Neural Abstractions

Abstract

We present a novel method for the safety verification of non-linear dynamical systems. We train a neural network so as to approximate the system from sample states whilst ensuring an arbitrarily tight bound on the approximation error, which we formally certify using symbolic reasoning. If the latter step refutes the bound, then we augment the samples set with a counterexample and repeat training in a counterexample-guided inductive synthesis loop (CEGIS). We show that, upon successful certification of the bound, this produces a neural ODE with bounded disturbances that constitutes a formal abstraction of the original system, which satisfies a fundamental property: if the abstract system is safe then the original system is safe. Neural networks have extensively been used before as approximators; in this work we make a step further and use them for the first time as abstractions. By using neural ODEs with ReLU activation functions as abstractions, we cast the verification problem for non-linear systems into that of hybrid automata with affine dynamics. We demonstrate that our overall approach is particularly effective for the verification of benchmarks that do not exhibit local Lipschitz continuity, which are out of reach to the existing technologies. Moveover, we demonstrate that it performs comparably to the mature tool Flow* over Lipschitz-continuous non-linear models.

1 Introduction

Dynamical models are ubiquitous in science and engineering, and describe a wide range of continuous- or discrete-time real-world processes whose safety verification is often crucial [7, 30, 34, 74]. The simplest of these models have linear dynamics, but most practical models contain nonlinear terms, which results in more involved or laborious safety analysis and formal verification [4, 13, 22–25, 27, 33, 35, 36, 38, 51, 52, 54, 58, 68, 69]. Whilst the analysis of linear models by and large boils down to linear algebra [11], for nonlinear models analysis and verification problems are much harder, requiring ingenuity and explicitly leveraging the structure of the dynamics; as such results are bespoke to restricted classes of models, often require parameter tuning, and in practice they rely on numerical implementation, which is bound to generate errors. Furthermore, the existence of models that do not exhibit local Lipschitz continuity causes further challenges for technologies that rely on Lipschitz-based assumptions. Despite this, dynamical models with vector fields that are not Lipschitz are abundant in literature, including instances from natural phenomena: from fluid dynamics to n-body orbits and chaotic systems, as well as engineering, such as analogue and digital electrical circuits.

For a given nonlinear model, it may be desirable to construct an approximation of its nonlinear dynamics, resulting in a simpler model, e.g. one with linear, affine (linear, plus drift), or polynomial dynamics. One can then resort to the simpler model for analysis and/or verification [6, 66, 67]. Evidently, such approximation often comes with an error: if one can bound the value of this error, then the approximated model can be cast as an *abstraction*, whereby one can sufficiently derive verification results that can be refined and thus applied to the original nonlinear model. In relevant literature, such abstractions can be either finite-state, such as finite transition systems, or alternatively can retain continuous dynamics: in the latter instance and for this work, we specifically refer to approximations that are structurally piecewise affine, namely affine within partitions of the state

Preprint. Under review.

space, and where the abstraction procedure to generate them is called *hybridisation* [5, 8–10, 15, 18, 31, 32, 39, 45, 50, 53, 56, 57, 59–61, 65, 73]. Hybridisations of general dynamical models relate to models known as *hybrid automata* or *hybrid systems*, whereby multiple modes are associated to the partitions and their corresponding dynamics.

Recently, models developed based on first-principles (also denoted as white-box models) are often replaced by generalistic, black-box architectures, namely models with a given general template that is successively best fit to data observed from the underlying physical process. Examples of parametrised black-box models are linear regressors, (deep) nonlinear neural networks, or linearly-weighted combinations of nonlinear basis functions; alternatively, non-parametrised alternatives are Gaussian processes (GP) or models based on support vector machines (SVM) [62]. In this work, we draw a formal connection from white-box, nonlinear dynamical models to black-box, neural dynamical models, showing that the latter structures can be built as sound abstractions of the original nonlinear models. Such abstractions are generated automatically by means of a counterexample-based inductive synthesis procedure, known as CEGIS [72] (cf. Figure 3, left): this procedure alternates between the training of a ReLU-based neural network approximation (learner), and the computation of a formal upper-bound on the error associated to it (certifier), which formalises it as an abstraction.

In view of the placed emphasis on the usefulness of abstraction for formal verification goals, we employ our ReLU-based abstraction on safety verification tasks, over multiple dynamical models from the benchmark set utilised above. We compare our approach with a state-of-the-art, mature reachability verification software tool, Flow*, performing calculations of reachable states from a given set of initial states - this is known as *flowpipe propagation* [23]; our partition-based hybridisation reachability approach displays comparable results, and is thus deemed promising in view of recent literature that has started developing computational results and tools for such black-box dynamical models [21], [48], for instance to study their stability [29], their robustness [76], and to perform reachability analysis [14, 42, 43]. The last problem is the focus of this work, and allows to perform safety verification, which is core to many theoretical results and practical problems in model-based engineering. Notably, [14, 42, 43] do not calculate deterministic over-approximations of reachable states so are not suitable for use in the safety analysis described in this work.

We summarise our main contributions in the following three points:

- we introduce the novel idea of leveraging neural networks to construct abstractions of continuous-time nonlinear dynamical systems, including those that do not exhibit local Lipschitz continuity;
- we present a procedure for the synthesis of neural abstractions based on counterexample-guided inductive synthesis (CEGIS);
- we design a method to cast neural abstractions as models known as hybrid automata, and use existing tools to perform safety verification on them, to assert the safety of the original nonlinear dynamical models.

We consider there to be no significant negative societal impact of our contributions.

2 Neural Abstractions of Dynamical Models

We study the verification question of whether a continuous-time, autonomous dynamical model with uncertain disturbances is safe within a region of interest and a given time horizon, and with respect to a region of initial states and a region of bad states. We define a dynamical model $\mathcal{F}$ in terms of an n-dimensional vector of real-valued variables x that take values over a compact set $\mathcal{X} \subset \mathbb{R}^n$ which we call region of interest, a nonlinear function $f\colon \mathbb{R}^n \to \mathbb{R}^n$, and a disturbance radius $\delta > 0$. Its dynamics are given by the system of nonlinear ODEs

$$\dot{x} = f(x) + d, \quad \|d\| \leq \delta, \quad x \in \mathcal{X}, \tag{1}$$

where $\|\cdot\|$ denotes a norm operator (unless explicitly stated, we assume all norms are given the same semantics across the paper). A trajectory of $\mathcal{F}$ defined over time horizon $T > 0$ is a differentiable function $\xi\colon [0, T] \to \mathbb{R}^n$ such that, for all $t \in [0, T]$, it holds true that $\xi(t) \in \mathcal{X}$ and $\dot{\xi}(t) = f(\xi(t)) + d$ for some $\|d\| < \delta$. The symbol d can be interpreted as a non-deterministic disturbance, which at any time can take any possible value within the disturbance bound provided by δ.

