Reachability Analysis of Nonlinear Systems Using Hybridization and Dynamics Scaling

Dongxu Li[1], Stanley Bak[3], and Sergiy Bogomolov[1,2]

[1] Australian National University, Australia
dongxu.li@anu.edu.au
[2] Newcastle University, UK
sergiy.bogomolov@newcastle.ac.uk
[3] Safe Sky Analytics, USA
stanleybak@gmail.com

Abstract. Reachability analysis techniques aim to compute which states a dynamical system can enter. The analysis of systems described by nonlinear differential equations is known to be particularly challenging. Hybridization methods tackle this problem by abstracting nonlinear dynamics with piecewise linear dynamics around the reachable states, with additional inputs to ensure overapproximation. This reduces the analysis of a system with nonlinear dynamics to the one with piecewise affine dynamics, which have powerful analysis methods. In this paper, we present improvements to the hybridization approach based on a *dynamics scaling* model transformation. The transformation aims to reduce the sizes of the linearization domains, and therefore reduces overapproximation error. We showcase the efficiency of our approach on a number of nonlinear benchmark instances, and compare our approach with Flow*.

1 Introduction

A hybrid automaton [26] is a widely used model for dynamical systems that exhibit complex mixed discrete-continuous behavior. *Reachability analysis* [30, 22, 12] computes an envelope on the set of the states the hybrid automaton can visit within a given time frame. While efficient approaches and tools exist for hybrid automata with affine dynamics [32, 24, 3, 23, 11, 10, 13], reachability analysis of nonlinear systems remains a challenging problem. The current approaches to analyze nonlinear systems can be roughly categorized as follows:

- *Hybridization based approaches* [26, 5, 25, 6, 4, 7, 27, 1, 9] reduce the analysis of nonlinear systems to the analysis of affine systems with uncertain inputs and thus leverage the power of reachability algorithms for simpler classes of dynamics.
- *Taylor model based approaches* [15, 16] approximate nonlinear dynamics using a Taylor expansion, i.e. a combination of polynomials and an interval remainder. The computation of Taylor models is done by iteratively applying Picard operator.

- *Constraint solving based approaches* [29, 21] encode the reachability problem as a satisfiability modulo theory (SMT) problem. Note that such approaches normally do not provide an explicit representation of the reachable set.
- *Simulation based approaches* compute the reachable set by simulating a hybrid automaton multiple times and then enclosing these simulations into reachability tubes. For example, the tool C2E2 [20] uses annotations to computes reachability tubes. Similarly, the tool Breach [19] employs sensitivity analysis for the same purpose.

In the rest of the paper, we focus on hybridization based approaches for purely continuous nonlinear dynamical systems. We note that the existing hybridization approaches can be mainly classified into *static* and *dynamic* approaches. Static approaches [26, 5, 7, 27, 9] partition the continuous state space and *abstract* nonlinear dynamics with its linear approximation in each of the partitions. The resulting model is then forwarded for further analysis to a reachability analysis tool which supports affine dynamics. Such approaches suffer from the following two limitations. First, as the partition and thus all the *abstraction domains* are fixed prior to the reachability analysis, the analysis cannot make use of any information about the system behavior. Therefore, the partition strategy can be ineffective and inaccurate. Second, state space partitioning usually leads to an exponential number of discrete modes in the resulting hybrid automaton, which might make the reachability analysis computationally infeasible for large dynamical systems. In contrast, in dynamic approaches [25, 6, 4, 1], the construction of abstraction domains is performed on-the-fly and namely is interleaved with the reachability analysis. In particular, a dynamic approach ensures that, for each time moment, the abstraction domain encloses the *currently-tracked set of states*, i.e. the set of states the system is currently at. As a larger domain normally results in a larger linearization error, the effectiveness of dynamic hybridization approaches crucially depends on the choice of the abstraction domains.

Due to system nonlinearity, individual system states can evolve in quite different ways. As a result, the currently-tracked set of states, can *stretch* in course of the analysis. Thus, the abstraction domain can quickly grow as well, which might lead to the drastic increase of the noise to be added to ensure conservativeness of the linearized dynamics. In order to mitigate this issue, in our approach, we combine a hybridization scheme with a model transformation technique named *dynamics scaling*, which works by manipulating the dynamics of the original system and aggregating reachable states over a time segment. We have implemented the proposed techniques and benchmarked them against `Flow*` [15], a state-of-art reachability analysis tool for nonlinear hybrid automata, on a number of challenging benchmarks. We observe that on the majority of the benchmarks our techniques show superior precision and runtime (of 1-2 orders of magnitude). As a consequence, our tool succeeds in verifying more safety properties within a given time limit.

The main contributions of the paper are as follows:

1. We present a novel dynamic hybridization approach to perform reachability analysis of nonlinear continuous dynamical systems, which relies on support-

function set representation. As part of our approach, we employ an enhanced error model for linear time-invariant systems which uses the input set decomposition.
2. We embed into our workflow a dynamic scaling technique, which helps to flatten the reachable sets, and in this way leads to reduction of the hybridization errors. We propose a scaling function that is particularly suitable in the hybridization context. In addition, we automate the process of dynamics scaling using a heuristic.
3. We implement the proposed techniques and evaluate their effectiveness in comparison to `Flow*` on a number of challenging benchmarks with 2-30 state variables.

The rest of the paper is organized as follows. Section 2 presents the necessary mathematical background to introduce our approach. Section 3 describes the hybridization reachability algorithm and the improved error model based on support functions. We present the enhancement of hybridization using dynamics scaling transformation in Section 4. In Section 5, we report the evaluation results. We conclude the paper in Section 6.

2 Preliminaries

In order to describe our method, we first review hybrid automata (Section 2.1) and reachability analysis of affine systems using support functions (Section 2.2).

2.1 Hybrid Automaton

Definition 1 (Hybrid automaton). *A hybrid automaton $\mathcal{H}$ is defined as a tuple, $\mathcal{H} = (\mathcal{M}, \mathcal{X}, Inv, Init, Flow, Trans)$, where $\mathcal{M} = \{m_1, ..., m_k\}$ is a finite set of modes; $\mathcal{X}$ is a finite set of n-dimensional real-valued variables; Inv is a mapping $\mathcal{M} \rightarrow 2^{\mathbb{R}^n}$, and $Inv(m_i)$ defines the invariant condition for the mode $m_i \in \mathcal{M}$; $Init \subseteq \mathcal{M} \times \mathbb{R}^n$ defines the initial condition for variables and the initial mode; Flow is a mapping of the locations to differential equations in the form of $\dot{x} = f(x)$, which defines how variables within a location evolve; Trans is a finite set of discrete transitions $t = (m, g, reset, m')$ that may change the mode of $\mathcal{H}$ from m to m' and update the variables according to $reset$ when the guard condition g is satisfied. A* state *of $\mathcal{H}$ is a tuple $s \in \mathcal{M} \times \mathbb{R}^n$.*

The behaviors of a hybrid automaton are formally described as *runs*, which are alternating sequences of time elapse, during which $\mathcal{X}$ evolves according to *Flow*, and discrete transitions *Trans*, which updates $\mathcal{X}$ on $reset$. A state s is *reachable* if there exists a run that starts from $s_0 \in Init$ and ends at s.

