Automating Numerical Parameters Along the Evolution of a Nonlinear System

Automating Numerical Parameters Along the Evolution of a Nonlinear System

Luca Geretti[1](✉), Pieter Collins[2], Davide Bresolin[3], and Tiziano Villa[1]

[1] Università di Verona, Verona, Italy
{luca.geretti,tiziano.villa}@univr.it
[2] Maastricht University, Maastricht, The Netherlands
pieter.collins@maastrichtuniversity.nl
[3] Università di Padova, Padova, Italy
davide.bresolin@unipd.it

Abstract. When analysing cyber-physical systems for runtime verification purposes, reachability analysis can be used to identify whether the set of reached points stays within given safe bounds. If the system dynamics exhibits nonlinearity, approximate numerical techniques (with rigorous numerics) are often necessary when dealing with system evolution. Since the error involved in numerical approximation should be kept low to perform verification successfully, the associated processing and memory costs become relevant especially when runtime verification is considered. Given a reachability analysis tool, the issue of controlling its numerical accuracy is not trivial from the user's perspective, due to the complex interaction between the configuration parameters of the tool. As a result, user intervention in the tuning of a specific problem is always required. This paper explores the problem of automatically choosing numerical parameters that drive the computation of the finite-time reachable set, when the configuration parameters of the tool are specified within bounds or lists of values. In particular, it is designed to be performed along evolution, in order to adapt to local properties of the dynamics and to reduce the setup overhead, essential for runtime verification.

1 Introduction

In the verification of a generic cyber-physical system, modeling nonlinearity is important in order to accurately capture the interaction of the digital control with the continuous environment. In fact, studying the full interaction between controller and environment, where continuous variables evolve in a possibly nonlinear way, represents a significantly harder problem compared to the analysis of a digital controller in partial isolation. However, the formal methods community in recent years has shown that the approach is feasible and applied it to different systems especially in the field of robotics (such as [2,7,13]).

https://doi.org/10.1007/978-3-031-17196-3_22

The formalism of *hybrid automata* is commonly used to define the composition of controller and environment along with the semantics for its behavior (see [3,6]). *Reachability analysis* in particular is concerned with the computation of the *reachable set*, i.e., the set of points reached from an initial set that evolves under the system's dynamics. Given a dynamical system, obtaining its reachable set allows to reason about its behavior, where a safety specification is represented as geometric constraints on the reached points.

For linear hybrid systems, tools like HyPro [17] and SpaceEx [11] allow an efficient representation of the evolution of the system. For nonlinear systems, computing the reachable set is more problematic. To solve this issue, different solutions are proposed in the literature, using either a numerical or symbolic approach: see the tools Ariadne [4], CORA [1], Flow* [8], HSolver [15], JuliaReach [5] and KeYmaera X [12] for some examples. In this paper we focus on a numerical approach based on computing over-approximations of the reachable set.

Regardless of the preferred tool, it is apparent that automation plays a very important role when approximate representations of the reachable set are involved. Typically, the user needs to provide sensible values for *configuration parameters* such as the integration step size or the polynomial order of a set representation. These parameters usually affect the quality of numerical approximation or enable/disable specific features, but in general they control the over-approximation error. The problem is that the optimal values of the parameters are difficult to know before the system under analysis is properly understood. As a result, the user ends up refining configuration parameter values iteratively until an acceptable result is obtained. This operation, usually done manually with a trial-and-error approach, can become very time-consuming. This is especially important for systems exhibiting nonlinear dynamics, where symbolic approaches are more difficult to pursue and where evaluating the reachable set can be computationally intensive. In any case, the overhead due to user interaction with the tool is non-negligible, in the worst case requiring to spend hours observing the behavior of the system and repeatedly trying with different configurations, for lack of an intuition on the complex interaction among all configuration parameters. It is apparent that such approach does not work very well for runtime verification, where real time constraints are incompatible with manual tuning.