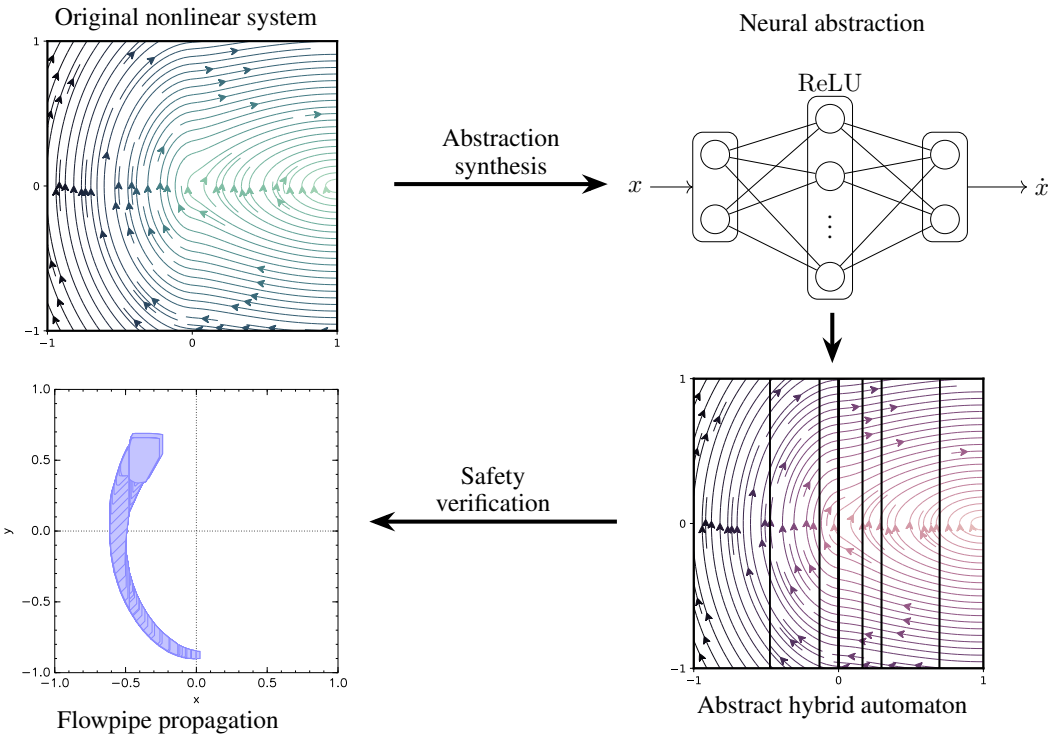

Figure 1: Overview of safety verification of a nonlinear non-locally Lipschitz (NL2) dynamical model using neural abstractions, illustrating flowpipe propagation through a region of non-Lipschitz continuity. The original dynamics are approximated using a neural ODE, from which an abstraction is constructed using an bound on the approximation-error. The abstraction is in the form of a hybrid automaton with affine dynamics, with which safety verification can be performed to analyse the original dynamics.

Let the compact sets $\mathcal{X}_0 \subset \mathcal{X}$ be a region of initial states and $\mathcal{X}_B \subset \mathcal{X}$ be a region of bad states. We say that the trajectory is safe with respect to $\mathcal{X}_0$ and $\mathcal{X}_B$ if $\xi(0) \in \mathcal{X}_0$ and $\xi(t) \notin \mathcal{X}_B$ for all $t \in [0, T]$; dually, we say that it is unsafe if $\xi(0) \in \mathcal{X}_0$ and $\xi(t) \in \mathcal{X}_B$ for some $t \in [0, T]$. The safety verification question consists of determining whether all trajectories of $\mathcal{F}$ defined over a given time horizon are safe with respect to some given regions of initial and bad states. If so, we say that the system is safe.

We tackle safety verification by abstractions, that is, we construct an abstract dynamical system that captures all behaviours the original system. In other words, if the original system is safe, then the abstraction is necessarily safe too [9]. We then apply a verification procedure over the abstraction, allowing us to conclude that the original nonlinear model is safe. We define our abstract dynamical system in terms of a feed-forward neural network with ReLU activation functions, and endowed with an error bound (cf. Fig. 1): this model can be seen as a neural ODE [21] with the addition of a bounded disturbance.

Our feed-forward neural network consists of an n-dimensional input layer y_0, k hidden layers $y_1, \ldots, y_k$ with dimensions $h_1, \ldots, h_k$ respectively, and an n-dimensional output layer y_{k+1}. Each hidden or output layer with index i are respectively associated matrices of weights $W_i \in \mathbb{R}^{h_i \times h_{i-1}}$ and a vectors of biases $b_i \in \mathbb{R}^{h_i}$. Upon a valuation of the input layer, the value of every subsequent hidden layer is given by the following equation:

$$y_i = \mathrm{ReLU}(W_i y_{i-1} + b_i). \tag{2}$$

Whereas many activation functions exist, we focus our study on ReLU activation functions, applying function $\max\{x, 0\}$ to every element $x \in \mathbb{R}$ of its h_i-dimensional argument. We emphasise that each neuron within a given hidden layer induces a hyperplane in the corresponding vector space of that layer, resulting in two half-spaces: when the ReLU is enabled and when it is disabled. For the jth neuron in the ith layer, this hyperplane is given by

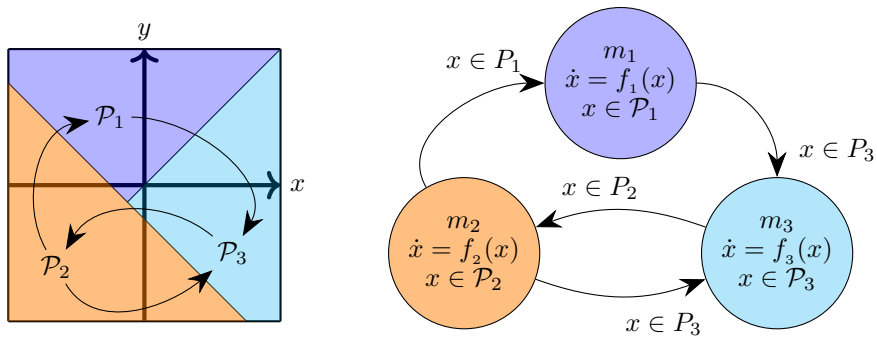

Figure 2: A hybrid automaton $\mathcal{H}$ corresponding to a state-space partitioning. Each of the 3 modes, m_1, m_2, m_3, corresponds to a unique partition $\mathcal{P}_m$ and vector field $f_m(x)$. The transitions are denoted by the edges of the directed graph, with a transition $t = (m, m')$ existing between two modes if the corresponding partitions $\mathcal{P}_m$ and $\mathcal{P}'_m$ are adjacent and a trajectory from f_m "crosses" the corresponding partition.

$$W_{i,j} y_{i-1} + b_{i,j} = 0, \tag{3}$$

where $W_{i,j}$ is the jth row of the weight matrix W_i, and $b_{i,j}$ is the jth element of the bias b_i. Finally, the value of the output layer is given by the affine map $y_{k+1} = W_{k+1} x_k + b_{k+1}$. Altogether, the network results in a function $\mathcal{N}$ whose output is $\mathcal{N}(x) = y_{k+1}$ for every given input $y_0 = x$.