2.2 Reachability Analysis Using Support Functions

We consider bounded-time reachability analysis problem, which aims at computing an over-approximation of the set of reachable states upon time T originating from a set of initial states $\mathcal{X}_0$, denoted as $\mathcal{R}_{[0,T]}(\mathcal{X}_0)$. Efficient reachability

analysis algorithms using *support functions* [30, 32, 22] were proposed for hybrid systems with *Flow* of the affine form $\dot{x} = Ax(t) + u(t),\ u(t) \in \mathcal{U}$, where $\mathcal{U}$ is the uncertain input to the system. These algorithms establish the basis of our work on nonlinear systems.

Support functions. The support function [14] of a compact continuous set $\mathcal{S} \subset \mathbb{R}^n$ given a direction vector $\ell \in \mathbb{R}^n$ is defined as

$$\rho(\ell, \mathcal{S}) = \max_{x \in \mathcal{S}} \ell \cdot x$$

and a set $\mathcal{S}$ is uniquely defined by its support functions on all the directions: $\mathcal{S} = \bigcap_{\ell \in \mathbb{R}^n} \{x \mid \ell \cdot x \leq \rho(\ell, \mathcal{S})\}$. In the case where $\mathcal{S}$ is defined as the intersection of hyperplanes, its support function can be computed by calling linear programmes (LP). Support functions enable efficient implementation of the majority of set operations used in reachability analysis:

- *linear map*: For a linear map $A \in \mathbb{R}^n \times \mathbb{R}^n$, $\rho(\ell, A\mathcal{S}) = \rho(A^T \ell, \mathcal{S})$
- *Minkowski sum*: For sets $\mathcal{S}$, $\mathcal{S}'$, denote their Minkowski sum as $\mathcal{S} \oplus \mathcal{S}'$, then $\rho(\ell, \mathcal{S} \oplus \mathcal{S}') = \rho(\ell, \mathcal{S}) + \rho(\ell, \mathcal{S}')$
- *convex hull*: For sets $\mathcal{S}$, $\mathcal{S}'$, denote their convex hull as $\mathrm{CH}(\mathcal{S}, \mathcal{S}')$, then $\rho(\ell, \mathrm{CH}(\mathcal{S}, \mathcal{S}')) = \max(\rho(\ell, \mathcal{S}), \rho(\ell, \mathcal{S}'))$

Checking emptiness of the intersection between a convex set with a halfspace using support functions is straightforward [31]. Given a set $\mathcal{S}$ and a halfspace $\mathcal{G} = \{x \in \mathbb{R}^n \mid a \cdot x \leq d\}$, where $a \in \mathbb{R}^n$ and $d \in \mathbb{R}$, $\mathcal{S} \cap \mathcal{G} \neq \emptyset$ if and only if $d \geq -\rho(-a, \mathcal{S})$.

Affine reachability algorithms The reachability algorithm for affine dynamics adopts the time discretization scheme with a fixed time step. Given a time step δ, the algorithm overapproximates the reachable states with the union of convex sets $\Omega_0, \ldots, \Omega_{N-1}$, called a *flowpipe*, where $\lceil N = {}^{T}/_{\delta} \rceil$. The approximation relies on the *error model* operating on $\mathcal{X}_0$, $\mathcal{U}$ and δ, which mainly constitutes two operators $\Psi_{[0,\delta]}(\cdot)$ and $\Psi_{\delta}(\cdot)$:

- $\Psi_{[0,\delta]}(\cdot)$ overapproximates the reachable states originating from $\mathcal{X}_0$ over the time *interval* $[0, \delta]$, i.e. $\mathcal{R}_{[0,\delta]}(\mathcal{X}_0) \subseteq \Psi_{[0,\delta]}(\mathcal{X}_0, \mathcal{U})$,
- $\Psi_{\delta}(\cdot)$ overapproximates the disturbance of the system due to the uncertain input, i.e. $\mathcal{R}_{[\delta,\delta]}(\{0\}) \subseteq \Psi_{\delta}(\mathcal{U})$.

Each convex set of the flowpipe is computed as follows:

$$\Omega_0 = \Psi_{[0,\delta]}(\mathcal{X}_0, \mathcal{U}) \tag{1}$$

$$\Omega_{i+1} = e^{A\delta} \Omega_i \oplus \Psi_{\delta}(\mathcal{U}) \tag{2}$$

The instantiation of $\Psi_{[0,\delta]}(\cdot)$ and $\Psi_{\delta}(\cdot)$ differs among different error models.

3 Hybridization with Support Functions

In this section, we describe the hybridization process, and improve one of the essential steps that causes overapproximation error during hybridization.

A nonlinear continuous system can be modeled as a single-mode hybrid automaton with nonlinear dynamics:

$$\dot{x} = f(x),\ x \in \mathcal{X} \tag{3}$$

where $f(\cdot)$ is a locally Lipschitz continuous vector function. For a single-mode hybrid automaton, we mainly refer to its continuous component of its state. Given an initial set of states $\mathcal{X}_0$ defined by a *hyperbox*, i.e. Cartesian product of intervals, and a time horizon T, we aim at computing the set of reachable states of $f(\cdot)$ originating from $\mathcal{X}_0$ over time interval $[0, T]$.

3.1 Overview of Hybridization Scheme

To compute an overapproximation of the time-bounded set of reachable states of a nonlinear system in Eq. 3, the hybridization approach first overapproximates the nonlinear dynamics with affine dynamics, and then performs reachability analysis on the resultant affine system. The overapproximation is performed multiple times, each time restricted to a portion of the state space, which in our case is a hyperbox. The process uses two concepts: (i) *abstraction domain* and (ii) *linearization function*.

Definition 2 (Abstraction domains and linearization functions). *An abstraction domain $\mathcal{D} \subset \mathbb{R}^n$ is a hyperbox enclosing the reachable sets. We denote the center of $\mathcal{D}$ as c. Given an abstraction domain $\mathcal{D}$ and nonlinear dynamics $f(\cdot)$, a linearization function $\mathcal{L}(\cdot)$ applies Jacobian linearization on $f(\cdot)$ with an additive input set:*

$$\mathcal{L}(f(\cdot)) = \begin{cases} \dot{x}(t) = Ax(t) + u(t) \\ u(t) \in \mathcal{U} \end{cases} \tag{4}$$

where A is the evaluation of Jacobian matrix at the domain center c, i.e. $\boldsymbol{J}_f(c)$. $\mathcal{U}$ is the set of conservative inputs such that $\forall x(t) \in \mathcal{D}, f(x(t)) - Ax(t) \in \mathcal{U}$.

One can show that $\mathcal{L}(f(\cdot))$ simulates $f(\cdot)$ in $\mathcal{D}$ and therefore proves the soundness of the hybridization approach.

In addition to Definition 2, our algorithm uses two procedures `next_discrete` and `next_dense`. The procedure `next_discrete` takes a sequence of linearized dynamics and computes an overapproximation of $\mathcal{R}_{[t,t]}(\mathcal{X}_0)$, i.e. the set of reachable states at t time instance. The procedure `next_dense` takes a sequence of linearized dynamics and a reachable set $\mathcal{X}_i$ at discrete time t and computes an overapproximation of $\mathcal{R}_{[0,\delta]}(\mathcal{X}_i)$, i.e. the set of reachable states over the *time interval* $[t, t + \delta]$. The details of these procedures are described in Section 3.2.

