EPiC Series in Computer Science
Volume 34, 2015, Pages 207–212
ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems

Progress on powertrain verification challenge with C2E2*

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

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

Abstract

In this paper, we present the progress we have made in verifying a benchmark powertrain control system. We implemented the on-the-fly algorithm for computing discrepancy of nonlinear dynamical systems in the C2E2 verification tool. We created Stateflow translations of the original models to aid the processing using C2E2 tool and we encoded the different driver behaviors in the form of state machines. With these customizations, we have been successful in verifying one of the benchmarks from the powertrain suite. In this paper, we discuss the engineering challenges and the lessons learned from the process.

1 The powertrain benchmarks

The benchmark suite of powertrain control systems were published in [11, 10] as challenge problems for hybrid system verification. The suite has a set of Simulink™ models with increasing levels of sophistication and fidelity. At a high-level, all 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 first model (model 1) is the most complex. It has look-up tables, delayed differential equations, and switches. Models 2 and 3 are simpler but still complicated enough for most hybrid verifcation tools. Model 3 is a hybrid automaton with polynomial differential equations and continuously computed control inputs, and Model 2 is similar but with nonlinear differential equations and both continuous and discretely sampled variables. 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 $x_{eq} \pm \epsilon$.

*We thank Jim Kapinski, Jyotirmoy Deshmukh, 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).

Breach [2] and STaliro [1] have been used for finding counterexamples (or falsifying) models in [13, 11, 12, 3]. Our main verification results for this benchmark have been reported in [4]. In this companion paper we report on several technical details, including the model transformation, and our experience in performing verification using C2E2 [6, 5].

2 Background on C2E2

C2E2 implements a generic, simulation-based, algorithm for bounded time verification of invariant and temporal precedence properties of nonlinear hybrid models (see [5, 6, 7] for details). The algorithm iteratively computes increasingly precise over-approximations of the reachable states of the system until it either proves the property (the requirement) or finds a counter-example.

Although, the benchmark models are hybrid systems, C2E2 does not use hybrid simulations. Instead, it generates over-approximations for each location, finds the intersection of the reachtube with the out-going guards from that location, and continues with these intersections as the initial sets in the next location. 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 that give a bound on the convergence (divergence) of trajectories starting from neighboring states [5].

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 [9]) 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.

In this paper, we use the algorithm presented in [9] for computing local discrepancy functions on-the-fly along validated simulations. This algorithm requires 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 [9] for the details of this algorithm.

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 [8]. For maximizing the norm of error matrices our implementation uses interval arithmetic.

3 Model transformation

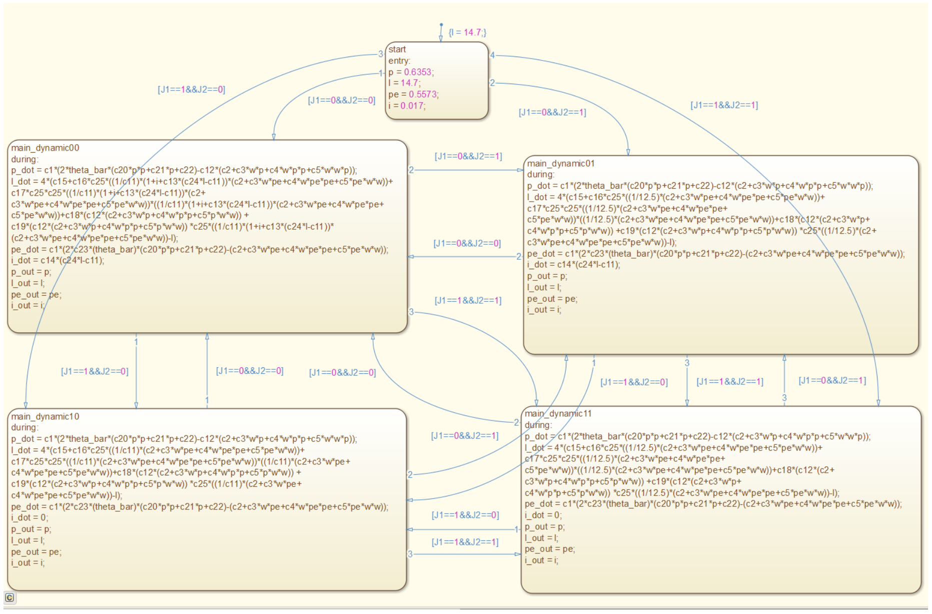
We manually transform the Simulink$^{\text{TM}}$ diagram of the benchmarks with switching blocks, to a Stateflow model which essentially captures the hybrid automaton. Models 2 and 3 of [11] translate to hybrid automata with 4 locations and 5 continuous variables. The locations are *startup*, *normal*, *power*, and *sensor_fail*. The continuous variables are: (a) intake manifold pressure (p), (b) intake manifold pressure estimate (p_e), (c) air-fuel ratio (λ), (d) integrator state (i),

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

(e) throttle angle ($theta_{in}$). These translated Stateflow models are made available as part of this paper.