Definition 1 (Neural Abstraction)**.** *A feed-forward neural network $\mathcal{N} \colon \mathbb{R}^n \to \mathbb{R}^n$ together with an error bound $\epsilon > 0$ define a neural abstraction $\mathcal{A}$ for the dynamical system $\mathcal{F}$ given by function f and disturbance radius δ, over a region of interest $\mathcal{X}$, if it holds true that*

$$\forall x \in \mathcal{X} \colon \|f(x) - \mathcal{N}(x)\| \leq \epsilon - \delta. \tag{4}$$

Then, the neural abstraction $\mathcal{A}$ consists of the following dynamical systems defined by a neural ODE with bounded disturbances:

$$\dot{x} = \mathcal{N}(x) + d, \quad \|d\| \leq \epsilon, \quad x \in \mathcal{X}. \tag{5}$$

Theorem 1. *If $\mathcal{A}$ is a neural abstraction for a dynamical system $\mathcal{F}$ over a region of interest $\mathcal{X} \subset \mathbb{R}^n$, then every trajectory of $\mathcal{F}$ is also a trajectory of $\mathcal{A}$.*

Proof. Let ξ be a trajectory of $\mathcal{F}$ and T be the time horizon over which ξ is defined. Let $t \in [0, T]$ and denote $x_t = \xi(t)$ and $\dot{x}_t = \dot{\xi}(t)$. By definition of trajectory we have that (i) $x_t \in \mathcal{X}$ and there exists d_t s.t. (ii) $\|d_t\| \leq \delta$ and (iii) $\dot{x}_t = f(x_t) + d_t$. By (i) and Eq. 4 we have that $\|f(x_t) - \mathcal{N}(x_t)\| + \delta \leq \epsilon$. Then, by (ii) we have that $\|f(x_t) - \mathcal{N}(x_t)\| + \|d_t\| \leq \epsilon$ which, by triangle inequality, implies that $\|f(x_t) + d_t - \mathcal{N}(x_t)\| \leq \epsilon$. We use (iii) and rewrite it into $\|\dot{x}_t - \mathcal{N}(x_t)\| \leq \epsilon$. Finally, we define $d'_t = \dot{x}_t - \mathcal{N}(x_t)$. As a result, we have that $\|d'_t\| \leq \epsilon$ and $\dot{x}_t = \mathcal{N}(x_t) + d'_t$ which, together with (i), show that ξ is a trajectory of $\mathcal{A}$. □

Corollary 1. *With respect to every initial region $\mathcal{X}_0 \subset \mathcal{X}$, bad region $\mathcal{X}_B \subset \mathcal{X}$, and time horizon, if $\mathcal{A}$ is safe then $\mathcal{F}$ is safe.*

Proof. By Thm. 1, if there exists an unsafe trajectory of $\mathcal{F}$ then this is also an unsafe trajectory of $\mathcal{A}$. The statement follows by contraposition. □

Altogether, we define the neural abstraction of a non-linear dynamical system $\mathcal{F}$ as a neural ODE with additive disturbance $\mathcal{A}$ that approximates the dynamics while also accounting for the approximation error. Notably, we place no assumptions on the vector field f. In particular, Thm. 1 does not require f to be Lipschitz continuous. This is because the soundness of a neural abstraction relies on the condition described by (4), whose certification we offload to an SMT solver (see Sect. 3). Then, we cast the resulting neural abstraction to a hybrid automaton with affine dynamics and non-deterministic

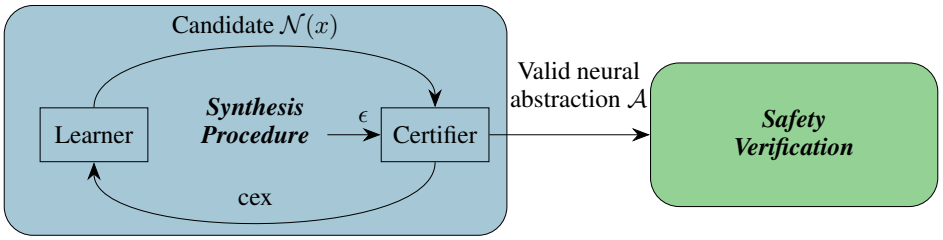

Figure 3: Architechture for safety verification using neural abstractions, including a CEGIS-based synthesis procedure consisting of a learner and a certifier.

disturbance (see Sect. 4), which does not rely on the Picard-Lindelof theorem to ensure uniqueness or existence of a solutions.

We represent a configuration $C = (c_1, \ldots, c_k)$ as a sequence of Boolean vectors $c_1 \in \{0,1\}^{h_1}, \ldots, c_k \in \{0,1\}^{h_k}$, where C represents whether the hidden neurons in the network are enabled (1) or disabled (0). Here, an enabled neuron corresponds to the positive half-space induced by the hyperplane in (3), and a disabled neurons corresponds to the negative half-space. It is clear that a configuration of neurons corresponds to a convex polyhedron $\mathcal{P}_m$, which will be bounded when taken with the intersection of the compact domain of interest $\mathcal{X}$. The polyhedra obtained from all possible neuron configurations induce a partitioning of the domain $\mathcal{X}$, with each polyhedron representing a partition of $\mathcal{X}$. The choice of ReLU activation functions also ensures that the dynamics (vector fields) in each partition are affine.

A consequence of this, and an important observation of our work is that a feed-forward ReLU neural ODE therefore induces a hybrid automaton. Hybrid automata are useful objects that model systems that contain both continuous and discrete dynamics [44, 70]. They consist of a collection of (discrete) modes, each with their own dynamics describing the dynamics of continuous states, and rules for transitioning amongst discrete modes. We denote a hybrid automaton as $\mathcal{H}$ and represent an example in Figure 2, which illustrates the key constituent components relevant for this work.

3 Formal Synthesis of Neural Abstractions

The procedure presented leverages a counterexample-guided, inductive approach to synthesis, known as CEGIS. First introduced in [72] for program synthesis, CEGIS has found many uses ranging from synthesis of invariants, stability analysis or unbounded-time safety verification through the synthesis of barrier certificates [2, 3, 19, 20, 28, 49, 63, 64].

CEGIS is an iterative procedure consisting of two opposing components: a numerical-based learner which seeks to learn a program or function that satisfies some conditions over a finite data set, and a formal certifier which seeks to certify the veracity of the learner's solution. If the certifier determines this solution is invalid, it provides a counterexample - i.e., a point within the state space where the desired conditions are invalid.

As previously described, the learner in this procedure is a ReLU-based neural network; the certification role is performed by an SMT-solver. Several options exist for the selection of the SMT solver used, with the requirement that the solver should reason over quantifier-free nonlinear real arithmetic formulae. The SMT-solver used in this work is dReal [41], which is advantageous in that it can be used to query satisfiability over non-polynomial formulae, including nonlinear functions with transcendental terms.

We shall now consider each component of the CEGIS loop depicted in Figure 3 in turn. In Sect. 3.1 we present the learning component and in Sect. 3.2 we present the certification component used to certify the neural abstractions.