The general hybridization algorithm is shown in Algorithm 1. At each step, we first compute a minimal enclosing box of the reachable sets (line 11) and enlarge it by pushing the boundaries outwards for μ distance (line 12). We then take

the enlarged enclosing box as the abstraction domain $\mathcal{D}$ and compute linearized dynamics within the domain (line 13). After we have the linearized dynamics, we attempt to compute the dense-time reachable set at step $i+1$ (line 15). Importantly, we need to ensure that the abstraction domain always contains the reachable sets to maintain the conservativeness (line 16). If $\Omega' \subseteq \mathcal{D}$ holds, we compute an overapproximation of $\mathcal{R}_{[(i+1)\delta,(i+1)\delta]}(\mathcal{X}_0)$, which is required by `next_dense` for further computations (line 18). At the end of the step, we reset μ and advance time (line 19, 20). In case the containment check fails, we increase μ (line 22) and compute a new abstraction domain by creating a box containing both Ω_i and Ω' (line 11) and redo the computation for Ω'. Note that for a readability reason we have omitted checking whether Ω_0 stays within $\mathcal{D}$ in Algorithm 1, although we have implemented this containment checking practically. Correctness of the algorithm follows from the following two observations: (i) $\mathcal{R}_{[i\delta,i\delta]}(\mathcal{X}_0) \subseteq \texttt{next\_discrete}(\mathcal{X}_0, \Theta)$, where Θ is the sequence of $\langle A_j, \mathcal{U}_j \rangle$, $0 \leq j < i$ and (ii) $\mathcal{R}_{[i\delta,(i+1)\delta]}(\mathcal{X}_0) \subseteq \texttt{next\_dense}(\mathcal{X}_i, \Theta) = \Omega_i$. Therefore, $\mathcal{R}_{[0,T]}(\mathcal{X}_0) = \cup_{i=0}^{N-1} \mathcal{R}_{[i,(i+1)\delta]}(\mathcal{X}_0) \subseteq \cup_{i=0}^{N-1} \Omega_i$.

Algorithm 1 differs from [17, 35] mainly in two aspects. Firstly, we construct a new abstraction domain at each time step. In principle, it might be beneficial to avoid doing so as long as the reachable set does not leave the current domain, in order to reduce the runtime cost in computing new linearized dynamics. However, since the domain is constructed by enclosing the reachable sets and enlarged by a small amount, the domain is rarely large enough for multiple steps in practice. Moreover, constructing domains tightly confining reachable sets helps to reduce the linearization errors and consequently improves the precision. Secondly, when there happens a switch in the linearized dynamics, existing approaches [17, 35] take the set of reachable states over a time interval (Ω_i in our notation) as the new initial set for the sebsequent computation, which is, however, not necessary since the switch always happens on a time instance. In contrary, at the $(i+1)^{th}$ step, instead of taking Ω_i as the initial set of states, we compute $\mathcal{X}_i$ which overapproximates the reachable set at the time instance $i\delta$. Therefore, the approximation error in the computation of Ω_i does not propagate. Additionally, as we will see later in Section 3.2, because support functions provide an exact representations for $\mathcal{X}_i$, we do not introduce wrapping effects when switching the abstraction domain.

3.2 Support Functions Computations

Since the hybridization approach uses affine reachability operators, the approximation quality is dependent on the accuracy of the error model. In this section, we present the error model we use for affine reachability analysis and describe the extension to the hybridization context through a recurrent formulation.

Improved Affine Reachability Error Model We use the same error model to compute the dense-time reachable sets (`next_dense`) as [32] and present an improved error model over [22] to compute the discrete-time reachable (`next_discrete`) sets by better approximating the input sets.

Algorithm 1: Hybridization Reachability Algorithm

Input: Initial state: $\mathcal{X}_0$, dynamics: $f(\cdot)$, total steps: N, time step: δ
Output: Sequence of reachable sets: $\{\Omega_0, \ldots, \Omega_{N-1}\}$

```
i ← 0;
μ_0 ← 10^{-9};
μ ← μ_0;
𝒟 ← enclosing_box({𝒳_0});
⟨A_0, U_0⟩ ← ℒ(f(·), 𝒟);
Θ ← List(⟨A_0, U_0⟩);
Ω_0 ← next_dense(𝒳_0, Θ);
Ω' ← Ω_0;
𝒳_1 ← next_discrete(𝒳_0, Θ);
while i < N do
    𝒟 ← enclosing_box({Ω_i, Ω'});
    𝒟 ← bloat(𝒟, μ);
    ⟨A_i, 𝒰_i⟩ ← ℒ(f(·), 𝒟);
    Θ.append(⟨A_i, 𝒰_i⟩);
    Ω' ← next_dense(𝒳_{i+1}, Θ);
    if Ω' ⊆ 𝒟 then
        Ω_{i+1} ← Ω';
        𝒳_{i+1} ← next_discrete(𝒳_0, Θ);
        i ← i + 1;
        μ ← μ_0;
    else
        μ ← 2μ;
        Θ.remove(⟨A_i, 𝒰_i⟩);
```

Lemma 1. *(adapted from [32]) Assuming $\mathcal{X}_i$ overapproximates the reachable set at time $i\delta$, A is the a linear map of the linearized dynamics during the time interval $[i\delta, (i+1)\delta]$, let Ω_i be the convex set defined by:*

$$\Omega_i = \mathrm{CH}(\mathcal{X}_i, e^{A\delta}\mathcal{X}_i \oplus \delta\mathcal{U} \oplus \alpha_\delta B) \tag{5}$$

where $\alpha_\delta = (e^{\|A\|\delta} - 1 - \delta\|A\|)(R_{\mathcal{X}_i} + \frac{R_\mathcal{U}}{\|A\|})$, B denotes the unit ball for the considered norm, $R_{\mathcal{X}_i} = \max_{x\in\mathcal{X}_i} \|x\|$ and $R_\mathcal{U} = \max_{u\in\mathcal{U}} \|u\|$. Then

$$\mathcal{R}_{[i\delta,(i+1)\delta]}(\mathcal{X}_i) \subseteq \Omega_i \tag{6}$$

We refer readers to [32] for the proof. Lemma 1 can be roughly understood as follows. $e^{A\delta}\mathcal{X}_i \oplus \delta\mathcal{U}$ is an overapproximation of the reachable set at time $(i+1)\delta$; the bloating operation and the convex hull operation give the overapproximation of the reachable set over the time interval $[i\delta, (i+1)\delta]$. The bloating factor α_δ is computed such as to ensure the overapproximation.

