Automatic reachability analysis for nonlinear hybrid models with C2E2 *

Chuchu Fan[1], Bolun Qi[1], Sayan Mitra[1],
Mahesh Viswanathan[1], and Parasara Sridhar Duggirala[2]

[1] University of Illinois, Urbana-Champaign
[2] University of Connecticut
cfan10, bolunqi2, mitras, vmahesh@illinois.edu, psd@uconn.edu.

Abstract. C2E2 is a bounded reachability analysis tool for nonlinear dynamical systems and hybrid automaton models. Previously it required users to annotate each system of differential equations of the hybrid automaton with *discrepancy functions*, and since these annotations are difficult to get for general nonlinear differential equations, the tool had limited usability. This version of C2E2 is improved in several ways, the most prominent among which is the elimination of the need for user-provided discrepancy functions. It automatically computes piece-wise (or local) discrepancy functions around the reachable parts of the state space using symbolically computed Jacobian matrix and eigenvalue perturbation bounds. The special cases of linear and constant rate differential equations are handled with more efficient algorithm. In this paper, we discuss these and other new features that make the new C2E2 a usable tool for bounded reachability analysis of hybrid systems.

1 Introduction

C2E2 is a tool for checking bounded time invariant properties of nonlinear hybrid automaton models through reachability analysis. A hybrid automaton combines ordinary differential equations (ODE) and with guarded-command program fragments, and is seen as a convenient mathematical formalism for describing a variety of cyber-physical systems. Since nonlinear differential equations often do not have analytical solutions, C2E2 implements a simulation-based approach for over-approximating the reachable states of a system of ODEs. This involves: (a) generating numerical simulations of the ODE from a finite set of representative initial states that cover the whole (uncountably many) initial set, say Θ, (b) bloating each of these simulations by *some factor* such that the bloated tubes together over-approximate the reachable states from Θ, and (c) checking if this computed over-approximation is adequate for proving invariance; otherwise, add more representative initial states to obtain a more precise over-approximation and repeat from (a).

The previous version of C2E2 [10,11] relied on the user to provide the bloating factor, formally called a *discrepancy function*, required in step (b). For linear ODEs, one

* This work was in part supported by the grants CCF 1422798 and CNS1054247 from the National Science Foundation.

could, in principle, find discrepancy functions automatically from the dynamics of the system, but for general nonlinear systems this is not the case. The primary improvement we present in this new version of C2E2 [1] relieves the user from this burden. We have implemented the algorithm presented in [13] which computes a piece-wise (or local) discrepancy function for the ODE. This algorithm is *on-the-fly* or *lazy* in that it only computes the discrepancy function around parts of the state-space that are known to be reachable (from step (a)). For linear models the new implementation automatically computes a global discrepancy function. Automatic handling of systems with constant dynamics, and a technique to carry-out coordinate transformation are also implemented to improve the overall performance of C2E2. The new C2E2 can automatically verify and find counter-examples in interesting nonlinear and linear hybrid systems created using Stateflow^{TM}: for example, a 5-dimensional highly nonlinear benchmark model of a powertrain control system [17], an auto-passing control system with 6 modes, and a 28-dimensional linear model of a helicopter [22].

2 Related tools

Several automatic verification tools for hybrid models have been developed over the past two decades and they have been used in verifying numerous systems. Uppaal [20], HyTech [15] and SpaceEx [14] target timed automata, rectangular hybrid automata, and linear hybrid automata, respectively. Nonlinear dynamical and hybrid models are handled by d/dt [4], Flow* [6], dReach [19], CORA [3], and Ariadne [5]. S-Taliro [21] finds counter-examples in complex and realistic models using Monte-carlo techniques and provides probabilistic guarantees. STRONG [7] is a MATLAB toolbox for analysis of linear hybrid systems and it uses a Lyapunov function-based approach.

