown in Figure 5, it took on average 1.6ms to automatically generate oracle models with an average number of 64.2 blocks and 72.6 connections for our 98 case study requirements. Further, the average size of RFOL formulas is 19.2, showing that the pre-requisite effort to write the input RFOL formulas for our approach is not high. The difference in size between RFOL formulas and their corresponding Simulink models is mostly due to the former being particularly suitable for expressing declarative properties, such as logical properties with several nested quantifiers. Given this property, and in addition the fact that verification and test engineers are not always very familiar with Simulink — a tool dedicated to control engineers, we expect significant benefits from translating RFOL into Simulink.

> The answer to RQ2 is that, for our industrial case studies, the translation into Simulink models is practical as the time required to generate the oracles is acceptable. It takes on average 1.6ms for SOCRaTeS to generate oracle models, and the average size of the input RFOL formulas is 19.2, showing that the pre-requisite effort of our approach is manageable.

RQ3 (Impact on the execution time). Online oracles can save time by stopping test executions before their completion when they find a failure. However, by combining a model M and a test oracle (i.e., generating $M + M_{\varphi}$), the model size increases, and so does its execution time. Hence, in RQ3, we compare the time saved by online oracles versus the time overhead of running the oracles together with the models. For this question, we focus on our two large industrial models, SatEx and Autopilot, since they have long

Table 5: Important characteristics of our case study systems (from left to right): (1) name, (2) description, (3) number of blocks of the Simulink model of each case study (#Blocks), (4) number of requirements in each case study (#Reqs) and (5) total number of blocks necessary to encode the requirements (#BlReqs).

Model Name	Model Description	#Blocks	#Reqs	#BlReqs
Autopilot	A full six degree of freedom simulation of a single-engined high-wing propeller-driven airplane with autopilot.	1549	12	978
SatEx	Discussed in Section 1.	2192	8	292
Neural Network	A two-input single-output predictor neural network model with two hidden layers.	704	6	131
Tustin	A numeric model that computes integral over time.	57	5	463
Regulator	A typical PID controller.	308	10	300
Nonlinear Guidance	A non-linear guidance algorithm for an Unmanned Aerial Vehicles (UAV) to follow a moving target.	373	1	186
System Wide Integrity Monitor	A numerical algorithm that computes warning to an operator when the airspeed is approaching a boundary where an evasive fly up maneuver cannot be achieved.	164	3	169
Effector Blender	A control allocation method to calculate the optimal effector configuration for a vehicle.	95	3	391
Two Tanks	A two tanks system where a controller regulates the incoming and outgoing flows of the tanks.	498	31	1791
Finite State Machine	A finite state machine executing in real-time that turn on the autopilot mode in case of some environment hazard.	303	13	748
Euler	A mathematical model to compute 3-dimensional rotation matrices for an Inertial frame in a Euclidean space.	834	8	834

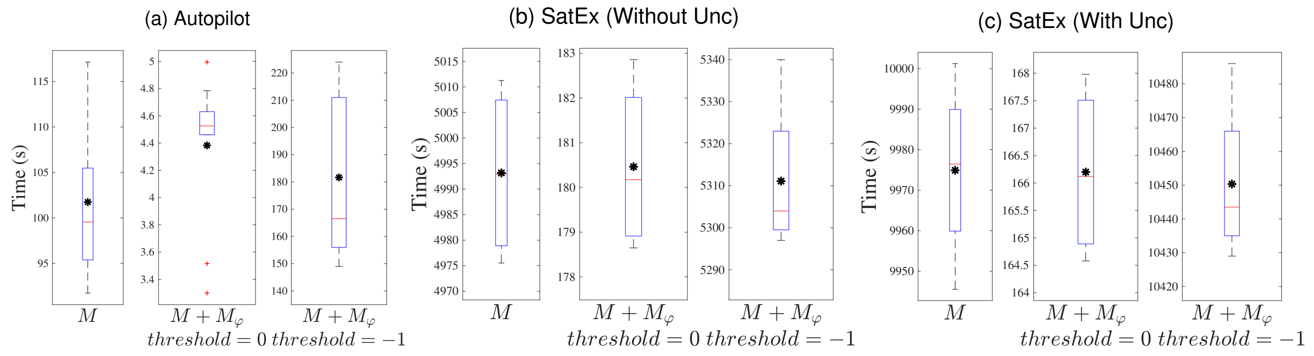


Figure 6: Test execution time on models without oracles (M), models with oracles (M + M$_\varphi$) with *threshold* = 0 and models with oracles (M + M$_\varphi$) with *threshold* = −1 for (a) Autopilot, (b) SatEx without uncertainty and (c) SatEx with uncertainty.

and time-consuming simulations while the other models in Table 5 are relatively small with simulation times less than one minute. For such models, both the time savings and the time overheads of our online oracles are practically insignificant.