The support function of Ω_i on ℓ is computed as follows:

$$\rho(\ell, \Omega_i) = \max(\rho(\ell, \mathcal{X}_i), \rho((e^{A\delta})^T\ell, \mathcal{X}_i) + \delta\rho(\ell, \mathcal{U}) + \alpha_\delta\rho(\ell, B) \tag{7}$$

Before we present the improved error model on the computation of discrete-time reachable sets, we introduce the following notations: $\boxdot(\mathcal{S})$, which denotes

the symmetric interval hull of a set $\mathcal{S} \subset \mathbb{R}^n$, is defined as $[-\overline{|x_1|}; \overline{|x_1|}] \times \ldots \times [-\overline{|x_n|}; \overline{|x_n|}]$ where $\forall i : 1 \leq i \leq n$, $\overline{|x_i|} = \max\{|x_i| \mid x \in \mathcal{S}\}$. $|\cdot|$ is the element-wise absolute operation over a matrix or vector. The model relies on the following matrices:

$$\Phi_1(A, \delta) = \sum_{i=0}^{\infty} \frac{\delta^{i+1}}{(i+1)!} A^i, \qquad \Phi_2(A, \delta) = \sum_{i=0}^{\infty} \frac{\delta^{i+2}}{(i+2)!} A^i \tag{8}$$

If A is invertible, Φ_1 and Φ_2 can be computed as $\Phi_1(A, \delta) = A^{-1}(e^{A\delta} - I)$, $\Phi_2(A, \delta) = A^{-2}(e^{A\delta} - I - A\delta)$. Otherwise they can be computed as sub-matrices of a block matrix exponential [22].

The rationale behind the improvement is as follows. Since the error model on the input relies on $\square(A\mathcal{U})$, the symmetric interval hull operation can be too coarse if the input set is not centered around the origin. From this observation, we decompose $\mathcal{U}$ into $\{u_c\} \oplus \mathcal{W}$, where u_c is the geometric center of $\mathcal{U}$ and $\mathcal{W}$ is a set that centers around the origin. This way, we reduce the overapproximation introduced during the symmetric hull operation. The improved error model is formalised by the following lemma:

Lemma 2. *Assuming A is the linear map of the linearized dynamics during the time interval $[i\delta, (i+1)\delta]$, $\mathcal{X}_i$ overapproximates the reachable set at time $i\delta$, let $\mathcal{X}_{i+1}$ be the set defined by*

$$\mathcal{X}_{i+1} = e^{A\delta}\mathcal{X}_i \oplus \Psi_\delta(\mathcal{U}) \tag{9}$$

$$\Psi_\delta(\mathcal{U}) = \delta\mathcal{W} \oplus \varepsilon_{\mathcal{W}} \oplus \Phi_1(A, \delta) \cdot u_c \tag{10}$$

$$\varepsilon_{\mathcal{W}} = \square(\Phi_2(|A|, \delta) \square (A\mathcal{W})) \tag{11}$$

Then $\mathcal{R}_{[(i+1)\delta,(i+1)\delta]} \subseteq \mathcal{X}_{i+1}$.

Proof. See a technical report [34].

Lemma 2 provides a way to compute the discrete-time reachable set of the next time instance given that of the current time instance and the linearized dynamics. As opposed to [22], our model improves the accuracy of the approximation by better handling the uncertainty in the input set. The support function of $\mathcal{X}_{i+1}$ on ℓ is computed as follows:

$$\rho(\ell, \mathcal{X}_{i+1}) = \rho(\ell, (e^{A\delta})^T\mathcal{X}_i) + \rho(\ell, \Psi_\delta(\mathcal{U})) \tag{12}$$

$$\rho(\ell, \Psi_\delta(\mathcal{U})) = \delta\rho(\ell, \mathcal{W}) + \rho(\ell, \varepsilon_{\mathcal{W}}) + \ell \cdot \Phi_1(A, \delta) \cdot u_c \tag{13}$$

Support Function Computations for Nonlinear Systems As described in Procedure 1, the reachability analysis of a nonlinear system is reduced to analyzing a sequence of linearized systems with uncertain inputs. Since we create a hybridization domain for each step, after k steps we have k pairs of $\langle A_i, \mathcal{U}_i \rangle$, using which we can extend the error model for the discrete reachable sets to the nonlinear systems.

Lemma 3. *Given the initial states $\mathcal{X}_0$ and a sequence of linearized dynamics $\Theta = \{\langle A_k, \mathcal{U}_k \rangle\}$ $(0 \leq k < i)$, let $\mathcal{X}_{i+1}$ be the set defined by:*

$$\mathcal{X}_{i+1} = \Big(\prod_{r=0}^{i} e^{A_{i-r}\delta} \Big) \mathcal{X}_0 \oplus \bigoplus_{r=1}^{i} \Big(\prod_{m=0}^{i-r} e^{A_{i-m}\delta} \Big) \Psi_\delta(\mathcal{U}_{r-1}) \oplus \Psi_\delta(\mathcal{U}_i) \tag{14}$$

Then it follows that $\mathcal{R}_{[(i+1)\delta,(i+1)\delta]} \subseteq \mathcal{X}_{i+1}$.

Proof. See a technical report [34].

The support function of $\mathcal{X}_{i+1}$ on the direction ℓ is as follows:

$$\begin{aligned} \rho(\ell,\, \mathcal{X}_{i+1}) = \rho\Big(\prod_{r=0}^{i} (e^{A_r\delta})^T \ell,\, \mathcal{X}_0 \Big) \\ + \sum_{r=1}^{i} \rho(\prod_{m=r}^{i} (e^{A_m\delta})^T \ell,\, \Psi_\delta(\mathcal{U}_{r-1})) + \rho(\ell,\, \Psi_\delta(\mathcal{U}_i)) \end{aligned} \tag{15}$$

The support function of $\Psi_\delta(\mathcal{U}_{p-1})$ and $\Psi_\delta(\mathcal{U}_i)$ can be computed according to Eq. 13. In Eq. 15, the number of linear programs to solve grows linearly in the number of steps i. As a result, the total number of linear programs to solve is quadratic in relation to the number of steps $\lceil T/\delta \rceil$, which can be several thousands in typical cases. Although the result from the computation perspective is polynomial, the number of calls needed to an LP solver is a source of significant slowdown. Nevertheless, by restricting $\mathcal{X}_0$ to be the Cartesian product of intervals of an $n-$dimensional space, all the convex sets involved in Eq. 15 are hyperboxes. And the following well-known property of hyperboxes allows us to compute $\rho(\ell,\, \mathcal{X}_{i+1})$ without calling an LP solver.

Proposition 1 (Support function of a hyperbox). *Given a hyperbox $\mathcal{B} = [a_1, b_1] \times \ldots \times [a_n, b_n]$, the support function of $\mathcal{B}$ on the direction $\ell = (\ell_1, \ldots, \ell_n)$ is given by:*

$$\rho(\ell, \mathcal{B}) = \sum_{i=1}^{n} \ell_i \cdot h_i, \text{ where } h_i = \begin{cases} a_i, & \text{if } l_i \leq 0 \\ b_i, & \text{otherwise} \end{cases} \tag{16}$$

Proposition 1 enables the computation of support functions for a set of hyperboxes in a batch by properly vectorizing the matrix multiplication operations, which leads to some performance gains in practice.

