Conformance Testing as Falsification for Cyber-Physical Systems

Bardh Hoxha, *Member, IEEE*, Houssam Abbas, *Member, IEEE*, and Georgios Fainekos, *Member, IEEE*
Jyotirmoy V. Deshmukh, *Member, IEEE*, James Kapinski, *Member, IEEE*, and Koichi Ueda, *Member, IEEE*

***Abstract*—In Model-Based Design of Cyber-Physical Systems (CPS), it is often desirable to develop several models of varying fidelity. Models of different fidelity levels can enable mathematical analysis of the model, control synthesis, faster simulation etc. Furthermore, when (automatically or manually) transitioning from a model to its implementation on an actual computational platform, then again two different versions of the same system are being developed. In all previous cases, it is necessary to define a rigorous notion of conformance between different models and between models and their implementations. This paper argues that conformance should be a measure of distance between systems. Albeit a range of theoretical distance notions exists, a way to compute such distances for industrial size systems and models has not been proposed yet. This paper addresses exactly this problem. A universal notion of conformance as closeness between systems is rigorously defined, and evidence is presented that this implies a number of other application-dependent conformance notions. An algorithm for detecting that two systems are not conformant is then proposed, which uses existing proven tools. A method is also proposed to measure the degree of conformance between two systems. The results are demonstrated on a range of models.**

***Index Terms*—Cyber-Physical Systems, Conformance, Temporal Logic.**

I. Introduction

In a typical Model-Based Design (MBD) process for Cyber-Physical Systems (see Fig. 1), a series of models and implementations are iteratively developed such that the end product satisfies a set of functional requirements Φ. Ideally, the initial (simpler) model M_S developed should have structural properties that make it amenable to formal synthesis and verification methods [1], [2] (cycle 1 in Fig. 1) through software tools like [3], [4], [5], [6], [7], [8]. Then, the fidelity of the models is increased by modeling more complex physical phenomena ignored initially and by introducing inaccuracies due to the computational platforms such as look-up-tables, time delays, 3^{rd} party black-box components, etc. In addition, this iterative process may be extended to Processor-in-the-Loop and Hardware-in-the-Loop testing before deployment of the system [9].

The development of a higher fidelity model raises the obvious question of what is the relationship between the "simple" M_S and "complex" M_C models developed (cycle 2 in Fig. 1). If the simpler model developed was a nondeterministic model and the structure of M_C was fully known, then the answer to the question could be established through behavioral inclusions [1], i.e., is it true that every behavior of M_C can be exhibited by M_S, in response to the same stimulus?

However, in practice, non-deterministic models are rarely utilized and supported by industry tools for MBD such as LabView^{TM} or Simulink/Stateflow^{TM}. Instead, a hierarchy of deterministic models is developed each capturing a more accurate representation of the final system, and it is important to know how 'close' two successive models are to each other. While the higher fidelity model introduces new, more realistic behavior, it should still follow, roughly, the behavior of M_S. Thus, in lieu of behavioral inclusion, an appropriate notion of *distance* between the models is required, i.e., $\mathbf{dist}(M_C, M_S)$. This we call conformance between the simple and complex models. Such distance notions have been developed for various classes of systems [1], [10], [11], [12], [13] over the years. Even though works such as [10], [12] treat systems with hybrid dynamics directly, they apply only to certain classes of hybrid systems and, most importantly, they rely on the full knowledge of the mathematical model of both M_S and M_C. For industrial size CPS models, such knowledge is not always available. Another limitation is that existing distance measures for systems either consider only distances in time, e.g., [13], or in space [10], [11], [12]. For CPS, both are extremely important especially if the end goal is to verify that the deployed system (S_D) satisfies formal specifications that involve timing requirements [14], [15].

The same observations hold for the important problem of verifying whether a system S_I, which is an implementation of a model M_C, behaves approximately similar to its model M_C (arrows labeled with 3 in Fig. 1). Irrespective of whether the automatic code generation process has formal guarantees, rarely does the model M_C capture accurately all physical phenomena. Thus, the prototype system S_I will be manually modified and calibrated into a final deployment S_D. Then, the deployment S_D should have a bounded, computable distance from the model M_C under an appropriate metric, i.e., $\mathbf{dist}(M_C, S_D) \leq \varepsilon$, and, S_D should satisfy the set of specifications.

In this paper, a framework is provided to address the aforementioned gaps in MBD for CPS, i.e., arrows 2 and 3 in the V process in Fig. 1. The framework is agnostic about whether the systems studied are both models or a model and its implementation, thus we will generically refer to one system

Bardh Hoxha and Georgios Fainekos were with the department of Computer Science, Arizona State University, Tempe, AZ, 85281, USA; e-mail: {bhoxha, fainekos}@asu.edu

Houssam Abbas was with the department of Computer Science, University of Pennsylvania.

Jyotirmoy V. Deshmukh, James Kapinski, and Koichi Ueda were with Toyota Technical Center.

Manuscript received ...

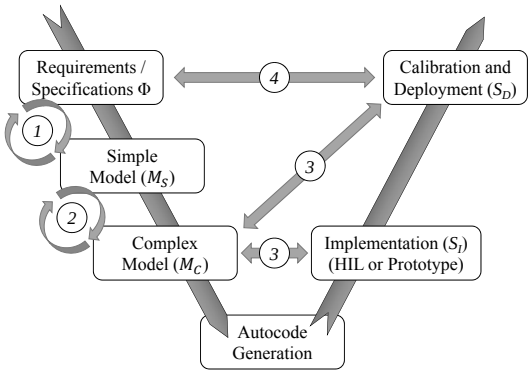


Fig. 1: Typical V process in MBD. (1) Verifying that the simple model satisfies the functional requirements; (2) Establishing a relationship between the simple and complex model; (3) Verifying conformance of implementation to the model; (4) Verifying that the end product satisfies the functional requirements.

as the Model and to the other as its Implementation.

More specifically, we utilize hybrid distance measures similar to [16], [17], [18] in order to define distances between system behaviors. Given two system behaviors (or trajectories), we compute a (τ, ε) distance between them that captures both their distances in time and in space. Then, given a bound $(\bar{\tau}, \bar{\varepsilon})$, we consider the problem of whether the Implementation conforms to its Model with degree $(\bar{\tau}, \bar{\varepsilon})$. We pose the aforementioned problem as an optimization problem which we solve using our tool S-Taliro [19], [20]. Our solution is a best effort framework and the guarantees provided are of a probabilistic nature as described for instance in [21], [22], [23].

A. Conformance testing versus specification checking

One question naturally arises at this point: why not just verify that the Implementation satisfies the same specification that the Model has been verified to satisfy? The reasoning behind the question is that if Φ is all that matters, it should be sufficient that the Implementation also satisfies it. We may answer this question as follows:

1) It is not always possible to verify formally that the Implementation satisfies the formal specification: for example, a component purchased from a third party might allow only limited observability and not lend itself to formal methods.
2) Parts of the specification are not formally expressed. For example, because the available formal tools can not handle the size of the design (e.g. reachability tools for nonlinear systems). Rather, the specification exists in plain language Test Plan documents [24] or implicitly in test suites.[1]
3) For a real-life CPS, much of the behavior is *de facto* left unspecified because of the complexity. Once triggered, a particular behavior may exhibit unspecified but undesired characteristics, even though it possesses the specified, desired, characteristics (and none of the specified, undesired characteristics).

Therefore, once we have an Implementation, it is not sufficient to check that it too conforms to the specification (if that is even possible). It is important to make sure that behaviors exhibited by Model and Implementation are close (in a sense to be defined). This then is conformance testing. This way, both Model and Implementation display similar unspecified characteristics, and our level of confidence in the Implementation derives from our confidence in the Model.

[1]Note the release of industrial tools that induce requirements from simulation traces, such as [25], in an effort to formalize requirements currently implicit in tests.

B. Summary of contributions

In the previous sections, we have argued that current ways of thinking about the relation between a Model and its Implementation are not sufficient for the verification of complex CPS. In the remainder of this paper,

1) We propose a universal definition of conformance between CPSs as a quantifiable closeness measure between the output behaviors of the two systems.
2) We argue that this universal notion implies most custom conformance notions which depend on the application.
3) We pose conformance testing as a logic property falsification problem. We then apply existing tools to this problem and show that they successsfully find non-conformant behavior.
4) We show that conformance satisfies a monotonicity property which allows us to search efficiently for the best conformance degree between two systems.
5) We perform conformance testing on several examples and a high-fidelity Toyota engine controller.

II. Problem Formulation

In Section I, it was argued that the verification of a CPS implementation in an MBD process requires conformance testing. The latter was described as checking that Model and Implementation display 'similar' behaviors, where 'similar' will be made precise. Because the objective is to detect bugs caused by the implementation process, Model and Implementation should be tested with the same inputs, and starting from the same initial conditions. Our high level goal is then to determine whether there exists a pair of (initial conditions, input signal) that causes the Model and its Implementation to produce significantly different outputs; and if such a pair exists, to find it and present it to the user as a debug guide.

To make this goal precise, this section starts by presenting the class of systems that we study. This class is illustrated with a running example of a fuel control system for an automotive application. Then, the conformance testing problem is formally stated as a search problem over the set of initial conditions and input signals. Finally, the constraints under which we seek to solve this problem are presented. This lays the groundwork for Section III, where we will mathematically define what it means for two CPSs to be conformant.

Notation. Given two sets A and B, B^A denotes the set of all functions from A to B. That is, for any $f \in B^A$ we have $f : A \to B$. Given a cartesian set product $A \times B$, pr_A is the projection onto A, i.e. for all $(a, b) \in A \times B, \mathrm{pr}_A((a, b)) = a$.