During their internal testing, our partners identified some faults in SatEx and Autopilot violating some of the model requirements. We received, from our partners, 10 failing test inputs for Autopilot defined over the time domain $\mathbb{T} = [0, 4000]$, and 4 failing test inputs for SatEx defined over the time domain $\mathbb{T} = [0, 86400]$. Recall that SatEx contains some parameters with uncertain values. We also received the value range for one uncertain parameter of SatEx, i.e., ACM_type, from our partner. We then performed the following three experiments. **EXPI**: We ran all the test inputs on the models alone without including oracle models. **EXPII**: We combined SatEx and Autopilot with all the test oracle models related to their respective requirements and ran all the test inputs on the models with oracles. We did not consider any uncertainty in SatEx and set ACM_type to a fixed value. **EXPIII**: We ran all the tests on SatEx combined with all the oracle models related to its requirements and defined ACM_type as an uncertain parameter with a value range. We repeated **EXPII** and **EXPIII** for two threshold values: *threshold* = 0 where test executions are stopped when tests fail according to their boolean semantics, and *threshold* = −1 where test executions are never stopped.

Figures 6(a) and (b), respectively, show the results of **EXPI** and **EXPII** for Autopilot and SatEx. Note that box plots have different scales. Specifically, the figures show the time required to run the test inputs on Autopilot and SatEx (1) without any oracle model (M), (2) with oracle models ($M + M_\varphi$) for *threshold* = 0, and (3) with oracle models ($M + M_\varphi$) for *threshold* = −1. Specifically, in the second case, test oracles stop test executions when test cases fail, and in the third case, test oracles are executed together with the model, but do not stop test executions. Our results show that on average it takes 101.1s and 4993.2s to run tests on Autopilot and SatEx, respectively (i.e., Case M). These averages, respectively, reduce to 4.3s and 180.4s when oracles stop test executions, and they, respectively, increase to 181.6s and 5311.1s when oracles do not stop test executions. That is, for Autopilot, the average time saving of our oracles is 95.6% (≈1.5m) while their average time overhead is 78% (≈1.2m). In contrast, for SatEx, our oracles lead to an average time saving of 96% (≈80m) and an average time overhead of 6% (≈5m). We note that Autopilot is less computationally intensive. In this case, the time savings and overheads are almost equivalent because the size and complexity of the generated oracles are comparable to those of the model. SatEx, on the other hand, is more computationally intensive, and as the results show, for SatEx our oracles introduce very little time overhead but are able to save a

Table 6: Classification of the related work based on the following criteria: Is the work about oracle generation (OG)? Does the work build on the A1 and A2 assumptions (see Section 2)? Does it achieve the O1 to O4 objectives (see Section 1)? Assumption A3 is satisfied by all the related work.

Ref	OG	A1	A2	O1*	O2	O3	O4
[33, 35, 37, 43]	✓	✓	✓	✗′	✓	✓	✗
[22, 25, 49, 56], [25, 28, 56]	✓	✓	✓	✗	✓	✗	✗
[50, 66]	✓	✓	✓	✗′	✓	✗	✓
[24]	✓	✓	✓	✓	✓	✗	✗
[19, 31]	✓	✓	✓	✗′	✗	✗	✗
[60, 65]	✓	✗	✗	✗′	✗	✗	✗
[29, 32, 41, 47, 48, 51, 54, 62, 64, 67, 68]	✓	✗	✗	✗	✗	✗	✗
[70]	✗	✗	✗	✗	✗	✓	✓

* The notation ✗′ indicates that the monitoring procedure assumes a fixed sample rate, and hence does not accurately handle variable-step outputs.

great deal of time when they identify failures. Finally, we note that the time saving depends also on the presence of faults in models and whether and when test cases trigger failures. Nevertheless, according to discussions with our partners, and as evidenced by our case studies, early CPS Simulink models typically contain faults, and hence, our approach can help in saving test execution times for such models.