4 Dynamics Scaling for Hybridization

The main source of overapproximation error in hybridization methods comes from the overapproximation of the nonlinear dynamics within the abstraction domains. Instead of hyperboxes, some methods have used simplices [18] or other polyhedra [3] as abstraction domains. However, since individual trajectories may evolve quite differently and end up with reaching different states given a specific

time instance, domains may need to stretch out irrespective of the domain shape, in order to contain all the reachable states within one step. In this section, we propose a dynamics scaling technique applied to the hybridization context, which helps to reduce the error in the reachable sets by properly manipulating the dynamics of the system. The dynamics scaling technique was first proposed in [8] to reduce the error during the conversion of guard conditions for *affine* systems. The main idea behind dynamics scaling is to create an additional mode in the automaton that multiplies the original dynamics by a scaling function. It is also shown in [8] that if the scaling function always outputs a nonnegative number for any states considered, the set of reachable states computed for time-bounded reachability does not change.

We employ the dynamics scaling technique for *nonlinear* system analysis and extend it in two aspects. Firstly, we propose a new scaling function so that the trajectories lagged behind is sped up while others in front are slowed down, therefore, the reachable set is flattened (Fig. 1). As a consequence, the size of abstraction domains is reduced, which eventually leads to less approximation errors and better reachability precision. Secondly, we propose a heuristic approach to select the dwelling time in the scaling mode. To our best knowledge, this is the first attempt to exploit and automate dynamics scaling for reachability analysis of nonlinear systems.

Dynamics Scaling Function Given nonlinear dynamics $f(\cdot)$ and the currently tracked set of states Ω, the scaled dynamics $h(\cdot)$ is detailed as below:

$$h(x) = m \cdot d(x) \cdot f(x) \tag{17}$$

$$d(x) = \frac{1}{\|a\|}(-ax + b) \tag{18}$$

$h(\cdot)$ scales the original nonlinear dynamics by the scaled distance function $d(\cdot)$, which measures the signed distance from the point x to the hyperplane defined by $ax \geq b$. Note that $d(\cdot)$ is nonnegative for any x that satisfies $ax \leq b$. The signed distance is scaled by a constant multiplier m, as we will explain later. We call $m \cdot d(x)$ the *dynamics scaling function*.

Now we describe how we choose the hyperplane $ax \geq b$. At each scaling step, we first evaluate the gradient of $f(\cdot)$ at the center of Ω, denoted as $l' = \frac{\mathrm{d}f(x)}{\mathrm{d}x}|_{x=c_\Omega}$, c_Ω is the geometric center of Ω. Then we use the complementary halfspace of the supporting hyperplane of Ω in the direction l', i.e. $l' \cdot x \geq \rho(l', \Omega)$, as the hyperplane. By scaling dynamics using a signed distance function, the speed of trajectories that are far from the hyperplane is increased while the speed of those near the hyperplane is decreased. As a result, the size of abstraction domains is reduced which in turn leads to smaller linearization errors.

Different from affine systems, the linear map A in our approach is the evaluation of the Jacobian matrix of $f(\cdot)$ at the center of the abstraction domain. The addition of the dynamics scaling function modifies the system dynamics and consequently its Jacobian matrix. Since the error model relies on $\|A\|$, such

linearization brings two possible downsides: i) as $e^{\|A\|\delta}$ grows exponentially in $\|A\|$, a linear map A of a large norm may result in a prohibitively large α_δ and leads the flowpipe to diverge; ii) a linear map A of a small norm slows down the progression of the trajectories and takes more steps to achieve the scaling effect. Therefore, we propose to add a multiplier $m = \frac{\|A_f\|}{\|A_h\|}$ which equates the norm of the scaled and unscaled linearized dynamics, where A_f is the linear map of linearized dynamics of $f(\cdot)$ and A_h is the linear map of the linearized dynamics of $d(\cdot) \cdot f(\cdot)$. We observed in practice that i) the addition of m helps maintaining the magnitude of bloating factors in a reasonable order and ii) a moderate amount of steps in the scaling mode leads to a decent scaling effect.

Heuristics for Dynamics Scaling We apply dynamics scaling periodically during the reachability analysis. To this end, we introduce a parameter *scaling period* $p \in (0, 1)$ to indicate that we perform dynamics scaling after each $\lceil p \cdot T \rceil$ time segment. A smaller period enables dynamics scaling more often and provides a stronger scaling effect. On the other hand, because dynamics scaling introduces more nonlinearity by adding a polynomial term, the time cost of computing the linearization errors could arise. Therefore, p balances the trade-off between a stronger scaling effect and additional nonlinearity and computation runtime.

In order to decide when to enter into a scaling mode and to revert to the original dynamics, we introduce a heuristic to measure the effect of dynamics scaling. The heuristic relies on the following operation: $\square(\mathcal{S})$ denotes the *interval hull* of a set $\mathcal{S} \subset \mathbb{R}^n$, defined as $[\underline{|x_1|}; \overline{|x_1|}] \times \ldots \times [\underline{|x_n|}; \overline{|x_n|}]$. For a reachable set Ω, we approximate the volume of $\mathcal{S}$ by the volume of its interval hull: $\sigma(\Omega) \sim \sigma(\square(\Omega)) = \prod_{i=1}^{n} (\overline{|x_i|} - \underline{|x_i|})$. When $\lceil p \cdot T \rceil$ time segments passes, we enable dynamics scaling and check whether the volume of the Ω would decrease. The system then alters to the scaling mode if the check succeeds, otherwise, it remains in the original mode for the next $\lceil p \cdot T \rceil$ segments. Similarly, when dynamics scaling does not help to decrease the volume of the reachable set, the system exits the scaling mode and reverts to the normal, i.e. unscaled, dynamics.

5 Evaluation

We implemented our techniques in a prototypical tool in Python. We employ NumPy [39] to perform matrix operations. Linearization errors are computed using Kodiak library [38], which provides rigorous bounds for nonlinear global optimization problems using interval arithmetic and Bernstein enclosure. All the experiments were run on a laptop running Ubuntu 16.04 equipped with Intel i7-7600U CPU (2.80GHz, 4 cores) and 16 GB RAM.

5.1 Benchmark Evaluation

We evaluate our tool on a number of nonlinear benchmark instances featuring from 2 to 30 dimensions with the aim to assess efficiency and precision of our approach. We compare our tool with the recent version of `Flow*` that participated in the ARCH competition [2].

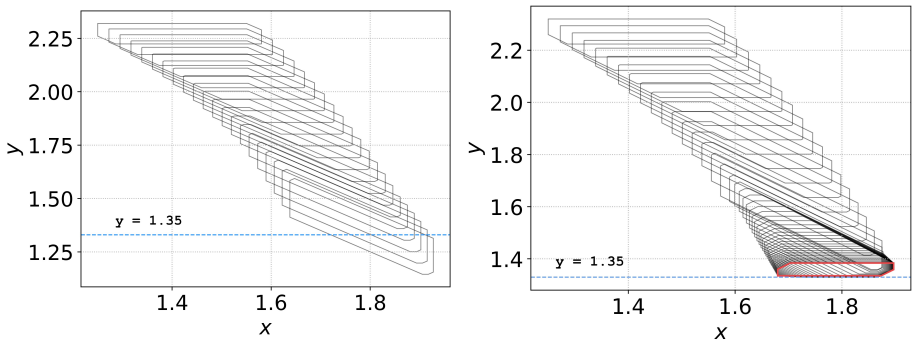


Fig. 1: Effect of dynamics scaling. For the illustration purpose, we apply dynamics scaling towards a hyperplane at $y = 1.35$. In the normal mode (left), the flowpipe passes the hyperplane while the reachable set stretches over y axis. In the dynamics scaling mode (right), the flowpipe contracts to the hyperplane (red). As a result, the size of the abstraction domain reduces.

