Powertrain Control Verification Benchmark

Xiaoqing Jin, Jyotirmoy V. Deshmukh, James Kapinski, Koichi Ueda, Ken Butts
firstname.lastname@tema.toyota.com

Powertrain Control and Model-based Development, Toyota Technical Center, USA

Abstract. Industrial control systems are often hybrid systems that are required to satisfy strict performance requirements. Verifying designs against requirements is a difficult task, and there is a lack of suitable open benchmark models to assess, evaluate, and compare tools and techniques. Benchmark models can be valuable for the hybrid systems research community, as they can communicate the nature and complexity of the problems facing industrial practitioners. We present a collection of benchmark problems from the automotive powertrain control domain that are focused on verification for hybrid systems; the problems are intended to challenge the research community while maintaining a manageable scale. We present three models of a fuel control system, each with a unique level of complexity, along with representative requirements in signal temporal logic (STL). We provide results obtained by applying a state of the art analysis tool to these models, and finally, we discuss challenge problems for the research community.

1 Introduction

Hybrid dynamical systems are prevalent in modern control designs, and many researchers continue to develop new technologies and tools to analyze such systems. Unfortunately, there is a paucity of standard benchmark systems and accompanying requirements that researchers can use to evaluate and compare the efficacy of their analysis techniques. Industrial benchmark problems can be of added value as they exemplify the context, scale, and complexity of real-world hybrid systems challenges. Often, however, industrial models are either proprietary and cannot be shared with the research community or they are too complex to be investigated with academic tools. To address this, we present a suite of benchmark hybrid systems verification problems in the form of a series of models of an automotive powertrain control system along with a collection of formal specifications.

An inherent challenge in the analysis of hybrid systems is that even the simplest of problems run into the problem of undecidability; for example, the problem of reachability analysis for linear hybrid automata was shown to be undecidable [21]. Many researchers have worked on this problem [5, 7], and continue to extend reachability analysis to systems with more interesting nonlinear dynamics [9]. This research area has been served well by efforts such as [15] that have formulated benchmark problems. The benchmarks proposed in [15] are easily extensible, for example a navigation system with a parameterized set of cells on a map-grid. While the original paper evaluated tools such as `d/dt`, CHARON, and a predicate abstraction based verifier on the benchmark problems, these benchmark problems have also successfully influenced more recent research leading to tools such as PHAver [16], SpaceEx [18] and `iSAT` [14].

In this paper, we present three benchmark models, a set of challenge problems and suite of requirements in Signal Temporal Logic (STL) specifying the behavior and performance of the models. The models that we present in Sec. 3 are successive simplifications of the model of an important control problem in the automotive domain, that of air-fuel (A/F) ratio control [10]. In an internal combustion engine, the ratio of air mass to injected fuel has direct implications on the rate of exhaust gas emissions, driveability and fuel efficiency. A three-way catalyst (TWC) system (used in catalytic converters that reduce the amount of undesirable exhaust gases) performs optimally when the A/F ratio is at the stoichiometric value. The control problem for this system is to maintain A/F ratio to this set value.

The first model is the most complex model, implemented using Simulink[1] and contains many features typical of an industrial closed-loop system; i.e., a plant model that captures the physical aspects of the fuel and air subsystems and has elements such as lookup tables (LUTs), variable transport delays, and highly nonlinear dynamics, and a controller model that is a time-triggered system with mode switching behavior that has hysteresis. This model is beyond the capabilities of most existing formal verification tools; with the state of the art, the best one could achieve with this model is simulation-based falsification.

The second model presents hybrid I/O automata (HIOA) models of the plant and the controller and an automaton representing their composition. In this model, we simplify dynamics of some subsystems, remove time delays, replace lookup tables with continuous functions, and remove certain subsystems. The plant is an HIOA model of a continuous dynamical system, while the controller is an HIOA model of a discrete-time multi-modal dynamical system. The third model is a further simplification in which the controller is modeled with continuous-time differential equations, and all nonpolynomial dynamics are approximated by polynomials. These final two models could, perhaps, be more amenable to formal techniques such as reachable set estimation or stability analysis.

The rest of the paper is organized as follows. We review definitions related to hybrid I/O automata and Signal Temporal Logic in Section 2. In Section 3, we present the three benchmark models and provide a collection of formal requirements for them in Section 4. Section 5 presents samples of analyses of the models using a falsification tool known as S-TaLiRo [6]. Finally, in Section 6, we present verification research challenges.

2 Preliminaries

In this section, we briefly review the terminology that we will use in this paper. We use the formalism of hybrid input/output automata (HIOA) similar to the one presented in [17, 24]. These are essentially hybrid automata extended with ability to process exogenous inputs and produce outputs. We further elaborate on how a hybrid system with a mixture of discrete-time and continuous-time state variables can be modeled by such a hybrid automaton. Finally, we introduce the falsification problem for hybrid systems.

[1] Simulink$^{\text{TM}}$ is a registered trademark of the MathWorks Inc.

2.1 Hybrid I/O Automata

A hybrid automaton is a useful model of a system that displays continuous-time behavior interleaved with discrete jumps. Hybrid automata with inputs and outputs have additional structure as they allow exogenous time-varying inputs, and observable outputs. Due to this additional structure, HIOA facilitate modular descriptions of subsystems and their compositions to obtain larger systems.

Let $\mathcal{V} = \{v_1, \ldots, v_k\}$ denote a set of variables, such that each of them takes values from possibly different domains. Let $\mathbb{D}(\mathcal{V})$ denote the domain of values for the variables in $\mathcal{V}$. Let $\nu(\mathcal{V})$ denote a valuation for the variables, i.e., a tuple of values from $\mathbb{D}(\mathcal{V})$. In the sequel, we follow the notation that caligraphic symbols (such as $\mathcal{V}$) denote sets of variables $\{v_1, \ldots, v_k\}$ and the corresponding normal capitalized, italic font letters (such as V) to denote $\mathbb{D}(\mathcal{V})$.

Formally, a hybrid input/output automaton (HIOA) $\mathcal{A}$ is defined as a tuple $(\mathcal{L}, \mathcal{X}, \mathcal{U}, \mathcal{M}, \mathcal{G}, \mathcal{R}, \Delta, \mathcal{T}, \mathcal{Y}, \mathcal{I})$, where:

- $\mathcal{L}$ is a finite set of *discrete modes*.
- $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a finite set of n state variables, and following the notation defined before, X denotes the set of all valuations of $\mathcal{X}$. The hybrid state space is thus a subset of the set $\mathcal{L} \times X$. We use $\mathbf{x}$ as an abbreviation for the tuple $(x_1, \ldots, x_n)$.
- $\mathcal{U} = \{u_1, \ldots, u_m\}$ denotes a set of m typed input variables. Note that input variables could be of different types such as $\mathbb{R}$, $\mathbb{Z}$, Booleans, etc. We use $\mathbf{u}$ as an abbreviation for the tuple $(u_1, \ldots, u_m)$.
- $\mathcal{M}$ maps each mode ℓ with a *mode invariant* $\mathcal{M}(\ell) \subseteq X \times U$.
- $\mathcal{G}$ is a set of predicates over $X \times U$.
- $\mathcal{R}$ is a set of functions from $X \times U$ to X.
- $\Delta \subseteq \mathcal{L} \times \mathcal{G} \times \mathcal{R} \times \mathcal{L}$, is a finite set of transitions. For each transition $\delta : (\ell, g, r, \ell') \in \Delta$, $g \in \mathcal{G}$ is its *guard predicate*, and $r \in \mathcal{R}$ its *reset map*.
- Let $\mathbb{T} \subseteq \mathbb{R}^{\geq 0}$ represent the domain of time values. A *trajectory* $\tau_{(\mathcal{X},\mathcal{U})}$ is a function from $\mathbb{T}$ to $(X \times U)$ that describes the valuations of the input variables and state variables over time. $\mathcal{T}$ is a finite or infinite set of trajectories. In an HIOA, a trajectory is often a sequence of alternating *flows* (within modes) and *resets* (consistent with mode transitions). For a given mode ℓ, the flow within the mode is typically the solution trajectory $\mathbf{x}(\cdot)$ of an initial value problem as described by an ordinary differential equation (ODE) of the form $\dot{\mathbf{x}} = f_\ell(\mathbf{x}, \mathbf{u})$ with $\nu(\mathbf{x}) = \mathbf{x}_0$ at time $t = t_0$.
- $\mathcal{Y} \subseteq \mathcal{X}$ denotes a set of typed output variables.
- $\mathcal{I} \subseteq \mathcal{L} \times X$ is the set of possible initial discrete modes and valuations of the state variables.

In what follows, we use $\mathbf{x}(t)$, $\mathbf{u}(t)$ to denote the valuation of the variables $\mathbf{x}$ and $\mathbf{u}$ at time t.