The simulation-based verification algorithm implemented in C2E2 is closest in spirit to the Matlab-based Breach tool [8]. Breach uses sensitivity analysis of the ODEs (related to our notion of discrepancy function) to verify signal temporal logic (STL) properties. Sensitivity analysis is known to be sound for linear ODEs and for more complex models Breach uses numerical procedures for estimating the Jacobian matrix of the system. This enables it to handle complex models at the expense of rigorous guarantees. The new version of C2E2 computes discrepancy functions from symbolically computed Jacobian matrix of the ODEs. This gives soundness and relative completeness guarantees, but restricts its application to ODEs with continuously differentiable right hand sides.

3 New features in C2E2

The architecture of C2E2 is shown in Figure 1. The GUI-based front end parses the input hybrid model which has to be given either as a Stateflow model (`.mdl`) or as an XML file (`.hyxml`). The front end produces (i) an executable for producing validated simulations for each system of ODEs for the hybrid automaton, (ii) a specification of the candidate invariant properties to be checked, and (iii) a newly implemented function for symbolically evaluating the Jacobian matrix for each system of ODEs in the hybrid model. The front end provides a property editor which checks syntactic correctness of

properties as they are typed. After the back end produces verification results, which includes the reach set and possibly counter-example, these objects can be plotted using the front end as well (see Figure 2).

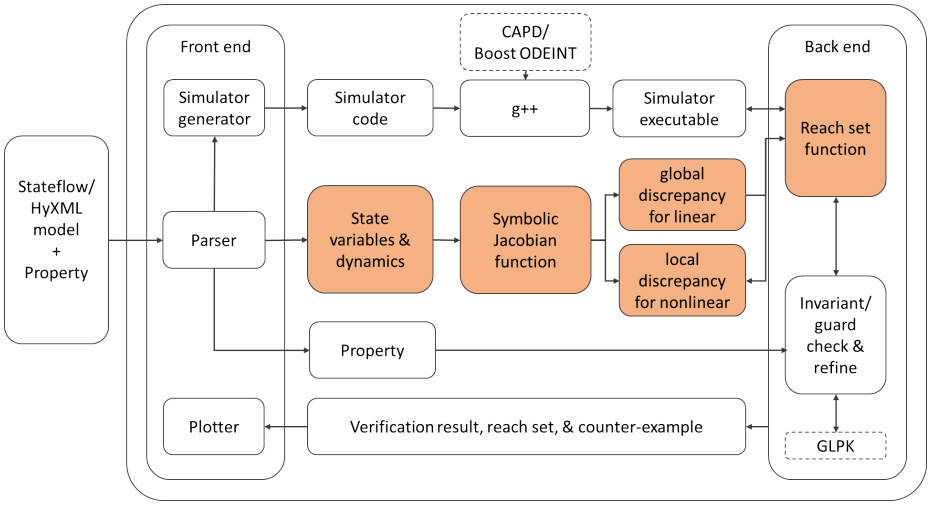

Fig. 1: Architecture of C2E2: The colored blocks are newly implemented. Improvements in existing blocks are discussed in Section 4.2.

The Jacobian function and the simulator are used by the new back end for computing reachable states using the approach described in Section 1. If the system or a discrete location of the hybrid system is linear (which is automatically checked by the front end), then the Jacobian matrix is used to compute a global discrepancy function once and for all. Otherwise, the back end iteratively calls the simulator as well as the Jacobian function, to over-approximate the reachable states over small time intervals (more details are given in Section 4).

The rest of the functions in the back end work with the computed reach set to check for the guards of the hybrid automaton. It also checks if the candidate invariant properties are provably satisfied or violated. Based on these decisions the main verification loop decides to (a) return results to the front end, or (b) start over the process by refining the initial set of states, or (c) start simulations from a new set of initial states in a new mode (that is, with a new system of ODEs).

