Statistical Verification of the Toyota Powertrain Control Verification Benchmark

Nima Roohi
Dep. of Computer Science
University of Illinois at
Urbana-Champaign, USA
roohi2@illinois.edu

Yu Wang
Coordinated Science Laboratory
University of Illinois at
Urbana-Champaign, USA
yuwang8@illinois.edu

Matthew West
Department of Mechanical
Science and Engineering
University of Illinois at
Urbana-Champaign, USA
mwest@illinois.edu

Geir E. Dullerud
Coordinated Science
Laboratory
University of Illinois at
Urbana-Champaign, USA
dullerud@illinois.edu

Mahesh Viswanathan
Department of Computer
Science
University of Illinois at
Urbana-Champaign, USA
vmahesh@illinois.edu

ABSTRACT

The Toyota Powertrain Control Verification Benchmark has been recently proposed as challenge problems that capture features of realistic automotive designs. In this paper we statistically verify the most complicated of the powertrain control models proposed, that includes features like delayed differential and difference equations, look-up tables, and highly non-linear dynamics, by simulating the `C++` code generated from the `Simulink`™ model of the design. Our results show that for at least 98% of the possible initial operating conditions the desired properties hold. These are the first verification results for this model, statistical or otherwise.

1. INTRODUCTION

The powertrain control problem is one of regulating the air-to-fuel ratio in a automotive engine. A series of models of such controllers, with increasing levels of sophistication and fidelity to real-world designs, have been recently proposed by Toyota researchers [6] as challenge problems for today's verification technologies. Ever since these models were proposed 2 years back, they have served as a high water mark for evaluating verification tools and techniques for cyber-physical systems. When the models were first proposed [6], the authors used the tool `S-Taliro` [1] to search for counter-examples in each of the controller models. While the search for counter-examples was unsuccessful, this analysis did not establish any formal statements of correctness for the designs. This is because `S-Taliro` is a *falsification* tool rather than a verification tool. Last year [4], the first (and to date, only) proof of correctness for one of the designs was established. Duggirala et. al. [4] used the tool `C2E2` to prove the correctness of Model 3, the simplest of the 3 models proposed by Toyota researchers. While Model 3 is a challenging non-linear hybrid system model, it is nonetheless extremely simple when compared with the most realistic Toyota model. All dynamics in Model 3 are described by polynomial differential equations.

In this paper we present our results in verifying the most complicated of the models proposed in [6] (Model 1, Section 3.1) that has features like delayed differential and difference equations, look-up tables, in addition to highly non-linear continuous dynamics for the state variables. Like `S-Taliro` and `C2E2`, our verification approach is also simulation-based, where the `Simulink`™ model (or, in our case, the `C++` code generated from the `Simulink`™ model) provided by the authors of [6] is executed to generate traces which are then analyzed. However, unlike previous approaches, in this paper, we use *statistical model checking* [8, 13] to analyze the traces.

Statistical model checking is an approach to check if a stochastic model satisfies certain quantitative properties. One example of a very simple quantitative property is that "75% of all system executions are safe". To check such a requirement, statistical model checking would probabilistic sample executions of the system, and see if the generated sample provides sufficient *statistical evidence* that 75% of the execution are indeed safe; this is determined based on the size of the sample, the evidence of the sample, and using statistical tests like Chernoff bounds or sequential probability ratio test. Since statistical model checking is a randomized algorithm, it can have both false positive and false negative errors. These are usually characterized by the Type I and Type II errors of hypothesis testing, and can be made as small as desired by increasing the number of samples drawn.

The power train control model (Model 1 in [6]) is not stochastic. In fact, if we fix the values of the input parameters, namely, engine speed and throttle angle signal profile, the model is completely deterministic. So how can we justify the use of statistical model checking which works only for stochastic models? The goal of verification is to show that for every setting of the input parameters, the execution of the controller satisfies the correctness requirements specified as *Signal Temporal Logic (STL)* properties given in [6]. In

HSCC'17, April 18-20, 2017, Pittsburgh, PA, USA

DOI: http://dx.doi.org/10.1145/3049797.3049804

this paper, our goal is to establish a weaker statement about correctness. Instead of proving that executions for *every* setting of initial parameters are correct, we aim to show that the system behaves correctly for *most* settings of the initial parameters. If the values of initial parameters are set *uniformly at random* from the input space, then we could view the powertrain controller as a stochastic model (where the only stochastic step is the choice of initial parameter values), and statistical model checking would indeed provide the relaxed correctness goals we are aiming for. Our experimental results show that Model 1 is correct for *at least* 98% of the settings for the input parameters; our probability of false positive and false negative errors were bounded to be at most 0.01.