Figure 6(c) shows the results of **EXPIII** for running SatEx with uncertainty. Since in the case of uncertainty, a set of outputs are generated, the total test execution time increases. Specifically, it takes, on average, 9974.9s to run SatEx with uncertainty without oracles, 166.2s to run it when oracles stop test executions, and 10450.0s to run it when oracles do not stop test executions. As the results show, for SatEx with uncertainty, the time saving is even higher (i.e., 98%, ≈163m) than the case of SatEx without uncertainty, because oracles stop simulations as soon as one output among the set of generated outputs fails.

The answer to RQ3 is that, for large and computationally intensive industrial models, our oracles introduce very little time overhead (6%) but are able to save a great deal of time when they identify failures (96%). When models contain uncertainty the time saving becomes even larger and the time overhead decreases, making our online oracles more beneficial.

Data Availability. Our data and tool are available online [1]. All the models except for the SatEx model are available on request [4].

6 RELATED WORK

We classified the related work (Table 6) by analyzing whether the work addresses the oracle generation problem (**OG**)? whether it satisfies assumptions **A1** and **A2** (Section 2)? and whether it aims to achieve objectives **O1** to **O4** (Section 1)? Note that assumption **A3** is considered in all the related work included here. As shown in the table, there is no work that achieves oracle generation and satisfies all our four objectives. Below, we discuss the closest lines of work to ours among those included in Table 6.

Dokhanchi et al. [33] propose an online monitoring procedure for Metric Temporal Logic (MTL) [45] properties implemented in the S-TaLiRo tool [20]. The authors use a prediction technique to handle temporal operators that refer to future time instants compared to the shifting procedures proposed in our work. As a result, their monitoring procedure has a higher running time complexity than our oracles (i.e., polynomial in the size of time history versus linear in the time domain $\mathbb{T}$ size). Furthermore, they do not translate their monitors into Simulink, and hence, cannot benefit from the execution time speed-up of efficient Simulink blocks and the Simulink variable step solvers to handle continuous behaviors. Thus, as shown by Dokhachi et al. [33], the time overhead of their approach is considerably high as the time history grows. Jakšić et al. [43] recently developed an online monitoring procedure for STL by translating STL into automata monitors with a complexity that is exponential in the size of the formula. In contrast to our work, such monitors are not able to handle continuous signals sampled at a variable rate directly. This such signals are approximated as fixed-step signals, hence decreasing the analysis precision of continuous behaviors. To the best of our knowledge and according to a recent survey [27], the only work that, like us, translates a logic into Simulink to enable online monitoring is the work of Balsini et al. [24]. The translation, however, is given for a restricted version of STL, which for example does not allow the nesting of more than two temporal operators. As discussed in Section 3.2, RFOL subsumes STL. Hence, our translation subsumes that of Balsini et al. [24]. Breach [34, 35] is a monitoring framework for continuous and hybrid systems that translates STL into online monitors specified in C++ or MATLAB S-functions. However, due to the overhead of integrating C++ or S-functions in Simulink, running monitors in the Breach framework greatly slows down model simulations, by 4.5 times [69], making the monitors impractical for computationally expensive CPS models such as our SatEx case study. Finally, Maler et al. [50] propose a monitoring procedure that receives signal segments sequentially, checks each segment and stops simulations if a failure is detected. This work, however, is only partially online since each segment is eventually checked in an offline mode.

7 CONCLUSIONS

In this paper, we presented SOCRaTes, an automated approach to generate online test oracles in Simulink able to handle CPS Simulink models with continuous behaviors and involving uncertainties. Our oracles are generated from a signal logic-based language and compute a quantitative degree of satisfaction or failure for each test input. Our results were obtained by applying SOCRaTes to 11 industry case studies and show that (i) our requirements language is able to express all the 98 requirements of our case studies; (ii) the effort required by SOCRaTes to generate online oracles in Simulink is acceptable; and (iii) for large models, our approach dramatically reduces the test execution time compared to when test outputs are checked in an offline manner.