Experimental setting. Our goal is to verify safety properties of considered benchmark instances. For each benchmark, we first considered a weak safety condition, for which the system can be proven safe easily upon finishing the reachability analysis over the time horizon. In case the condition is too weak to show the difference in precision, we either strengthened the safety condition or increased the time horizon until one or both tools failed to verify the safety. We tuned the parameters of both tools and reported the configurations with the minimal runtime on success, otherwise, the best possible configuration with which the flowpipe still contract. The timeout is set to 900 seconds.

Benchmarks We use the following benchmarks in our evaluation:

Brusselator. The Brusselator is a theoretical model for a class of autocatalytic reaction [36]. The dynamics are given by $\dot{x} = 1 + x^2 \cdot y - 0.5x$, $\dot{y} = 1.5x - x^2 y$. We use the same initial set as [15], i.e. $(x, y) \in [0.8, 1] \times [0, 0.2]$.

Lotka-Volterra. The Lotka-Volterra describes the dynamics of population changes of two species that interact in a predator-prey relation. The dynamics are given by $\dot{x} = x(1.5 - y)$, and $\dot{y} = -y(3 - x)$. We use the same initial set as [15], i.e. $(x, y) \in [4.8, 5.2] \times [1.8, 2.2]$.

Biological models. Biology I, Biology II are benchmarks presented in [15] modeling biological systems from [28]. The dynamics of Biology I (7 dimensions) are given by $\dot{x}_0 = -0.4x_0 + 5x_2x_3$, $\dot{x}_1 = 0.4x_0 - x_1$, $\dot{x}_2 = x_1 - 5x_2x_3$, $\dot{x}_3 = 5x_4x_5 - 5x_2x_3$, $\dot{x}_4 = -5x_4x_5 + 5x_2x_3$, $\dot{x}_5 = 0.5x_6 - 5x_4x_5$, $\dot{x}_6 = -0.5x_6 + 5x_4x_5$. We consider the initial set[4] $x_i \in [0.99, 1.01]$. The dynamics of Biology II (9 dimensions) are given by $\dot{x}_0 = 3x_2 - x_0x_5$, $x_1 = x_3 - x_1x_5$, $\dot{x}_2 = x_0x_5 - 3x_2$, $\dot{x}_3 = x_1x_5 - x_3$, $\dot{x}_4 = 3x_2 + 5x_0 - x_4$, $\dot{x}_5 = 5x_4 + 3x_2 + x_3 - x_5(x_0 + x_1 + 2x_7 + 1)$, $\dot{x}_6 = 5x_3 + x_1 - 0.5x_6$, $\dot{x}_7 = 5x_6 - 2x_5x_7 + x_8 - 0.2x_7$, $\dot{x}_8 = 2x_5x7 - x_8$. We consider the same initial set as [15], i.e. $x_i \in [0.99, 1.01]$.

[4] `https://ths.rwth-aachen.de/research/projects/hypro/biological-model-i/`

(Coupled) Van der Pol Oscillator. The model of a two-dimensional Van der Pol oscillator arises in the study of circuits containing vacuum tubes and is known to exhibit a limit cycle. The dynamics are given by $\dot{x} = y$, $\dot{y} = (1 - x^2) \cdot y - x$. We scaled up the benchmark up by coupling more oscillators in the way similar to the two-coupled Van der Pol oscillators (4 dimensions) [37]. The dynamics of N-coupled Van der Pol Oscillators ($N \geq 2$) are given as $\dot{x_i} = y_i$, $\dot{y_0} = (1 - x_0^2)y_0 - x_0 + (x_1 - x_0)$, $\dot{y_i} = (1 - x_i^2)y_i - x_i + (x_{i-1} - x_i) + (x_{i+1} - x_i)$ $(1 < i < N - 1)$, $\dot{y}_{N-1} = (1 - x_{N-1}^2)y_{N-1} - x_{N-1} + (x_{N-2} - x_{N-1})$. We use the same initial set as [15] and extends it to the high-dimensional instances, i.e. $x_i, y_i \in [1.25, 1.55] \times [2.25, 2.35]$.

(Coupled) Oscillator. We considered the model used in [16] to measure the scalability of the reachability analysis approaches. The model was adapted from [33] that describes the dynamics of synchronization among genetic oscillators. The model consists of N oscillators, each of which is described by five continuous variables. The dynamics are given by $\dot{x_i} = 0.1u_i - 3x_i + \frac{10}{N}\sum_{j=0}^{N-1} v_j$, $\dot{y_i} = 10x_i - 2.2y_i$, $\dot{z_i} = 10y_i - 1.5z_i$, $\dot{v_i} = 2x_i - 20v_i$, $\dot{u_i} = -5u_i^2 z_i^4(10y_i - 1.5z_i)$. We use the same initial set as [16], i.e. $x_i \in [-0.003 + 0.002i, -0.001 + 0.002i]$, $y_i \in [0.197 + 0.002i, 0.199 + 0.002i]$, $z_i \in [0.997 + 0.002i, 0.999 + 0.002i]$, $v_i \in [-0.003 + 0.002i, -0.001 + 0.002i]$, $u_i \in [0.497 + 0.002i, 0.499 + 0.002i]$.

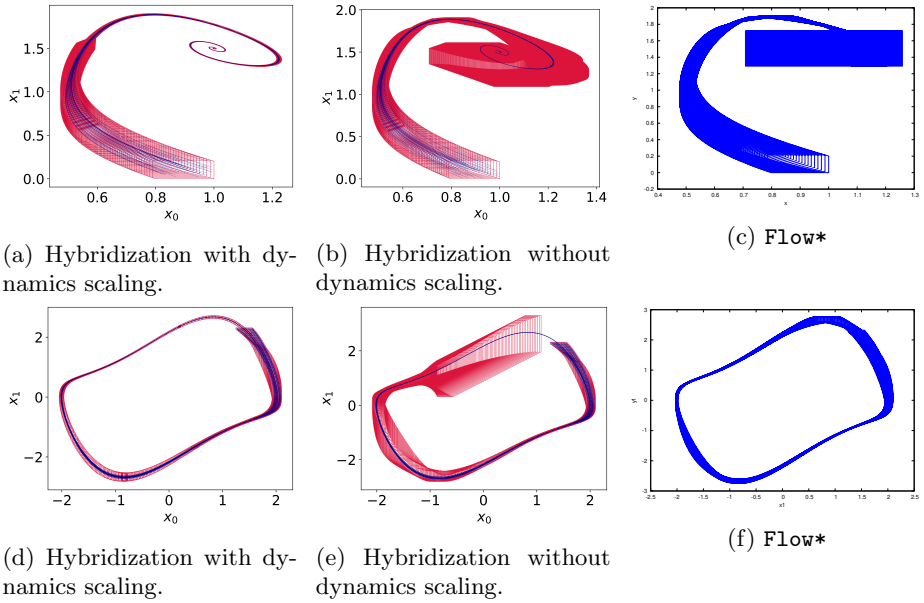
Tuning of the tools We tuned parameters of both tools to minimize the runtime. For our tool, we fixed the dynamics scaling period as 0.1 for all benchmarks except for (coupled) oscillators. We disabled dynamics scaling on (coupled) oscillators due to two observations: i) without dynamics scaling, the precision is sufficient to verify the safety; ii) the benefit of applying dynamics scaling did not always payoff the loss in the runtime on this particular benchmark. For each benchmark, we searched for a minimal time step until i) it is sufficiently small to verify the safety, then we increased it until the safety is violated and reported the largest safe time step; ii) it is so small that the tool timed out, we then reported TO. We used `hypy` [9] to script a grid-search strategy to identify the optimal tuning parameters for `Flow*` . `Hypy` is a Python library that is able to automatically run `Flow*` with a given configuration and parse the result. We first specified a minimal time step, an increment for time step and a subset of Taylor model orders such that `Flow*` could complete the analysis. Our script then exhaustively tried various combinations of Taylor model orders and time step. The time step was increased until safety is violated. We reported the setting that proved the safety with a minimal runtime if one existed. The cut-off threshold in `Flow*` was fixed as 10^{-9} in all the experiments.