4 Automatic discrepancy computation

4.1 Overview

The block labeled *local discrepancy for nonlinear* in Figure 1 implements the algorithm for computing piece-wise discrepancy function for general nonlinear ODEs using

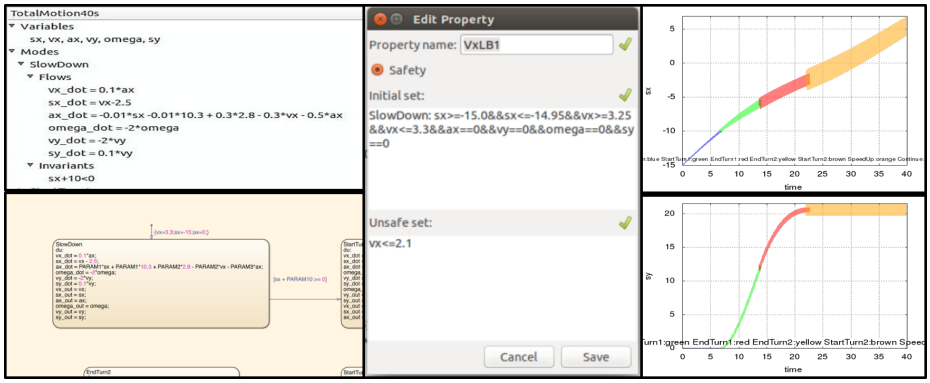

Fig. 2: Left to right: figures showing a snippet of partial auto-passing control model in C2E2 front end and Stateflow^{TM}, property dialog, plots of reachable set for auto-passing control model.

Jacobian matrix and Lipschitz constant. It takes one simulation trajectory and initial partition size as input at one time, and produces a sequence of coefficients. These coefficients define the piece-wise exponential discrepancy function. The algorithm consists of the following steps:

a) First, using the Lipschitz constant a coarse over-approximation of the reachable set up to a short time horizon T_s is constructed. Let this set be S.
b) The largest eigenvalue $\lambda_{max}\left((J(s_0) + J^T(s_0))/2\right)$ of the symmetric part of the Jacobian matrix $J(s_0)$ at the center s_0 of S is computed.
c) From $\lambda_{max}\left((J(s_0) + J^T(s_0))/2\right)$ an upper bound b of the eigenvalue of the symmetric part of all the Jacobian matrices $J(s), s \in S$ is computed. This uses a theorem from matrix perturbation theory and involves bounding the terms of the symbolic Jacobian over S.
d) The upper bound b (possibly negative) defines the discrepancy function $\beta(t) = \beta'(t_0)e^{b(t-t_0)}$ over the simulation time interval $[t_0, t_0 + T_s]$, where $\beta'(\cdot)$ is the previous piece of the discrepancy function. Using this piece-wise discrepancy function an over-approximation of the reachable set is computed.

The soundness of the algorithm comes from the fact that the computed bound b over a certain region S provides an exponential bound on the distance of any two trajectories in that region.

4.2 Implementation and Enhancements

We discuss the design decisions made in implementing the above mentioned functions and how they impact C2E2.

Symbolic Jacobian computation. From the parse tree generated by the front end, the state variables and ODEs for each location of the hybrid automaton are extracted. For

an ODE $\frac{dx}{dt} = f(x)$, where f is a vector valued function, the Jacobian matrix $J(x)$ is the matrix of partial derivatives $J_{ij}(x) = \frac{\partial f_i}{\partial x_j}$. We use the Python Sympy[3] library for computing derivatives of f symbolically. This library handles a general class of functions and as a result our implementation of symbolic Jacobian computation works for all standard polynomial, trigonometric, exponential and logarithmic functions. Our approach works for complicated models like the powertrain benchmark [18,17] which has more than 30 nonlinear terms in f. C2E2 compiles the symbolic Jacobian matrices into a Python module, which is then used to evaluate their numerical values.

Discrepancy function computation To compute local discrepancy functions on-the-fly, the upper bound of the eigenvalues of (the symmetric part of) Jacobian matrices and the upper bound of the matrix perturbations are obtained along the simulation traces using Python linear algebra library[4]. The local discrepancy function module communicates with the reach set function, takes simulation traces and initial set, and returns a local discrepancy function designed specially for the given simulation trace. In C2E2, we provide multiple simulator options: the validated simulator CAPD [2] as well as the standard ODE solver in the Boost library[5]

Global discrepancy for linear ODEs For linear time invariant hybrid models, the entries in the symbolic Jacobian matrix are constants. Thus, the local discrepancy function will be the same as the global one. C2E2 takes advantage of this fact and evaluates the Jacobian matrix just once and computes a global exponential discrepancy function to be used throughout, instead of on-the-fly local discrepancy. For example, analysis of the 28-dimensional linear model of the helicopter with this approach completes in seconds.