Hence, in this paper we propose a methodology for automated choice of the values of configuration parameters. Differently from approaches that compute a sequence of converging approximations to the exact result, the methodology aims to solve the problem within a single run of execution in order to be compatible with runtime verification. The approach exploits concurrency by executing a single step of evolution on multiple configurations. The user is required to supply only a reasonable range of values for each configuration parameter of interest, hence its active role in optimising the reachable set calculation is greatly reduced. In particular, safety specifications drive optimisation by generating geometric constraints to hold along evolution. The distance to the specification is the metric that allows to rank the various configurations and consequently use the most effective parameter values required. This approach has the advantage that the chosen parameter values have *local* validity, i.e., they are selected at each integration step, and thus they adapt across reachability problems and also across different integration steps of the same problem, since different subspaces usually have different dynamical responses.

This approach should be considered a heuristic compared to, say, the generation of invariants from a formal analysis of the dynamics [16]. In fact, rigorously guaranteeing bounds on the numerical over-approximation error would mean to rely on worst-case formulae for error. Those in turn would yield overly-conservative values for numerical configuration parameters, resulting in excessive accuracy (e.g., a smaller integration step size than what is actually necessary). By pursuing a heuristic approach instead we privilege performance.

The methodology proposed is generic enough to be adopted by any tool that performs approximate reachability by integration of nonlinear vector field dynamics from an initial set. The actual implementation and the corresponding experimental evaluation comes from a general framework for configuration tuning available in the ARIADNElibrary. ARIADNEdiffers from existing packages since it is based on the theory of computable analysis and on a rigorous *function calculus* to achieve provable approximation bounds on the computations [9,10].

In this work we describe the methodology for the continuous behavior, while the extension to hybrid systems will be the subject of a future publication. We remark that the actual challenge lies in the continuous aspects of evolution, so we deem the continuous benchmark in the reported experiments sufficient to confirm the validity of the approach.

To the best of our knowledge, this is the first work that addresses automation in such a general way, in particular for nonlinear dynamics where numerical approaches are preferred. Current tools seem to focus specifically on automated refinement of the integration step size or possibly the polynomial order (which were already accounted for in our tool before). A recent work [19] proposed a solution for nonlinear systems, which tunes values of parameters introduced in their reachability algorithm performing linearization. In this solution however parameters are hard-coded, being related to the specific algorithm. On the contrary, our approach is generic in two aspects: 1) the parameters are not defined a priori, meaning that any subset of parameters involved in any reachability algorithm can be chosen for tuning, and 2) no assumptions on the impact of a given parameter are made.

In the following, in Sect. 2 we explain the general methodology involved, also describing the details of dealing with a set of configuration values and ranking the results obtained from them. Section 3 briefly comments on some preliminary results obtained by using the methodology in our tool. Finally, Sect. 4 draws the conclusions of the paper.

2 Methodology

The proposed methodology is based on the following assumption:

Error Control Assumption. *Given a system whose finite-time reachable set is within a safe set, it is possible to obtain an over-approximation of the reachable set within the same safe set by controlling the growth of the approximation error.*

In practice this is the assumption on which Computable Analysis [18] relies, stating that the over-approximated output converges to the exact output when the approximated input converges to the exact input, where the initial set and configuration parameter values are the input, and the reachable set R is the output. The objective is not to repeat evolution of the system with progressively finer configuration parameter values, until the over-approximation of the reachable set lies within the safe set. Such approach would be easy from the point of view of tuning configuration parameters: once a reasonable initial configuration is chosen, safety would be ultimately verified by progressively changing configuration parameters to increase the accuracy of computation. Instead, in this paper we aim to identify a strategy that achieves the desired result in a *single* run of evolution by adjusting configuration parameter values along evolution steps.

