Meeting a Powertrain Verification Challenge

Parasara Sridhar Duggirala, Chuchu Fan, Sayan Mitra, and Mahesh Viswanathan

University of Illinois, Urbana-Champaign
duggira3@illinois.edu, cfan10@illinois.edu,
mitras@illinois.edu, vmahesh@illinois.edu.

Abstract. We present the verification of a benchmark powertrain control system using the hybrid system verification tool C2E2. This model comes from a suite of benchmarks that were posed as a challenge problem for the hybrid systems community, and to our knowledge, we are reporting its first verification. For this work, we implemented the algorithm reported in [10] in C2E2, to automatically compute local discrepancy (rate of convergence or divergence of trajectories) of the model. We verify the key requirements of the model, specified in signal temporal logic (STL), for a set of driver behaviors.

1 A Challenge Problem

As the targets for fuel efficiency, emissions, and drivability become more demanding, automakers are becoming interested in pushing the design automation and verification technologies for automotive control systems. The benchmark suite of powertrain control systems were published in [12,11] as challenge problems that capture some of the difficulties that arise in verification of realistic systems. It consists of a sequence of Simulink™/Stateflow™ models of the engine with increasing levels of sophistication and fidelity. At a high-level, the models take inputs from a driver (throttle angle) and the environment (sensor failures), and define the dynamics of the engine. The key controlled quantity is the air to fuel ratio which in turn influences the emissions, the fuel efficiency, and torque generated. The requirements for the system are stated in signal temporal logic (STL). A typical property, for example, $\diamond_t(x \in [x_{eq} - \epsilon, x_{eq} + \epsilon])$, states that after t units of time, the continuous variable x is within the range $x_{eq} \pm \epsilon$. Breach [4] and STaliro [2] have been used for finding counterexamples (or falsifying) models in [13,12,14,5]. These techniques can show the presence of executions that violate a requirement, but not their absence. The technique used in this paper proves that all the executions from a given set of initial states and a set of switching signals satisfies (or violates) the requirement. To the best of our knowledge, this is the first time a model in the powertrain control benchmark is verified.

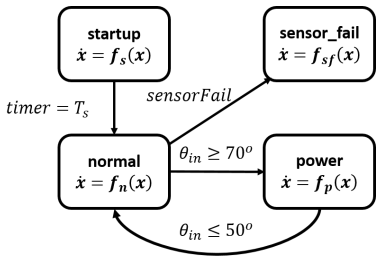
The model we consider in this paper is polynomial hybrid automata model (Model 3, Section 3.3) of [12]. Although this model is given as a Simulink™ diagram with switch blocks, it can be transformed to a hybrid automaton with 4 locations and 5 continuous variables. The dynamics of the system is given by

highly nonlinear polynomial differential equations. The mode transitions are brought about by the input signal from the driver and there are uncertainties in the initial set owing to measurement inaccuracies. Using an improved version of the C2E2 tool [6,7] we are able to perform reachability analysis of this model and we verify the requirements with respect to a set of relevant driver behaviors. In principle, Flow* [3] is designed to handle polynomial hybrid automata models, however, it was unable to verify the models considered in this paper, owing to the complexity of nonlinear dynamics.

C2E2 is a verification tool for a general class of nonlinear hybrid systems. The previous version of C2E2 [6,7] required the user to provide a special type of annotation for the model, called *discrepancy function*, which essentially captures the rate of convergence (or divergence) of neighboring trajectories. Finding discrepancy functions for nonlinear models can be challenging. One of the main developments that enabled this verification, is the implementation of a new algorithm in C2E2 (presented in detail in [10]) for automatic computation of local discrepancy along trajectories of the system. Using this improved C2E2, we were not only able to find counterexamples, but also verify the key STL requirements of the powertrain benchmark in the order of minutes.

2 Nonlinear Hybrid Powertrain Model

Simulink^{TM} model for the powertrain control system is shown in Figure 1(a). The system has four continuous variables p, λ, p_e, i (see Figure 1(b)), and four modes of operation: *startup*, *normal*, *power*, and *sensor_fail*. The mode switches (also called *transitions*) are brought about by changes in the input *throttle angle* θ_{in} or *failure* events. The rest of the Simulink^{TM} diagram defines polynomial



(a) Hybrid automata model of powertrain control system.