Results We show results in Table 1, which leads to the following observations: i) On Brusselator, Biology I, II, (coupled) Van der Pol oscillators benchmarks, our approach was superior in both runtime and precision. In particular, on (coupled) Van der Pol oscillators benchmark, our approach was 10-20 times faster than `Flow*` (2-coupled Vanderpol) and on high dimensional cases (3-coupled Vanderpol, 4-coupled Vanderpol), `Flow*` cannot finish within the time limit.

Table 1: Comparison results on benchmarks. Dim.: Dimension of benchmarks; Horizon: time horizon. Safe: safety property; TM: Taylor model order; δ: time step; t: runtime, TO: The tool/approach timed out after 900 seconds.

Benchmarks	**Dim.**	**Horizon**	**Safe**	**Flow***			**Ours**	
				TM	$\boldsymbol{\delta}$	$\boldsymbol{t}$	$\boldsymbol{\delta}$	$\boldsymbol{t}$
Brusselator	2	10	$y \leq 2$	6	0.04	6.49	0.02	**5.35**
Brusselator	2	25	$y \leq 2$	9	0.001	TO	0.01	**22.46**
Lotka-Volterra	2	3	$y \leq 6$	5	0.01	**2.34**	0.01	5.14
Lotka-Volterra	2	3	$y \leq 5.6$	5	0.01	**2.32**	0.0001	TO
biology I	7	2	$x_3 \geq 0.9$	4	0.02	28.26	0.005	**9.30**
biology I	7	2	$x_3 \geq 0.92$	4	0.01	49.99	0.002	**16.38**
biology II	9	2	$x_6 \geq 10$	7	0.02	TO	0.001	**236.79**
Vanderpol	2	7	$y \leq 3$	5	0.02	**2.42**	0.04	2.83
Vanderpol	2	7	$y \leq 2.7$	12	0.001	TO	0.02	**4.17**
2-coupled Vanderpol	4	7	$y_0 \leq 3$	6	0.02	100.41	0.02	**5.25**
2-coupled Vanderpol	4	7	$y_0 \leq 2.75$	7	0.015	227.76	0.01	**11.48**
3-coupled Vanderpol	6	7	$y_0 \leq 3$	5	0.01	TO	0.005	**72.31**
4-coupled Vanderpol	8	7	$y_0 \leq 3$	5	0.025	TO	0.005	**158.51**
oscillator	5	3	$y_1 \geq 0.08$	4	0.02	**4.17**	0.005	5.99
oscillator	5	3	$y_1 \geq 0.085$	4	0.02	**3.98**	0.0015	31.86
2-coupled oscillator	10	3	$y_1 \geq 0.08$	4	0.02	32.26	0.005	**12.30**
2-coupled oscillator	10	3	$y_1 \geq 0.085$	4	0.02	**31.63**	0.0015	63.97
3-coupled oscillator	15	3	$y_1 \geq 0.08$	4	0.02	140.39	0.005	**22.04**
3-coupled oscillator	15	3	$y_1 \geq 0.085$	4	0.02	**136.99**	0.0015	146.91
4-coupled oscillator	20	3	$y_1 \geq 0.08$	4	0.015	291.88	0.005	**32.46**
4-coupled oscillator	20	3	$y_1 \geq 0.085$	4	0.005	TO	0.0015	**284.61**
5-coupled oscillator	25	3	$y_1 \geq 0.08$	4	0.01	603.98	0.005	**50.03**
5-coupled oscillator	25	3	$y_1 \geq 0.085$	4	0.005	TO	0.0015	**398.5**
6-coupled oscillator	30	3	$y_1 \geq 0.08$	4	0.025	TO	0.005	**73.98**

Further investigations showed that these benchmarks are also the ones where the effect of dynamics scaling was significantly beneficial. The projections of the flowpipe (red) and the numerical simulations (dark blue) of Brusselators (horizon=25 seconds) and 2-coupled Van der Pol oscillators are shown in Fig. 2. We showed the best result `Flow*` produced without exceeding the time limit. ii) On (coupled) oscillators, our tool usually proved the weak safety property faster and scaled better than `Flow*`. By using smaller time steps, our approach can prove the strengthened safety properties using a runtime that is comparable (within 1 order) to `Flow*` on instances with 2 or 3 oscillators. `Flow*` again failed on high



(a) Hybridization with dynamics scaling.

(b) Hybridization without dynamics scaling.

(c) `Flow*`

(d) Hybridization with dynamics scaling.

(e) Hybridization without dynamics scaling.

(f) `Flow*`

Fig. 2: Flowpipes using different approaches/tools on brusselator (above) and 2-coupled Van der Pol oscillators (below) benchmarks.

dimensional cases while our approach succeeded in a reasonable amount of time; iii) `Flow*` showed better precision and speed on Lotka-Volterra model while our approach failed to prove the strengthened property. A possible explanation is that the imprecision due to nonlinearity introduced by applying the dynamics scaling outweighs the benefits of the flattening.This raises the question of how to best use dynamics scaling to improve the precision of flowpipe computation, which we will investigate in the future.

6 Conclusion

In this paper, we have proposed a novel hybridization approach which employs the dynamics scaling model transformation. In this way, we can reduce the size of abstraction domains, which in turn leads to better analysis precision. Our approach uses an enhanced error model to handle affine dynamics based on the input set decomposition. We have shown the effectiveness and precision of our approach by a comparative evaluation against the tool `Flow*` on a number of challenging nonlinear system benchmarks which feature 2 to 30 state variables. In the future, we plan to explore further strategies to guide dynamics scaling.

Acknowledgment. This research was supported in part by the Air Force Office of Scientific Research under award numbers FA2386-17-1-4065 and FA9550-19-1-0288. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Air Force.