ACKNOWLEDGMENTS

This work has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant agreement No 694277), from QRA Corp, and from the University of Luxembourg (grant "ReACP").
We thank our partners LuxSpace and QRA Corp for their support.

REFERENCES

[1] 2019. https://github.com/SNTSVV/SOCRaTEs.
[2] 2019. LuxSpace. https://luxspace.lu/
[3] 2019. Mathworks. https://mathworks.com.
[4] 2019. Online form for accessing companion material. https://claudiomenghi.github.io/socratesForm.html.
[5] 2019. QRA Corp. https://qracorp.com/
[6] 2019. QVtrace. https://qracorp.com/qvtrace/.
[7] 2019. reactive-systems. https://www.reactive-systems.com.
[8] 2019. Robust Control Toolbox. https://nl.mathworks.com/products/robust.html.
[9] 2019. Signal To Noise Ratio. https://en.wikipedia.org/wiki/Signal-to-noise_ratio.
[10] 2019. Simulink Design Verifier. https://nl.mathworks.com/products/sldesignverifier.html.
[11] 2019. Simulink Solvers. https://nl.mathworks.com/help/simulink/ug/types-of-solvers.html.
[12] 2019. Sirius. https://www.eclipse.org/sirius/.
[13] 2019. Uncertain Real Parameters. https://nl.mathworks.com/help/robust/ug/uncertain-real-parameters.html.
[14] 2019. Xtext. https://www.eclipse.org/Xtext/.
[15] Houssam Abbas, Georgios E. Fainekos, Sriram Sankaranarayanan, Franjo Ivancic, and Aarti Gupta. 2013. Probabilistic Temporal Logic Falsification of Cyber-Physical Systems. *ACM Transactions on Embedded Computing Systems (TECS)* 12, 2s (2013), 95:1–95:30.
[16] R. Alur. 2011. Formal verification of hybrid systems. In *International Conference on Embedded Software (EMSOFT)*. 273–278.
[17] Rajeev Alur. 2015. *Principles of Cyber-Physical Systems*. MIT Press.
[18] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A Henzinger, P-H Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. 1995. The algorithmic analysis of hybrid systems. *Theoretical computer science* 138, 1 (1995), 3–34.
[19] César Andrés, Mercedes G. Merayo, and Manuel Núñez. 2012. Formal passive testing of timed systems: theory and tools. *Software Testing, Verification and Reliability* 22, 6 (2012), 365–405. https://doi.org/10.1002/stvr.1464
[20] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. 2011. S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, Parosh Aziz Abdulla and K. Rustan M. Leino (Eds.). Springer, 254–257.
[21] Kendall E Atkinson. 2008. *An introduction to numerical analysis*. John Wiley & Sons.
[22] Alexey Bakhirkin, Thomas Ferrère, Thomas A Henzinger, and Dejan Ničković. 2018. The first-order logic of signals: keynote. In *International Conference on Embedded Software*. IEEE Press, 1.
[23] Alexey Bakhirkin, Thomas Ferrère, Thomas A. Henzinger, and Dejan Ničković. 2018. The First-order Logic of Signals: Keynote. In *International Conference on Embedded Software (EMSOFT)*. IEEE, Article 1, 10 pages. http://dl.acm.org/citation.cfm?id=3283535.3283536
[24] Alessio Balsini, Marco Di Natale, Marco Celia, and Vassilios Tsachouridis. 2017. Generation of Simulink monitors for control applications from formal requirements. In *Industrial Embedded Systems (SIES)*. IEEE, 1–9.
[25] Luciano Baresi, Marcio Delamaro, and Paulo Nardi. 2017. Test oracles for simulink-like models. *Automated Software Engineering* 24, 2 (2017), 369–391.
[26] Earl T Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2015. The oracle problem in software testing: A survey. *Transactions on Software Engineering* 41, 5 (2015), 507–525.
[27] Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donzé, Georgios Fainekos, Oded Maler, Dejan Ničković, and Sriram Sankaranarayanan. 2018. Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. In *Lectures on Runtime Verification*. Springer, 135–175.
[28] Ezio Bartocci, Thomas Ferrère, Niveditha Manjunath, and Dejan Ničković. 2018. Localizing Faults in Simulink/Stateflow Models with STL. In *International Conference on Hybrid Systems: Computation and Control (HSCC)*. ACM, 197–206.
[29] D. Coppit and J. M. Haddox-Schatz. 2005. On the Use of Specification-Based Assertions as Test Oracles. In *NASA Software Engineering Workshop*. IEEE, 305–314.
[30] Eckert Claudia M de Weck Olivier, Clarkson P John, et al. 2007. A classification of uncertainty for early product and system design. *Guidelines for a Decision Support Method Adapted to NPD Processes* (2007), 159–160.
[31] L. K. Dillon and Y. S. Ramakrishna. 1996. Generating Oracles from Your Favorite Temporal Logic Specifications. In *Symposium on Foundations of Software Engineering (SIGSOFT)*. ACM.
[32] Laura K Dillon and Qing Yu. 1994. Specification and Testing of Temporal Properties of Concurrent System Designs. University of California at Santa Barbara.
[33] Adel Dokhanchi, Bardh Hoxha, and Georgios Fainekos. 2014. On-line monitoring for temporal logic robustness. In *International Conference on Runtime Verification*. Springer, 231–246.
[34] Alexandre Donzé. 2010. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification*. Springer, 167–170.