3.1 Training of ReLU Network

As with any machine learning-based algorithm, learning neural abstractions hinges on the loss function used as part of the gradient descent scheme for optimising parameters. The task is that of a

regression problem, so the choice of loss function to be minimised is simple, namely,

$$\mathcal{L} = \sum_{s \in S} \|f(s) - \mathcal{N}(s)\|_2, \tag{6}$$

where $\|\cdot\|_2$ represents the $2-\texttt{norm}$ of its input, and S represents a finite set of data points that are sampled over $\mathcal{X}$. In other words, the neural abstractions are synthesised using a scheme based on gradient descent, to find the parameters that minimise the mean square error over the sampled data.

3.2 Verification of the Error Bound of Neural Abstractions

As previously mentioned, certification of the accuracy of the neural abstractions is performed by dReal. This certification step is described and performed as follows. The purpose of the certification is to check that at no point in the domain of interest $\mathcal{X}$ is the maximum error greater than some upper bound ϵ, as in the specification in equation (4). Therefore, the certifier is provided with the negation of the specification, namely

$$\phi = \exists x \in \mathcal{X}\colon\ \|f(x) - \mathcal{N}(x)\| > \epsilon. \tag{7}$$

The certifier seeks an assignment `cex` of the variable x such that ϕ is *true*, namely that the specified bound ϵ is violated. If this search is successful, then the network $\mathcal{N}$ has not achieved the specified accuracy over $\mathcal{X}$, and is thus not a valid neural abstraction. The corresponding assignment `cex` forms the counterexample that is provided back to the learner (the machine learning procedure from the previous subsection). Alternatively, if no assignment is found, then the specification in Eq. (4) is valid and the synthesis is soundly complete.

3.3 Summary of Synthesis Procedure for Neural Abstractions

The procedure we use to synthesise neural abstractions is summarised as follows. The inputs to the procedure are the dynamical model to be abstracted f, the domain of interest $\mathcal{X}$ over which the abstraction is to be generated, and the parameters for the learning. In addition to the learning rate, these include the stopping criterion for the learning procedure. For the latter, there are two possible options: a target error which all data points must satisfy, or a bound on the value of the loss function. We sample a set of data points over $\mathcal{X}$ to construct a set of training data S. Then, we train a feed-forward neural network $\mathcal{N}$ which seeks to minimise the `2-norm` error in Eq. (6). The learning procedure terminates depending on the stopping condition. If a target error is provided, this is when all points in the data set S satisfy the specification given by ϕ and a target error. If an alternative loss-based stopping criterion is provided then the learning terminates when this is met. If the stopping criterion is not a target error, then an error bound on the approximation is estimated using the maximum approximation error over the data set S for use in certification.

After learning, the network $\mathcal{N}(x)$ is translated to symbolic form and passed to the certification block, which checks for the specification characterised by ϕ as mentioned. The certifier returns that either ϕ is false and thus no counterexample is generated, or that ϕ is true and a cex supporting this. In practice, such a generated counterexample is augmented by sampling for additional points nearby in order to maximise the efficiency of the CEGIS loop: this new *set of counterexamples* is then returned to the learning procedure.

A successful verification process allows for the full abstraction to be constructed using the achieved error ϵ and neural network $\mathcal{N}(x)$. CEGIS has been shown to perform well and terminate successfully across a wide variety of problems. We demonstrate the robustness of our procedure Appendix B.

4 Safety Verification of Neural Abstractions

Neural abstractions are dynamical systems expressed in terms of neural ODEs with additive disturbances (see Eq. 5). Corollary 1 ensures that concluding that a neural abstraction is safe suffices to assert that the original dynamical model is also safe. Consequently, once a neural ODE is formally proven to be an abstraction for the original system, our method enables a generic procedure for the safety verification of neural ODEs with disturbances to be a valid safety verification procedure for the corresponding dynamical model. While formally reasoning about the original model in relation to the neural abstraction is delegated to our synthesis procedure (Sect. 3), the safety

verification procedure needs only to reason about the dynamics of the neural abstraction. Recent safety analysis approaches for dynamical systems controlled by neural networks solve a similar problem [12, 37, 46, 47, 71, 75, 77, 78], yet with a subtle difference: neural network controllers take control actions at discrete points in time. By contrast, neural ODEs characterise the dynamics over continuous time horizons. Approaches for the verification of neural ODEs have been only recently introduced [42, 43], however they are non-deterministic tools as they only provide a probabilistic bound on whether the generated flowpipes over-approximate reach sets. For the verification of neural abstractions, we design a procedure to construct a hybrid automaton from it, and rely on an existing tool for the safety verfication of hybrid automata: SpaceEx [40]. SpaceEx performs computation of over-approximations of reachable sets for hybrid automata with affine dynamics and additive non-determinism (disturbances). We now describe how we construct a hybrid automaton from a neural abstraction: by doing so, the neural abstraction is tailored to be compatible with SpaceEx.

4.1 Casting Neural Abstractions as Hybrid Automata

We begin with the observation that a neural ODE with ReLU activation functions induces a hybrid automaton (cf. Fig. 2), where every mode is determined by a configuration of the hidden neurons (active or inactive), and each of these configurations induces a system of affine ODEs. We seek to automatically generate such hybrid automaton that is induced by the neural abstraction. A hybrid automaton $\mathcal{H}$ consists of a finite number of locations (or modes). Each mode is associated with a corresponding partition of the state space, and with dynamics described by a vector field. Transitions represent allowed changes in the discrete location from one mode to another. Notably, for the hybrid automata in this work the state remains in a given mode until the dynamics leave the corresponding partition, whereby the discrete state is reset to whichever mode corresponds to the new partition that the continuous state enters in.

Next, we describe the procedures to determine each of these constituent components, in order to construct the hybrid automaton that corresponds to a neural abstraction.

Modes A mode corresponds to a configuration of neurons of the neural network. We represent a configuration $C = (c_1, \dots, c_k)$ as a sequence of Boolean vectors $c_1 \in \{0,1\}^{h_1}, \dots, c_k \in \{0,1\}^{h_k}$, where k denotes the number of hidden layers and $h_1, \dots, h_k$ denote the number neurons in each of them (see Sect. 2). We restrict the state space of the mode to a region $\mathcal{X}_C \subseteq \mathcal{X}$, which denotes the maximal set of states that enables configuration C. This is defined as follows:

$$\mathcal{X}_C = \left\{ y_0 \;\middle|\; \begin{array}{c} \exists y_1, \dots, y_k \colon \wedge_{i=1}^{k} y_i = \mathrm{diag}(c_i)(W_i y_{i-1} + b_i) \wedge \\ \mathrm{diag}(2c_i - 1)(W_i y_{i-1} + b_i) \geq 0 \end{array} \right\} \cap \mathcal{X}, \tag{8}$$

where the diag operator denotes the square diagonal matrix whose coefficients are those of its argument.

It is clear that a given configuration C is active if and only if the corresponding set $\mathcal{X}_C$, which is in fact a convex polyhedron in $\mathbb{R}^n$, is nonempty; this consists of verifying that the linear program (LP) constructed from the polyhedron is feasible. Finding all modes of the hybrid automaton therefore consists of solving 2^H linear programs, where H is the total number of neurons in the network. However, this exponential scaling with the number of neurons is limiting in terms of network size. Therefore, we propose an approach that works very well in practice to determine all valid neuron configurations.