Parallel Composition. Given two HIOA $\mathcal{A}_1$ and $\mathcal{A}_2$, where $\mathcal{A}_i$ is defined as the tuple $(\mathcal{L}_i, \mathcal{X}_i, \mathcal{U}_i, \mathcal{M}_i, \mathcal{G}_i, \mathcal{R}_i, \Delta_i, \mathcal{T}_i \mathcal{Y}_i, \mathcal{I}_i)$, for $i \in \{1, 2\}$, we say that $\mathcal{A}_1$ and $\mathcal{A}_2$ are *compatible* if $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$, $\mathcal{Y}_1 \subseteq \mathcal{U}_2$ and $\mathcal{Y}_2 \subseteq \mathcal{U}_1$.

The parallel composition operation allows compatible HIOA representing two modules to be composed to form a composite module. Note that the input and output variables that are part of the interface between two HIOA disappear as a result of composition. For a given set X, let $X \downarrow Y$ denote projection or the restriction of X to the elements of Y. The composite HIOA $\mathcal{A}_c = \mathcal{A}_1 \parallel \mathcal{A}_2$ representing the parallel composition of $\mathcal{A}_1$ and $\mathcal{A}_2$ as shown above is defined as the tuple $(\mathcal{L}_c, \mathcal{X}_c, \mathcal{U}_c, \mathcal{M}_c, \mathcal{G}_c, \mathcal{R}_c, \Delta_c, \mathcal{Y}_c, \mathcal{T}_c, \mathcal{I}_c)$, where:

- $\mathcal{L}_c = \mathcal{L}_1 \times \mathcal{L}_2$,
- $\mathcal{X}_c = \mathcal{X}_1 \cup \mathcal{X}_2$,
- $\mathcal{U}_c = (\mathcal{U}_1 \cup \mathcal{U}_2) \setminus (\mathcal{Y}_1 \cup \mathcal{Y}_2)$,
- $\nu(\mathcal{X}_c) \in \mathcal{M}_c(\ell_1, \ell_2)$ iff for $i \in \{1, 2\}$, $\nu(\mathcal{X}_c \downarrow \mathcal{X}_i) \in \mathcal{M}_i(\ell_i)$,
- $\mathcal{G}_c$ is a set of predicates over $X_c \times U_c$ and $\mathcal{R}_c$ is a set of functions from $X_c \times U_c$ to X_c.
- A transition δ_c of the form $(\ell_c, g_c, r_c, \ell'_c)$ is in the set Δ_c if:
  - There is a transition $\delta_1 \in \Delta_1$ of the form $(\ell_1, g_1, r_1, \ell'_1)$, and $\ell_c = (\ell_1, \ell_2)$, $g_c = g_1$, $r_c = r_1$ and $\ell'_c = (\ell'_1, \ell_2)$, or,
  - there is a transition $\delta_2 \in \Delta_2$ of the form $(\ell_2, g_2, r_2, \ell'_2)$, and $\ell_c = (\ell_1, \ell_2)$, $g_c = g_2$, $r_c = r_2$ and $\ell'_c = (\ell_1, \ell'_2)$ or,
  - there are transitions δ_1 and δ_2 as described in the previous cases, and $\ell_c = (\ell_1, \ell_2)$, $\ell'_c = (\ell'_1, \ell'_2)$, $g_c = g_1 \wedge g_2$, and $r_c = r_1 \cup r_2$.
- $\mathcal{Y}_c = (\mathcal{Y}_1 \cup \mathcal{Y}_2) \setminus (\mathcal{U}_1 \cup \mathcal{U}_2)$,
- A trajectory $\tau_{(\mathcal{X}_c, \mathcal{U}_c)}$ is in $\mathcal{T}_c$ iff for $i = 1, 2$, $\tau_{(\mathcal{X}_i, \mathcal{U}_i)} \in \mathcal{T}_i$.
- $((\ell_1, \ell_2), \nu(\mathcal{X}_c)) \in \mathcal{I}_c$ iff for $i = \{1, 2\}$, $(\ell_i, \nu(\mathcal{X}_c \downarrow \mathcal{X}_i)) \in \mathcal{I}_i$.

Modeling discrete-time updates. In our model, we have a mixture of states that evolve in continuous-time according to dynamics defined by an ODE, as well as states that evolve in discrete-time that stay constant between discrete time-steps. The latter can be modeled by a traditional hybrid automaton using resets.

2.2 Polynomial Hybrid I/O Automata

Let $\mathcal{U}_R$ denote the variables $\mathcal{U}$ restricted to those u_j that take values over $\mathbb{R}$. Let $\mathbb{P}(\mathbf{x}, \mathbf{u}, k)$ denote the set of all polynomial functions from $(\mathcal{X} \times \mathcal{U}_R)$ to $\mathbb{R}$, such that for any polynomial function $f(\mathbf{x}, \mathbf{u}) \in \mathbb{P}(\mathbf{x}, \mathbf{u}, k)$, the maximum degree of any monomial in $f(\mathbf{x}, \mathbf{u})$ is k. Further, let $\mathbb{B}(\mathbf{x}, \mathbf{u}, k)$ denote the infinite set of predicates of the form $g(\mathbf{x}, \mathbf{u}) < c$, where $g \in \mathbb{P}(\mathbf{x}, \mathbf{u}, k)$ and $c \in \mathbb{R}$. A *polynomial hybrid I/O automaton* (PHA) is an HIOA where:

- For each mode ℓ, the flow within the mode is described as a solution trajectory of the initial value problem of an ODE of the form $\dot{\mathbf{x}} = f_\ell(\mathbf{x}, \mathbf{u})$, where $f_\ell \in \mathbb{P}(\mathbf{x}, \mathbf{u}, k)$.
- For each mode ℓ, $\mathcal{M}(\ell)$ is a finite Boolean combination of predicates in $\mathbb{B}(\mathbf{x}, \mathbf{u}, k)$.
- The set $\mathcal{G}$ is a finite subset of the set $\mathbb{B}(\mathbf{x}, \mathbf{u}, k)$.
- The set $\mathcal{R}$ is a finite subset of $\mathbb{P}(\mathbf{x}, \mathbf{u}, k)$.

A well-known subclass of polynomial hybrid I/O automaton is that of *linear hybrid automata* (LHA), where the sets in the above definition are restricted to those with degree $k = 1$. As reachability analysis tools such as SpaceEx [18], and Flow* [9] are applicable for models expressed as LHA and PHA respectively, these are interesting subclasses of the general model.

2.3 Falsification Analysis

In Section 4, we provide some benchmark specifications that characterize desired behavior of the closed-loop system consisting of the plant and the controller. Such *formal* specifications, also known in the automotive world as *verifiable requirements* are typically not expressed in a format amenable to analysis tools. The ultimate goal for automotive control systems is improvement in metrics such as fuel economy, driveability, and lower exhaust gas emissions. With the benchmark specifications, our intent is to map these high-level requirements to checkable requirements on individual subsystems (such as the A/F control subsystem).

To specify requirements, we use the formalism of signal temporal logic (STL) or its close variant metric temporal logic (MTL). These logics are extensions of linear temporal logic (LTL), which is used in software verification to specify correctness of finite or infinite traces of programs. STL and MTL extend the temporal operators provided by LTL with time-intervals, and this allows us to specify real-time requirements. While MTL characterizes time-varying behavior of pre-defined Boolean predicates, STL admits time and frequency-domain properties on signals [12, 13]. Here, we focus on the time-domain specifications using STL. Below we give the formal syntax and semantics of STL, which are similar to those of MTL [23, 4].

Syntax and Semantics of STL. STL is a temporal logical formalism that allows predicates on real-valued *signals*; it is used to specify precise real-time relations between predicates. An n-dimensional signal $\mathbf{x}$ is defined as a function from a time domain (in our case this is some *interval*, i.e., a bounded and closed subset of $\mathbb{R}$) to $\mathbb{R}^n$.