While the statistical techniques we use to bound sample sizes are standard, we need a couple of ideas to enable checking whether a single sample execution satisfies the prescribed STL property. Mathematically, an execution of the powertrain controller is a function that specifies for each time instant, the state of the system. Clearly, such a "continuous" function cannot be sampled/simulated. Instead, what the simulation can generate, and what we can hope to analyze, is the sequence of system states sampled at discrete time points. But can analyzing the execution sampled at discrete time points determine the correctness of the actual, continuous execution? This is a standard problem that confronts all simulation-based verification methods for cyber-physical systems. We use ideas similar to those outlined in [5] to come up with a list of conditions on how finely the execution must be sampled (based on the correctness property) and an approximate satisfaction relation that together guarantee that if the discretely sampled execution satisfies an STL property then so does the actual, continuous execution. However, for such a property preservation result to go through, one needs to know the Lipschitz constant for the functions used to define the basic atomic propositions of the STL properties, over the set of reachable states of the systems. Determining such a Lipschitz bound for the powertrain controller is not easy because of its complicated dynamics. We overcome this through an innovative technique. The sampling algorithm starts with some value for the Lipschitz constant (which maybe wrong). We add a monitor to the `Simulink`™ model that *continuously* checks the derivative of the function whose Lipschitz bound we need, raising an "alarm" when this exceeds the guessed bound. When an execution generated by the `Simulink`™ model + monitor has no alarms, then the sample execution can be checked against the STL property because the Lipschitz bound holds for this execution, and if an alarm is raised then we need to abandon the execution, change the Lipschitz bound in the monitor, and resimulate.

The rest of the paper is organized as follows. In Section 2, we introduce the basic mathematical notation we use, and an introduction to statistical model checking; we assume that the reader is comfortable with the model hybrid automata, but for completeness it is defined in the Appendix. Section 3 describes in detail Model 1 of [6] that we analyze statistically. Our approach to verifying the model is described in Section 4. Our experimental results are in Section 5. Finally, we present our conclusions in Section 6.

2. PRELIMINARIES

The sets of *natural*, *real*, *non-negative real*, and *positive real* numbers are represented by $\mathbb{N}$, $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, and $\mathbb{R}_+$, respectively. For an element x and a set A, we use $x : A$ to denote that x is "of type" A or $x \in A$. For any two sets A and B, the set of functions from B to A is represented by A^B. In order to make the notations simpler, we denote $A^{\{0,1,\ldots,n-1\}}$ by A^n. Furthermore, *the set of subsets* of A is represented by 2^A. $A \times B$ is the *Cartesian product* of A and B. For any function $f : A \to B$ and a set C, by $f \restriction_C$ we mean a function from $A \cap C$ to B that maps a to $f(a)$. Also, by $\mathsf{range}(f)$ we mean $\{b : B \mid \exists a : A \bullet b = f(a)\}$. For any two metric spaces (M_1, ρ_1) and (M_2, ρ_2), and a function $f : M_1 \to M_2$, function f is called *Lipschitz continuous* if there is a constant $c : \mathbb{R}$ such that $\forall \epsilon : \mathbb{R}_+ \bullet x, y : M_1 \bullet \rho_1(x, y) < \epsilon \Rightarrow \rho_2(f(x), f(y)) < c\epsilon$. For any two sets $A, B \subseteq \mathbb{R}$ we define $A + B$ to be $\{x + y \mid x \in A \wedge y \in B\}$. Finally, $\mathcal{I}$ is the set of non-empty intervals over $\mathbb{R}_{\geq 0}$, and for any $I : \mathcal{I}$, infimum (supremum) of I is denoted by $\inf I$ ($\sup I$).

2.1 Signal Temporal Logic

Signal Temporal Logic (STL) is used to specify predicates over real-time signal. The most basic ingredient of STL is its atomic propositions. Each atomic proposition is a predicate over valuation of the state variables expressed as an inequality. Let $f : \mathbb{R}^{\mathbf{x}} \to \mathbb{R}$ be any Lipschitz continuous real valued function. An atomic proposition is then of the form $f > 0$. Syntax of a STL formula ϕ is defined using the following BNF grammar:

$$\phi := \top \mid f > 0 \mid \neg\phi \mid \phi \vee \phi \mid \phi \mathcal{U}_{\mathcal{I}} \phi$$

