Finding Errors of Hybrid Systems by Optimising an Abstraction-Based Quality Estimate

Stefan Ratschan[1] and Jan-Georg Smaus[2]

[1] Academy of Sciences of the Czech Republic, stefan.ratschan@cs.cas.cz
[2] University of Freiburg, Germany, smaus@informatik.uni-freiburg.de

1 Introduction

Hybrid systems are a formalism for modelling embedded systems. An important problem is to ensure correctness, i.e., *verification*. However, during the design process, hybrid systems are usually not correct yet, and hence *error detection* is equally important. We address here the problem of automatically finding error trajectories that lead the system from an initial to an unsafe state. In contrast to previous works [1, 3], we consider systems with deterministic evolution. Moreover, we do not assume a-priori that our system is incorrect, but rather, we interleave verification, using abstractions of the system [4], and falsification attempts.

We define a real-valued function (the *quality estimate*) that approximates the notion of a given point being close to an initial point of an error trajectory. Then we use numerical optimisation to search for an optimum of this function. The function is computed from a simulation of the hybrid system, using information from the abstraction. For each simulation, the point at which it is cancelled depends on a quality estimate computed on-the-fly. The accuracy of the quality estimate improves as the abstraction is refined.

Analysing the related work, one sees that methods designed for non-deterministic systems [1, 3] try to fill the state space as much as possible (according to some measure) with simulations. As a result, they would start a huge number of simulations in parallel—either from a grid (similar to our naïve algorithm from Sec. 3) or from random sample points. In the case of highly non-deterministic systems, such a strategy is promising since the probability of hitting upon an error trajectory is high. However, for systems with only a small amount of non-determinism, and especially, completely deterministic evolution, this creates a huge number of useless simulations. We avoid this by guiding our search using abstractions in order to quickly arrive at a simulation close to an error trajectory.

Our work has some resemblance with *pure optimisation* problems in artificial intelligence [5]. It is distinctive of our work that the objective *function* itself improves over time. Our work also resembles reinforcement learning [6], because we compute the quality as we do the simulation, and depending on this quality we will do other simulations in the neighbourhood.

2 Hybrid Systems and Abstractions

Hybrid systems are systems with both continuous and discrete state. A hybrid system has a finite set S of *modes* and n continuous variables. In each mode, the behaviour of the variables is controlled by an arithmetic expression involving the variables and their derivatives, called the *Flow* constraint. Each possible transition between modes is controlled by a *Jump* constraint, and there are constraints specifying the initial and unsafe states, respectively. A *trajectory* is a sequence of flows connected by jumps. An *error trajectory* is a trajectory starting in an initial state and ending in an error state. A system is *safe* if it does not have an error trajectory.

An *abstraction* of a hybrid system H is a directed graph whose nodes (the *abstract states*) are subsets of the state-space of H. The transition relation of the abstraction is defined so that each concrete *error* trajectory corresponds to an abstract error trajectory.

Abstractions are useful for verification because if the abstraction is safe, the original system is necessarily also safe. Abstractions can be useful for falsification because the abstract error trajectories narrow down the search space for concrete error trajectories.

We use here a technique that decomposes the state space into hyper-rectangles (*boxes*), as implemented in the tool HSolver [4]. In HSolver, an abstraction that is not fine enough yet to verify the desired property is refined by *splitting* a box.

A *simulation* is an explicitly constructed sequence of points in Φ corresponding to the points of a trajectory at discrete moments in time.

3 The Search Algorithm

We want to find an error trajectory of a hybrid system. Since we focus on deterministic systems, the problem reduces to determining the startpoint of an error trajectory among the initial states. A naïve solution to our problem would be obtained by running simulations of a certain length starting at points lying on a grid covering the entire state space. If no error trajectory is found, the grid width should be reduced and the simulation length increased.

The aim of our work is to find an error trajectory with as little simulation effort as possible. More precisely, we want to: (a) interleave falsification with verification; (b) start simulations at the most promising points; (c) cancel simulations when they do not look promising enough anymore. To address these three aims we designed a search procedure that exploits information available from verification. The procedure uses a quality estimate for simulations to determine which startpoints are the most promising, and when to cancel a simulation.

The quality estimate measures how close the simulation *eventually* gets to an unsafe state. We compute the closeness of all individual simulation points to an unsafe state, and take the optimum of these. Note that this optimum can be easily computed on-the-fly.