Tree search / path enumeration to characterise the Modes The approach relies on the observation that within a bounded polyhedron $\mathcal{P}$, a given neuron has two modes (ReLU enabled or disabled) only if the induced hyperplane intersects $\mathcal{P}$. If it does not, only one of the two possible half-spaces contributes to any possible active configuration, and the other neuron mode can be disregarded. Therefore, this approach involves iterating through each neuron in turn and constructing two LPs - one for each halfspace intersected with the domain of interest $\mathcal{X}$. If only one LP is valid, we can fix the neuron to this mode, i.e., from this point onward only consider the intersection with the halfspace corresponding to the feasible LP, and construct a new polyhedron from the intersection of $\mathcal{X}$ and the feasible half-space.

If both LPs are valid, then we add it to a last-in first-out queue of non-fully explored neurons. Next, we choose one of the half-spaces, take the intersection with the current polyhedron and continue.

Once all neurons have been iterated through, we have an enabled configuration. In order to obtain another configuration, we take a neuron from the queue of non-fully explored neurons and choose the opposite half-space to before, then continue iterating through the successive neurons. We must re-compute both LPs for any previous 'fixed' neurons that succeed the non-fully explored neuron in the configuration. Once this queue is empty and the final configuration is obtained then the procedure terminates with all valid configurations found.

In short, we consider the neurons of the network as a binary tree, with the branches representing the enabled and disabled state of this neuron. We walk through this tree by intersecting with the corresponding half-spaces. Upon reaching an end node, we store this configuration (branches taken) and revert back to the most recent unexplored branch and continue. We include a more detailed description of this algorithm in Appendix B. This approach is inspired by that presented in [16], which similarly enumerates through the path of neurons using sets to determine the output range of a network.

Partitions For each mode in the automaton, the associated partition is defined by the affine conditions that construct the set $\mathcal{X}_C$, as described in equation (8). In other words, for a given mode m corresponding to a configuration C, $\mathcal{P}_m = \mathcal{X}_C$.

Dynamics The dynamics of each mode can be seen itself as a dynamical system with bounded disturbance:

$$\dot{x} = A_C x + b_C + d, \quad \|d\| \leq \epsilon, \quad x \in \mathcal{X}_C. \tag{9}$$

The matrix $A_C \in \mathbb{R}^{n \times n}$ and the vector of drifts $b_C \in \mathbb{R}^n$ determine the linear ODE of the mode, whereas $\epsilon > 0$ is the error bound derived from the neural abstraction. The coefficients of the system are given by the weights and biases of the neural network as follows:

$$A_C = W_{k+1} \prod_{i=1}^{k} \operatorname{diag}(c_i) W_i, \tag{10}$$

$$b_C = b_{k+1} + \sum_{i=1}^{k} ((W_{k+1} \prod_{j=i+1}^{k} \operatorname{diag}(c_j) W_j) \operatorname{diag}(c_i) b_i). \tag{11}$$

Transitions A transition exists between any two given modes m, m' if and only if the two polyhedra that define their partitions share a facet and the dynamics pass through at some point along the facet (this can be checked by considering the inner product between the dynamics and the corresponding normal vector to the facet). In practice, we take a faster but more conservative approach by considering that a transition exists when any two partitions share a vertex.

5 Experimental Results

5.1 Safety Verification using Neural Abstractions

We benchmark the results obtained by the safety verification algorithm proposed in Section 4 against Flow* [23] (available under GPL), which is a mature tool for computing reachable regions of hybrid automata. It relies on computing flowpipes, i.e., sets of reachable states across time, which are propagated from a given set of initial states. The flowpipes are generated from Taylor series approximations of the model's vector field in (1), over discrete time steps. Crucially, the use of a higher-order Taylor series, or of smaller time steps, leads to more precise computation of reachable sets. Since Flow* (like SpaceEx) is able to calculate over-approximations of flowpipes, it is suitable for use in safety verification, and is a state-of-the-art tool for verifying safety of nonlinear models.

Making a fair comparison around metrics for accuracy between Flow* and SpaceEx is challenging, as they represent flowpipes differently [26]. We therefore ask them to perform safety verification for a given set of initial and bad states, on a collection of nonlinear models. These models, and their parameters, are detailed in Appendix A. As described in Section 2, the task of safety verification consists of ensuring that no trajectory starting within the set of initial states enters the set of bad states, within a given time horizon.

Our setup is as follows. Firstly, for a given benchmark model we define a finite time horizon T, a region of initial states $\mathcal{X}_0$ and a region of bad states $\mathcal{X}_B$. Then, we run flowpipe computations with Flow* using high-order Taylor models. Similarly we run the procedure described in Section 3.3, and construct a hybrid automaton as described in section 4 to perform flowpipe computations

Model	T	Flow*				Neural Abstractions			
		TM	δ	Safety Verif.	t	W	M	Safety Verif.	t
Jet Engine	1.5	10	0.1	Yes	**1.3**	[10, 16]	8	Yes	215
Steam Governor	2.0	10	0.1	Yes	**62**	[12]	29	Yes	219
Exponential	1.0	30	0.05	Blw	1034	[14, 14]	12	Yes	**308**
Non-Lipschitz 1	1.4	-	-	No	-	[10]	12	Yes	**19**
Non-Lipschitz 2	1.5	-	-	No	-	[8, 10]	10	Yes	**107**

Table 1: Comparison of safety verification between Flow* and the combination of our Neural Abstractions plus SpaceEx. Here, T: time horizon, *TM*: order of Taylor model, δ: time-step, t: total computation time (better times denoted by **bold**), W: network neural structure, M: total number of modes in resulting hybrid automaton, *Blw*: blowup in the error before T is reached, and -: no results could be obtained.

using SpaceEx (available under GPLv3). We present the results of these in Table 1. In the table, we show the Taylor model (TM) and time-step used within Flow*, as well as the structure of the neural network used for neural abstractions. For example, we denote a network with two hidden layers with h_1 neurons in the first layer and h_2 neurons in the second hidden layer as $[h_1, h_2]$.

Notably, Flow* is unable to handle the two models that do not exhibit local Lipschitz continuity. This makes sense, as it constructs Taylor models that incorporate the derivatives of the dynamics: unbounded derivatives will expectedly cause issues for this approach. Meanwhile, Ariadne [17] a is further tool for over-approximating flowpipes of nonlinear systems. While Ariadne does not explicitly require Lipschitz continuity, it is also unable to perform analysis on tools with nth root terms at zero due to numerical instability. However, our abstraction method works directly on the dynamics themselves, rather than their derivatives, in order to construct a much simpler model, amenable to be verified. Since we quantify how different this simpler model is through the approximation error, we are able to formally perform safety verification on such challenging models.

We also outperform Flow* for a Lipschitz-continuous model (*Exponential* in Table 1), where the composition of functions that make up the model's dynamics result in high errors in Flow* before the flowpipe can be calculated across the given time horizon.