For a STL formula ϕ, duration $T : \mathbb{R}_{\geq 0}$, a signal $\tau : [0, T] \to \mathbb{R}^{\mathbf{x}}$, and a time $t : [0, T]$, satisfaction of ϕ by τ at time t is shown by $(\tau, t) \models \phi$ are defined using the following inductive rules. If $t > T$ then $(\tau, t) \models \phi$ is undefined. Otherwise,

$$\begin{array}{lll}
(\tau, t) \models \top & & \\
(\tau, t) \models f > 0 & \text{iff} & f(\tau(t)) > 0 \\
(\tau, t) \models \neg\phi & \text{iff} & (\tau, t) \not\models \phi \\
(\tau, t) \models \phi \vee \psi & \text{iff} & (\tau, t) \models \phi \text{ or } (\tau, t) \models \psi \\
(\tau, t) \models \phi \mathcal{U}_I \psi & \text{iff} & \exists r : I \bullet (\tau, t + r) \models \psi \wedge \\
 & & \quad \forall r' : [0, r) \bullet (\tau, t + r') \models \phi
\end{array}$$

Other connectives can be defined as well. $\bot := \neg\top$, $\phi \wedge \psi := \neg(\neg\phi \vee \neg\psi)$, $\phi \mathcal{R}_I \psi := \neg(\neg\phi \mathcal{U}_I \neg\psi)$, $\Diamond_I \phi := \top \mathcal{U}_I \phi$, and $\Box_I \phi := \neg(\Diamond_I \neg\phi)$. Intuitively, $\phi \mathcal{U}_I \psi$ means there is a time in I such that ψ is true at that time and ϕ is true up until that time. $\phi \mathcal{R}_I \psi$ means that ψ should always be true during the time interval I, a requirement which is released when ϕ becomes true. $\Diamond_I \phi$ means there is a time in I such that ϕ is true at that time. $\Box_I \phi$ means ϕ is always true during I. Finally, $\tau \models \phi$ is defined as $(\tau, 0) \models \phi$. Note that since we want to do statistical model checking, we won't be able to distinguish $f > 0$ from $f \geq 0$. Therefore, we only consider the strict form of inequality. Furthermore, having negation ($\neg$), disjunction ($\vee$) and conjunction ($\wedge$), among the set of STL operators, we know $f \leq 0$, $f < 0$, $f \geq 0$, $f = 0$, and $f \neq 0$ can be defined using $\neg(f > 0)$, $-f > 0$, $\neg(f < 0)$, $(f \geq 0) \wedge (f \leq 0)$, and $(f > 0) \vee (f < 0)$. Note that with the help of $\bot$, $\wedge$, and $\mathcal{R}$ operators, one can always push the negations to the front of atomic formulas and have a formula in negated normal form.

2.2 Sampling Based Algorithms

Statistical model checking [13] is a technique to verify stochastic systems against quantitative properties expressed

in a temporal logic. The approach draws sample trajectories from the model by discrete-event simulations, and uses hypothesis testing to infer whether the samples provide statistical evidence for the satisfaction or violation of the specification. The crux of this approach is that sample runs of a stochastic system are drawn according to the distribution defined by the system, and hence can be used to get estimates of the probability measure of executions.

In this paper, the properties we need to check are very simple quantitative properties and so here we outline the broad ideas behind checking such properties. Suppose the actual probability that executions of a system satisfy a STL property ϕ is θ'. Let H_0 be $\theta' < 1$ and H_1 be $\theta' \geq \theta$ for any value $\theta : (0, 1)$. Our goal is to determine which of the two hypotheses is true (with a small error probability); that is, the probability of saying H_0 (similarly H_1) is true while it is false should be small. Now every time we sample an execution, with probability θ' the execution will satisfy ϕ, and this can be thought of as a Bernoulli variable. Therefore, in n samples the number of executions satisfying ϕ is expected to be $n\theta'$. Though we don't know what θ' is, the fraction of executions satisfying ϕ is statistically a good estimate for θ', especially if the number of samples is large. This is the basis of statistical model checking algorithm. If θ and θ' are equal to each other then no matter how large of a sample we take, statistically we will not be able to decide whether $\theta' \geq \theta$ is true or not. Therefore, statistical tests work only when θ' is δ-away from θ for some arbitrarily small $\delta : \mathbb{R}_+$. Under this condition, several algorithms have been proposed using either Chernoff bounds, or sequential probability ratio test [8, 11, 12] that determine how many samples need to be drawn in order to ensure that the likelihood of our making a mistake is small. Taking S to be a Bernoulli random variable with mean θ', a typical algorithm $\mathcal{A}^{\delta}(S, t, \alpha, \beta, \gamma)$ takes the indifference parameter δ, and error bounds α, β, and γ as parameters and outputs either H_1, H_0, or `unknown`. The answer `unknown` is returned when $|\theta - \theta'| \leq \delta$. The algorithm is such that parameters δ, α, β, and γ can be made arbitrary small at the cost of more samples. It guarantees the following.