Variable	Description
p	Intake manifold pressure
p_e	Intake manifold pressure estimate
λ	Air-fuel ratio
i	Integrator state, control variable
θ_{in}	Throttle angle

(b) Table with state variables and their description.

Fig. 1: Figure showing the model in (a) and the model variables in (b).

differential equations that govern the evolution of the continuous variables in the four different modes. As an example, we reproduce the differential equation for normal mode of operation.

$$\begin{aligned}
\dot{p} &= c_1(2\theta_{in}(c_{20}p^2 + c_{21}p + c_{22}) - c_{12}(c_2 + c_3\omega p + c_4\omega p^2 + c_5\omega^2 p)) \\
\dot{\lambda} &= c_{26}(c_{15} + c_{16}c_{25}F_c + c_{17}c_{25}^2F_c^2 + c_{18}\dot{m}_c + c_{19}\dot{m}_c c_{25}F_c - \lambda) \\
\dot{p_e} &= c_1(2c_{23}\theta_{in}(c_{20}p^2 + c_{21}p + c_{22}) - (c_2 + c_3\omega p_e + c_4\omega p_e^2 + c_5\omega^2 p_e)) \\
\dot{i} &= c_{14}(c_{24}\lambda - c_1 1).
\end{aligned}$$

Here $F_c = \frac{1}{c_{11}}(1 + i + c_{13}(c_{24}\lambda - c_{11}))(c_2 + c_3\omega p_e + c_4\omega p_e^2 + c_5\omega^2 p_e)$, $\dot{m}_c = c_{12}(c_2 + c_3\omega p + c_4\omega p^2 + c_5\omega^2 p)$, and all the c_i's are constant parameters of the model.

This model is translated to a hybrid automaton form that is accepted by C2E2. The operating modes correspond to the locations of the automaton, the variables correspond to the above continuous variables, the differential equations define the trajectories, and the discrete transitions among the locations is defined by a piecewise constant input signal θ_{in} from the driver behavior. C2E2 currently handles only closed automaton models. Therefore, for every driver behavior of interest, we explicitly construct a family of switching signals that determine the timing of the mode switches. The initial set of the automaton is a ball in the state space which corresponds to the measurement uncertainty in state components.

The goal of the powertrain control system is to maintain the air-fuel ratio at a desired value for optimal functioning of internal combustion engine under different driving behaviors and conditions. These control objectives or requirements are stated in [12] using STL formulas. An example requirement for the *normal* mode of operation is the following:

$$rise \Rightarrow \Box_{(\eta,\zeta)}(0.98\lambda_{ref} \leq \lambda \leq 1.02\lambda_{ref}), \tag{1}$$

which can be read as "If the throttle angle θ_{in} changes from 0 to 60, denoted by the event $rise$, then the air-fuel ratio λ should be in the range $[0.98\lambda_{ref}, 1.02\lambda_{ref}]$ after η time units and stay in that region until ζ time units. Here λ_{ref} is the reference (desired) air-fuel ratio and η and ζ are parameters of the property. We note that this type of requirements can also be expressed as bounded time invariants— the class of properties currently handled by C2E2. We simply need to introduce a $timer$ variable that keeps track of time elapsed since the last occurrence of the relevant events like $rise$ in the above example.

3 Verification using C2E2 with Local Discrepancy

C2E2 implements a generic, simulation-based, algorithm for bounded time verification of invariant and temporal precedence properties of nonlinear hybrid models (see [6,7,8] for details). The algorithm iteratively computes more precise over-approximations of the reachable states of the system until it either proves the property (the requirement) or finds a counter-example. These over-approximations are computed for each location and for the duration that the system is in that location. The set of reachable states at the end of that interval serves as the starting set for the next location and so on. Thus, the key step in the algorithm is to compute and refine reach set over-approximations for ODEs for a given location. This step uses validated simulations and discrepancy functions [6].