Automatic handling of constant dynamics Often hybrid models have timers and other variables that evolve at a constant rate with time. The ODEs for such systems have the form:

$$\begin{cases} \frac{dx}{dt} = f(x) \\ \frac{dy}{dt} = k \end{cases}$$

where k is a constant and y changes at that constant rate with time. Although the simple dynamics of y should make it easier to compute its reach set—at any time t, $y(t) = y(0) + kt$—our discrepancy-based algorithm has problems dealing with such systems. These constant-rate variables introduce all 0 rows and all 0 columns in the Jacobian matrix. This not only increases the dimension of the system, but also introduces extra conservatism in the estimation of the eigenvalues. For example, the Jacobian matrix of such systems has 0 eigenvalue even when the rest of the system is stable. The new C2E2 mitigates this problem by automatically decomposing the system by handling the constant-rate part independently.

For example, the Cardiac cell model in [11] uses a timer $\frac{d(\text{timer})}{dt} = 1$ to transit between the location where stimulate is on and the location where it is off. Systems with such constant dynamics are detected and decomposed automatically. That is, C2E2 will

[3] http://www.sympy.org/en/index.html
[4] http://docs.scipy.org/doc/numpy/reference/routines.linalg.html
[5] https://headmyshoulder.github.io/odeint-v2/

first compute the reach set of $\frac{dx}{dt} = f(x)$ using our standard technique, then bloat $y(t)$ by δ_y for $\frac{dy}{dt} = k$, where δ_y is the size of initial set for variable y.

Coordinate transformation Coordinate transformation can help produce less conservative over-approximations of the reach set. Coordinate transformations are done automatically in the new C2E2 in the following manner: first, Jacobian matrix is transformed to the real Jordan form by a similarity transformation, and then the similarity transformation matrix is used to perform the linear coordinate transformation. Such transformation decreases the conservatism of exponential bound (the factor b mentioned in 4.1), but comes at the price of a constant multiplicative factor in $\beta(t)$. C2E2 allows the users to set a parameter in the GUI that helps explore this trade off.

Other enhancements We re-implemented the reach set plotter for C2E2 which now uses gnuplot and is much faster. It also shows unsafe regions and counter-example segments. C2E2 now comes with testing scripts and a command line interface. The tests check the reach sets computed on a new installation against the corresponding reference versions computed in our lab machine. Examples inputs and outputs are documented in the website[6].

Detailed comparison of the performance of the new C2E2 with other verification tools will be presented in a future paper and in the tool's website. In several examples, it performs favorably in comparison with Flow*[6]. For example, it verifies a 10 dimensional nonlinear cardiac cell model from [16] (Figure 1) in less than 10 seconds where Flow* took 500 seconds. The dynamics is given by, for example, $f_1(x_1, x_2, u_1, u_2, stim) = -0.9x_1^2 - x_1^3 - 0.9x_1 - x_2 + 10(u_1 + u_2 - 2x_1) + stim$ and $f_2(x_1, x_2) = x_1 - 2x_2$, with $S_{on} = 5$ and $S_{off} = 20$.

5 Discussion of performance and Conclusions

The new version of C2E2 comes with a growing set of interesting example models such as a powertrain control system with highly nonlinear dynamics, a 28-dimensional linear helicopter, a hybrid auto-passing control model with 6 locations, a cardiac cell model and others. Although some of these (for example, the powertrain control system model [12,9]) had been verified earlier, those analyses involved hand-crafting special functions inside C2E2 for computing Jacobian matrices, handling constant dynamics, etc. The new C2E2 checks the examples automatically, without the need for annotations, typically in minutes.