$$\mathbb{P}[result = H_0 \quad | \ H_0 \text{ is false}] \leq \alpha \tag{1}$$

$$\mathbb{P}[result = H_1 \quad | \ H_1 \text{ is false}] \leq \beta \tag{2}$$

$$\mathbb{P}[result = \texttt{unknown} \quad | \ |\theta - \theta'| > \delta] \leq \gamma \tag{3}$$

Formula 1 (respectively Formula 2) guarantees that the probability of returning a wrong hypothesis is bounded by α (and β). Formula 3 guarantees that the probability of returning `unknown` when θ is δ-away from θ' is bounded by γ. Notice that α, β, and γ are parameters and can be made as small as desired; the number of samples needed will grow proportionally. Second, the algorithm requires δ as a parameter as well, and this might seem difficult to obtain. However, observe that even if δ is given incorrectly (i.e., $|\theta - \theta'| < \delta$) the answers of the algorithm are not incorrect; it is just more likely to return `unknown`. In Section 4 we show that in the case of specific H_0 and H_1 defined here, we can set indifference and some of the error probabilities to zero while still guaranteeing conditions 1 through 3.

3. POWERTRAIN CONTROL PROBLEM

The goal of the powertrain control system is to maintain the air-fuel ratio at a desired value for optimal function of the internal combustion engine under different driving behaviors and conditions. It is modeled in Simulink™. Intuitively, the model takes input from a driver (input throttle angle θ_{in}) and from the environment (engine speed ω and `failed` event) and controls the behavior of the engine. It is a non-linear hybrid model with 4 control modes and around 15 state and intermediate variables defined as parallel composition of two hybrid i/o automaton (HIOA): a controller (A_c) and a plant (A_p). [6] introduced 3 different models of this controller, with increasing fidelity to the real world. In this paper, we analyzed the most complicated model. Dynamics of both controller and plant are non-linear. Furthermore, plant's dynamics are highly non-linear and involve delayed differential equation. Table 1 briefly describes different variables used in the specification of the system.

State	Unit	Description
p	bar	Intake Manifold Pressure
λ_c	-	A/F Ratio in Cylinder
λ_m	-	Transfer Function Output
p_e	bar	Estimated 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
$\dot{m}_\phi$	g/s	Fuel Mass Aspirated into the Cylinder
$\dot{m}_\psi$	g/s	Fuel Mass Injected into Intake Manifold
θ_{in}	degrees	Throttle Angle Input
θ	degrees	Delay-Filtered Throttle Angle
$\hat{\theta}$	-	O/P of Throttle Polynomial
F_c	g/s	Command fuel
ω	rad/sec	Engine Speed
n	round/sec	Engine Speed ($\frac{\omega}{2\pi}$)

Table 1: States and Indeterminate Variables

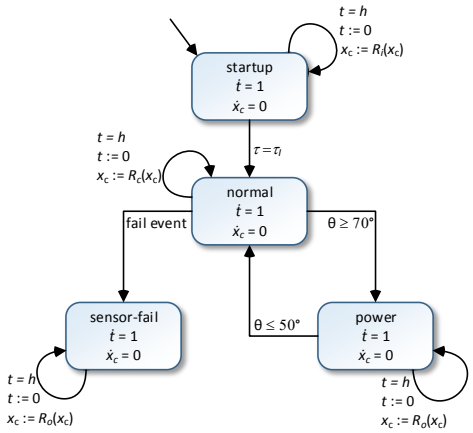

Figure 1: Hybrid automaton A_c Modeling the Controller Part of the Powertrain System

The hybrid automaton $A = A_c || A_p$ has two inputs: 1. the engine speed (ω) which is a constant value in $[900, 1100]$. 2. the throttle angle (θ_{in}) signal profile, which is the set of pulse train signals (with pulse width equal to half the period). In this work, we verify the model when the period (ζ) is in $[10, 30]$ and amplitude (a) is in $[8.8, 70]$.