A *validated simulation* of an ordinary differential equation (ODE) $\dot{x} = f(x)$ from an initial state x_0 with error bound ϵ is a sequence of time-stamped regions

$\psi = (R_0, t_0), \ldots, (R_k, t_k)$ such that for each time interval $[t_{i-1}, t_i]$ the solution $\xi(x_0, .)$ resides in the region R_i and $dia(R_i) \leq \epsilon$. A uniformly continuous function $\beta : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a *discrepancy function* of the above ODE if (a) for any pair of states $x, x' \in \mathbb{R}^n$, and any time $t > 0$, $\|\xi(x,t) - \xi(x',t)\| \leq \beta(x, x', t)$, and (b) for any t, as $x \to x'$, $\beta(., ., t) \to 0$. Thus, β gives an upper bound on the rate of divergence of two neighboring trajectories and this bound vanishes as their initial states approach each other.

In order to check whether the system satisfies an invariant I over a time horizon T, the C2E2 algorithm starts with a δ-cover of the initial set and proceeds as follows: from each point x_0 in the cover a validated simulation is generated and then bloated by a factor given by the discrepancy function. This bloated set is an over-approximation of the reachset from the δ-neighborhood ($B_\delta(x_0)$) of x_0. If this set is disjoint from (or contained in) I^c then the algorithm infers that the initial set $B_\delta(x_0)$ satisfies (or violates, respectively) I. Otherwise, a finer cover of $B_\delta(x_0)$ is created and added to C for computing a more precise over-approximation of the reach set from $B_\delta(x_0)$. The first property of the discrepancy function gives the soundness of this algorithm, and the second property gives relative completeness (see, Theorem 13 from [6]).

This approach requires the user to provide discrepancy functions which can be burdensome. Although Lipschitz constants, contraction metrics [15], and incremental Lyapunov functions [1] can be used to get discrepancy for certain classes of models, none of these approaches give an algorithm for computing β for general nonlinear ODEs. In this paper, we use the algorithm presented in [10] for computing local discrepancy functions on-the-fly along validated simulations. This algorithm uses the Jacobian J_f and a Lipschitz constant L_f of the ODE. First it computes a coarse over-approximation $S(x_i)$ of the reach set from a simulation point for a short duration. Then it computes an exponential (possibly negative) bound on the divergence rate of trajectories over $S(x_0)$ by finding a bound on the maximum eigenvalue of the symmetric part of the Jacobian J_f over the region $S(x_0)$. We refer the reader to the technical report [10] for the details of this algorithm.

3.1 Tool Implementation and Engineering

Implementation. For verifying the powertrain system, we implemented the local discrepancy algorithm in C2E2[1]. This modified implementation only requires the user to supply the Jacobian matrix of the system. The eigenvalues of the symmetric parts of the Jacobian are computed using Eigen library [9]. For maximizing the norm of error matrices our implementation uses interval arithmetic.

Coordinate Transformation. An important technical detail that makes the implementation scale is the coordinate transformation proposed in [10]. For Jacobian matrices with complex eigenvalues the local discrepancy computed directly using the above algorithm can be a positive exponential even though

[1] The modified tool and related files are available from
http://publish.illinois.edu/c2e2-tool/powertrain-challenge/

the actual trajectories are not diverging. This problem can be avoided by first computing a local coordinate transformation and then applying the algorithm. Coordinate transformation provides better convergence, but comes with a multiplicative cost in the error term. This trade-off between the exponential divergence rate and the multiplicative error has be tuned by choosing the time horizon over which the coordinate transformation is computed.

Model Reduction. In *start up* and *power* mode of the system, the differential equation does not update the value of the integrator variable i, i.e., $\dot{i} = 0$. Moreover, i does not appear in the right hand side of the differential equations for variables p, λ, p_e. We take advantage of these observations, and consider only the dynamics of the variables p, λ, and p_e for computing local discrepancy.

4 Experimental Results on Powertrain Challenge

We have implemented the algorithm described in Section 3 as a prototype extension of the tool C2E2. Verification of key properties of powertrain systems is typically performed on a standard set of driver behaviors as the number of switching signals corresponding to driver behaviors are infinite. In this paper, we pick two sets of driver behaviors provided in [12] that visit all the modes of the system. Further, to enable verification with C2E2, the STL properties were encoded as bounded time safety properties. Hence, the properties in [12] which involved integrals over paths, could not be verified. Table 1 provides the results of verifying different STL properties.