A single reach set computation by bloating a single simulation trace using discrepancy computation usually takes less than one second for nonlinear systems with 5 - 6 dimensions or linear systems, up to a time horizon of 10 seconds. The verification time of each example, of course, depends on the complexity of the system, the distance of the unsafe set from the reachable set, the stability of the dynamics, and the time horizon.

In summary, this paper presents several new features implemented in C2E2, the most prominent one being an algorithm for computing discrepancy functions for linear, nonlinear, and constant ODEs. These features make the new C2E2 a more usable

[6] http://publish.illinois.edu/c2e2-tool/example/

tool for verifying nonlinear hybrid models while preserving the original soundness and relative completeness guarantees.

References

1. *C2E2 Webpage*. http://publish.illinois.edu/c2e2-tool/.
2. *Computer Assisted Proofs in Dynamic Groups (CAPD)*. http://capd.ii.uj.edu.pl/index.php.
3. Matthias Althoff. An introduction to cora 2015. In *ARCH*, 2015.
4. Eugene Asarin, Thao Dang, and Oded Maler. The d/dt tool for verification of hybrid systems. In *CAV*, pages 365–370. Springer, 2002.
5. Andrea Balluchi, Alberto Casagrande, Pieter Collins, Alberto Ferrari, Tiziano Villa, and Alberto L Sangiovanni-Vincentelli. Ariadne: a framework for reachability analysis of hybrid automata. In *MTNS*. Citeseer, 2006.
6. Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *CAV*, pages 258–263. Springer, 2013.
7. Yi Deng, Akshay Rajhans, and A Agung Julius. Strong: A trajectory-based verification toolbox for hybrid systems. In *QEST*, pages 165–168. Springer, 2013.
8. Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *CAV*, pages 167–170. Springer, 2010.
9. Parasara Sridhar Duggirala, Chuchu Fan, Sayan Mitra, and Mahesh Viswanathan. Meeting a powertrain verification challenge. In *CAV*, pages 536–543. Springer, 2015.
10. Parasara Sridhar Duggirala, Sayan Mitra, and Mahesh Viswanathan. Verification of annotated models from executions. In *EMSOFT*, page 26. IEEE Press, 2013.
11. Parasara Sridhar Duggirala, Sayan Mitra, Mahesh Viswanathan, and Matthew Potok. C2e2: A verification tool for stateflow models. In *TACAS*, pages 68–82. Springer, 2015.
12. Chuchu Fan, Parasara Sridhar Duggirala, Sayan Mitra, and Mahesh Viswanathan. Progress on powertrain verification challenge with c2e2. In *ARCH*, 2015.
13. Chuchu Fan and Sayan Mitra. Bounded verification with on-the-fly discrepancy computation. In *ATVA*, pages 446–463. 2015.
14. Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In *CAV*, pages 379–395. Springer, 2011.
15. Thomas A Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech: A model checker for hybrid systems. In *CAV*, pages 460–463. Springer, 1997.
16. Zhenqi Huang, Chuchu Fan, Alexandru Mereacre, Sayan Mitra, and Marta Kwiatkowska. Invariant verification of nonlinear hybrid automata networks of cardiac cells. In *CAV*, pages 373–390. Springer, 2014.
17. Xiaoqing Jin, Jyotirmoy V Deshmukh, James Kapinski, Koichi Ueda, and Ken Butts. Benchmarks for model transformations and conformance checking. In *ARCH*, 2014.
18. Xiaoqing Jin, Jyotirmoy V Deshmukh, James Kapinski, Koichi Ueda, and Ken Butts. Powertrain control verification benchmark. In *HSCC*, pages 253–262. ACM, 2014.
19. Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. dreach: δ-reachability analysis for hybrid systems. In *TACAS*, pages 200–205. Springer, 2015.
20. Kim G Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):134–152, 1997.
21. Truong Nghiem, Sriram Sankaranarayanan, Georgios Fainekos, Franjo Ivancić, Aarti Gupta, and George J Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *HSCC*, pages 211–220. ACM, 2010.
22. Sigurd Skogestad and Ian Postlethwaite. Multivariable feedback control-analysis and design: Solutiona manual part i. 2005.