A. System model and running example

At its most general, a CPS $\mathcal{H}$ may be thought of as an input-output map. Specifically, let $N = \{1, 2, \ldots, |N|\} \subset \mathbb{N}$ be a finite set of integers, $T > 0$ be a positive real, $H_0 \subset \mathbb{R}^{n_h}$ be a set of *initial operating conditions* of the system, $U \subset \mathbb{R}^{n_u}$ be a compact set of *input values*, and let $Y \subset \mathbb{R}^{n_y}$ be a set of *output values*.

Definition 2.1: A **real-timed state sequence** (real-TSS) is a pair $(\mathbf{y}, \sigma)$ where $\mathbf{y} \in Y^{|N|}$ and $\sigma \in [0, T]^{|N|}$.
A **hybrid-timed state sequence** (hybrid-TSS) is a pair $(\mathbf{y}, \sigma)$ where $\mathbf{y} \in Y^{|N|}$ and $\sigma \in ([0, T] \times \mathbb{N})^{|N|}$.
When a statement applies to both real-timed and hybrid-timed state sequences, we will simply say 'timed state sequence' (TSS). A TSS can be the result of a sampling process or a numerical integration. Then the vector of 'timestamps' σ represents the sequence of sampling times, or times at which a numerical solution is computed. A timed state sequence will also be referred to as a *signal*, and a Y-valued timed state sequence will also be referred to as a *trajectory*. The latter is standard dynamical systems theory terminology. Note that a real-TSS may be viewed as a special hybrid-TSS $(\mathbf{y}, \sigma)$ such that $\sigma \in ([0, T] \times \{1\})^{|N|}$.

A CPS is modeled as a map between initial conditions $h_0 \in H_0$ and *input* timed state sequences $(u, \mu_{\mathfrak{u}}) \in U^{|N|} \times \mathbb{T}^{|N|} := \mathfrak{U}$ to *output* timed state sequences $(\mathbf{y}, \mu_{\mathbf{y}}) \in Y^{|N|} \times \mathbb{T}^{|N|}$, where $\mathbb{T}$ is either $[0, T]$ (for real-timed) or $[0, T] \times \mathbb{N}$ (for hybrid-timed). Note that input and output signals must either both be real-timed, or both be hybrid-timed. We model discrete states as integers, so H, U and Y could be hybrid spaces of the form $X \times L$ with $X \subset \mathbb{R}^n$ and L finite. The system $\mathcal{H}$ can then be viewed as a map:

$$\mathcal{H} : (h_0, \mathfrak{u}) \in H_0 \times \mathfrak{U} \mapsto (\mathbf{y}, \sigma) \in Y^{|N|} \times \mathbb{T}^{|N|} \tag{1}$$

We impose the following restrictions on the systems that we consider:

1) The output space Y must be equipped with a generalized metric $\mathbf{d}$. See [22] for implications.
2) For every initial condition $\eta_0 \in H_0$ and input signal $\mathfrak{u} \in \mathfrak{U}$, the system $\mathcal{H}$ produces an output signal. This is imposed to avoid modeling issues where the Model's and/or Implementation's equations have no solutions.

Further details on the necessity and implications of the aforementioned assumptions can be found in [22].

As it is standard in systems theory, the system's output can be expressed as a function of its internal state $\eta \in H \supset H_0$:

$$\eta \in H \mapsto y = g(\eta) \in Y$$

Here, H is the state-space of the system. We do not always assume that the internal state is observable. Given a real-timed state sequence $(\mathbf{y}, \mu)$, its i^{th} element is denoted $(\mathbf{y}, \mu)_i = (\mathbf{y}_i, \mu_i) \in Y \times \mathbb{T}$. Similarly, given a hybrid-timed state sequence $(\mathbf{y}, \sigma)$, the i^{th} element $(\mathbf{y}, \sigma)_i$ is denoted $(\mathbf{y}_i, \sigma_i)$, with $\sigma_i = (t, j) \in [0, T] \times \mathbb{N}$.

Example 1: We consider a fuel control (FC) system for an automotive application. Environmental concerns and government legislation require that the fuel economy be maximized and the rate of emissions (e.g., hydrocarbons, carbon monoxide, and nitrogen oxides) be minimized. Control of automobile engine air-to-fuel (A/F) ratio is crucial to optimize fuel economy and to minimize emissions. Ideal A/F levels are given by the *stoichiometric* value, which is the optimal A/F ratio to minimize both fuel consumption and emission of pollutants. The purpose of the FC system is to maintain the ratio of air-to-fuel (A/F) within a given range of the stoichiometric value.

The scenario that we model involves an engine connected to a dynamometer, which is a device that can control the speed of the engine and measure the output torque. For our experiment, the dynamometer maintains the engine at a constant rotational velocity, as the engine is tested. There is only one input to the model: the throttle position command from the driver.

The conformance testing scenario for this example is unique, in that the Model was derived from the Implementation, for reasons on which we will now elaborate. The Implementation was derived from a textbook model of an engine control system [26], and contains implementation details such as look-up-tables (LUTs). The Model was then abstracted from this Implementation for the purposes of formal analysis [27].

Despite the counter-intuitive relationship between the Model and Implementation for this case, the conformance task remains: to verify that these two versions satisfy some similarity criterion. △

The discussion and results in this paper apply to this input-output map model of a CPS. To define some of the conformance notions in this paper, it will be useful to sometimes work with the more specialized *hybrid automaton* model of a CPS [16]: broadly speaking, a hybrid automaton has countably many modes $\{\ell_1, \ell_2, \ldots\} := L$, with possibly different dynamics F_ℓ active in each mode: $\dot{\eta} = F_\ell(\eta, u)$. The automaton switches (or 'jumps') between modes whenever the internal state η enters specific subsets $G \subset H$ of the state space, called *switching guards*. In general, a switching guard might depend on time and on the current state; different jumps $\ell \to \ell'$ will have different guards: $G = G(e, t, \eta)$, $e = (\ell, \ell')$. Finally, when the system switches modes, the internal state might be reset to a switch-specific value: $\eta^+ = Re(\eta, e)$ if $\eta \in G(e, t, \eta)$. If we explicitly model the system mode as part of the internal state $\eta = (x, \ell) \in X \times L$, we may write the automaton's equations as [28]

$$\mathcal{H} \begin{cases} (\dot{x}, \dot{\ell}) & = (F_\ell(x, u), 0) & (x, u) \in C \times U \\ (x^+, \ell^+) & = Re(e, x) & (x, u) \in D \times U \\ y & = g(\eta) & \end{cases} \tag{2}$$

where $C \subset X$ is the 'flow set' of continuous evolution, and D is the jump set, which equals the union of all guard sets. Apart from the requirement that the dynamics have at least one solution for every $(\eta_0, \mathfrak{u})$, they are arbitrary.

Remark 2.1: The notion of a system mode applies to the general input-output model of a system, so in what follows we will often be referring to the 'mode' of the CPS without necessarily requiring that it be modeled as a hybrid automaton. For example, a powertrain Implementation might be outputting the current gear, or the mode of operation e.g. Economy vs. Sport.

The trajectories (or 'solutions') of purely continuous dynamical systems (with only one mode) are parameterized by the time variable t, and those of purely discrete dynamical systems (with no continuous evolutions) are parametrized by the number of discrete jumps j. Following Goebel and Teel [29], the trajectories to hybrid automata are parametrized by both t and j, to reflect that both evolution mechanisms are present. So we write $\eta(t,j)$ for the state and $\mathbf{y}(t,j)$ for the output of the automaton at time t and after j jumps, or mode switches. Because jumps take 0 time, it is possible to have the automaton go through several states in 0 time: $\eta(t,j) \to \eta(t,j+1) \to \eta(t,j+2)\ldots$. This can't happen in a physical Implementation, but it may be allowed in the Model. We refer the reader to [29] for exact definitions of discrete and hybrid time domains, arcs and trajectories.

We now introduce the behavior of a system which is applicable to both input-output maps and hybrid automata.

Definition 2.2 (Behavior): Take a system $\mathcal{H}$, an initial point $h_0 \in H$ and input signal $\mathfrak{u}$. The behavior of the CPS $\mathcal{H}$ from h_0 and $\mathfrak{u}$, denoted $\mathcal{B}_{\mathcal{H}}(h_0,\mathfrak{u})$, consists of

- $h_0 = (x_0, \ell_0)$
- the output trajectory $\mathbf{y}_{\mathcal{H}}(h_0,\mathfrak{u})$ generated by $\mathcal{H}$ in response to $(h_0,\mathfrak{u})$

The behavior of $\mathcal{H}$ is then

$$\mathcal{B}_{\mathcal{H}} = \{(h, \mathbf{y}_{\mathcal{H}}(h,\mathfrak{u})) \mid h \in H_0, \mathfrak{u} \in \mathfrak{U}\}$$

Example 2 (Example 1 Continued): For the FC, the outputs consist of the normalized air-to-fuel ratio λ and the fuel commanded into the Cylinder-and-Exhaust. Thus $Y = \mathbb{R}^2_+$. The presence of a switch in the Throttle block, and an LUT in the Cylinder-and-Exhaust block, induces 8 modes so $L = \{1,\ldots,8\}$. The outputs of the FC are sampled at a fixed rate. The output signals can be modeled as real-TSS. If we could observe the mode changes during a simulation, then we can use a counter j to count the mode switches, or 'jumps', and model the output as a hybrid-TSS. E.g. the following sequence of sampled (λ, mode)

$$(1,\ell_1),(0.99,\ell_1),(0.98,\ell_2),(0.87,\ell_2),(0.88,\ell_1)$$

is interpreted as the following hybrid-TSS

$$(\mathbf{y},\sigma) = ((1,0.99,0.98,0.87,0.88),\\ ((0,0),(T/|N|,0),(2T/|N|,1),(3T/|N|,1),(3T/|N|,2))$$

Note that j counts the jumps so far, but does not indicate what mode the system is in.