Informally, an atomic unit of an STL formula is some predicate μ on an n-dimensional signal $\mathbf{x}$, and can be converted to an inequality of the form $f(\mathbf{x}) > 0$, where f is any function from $\mathbb{R}^n$ to $\mathbb{R}$. Note that as $\mathbf{x}$ is an n-dimensional signal, this definition allows constraints relating multiple signals. Further, equality constraints can be specified by using Boolean combinations of predicates. Relations between predicate values at different time instants in a signal are specified using *timed temporal operators*. Temporal operators such as $\Box$ ("always"), $\Diamond$ ("eventually") and $\mathbf{U}$ ("until") are commonly used. Each temporal operator ranges over an open time-interval of the form (a, b) where $a, b \in \mathbb{R}$. Other time-intervals such as those with one or both end-points included in the interval are also permitted. We use I to denote any such time-interval. While it is known that the $\Box$ and $\Diamond$ operators can be rewritten using the $\mathbf{U}$ operator, in the following formal syntax we include them for completeness:

$$\begin{array}{ll}
\varphi ::= \mathit{true} & \text{/* the } \textit{true} \text{ predicate */} \\
\quad \mid \mu & \text{/* } \mu\text{: signal predicate } f(\mathbf{x}) > 0 \text{ */} \\
\quad \mid \neg\varphi \mid \varphi \wedge \psi & \text{/* Boolean combinations */} \\
\quad \mid \Box\varphi & \text{/* always */} \\
\quad \mid \Diamond\varphi & \text{/* eventually */} \\
\quad \mid \varphi\, \mathbf{U}_I\, \psi & \text{/* timed until */}
\end{array}$$

The satisfaction of an STL property by a signal is always relative to a specified time instant in the signal. We denote by $(\mathbf{x}, t_0) \models \varphi$ the satisfaction of φ by $\mathbf{x}$ at time t_0. The semantics of STL formulas are inductively defined using first-order logic with quantifiers as follows:

$$
\begin{array}{ll}
(\mathbf{x}, t_0) \models & \\
\mu & \text{iff } \mathbf{x} \text{ satisfies } \mu \text{ at time } t_0 \\
\neg\varphi & \text{iff } (\mathbf{x}, t_0) \not\models \varphi \\
\varphi_1 \wedge \varphi_2 & \text{iff } (\mathbf{x}, t_0) \models \varphi_1 \text{ and } (\mathbf{x}, t_0) \models \varphi_2 \\
\Box_{(a,b)}\varphi & \text{iff } \forall t : t \in (t_0{+}a, t_0{+}b) \Rightarrow (\mathbf{x}, t) \models \varphi \\
\Diamond_{(a,b)}\varphi & \text{iff } \exists t : t \in (t_0{+}a, t_0{+}b) \wedge (\mathbf{x}, t) \models \varphi \\
\varphi \, \mathbf{U}_{(a,b)} \, \psi & \text{iff } \exists t_1 : t_1 \in (t_0{+}a, t_0{+}b) \wedge (\mathbf{x}, t_1) \models \psi \wedge \\
& \qquad \forall t_2 : t_2 \in (t_0, t_1) \Rightarrow (\mathbf{x}, t_2) \models \varphi
\end{array}
$$

Falsification Problem. In simple terms, the falsification problem is to determine if given a hybrid system, there is a trajectory of the system that does not satisfy a specific logical requirement. In addition to the Boolean semantics specified above, the logics STL and MTL also allow quantitative semantics. These allow us to define a numeric value known as the *robustness value* or the *satisfaction value* for a property with respect to a given $(\mathbf{x}, t_0)$ (denoted as $\rho(\mathbf{x}, t_0, \varphi)$). Informally, $\rho(\mathbf{x}, t_0, \varphi)$ indicates how strongly $(\mathbf{x}, t_0)$ satisfies φ. A large positive value means $(\mathbf{x}, t_0)$ easily satisfies φ, a small positive value means that $(\mathbf{x}, t_0)$ is very close to violating φ, and a negative value indicates $(\mathbf{x}, t_0) \not\models \varphi$. The quantitative semantics for STL can be found in [12].

When equipped with the quantitative semantics, previous work has framed the falsification problem as the problem of minimizing the robustness value. If the minimum value is negative, a counterexample is obtained. This optimization problem is over a highly nonlinear and hybrid space, and hence traditional optimization techniques may not succeed in finding counterexamples. In [6, 28] the authors have proposed using techniques based on stochastic optimization to find traces with minimum robustness value. In Sec. 5, we present results from experiments using S-TaLiRo to falsify some of our benchmark requirements for each of the benchmark models.

3 Models

In this section, we present three different versions of an automotive air-fuel control model. We begin with the most complex version and end with the simplest. All parameter values are listed in the Appendix.

3.1 Abstract Fuel Control System

We present a description of a fuel control model implemented in Simulink. The model contains the air-fuel controller and a mean-value model of the engine dynamics, such as the throttle and intake manifold air dynamics. The controller has provision to operate in either a.) a closed-loop mode using Proportional + Integral (PI) feedback control along with feedforward control based on an observer or b.) an open-loop mode using feedforward control based on the observer.

The plant model consists of the throttle and intake manifold and the air-path dynamics. The throttle and intake manifold models are taken from the Simulink Demo palette

[30], which was based on the work by Crossley and Cook [11] (excluding an exhaust gas recirculation system). We assume that the throttle angle θ_{in} (in degrees) and the engine speed ω (in rad/sec) are exogenous inputs to the model.

Throttle Control. An electronic throttle control system takes exogenous throttle commands θ_{in} as input and regulates the throttle plate position θ to the desired setpoint. We model this system with the following first-order ODE:

$$\frac{d\theta}{dt} = 10(\theta_{in} - \theta). \tag{1}$$

Throttle Air Dynamics. The throttle air dynamics subsystem defines the rate at which air flows past the throttle plate, or in other words the rate at which the throttle introduces air into the intake manifold. This rate is denoted by $\dot{m}_{af}$ and often called the *inlet air mass flow rate*. The quantity $\dot{m}_{af}$ is a product of two functions, one is an empirical function of the throttle plate angle given in degrees (denoted by the intermediate variable $\hat{\theta}$), and the other is a function of the atmospheric and manifold pressure. The former is given by the following polynomial equation:

$$\hat{\theta} = c_6 + c_7\theta + c_8\theta^2 + c_9\theta^3.$$

Let state variable p be the manifold pressure (in units of bar), and let c_{10} represent atmospheric pressure. The inlet air mass flow rate (denoted $\dot{m}_{af}$) in grams per second is given by:

$$\dot{m}_{af} = 2\hat{\theta}\sqrt{\frac{p}{c_{10}} - \left(\frac{p}{c_{10}}\right)^2}. \tag{2}$$

Intake Manifold. The following details the manifold dynamics, as found in the Simulink Demo palette [30]. The rate $\dot{m}_c$ at which air flows into the cylinder is a function of the throttle, and is known as the pumping polynomial (the quantity in parentheses in the RHS of (3)). In the controller, we implement an estimator for p, and to model error in estimation, we use the parameter c_{12}.

$$\dot{m}_c = c_{12}\left(c_2 + c_3\omega p + c_4\omega p^2 + c_5\omega^2 p\right) \tag{3}$$

According to the ideal gas law, the derivative of the manifold pressure is proportional to the net rate of change of air mass in the intake, which is the difference between the rate at which air enters the manifold ($\dot{m}_{af}$) and the rate at which it is pumped into the cylinder ($\dot{m}_c$). In other words, the ODE for p is given as:

$$\frac{dp}{dt} = c_1\left(\dot{m}_{af} - \dot{m}_c\right), \tag{4}$$

Substituting (2) and (3) into (4), we have

$$\frac{dp}{dt} = c_1\left(2\hat{\theta}\sqrt{\frac{p}{c_{10}} - \left(\frac{p}{c_{10}}\right)^2} - c_{12}\left(c_2 + c_3\omega p + c_4\omega p^2 + c_5\omega^2 p\right)\right). \tag{5}$$

Cylinder and Exhaust. The air-fuel path aspects of the model, including the cylinder and exhaust dynamics, are largely based on the development in [20]. Let $\dot{m}_\varphi$ be the rate of flow of fuel into the cylinder, in g/s. The A/F ratio in the cylinder is then given by:

$$\lambda_c(t) = \frac{\dot{m}_c}{\dot{m}_\varphi}. \tag{6}$$

In reality, there is a variable transport delay incurred when the exhaust gas produced by the engine reaches the oxygen (O_2) sensor for the A/F ratio measurement. This delay is given by a 2D lookup table (LUT 4(b) in the appendix). The LUT values are estimated values, taken from Figure 4.24 in [20]. The table axes are the air mass into the cylinder m_c and the engine speed (n) in rpm (revolutions per minute). The former can be obtained from $\dot{m}_c$ and the engine speed in rad/sec (ω) as $m_c = \frac{\pi \dot{m}_c}{\omega}$. Let the delay value obtained from the table be denoted $\Delta(m_c, n)$. The delayed A/F ratio passes through two first-order transfer functions, representing the exhaust system transport dynamics and the O_2 sensor dynamics. The output of the transfer functions is the measured A/F ratio, denoted $\lambda_m(t)$. Then the ODE governing $\lambda_m(t)$ is given by:

$$\begin{aligned}&\frac{d^2\lambda_m(t)}{dt^2} = \\ &\frac{1}{0.002}\left(-0.12\frac{d\lambda_m(t)}{dt} - \lambda_m(t) + \lambda_c(t - \Delta(m_c, n))\right).\end{aligned} \tag{7}$$

Sensor Fault. The oxygen sensor measures the amount of oxygen in the exhaust gas, which is proportional to the air-fuel ratio. The oxygen sensor measurements are fed back to the controller. An external event can trigger a fault in the oxygen sensor. This fault causes the output of the oxygen sensor to raise the `sensor_failed` flag. The controller detects this fault condition and reacts by switching to a mode of operation where only feedforward control is applied, and feedback control is disabled.