Properties that need to be verified are specified using Signal Temporal Logic (STL) [3]. The following two formulas are used to specify points in time that input signal θ_{in} is

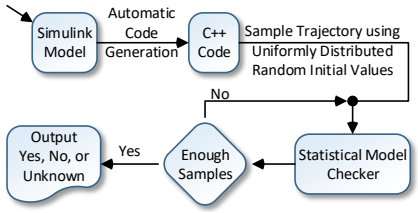

Figure 2: Steps of Our Approach to Model Check Powertrain Controller Verification Problem

about to go up or down:

$$\texttt{rise}(a) := (\theta_{in} = 8.8) \wedge \Diamond_{(0,\epsilon)}(\theta_{in} = a) \quad (4)$$

$$\texttt{fall}(a) := (\theta_{in} = a) \wedge \Diamond_{(0,\epsilon)}(\theta_{in} = 8.8) \quad (5)$$

A normalized error signal μ that measures the error in A/F Ratio (λ_m) (a state variable of the system) from the reference stoichiometric value λ_{ref} is defined as follows:

$$\mu(t) = \frac{\lambda_m(t) - \lambda_{ref}}{\lambda_{ref}} \quad (6)$$

Finally, we want to check the following two properties [1]:

$$\Box_{(\tau_s,T)}|\mu| < 0.05 \quad (7)$$

$$\Box_{(\tau_s,T)}\big(\texttt{rise}(a) \vee \texttt{fall}(a) \Rightarrow \Box_{(\eta,5)}|\mu| < 0.02\big) \quad (8)$$

Formula 7 specifies the maximum allowed overshoot or undershoot. Formula 8 specifies that η seconds after a signal rises or falls, the signal settles and remains in the settling region for at least 4 seconds. Parameters ϵ, λ_{ref}, τ_s, η, and T are set to 0.02, 14.7, 11, 1, and 50, respectively [2].

4. OUR APPROACH

We statistically model check [9,13] the powertrain control design. More precisely, given a model M, STL formula ϕ, and threshold θ, the goal of the verification procedure is to check that the measure of initial states satisfying formula ϕ is less that 1 (hypothesis H_0) or larger than θ (hypothesis H_1). Our algorithm randomly simulates the model (or more precisely the C++ code automatically generated from the Simulink™ model) from different uniformly chosen initial states. On any sampled execution τ, our algorithm first checks if τ satisfies ϕ (see Section 4.1). If the answer is `unknown` or `no`, our algorithm immediately returns `unknown` or H_0, respectively. If τ satisfies ϕ, our algorithm returns H_1 if the number of sampled executions is $> \frac{\ln(\beta)}{\ln(\theta)}$. Otherwise, it draws another sample and repeats the whole process. Figure 2 depicts the high level steps of our approach.

If we ever find an execution that does not satisfy ϕ, then clearly H_0 is true and there will be no error in returning it. When number of samples, n, is larger than $\frac{\ln(\beta)}{\ln(\theta)}$ it means $\theta^n < \beta$. If H_1 is false, it means that the measure of initial states satisfying ϕ (θ') is less than or equal to θ. Therefore, $\theta'^n \leq \theta^n < \beta$. Hence, the probability of sampling n executions all satisfying ϕ will be less than the error probability β. Therefore, our algorithm guarantees the two conditions given by inequalities (1) and (2). Note that we do not have any indifference region (*i.e.* δ is zero). β the only parameter to our algorithm (the number of samples will increase as β becomes smaller). Thus our algorithm ensures

$$\mathbb{P}[result = H_0 \mid H_0 \text{ is false}] = 0 \quad (9)$$

$$\mathbb{P}[result = H_1 \mid H_1 \text{ is false}] \leq \beta \quad (10)$$

In Section 4.1, we define a robustness condition and show that our algorithm also guarantees the following condition:

$$\mathbb{P}[result = \texttt{unknown} \mid \text{sampled executions are robust}] = 0 \quad (11)$$

4.1 Checking Executions against STL Specification

To complete the description of our model checking algorithm, we need to describe how to check if a sampled execution satisfies an STL specification. Now the model of an STL formula is a signal which is function that gives for each time instant the system state. Clearly such an execution must be represented in some finite, computationally tractable manner. Simulation will result in an execution that is sampled at discrete time points, and the system state is only known at the sample time points.

Given a time step $\Delta : \mathbb{R}_+$, an execution sampled at discrete time points, will be a sequence of systems states at times $n\Delta$, where $n : \mathbb{N}$. Because the execution only gives the state at times of the form $n\Delta$, the challenge is discover conditions when we can conclude the satisfaction of the STL property ϕ at *all* times. The problem has been considered previously, and we borrow ideas from [5] to overcome this challenge.

The key to addressing this issue is to introduce a new "approximate satisfaction" relation. Let $\Delta : \mathbb{R}_+$ be a time step. Table 2 defines a ternary discrete semantics we use in our approach. In this table ΔI is defined to be $\{n : \mathbb{N} \mid n\Delta \in I\}$, and ΔI^- is defined to be $\{n : \mathbb{N} \mid \forall r : \Delta I \bullet n < r\}$. Thus, ΔI is the set of all natural numbers n such that $n\Delta$ is in I and, ΔI^- is the set of all natural numbers n that are smaller than all numbers in ΔI.