B. Conformance testing

Based on the preceding discussion, we adopt the premise that conformance is a relation of similarity between the behaviors of two systems when subjected to the same stimulus. The behavior is defined in Def. 2.2. So we may speak of conformant (i.e., similar) behaviors, or conformant output trajectories. Conformance testing can then be formulated as a search problem: find a pair of trajectories, generated by the two CPSs in response to the same initial condition and input, that are not conformant. The search for a non-conformant pair of trajectories is called *falsification*.

Problem 1 (Conformance testing): Let $\mathcal{H}_M$ and $\mathcal{H}_I$ be a Model and Implementation of a CPS, respectively. Find a pair $\theta = (\eta,\mathfrak{u}) \in H \times \mathfrak{U}$ such that $\mathbf{y}_{\mathcal{H}_M}(\theta)$ and $\mathbf{y}_{\mathcal{H}_I}(\theta)$ are non-conformant.

Because Implementations typically have limited observability, we assume testing happens under the following restriction:

Assumption 2.1 (Black box testing): The behaviors of Model and Implementation are observable: i.e. it is always possible, for either system, to obtain an element of the behavior by executing the system. Only the behavior of the Implementation is observable - i.e. we know nothing else about it.

In particular, the sequence of modes that Model and Implementation go through can be an important variable to track to decide whether the two systems conform. As an example, a silicon microchip has 'scan chains', which are chains of buffers that pass to the outside world the values of internal registers. These are only used during testing, and are burnt before customer delivery. In control systems, mode estimation [30] could be used when applicable. While we leave it possible that more is known about the Model, we won't need to know more to apply the methods of this paper. More knowledge of the Model will make applicable grey box testing methods such as [31], [32].

III. Defining conformance

In general, *conformance is an application-dependent notion* to help determine that the implementation process does not use components or methods that alter the functionality (or safety or performance) of the final product in any significant manner. What 'significant' means will, naturally, depend on the application. This makes conformance testing itself application-dependent. Our first contribution is made in this section: we present a notion of distance between output trajectories, called $(T,J,(\tau,\varepsilon))$-closeness, and argue that this is an appropriate *universal* notion of conformance; that is, it is generally applicable regardless of the underlying application. The price we pay for this universality is that this notion is stronger than the application-dependent ones: two systems may not be conformant according to $(T,J,(\tau,\varepsilon))$-closeness, but they may be conformant according to a weaker custom notion which is sufficient for the task at hand. In the second part of this section, we give real-life examples where the application-dependent conformance turns out to be implied by $(T,J,(\tau,\varepsilon))$-closeness.

Thus we may develop a general theory of conformance based on $(T,J,(\tau,\varepsilon))$-closeness, and 'generic' algorithms that decide conformance which do not depend on the application. This is advantageous for two reasons: one of the challenges today for testing of hybrid systems (and CPS in general) is to define conformance in a rigorous manner, and $(T,J,(\tau,\varepsilon))$-closeness provides an answer. Secondly, generic conformance tools can be used early in the design cycle, before the instrumentation is all there for a deeper analysis of the difference between Model and Implementation. Moreover, a feature of the universal notion is that it uses only the outputs of the system (and possibly the mode sequence if available). Thus, the analysis and methods herein are applicable to potentially

complicated systems with very general system models, including the input-output map model in Section II-A.

A. A universal conformance notion

The proposed universal notion of conformance is $(T, J, (\tau, \varepsilon))$-closeness. $(T, J, (\tau, \varepsilon))$-closeness expresses proximity between the outputs, their time sequences (real-TSS and hybrid-TSS), and their modes if applicable. It is derived from [29].

Definition 3.1 ($(T, J, (\tau, \varepsilon))$-closeness): Take a test duration $T > 0$, a maximum number of jumps $J \in \mathbb{N}$, and parameters $\tau, \varepsilon > 0$. Two timed state sequences, or *trajectories*, $(\mathbf{y}, \sigma)$ and $(\mathbf{y}', \sigma')$ are $(T, J, (\tau, \varepsilon))$-close if
(a) for all $i \in N$ such that $\sigma_i = (t, j)$ satisfies $t \leq T, j \leq J$, there exists $k \in N$ such that $\sigma'_k = (s, j)$, $|t - s| < \tau$, and

$$\|\mathbf{y}_i - \mathbf{y}'_k\| < \varepsilon$$

(b) for all $i \in N$ such that $\sigma'_i = (t, j)$ satisfies $t \leq T, j \leq J$, there exists $k \in N$ such that $\sigma_k = (s, j)$, $|t - s| < \tau$, and

$$\|\mathbf{y}'_i - \mathbf{y}_k\| < \varepsilon$$

We will also say that $\mathbf{y}_1$ and $\mathbf{y}_2$ are **conformant** with degree $(T, J, (\tau, \varepsilon))$.
When T and J are clear from the context, we simply say (τ, ε)-close. Because a real-TSS is a special case of a hybrid-TSS, the above definition applies to both.

$(T, J, (\tau, \varepsilon))$-closeness may be tought of as giving a proximity measure between the two hybrid arcs, both in time and space. The definition says that within any time window of size 2τ, there must be a time when the trajectories are within ε or less of each other. Allowing some 'wiggle room' in both time and space is important for conformance testing: when implementing a Model, there are inevitable errors. These are due to differences in computation precision, clock drift in the implementation, the use of inexpensive components, unmodeled environmental conditions, etc, leading to the Implementation's output to differ in value from the Model's output, and to have different timing characteristics. Thus $(T, J, (\tau, \varepsilon))$-closeness captures nicely the intuitive notion that 'the outputs should still look alike'. Our definition of $(T, J, (\tau, \varepsilon))$-closeness differs slightly from the original definition in [29] in that we use two 'precision' parameters τ and ε instead of one. In practice, using only one precision parameter is too restrictive, since the outputs can have a different order of magnitude from the time variable. It can be verified that the HIOCO relation of Van Osch [33] is an exact version of $(T, J, (\tau, \varepsilon))$-closeness $(\tau = \varepsilon = 0)$, with the role of inputs and outputs explicitly differentiated.

Remark 3.1: If it is not possible to observe the number of jumps j, then we simplify the above definition by assuming that j is always equal to 1. In other words, we interpret the definition over real-TSS and assume the system only has one mode.

Definition 3.2: Take a test duration $T > 0$, a maximum number of jumps $J \in \mathbb{N}$, and parameters $\tau, \varepsilon > 0$. Two CPSs $\mathcal{H}_M$ and $\mathcal{H}_I$ are said to be $(T, J, (\tau, \varepsilon))$-close if for any initial condition $h_0 \in H_0$ and input signal $\mathfrak{u} \in \mathfrak{U}$, the trajectories $\mathbf{y}_{\mathcal{H}_M}(h_0, \mathfrak{u})$ and $\mathbf{y}_{\mathcal{H}_I}(h_0, \mathfrak{u})$ are $(T, J, (\tau, \varepsilon))$-close. The two systems are also said to be conformant with degree $(T, J, (\tau, \varepsilon))$.

Remark 3.2: A Model and Implementation generally won't have the same state-space, and so won't accept the same initial conditions. So when we provide the same initial condition h_0 to both, one of them might use a projection of h_0 or a more general mapping $f(h_0)$ to obtain its appropriate initial conditions.

From a conformance perspective, it is preferable to have a smaller ε and a smaller τ. We use this to define a partial order on the (τ, ε) pairs.

Definition 3.3: The partial order relation $\preceq$ over (τ, ε) pairs is given by $(\tau, \varepsilon) \preceq (\tau', \varepsilon')$ if and only if $\tau \leq \tau'$ and $\varepsilon \leq \varepsilon'$. The inequality is strict if and only if at least one of the component-wise inequalities is strict.

Remark 3.3: $(T, J, (\tau, \varepsilon))$-closeness has the valuable advantage of being monotonic: if two trajectories are $(T, J, (\tau, \varepsilon))$-close, then they are $(T, J, (\tau', \varepsilon'))$-close for any $(\tau', \varepsilon') \succeq (\tau, \varepsilon)$. This allows us to use a simple binary search for a smallest (τ, ε) pair such that the trajectories, and the systems, are $(T, J, (\tau, \varepsilon))$-close. We make use of this property in the experiments.

B. Examples

We conclude this section with examples where application-specific notions of conformance are implied by $(T, J, (\tau, \varepsilon))$-closeness. Thus if we find trajectory pairs $(\eta_{\mathcal{H}_M}, \eta_{\mathcal{H}_I})$ that violate the latter, they automatically violate the former.

Example 3 (Example 1 continued): Because the look-up-tables (LUTs) in the Implementation $\mathcal{H}_I$ are replaced by polynomials in the Model $\mathcal{H}_M$, some error is expected between the outputs of the two systems. The designer hopes, however, that the error at the output of the Implementation, is in the same order of magnitude as the error between the outputs of the LUTs and the outputs of the corresponding polynomials. If not, then more entries are needed in the LUT. Moreover because LUT look-ups are typically faster than polynomial computations, some delay between the two outputs is expected to be observed. The designer has a pre-specified maximum acceptable delay. In this case, conformance imposes upper bounds on the spatial and temporal differences between the outputs of Model and Implemenation.

Conformance testing is applicable to application domain areas other than the automotive industry. E.g. in the microchip design cycle, as shown in the following example.