In order to achieve this result while being as general as possible with respect to the system dynamics, we need to control at each time step k the growth of the corresponding *evolve set* E_k, i.e., the section of the flow tube of evolution. If we control the evolve set, then the *reach set* R_k, i.e., the flow tube between steps $k-1$ and k, is controlled as well according to the integration scheme described later in the paper. In fact, if we focus on rigorous numerical integration, the value of the over-approximation error is bounded by a remainder term whose addition to an under-approximation of the flow tube guarantees the enclosure of the exact flow tube. Additionally, the so-called *reconditioning* operations on an evolve set E_k transform the error into additional parameters in the set representation: this causes a loss of information on the error component of the set along evolution. Finally, based on the local contractive or expansive character of the dynamics, the evolve set may even be reduced for a given evolution step (and the error with it, if reconditioning is used).

2.1 General Approach

Due to the previous considerations, a global strategy for control of the growth of the evolve set is difficult to devise. Conversely, an adaptive control relying on the growth of the evolve set within a single integration step is more feasible. Nevertheless, in order to drive configuration tuning based on safety objectives it is necessary to set some global targets in terms of the growth of the evolve set. To identify these targets we rely on fixed-step *simulation* of the system, as opposed to rigorous evolution. Simulation returns a sequence of approximate points, but its computation cost is mostly negligible with respect to rigorous evolution and consequently can be performed as a pre-analysis phase that gathers valuable information on the expected reachable set.

Once pre-analysis has identified the constraints for error control, rigorous evolution can be performed while tuning configuration parameter values according to their ability to satisfy the constraints. Due to the complex interaction of these numerical parameters with the actual value of the error, we don't pursue an analytical approach based on, e.g., a gradient descent algorithm optimising a cost function. Instead, we rather *rank* the results obtained from running each integration step concurrently on a multitude of configuration values. In order to explore a finite amount of configuration values, we rely on an automated discretisation of the configuration parameters.

Summarising, the approach proposed is divided into the following main phases:

1. Simulate the system in a non-rigorous way, returning a set of timed approximate points
2. Identify the points whose distance from the unsafe set has a (local) minimum and construct a list of timed distances called *targets* $\{\tau_i\}$
3. From each τ_i construct a ranking parameter related to the rate of growth of the evolve set E_k, which will drive the optimisation for rigorous evolution
4. From the safety specification construct additional ranking parameters, which will check if a given reachable subset R_k is possibly unsafe
5. Evolve the system rigorously, where for each evolution step k:
   (a) Select a set of points in the configuration search space
   (b) For each point, concurrently, compute E_k and R_k
   (c) Rank the results using the ranking parameters
   (d) Take the best result as the actual E_k and R_k and generate the next set of search points

Termination happens as soon as the evolution time is hit, or if any safety objective is missed for all points. In the following two Subsections, we provide the necessary details related to the search space and ranking respectively.

2.2 Constructing and Exploring the Search Space

The search space for configuration parameter values is defined as a space over the integers. For parameters defined in the boolean domain or as an enumeration, the conversion onto integer values is trivial by using the index in the enumeration (i.e., between 0 and $n-1$ for an enumeration of size n). For values defined in an interval we need to define a conversion policy. Most commonly, values for continuous parameters are discretised in their use: e.g., if a value ranges between 10^{-2} to 10^{-8} typically, we are driven to change the value by multiplication/division by ten. Given this consideration, we shall define parameters along with the conversion rule they are expected to adopt, for example:

– linear: rounds to the nearest integer;
– log_2, log_{10}: takes the logarithm and rounds to the nearest integer.

By supplying the rule and a conversion back and forth, we can map values for continuous (or relatively dense) parameters into a bounded integer space.

Finally, we must decide how to explore such space. For that purpose, we assume that the search space is also *bounded.* The space is necessarily bounded for enumeration parameters, like e.g. the choice between different integration schemes. For parameters defined within intervals, such as the integration step size, an unbounded approach is feasible and offers more freedom to the user but it has some drawbacks. First, it does not allow to assess the size of the search space, which would be useful to drive the exploration. Second, it does not allow to choose a random initial point in the search space at the beginning of evolution.