[35] Alexandre Donzé and Oded Maler. 2010. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 92–106.
[36] Sebastian Elbaum and David S Rosenblum. [n. d.]. Known unknowns: testing in the presence of uncertainty.
[37] Georgios E Fainekos and George J Pappas. 2009. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science* 410, 42 (2009), 4262–4291.
[38] Farid Golnaraghi and BC Kuo. 2010. Automatic control systems. *Complex Variables* 2 (2010), 1–1.
[39] Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2013. A comprehensive survey of trends in oracles for software testing. *University of Sheffield, Department of Computer Science, Tech. Rep. CS-13-01* (2013).
[40] Thomas A Henzinger, Peter W Kopke, Anuj Puri, and Pravin Varaiya. 1998. What's decidable about hybrid automata? *Journal of computer and system sciences* 57, 1 (1998), 94–124.
[41] Bardh Hoxha, Nikolaos Mavridis, and Georgios Fainekos. 2015. VISPEC: A graphical tool for elicitation of MTL requirements. In *Intelligent Robots and Systems (IROS)*. IEEE, 3486–3492.
[42] IEC 61508 (2010) [n. d.]. *Functional safety of electrical/electronic/programmable electronic safety-related systems* (2.0 ed.). International Standard. IEC.
[43] Stefan Jakšić, Ezio Bartocci, Radu Grosu, and Dejan Ničković. 2016. Quantitative monitoring of STL with edit distance. In *International Conference on Runtime Verification*. Springer, 201–218.
[44] Allen Kent, Albert George Holzman, and Jack Belzer. 1975. Encyclopedia of computer science and technology. (1975).
[45] Ron Koymans. 1990. Specifying real-time properties with metric temporal logic. *Real-time systems* 2, 4 (1990), 255–299.
[46] Kim G Larsen and Bent Thomsen. 1988. A modal process logic. In *Logic in Computer Science*. IEEE, 203–210.
[47] Pascale Le Gall and Agnès Arnould. 1996. Formal specifications and test: Correctness and oracle. In *Recent Trends in Data Type Specification*, Magne Haveraaen, Olaf Owe, and Ole-Johan Dahl (Eds.). Springer.
[48] Jin-Cherng Lin and Ian Ho. 2001. Generating timed test cases with oracles for real-time software. *Advances in Engineering Software* 32, 9 (2001), 705 – 715. https://doi.org/10.1016/S0965-9978(01)00021-7
[49] Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 152–166.
[50] Oded Maler and Dejan NiăźKović. 2013. Monitoring Properties of Analog and Mixed-signal Circuits. *Int. J. Softw. Tools Technol. Transf.* 15, 3 (June 2013), 247–268. https://doi.org/10.1007/s10009-012-0247-9
[51] Dino Mandrioli, Sandro Morasca, and Angelo Morzenti. 1995. Generating Test Cases for Real-time Systems from Logic Specifications. *ACM Trans. Comput. Syst.* 13, 4 (Nov. 1995), 365–398. https://doi.org/10.1145/210223.210226
[52] Reza Matinnejad, Shiva Nejati, Lionel Briand, and Thomas Bruckmann. 2018. Test Generation and Test Prioritization for Simulink Models with Dynamic Behavior. *IEEE Transactions on Software Engineering* (2018).
[53] Reza Matinnejad, Shiva Nejati, Lionel C. Briand, and Thomas Bruckmann. 2016. Automated Test Suite Generation for Time-continuous Simulink Models. In *International Conference on Software Engineering (ICSE)*. ACM.
[54] Sandro Morasca, Angelo Morzenti, and Pieluigi SanPietro. 1996. Generating Functional Test Cases In-the-large for Time-critical Systems from Logic-based Specifications. In *1996 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '96)*. ACM, New York, NY, USA, 39–52. https://doi.org/10.1145/229000.226300
[55] Paulo A Nardi and Eduardo F Damasceno. 2015. A Survey on Test Oracles. (2015).
[56] P. A. Nardi, M. E. Delamaro, and L. Baresi. 2013. Specifying automated oracles for Simulink models. In *International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, 330–333.
[57] Isaac Newton. 1774. Methodus fluxionum et seriarum infinitarum. *Opuscula mathematica, philosophica et philologica* 1 (1774).
[58] Rafael AP Oliveira, Upulee Kanewala, and Paulo A Nardi. 2014. Automated test oracles: State of the art, taxonomies, and trends. In *Advances in computers*. Vol. 95. Elsevier, 113–199.
[59] Mauro Pezze and Cheng Zhang. 2014. Automated test oracles: A survey. In *Advances in Computers*. Vol. 95. Elsevier, 1–48.
[60] Ingo Pill and Franz Wotawa. 2018. Automated generation of (F)LTL oracles for testing and debugging. *Journal of Systems and Software* 139 (2018), 124–141.
[61] André Platzer. 2018. Foundations of Cyber-Physical Systems.
[62] Debra J. Richardson, Stephanie Leif Aha, and T. Owen O'Malley. 1992. Specification-based Test Oracles for Reactive Systems. In *International Conference on Software Engineering (ICSE)*. ACM.
[63] Pritam Roy and Natarajan Shankar. 2011. SimCheck: a contract type system for Simulink. *Innovations in Systems and Software Engineering* 7, 2 (2011), 73–83.
[64] Manoranjan Satpathy, Michael Butler, Michael Leuschel, and S. Ramesh. 2007. Automatic Testing from Formal Specifications. In *Tests and Proofs*, Yuri Gurevich and Bertrand Meyer (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 95–113.