Example 4 (State retention): $\mathcal{H}_M$ is an RTL description of an electrical circuit, and $\mathcal{H}_I$ is equal to $\mathcal{H}_M$ with power gating and state retention added to some of its subsystems. With state retention, the contents of certain critical memory elements of the power-gated subsystem are retained in 'shadow' registers prior to power-down, and restored after power-up. This creates a temporary difference between the state of the non-state retained circuit ($\mathcal{H}_M$) and the state-retained circuit ($\mathcal{H}_I$). This difference lasts until the reset sequence is completed. Thus in this case, conformance means that a temporary difference

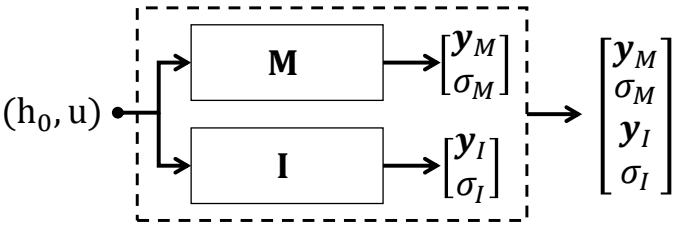


Fig. 2: Parallel interconnection of Model and Implementation.

in modes between the two systems is allowed, but they must re-converge after a pre-defined amount of time.

IV. Solution approach

In this section, we present a general method for determining whether two systems are conformant or not. We also provide a way to quantify the degree of conformance between them.

A. Conformance as falsification

Our approach is based on the observation that $(T, J, (\tau, \varepsilon))$-closeness can be expressed as a formal logical property defined over the output timed state sequences of the parallel interconnection of systems $\mathcal{H}_M$ and $\mathcal{H}_I$. See Fig. 2. A TSS of the interconnection system $\mathcal{H}_{||}$ is just the concatenation of the TSS of the component systems: $(\mathbf{y}_{||}, \sigma_{||}) = ((\mathbf{y}_M, \sigma_M), (\mathbf{y}_I, \sigma_I))$. If we can find a (parallel) TSS (or 'trajectory') $(\mathbf{y}_{||}, \sigma_{||})$ which falsifies the $(T, J, (\tau, \varepsilon))$-closeness property, then by definition, the component trajectories are non-conformant, and by extension, the systems $\mathcal{H}_M$ and $\mathcal{H}_I$ are non-conformant. In what follows, we will use the terms 'falsifying trajectory pairs' and 'non-conformant trajectory pairs' interchangeably.

The logic we use to express $(T, J, (\tau, \varepsilon))$-closeness is Metric Temporal Logic (MTL) [14] (see the appendix in the extended version of the paper [xxx] for a review of MTL). We first present the following construction for real-TSS, then generalize it to hybrid-TSS.

Real-TSS: Fix $\tau > 0$, $\varepsilon > 0$. Our goal is to express (τ, ε)-closeness as an MTL formula. Let $(\mathbf{y}_M, \mu_M)$ and $(\mathbf{y}_I, \mu_I)$ be the outputs of Model and Implementation CPSs, respectively, in response to the same initial conditions and input signal. Because (τ, ε)-closeness requires comparing the current value of $\mathbf{y}_M$ to current, past and future values of $\mathbf{y}_I$ (over a window of width 2τ), we will create shifted versions of $\mathbf{y}_I$. Given the symmetry of (τ, ε)-closeness, we will also define shifted versions of $\mathbf{y}_M$. The amount of the shift will depend on τ: how many samples of $\mathbf{y}_I$ ($\mathbf{y}_M$) fit within a window of width 2τ?

The shifted versions are now defined. Recall that $\mathbf{y}_{M,i}$ is the i^{th} sample in the TSS $\mathbf{y}_M$, and similarly for $\mathbf{y}_{I,i}$. Consider the Model's output: for each $i \in N$, compute the largest $k \geq i$ such that $|\mu_{M,i} - \mu_{M,k}| < \tau$. Define $m(\tau, i) = k - i$: this is the number of samples in the largest window of duration less than τ starting at μ_i. Similarly, we compute $n(\tau, i)$ for the Implementation TSS for every $i \in N$. The numbers n and m could in general vary with i due to an adaptive sampling period. Define $m(\tau) = \min\{m(\tau, i), i \in N\}$. $m(\tau)$ is the smallest number of samples in a window of size less than τ anywhere in $(\mathbf{y}_M, \mu_M)$. Assuming that $|\mu_M| > 1$, it comes that $m(\tau) < \infty$.[2] This constitues the size of the shift (forward and backward) to apply to $(\mathbf{y}_M, \mu_M)$. Similarly, define $n(\tau)$ for the Implementation. We may now define shifted versions of the output trajectories via the discrete shift operator: for $k \in \mathbb{Z}$, $\mathcal{S}_k(\mathbf{y}_M, \mu_M) := (\mathcal{S}_k \mathbf{y}_M, \mathcal{S}_k \mu_M)$, with

$$\mathcal{S}_k \mathbf{y} = (\mathbf{y}_{k+1}, \mathbf{y}_{k+2}, \ldots, \mathbf{y}_{|N|}, \underbrace{\mathbf{y}_{|N|}, \ldots, \mathbf{y}_{|N|}}_{\text{k times}}) \quad (3)$$

$$\mathcal{S}_k \mu = (\mu_{k+1}, \mu_{k+2}, \ldots, \mu_{|N|}, \mu_{|N|} + \frac{T}{|N|}, \ldots, \mu_{|N|} + \frac{kT}{|N|})$$

when $k > 0$, and

$$\mathcal{S}_k \mathbf{y} = (\underbrace{\mathbf{y}_1, \ldots, \mathbf{y}_1}_{\text{k times}}, \mathbf{y}_1, \mathbf{y}_{k+2}, \ldots, \mathbf{y}_{|N|-k}) \quad (4)$$
$$\mathcal{S}_k \mu = \mu$$

when $k < 0$. Note that the filler values at both ends of the shifted sequences $\mathcal{S}_k \mathbf{y}$ (3),(4) are obtained by constant interpolation.

Recall Def. 3.1(a). This condition can be captured by saying that at all i, there exists a $k \in \{-n(\tau), \ldots, n(\tau)\}$ such that $\|\mathbf{y}_{M,i} - (\mathcal{S}_k \mathbf{y}_I)_i\| < \varepsilon$. Analogously for Def. 3.1(b).

Now $(T, J, (\tau, \varepsilon))$-closeness may be expressed as the following MTL formula $\varphi_{(\tau,\varepsilon)}$. Here, $\vee$ is the logical OR operator, $\wedge$ is the logical AND operator, and $\Box_{\mathcal{I}}$ is the temporal 'Always over the time interval $\mathcal{I}$' operator.

$$p_1(\tau, \varepsilon) = \bigvee_{k=-n(\tau)}^{n(\tau)} \|\mathbf{y}_{M,i} - (\mathcal{S}_k \mathbf{y}_I)_i\| < \varepsilon \quad (5)$$

$$p_2(\tau, \varepsilon) = \bigvee_{k=-m(\tau)}^{m(\tau)} \|\mathbf{y}_{I,i} - (\mathcal{S}_k \mathbf{y}_M)_i\| < \varepsilon \quad (6)$$

$$\varphi_{(\tau,\varepsilon)} := \Box_{[0,T]}(p_1(\tau, \varepsilon) \wedge p_2(\tau, \varepsilon)) \quad (7)$$

Because $(T, J, (\tau, \varepsilon))$-closeness only requires that the two signals be within ε of each other at least once in a window of size 2τ, p_1 and p_2 use disjunction: it is sufficient for one shifted comparison to be less than ε.

Hybrid-TSS: To define the MTL formula over hybrid-TSS, we must break up each trajectory into segments, such that there are no jumps within a segment. Specifically, consider the hybrid-TSS $(\mathbf{y}, \sigma)$, with

$$\sigma = ((t_1, j_1), (t_2, j_2), \ldots, (t_{|N|}, j_{|N|}))$$

Assume that there are only G unique values of j that appear in σ, corresponding to $G - 1$ jumps. We divide the hybrid-TSS into G segments $g_1, \ldots, g_G$, such that j is constant over a segment. Each segment can be viewed as a real-TSS. If we apply this procedure to $(\mathbf{y}_M, \sigma_M)$ and $(\mathbf{y}_I, \sigma_I)$, we get G_M Model segments $\{g_i^M\}_{i=1}^{G_M}$ and G_I Implementation segments $\{g_i^I\}_{i=1}^{G_I}$. Let $G = \max\{G_M, G_I\}$. We can now apply the above procedure to every pair (g_i^M, g_i^I), with the important

[2]The case where $m(\tau) = \infty$ occurs when the Model trajectory $\mathbf{y}_M$ is Zeno: i.e., when it contains an infinite number of samples without advancing time. This can result from a modeling artifact [34]. The condition $|\mu_M| > 1$ effectively says we have at least two different timesteps, and so the trajectory is not initially Zeno.

difference that the shifted sequences $\mathcal{S}_k\mathbf{y}$ (3),(4) are filled with an arbitrarily large value (or $+\infty$), and not by constant interpolation. This is to reflect that a comparison past the jump point is not valid. This results in G formulae $\varphi^i_{(\tau,\varepsilon)}$ obtained via (7). The complete formula can now be written

$$\Phi_{(\tau,\varepsilon)} = \bigwedge_i \varphi^i_{(\tau,\varepsilon)} \tag{8}$$

Note there are other ways of defining the MTL formula for hybrid-TSS that directly incorporate the jump counter in the formula. Comparing these different methods is outside the scope of this paper. Unless otherwise indicated, all the discussion that follows applies equally to the formula obtained via (7) (for real-TSS) or (8) (for hybrid-TSS).

We can now use existing tools, like S-TaLiRo [19], [20], to find a pair of trajectories (equivalently, a trajectory of the parallel interconnection) which falsify $\varphi_{(\tau,\varepsilon)}$. S-TaLiRo uses, among others, Simulated Annealing (SA) to find falsifying trajectories. If such a trajectory is not found, convergence properties of SA imply that with probability approaching 1, the property is satisfied by the systems; equivalently, that the two systems are indeed conformant.