It is worth highlighting that despite relying on affine approximations (i.e., 1st order models), neural abstractions are able to compete with, and even outperform, methods that use much higher order functions (10th and 30th in the benchmarks) for approximation.

5.2 Limitations

Our approach is limited in terms of scalability, both with regards to the dimension of the models and the size of the neural networks. The causes of this limitation are twofold: firstly we are bound by the computational complexity of SMT-solving - known to be NP-hard - which can struggle with complex formaulae with many variables. Secondly, we are limited in terms of the complexity of our abstractions by SpaceEx. While SpaceEx is a highly efficient implementation of LGG [55], the presence of a large number of modes poses a significant challenge. It future work, we hope to investigate the balance between abstraction complexity and accuracy.

6 Conclusion

We have proposed a novel abstraction technique for nonlinear dynamical models using ReLU-based neural networks: *neural abstractions*. These neural structures simplify nonlinear dynamical models by abstracting them to piecewise affine models, which can be interpreted as hybrid automata consisting of modes with affine dynamics. By casting a neural ode as a hybrid automaton and constructing an abstraction by bounding the approximation error, we can conduct safety verification. Empirically we show that for safety verification our abstractions, despite consisting of only affine dynamics, are competitive with state-of-art methods (and a corresponding mature software tool) that depend on higher-order Taylor models of the underlying dynamics, and that they are able to perform safety verification on models that do not exhibit local Lipschitz continuity.

References

[1] Benchmarks of continuous and hybrid systems | Theory of Hybrid Systems. https://ths.rwth-aachen.de/research/projects/hypro/benchmarks-of-continuous-and-hybrid-systems/

[2] Abate, A., Ahmed, D., Edwards, A., Giacobbe, M., Peruffo, A.: Fossil: a software tool for the formal synthesis of lyapunov functions and barrier certificates using neural networks. In: Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control. p. 1–11. HSCC '21, Association for Computing Machinery, New York, NY, USA (May 2021). https://doi.org/10.1145/3447928.3456646, https://doi.org/10.1145/3447928.3456646

[3] Abate, A., David, C., Kesseli, P., Kroening, D., Polgreen, E.: Counterexample guided inductive synthesis modulo theories. In: CAV (1). Lecture Notes in Computer Science, vol. 10981, pp. 270–288. Springer (2018)

[4] Althoff, M.: Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In: HSCC. pp. 173–182. ACM (2013)

[5] Althoff, M., Stursberg, O., Buss, M.: Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In: CDC. pp. 4042–4048. IEEE (2008)

[6] Alur, R., Henzinger, T., Lafferriere, G., Pappas, G.: Discrete abstractions of hybrid systems. Proceedings of the IEEE **88**(7), 971–984 (2000)

[7] Alur, R., Giacobbe, M., Henzinger, T.A., Larsen, K.G., Mikucionis, M.: Continuous-time models for system design and analysis. In: Computing and Software Science, Lecture Notes in Computer Science, vol. 10000, pp. 452–477. Springer (2019)

[8] Asarin, E., Dang, T.: Abstraction by projection and application to multi-affine systems. In: HSCC. Lecture Notes in Computer Science, vol. 2993, pp. 32–47. Springer (2004)

[9] Asarin, E., Dang, T., Girard, A.: Reachability analysis of nonlinear systems using conservative approximation. In: HSCC. Lecture Notes in Computer Science, vol. 2623, pp. 20–35. Springer (2003)

[10] Asarin, E., Dang, T., Girard, A.: Hybridization methods for the analysis of nonlinear systems. Acta Informatica **43**(7), 451–476 (2007)

[11] Åström, K.J., Murray, R.M.: Feedback Systems: An Introduction for Scientists and Engineers. Princeton University Press, Princeton (2008)

[12] Bacci, E., Giacobbe, M., Parker, D.: Verifying reinforcement learning up to infinity. In: IJCAI. pp. 2154–2160. ijcai.org (2021)

[13] Bae, K., Gao, S.: Modular smt-based analysis of nonlinear hybrid systems. In: FMCAD. pp. 180–187. IEEE (2017)

[14] Bak, S., Bogomolov, S., Duggirala, P.S., Gerlach, A.R., Potomkin, K.: Reachability of black-box nonlinear systems after koopman operator linearization. arXiv:2105.00886 [cs, eess] (May 2021), http://arxiv.org/abs/2105.00886, arXiv: 2105.00886

[15] Bak, S., Bogomolov, S., Henzinger, T.A., Johnson, T.T., Prakash, P.: Scalable static hybridization methods for analysis of nonlinear systems. In: HSCC. pp. 155–164. ACM (2016)

[16] Bak, S., Tran, H.D., Hobbs, K., Johnson, T.T.: Improved Geometric Path Enumeration for Verifying ReLU Neural Networks, Lecture Notes in Computer Science, vol. 12224, p. 66–96. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_4, http://link.springer.com/10.1007/978-3-030-53288-8_4

[17] Balluchi, A., Casagrande, A., Collins, P., Ferrari, A., Villa, T., Sangiovanni-Vincentelli, A.L.: Ariadne: a framework for reachability analysis of hybrid automata p. 8

[18] Bogomolov, S., Giacobbe, M., Henzinger, T.A., Kong, H.: Conic abstractions for hybrid systems. In: FORMATS. Lecture Notes in Computer Science, vol. 10419, pp. 116–132. Springer (2017)

[19] Chang, Y., Roohi, N., Gao, S.: Neural lyapunov control. In: NeurIPS. pp. 3240–3249 (2019)

[20] Chen, S., Fazlyab, M., Morari, M., Pappas, G.J., Preciado, V.M.: Learning lyapunov functions for hybrid systems. In: Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control. pp. 1–11. ACM, Nashville Tennessee (May 2021). https://doi.org/10.1145/3447928.3456644

[21] Chen, T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.: Neural ordinary differential equations. In: NeurIPS. pp. 6572–6583 (2018)

[22] Chen, X., Ábrahám, E., Sankaranarayanan, S.: Taylor model flowpipe construction for non-linear hybrid systems. In: RTSS. pp. 183–192. IEEE Computer Society (2012)

[23] Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: CAV. Lecture Notes in Computer Science, vol. 8044, pp. 258–263. Springer (2013)

[24] Chen, X., Mover, S., Sankaranarayanan, S.: Compositional relational abstraction for nonlinear hybrid systems. ACM Trans. Embed. Comput. Syst. **16**(5s), 187:1–187:19 (2017)

[25] Chen, X., Sankaranarayanan, S.: Decomposed reachability analysis for nonlinear systems. In: RTSS. pp. 13–24. IEEE Computer Society (2016)

[26] Chen, X., Sankaranarayanan, S., Abraham, E.: Flow* 1.2: More effective to play with hybrid systems. p. 152–143. https://doi.org/10.29007/1w4t, `https://easychair.org/publications/paper/QrVj`

[27] Cimatti, A., Mover, S., Tonetta, S.: A quantifier-free SMT encoding of non-linear hybrid automata. In: FMCAD. pp. 187–195. IEEE (2012)