[65] Joshua Schneider, David Basin, Frederik Brix, Srđan Krstić, and Dmitriy Traytel. 2018. Scalable Online First-Order Monitoring. In *Runtime Verification*, Christian Colombo and Martin Leucker (Eds.). Springer, Cham, 353–371.

[66] Simone Silvetti, Laura Nenzi, Ezio Bartocci, and Luca Bortolussi. 2018. Signal Convolution Logic. *arXiv preprint arXiv:1806.00238* (2018).

[67] J. Srinivasan and N. Leveson. 2002. Automated testing from specifications. In *Digital Avionics Systems Conference*. IEEE, 6A2–6A2.

[68] P. A. Stocks and D. A. Carrington. 1993. Test Templates: A Specification-based Testing Framework. In *International Conference on Software Engineering (ICSE)*. IEEE.

[69] Kosuke Watanabe, Eunsuk Kang, Chung-Wei Lin, and Shinichi Shiraishi. 2018. Runtime Monitoring for Safety of Intelligent Vehicles. In *Annual Design Automation Conference (DAC)*. ACM.

[70] Jon Whittle, Pete Sawyer, Nelly Bencomo, Betty HC Cheng, and Jean-Michel Bruel. 2009. Relax: Incorporating uncertainty into the specification of self-adaptive systems. In *Requirements Engineering Conference*. IEEE, 79–88.

[71] Justyna Zander, Ina Schieferdecker, and Pieter J Mosterman. 2012. *Model-based testing for embedded systems*. CRC Press.

[72] Xi Zheng, Christine Julien, Miryung Kim, and Sarfraz Khurshid. 2017. Perceptions on the state of the art in verification and validation in cyber-physical systems. *Systems Journal* 11, 4 (2017), 2614–2627.