Wall Wetting. The wall wetting dynamics are based on the description of the Aquino model described in [20]. The mass of the fuel flowing into the cylinder is given by (8). Parameters $\kappa(\cdot)$ and $\tau(\cdot)$ appear in the Aquino model and depend on several factors including engine speed (in rpm) and the air mass in the cylinder, and are approximated by 2D LUTs (LUT 4(a)). The LUTs presented contain values estimated from those in [20] Figure 2.21.

$$\dot{m}_\varphi = (1 - \kappa(n, m_c))\dot{m}_\psi + \frac{m_f}{\tau(n, m_c)}, \tag{8}$$

In (8), $\dot{m}_\varphi$ is the fuel mass aspirated into the cylinder, $\dot{m}_\psi$ is the fuel mass injected into the intake manifold. The dynamic equation for the mass of fuel stored in the fuel film, m_f, is given by:

$$\frac{dm_f}{dt} = \kappa(\omega, m_c)\dot{m}_\psi - \frac{m_f}{\tau(\omega, m_c)}. \tag{9}$$

Equations (8), (9) are taken from (2.60) and (2.61) in [20].

State	Unit	Description
p	bar	Intake manifold pressure
λ	-	A/F Ratio
p_e	bar	Est. manifold pressure
i	-	Integrator state, PI
$\dot{m}_{af}$	g/s	Inlet air mass flow rate
$\dot{m}_c$	g/s	air flow Rate to cylinder
θ_{in}	degrees	Throttle angle input
θ	degrees	Delay-filtered throttle angle
$\hat{\theta}$	-	O/P of Throttle polynomial
ω	rad/sec	Engine speed
F_c	g/s	Commanded fuel

Table 1: States and Intermediate Variables

To summarize, a modular view of the plant model is considered with the following interface specifications, and main internal states:

– *Exogenous Inputs*: Throttle Angle (θ_{in}), Engine Speed (ω).
– *Inputs from Controller*: Fuel command (F_c).
– *Outputs to Controller*: Measured inlet air mass flow rate ($\dot{m}_{af}$) and the measured A/F ratio (λ_m).
– *Continuous-valued States*: Intake Manifold pressure (p), two states associated with (7), the state associated with the filter block for the throttle input (θ), fuel stored in the fuel film (m_f), and states associated with the variable delay[2].

Controller. The controller is a sampled-time subsystem with two high-level subsystems: 1.) a Proportional + Integral (PI) feedback controller, 2.) a feedforward controller based on an estimate of the air mass entering the cylinder. The controller is compliant with a standard published by the MathWorks automotive advisory board (MAAB), which is used by the automotive industry. We used Version 3.0 of the MAAB standard [26].

The feedforward controller estimates the rate of air flow into the cylinder based on a measurement of the inlet air mass flow rate ($\dot{m}_{af}$). In a real system, such an observer is a carefully designed system that compensates for delays and noise in the plant model; for example an extended Kalman filter. For simplicity, we choose an "almost perfect" observer, i.e., we assume almost perfect knowledge of the pumping polynomial (modulo some multiplicative error factor) to observe state p (intake manifold pressure), and use the observer state (denoted p_e) to compute the estimated air mass flowing into the cylinder. The feedback PI controller regulates the A/F ratio in closed-loop, and uses the measured A/F ratio to compute the fuel command (F_c). The controller has four modes of operation, that we now detail.

1. *Startup mode*: Standard O_2 sensors produce accurate measurements only when they reach a particular operating temperature; until then the controller operates in an

[2] Strictly speaking, a variable system delay requires an infinite number of states to model precisely.

"open-loop" mode, i.e., with only feedforward control. In lieu of modeling the heating dynamics, we model this phenomenon with a timer. The timer limit is assumed to be a range, and once the timer expires, the controller changes mode to a normal mode of operation.
2. *Normal mode*: In this mode, the controller uses both feedback PI control and feedforward control to regulate the A/F ratio.
3. *Power enrichment mode*: This mode represents a situation where the user provides a wide throttle angle (i.e., by depressing the gas pedal more than a certain threshold). In this mode, to satisfy the power and torque demands, the controller uses only feedforward control, and disables feedback correction until the throttle angle is reduced to below a certain threshold. Also, the desired A/F setpoint is adjusted from the standard value of 14.7 down to 12.5.
4. *Fault mode*: This mode represents one or more sensor failures. Again, in this mode the controller switches to open-loop control. Once entered, the system remains in this mode.

Note that in a real system, it is certainly possible for the system to transition to power enrichment mode directly from the startup mode or to encounter a fault in startup mode. We choose not to model these behaviors for simplicity in modeling and analysis. To summarize, the controller can be viewed as a module with the following inputs, outputs and internal states:

– *Inputs from Plant (sensors)*: Inlet air mass flow rate measurement ($\dot{m}_{af}$), Measured A/F ratio (λ_m).
– *Exogenous Inputs*: Throttle angle (θ), Engine speed (ω), Sensor failure event ($fail\_event$).
– *Outputs to Plant*: Fuel command (F_c).
– *States*: Estimate of the rate of air mass pumped into the cylinder (p_e), Integrator state for the PI controller (i), and fuel command state (F_c).

Error Factor Correction. A constant error factor, c_{24}, is included in the oxygen sensor measurement. Also, the inlet air mass flow rate $\dot{m}_{af}$ is measured by the controller; a constant error factor, c_{23}, is included in this measurement. The fuel command produced by the controller and the actual fuel produced by the actuator may be different due to actuator error; a constant fuel injector actuator error c_{25} is included to account for this.

3.2 Hybrid I/O Automaton Model

In this section, we present a simplification of the system dynamics described in Sec. 3.1. Ideally, we would like to perform analysis, including formal verification, on the system introduced in Sec. 3.1, but limitations in existing verification techniques for such systems prevent this. Hence, we present a hybrid automaton version of the system more amenable to extant analysis techniques.

Plant HIOA. Certain aspects of the model described in Sec. 3.1 can present difficulties to some analysis tools, such as the transport delays and LUTs. To address these concerns, this version of the model presents several significant simplifications. We remove the wall wetting subsystem entirely. For the exhaust subsystem, we approximate the variable transport delay (which is accurately described using a delay-differential equation), the sensor dynamics and the transport dynamics by a single first order filter. Thus, we simplify the dynamics of the A/F ratio to the following equation:

$$\frac{d\lambda}{dt} = c_{26}\left(\lambda_c - \lambda\right). \tag{10}$$

Substituting (3) into (6) and using the fuel command from the controller, F_c, modified by the fuel tolerance factor, c_{25}, we get

$$\frac{d\lambda}{dt} = c_{26}\left(\frac{c_{12}\cdot\left(c_2 + c_3\omega p + c_4\omega p^2 + c_5\omega^2 p\right)}{c_{25}\cdot F_c} - \lambda\right). \tag{11}$$

We leave the dynamics of the intake manifold pressure (p) unchanged. In effect, the plant dynamics can be modeled by a single mode system, with the continuous states p and λ, where $\mathbf{x}_p = [p\ \lambda]^T$ evolves according the to ODE $\dot{\mathbf{x}}_p = f(\mathbf{x}_p)$, where f is the multi-valued function given by (5) and (11).

We skip the tuple definition of the plant HIOA for brevity. As it has only one discrete mode, the sets Δ_P, $\mathcal{G}_P$, $\mathcal{R}_P$ are empty. The rest of the model (input variables, state variables, output variables, states[3]) are as specified in the modular definition of the plant in Sec. 3.1, except for states that were eliminated. The plant HIOA has a single initial state given by $(\theta \mapsto 8.8°, p \mapsto 0.9833, \lambda \mapsto 14.7)$.

Controller HIOA. The controller HIOA has four discrete modes, and the transition structure is as shown in Fig. 1. The controller HIOA tuple is defined as follows:
– The set of modes $\mathcal{L}_C$ has four modes ($\texttt{startup}, \texttt{normal}, \texttt{power}, \texttt{sensor\_fail}$),
– The set of state variables $\mathcal{X}_C$ is $\{p_e, i, \tau, F_c\}$. We use $\mathbf{x}_c$ as shorthand to represent the tuple of the state variables.
– The set of input variables $\mathcal{U}_C$ is $\{\dot{m}_{af}, fail\_event, \lambda\}$,
– As the controller is a discrete-time system, for any trajectory in $\mathcal{T}$, the flow function for the trajectory in any mode is described by the ODE $\dot{\mathbf{x}}_c = 0$.
– For each mode ℓ, $\mathcal{M}$ maps ℓ to the negation of the conjunction of all the guards on its outgoing transitions.
– The set of guards is $\{fail\_event = true, \tau = \tau_I, \theta \leq 50°, \theta \geq 70°\}$.
– The set of reset functions is a union of two functions $g_c(\cdot)$ and $g_o(\cdot)$. We define the actual functions below.
– The transitions are as depicted in Fig. 1.
– The set of output variables is $\{F_c\}$.
– The set of initial states is the singleton set: $\{(\texttt{startup}, p_e \mapsto 0, i \mapsto 0, \tau \mapsto 0, F_c \mapsto 0.6537)\}$.