[28] Dai, H., Landry, B., Pavone, M., Tedrake, R.: Counter-example guided synthesis of neural network Lyapunov functions for piecewise linear systems. In: 2020 59th IEEE Conference on Decision and Control (CDC). pp. 1274–1281 (Dec 2020). https://doi.org/10.1109/CDC42340.2020.9304201

[29] Dai, H., Landry, B., Yang, L., Pavone, M., Tedrake, R.: Lyapunov-stable neural-network control. In: Robotics: Science and Systems XVII. Robotics: Science and Systems Foundation (Jul 2021)

[30] Dang, T., Frehse, G., Girard, A., Guernic, C.L.: Tools for the analysis of hybrid models. In: Communicating Embedded Systems, pp. 227–251. Wiley (2013)

[31] Dang, T., Maler, O., Testylier, R.: Accurate hybridization of nonlinear systems. In: HSCC. pp. 11–20. ACM (2010)

[32] Dang, T., Testylier, R.: Hybridization domain construction using curvature estimation. In: HSCC. pp. 123–132. ACM (2011)

[33] Dang, T., Testylier, R.: Reachability analysis for polynomial dynamical systems using the bernstein expansion. Reliab. Comput. **17**, 128–152 (2012)

[34] Doyen, L., Frehse, G., Pappas, G.J., Platzer, A.: Verification of hybrid systems. In: Handbook of Model Checking, pp. 1047–1110. Springer (2018)

[35] Dreossi, T.: Sapo: Reachability computation and parameter synthesis of polynomial dynamical systems. In: HSCC. pp. 29–34. ACM (2017)

[36] Dreossi, T., Dang, T., Piazza, C.: Reachability computation for polynomial dynamical systems. Formal Methods Syst. Des. **50**(1), 1–38 (2017)

[37] Dutta, S., Chen, X., Sankaranarayanan, S.: Reachability analysis for neural feedback systems using regressive polynomial rule inference. In: HSCC. pp. 157–168. ACM (2019)

[38] Fan, C., Qi, B., Mitra, S., Viswanathan, M., Duggirala, P.S.: Automatic reachability analysis for nonlinear hybrid models with C2E2. In: CAV (1). Lecture Notes in Computer Science, vol. 9779, pp. 531–538. Springer (2016)

[39] Frehse, G.: Phaver: Algorithmic verification of hybrid systems past hytech. In: HSCC. Lecture Notes in Computer Science, vol. 3414, pp. 258–273. Springer (2005)

[40] Frehse, G., Guernic, C.L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: Spaceex: Scalable verification of hybrid systems. In: CAV. Lecture Notes in Computer Science, vol. 6806, pp. 379–395. Springer (2011)

[41] Gao, S., Kong, S., Clarke, E.M.: dReal: An SMT Solver for Nonlinear Theories over the Reals. In: Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M.Y., Weikum, G., Bonacina, M.P. (eds.) Automated Deduction – CADE-24, vol. 7898, pp. 208–214. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

[42] Gruenbacher, S., Hasani, R.M., Lechner, M., Cyranka, J., Smolka, S.A., Grosu, R.: On the verification of neural odes with stochastic guarantees. In: AAAI. pp. 11525–11535. AAAI Press (2021)

[43] Gruenbacher, S., Lechner, M., Hasani, R., Rus, D., Henzinger, T.A., Smolka, S., Grosu, R.: Gotube: Scalable stochastic verification of continuous-depth models. In: AAAI (2022)

[44] Henzinger, T.A.: The theory of hybrid automata. In: LICS. pp. 278–292. IEEE Computer Society (1996)

[45] Henzinger, T.A., Wong-Toi, H.: Linear phase-portrait approximations for nonlinear hybrid systems. In: Hybrid Systems. Lecture Notes in Computer Science, vol. 1066, pp. 377–388. Springer (1995)

[46] Huang, C., Fan, J., Li, W., Chen, X., Zhu, Q.: Reachnn: Reachability analysis of neural-network controlled systems. ACM Trans. Embed. Comput. Syst. **18**(5s), 106:1–106:22 (2019)

[47] Ivanov, R., Carpenter, T.J., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In: CAV (1). Lecture Notes in Computer Science, vol. 12759, pp. 249–262. Springer (2021)

[48] Johnson, T.T., Lopez, D.M., Benet, L., Forets, M., Guadalupe, S., Schilling, C., Ivanov, R., Carpenter, T.J., Weimer, J., Lee, I.: Arch-comp21 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In: Frehse, G., Althoff, M. (eds.) 8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21). EPiC Series in Computing, vol. 80, pp. 90–119. EasyChair (2021)

[49] Kapinski, J., Deshmukh, J.V., Sankaranarayanan, S., Arechiga, N.: Simulation-guided lyapunov analysis for hybrid dynamical systems. In: Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control. pp. 133–142. HSCC '14, Association for Computing Machinery, New York, NY, USA (Apr 2014)

[50] Kekatos, N., Forets, M., Frehse, G.: Constructing verification models of nonlinear simulink systems via syntactic hybridization. In: CDC. pp. 1788–1795. IEEE (2017)

[51] Khalil, H.K.: Nonlinear Systems. Prentice Hall, Upper Saddle River, N.J, 3rd ed edn. (2002)

[52] Kim, E., Bak, S., Duggirala, P.S.: Automatic dynamic parallelotope bundles for reachability analysis of nonlinear systems. In: FORMATS. Lecture Notes in Computer Science, vol. 12860, pp. 50–66. Springer (2021)

[53] Kong, H., Bartocci, E., Bogomolov, S., Grosu, R., Henzinger, T.A., Jiang, Y., Schilling, C.: Discrete abstraction of multiaffine systems. In: HSB. Lecture Notes in Computer Science, vol. 9957, pp. 128–144 (2016)

[54] Kong, S., Gao, S., Chen, W., Clarke, E.M.: dreach: δ-reachability analysis for hybrid systems. In: TACAS. Lecture Notes in Computer Science, vol. 9035, pp. 200–205. Springer (2015)

[55] Le Guernic, C., Girard, A.: Reachability Analysis of Hybrid Systems Using Support Functions. In: Bouajjani, A., Maler, O. (eds.) Computer Aided Verification, vol. 5643, pp. 540–554. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)

[56] Li, D., Bak, S., Bogomolov, S.: Reachability analysis of nonlinear systems using hybridization and dynamics scaling. In: FORMATS. Lecture Notes in Computer Science, vol. 12288, pp. 265–282. Springer (2020)

[57] Majumdar, R., Zamani, M.: Approximately bisimilar symbolic models for digital control systems. In: CAV. Lecture Notes in Computer Science, vol. 7358, pp. 362–377. Springer (2012)

[58] Mover, S., Cimatti, A., Griggio, A., Irfan, A., Tonetta, S.: Implicit semi-algebraic abstraction for polynomial dynamical systems. In: CAV (1). Lecture Notes in Computer Science, vol. 12759, pp. 529–551. Springer (2021)

[59] Pola, G., Girard, A., Tabuada, P.: Approximately bisimilar symbolic models for nonlinear control systems. arXiv:0706.0246 [math] (Jan 2008)