For parameters defined as enumerations, any value is acceptable and we shall try all values with equal probability. For parameters defined in intervals instead, it is reasonable to assume that adjacent values return similar results, or equivalently that results obtained by varying the value of the parameter have some regular behavior such as monotonicity or concavity.

The distance between values of an enumeration parameter is taken as 1, i.e., all values are adjacent. In summary, exploration of the search space is made by adjacency, in the case of interval parameters meaning that we either choose one of the two adjacent values randomly, or we choose the only adjacent value of the upper/lower bound of the interval. Given a concurrency level γ, the procedure to evolve a set of γ points in the search space is the following:

1. At the beginning of evolution, construct a random initial point $\hat{P}_0$ in the space of integer representations of parameters, and add it to the set of initial points Π;
2. Choose a random point p from Π and construct a random adjacent point $\hat{p}$; add $\hat{p}$ to Π; repeat 2. until $\|\Pi\| = \gamma$;
3. For each point in Π, convert it into the space of the parameters values and execute the integration step;
4. Outputs from all points are ranked, yielding an ordering of the points in Π;
5. Choose the output from the highest ranked point of Π as the effective output for the step;
6. Exit if the evolution time is hit;
7. Otherwise discard the lowest ranked half of the points and return to 2.

Here the update strategy for Π is very simple but it can be improved upon in any way, for example to discourage the addition of points that have been ranked particularly low in the past.

Note that the concurrent approach also introduces failure tolerance as a byproduct: if one or more integration steps are not successful (e.g., by failing to construct the flow function, a likely occurrence when an evolve set becomes particularly large), we can simply discard the failing configuration points and regenerate Π up to γ using the remaining points.

2.3 Ranking the Search Points

In order to select the best configuration point, it is first necessary to identify ranking parameters (where we use the term *parameter* again, not to be confused with a configuration parameter, for lack of a better term) dependent on the safety specification. We distinguish between *safety* ranking parameters and *optimisation* ranking parameters: the former come directly from the constraints that define the safe set, while the latter are associated to some error growth rate targets (which again depend on the constraints). Both kinds of ranking parameters identify a score when applied to one evolution step.

Safety parameters are the simplest, each associated to a safety constraint function $c(S)$, where if $c(R_k) > 0$ then the reach subset R_k is safe. The score for the ranking parameter is represented by $c(R_k)$ and the threshold for the score is zero. If the threshold is crossed, we say that there is a *hard failure* with respect to the specific ranking parameter.

An optimisation ranking parameter is more complicated, since it uses a target to compare *set growth rates*. The growth rate ρ_k for an integration step k, with starting evolve set E_{k-1} and finishing evolve set E_k, is defined as

$$\rho_k = \frac{|E_k| - |E_{k-1}|}{T_k - T_{k-1}} \tag{1}$$

where $|E_k|$ is a measure of the radius of the set, i.e., ρ_k is the increase of radius in units of evolution time and T_k is the initial time at step k.

In particular, from a constraint we can construct a target $\tau = (W^\tau, T^\tau)$, made of a radius W^τ at a time T^τ, which represents (an approximation of) the maximum flow tube radius that does not intersect the boundary of the safe set. This identifies a target growth rate ρ_k^τ:

$$\rho_k^\tau = \frac{W^\tau - |E_{k-1}|}{T^\tau - T_{k-1}} \tag{2}$$

Here we notice that ρ_k^τ is a value projected from the current step and consequently it adapts to variations of $|E_k|$ along evolution. In particular, it allows to compensate for any positive or negative $\rho - \rho^\tau$ deviation on the successive steps.

The score is given by ρ_k, where ρ_k^τ represents the threshold; when $\rho_k > \rho_k^\tau$ we say that there is a *soft failure* with respect to the ranking parameter. When $T_{k-1} > T^\tau$ instead we say that the target *expired* and the corresponding ranking parameter correspondingly expires, i.e., it is not used for ranking.