Intuitvely, the semantics in Table 2 tries to achieve the following: $((\tau, n) \models_\Delta \phi)$ is $\top$ if we can conclude that $(\tau, t) \models \phi$ for all $t \in [(n-1)\Delta, (n+1)\Delta]$ based on the state at time $n\Delta$; similarly, $((\tau, n) \models_\Delta \phi)$ is $\bot$ if we can conclude that $(\tau, t) \not\models \phi$ for all $t \in [(n-1)\Delta, (n+1)\Delta]$ based on the state at time $n\Delta$; and finally, $((\tau, n) \models_\Delta \phi)$ is `unknown` otherwise. The semantics can be understood being built up from the case of atomic propositions. If for any $t = n\Delta$, $f(\tau(t)) > c\Delta$, where c is the Lipschitz constant for f, then one can conclude that $f(\tau(t')) > 0$ for all $t' : [t - \Delta, t + \Delta]$. Similarly, if $f(\tau(t)) < -c\Delta$, then one can conclude that $f(\tau(t')) < 0$ for all $t' : [t - \Delta, t + \Delta]$. On the other hand, if $|f(\tau(t))| \leq c\Delta$ then by just looking at $f(\tau(t))$ we cannot conclude whether $f(\tau(t'))$ is above or below 0 for all $t' : [t - \Delta, t + \Delta]$; hence we define this to be `unknown`. The rest of the semantics is built naturally from this observation. The main result of this section is that this property holds for the approximate semantics we have defined above.

Theorem 1. *Consider a signal $\tau : [0, T] \to \mathbb{R}^{\mathtt{X}}$, an STL formula ϕ, and sampling time Δ that satisfy the following conditions.*

[1] Formulas 7 and 8 are required to be true only when the controller is in the `normal` mode. But in our case, the input signals guarantee that the controller mode us normal for all times in (τ_s, T).

[2] T is the default value set in [2]. Other parameters are directly from [6].

$((\tau, n) \models_\Delta \top)$	$:= \top$	iff	always true
$((\tau, n) \models_\Delta f > 0)$	$:= \top$	iff	$f(\tau(n\Delta)) > c\Delta$
$((\tau, n) \models_\Delta f > 0)$	$:= \bot$	iff	$f(\tau(n\Delta)) < -c\Delta$
$((\tau, n) \models_\Delta f > 0)$	$:=$ `unknown`	iff	otherwise
$((\tau, n) \models_\Delta \neg(f > 0))$	$:= \top$	iff	$f(\tau(n\Delta)) < -c\Delta$
$((\tau, n) \models_\Delta \neg(f > 0))$	$:= \bot$	iff	$f(\tau(n\Delta)) > c\Delta$
$((\tau, n) \models_\Delta \neg(f > 0))$	$:=$ `unknown`	iff	otherwise
$((\tau, n) \models_\Delta \phi \vee \psi)$	$:= \top$	iff	$((\tau, n) \models_\Delta \phi) = \top$ or $((\tau, n) \models_\Delta \psi) = \top$
$((\tau, n) \models_\Delta \phi \vee \psi)$	$:= \bot$	iff	$((\tau, n) \models_\Delta \phi) = \bot$ and $((\tau, n) \models_\Delta \psi) = \bot$
$((\tau, n) \models_\Delta \phi \vee \psi)$	$:=$ `unknown`	iff	otherwise
$((\tau, n) \models_\Delta \phi \wedge \psi)$	$:= \top$	iff	$((\tau, n) \models_\Delta \phi) = \top$ and $((\tau, n) \models_\Delta \psi) = \top$
$((\tau, n) \models_\Delta \phi \wedge \psi)$	$:= \bot$	iff	$((\tau, n) \models_\Delta \phi) = \bot$ or $((\tau, n) \models_\Delta \psi) = \bot$
$((\tau, n) \models_\Delta \phi \wedge \psi)$	$:=$ `unknown`	iff	otherwise
$((\tau, n) \models_\Delta \phi \mathcal{U}_I \psi)$	$:= \top$	iff	$\exists r : \Delta I \bullet ((\tau, n + r) \models_\Delta \psi) = \top \wedge$ $\forall r' : \{0, \ldots, r\} \bullet ((\tau, n + r') \models_\Delta \phi) = \top$
$((\tau, n) \models_\Delta \phi \mathcal{U}_I \psi)$	$:= \bot$	iff	$\forall r : \Delta I + \{-\Delta, 0, \Delta\} \bullet ((\tau, n + r) \models_\Delta \psi) = \bot$ or $\exists r : \Delta I^- \bullet ((\tau, n + r) \models_\Delta \phi) = \bot$ or $\exists r : \Delta I \bullet ((\tau, n + r) \models_\Delta \phi) = \bot \wedge$ $\forall r' : \Delta I \cap \{0, \ldots, r\} \bullet ((\tau, n + r') \models_\Delta \psi) = \bot$
$((\tau, n) \models_\Delta \phi \mathcal{U}_I \psi)$	$:=$ `unknown`	iff	otherwise
$((\tau, n) \models_\Delta \phi \mathcal{R}_I \psi)$	$:= \top$	iff	$\forall r : \Delta I + \{-\Delta, 0, \Delta\} \bullet ((\tau, n + r) \models_\Delta \psi) = \top$ or $(\exists r : \Delta I \bullet ((\tau, n + r) \models_\Delta \phi) = \top \wedge$ $\forall r' : \{0, \ldots, r\} \cap \Delta I \bullet ((\tau, n + r') \models_\Delta \psi) = \top)$
$((\tau, n) \models_\Delta \phi \mathcal{R}_I \psi)$	$:= \bot$	iff	$\exists r : \Delta I \bullet ((\tau, n + r) \models_\Delta \psi) = \bot \wedge$ $\forall r' : \Delta I \cap \{0, \ldots, r\} \bullet ((\tau, n + r') \models_\Delta \phi) = \bot$
$((\tau, n) \models_\Delta \phi \mathcal{R}_I \psi)$	$:=$ `unknown`	iff	otherwise