We should stress at this point that the proposed method is not specific to $(T, J, (\tau, \varepsilon))$-closeness. It is more widely applicable to any application-dependent conformance notion that can be expressed as an MTL formula, including those from the examples in Section III. For example, for the case when mode sequences are allowed to diverge for at most a pre-defined duration D (Example 4), the conformance relation is expressed as: "For every initial condition $h_0 \in H_0$ and every input signal $\mathfrak{u} \in \mathfrak{U}$, whenever the two systems are in different modes, they will be back in the same mode within D sec". This can now be written as the MTL formula:

$$\varphi_{PWC} := \Box_{[0,T-D]}(\ell(t) \neq \ell'(t) \Rightarrow \Diamond_{[0,D]}\ell(t) = \ell'(t)) \tag{9}$$

We conclude this section with a word on how to practically falsify $\varphi_{(\tau,\varepsilon)}$ (or any of the other application-dependent notions). A method that has proved efficient is to minimize the *robustness* of the trajectories w.r.t the MTL property. In this work, we use spatial robustness [22], [35] and time robustness [36]. Spatial robustness measures how far *in the output space* a given trajectory is from the nearest trajectory with opposite truth value for φ.[3] The spatial robustness of trajectory $(\mathbf{y}, \mu)$ starting at time t w.r.t. formula φ is denoted as follows

$$[\![\varphi]\!]((\mathbf{y}, \mu), t) = r \in \mathbb{R} \cup \{\pm\infty\}$$

Computing r is done on the output trajectory without any reference to the system that generated it.

Time robustness measures by how much to shift the given trajectory *in time*, to change its truth value w.r.t. φ. Two time robustness values may be measured for each trajectory: the future robustness θ^+ and the past robustness θ^-, depending on whether the signal is shifted left (so future values are introduced) or right (so past values are introduced). In this work we explicitly denote time robustness by

$$[\![\varphi]\!]_\theta((\mathbf{y}, \mu), t) = \min\{\theta^-, \theta^+\} \in \mathbb{R} \cup \{\pm\infty\}$$

The spatial [35] and temporal [36] robust semantics of MTL formulae are reviewed in the appendix of the extended version of this paper [xxx].

Both types of robustness (spatial and temporal) satisfy the fundamental theorem that a negative robustness value indicates falsification, a positive value indicates satisfaction, and a value of 0 indicates that an infinitesimal change in the trajectory (in space or in time) will change its truth value. Therefore, the search for a falsifying trajectory $\mathbf{y}_{||}$ can be re-cast as the problem of minimizing $[\![\varphi]\!](\mathbf{y}_{||}, 0)$ over $H_0 \times \mathfrak{U}^{|N|}$. To make this a finite-dimensional optimization, the input signals are parameterized with a finite number of parameters. (This parametrization effectively limits the search space, and the global minimum returned by falsification is a minimum over this limited space. But the parametrization can typically be made as precise as desired, e.g. to within the approximation error of the minimization algorithm). As our objective is to find falsifying trajectories, we stop the search as soon as it encounters a trajectory with negative robustness.

Now it is possible to create an example which displays a (graphically) convergent sequence of trajectories $(\mathbf{y}_{||,i}, \mu_{||,i}) \to (\mathbf{y}_{||}, \mu_{||})$ such that $[\![\varphi_{(\tau,\varepsilon)}]\!]((\mathbf{y}_{||,i}, \mu_{||,i}), 0)$ does not converge to $[\![\varphi_{(\tau,\varepsilon)}]\!]((\mathbf{y}_{||}, \mu_{||}), 0)$. This holds true for both spatial and temporal robustness. So even if Model and Implementation are not conformant (for a given value of $(T, J, (\tau, \varepsilon))$), local optimization algorithms can get trapped in local minima with positive robustness. On the other hand, non-conformant trajectory pairs will necessarily have negative robustness, so that if a Model/Implementation pair is non-conformant, all global minima of the robustness are negative, and correspond to non-conformant pairs of trajectories. Thus we need to use global optimizers, like Simulated Annealing, Cross-Entropy [37] or other methods supported by [19].

B. Degree of conformance

In addition to verifying whether two systems are (τ, ε)-close for a given (τ, ε), we may find a smallest such pair with the order defined in Def.3.3. Recall now that $\varphi_{(\tau,\varepsilon)}$ is monotonic in (τ, ε) (remark 3.3). The following theorem shows that the robustness values are also monotonic in the parameters τ, ε. The proof is in the appendix of the extended version of this paper [xxx].

Theorem 4.1: Take two TSS $(\mathbf{y}, \sigma)$ and $(\mathbf{y}', \sigma')$, a test duration T, a number of jumps J, and a time $t \leq T$. Consider the parallel concatenation

$$(\mathbf{y}_{||}, \sigma_{||}) = ((\mathbf{y}, \sigma), (\mathbf{y}', \sigma'))$$

(i) Fix $\tau > 0$. If $0 < \varepsilon_1 \leq \varepsilon_2$, then

$$[\![\varphi_{(\tau,\varepsilon_1)}]\!]((\mathbf{y}_{||}, \sigma_{||}), t) \leq [\![\varphi_{(\tau,\varepsilon_2)}]\!]((\mathbf{y}_{||}, \sigma_{||}), t)$$

(ii) Now fix $\varepsilon > 0$. If $0 < \tau_1 \leq \tau_2$, then

$$[\![\varphi_{(\tau_1,\varepsilon)}]\!]_\theta((\mathbf{y}_{||}, \sigma_{||}), t) \leq [\![\varphi_{(\tau_2,\varepsilon)}]\!]_\theta((\mathbf{y}_{||}, \sigma_{||}), t)$$

[3] If the mode is observable, spatial robustness also computes the (quasi-) distance between the modes of the two trajectories [22], but we don't make use of this here.

Therefore, we can combine S-TaLiRo with a binary search over the values of τ and ε to find a smallest pair such that $\varphi_{(\tau,\varepsilon)}$ is satisfied. Because the order on (τ, ε) pairs is only partial, binary search is applied to each component while fixing the other, thus exploring the Pareto-optimal front (e.g. [38]). Algorithm 1 shows the binary search for the smallest ε given a τ. A search over τ can be done similarly with obvious modifications. The initial ε_h can be found by using an initial binary search that doubles some ε_0 until $[\![\varphi_{(\tau,\varepsilon)}]\!] > 0$.

Algorithm 1 Searching for a smallest ε given τ.

Require: Number of iterations K, parameter $\tau > 0$, low value $\varepsilon_l = 0$, high value $\varepsilon_h > 0$ such that $[\![\varphi_{(\tau,\varepsilon_h)}]\!] > 0$.
  for $i = 0$ to $K - 1$ **do**
    $\varepsilon = 0.5 * (\varepsilon_h + \varepsilon_l)$
    Run S-TaLiRo to falsify $\varphi_{(\tau,\varepsilon)}$.
    if $([\![\varphi_{(\tau,\varepsilon)}]\!] < 0)$ **then**
      $\varepsilon_l = \varepsilon$
    else
      $\varepsilon_h = \varepsilon$;
    end if
  end for
    return $[\varepsilon_l, \varepsilon_h]$

The value $(\bar{\tau}, \bar{\varepsilon})$ returned by this procedure gives a *quantitative measure of conformance between the two systems*, and allows the designer to make informed trade-offs between, say, output accuracy of the Implementation, and its timing characteristics.

Remark 4.1: For a given τ, the smallest ε such that two trajectories $(\mathbf{y}, \sigma)$ and $(\mathbf{y}', \sigma')$ are (τ, ε)-close can be calculated as

$$\varepsilon_M(\tau) = \min_{i \in N} \max_{|\sigma'_k - \sigma_i| < \tau} \|\mathbf{y}_i - \mathbf{y}'_k\| \quad (10)$$

$$\varepsilon_I(\tau) = \min_{i \in N} \max_{|\sigma'_k - \sigma_i| < \tau} \|\mathbf{y}_k - \mathbf{y}'_i\| \quad (11)$$

$$\varepsilon(\tau) = \max\{\varepsilon_M(\tau), \varepsilon_I(\tau)\} \quad (12)$$

Similar definitions hold for the smallest τ given an ε. We can minimize $\varepsilon(\tau)$ over the space of TSS to determine a smallest $(\tau, \varepsilon_*(\tau))$ such that the two systems are $(T, J, (\tau, \varepsilon))$-closeness. The approach in Algorithm 1 has the advantage of working not just for $(T, J, (\tau, \varepsilon))$-closeness, but any other, application-dependent, notion of conformance.

V. Experiments

We illustrate the proposed approach on three systems, including a commercial high fidelity engine model. In all experiments, we didn't restrict the maximum number of jumps J in a given trajectory; rather, the simulation ended only when simulation time reached T. So below, we set J equal to some appropriately large J_M.

Example 5 (Example 1 continued): We use the FC Model and Implementation from Example 1 to illustrate the application of Algorithm 1 to find the tightest values of τ and ε such that $(T, J, (\tau, \varepsilon))$-closeness is true. Because the (τ, ε) pairs are partially ordered, we are looking for the Pareto-optimal front. We decided to fix τ at 0.01, and do a search over ε. To determine which value of ε to start the search from, we computed the maximum relative error between the outputs of the LUTs and the outputs of the corresponding polynomials over a window of 85 seconds, using randomly generated inputs. The maximum relative error was 0.4091. Obviously, because the LUTs are deep in the system, we do not expect the same relative error at their outputs as that at the output of the entire system. However, this duplicates the typical procedure for deciding how many entries to have in an LUT: fewer levels consumes less memory and makes for a faster computation, but causes greater error. So the designer starts from a few entries and observes the output of the system. If the error in the oputput is not acceptable, entries are added to the LUT to provide a better approximation. And so on.