[60] Prabhakar, P., Dullerud, G.E., Viswanathan, M.: Stability preserving simulations and bisimulations for hybrid systems. IEEE Trans. Autom. Control. **60**(12), 3210–3225 (2015)

[61] Prabhakar, P., Soto, M.G.: Abstraction based model-checking of stability of hybrid systems. In: CAV. Lecture Notes in Computer Science, vol. 8044, pp. 280–295. Springer (2013)

[62] Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. Adaptive Computation and Machine Learning, MIT Press, Cambridge, Mass (2006)

[63] Ravanbakhsh, H., Sankaranarayanan, S.: Counter-Example Guided Synthesis of control Lyapunov functions for switched systems. In: 2015 54th IEEE Conference on Decision and Control (CDC). pp. 4232–4239 (Dec 2015). https://doi.org/10.1109/CDC.2015.7402879

[64] Ravanbakhsh, H., Sankaranarayanan, S.: Learning control lyapunov functions from counterexamples and demonstrations. Autonomous Robots **43**(2), 275–307 (Feb 2019). https://doi.org/10.1007/s10514-018-9791-9

[65] Roohi, N., Prabhakar, P., Viswanathan, M.: Hybridization based CEGAR for hybrid automata with affine dynamics. In: TACAS. Lecture Notes in Computer Science, vol. 9636, pp. 752–769. Springer (2016)

[66] Sankaranarayanan, S.: Automatic abstraction of non-linear systems using change of bases transformations. In: HSCC. pp. 143–152. ACM (2011)

[67] Sankaranarayanan, S., Tiwari, A.: Relational abstractions for continuous and hybrid systems. In: CAV. Lecture Notes in Computer Science, vol. 6806, pp. 686–702. Springer (2011)

[68] Sassi, M.A.B., Testylier, R., Dang, T., Girard, A.: Reachability analysis of polynomial systems using linear programming relaxations. In: ATVA. Lecture Notes in Computer Science, vol. 7561, pp. 137–151. Springer (2012)

[69] Sastry, S.: Nonlinear Systems, Interdisciplinary Applied Mathematics, vol. 10. Springer New York, New York, NY (1999)

[70] van der Schaft, A., Schumacher, H.: An introduction to hybrid dynamical systems, Lecture Notes in Control and Information Sciences, vol. 251. Springer London, London (2000). https://doi.org/10.1007/BFb0109998, `http://link.springer.com/10.1007/BFb0109998`

[71] Schilling, C., Forets, M., Guadalupe, S.: Verification of neural-network control systems by integrating taylor models and zonotopes. In: AAAI (2022)

[72] Solar-Lezama, A., Tancau, L., Bodik, R., Saraswat, V., Seshia, S.: Combinatorial Sketching for Finite Programs p. 12

[73] Soto, M.G., Prabhakar, P.: Hybridization for stability verification of nonlinear switched systems. In: RTSS. pp. 244–256. IEEE (2020)

[74] Tabuada, P.: Verification and Control of Hybrid Systems - A Symbolic Approach. Springer (2009)

[75] Tran, H., Cai, F., Lopez, D.M., Musau, P., Johnson, T.T., Koutsoukos, X.D.: Safety verification of cyber-physical systems with reinforcement learning control. ACM Trans. Embed. Comput. Syst. **18**(5s), 105:1–105:22 (2019)

[76] Tran, H.D., Pal, N., Musau, P., Lopez, D.M., Hamilton, N., Yang, X., Bak, S., Johnson, T.T.: Robustness Verification of Semantic Segmentation Neural Networks Using Relaxed Reachability. In: Silva, A., Leino, K.R.M. (eds.) Computer Aided Verification. pp. 263–286. Lecture Notes in Computer Science, Springer International Publishing, Cham (2021)

[77] Tran, H., Yang, X., Lopez, D.M., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: CAV (1). Lecture Notes in Computer Science, vol. 12224, pp. 3–17. Springer (2020)

[78] Xiang, W., Tran, H., Rosenfeld, J.A., Johnson, T.T.: Reachable set estimation and safety verification for piecewise linear systems with neural network controllers. In: ACC. pp. 1574–1579. IEEE (2018)

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Please see section 5.2

(c) Did you discuss any potential negative societal impacts of your work? [Yes] See 1

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [Yes] See 2

(b) Did you include complete proofs of all theoretical results? [Yes] See 2

3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The code and data will be part of the supplementary material. Reproducing the results will be possible from this but is not the intention of the authors.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] The hyper-parameters for the learning procedure are chosen heuristically, but we include the relevant configuration files in the supplementary material.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Table 1.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We have cited all used tools.

(b) Did you mention the license of the assets? [Yes] See section 5

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] The code will be included in the supplementary material.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

7 Appendix A: Benchmarks of NonLinear Dynamical Systems

For each dynamical model, we report the vector field $f : \mathbb{R}^n \to \mathbb{R}^n$ and the spatial domain $\mathcal{X}$ over which the abstraction is performed and which, unless otherwise stated, is taken to be the hyper-rectangle $[-1, 1]^n$.

Jet Engine [1]

$$\begin{cases} \dot{x} = -y - 1.5x^2 - 0.5x^3 - 0.1, \\ \dot{y} = 3x - y, \\ \mathcal{X}_0 = [0.45, 0.50] \times [-0.60, -0.55] \\ \mathcal{X}_B = [0.3, 0.35] \times [0.5, 0.6] \end{cases} \tag{12}$$

Steam Governor [1]

$$\begin{cases} \dot{x} = y, \\ \dot{y} = z^2 \sin(x)\cos(x) - \sin(x) - 3y, \\ \dot{z} = -(\cos(x) - 1), \\ \mathcal{X}_0 = [0.70, 0.75] \times [-0.05, 0.05] \times [0.70, 0.75] \\ \mathcal{X}_B = [0.5, 0.6] \times [-0.4, -0.3] \times [0.7, 0.8] \end{cases} \tag{13}$$

Exponential

$$\begin{cases} \dot{x} = -\sin(\exp(y^3 + 1)) - y^2 \\ \dot{y} = -x, \\ \mathcal{X}_0 = [0.45, 0.5] \times [0.86, 0.91] \\ \mathcal{X}_B = [0.3, 0.4] \times [0.5, 0.6] \end{cases} \tag{14}$$

Non-Lipschitz Vector Field 1 (NL1)

$$\begin{cases} \dot{x} = y \\ \dot{y} = \sqrt{x} \\ \mathcal{X} = [0, 1] \times [-1, 1], \\ \mathcal{X}_0 = [0, 0.05] \times [0, 0.1] \\ \mathcal{X}_B = [0.35, 0.45] \times [0.1, 0.2] \end{cases} \tag{15}$$

Non-Lipschitz Vector Field 2 (NL2)

$$\begin{cases} \dot{x} = y \\ \dot{y} = \sqrt[3]{x^2} - x, \\ \mathcal{X}_0 = [-0.025, 0.025] \times [-0.9, -0.85] \\ \mathcal{X}_B = [-0.05, 0.05] \times [-0.8, -0.7] \end{cases} \tag{16}$$