Table 2: Semantic Rules for Ternary $\models_\Delta$. Parameters $\Delta : \mathbb{R}_+$ is a time step and $c : \mathbb{R}_+$ is a *guessed* Lipschitz Constant.

1. *There is $c : \mathbb{R}_+$ such that all functions used in atomic formulas of ϕ are c-Lipschitz continuous in the domain of τ.*
2. *The sampling time Δ is such that for any interval I appearing in ϕ, $\Delta \leq \frac{1}{2}(\sup I - \inf I)$. This condition ensures that there will be at least one sample point inside each interval appearing in ϕ.*
3. *All intervals appearing in ϕ are bounded* [3].
4. *Let $M = \sum_{I:\phi} \sup I$. Then $T > |\phi|\Delta + M$.*

Then, the semantics in Table 2 guarantees

$$\forall n : \mathbb{N} \bullet (((\tau, n) \models_\Delta \phi) = \top) \Rightarrow \forall t : [(n-1)\Delta, (n+1)\Delta] \cap [0, \infty) \bullet (\tau, t) \models \phi$$
and
$$\forall n : \mathbb{N} \bullet (((\tau, n) \models_\Delta \phi) = \bot) \Rightarrow \forall t : [(n-1)\Delta, (n+1)\Delta] \cap [0, \infty) \bullet (\tau, t) \not\models \phi$$

Corollary 2. *Under the conditions of Theorem 1 we have:*

$$((\tau \models_\Delta \phi) = \top) \Rightarrow \tau \models \phi$$
$$((\tau \models_\Delta \phi) = \bot) \Rightarrow \tau \not\models \phi$$

[3]Note that since τ is a signal over bounded time $[0, T]$, this is not a restriction.

4.2 Checking Lipschitzness

As mentioned in Section 4.1, we are using discrete time sampling. Therefore, we need to make sure that the conditions in Theorem 1 are satisfied. The main problem is that dynamics of the input system is too complex and we cannot analytically find a Lipschitz constant for the functions in atomic formulas. In order to solve this problem we guess a Lipschitz constant and augment the `Simulink`™ model (before generating `C++` code) such that if our guess ever falsified a flag will be set. Note, that the flag will be set if our guess for the Lipschitz constant is violated at *any time*, and not just at the sampled times. Then after simulating a trajectory we check the flag. If it indicates our guess was wrong, we guess a new value for the constant and repeat the whole process. We use simulations in `Simulink`™ as a source for our guess. In our experiment Lipschitz constant $c = 2$ was never falsified. So we use $c = 2$ as our final guess. The correctness of our guess for the Lischtiz constant relies on the correctness of the zero finder algorithms implemented in `C++` libraries of `Simulink`™.

Finally, if we want to prove a probability is larger than the given threshold, we treat $(\tau \models_\Delta \phi) =$ `unknown` as $(\tau \models_\Delta \phi) = \bot$. And similarly, if we want to prove a probability is smaller than the given threshold, we treat $(\tau \models_\Delta \phi) =$ `unknown` as $(\tau \models_\Delta \phi) = \top$.