Property	Mode	Sat.	Sim.	Time
$\Box_{T_s,T}\lambda \in [0.8\lambda_{ref}, 1.2\lambda_{ref}]$	*all modes*	yes	53	11m58s
$\Box_{[0,T_s]}\lambda \in [0.8\lambda_{ref}, 1.2\lambda_{ref}]$	*startup*	yes	50	10m21s
$\Box_{[T_s,T]}\lambda \in [0.95\lambda_{ref}, 1.05\lambda_{ref}]$	*normal*	yes	50	10m28s
$\Box_{[T_s,T]}\lambda \in [0.8\lambda^{pwr}_{ref}, 1.2\lambda^{pwr}_{ref}]$	*power*	yes	53	11m12s
$\Box_{[0,T_s]}\lambda \in [0.98\lambda_{ref}, 1.02\lambda_{ref}]$	*startup*	no	2	0m24s
$\Box_{[T_s,T]}\lambda \in [0.9\lambda^{pwr}_{ref}, 1.1\lambda^{pwr}_{ref}]$	*power*	no	4	0m43s
$rise \Rightarrow \Box_{(\eta,\zeta)}\lambda \in [0.9\lambda_{ref}, 1.1\lambda_{ref}]$	*startup*	yes	50	10m40s
$rise \Rightarrow \Box_{(\eta,\zeta)}\lambda \in [0.98\lambda_{ref}, 1.02\lambda_{ref}]$	*normal*	yes	50	10m15s
$(\ell = power) \Rightarrow \Box_{(\eta^{pwr},\zeta)}\lambda \in [0.95\lambda^{pwr}_{ref}, 1.05\lambda^{pwr}_{ref}]$	*power*	yes	53	11m35s
$(\ell = power) \Rightarrow \Box_{(\eta^{s},\zeta)}\lambda \in [0.95\lambda^{pwr}_{ref}, 1.05\lambda^{pwr}_{ref}]$	*power*	no	4	0m45s

Table 1: Table showing the result and the time taken for verifying STL specification of the powertrain control system. Sat: Satisfied, Sim: Number of simulations performed. All the experiments are performed on Intel Quad-Core i7 processor, with 8 GB ram, on Ubuntu 11.10.

The first six properties provided in Table 1 are invariant properties. These invariant properties can be global (i.e. correspond to all modes) or could be restricted to a certain mode of operation provided in the $Mode$ column. The invariants assert that the air-fuel ratio should not go out of the specified bounds.

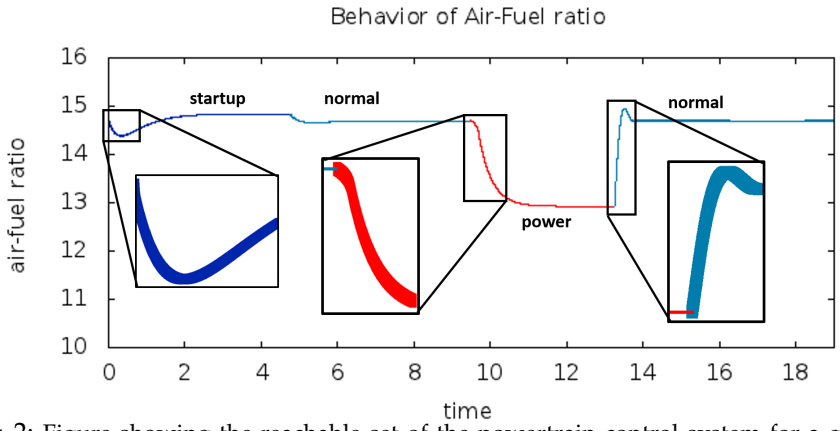


Fig. 2: Figure showing the reachable set of the powertrain control system for a given user behavior that visits different modes.