Figure 4 shows a close-up of the the output trajectories from System and Implementation. Note that, as shown in Fig. 3 for the Fuel output, the two trajectories don't simply diverge and maintain one distance from each other, but rather, they diverge for a period only to meet up again. This interplay between time difference and space difference is well-captured by $(T, J, (\tau, \varepsilon))$-closeness.

S-TaLiRo [19] was run at each iteration of the binary search to falsify $\varphi_{(0.01,\varepsilon)}$. Algorithm 1 found an interval $\varepsilon \in [0.71752, 0.71832]$ over which the robustness varies between between -0.0029 and 0.027. That is, The two systems are $(85, J_M, (0.01, \varepsilon))$-close with $\varepsilon \in [0.71752, 0.71832]$.

Example 6: To illustrate the falsification of application-dependent notions, we choose φ_{PWC} given by (9), and apply it to the navigation benchmark Nav0 from [32]. Nav0 is a 4D hybrid automaton with 16 modes. Its guard sets are categorized as either 'horizontal' or 'vertical'. Fifteen implementations are generated by varying the continuous dynamics in each mode (resulting in Implementations Dyn_1-Dyn_9), and varying the horizontal guards (resulting in Implementations HG_1-HG_3) and vertical guards (resulting in Implementations VG_1-VG_3). The variations are such that the difference between Nav0 and Dyn_k is smaller than the difference between Nav0 and Dyn_{k+1}. Similarly, the difference between Nav0 and HG_k is smaller than the difference between Nav0 and HG_{k+1}, and comparable to that between Nav0 and VG_k.

We ran S-TaLiRo to minimize $[\![\varphi_{PWC}]\!]_\theta$, the temporal robustness of φ_{PWC}. Simulated Annealing (SA) was used as optimizer. Since it is a stochastic algorithm, to collect statistics, we ran 20 runs of 500 tests each, and each test lasts for $T = 20$ seconds. D was set to 0.5. The results are presented in Table I. 12 out of the 15 implementations were falsified, i.e. found to be non-conformant to the Model. Implementations HG_{1-3} are robustly conformant to the Model, as their robustness was infinite: this means that modifying the horizontal guards within the amounts prescribed by HG_3 can not affect PWC conformance. On the other hand, only one test was sufficient to falsify φ_{PWC} with the vertical guard modifications. This shows great sensitivity of the system to the vertical guard conditions. This is useful design input, as it tells the designers that they can trade-off horizontal guard implementation accuracy for greater accuracy in implementing the vertical guards.

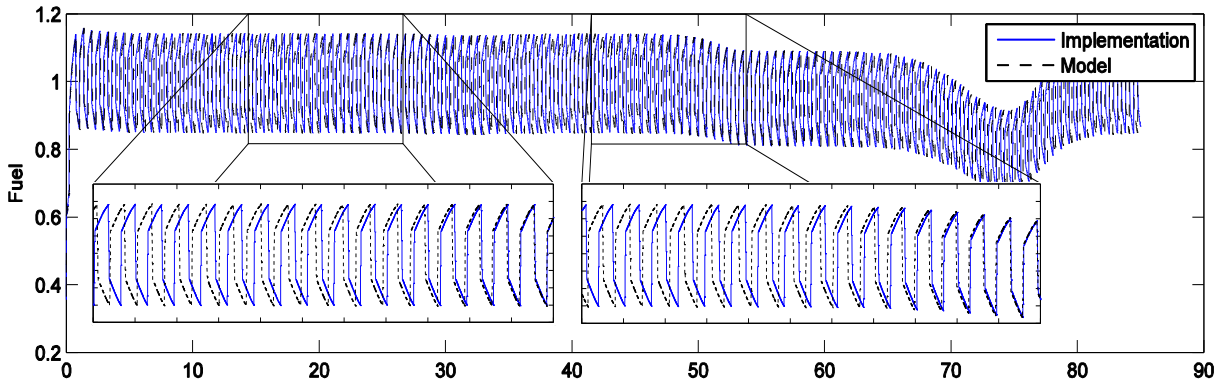


Fig. 3: Example 5. The Fuel output trajectories periodically separate from each other and converge again.

Implementations	Nb falsifying runs (out of 20)	Avg nb of tests required for falsification	Avg robustness	Avg falsification time
Dyn_1	17	181.47	$-\infty$	153.12
Dyn_2	13	119.3	$-\infty$	98.08,
Dyn_3	18	141.77	$-\infty$	117.71
Dyn_4	20	41,45	$-\infty$	33.12
Dyn_5	20	31.65	$-\infty$	24.82
Dyn_6	20	27.55	$-\infty$	21.71
Dyn_7	20	11.6	$-\infty$	8.60
Dyn_8	20	2.15	-0.081	1.59
Dyn_9	20	1.15	$-\infty$	0.90
HG_1	0	N/A	$+\infty$	N/A
HG_2	0	N/A	$+\infty$	N/A
HG_3	0	N/A	$+\infty$	N/A
VG_1	20	1	$-\infty$	0.46
VG_2	20	1	$-\infty$	0.47
VG_3	20	1	$-\infty$	0.48

TABLE I: Results of minimizing $[\![\varphi_{PWC}]\!]_\theta$ using S-TaLiRo. For each Implementation, are given the number of runs that showed it to be non-conforming to the Model (second column), the average number of tests (or trajectories) needed before a falsifying trajectory is found (third column), the average robustness, and the average runtime until falsification. Average quantities are taken over the 20 runs.

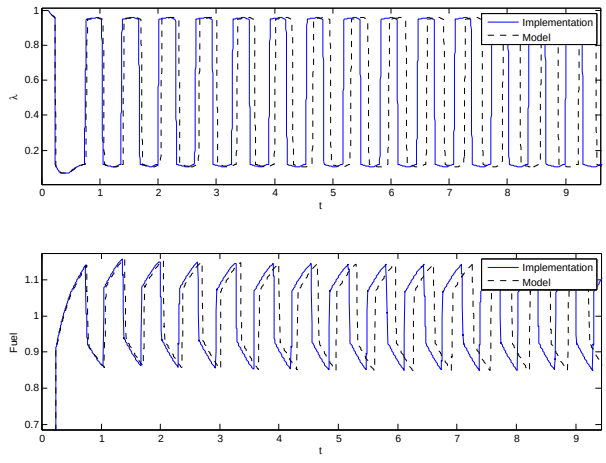


Fig. 4: Example 5. Close-up on the trajectories.

VI. Case Study: Toyota Engine Controller

We consider two models of varying fidelity of an internal combustion engine. The models are experimental and are provided by Toyota Technical Center[4]. The first model Σ_1 is a high-fidelity engine plant and controller while the second one Σ_2 is a simplified, polynomial approximation of Σ_1. In this case, with conformance testing we aim to explore how close the two models are and whether we can utilize Σ_2 for model predictive control. Both Σ_1 and Σ_2 are modeled in Matlab/Simulink. Σ_1 has 1878 blocks, including 10 integrator blocks, 47 lookup tables, 19 saturation blocks, 27 switch blocks, and 44 subsystem blocks. The model has 11 continuous and 68 discrete states. The controller is defined in a function block and contains $\approx$ 500 lines of code with multiple if-else conditional statements. Model Σ_2 has 1858 blocks, including 47 lookup tables, 17 saturation blocks, 27 switch blocks and 49 subsystem blocks. with no continuous states and 60 discrete states. Both Σ_1 and Σ_2 have two inputs and one output each. The inputs are *Fuel Level* and the number of engine revolutions (*Ne*). The output is the pressure intake manifold (*pin*). In the following, we run several experiments with different optimization functions. We also provide partial coverage of the input space as well as branch coverage for the controller code.

[4]We note that due to confidentiality agreements, we have removed the unit measurements of the y-axes from the figures presented in this section.

A. Experimental Design Parameters

1) Simulated Annealing: The search space over system inputs is defined over control points which are then interpolated to generate an input signal. Here, both inputs have 10 control points. The search space also includes timing of the control points. Since the initial and terminal timing control points are fixed at the start and end of the simulation, respectively, for each input, we have an additional 8 search variables. The signals are interpolated through a piecewise constant interpolation function. An example input signal is presented in Fig. 5 (a) (Left). In total, we have 36 search variables. The stochastic optimizer utilized is a simulated annealing algorithm. After 1000 tests, we found the system inputs that generate the outputs in Fig. 5. In the left figure, system inputs are presented. Figure 5 (a) (Middle) shows the system outputs, where the red out is from the complex model Σ_1, and the blue line is from the simplified model Σ_2. The right figure is the Pareto front over (τ, ϵ) which illustrates the ϵ difference over the τ range. Note that the Pareto Front is over 1000 tests and is therefore an under-approximation of the *true* Pareto Front over all system behaviors. We can guarantee that (τ, ϵ) is at least as large as shown in Fig. 5 (a) (Right).

2) Grid Search: One of the goals in the conformance testing process was to provide a level of input search space coverage over the systems. To do so, we developed a grid search algorithm which divides the input search space in a grid. Formally, search is conducted over the following set $S = \{(x, y) : x \in [min(U_1) : \frac{range(U_1)}{g-1} : max(U_1)] \; and \; y \in [min(U_2) : \frac{range(U_2)}{g-1} : max(U_2)]\}$, where U_1, U_2 are input ranges for inputs 1 and 2, respectively. Here, we g we set the granularity of the grid. The results of the grid search algorithm are presented in Fig. 5 (b). The middle figure shows highly non-conformant behavior between Σ_1 and Σ_2, indicating a possible singularity in either model at this particular input. Here, since the maximal difference between the two trajectories constant over a period of $2 \times \tau$, then the Pareto Front on the right figure is flat.