We now elaborate on the discrete-time update equations that appear in the reset transitions in Fig. 1. In what follows, h denotes the sample period for the controller, and we use the notation $x[k]$ to mean the value of continuous-time variable x at time $t = kh$, and $k \in \mathbb{Z}_{\geq 0}$ is a sample number.

[3] Technically, $\dot{m}_{af}$ is not a state in the plant, but is defined as an output variable. To make the plant HIOA well-defined, we can introduce a dummy state in the plant, and allow only those trajectory functions in $\mathcal{T}$ that are consistent with equation 2.

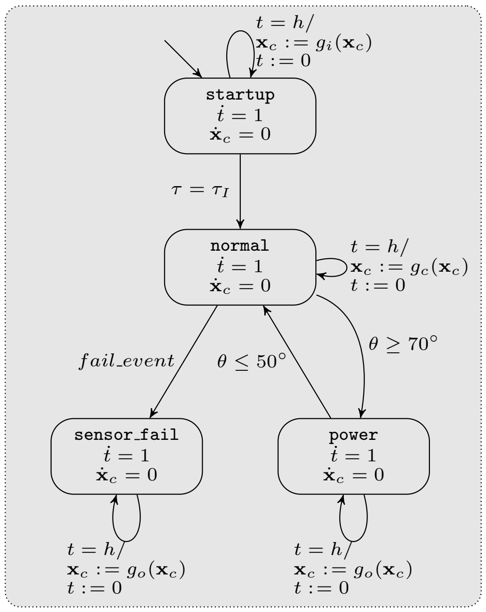


Fig. 1: Hybrid I/O Automaton $\mathcal{A}_c$ modeling the controller. The HIOA $\mathcal{A}_p$ for the plant model is a single discrete mode automaton with the ODE $\dot{\mathbf{x}}_p = f(\mathbf{x}_p)$, and we omit its depiction for brevity. Note that the state t in $\mathcal{A}_c$ is not an actual state in the respective models, but is an artifact of modeling a discrete-time system using a HIOA formalsim.

The quantity p_e denotes the estimated manifold pressure. The dynamics for the estimated manifold pressure are:

$$p_e[k+1] = p_e[k] + hc_1(c_{23} \cdot \dot{m}_{af} - \big(c_2 + c_3\omega p_e[k] + c_4\omega p_e[k]^2 + c_5\omega^2 p_e[k]\big)). \tag{12}$$

Closed-loop discrete dynamics. In the `normal` mode, which is the common operating mode, the controller uses feedback control. The dynamics for the PI controller state and the fuel command are given by

$$i[k+1] = i[k] + hc_{14}(c_{24}\lambda[k] - c_{11}), \tag{13}$$

$$F_c[k+1] = \frac{1}{c_{11}}\left(1 + i[k] + c_{13}(c_{24}\lambda[k] - c_{11})\right) \cdot \big(c_2 + c_3\omega p_e[k] + c_4\omega p_e[k]^2 + c_5\omega^2 p_e[k]\big). \tag{14}$$

Note that the timer is not used in this mode, thus the corresponding update equation is

$$\tau[k+1] = 0.0. \tag{15}$$

The update function g_c is a multi-valued function consisting of the RHSs of equations (12), (13), and (15).

Open-loop discrete dynamics. In the `power` and `sensor_fail` modes, the signal from the O_2 sensor is ignored, and the controller uses only feedforward control, hence the state of the integrator in the PI control does not change. Note that the timer is not used in this mode. Also, in `power` mode, we use $c_{11} = 12.5$, and in `sensor_fail` mode we use $c_{11} = 14.7$. The update equations for the integrator state and the fuel command are given as follows:

$$\begin{aligned} F_c[k{+}1] &= \frac{1}{c_{11}}\left(c_2 + c_3\omega p_e[k] + c_4\omega p_e[k]^2 + c_5\omega^2 p_e[k]\right), \\ i[k{+}1] &= 0.0, \\ \tau[k{+}1] &= 0.0. \end{aligned} \tag{16}$$

Thus, the update function g_o is a multi-valued function consisting of the RHSs of equations (12) and (16).

Startup mode dynamics. In the `startup` mode, in addition to having open-loop control, the controller also uses the timer to count up to τ_I seconds. The update function g_i consists of the the first two update equations from (16), and $\tau[k{+}1] = \tau[k] + h$.

Controller transitions. As shown in Fig. 1, the controller starts in the `startup` mode using open-loop dynamics, and after τ_I seconds, enters the `normal` mode of operation. If at any time, the effective throttle angle input (θ) is greater than $70°$, the controller switches to the open-loop dynamics, and when θ drops below $50°$, it switches back to the closed-loop dynamics. In the `normal` mode if there is a sensor failure event detected, the controller switches to the `sensor_fail` mode, and remains there. In `sensor_fail` mode it again uses open-loop dynamics.

Closed-loop HIOA. The closed-loop HIOA is obtained by parallel composition of the plant HIOA and the controller HIOA. As some of the interface variables are eliminated during the composition, the closed-loop HIOA has fewer input and output variables. The closed-loop HIOA has the same basic structure as the HIOA for the controller, except each state also contains the plant dynamics. Further, the update equations and the dynamics do not contain the interface variables, as for each output variable of the plant (resp. controller), the corresponding output variable of the plant (resp. controller) is substituted in the respective input variable of the controller (resp. plant). The final closed-loop HIOA has 6 states (θ, p, λ, p_e, i, F_c), three exogenous inputs (θ_{in}, ω, $fail\_event$), and no outputs.

3.3 PHA model

The model presented in Sec. 3.2 contains features that present unique challenges to hybrid systems verification tools. Specifically, the dynamic equations are not explicitly expressed as polynomial equations. We now present a simplified version of the system, where the continuous-valued dynamics are restricted to the class of polynomials and are converted from mixed continuous/discrete time to ODEs. Such a hybrid system can be analyzed by some tools, including Flow* [9]. We also eliminate the fuel command state in the controller, as the purpose of that state is simply to implement a zero-order hold, which is not required for a model with fully continuous dynamics in each mode (with no resets).

We now present a PHA model of the closed-loop fuel control problem, obtained by applying a host of simplification techniques to the model in Sec. 3.2. We approximate the rational function in (11) with the following second order polynomial function, which is accurate in the range $1.0 \leq \dot{m}_c \leq 20.0$ and $0.5 \leq F_c \leq 1.2$:

$$\begin{aligned}\frac{d}{dt}\lambda \approx c_{26}(c_{15} + c_{16}c_{25}F_c + c_{17}c_{25}^2F_c^2 + \\ c_{18}\dot{m}_c + c_{19}\dot{m}_c c_{25}F_c - \lambda).\end{aligned} \tag{17}$$

Next, we approximate the square root function in (2) with a polynomial, which is accurate in the range $0.5 \leq p \leq 1.0$:

$$\sqrt{\frac{p}{c_{10}} - \left(\frac{p}{c_{10}}\right)^2} \approx c_{20}p^2 + c_{21}p + c_{22}. \tag{18}$$

Applying these substitutions, and using continuous-time versions of the discrete-time update equations, we obtain the following alternative system representation:

$$\begin{aligned}
\tfrac{d}{dt}\theta &= 10(\theta_{in} - \theta) \\
\tfrac{d}{dt}p &= c_1(2\hat{\theta}\left(c_{20}p^2 + c_{21}p + c_{22}\right) - \\
&\qquad c_{12}\left(c_2 + c_3\omega p + c_4\omega p^2 + c_5\omega^2 p\right)) \\
\tfrac{d}{dt}\lambda &= c_{26}(c_{15} + c_{16}c_{25}F_c + c_{17}c_{25}^2F_c^2 + c_{18}\dot{m}_c + \\
&\qquad c_{19}\dot{m}_c c_{25}F_c - \lambda) \\
\tfrac{d}{dt}p_e &= c_1(2c_{23}\hat{\theta}\left(c_{20}p^2 + c_{21}p + c_{22}\right) - \\
&\qquad \left(c_2 + c_3\omega p_e + c_4\omega p_e^2 + c_5\omega^2 p_e\right)) \\
\tfrac{d}{dt}i &= c_{14}(c_{24}\lambda - c_{11}),
\end{aligned} \tag{19}$$

where F_c is given by:

$$\begin{aligned}F_c = \tfrac{1}{c_{11}}\left(1 + i + c_{13}(c_{24}\lambda - c_{11})\right) \cdot \\ \left(c_2 + c_3\omega p_e + c_4\omega p_e^2 + c_5\omega^2 p_e\right),\end{aligned} \tag{20}$$

and $\dot{m}_c$ is given by (3).