5. EXPERIMENTAL RESULTS

We used `EmbeddedCoder` of `Simulink`™ to generate `C++` code from the model files in [2]. "Sample Rate" (Δ for us) is fixed to 0.0001 seconds. "Device Vendor" is set to `Intel`™, and "Device Type" is set to `x86 − 64 (Mac OS X)`. We also set for "Execution efficiency". ODE solver is set to be selected automatically by `Simulink`™. Finally, we used `C++03 (ISO)` for "Standard math library". Note that automatic code generation is only for simulation. The model checker was written manually in `C++`. All executions ran on an `iMac` with `3.5 GHz Intel Core i7` processor and `32 GB 1600 MHz DDR3` memory.

In our experiment, we set $\beta = 0.001$ and $t = 0.98$. Both Formulas 7 and 8 could be verified after an average of 453 samples. Verification of Formula 7 takes about 2.2 minutes and verification of Formula 8 takes about 2.8 minutes.

5.1 Other Requirements

There are other requirements specified in [6], specially regarding the `power` mode and when the controller switches between `power` and `normal` multiple times. We failed to verify these requirements. The main reason is that since neither `Simulink`™ nor `C++` has formal semantics, there is no guarantee that the translation from `Simulink`™ to `C++` preserves the behavior of the modeled system. The requirement in this case, demands that when the amplitude is set to 80 degrees in the input throttle, the controller mode switches between modes `normal` and `power` periodically. This behavior can be easily observed in simulations performed by `Simulink`™. However, the `C++` code that is *automatically generated* from the model does not exhibit such a behavior.

6. CONCLUSION AND PLANNED FUTURE WORK

The results presented here are the first results providing mathematical guarantees of correctness for Model 1, the most complex design, presented in [6]. It relied on statistically model checking, discretely sampled executions of the `C++` code generated from the `Simulink`™ model of the controller. Our approach did not account for any errors introduced due to numerical integration. Our take here was that the `C++` program generated automatically from the `Simulink`™ model is the artifact to be checked as it may be directly running on an embedded chip. If such errors are important and can be quantified, future analyses of this model will need to account for them.

Currently, our results certify that with high probability more than 98% of the initial states satisfy the correctness requirements. We would like to push our analysis closer towards 100%. Naïve sampling becomes infeasible in this case. We believe this problem can be solved in two complementary ways: 1. anti-correlated sampling [7] 2. importance sampling [10].

7. ACKNOWLEDGEMENT

The authors acknowledge partial support for this work from grants NSF CPS 1329991 and NSF CCF 1422798.

8. REFERENCES

[1] Y. Annpureddy, C. Liu, G. 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: 17th International Conference, TACAS*, pages 254–257, 2011.

[2] J. V. Deshmukh. Simulink/Stateflow models of the Powertrain Controller Verification Benchmark. Personal Communication.

[3] A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *Formal Modeling and Analysis of Timed Systems: 8th International Conference, FORMATS*, pages 92–106, 2010.

[4] P. S. Duggirala, C. Fan, S. Mitra, and M. Viswanathan. Meeting a powertrain verification challenge. In *Computer Aided Verification: 27th International Conference, CAV*, pages 536–543. Springer International Publishing, 2015.

[5] G. E. Fainekos and G. J. Pappas. Robust Sampling for MITL Specifications. In *Formal Modeling and Analysis of Timed Systems: 5th International Conference, FORMATS*, pages 147–162, 2007.

[6] X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda, and K. Butts. Powertrain control verification benchmark. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control, HSCC*, pages 253–262, New York, NY, USA, 2014. ACM.

[7] P. A. Maginnis, M. West, and G. E. Dullerud. Anticorrelated discrete-time stochastic simulation. In *Decision and Control (CDC), IEEE 52nd Annual Conference on*, pages 618–623, 2013.

[8] K. Sen, M. Viswanathan, and G. Agha. On statistical model checking of stochastic systems. In K. Etessami and S. K. Rajamani, editors, *Computer Aided Verification*, number 3576 in Lecture Notes in Computer Science, pages 266–280. Springer Berlin Heidelberg, 2005.

[9] K. Sen, M. Viswanathan, and G. Agha. On statistical model checking of stochastic systems. In *Computer Aided Verification: 17th International Conference, CAV*, pages 266–280, 2005.

[10] R. Srinivasan. *Importance sampling : applications in communications and detection*. Springer, Berlin, New York, 2002.

[11] A. Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2):pp. 117–186, 1945.

[12] H. L. S. Younes. Error control for probabilistic model checking. In *Verification, Model Checking, and Abstract Interpretation, 7th International Conference, VMCAI*, pages 142–156, 2006.

[13] H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Computer Aided Verification: 14th International Conference, CAV*, pages 223–235, 2002.