References

1. Althoff, M.: Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In: Proceedings of the 16th international conference on Hybrid systems: computation and control. pp. 173–182. ACM (2013)
2. Althoff, M., Bak, S., Chen, X., Fan, C., Forets, M., Frehse, G., Kochdumper, N., Li, Y., Mitra, S., Ray, R., et al.: Arch-comp18 category report: Continuous and hybrid systems with linear continuous dynamics. In: Proc. of the 5th International Workshop on Applied Verification for Continuous and Hybrid Systems. pp. 23–52 (2018)
3. Althoff, M., Le Guernic, C., Krogh, B.H.: Reachable set computation for uncertain time-varying linear systems. In: Proceedings of the 14th international conference on Hybrid systems: computation and control. pp. 93–102. ACM (2011)
4. Althoff, M., Stursberg, O., Buss, M.: Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In: Proc. of the 47th IEEE Conference on Decision and Control (2008)
5. Asarin, E., Dang, T., Girard, A.: Reachability analysis of nonlinear systems using conservative approximation. In: International Workshop on Hybrid Systems: Computation and Control. pp. 20–35. Springer (2003)
6. Asarin, E., Dang, T., Girard, A.: Hybridization methods for the analysis of nonlinear systems. Acta Informatica **43**(7), 451–476 (2007)
7. Azuma, S.i., Imura, J.i., Sugie, T.: Lebesgue piecewise affine approximation of nonlinear systems. Nonlinear Analysis: Hybrid Systems **4**(1), 92–102 (2010)
8. Bak, S., Bogomolov, S., Althoff, M.: Time-triggered conversion of guards for reachability analysis of hybrid automata. In: International Conference on Formal Modeling and Analysis of Timed Systems. pp. 133–150. Springer (2017)
9. Bak, S., Bogomolov, S., Schilling, C.: High-level hybrid systems analysis with hypy. In: ARCH@ CPSWeek. pp. 80–90 (2016)
10. Bak, S., Duggirala, P.S.: Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In: Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control. pp. 173–178. ACM (2017)
11. Bak, S., Tran, H.D., Johnson, T.T.: Numerical verification of affine systems with up to a billion dimensions. arXiv preprint arXiv:1804.01583 (2018)
12. Bogomolov, S., Forets, M., Frehse, G., Podelski, A., Schilling, C., Viry, F.: Reach set approximation through decomposition with low-dimensional sets and high-dimensional matrices. In: 21th International Conference on Hybrid Systems: Computation and Control (HSCC 2018). pp. 41–50. ACM (2018)
13. Bogomolov, S., Forets, M., Frehse, G., Potomkin, K., Schilling, C.: JuliaReach: a toolbox for set-based reachability. In: 22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2019). pp. 39–44. ACM (2019)
14. Borwein, J., Lewis, A.S.: Convex analysis and nonlinear optimization: theory and examples. Springer Science & Business Media (2010)
15. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: International Conference on Computer Aided Verification. pp. 258–263. Springer (2013)
16. Chen, X., Sankaranarayanan, S.: Decomposed reachability analysis for nonlinear systems. In: Real-Time Systems Symposium (RTSS), 2016 IEEE. pp. 13–24. IEEE (2016)
17. Dang, T., Le Guernic, C., Maler, O.: Computing reachable states for nonlinear biological models. In: International Conference on Computational Methods in Systems Biology. pp. 126–141. Springer (2009)

18. Dang, T., Maler, O., Testylier, R.: Accurate hybridization of nonlinear systems. In: Proceedings of the 13th ACM international conference on Hybrid systems: computation and control. pp. 11–20. ACM (2010)
19. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: International Conference on Computer Aided Verification. pp. 167–170. Springer (2010)
20. Duggirala, P.S., Mitra, S., Viswanathan, M., Potok, M.: C2e2: a verification tool for stateflow models. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 68–82. Springer (2015)
21. Franzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. Journal on Satisfiability, Boolean Modeling and Computation **1**, 209–236 (2007)
22. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: Spaceex: Scalable verification of hybrid systems. In: International Conference on Computer Aided Verification. pp. 379–395. Springer (2011)
23. Girard, A.: Reachability of uncertain linear systems using zonotopes. In: International Workshop on Hybrid Systems: Computation and Control. pp. 291–305. Springer (2005)
24. Gurung, A., Deka, A.K., Bartocci, E., Bogomolov, S., Grosu, R., Ray, R.: Parallel reachability analysis for hybrid systems. In: 14th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE 2016). pp. 12–22. ACM-IEEE (2016)
25. Han, Z., Krogh, B.H.: Reachability analysis of nonlinear systems using trajectory piecewise linearized models. In: American Control Conference, 2006. pp. 6–pp. IEEE (2006)
26. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: Algorithmic analysis of nonlinear hybrid systems. IEEE transactions on automatic control **43**(4), 540–554 (1998)
27. Johnson, T.T., Green, J., Mitra, S., Dudley, R., Erwin, R.S.: Satellite rendezvous and conjunction avoidance: Case studies in verification of nonlinear hybrid systems. In: International Symposium on Formal Methods. pp. 252–266. Springer (2012)
28. Klipp, E., Herwig, R., Kowald, A., Wierling, C., Lehrach, H.: Systems biology in practice: concepts, implementation and application. John Wiley & Sons (2008)
29. Kong, S., Gao, S., Chen, W., Clarke, E.: dreach: δ-reachability analysis for hybrid systems. In: International Conference on TOOLS and Algorithms for the Construction and Analysis of Systems. pp. 200–205. Springer (2015)
30. Le Guernic, C.: Reachability analysis of hybrid systems with linear continuous dynamics. Ph.D. thesis, Université Joseph-Fourier-Grenoble I (2009)
31. Le Guernic, C., Girard, A.: Reachability analysis of hybrid systems using support functions. In: International Conference on Computer Aided Verification. pp. 540–554. Springer (2009)
32. Le Guernic, C., Girard, A.: Reachability analysis of linear systems using support functions. Nonlinear Analysis: Hybrid Systems **4**(2), 250–262 (2010)
33. Li, C., Chen, L., Aihara, K.: Synchronization of coupled nonidentical genetic oscillators. Physical Biology **3**(1), 37 (2006)
34. Li, D., Bak, S., Bogomolov, S.: Reachability analysis of nonlinear systems using hybridization and dynamics scaling: Proofs. Tech. Rep. CS-TR-1534, Newcastle University (2020)
35. Matthias, A., Ahmed, E.G., Bastian, S., Goran, F.: Report on reachability analysis of nonlinear systems and compositional verification, `https://cps-vo.org/node/24199`

36. Prigogine, I., Balescu, R.: Phénomènes cycliques dans la thermodynamique des processus irréversibles. Bull. Cl. Sci. Acad. R. Belg **42**, 256–265 (1956)
37. Rand, R., Holmes, P.: Bifurcation of periodic motions in two weakly coupled van der pol oscillators. International Journal of Non-Linear Mechanics **15**(4-5), 387–399 (1980)
38. Smith, A.P., Muñoz, C.A., Narkawicz, A.J., Markevicius, M.: Kodiak: An implementation framework for branch and bound algorithms (2015)
39. Walt, S.v.d., Colbert, S.C., Varoquaux, G.: The numpy array: a structure for efficient numerical computation. Computing in Science & Engineering **13**(2), 22–30 (2011)