Summarising, the i-th ranking parameter yields an individual score σ_i and possibly a soft or hard failure. In order to compare the scores on an equal basis, each score must be normalised based on the best and worst values across all search points. Consequently, the normalised score $\hat{\sigma}_i$ becomes

$$\hat{\sigma}_i = \frac{\sigma_i - \sigma_i^m}{\sigma_i^M - \sigma_i^m} \tag{3}$$

with m and M the minimum and maximum values respectively, where clearly $0 \leq \hat{\sigma}_i \leq 1$ holds. The score function for a search point P then becomes

$$\sigma(P) = \sum_i \hat{\sigma}_i(P) \tag{4}$$

To each score σ we also associate the number of soft failures n_s and hard failures n_h.

In order to establish which of two search points P_1 and P_2 rank higher, we compare score, soft failures and hard failures:

- $n_h(P_1) > n_h(P_2) \implies P_1 < P_2$
- $n_h(P_1) = n_h(P_2) \wedge n_s(P_1) > n_s(P_2) \implies P_1 < P_2$
- $n_h(P_1) = n_h(P_2) \wedge n_s(P_1) = n_s(P_2) \wedge \sigma(P_1) < \sigma(P_2) \implies P_1 < P_2$

where $P_1 < P_2$ means that P_2 is ranked higher, i.e., it is more likely to be chosen as the winning point for the evolution step, as well as being kept as a point for the next step.

In particular, if all the points used for computing the k-th step have at least 1 hard failure, then safety verification necessarily fails and evolution is stopped. Conversely, progress is not prevented by all the points having soft failures: it simply means that it was not possible to satisfy one or more of the target growth rates on the k-th step, but that could be amended in the following steps with the updated target growth rates.

3 Preliminary Experimental Results

In this Section we briefly provide some preliminary results applied to the well-known van der Pol oscillator, often used in the verification community as a benchmark for nonlinear dynamics [14]. Due to space reasons, we only provide summary information on the setup. The number of evolution parameters chosen was 4, with 4–4–4–3 possible discretised values each, yielding 192 points in the search space. We performed evolution with increasing concurrency $\gamma = \{1, ..., 5\}$, where $\gamma = 1$ means that no search is performed and $\gamma = 5$ means that 5 threads are used to evaluate 5 different points. Results were averaged over 1000 tries for each value of γ. For $\gamma = 1$, failure in completing evolution was $\%f = 62.9$, with an average execution time $t_x = 5.2$ seconds. With $\gamma = 2$, we got $\%f = 0.7$ and $t_x = 7.6$, up to $\gamma = 5$ yielding $\%f = 0.0$ and $t_x = 10.8$. Summarising, searching using our approach gave a dramatic decrease in average failures even for minimal concurrency, with a contained increase in execution time, and showed practically no failures with only 5 threads used.

4 Conclusions

In this paper we described a methodology to perform safety verification of a nonlinear system with a significant reduction in the time spent by the user on tuning the tool configuration. The approach leverages concurrent execution of the integration step, where configuration values are explored to optimise their choice. Our preliminary results on a benchmark system show that it is possible to succeed at the task practically 100% of the times within a single run with minimum concurrency used. Conversely, manual tuning using the same configuration space would yield below 40% success. While the framework has been validated in the continuous case, a future publication will cover the extension to the hybrid case in detail. Additionally, different search strategies can be envisioned and implemented. The research activity on tuning is still in its infancy, with no comparable papers at this level of generality in the literature as far as we are aware. From our preliminary hands-on experience using Ariadne, we can already state that the leap in tool usability introduced by this approach is very significant. In particular, by leveraging the corresponding improvement in the ratio between evolution time and processing time, more sophisticated runtime verification routines can be envisioned.