3) Controller Branch Coverage: The next step in the analysis is to conduct conformance testing while making sure that we have controller branch coverage. We will consider three if-else blocks to be of particular importance. Namely M_1 with 12 branches, M_2 with 2 branches, and M_3 with 4 branches. These if-else blocks are instrumented automatically from the simulink model by utilizing the approach provided in [xxx]. The instrumentation process extracts the branch information from the function block and passes it to S-TaLiRo for conformance testing. In the rest of the paper, we treat M_1, M_2, and M_3 as State Machines and refer to branches hit as locations. We modify the cost function of the optimization problem so that the system is guided towards a location n-tuple. For example, in our case, we could set as a target locations (6,2,3). If the target location is reached, then the cost function is the conformance metric which we aim to maximize.

After running our algorithm we found non-conformant behavior in locations $(5, 2, 3)$. The results are presented in Fig. 5 (c).

VII. Related Work

A. Testing, Verification and Validation

Verification methods have been very successful in finding bugs in software systems [40], [41]. Similar success stories would be desirable for testing and verification of CPS. To achieve this, there has been a substantial level of research in this direction (see [42] for an overview). The methodologies range from simulation-based verification methods to exhaustive verification methods. The former have been shown to be more scalable and applicable to real world designs, however, in general, they are less formal and exhaustive than the latter. In [43], the authors provide a literature review as well as a collection of online surveys and interviews with practitioners on common and successful practices in development, verification and validation of CPS. In [44], the authors process of verification and validation of space flight software.

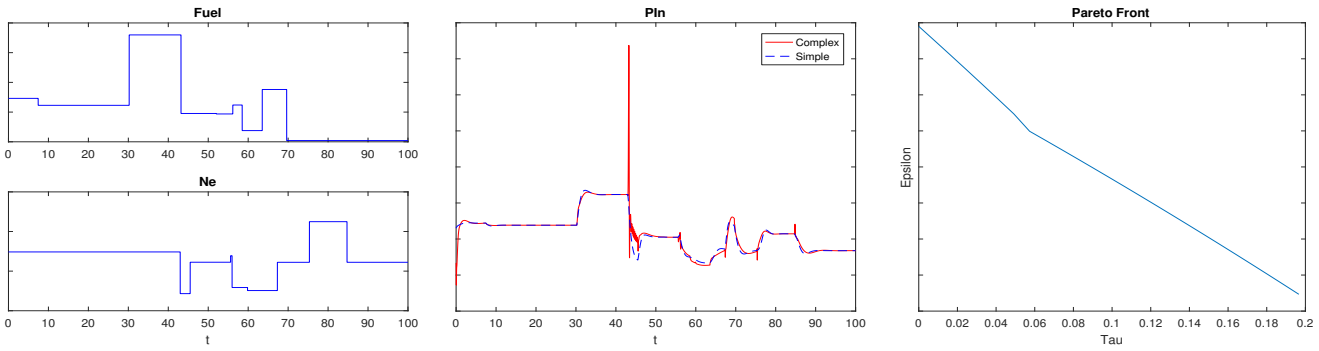
B. Conformance

Tretmans [45] defined Input-Output conformance (IOCO) as requiring that the Implementation never produces an output that can not be produced by the specification, and it is never the case that the Implementation fails to produce an output when the specification requires one. Both Implementation and specification are modeled as (discrete) labeled transition systems. Van Osch [33] later extended IOCO to hybrid transition systems (HTS) by incorporating continuous-time inputs. This hybrid IOCO is not testable in practice because the state space and transition relations of an HTS are uncountable, and the test generation algorithm proposed in [33] doesn't contain a mechanism for judiciously choosing tests from the infinite set of possible tests.

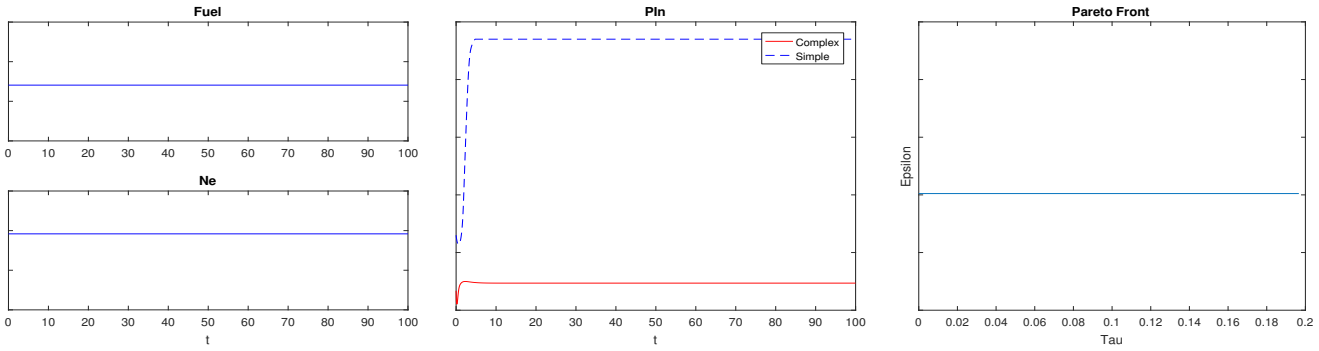
Later work [46] also extends [45] by treating the Implementation as a black box that generates timed traces, and representing the specification as a timed automaton. The objective is to verify, for each trace generated by the Implementation, whether it satisfies the invariants of the specification automaton. As such, this conformance notion does does not address this paper's goal of verifying 'similarity' between an Implementation and its Model, which is a more comprehensive problem. The work by Brandl et al. [47] utilizes (discrete) action systems [48] to provide a discrete view of hybrid systems (a modeling formalism for CPS). Thus Tretmans' IOCO can be applied to the now-discrete system. This method requires knowledge of the internal system structure, which we do not assume in our work.

In [49], a distance between systems is also defined via a distance between trajectories. The closeness notion used there can be shown to be weaker than $(T, J, (\tau, \varepsilon))$-closeness, so that proving two systems to be $(T, J, (\tau, \varepsilon))$-close implies they are close in the sense of [49]. In fact, $(T, J, (\tau, \varepsilon))$-closeness provides a continuum of closeness degrees between the two extremes presented in [49].

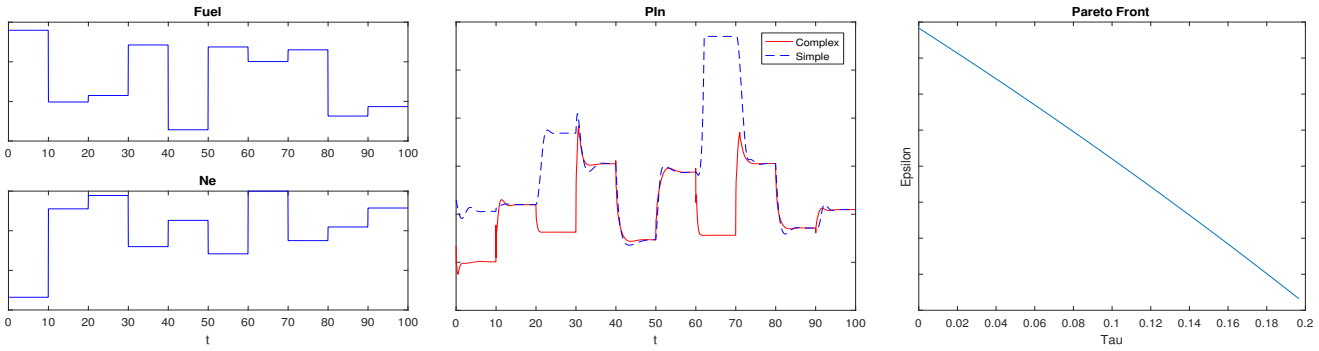
In [50], the authors propose a notion of conformance based on the Skorokhod metric. This notion captures both timing and spacial differences between trajectories and supports transference of properties in the development process. However,



(a) Simulated Annealing. Note that the timing between the interpolated input signal (Left) varies.



(b) Grid Search



(c) Controller Branch Coverage

Fig. 5: Experimental results under various design parameters. Left: System inputs for *Fuel* and *Ne*. Middle: System output for *PIn*. Right: Pareto front over all system behaviors that illustrates the ϵ difference over the τ range.

the physical interpretation of the Skorohod distance is not as intuitive as the (τ, ϵ) metric.

VIII. Conclusions

In this paper, we have defined conformance between a Model and its Implementation as a degree of closeness between the outputs of the two systems. This notion is quantifiable, thus allowing us to speak of degrees of conformance, giving a richer picture of the relation between the two systems. It is also applicable to very general system models, which allows us to study the conformance of Models to complex Implementations. This conformance was then expressed as an MTL formula, allowing us to use existing falsification tools to find non-conformant behavior of Model and Implementation, if it exists.

Because a CPS will usually have several operating modes with different dynamics, it will be interesting in future work to explicitly incorporate the mode switching into the MTL formulae. Finally, a more complete theory of conformance should also account for different time domains between the Model's trajectories and the Implementation's trajectories.

Acknowledgment

The work presented here benefited from the input of Raymond Turin, Founder and CTO at SimuQuest, who provided assistance in working with the SimuQuest Enginuity model.

This work was partially funded under NSF award CNS 1319560.

REFERENCES

[1] P. Tabuada, *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer, 2009.

[2] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*. Online http://leeseshia.org/, 2011.

[3] G. Frehse, C. L. Guernic, A. Donze, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable verification of hybrid systems," in *Proceedings of the 23d CAV*, 2011.