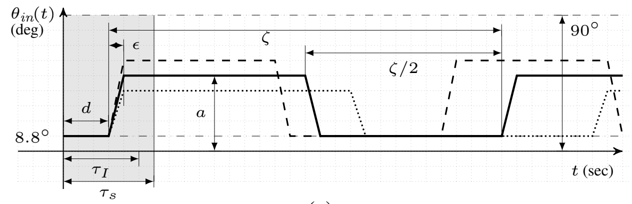
Again, in any mode with open loop dynamics, the air-fuel ratio signal measurement is not used by the controller, and it only uses feedforward control. The ODEs for states p, λ, and p_e remain as in (19), and the updates for F_c and i are given by the following:

$$F_c = \frac{1}{c_{11}}\left(c_2 + c_3\omega p_e + c_4\omega p_e^2 + c_5\omega^2 p_e\right) \tag{21}$$

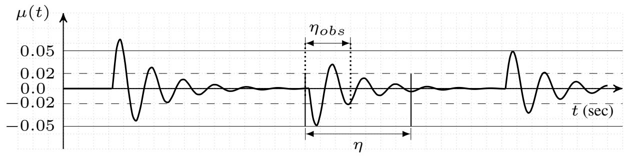
$$\frac{d}{dt}p_e = 0.0. \tag{22}$$

4 Requirements

In this section, we present typical formal requirements for an A/F ratio control problem. These requirements were formulated by by surveying the A/F ratio control design



(a)



(b)

Fig. 2: (a) shows a pulse train input (with period ζ, amplitude a, rise time ϵ, and delayed by time d) depicted by a solid line. The dashed and dotted pulse trains lines illustrate other inputs that match the profile. (b) shows the output $\mu(t)$ when the system is excited by the input depicted by the solid line. η_{obs} denotes the observed settling time for the input, while η represents the maximum settling time as specified by requirement (27). The dashed lines denote the settling region, and the solid lines parallel to the dashed lines denote the maximum permitted undershoot/overshoot.

literature (cf. [10, 29]). We express requirements in STL; please see Sec. 2 for its syntax and semantics. In the sequel, we assume that the simulation time horizon is denoted by the symbol T.

Requirement Format. The expected behavior of most industrial control systems is usually specified with respect to a set of input profiles. Informally, an input profile is a parameterized representation of a possibly infinite set of input signals. For all the requirements described here, we assume the following input profiles: (1) the engine speed (ω) signal profile is the set of constant signals with amplitude in $[900, 1100]$ rpm (approx. $[94.25, 115.19]$ rad/sec), (2) the throttle angle (θ_{in}) signal profile is the set of pulse train signals (with pulse width equal to half the period). We use $\texttt{pulse}(a, \zeta, d)$ as shorthand to symbolically denote a class of pulse train signals, such as those shown in Fig. 2, where a is the pulse amplitude, ζ is the period ($\zeta_{min} < \zeta < \zeta_{max}$) in seconds, and d is some initial delay in seconds before the first pulse (to allow for effects of transients due to initial conditions to dissipate). Here, ζ_{min}, ζ_{max}, and d are fixed constants.

We assume that an input profile can be specified using an STL formula φ_I. For instance, the θ_{in} input profile can be defined using events $\texttt{rise}(a)$ and $\texttt{fall}(a)$ (23 and 24). The `rise` event specifies that within time ϵ, the throttle angle rises from $8.8°$ to some value a (where $8.8° < a$), and similarly the `fall` event specifies a falling transition. The time ϵ represents a small unit of time, typically comparable to the the time-step selected by the numerical integration solver.

$$\texttt{rise}(a) \equiv (\theta = 8.8^\circ) \wedge \Diamond_{(0,\epsilon)}(\theta = a) \tag{23}$$

$$\texttt{fall}(a) \equiv (\theta = a) \wedge \Diamond_{(0,\epsilon)}(\theta = 8.8^\circ) \tag{24}$$

Let $\mathbf{u}$ denote the input signals, and $\mathbf{y}$ denote the output signals of the model. A behavioral I/O requirement on the closed-loop model specifies that for a signal u satisfying the input profile φ_I from time 0 onwards, the corresponding output y must satisfy the output requirement φ_O, i.e., $(\mathbf{u}, 0) \models \varphi_I \;\Rightarrow\; (\mathbf{y}, 0) \models \varphi_O$. In what follows, we omit the input portion of the requirements, only presenting the requirements on the output signals. For convenience, we define a normalized error signal μ that indicates the error in λ from the reference stoichiometric value λ_{ref}.

$$\mu(t) = \frac{\lambda(t) - \lambda_{ref}}{\lambda_{ref}} \tag{25}$$

Requirements in the `normal` **mode.** For all the requirements in `normal` mode, we use the input profile $\texttt{pulse}(a, \zeta, d)$ and constrain a such that $8.8^\circ < a < 70^\circ$, where 70° is the value of θ_{in} that triggers a transition to the `power` mode. Also, we do not allow any failure event to occur.

Transient and Steady-state behavior: Measuring both the steady-state and the transient response of the control system with periodic pulse inputs is important. If the control system's response time is not fast enough, then the pulse events can occur before the transients have dissipated, leading to insufficient performance or even instability. Also the rising and falling edges of the pulse signal can trigger overshoots and undershoots in the controlled signal. The first requirement (26) specifies the maximum permitted overshoot or undershoot. Note that the requirement does not include behavior over the initial time interval of $(0, \tau_s)$ (where $\tau_s = \tau_I + \eta_{S \to N}$). This is to exclude behaviors in the startup mode for the first τ_I seconds, and then to allow the controller to settle after transitioning into the `normal` mode. For our models, we use τ_I = 10 secs, $\eta_{S \to N}$ = 1 second. The STL requirement (27) specifies that after each rising or falling edge of the pulse, the signal settles within η seconds, and remains within the settling region (that we define as $\lambda_{ref} \pm 0.02\lambda_{ref}$) until the next rising or falling edge of the pulse can occur. (See Fig. 2(b) for an illustration of these requirements).

$$\Box_{(\tau_s, T)} \, |\mu| < 0.05 \tag{26}$$

$$\Box_{(\tau_s, T)} \left(\texttt{rise}(a) | \texttt{fall}(a) \Rightarrow \Box_{(\eta, \frac{\zeta}{2})} \, |\mu| < 0.02 \right) \tag{27}$$

Error tolerance: An important quantity for a control system is the error between the desired setpoint and the controlled signal. A standard way of measuring this error is by taking the *root mean square* (RMS) value of the error over the period of time when the controller is in the `normal` mode. We first define the RMS error accumulated over time t as a signal in (28), where $u(\tau)$ is the Heaviside step function that masks the error in the initial τ_I seconds where the system is not in the `normal` mode. Here, the calculation starts at τ_I to include the transient error introduced by the transition from the `startup` mode to the `normal` mode.

$$\mathbf{x}_{rms}(t) = \sqrt{\frac{1}{t-\tau_I}\int_0^t (\lambda(\tau) - \lambda_{ref})^2 u(\tau - \tau_I) d\tau} \tag{28}$$

Recall that $(\mathbf{x}, 0) \models \Diamond_{[T,T]}\varphi$ means that the property φ holds for $(\mathbf{x}, 0)$ exactly at time T. Thus, requirement (29) states that the RMS error at time T (i.e., the accumulated error over the duration (τ_I, T)) is less than some value c.

$$\Diamond_{[T,T]}\ \mathbf{x}_{rms} < c \tag{29}$$

Worst-case excursions in the `normal` *mode:* The performance of the controller is sensitive to the accuracy of sensors and actuators. At extremal tolerance values, we require that the worst-case excursions in the A/F ratio remain within a given range around the stoichiometric value. We identified two extreme scenarios corresponding to certain fixed values of the sensor tolerances c_{23} and c_{24} and the actuator error c_{25}.

In the first case, the following set of extremal tolerances: $\{c_{23} = 1.05, c_{24} = 1.01, c_{25} = 1.05\}$, gives rise to a steady-state error in λ, where λ settles to a setpoint that corresponds to the A/F mixture being fuel-rich (i.e., $\mu < 0$). Thus, this set of tolerances leads to the worst case behavior for the undershoot on μ, and requirement (30) specifies the worst-case permitted undershoot $-c_r$.