Observe that C2E2 could not only prove that the given specification is satisfied, but also that a stricter version of invariants for *startup* and *power* modes is violated. The next four properties are about the settling time requirements. These requirements enforce that in a given mode, whenever an action is triggered, the fuel air ratio should be in the given range provided after η (or η^{pwr} for power mode) time units. Similar to the invariant properties, C2E2 could also find counterexample for a stricter version of the settling time requirement (η^s settling time instead of η) in *power* mode. When C2E2 finds an overapproximation that violates a given property, it immediately terminates and hence C2E2 takes less time when it finds counterexamples. The parameters used for verification are $\eta = \eta^{pwr} = 1$, $\eta^s = 0.5$, $T_s = 9$, $T = 20$, $\lambda_{ref} = 14.7$, $\lambda_{ref}^{pwr} = 12.5$, and $\zeta = 4$. Set of reachable states of the powertrain control system for a given driver behavior is provided in Figure 2.

5 Conclusions And Future Work

In this paper, we have successfully applied the simulation based verification technique with local discrepancy functions to find counterexamples and verify the polynomial hybrid automata model of powertrain benchmark challenge. This case study suggests that verification using on-the-fly discrepancy function along with the coordinate transformation can handle complex nonlinear dynamics. In future, we wish to extend these techniques to handle higher fidelity models in the powertrain verification challenge. These models contain delay differential equations, actuation delays, and look up tables, which C2E2 cannot currently handle.

Acknowledgment: We thank Jim Kapinski, Jyo Desmukh, and Xiaoqing Jin of Toyota for several useful discussions on the powertrain models. This research is funded by research grants from the National Science Foundation (grant: CAR 1054247 and NSF CSR 1016791) and the Air Force Office of Scientific Research (AFOSR YIP FA9550-12-1-0336).

References

1. David Angeli. A lyapunov approach to incremental stability properties. In *IEEE Transactions on Automatic Control*, 2000.
2. Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2011.
3. Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, pages 258–263, 2013.
4. Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification*, pages 167–170. Springer, 2010.
5. Tommaso Dreossi, Thao Dang, Alexandre Donzé, James Kapinski, Jyotirmoy V Deshmukh, and Xiaoqing Jin. Efficient guiding strategies for testing of temporal properties of hybrid systems. In *Proceedings of NASA Formal Methods Conference (to appear).*, 2015.
6. Parasara Sridhar Duggirala, Sayan Mitra, and Mahesh Viswanathan. Verification of annotated models from executions. In *Proceedings of the International Conference on Embedded Software, EMSOFT 2013*, pages 1–10. IEEE, 2013.
7. Parasara Sridhar Duggirala, Sayan Mitra, Mahesh Viswanathan, and Matthew Potok. C2E2: A verification tool for stateflow models. In *21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2015)*, 2015.
8. Parasara Sridhar Duggirala, Le Wang, Sayan Mitra, Mahesh Viswanathan, and César Muñoz. Temporal precedence checking for switched models and its application to a parallel landing protocol. In *FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings*, volume 8442 of *Lecture Notes in Computer Science*, pages 215–229. Springer, 2014.
9. Eigen. *a C++ template library for linear algebra*, (accessed February, 2015). http://eigen.tuxfamily.org.
10. Chuchu Fan and Sayan Mitra. Bounded verification using on-the-fly discrepancy computation. Technical Report UILU-ENG-15-2201, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, February 2015.
11. Xiaoqing Jin, Jyotirmoy V Deshmukh, James Kapinski, Koichi Ueda, and Ken Butts. Benchmarks for model transformations and conformance checking. In *1st International Workshop on Applied Verification for Continuous and Hybrid Systems (ARCH)*, 2014.
12. Xiaoqing Jin, Jyotirmoy V Deshmukh, James Kapinski, Koichi Ueda, and Ken Butts. Powertrain control verification benchmark. In *Proceedings of the 17th international conference on Hybrid systems: computation and control*, pages 253–262. ACM, 2014.
13. Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V Deshmukh, and Sanjit A Seshia. Mining requirements from closed-loop control models. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 43–52. ACM, 2013.
14. Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V Deshmukh, and Sanjit A Seshia. Mining requirements from closed-loop control models. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (to appear)*, 2016.
15. W. Lohmiller and J. J. E. Slotine. On contraction analysis for non-linear systems. *Automatica*, 1998.