This transformation is relatively straightforward and has been described in [14]. The Simulink model uses several function blocks connected by feedback lines. While the Stateflow model uses differential equations and transitions. The transitions are decided by the boolean operation of several user inputs like throttle angle and sensor failure. Keeping these input signals constant, we rewrite the differential equations of the four discrete modes in Stateflow blocks, and then replace the function block *Switch* in Simulink with *Transitions*.

Model 2 (the second model in [11]) differs in two aspects: (1) the right-hand side of the system equations are general nonlinear functions instead of polynomial functions; (2) only two of the four variables are continuous, other two are discrete variables updated periodically. Only the differential equations of the two continuous variables would appear in the Stateflow modes. We introduce the third variable t with the dynamic $\dot{t} = 1$. Initially $t = 0$, whenever $t =$ discrete sample time, there will be a transition to the mode itself with transition action $t = 0$ and the update of the two discrete variables.



```
start
entry:
p = 0.6353;
l = 14.7;
pe = 0.5573;
i = 0.017;
```

```
main_dynamic00
during:
p_dot = c1*(2*theta_bar*(c20*p*p+c21*p+c22)-c12*(c2+c3*w*p+c4*w*p*p+c5*w*w*p));
l_dot = 4*(c15+c16*c25*((1/c11)*(1+i+c13*(c24*l-c11))*(c2+c3*w*pe+c4*w*pe*pe+c5*pe*w*w))+
c17*c25*c25*((1/c11)*(1+i+c13*(c24*l-c11))*(c2+
c3*w*pe+c4*w*pe*pe+c5*pe*w*w))*((1/c11)*(1+i+c13*(c24*l-c11))*(c2+c3*w*pe+c4*w*pe*pe+
c5*pe*w*w))+c18*(c12*(c2+c3*w*p+c4*w*p*p+c5*p*w*w)) +
c19*(c12*(c2+c3*w*p+c4*w*p*p+c5*p*w*w)) *c25*((1/c11)*(1+i+c13*(c24*l-c11))*
(c2+c3*w*pe+c4*w*pe*pe+c5*pe*w*w))-l);
pe_dot = c1*(2*c23*(theta_bar)*(c20*p*p+c21*p+c22)-(c2+c3*w*pe+c4*w*pe*pe+c5*pe*w*w));
i_dot = c14*(c24*l-c11);
p_out = p;
l_out = l;
pe_out = pe;
i_out = i;
```

```
main_dynamic01
during:
p_dot = c1*(2*theta_bar*(c20*p*p+c21*p+c22)-c12*(c2+c3*w*p+c4*w*p*p+c5*w*w*p));
l_dot = 4*(c15+c16*c25*((1/12.5)*(c2+c3*w*pe+c4*w*pe*pe+c5*pe*w*w))+
c17*c25*c25*((1/12.5)*(c2+c3*w*pe+c4*w*pe*pe+
c5*pe*w*w))*((1/12.5)*(c2+c3*w*pe+c4*w*pe*pe+c5*pe*w*w))+c18*(c12*(c2+c3*w*p+
c4*w*p*p+c5*p*w*w)) +c19*(c12*(c2+c3*w*p+c4*w*p*p+c5*p*w*w)) *c25*((1/12.5)*(c2+
c3*w*pe+c4*w*pe*pe+c5*pe*w*w))-l);
pe_dot = c1*(2*c23*(theta_bar)*(c20*p*p+c21*p+c22)-(c2+c3*w*pe+c4*w*pe*pe+c5*pe*w*w));
i_dot = c14*(c24*l-c11);
p_out = p;
l_out = l;
pe_out = pe;
i_out = i;
```

```
main_dynamic10
during:
p_dot = c1*(2*theta_bar*(c20*p*p+c21*p+c22)-c12*(c2+c3*w*p+c4*w*p*p+c5*w*w*p));
l_dot = 4*(c15+c16*c25*((1/c11)*(c2+c3*w*pe+c4*w*pe*pe+c5*pe*w*w))+
c17*c25*c25*((1/c11)*(c2+c3*w*pe+c4*w*pe*pe+c5*pe*w*w))*((1/c11)*(c2+c3*w*pe+
c4*w*pe*pe+c5*pe*w*w))+c18*(c12*(c2+c3*w*p+c4*w*p*p+c5*p*w*w)) +
c19*(c12*(c2+c3*w*p+c4*w*p*p+c5*p*w*w)) *c25*((1/c11)*(c2+c3*w*pe+
c4*w*pe*pe+c5*pe*w*w))-l);
pe_dot = c1*(2*c23*(theta_bar)*(c20*p*p+c21*p+c22)-(c2+c3*w*pe+c4*w*pe*pe+c5*pe*w*w));
i_dot = 0;
p_out = p;
l_out = l;
pe_out = pe;
i_out = i;
```

```
main_dynamic11
during:
p_dot = c1*(2*theta_bar*(c20*p*p+c21*p+c22)-c12*(c2+c3*w*p+c4*w*p*p+c5*w*w*p));
l_dot = 4*(c15+c16*c25*((1/12.5)*(c2+c3*w*pe+c4*w*pe*pe+c5*pe*w*w))+
c17*c25*c25*((1/12.5)*(c2+c3*w*pe+c4*w*pe*pe+
c5*pe*w*w))*((1/12.5)*(c2+c3*w*pe+c4*w*pe*pe+c5*pe*w*w))+c18*(c12*(c2+
c3*w*p+c4*w*p*p+c5*p*w*w)) +c19*(c12*(c2+c3*w*p+
c4*w*p*p+c5*p*w*w)) *c25*((1/12.5)*(c2+c3*w*pe+c4*w*pe*pe+c5*pe*w*w))-l);
pe_dot = c1*(2*c23*(theta_bar)*(c20*p*p+c21*p+c22)-(c2+c3*w*pe+c4*w*pe*pe+c5*pe*w*w));
i_dot = 0;
p_out = p;
l_out = l;
pe_out = pe;
i_out = i;
```