References

1. Althoff, M.: An introduction to CORA 2015. In: Frehse, G., Althoff, M. (eds.) ARCH14-15. 1st and 2nd International Workshop on Applied Verification for Continuous and Hybrid Systems, volume 34 of EPiC Series in Computing, pp. 120–151. EasyChair (2015)
2. Althoff, M., Dolan, J.M.: Online verification of automated road vehicles using reachability analysis. IEEE Trans. Rob. **30**(4), 903–918 (2014)
3. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.-H.: Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In: Grossman, R.L., Nerode, A., Ravn, A.P., Rischel, H. (eds.) HS 1991-1992. LNCS, vol. 736, pp. 209–229. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57318-6_30
4. Benvenuti, L., Bresolin, D., Collins, P., Ferrari, A., Geretti, L., Villa, T.: Assume-guarantee verification of nonlinear hybrid systems with Ariadne. Int. J. Robust Nonlinear Control **24**(4), 699–724 (2014)
5. Bogomolov, S., Forets, M., Frehse, G., Potomkin, K., Schilling, C.: JuliaReach: a toolbox for set-based reachability. In: HSCC 2019 - Proceedings of the 2019 22nd ACM International Conference on Hybrid Systems: Computation and Control, pp. 39–44 (2019)
6. Bresolin, D., Collins, P., Geretti, L., Segala, R., Villa, T., Zivanovic, S.: A computable and compositional semantics for hybrid automata. In: Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control, pp. 1–11 (2020)
7. Bresolin, D., Geretti, L., Muradore, R., Fiorini, P., Villa, T.: Formal verification applied to robotic surgery. In: van Schuppen, J.H., Villa, T. (eds.) Coordination Control of Distributed Systems. LNCIS, vol. 456, pp. 347–355. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-10407-2_40
8. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: an analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 258–263. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_18
9. Collins, P.: Semantics and computability of the evolution of hybrid systems. SIAM J. Control. Optim. **49**, 890–925 (2011)
10. Collins, P., Bresolin, D., Geretti, L., Villa, T.: Computing the evolution of hybrid systems using rigorous function calculus. In: Proceedings of the 4th IFAC Conference on Analysis and Design of Hybrid Systems (ADHS12), Eindhoven, The Netherlands, pp. 284–290 (2012)

11. Frehse, G., et al.: SpaceEx: scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_30
12. Fulton, N., Mitsch, S., Quesel, J.-D., Völp, M., Platzer, A.: KeYmaera X: an axiomatic tactical theorem prover for hybrid systems. In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS (LNAI), vol. 9195, pp. 527–538. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21401-6_36
13. Geretti, L., Muradore, R., Bresolin, D., Fiorini, P., Villa, T.: Parametric formal verification: the robotic paint spraying case study. In: Proceedings of the 20th IFAC World Congress, pp. 9248–9253 (2017)
14. Immler, F., et al.: Arch-comp19 category report: Continuous and hybrid systems with nonlinear dynamics. In: Frehs, G., Althoff, M. (eds.) 6th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH19), volume 61 of EPiC Series in Computing, pp. 41–61. EasyChair (2019)
15. Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation based abstraction refinement. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 573–589. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31954-2_37
16. Sankaranarayanan, S.: Automatic invariant generation for hybrid systems using ideal fixed points. In: Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2010, New York, NY, USA, pp. 221–230. Association for Computing Machinery (2010)
17. Schupp, S., Ábrahám, E., Makhlouf, I.B., Kowalewski, S.: HyPro: A C++ library of state set representations for hybrid systems reachability analysis. In: Barrett, C., Davies, M., Kahsai, T. (eds.) NFM 2017. LNCS, vol. 10227, pp. 288–294. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57288-8_20
18. Weihrauch, K.: Computable Analysis - An Introduction. Texts in Theoretical Computer Science, 1st edn. Springer-Verlag, Heidelberg (2000). https://doi.org/10.1007/978-3-642-56999-9
19. Wetzlinger, M., Kulmburg, A., Althoff, M.: Adaptive parameter tuning for reachability analysis of nonlinear systems. In: HSCC 2021 - Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control (part of CPS-IoT Week) (2021)