In the second case, the set of tolerances $\{c_{23} = 0.95, c_{24} = 0.99, c_{25} = 0.95\}$, gives rise to a similar steady-state error in λ; however, here λ settles to a lean A/F mixture (i.e., $\mu > 0$). Thus, this set of tolerances leads to the worst case overshoot on μ, and requirement (31) specifies the worst-case permitted overshoot c_l.

$$\Box_{(\tau_s,T)}\ \mu > -c_r \tag{30}$$

$$\Box_{(\tau_s,T)}\ \mu < c_l \tag{31}$$

Transitioning out of the `power` *mode:* As λ in the `power` mode is regulated in an open-loop fashion and has a different set-point, we want to ensure that the controller is able to settle to a value close to λ_{ref} within a specified time after switching back from the `power` mode. The input profile $\texttt{pulse}(a, \zeta, d)$ for this requirement constrains a differently; the constraint used here is $8.8° \leq a \leq 90°$. This allows the controller to transition to the `power` mode and stay in that mode for approximately $\frac{\zeta}{2}$ seconds. Requirement (32) characterizes this settling-time requirement.

$$\Box_{(\tau_s,T)} \begin{pmatrix} \ell{=}\,\texttt{power} \wedge \Diamond_{(0,\epsilon)}\ell{=}\,\texttt{normal} \Rightarrow \\ \Box_{(\eta,\frac{\zeta}{2})}\ |\mu| < 0.02 \end{pmatrix} \tag{32}$$

Requirements in the `power` **mode.** In the `power` mode, the feedforward control is expected to provide a fuel-rich air mixture, i.e., the setpoint for the A/F ratio is λ_{ref}^{pwr} = 12.5. Let $\mu_p(t)$ define the error signal in `power` mode, where λ_{ref} in equation (25) is replaced by λ_{ref}^{pwr}. The input profile for θ_{in} is $\texttt{pulse}(a, \zeta, d)$, where a is constrained such that $70^\circ \leq a \leq 90^\circ$. Such a pulse input maintains the controller in the `power` mode after the initial delay of τ_s seconds. The transient and steady-state requirements for `power` mode are similar in form to the ones for the `normal` mode; however, as there is only open-loop control in this mode, the requirements on λ are typically relaxed. We omit the requirement on settling time for brevity, and in (33) specify that the maximum undershoot/overshoot on λ is within 20% of λ_{ref}^{pwr}.

$$\square_{(\tau_s,T)} \left(\ell = \texttt{power} \Rightarrow |\mu_p| < 0.2\right) \tag{33}$$

Requirement in the `startup` and `sensor_fail` mode. When the controller is in the `startup` or `sensor_fail` mode, the controller uses open-loop control. The requirements on λ are similar in form to the ones in `normal` mode, but with relaxed bounds on the settling region and transient excursions (we omit the latter for brevity). The input profile $\texttt{pulse}(a, \zeta, d)$ has similar constraints on a as the one for requirements (26),(27). Further, we force a sensor failure event to occur at 15 seconds.

$$\square_{(0,T)}\left(\begin{pmatrix}\ell = \texttt{startup}|\texttt{sensor\_fail} \\ \wedge\ \texttt{rise}(a)|\texttt{fall}(a)\end{pmatrix} \Rightarrow \square_{(\eta,\frac{\zeta}{2})} |\mu| < 0.1\right) \tag{34}$$

5 Analysis: Case study

Table 2: Experimental results of falsification analysis with S-TaLiRo.

		Model 1 (Sec. 3.1)			Model 2 (Sec. 3.2)			Model 3 (Sec. 3.3)		
Req.	Parameters	Time (sec)	Sim.	Falsified?	Time (sec)	Sim.	Falsified?	Time (sec)	Sim.	Falsified?
(26)	$\eta = 1$	5173.27	1000	no	1026.57	1000	no	1569.81	1000	no
(27)	$\eta = 1$	5421.03	1000	no	1097.09	1000	no	1727.83	1000	no
(29)	$c = 0.05$	4188.39	1000	no	1044.54	1000	no	1399.54	1000	no
(30)	$c_r = 0.1$	5296.41	1000	no	1058.50	1000	no	1586.46	1000	no
(31)	$c_l = 0.1$	5589.76	1000	no	1061.15	1000	no	1560.16	1000	no
(32)	$\eta = 1$	5.44	1	yes	1.27	1	yes	1.73	1	yes
(33)	$\lambda_{ref}^{pwr} = 12.5$	4193.46	1000	no	996.58	1000	no	1459.73	1000	no
(34)	$\eta = 1$	3893.06	1000	no	1019.28	1000	no	1521.63	1000	no

In this section, we provide the results of performing falsification analysis on the three models presented in Sec. 3 with the tool S-TaLiRo. Given the requirements in the previous section[4], S-TaLiRo searches each model for the falsifying counterexam-

[4] Implementing requirements (27), (32), and (34) in S-TaLiRo is difficult, as strictly speaking each is not a single requirement, but specifies a class of requirements for varying values of ζ. For any fixed value of ζ, each requirement is indeed an STL formula. But, we allow S-TaLiRo to choose ζ as an optimization variable, which changes the requirement in each iteration of S-TaLiRo; this is not currently supported. In the actual analysis, we thus use a relaxed form of the requirement, where the term $\frac{\zeta}{2}$ is replaced by $\frac{\zeta_{min}}{2}$.

ple. Table 2 lists the requirement under consideration, the parameter values used, total run-time, number of simulations, and indicates if S-TaLiRo was able to find a counterexample. When testing the requirements, we use the following values for the input profile parameters: $\zeta_{min} = 10$, $\zeta_{max} = 30$, $d = 3$, and $\epsilon = 0.02$.

Conclusions. We present a brief analysis of the experimental results presented in Table 2. The results indicate that each requirement is either falsified by S-TaLiRo, or S-TaLiRo exhausted a given simulation-run budget (1000 simulations) and was unable to falsify the requirement. S-TaLiRo is unable to falsify most of the properties (either transient or steady-state), which may indicate that the quality of the manual abstractions that we performed vis-à-vis high-level requirements is reasonable. However, this can be confirmed only after more rigorous conformance checking.

The tables also indicate that the requirement related to the transition out of the `power` mode (32) is easily falsified for all three models. For all of the simulations explored by S-TaLiRo, once the system transitions from `power` mode back to `normal`, we observe that the normalized error never remains within the designated error bound within the specified settling time η. This is due to the large step in θ_{in} that is applied and also because the λ_{ref} changes instantaneously from 12.5 to 14.7 at the instant of the transition, which causes a significant transient behavior. It is a challenge to design the controller to behavior well in this scenario. Finally, we wish to remark that the requirements expressed in this paper are meant to convey the flavor of how desirable design behavior may be expressed, and should not be interpreted as formal requirements that are valid for actual industrial models. That said, in future work, we aspire to produce a comprehensive suite of requirements that would comprise a benchmark suite to certify desired behavior.

6 Analysis: Challenge Problems

Abstraction Techniques. We presented three models: the first created by control design experts, and the second and third obtained by successive simplifications. The simplified models represent manual abstractions with no guarantees on fidelity across model versions. Transformations with formal guarantees would be preferable; especially techniques for transforming Simulink models into formal representations such as hybrid automata. This is a stiff challenge as the semantics of the Simulink language are obscure. For continuous-valued aspects of a Simulink model, we can perform numerical linearizations automatically, but the resulting models are only valid near some operating point. Existing works to transform Simulink models to hybrid automata typically focus on small fragments or focus on discrete-time models [1, 25].

Reachable Set Estimation. Bounded-time reachability analysis techniques for hybrid systems try to obtain overapproximations of the set of reachable states of the system over a fixed time horizon, which are then used to prove system safety. Tools that address this problem continue to mature [2, 3, 9, 16]. The SpaceEx tool [18] provides algorithms currently limited to affine hybrid systems, while Flow* [9] can reason about a larger class of dynamical systems by using higher order approximations. These and other tools in this space face fundamental challenges related to the curse of dimensionality and approximation error with its concomitant false positives.

Certain features of our models, for example the LUTs, exacerbate issues faced by reachability analysis tools. Precise analysis of LUTs requires that each grid element of the LUT be treated as a system mode, typically worsening the conservativeness of the analysis. This is significant, as models of industrial systems often contain LUTs. The variable transport delays indicate that system dynamics are given by a delay differential equation (DDE); DDEs are not handled directly by current reachable set estimation techniques. Finally, a practical limitation is that verification tools require systems to be described in tool-specific formats.