Figure 1: Transformed stateflow model of powertrain control system.

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. As the resulting transformed model cannot be parsed by the current C2E2 parser, we had to directly encode this mode in the intermediate form used by the C2E2 verification engine.

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 [11] using STL formulas. An example requirement for the *normal* mode of operation is the following:

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

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 desired value of 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.

4 Experience using C2E2 on the powertrain models

Encoding drivers and properties. The C2E2 parser currently does not support parameters that are specified in a table (for example, the various coefficients in the polynomial differential equations). For this reason, we had to partially hand-code the C++ simulation files[2] for these models that are otherwise generated automatically. Modifying these C++ files, one can also verify different driver behaviors. The file `simulator.cpp` models the ODEs of different modes in the model, the file `guard.cpp` models the guard conditions for enabling the transitions between the modes, and `invariant.cpp` models the invariants for each mode. We have considered two sets of driver behaviors in this paper. In the first set, the system starts in the *startup*' mode, and after [9.00,9.01] time units, it switches to *normal* mode. In the second set, the system starts in the *startup* mode, switches to *normal* mode, then switches to *power* mode, and finally returns again to the *normal* mode. The property that can be verified in the given version of C2E2 are invariants such as the air-fuel ratio always being in a given range. The initial set for the behaviors and the unsafe set are specified as polytope in Configuration file given as input to C2E2.

Coordinate transformation. An important technical detail that makes the implementation scale is the coordinate transformation proposed in [9]. For Jacobian matrices with complex eigenvalues the local discrepancy computed directly using the above algorithm can be a positive exponential even though 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 given by the condition number of the matrix. 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.

In our experiments, we have observed that the condition number for *startup* mode is 20 and for all other modes are of the order of 200. Thus, one cannot perform this coordinate transformation over small periods as this would lead to large errors in the overapproximations. Thus, the number of steps for which coordinate transformation should be applied is an engineering decision based on the condition number and the exponential rate of convergence. For verifying the powertrain control system, we have analyzed different possibilities and observed that coordinate transformation after every 3000 steps (i.e. 3 time units) provides overapproximation that is adequate for verification.

[2]These files are also made available as part of this submission.

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.

Verification results. Table 1 provides the results of verifying different STL properties. 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. 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 $\eta = \eta^{pwr} = 1$, $\eta^s = 0.5$, $T_s = 9$, $T = 20$, $\lambda_{ref} = 14.7$, $\lambda^{pwr}_{ref} = 12.5$, and $\zeta = 4$.

5 Conclusion

In [4] we reported that with some additional engineering, it is possible to tackle the problem of verifying a challenging powertrain control system benchmark using the C2E2 tool with the local discrepancy method. In this paper, we provide additional details about that verification process including model tranformation and tool usage. In future, we wish to extend these techniques to handle higher fidelity models in the powertrain verification challenge that involve discrete updates and delayed differential equations.

References

[1] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. *S-taliro: A tool for temporal logic falsification for hybrid systems*. Springer, 2011.

[2] Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification*, pages 167–170. Springer, 2010.

[3] 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.

[4] Parasara Sridhar Duggirala, Chuchu Fan, Sayan Mitra, and Mahesh Viswanathan. Meeting a powertrain verification challenge. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA*, volume 9206 of *LNCS*, pages 536–543. Springer, 2015.

[5] 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, Montreal, QC, Canada, September 29 - Oct. 4, 2013*, pages 1–10. IEEE, 2013.

[6] 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.

[7] 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.

[8] Eigen. *a C++ template library for linear algebra*, (accessed February, 2015). `http://eigen.tuxfamily.org`.

[9] Chuchu Fan and Sayan Mitra. Bounded verification using on-the-fly discrepancy computation. In *Automated Technology for Verification and Analysis (ATVA 2015), Shanghai, China*, LNCS. Springer, October 2015.

[10] 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.

[11] 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.

[12] 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)*. IEEE.

[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] Karthik Manamcheri Sukumar and Sayan Mitra. A step towards verification and synthesis from simulink/stateflow models. In *Tools paper in Hybrid Systems: Computation and Control (HSCC 2011)*, 2011.