[4] P. Roy, P. Tabuada, and R. Majumdar, "Pessoa 2.0: a controller synthesis tool for cyber-physical systems," in *Proceedings of the 14th international conference on Hybrid systems: computation and control*. New York, NY, USA: ACM, 2011, pp. 315–316.

[5] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray, "Tulip: a software toolbox for receding horizon temporal logic planning," in *Proceedings of the 14th international conference on Hybrid systems: computation and control*. ACM, 2011, pp. 313–314.

[6] Z. Huang and S. Mitra, "Computing bounded reach sets from sampled simulation traces," in *The 15th International Conference on Hybrid Systems: Computation and Control*. ACM, 2012.

[7] A. Tiwari, "HybridSAL relational abstracter," in *Computer Aided Verification*, ser. LNCS, vol. 7358. Springer, 2012, pp. 725–731.

[8] A. Platzer and J.-D. Quesel, "KeYmaera: A hybrid theorem prover for hybrid systems." in *International Joint Conference on Automated Reasoning*, ser. LNCS, vol. 5195. Springer, 2008, pp. 171–178.

[9] R. S. Kalawsky, J. O'Brien, S. Chong, C. Wong, H. Jia, H. Pan, and P. R. Moore, "Bridging the gaps in a model-based system engineering workflow by encompassing hardware-in-the-loop simulation," *IEEE Systems Journal*, vol. 7, no. 4, pp. 593–605, 2013.

[10] A. Girard, A. Julius, and G. Pappas, "Approximate simulation relations for hybrid systems," *Discrete Event Dynamic Systems*, vol. 18, no. 2, pp. 163–179, 2008. [Online]. Available: http://dx.doi.org/10.1007/s10626-007-0029-9

[11] A. C. Antoulas, D. C. Sorensen, and S. Gugercin, "A survey of model reduction methods for large-scale systems," *Contemporary Mathematics*, vol. 280, pp. 193–219, 2000.

[12] E. Mazzi, A.-S. Vincentelli, A. Balluchi, and A. Bicchi, "Hybrid system reduction," in *47th IEEE Conference on Decision and Control*, 2008, pp. 227–232.

[13] T. A. Henzinger, R. Majumdar, and V. S. Prabhu, "Quantifying similarities between timed systems." in *FORMATS*, ser. LNCS, vol. 3829. Springer, 2005, pp. 226–241.

[14] R. Koymans, "Specifying real-time properties with metric temporal logic." *Real-Time Systems*, vol. 2, no. 4, pp. 255–299, 1990.

[15] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Proceedings of FORMATS-FTRTFT*, ser. LNCS, vol. 3253, 2004, pp. 152–166.

[16] J. Lygeros, K. H. Johansson, S. N. Simic, J. Zhang, and S. Sastry, "Dynamical properties of hybrid automata," *IEEE Transactions on Automatic Control*, vol. 48, pp. 2–17, 2003.

[17] R. G. Sanfelice and A. R. Teel, "Dynamical properties of hybrid systems simulators," *Automatica*, vol. 46, no. 2, pp. 239–248, 2010.

[18] P. Caspi and A. Benveniste, "Toward an approximation theory for computerized control," in *Embedded Software*, ser. LNCS, vol. 2491. Springer, 2002, pp. 294–304.

[19] Y. S. R. Annapureddy, C. Liu, G. E. Fainekos, and S. Sankaranarayanan, "S-taliro: A tool for temporal logic falsification for hybrid systems," in *Tools and algorithms for the construction and analysis of systems*, ser. LNCS, vol. 6605. Springer, 2011, pp. 254–257.

[20] G. Fainekos, "S-TALIRO," [Online at: https://sites.google.com/a/asu.edu/s-taliro/s-taliro].

[21] H. Abbas and G. Fainekos, "Convergence proofs for simulated annealing falsification of safety properties," in *Proc. of 50th Annual Allerton Conference on Communication, Control, and Computing*. IEEE Press, 2012.

[22] H. Abbas, G. E. Fainekos, S. Sankaranarayanan, F. Ivancic, and A. Gupta, "Probabilistic temporal logic falsification of cyber-physical systems," *ACM Transactions on Embedded Computing Systems*, vol. 12, no. s2, May 2013.

[23] A. Lecchini-Visintini, J. Lygeros, and J. Maciejowski, "Stochastic optimization on continuous domains with finite-time guarantees by markov chain monte carlo methods," *Automatic Control, IEEE Transactions on*, vol. 55, no. 12, pp. 2858 –2863, dec. 2010.

[24] P. James, *Verification Plans: The Five-Day Verification Strategy for Modern Hardware Verification Languages*. Kluwer Academic Publishers, 2004. [Online]. Available: http://books.google.com/books?id=JSA1kJ-LiN8C

[25] Atrenta, "Bugscope ^{TM}," [Online at: http://www.atrenta.com/solutions/bugscope.htm5].

[26] L. Guzzella and C. Onder, *Introduction to Modeling and Control of Internal Combustion Engine Systems*, 2nd ed. Springer-Verlag, 2010.

[27] X. Jin, J. Kapinski, J. V. Deshmukh, K. Ueda, and K. Butts, "Fuel control system verification benchmark problems," in *Submitted to: Hybrid Systems: Computation and Control*, 2014.

[28] R. G. Sanfelice, "Interconnections of hybrid systems: Some challenges and recent results," *Journal of Nonlinear Systems and Applications*, vol. 2, no. 1-2, pp. 111–121, 2011.

[29] R. Goebel and A. Teel, "Solutions to hybrid inclusions via set and graphical convergence with stability theory applications," *Automatica*, vol. 42, no. 4, pp. 573 – 587, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0005109806000185

[30] L. Bako, V. L. Le, F. Lauer, and G. Bloch, "Identification of MIMO switched state-space models," in *American Control Conference (ACC), 2013*, 2013, pp. 71–76.

[31] H. Abbas and G. Fainekos, "Computing descent direction of mtl robustness for non-linear systems," in *American Control Conference (ACC)*, 2013, pp. 4411–4416.

[32] ——, "Linear hybrid system falsification through local search," in *Automated Technology for Verification and Analysis*, ser. LNCS, vol. 6996. Springer, 2011, pp. 503–510.

[33] M. Osch, "Hybrid input-output conformance and test generation," in *Formal Approaches to Software Testing and Runtime Verification*, ser. LNCS. Springer Berlin Heidelberg, 2006, vol. 4262, pp. 70–84.

[34] K. H. Johansson, J. Lygeros, S. Sastry, and M. Egerstedt, "Simulation of hybrid zeno automata," in *Conference on Decision and Control*, vol. 4, December 1999, pp. 3538–3543.

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

[36] A. Donze and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Formal Modeling and Analysis of Timed Systems*, ser. LNCS. Springer Berlin Heidelberg, 2010, vol. 6246, pp. 92–106. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15297-9_9

[37] S. Sankaranarayanan and G. Fainekos, "Falsification of temporal properties of hybrid systems using the cross-entropy method," in *ACM International Conference on Hybrid Systems: Computation and Control*, 2012.

[38] J. Legriel, C. Guernic, S. Cotton, and O. Maler, "Approximating the pareto front of multi-criteria optimization problems," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, J. Esparza and R. Majumdar, Eds. Springer Berlin Heidelberg, 2010, vol. 6015, pp. 69–83. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12002-2_6

[39] Simuquest, "Enginuity," http://www.simuquest.com/products/enginuity, accessed: 2013-10-04.

[40] J. C. Corbett, M. B. Dwyer, J. Hatcliff, S. Laubach, C. S. Păsăreanu, R. Bby, and H. Zheng, "Bandera: Extracting finite-state models from java source code," in *Software Engineering, 2000. Proceedings of the 2000 International Conference on*. IEEE, 2000, pp. 439–448.

[41] G. J. Holzmann, "The model checker spin," *IEEE Transactions on software engineering*, no. 5, pp. 279–295, 1997.

[42] J. Kapinski, J. V. Deshmukh, X. Jin, H. Ito, and K. Butts, "Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques," *IEEE Control Systems*, vol. 36, no. 6, pp. 45–64, 2016.

[43] X. Zheng, C. Julien, M. Kim, and S. Khurshid, "Perceptions on the state of the art in verification and validation in cyber-physical systems," 2015.

[44] M. C. B. Alves, D. Drusinsky, J. B. Michael, and M.-T. Shing, "End-to-end formal specification, validation, and verification process: A case study of space flight software," *IEEE Systems Journal*, vol. 7, no. 4, pp. 632–641, 2013.

[45] J. Tretmans, "Testing concurrent systems: A formal approach," in *CONCUR 1999 Concurrency Theory*. Springer, 1999, pp. 46–65.

[46] M. Woehrle, K. Lampka, and L. Thiele, "Conformance testing for cyber-physical systems," *ACM Trans. Embed. Comput. Syst.*, vol. 11, no. 4, pp. 84:1–84:23, Jan. 2013. [Online]. Available: http://doi.acm.org/10.1145/2362336.2362351

[47] H. Brandl, M. Weiglhofer, and B. K. Aichernig, "Automated conformance verification of hybrid systems," in *Quality Software (QSIC), 10th International Conference on*. IEEE, 2010, pp. 3–12.

[48] R. Back and K. Sere, "Stepwise refinement of parallel algorithms," *Science of Computer Programming*, vol. 13, no. 2, pp. 133–180, 1990.

[49] A. Abate and M. Prandini, "Approximate abstractions of stochastic systems: A randomized method," in *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, 2011, pp. 4861–4866.

[50] J. V. Deshmukh, R. Majumdar, and V. S. Prabhu, "Quantifying conformance using the skorokhod metric," in *International Conference on Computer Aided Verification*. Springer, 2015, pp. 234–250.