Conformance Checking. As seen in this paper, we often have models representing varying levels of system abstraction. In addition to manual abstractions created for formal analysis, model refinements naturally occur during the development process. Refined models usually include implementation details, e.g., controller output saturation and fixed-point number formats. The general notion of checking behavioral proximity of models is sometimes called *conformance* checking. What it means for two models to be conformant is itself an open problem, as notions of logical equivalence are often inadequate in a hybrid setting. Previous work has focused on notions such as *bisimulation equivalence* [19] using *bisimulation functions*. Existence of a bisimulation function is a property of a model's behaviors over an infinite time horizon, and could be a much stronger notion of conformance than required. On the other hand, bisimulation relations may not be a sufficient comparison; for example, there are examples where a stable system is bisimilar to an unstable system [27]. A possible direction of research is to use hybrid distance metrics [8] to establish conformance.

Stability proofs. Lyapunov techniques have been used for proving system stability [22]. A critical requirement of such analyses is a Lyapunov function. The problem of finding Lyapunov functions for systems with nonpolynomial, hybrid dynamics, such as those described in this paper is open. Furthermore, systems such as those shown in this paper may not have an analytic form of the dynamics for proprietary components, or may have features such as delays and look-up tables. Obtaining Lyapunov functions for both stability proofs and more general verification tasks is a challenge.

References

1. A. Agrawal, G. Simon, and G. Karsai. Semantic translation of simulink/stateflow models to hybrid automata using graph transformations. *Electronic Notes in Theoretical Computer Science*, 109:43–56, 2004.
2. M. Althoff. Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In *Proceedings of Hybrid Systems: Computation and Control*, pages 173–182, 2013.
3. M. Althoff, O. Stursberg, and M. Buss. Computing reachable sets of hybrid systems using a combination of zonotopes and polytopes. *Nonlinear Analysis: Hybrid Systems*, 4(2):233–249, 2010.
4. R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

5. R. Alur, T. Henzinger, and P.-H. Ho. Automatic symbolic verifcation of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, March 1996.
6. Y. Annapureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan. S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems*, pages 254–257, 2011.
7. G. Behrmann, R. David, and K. G. Larsen. A tutorial on UPPAAL. In *Formal Methods for the Design of Real-time Systems*, pages 200–236, 2004.
8. P. Caspi and A. Benveniste. Toward an approximation theory for computerised control. In *Proceedings of 2nd International Conference on Embedded Software*, pages 294–304, 2002.
9. X. Chen, E. Abraham, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Proceedings of Computer Aided Verification*, pages 258–263, 2013.
10. J. A. Cook, J. Sun, J. H. Buckland, I. V. Kolmanovsky, H. Peng, and J. W. Grizzle. Automotive powertrain control - a survey. *Asian Journal of Control*, 8:237–260, 2006.
11. P. R. Crossley and J. A. Cook. A nonlinear engine model for drivetrain system development. In *International Conference on Control*, volume 2, pages 921–925, 1991.
12. A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *Proceedings of Formal modeling and analysis of timed systems*, pages 92–106, 2010.
13. A. Donzé, O. Maler, E. Bartocci, D. Nickovic, R. Grosu, and S. A. Smolka. On temporal logic and signal processing. In *Proceedings of Automated Technology for Verification and Analysis*, pages 92–106, 2012.
14. A. Eggers, N. Ramdani, N. Nedialkov, and M. Fränzle. Improving SAT modulo ODE for hybrid systems analysis by combining different enclosure methods. In *Proceedings of Software Engineering and Formal Methods*, pages 172–187, 2011.
15. A. Fehnker and F. Ivancic. Benchmarks for hybrid systems verification. In *Proceedings of Hybrid Systems: Computation and Control*, pages 326–341, 2004.
16. G. Frehse. PHAVer: algorithmic verification of hybrid systems past HyTech. *International journal on Software Tools for Technology Transfer*, 10(3):263–279, 2008.
17. G. Frehse, Z. Han, and B. Krogh. Assume-guarantee reasoning for hybrid i/o-automata by over-approximation of continuous interaction. In *Proceedings of IEEE Conf. on Decision and Control*, volume 1, pages 479–484, 2004.
18. G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. Spaceex: Scalable verification of hybrid systems. In *Proceedings of Computer Aided Verification*, pages 379–395, 2011.
19. A. Girard and G. J. Pappas. Approximate bisimulation: A bridge between computer science and control theory. *European Journal of Control*, 17(5-6):568–578, 2011.
20. L. Guzzella and C. Onder. *Introduction to Modeling and Control of Internal Combustion Engine Systems*. Springer-Verlag, 2nd edition edition, 2010.
21. T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's Decidable about Hybrid Automata? *Proceedings of the ACM Symposium on Theory of Computing*, 57(1):94 – 124, 1998.
22. H. Khalil. *Nonlinear Systems*. Prentice Hall PTR, 2002.
23. R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
24. N. Lynch, R. Segala, and F. Vaandrager. Hybrid I/O automata. *Information and Computation*, 185(1):105 – 157, 2003.
25. K. Manamcheri, S. Mitra, S. Bak, and M. Caccamo. A step towards verification and synthesis from Simulink/Stateflow models. In *Hybrid Systems: Computation and Control*, pages 317–318, 2011.
26. Mathworks Automotive Advisory Board. Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow, 2012.

27. P. Prabhakar, G. E. Dullerud, and M. Viswanathan. Pre-orders for reasoning about stability. In *Proceedings of Hybrid Systems: Computation and Control*, pages 197–206, 2012.
28. S. Sankaranarayanan and G. E. Fainekos. Falsification of temporal properties of hybrid systems using the cross-entropy method. In *Proceedings of Hybrid Systems: Computation and Control*, pages 125–134, 2012.
29. A. A. Stotsky. *Automotive Engines: Control, Estimation, Statistical Detection*. Springer, 2009.
30. The MathWorks, Inc. *Simulink User's Guide*. Natick, MA, 2012.

Symbol	Value	Description
c_1	0.41328	Constant from Ideal Gas Law
c_2	-0.366	Coefficient for Pumping polynomial
c_3	0.08979	Coefficient for Pumping polynomial
c_4	-0.0337	Coefficient for Pumping polynomial
c_5	0.0001	Coefficient for Pumping polynomial
c_6	2.821	Coefficient for $\hat{\theta}$ polynomial
c_7	-0.05231	Coefficient for $\hat{\theta}$ polynomial
c_8	0.10299	Coefficient for $\hat{\theta}$ polynomial
c_9	-0.00063	Coefficient for $\hat{\theta}$ polynomial
c_{10}	1.0	Atmospheric pressure (bar)
c_{11}	14.7/12.5	$\lambda_{ref}/\lambda_{ref}^{pwr}$
c_{13}	0.04	Proportional gain for controller
c_{14}	0.14	Integral gain for controller

Table 3: Model Parameters

n	m_c	$1-\kappa(\cdot)$	$\tau(\cdot)$
1000	0.1	0.80	0.40
1500	0.1	0.70	0.30
2000	0.1	0.70	0.35
2500	0.1	0.80	0.30
3000	0.1	0.90	0.20
1000	0.2	0.70	0.22
1500	0.2	0.66	0.22
2000	0.2	0.65	0.40
2500	0.2	0.73	0.35
3000	0.2	0.85	0.50
1000	0.3	0.66	0.20
1500	0.3	0.66	0.22
2000	0.3	0.63	0.50
2500	0.3	0.66	0.40
3000	0.3	0.80	0.35
1000	0.4	0.60	0.35
1500	0.4	0.60	0.30
2000	0.4	0.60	0.45
2500	0.4	0.60	0.50
3000	0.4	0.70	0.40

(a) LUTs for $1-\kappa(\cdot)$ and $\tau(\cdot)$

n	m_c	Delay
800	0.05	0.25
1000	0.05	0.20
1500	0.05	0.20
2000	0.05	0.20
3000	0.05	0.20
800	0.15	0.30
1000	0.15	0.25
1500	0.15	0.20
2000	0.15	0.20
3000	0.15	0.20
800	0.20	0.40
1000	0.20	0.30
1500	0.20	0.20
2000	0.20	0.20
3000	0.20	0.20
800	0.25	0.80
1000	0.25	0.60
1500	0.25	0.40
2000	0.25	0.30
3000	0.25	0.20

(b) Delay LUT

Table 4: Look-up Tables

Symbol	Value	Description
c_{15}	13.893	Coefficient for A/F polynomial
c_{16}	-35.2518	Coefficient for A/F polynomial
c_{17}	20.7364	Coefficient for A/F polynomial
c_{18}	2.6287	Coefficient for A/F polynomial
c_{19}	-1.592	Coefficient for A/F polynomial
c_{20}	-2.3421	Coefficient for square root polynomial
c_{21}	2.7799	Coefficient for square root polynomial
c_{22}	-0.3273	Coefficient for square root polynomial
c_{12}	0.9	Pressure est. error factor
c_{23}	1.0	MAF sensor error factor
c_{24}	1.0	O_2 sensor error factor
c_{25}	1.0	Fuel inj. actuator error factor
c_{26}	4.0	First-order transfer fun. const.

Table 5: Polynomial Approximation Coefs., Error Tolerances