For an individual point p, we want to estimate how far p is from an error state along the current simulation. To do so, we use information from the abstraction. We interpret the HSolver abstraction in a geometrical way as illustrated in the figure to the right. Here a_0 is an initial abstract state and a_4 is an unsafe abstract state. The dashed lines are the line segments between connected abstract states. For the point p, the estimated distance is the length of the solid line segment sequence. Note that the abstraction approximates the actual trajectories, and in particular, the present abstraction is sufficiently fine to capture the fact that the trajectories first move away from a_4 before approaching a_4. Thus the quality estimate will capture that moving roughly along the solid line leads towards the unsafe state. The quality estimate will become more faithful as the abstraction becomes refined.

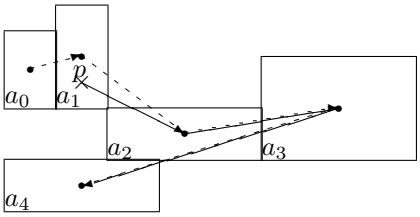

Fig. 1: The distance estimate

The falsification is interleaved with verification. The falsification algorithm is called from the verification after a refinement whenever an initial abstract state is split.

We understand our search problem as the problem of optimising the quality estimate. We use direct search methods [2], specifically the *compass method*. The method takes an initial box I and an n-dimensional cross that fits exactly into I. For the midpoint and each cross tip, we start a simulation and compute the quality estimate f. If f attains an optimum in some cross tip, we move the cross to this cross tip and continue. Otherwise, we halve the size of the cross and continue. The compass method terminates when either the number of cross shrinkings or of cross moves has exceeded a threshold.

We cancel simulations when an unsafe state is hit and thus we have found an error trajectory. Moreover, we cancel a simulation when the quality estimate has not improved for *sim_cnc* steps. There is of course the risk that a simulation is cancelled too early.

	our algorithm					naïve algorithm		
Example	time	ref.	sim.	sim. steps	jumps	time	sim.	sim. steps
convoi	0.5	0	1	7	0	∞	∞	∞
eco *sim_cnc*=400	0.1	0	1	328	2	0.1	1	313
eco	2.8	10	87	29027	2	0.1	1	313
focus	0.1	0	9	2312	0	0.04	1	131
focus *sim_cnc*=20	33.9	434	319	14176	0	0.04	1	131
parabola *sim_cnc*=105	0.0	0	1	201	0	∞	∞	∞
parabola *sim_cnc*=30	18.3	353	113	7665	0	∞	∞	∞

Table 1. Experiments

This risk is countered by the fact that our abstraction is refined over time so that the quality estimate will eventually be faithful enough to tell whether a simulation is "really" improving.

4 Implementation and Experiments

We ran experiments on modifications of various well-known benchmarks from the literature, see `http://hsolver.sourceforge.net/benchmarks/falsification`. Since the benchmarks were mostly safe, we injected an error into those systems.

Table 1 shows the results for a small selection of benchmarks. The table shows the runtime in seconds, the number of abstraction refinements, simulations, the total number of single simulation steps, and the number of jumps of the trajectory that was found, and some figures for the *naïve* algorithm of Sec. 3.

The naïve algorithm performs very well on some apparently easy examples, where the method we propose here also performs well, but on numerous examples it does not terminate within several hours, indicated by ∞.

We did an experiment with `focus` showing that even for a too small value of *sim_cnc*, simulations will eventually "survive" long enough thanks to the refinement of the quality function. The example is extremely easy for HSolver, provided *sim_cnc* is not too small, but hard otherwise. The same effect occurred for `eco`. We have also created an example where we isolate the aspect just mentioned: `parabola`. In this example, the error trajectory looked for is an extremely tight parabola, i.e., at the beginning, one must move away from the unsafe state. If *sim_cnc* is too small and the quality function is not faithful enough yet, then the simulations will be cancelled prematurely.

References

1. A. Bhatia and E. Frazzoli. Incremental search methods for reachability analysis of continuous and hybrid systems. In Rajeev Alur and George J. Pappas, editors, *HSCC'04*, number 2993 in LNCS. Springer, 2004.
2. Tamara G. Kolda, Robert Michael Lewis, and Virginia Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, 45(3):385–482, 2003.
3. Erion Plaku, Lydia Kavraki, and Moshe Vardi. Hybrid systems: From verification to falsification. In Holger Hermanns and Werner Damm, editors, *Proceedings of the 19th International Conference on Computer Aided Verification*, volume 4590 of *LNCS*, pages 463–476. Springer-Verlag, 2007.
4. Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. *ACM Transactions in Embedded Computing Systems*, 6(1), 2007.
5. Stuart J. Russell and Peter Norvig. *Artificial Intelligence: a Modern Approach*. Series in artificial intelligence. Prentice Hall, 2003.
6. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. The MIT Press, 1998.