Neural Abstractions

Alessandro Abate*
Department of Computer Science
University of Oxford, UK

Alec Edwards*
Department of Computer Science
University of Oxford, UK

Mirco Giacobbe*
School of Computer Science
University of Birmingham, UK

Abstract

We present a novel method for the safety verification of dynamical models, which uses neural networks to represent abstractions of the dynamics. 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. We synthesise a neural network so as to approximate a nonlinear model whilst ensuring an arbitrarily tight, formally certified bound on the approximation error, using counterexample-guided inductive synthesis. We show that this produces a neural ODE with non-deterministic disturbances that constitutes a formal abstraction of the concrete system under analysis. This guarantees a fundamental property: if the abstract system is safe, i.e., free from any initialised trajectory that reaches an undesirable state, then the concrete system is also safe. By using neural ODEs with ReLU activation functions as abstractions, we cast the safety verification problem for nonlinear dynamical models into that of hybrid automata with affine dynamics, which we verify using SpaceEx. We demonstrate that our approach performs comparably to the mature tool Flow* on existing benchmark nonlinear models. We additionally demonstrate and that it is effective on models that do not exhibit local Lipschitz continuity, which are out of reach to the existing technologies.

1 Introduction

Dynamical models describe processes that are ubiquitous in science and engineering. They are widely used to model the behaviour of cyber-physical system designs, whose correctness is crucial when they are deployed in safety-critical domains [10, 12, 46]. To guarantee that a dynamical model satisfies a safety specification, simulations are useful but insufficient because they are inherently non-exhaustive and they suffer from numerical errors, which may leave unsafe behaviours unidentified. Formal verification of continuous dynamical models tackles the question of determining with formal certainty whether every possible behavior of the model satisfies a safety specification [42, 48, 109]. In this paper, we present a method to combine machine learning and symbolic reasoning for a sound and effective safety verification of nonlinear dynamical models.

The formal verification problem for continuous-time and hybrid dynamical models is unsolvable in general and, even for models with linear dynamics, complete procedures are available under stringent conditions [11, 69, 83, 84]. For most practical models that contain nonlinear terms [78, 102], methods for formal verification with soundness guarantees involve laborious safety and reachability procedures

*The authors are listed alphabetically

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

whose efficacy can only be demonstrated in practice. Formal verification of nonlinear models require ingenuity, and has involved sophisticated analysis techniques such as mathematical relaxations [25, 32–34, 45, 100, 101], abstract interpretation [49, 50, 53, 79, 93], constraint solving [18, 37, 82], and discrete abstractions [5, 9, 28, 35]. Notwithstanding recent progress, both scalability and expressivity remain open challenges for nonlinear models: the largest model used in the annual competition has 7 variables [63]. In addition, existing formal approaches rely on symbolic reasoning techniques that explicitly leverage the structure of the dynamics. This results in verification procedures that are bespoke to restricted classes of models. For example, it is common for formal verification procedures to require the input model to be Lipschitz continuous. Yet, dynamical models with vector fields that violate this assumption are abundant in literature, and a wide variety of models of natural phenomena are non-Lipschitz, from fluid dynamics to n-body orbits and chaotic systems, as well as in engineering, from electrical circuits and hydrological systems [47, 52]. Our approach makes progress in expressivity, showing that using *neural networks as abstractions* of dynamical systems enables an effective formal verification of nonlinear dynamical models, including models that do not exhibit local Lipschitz continuity.

Abstraction is a standard process in formal verification that aims at translating the model under analysis—the concrete model—into a model that is simpler to analyse—the abstract model—such that verification results from the abstract model carry over to the concrete model [19, 38]. In verification of continuous-time systems, an abstraction usually consists of a partitioning of the state space of the concrete model into a finite set of regions that define the states of an abstract, finite-state machine with corresponding dynamics. Our method follows an approach that constructs abstract, finite-state machines whose states are augmented with continuous linear dynamics and non-deterministic drifts. Finite-state machines with continuous, possibly non-deterministic dynamics are known as hybrid automata [68], and the process of abstracting dynamical nonlinear models into hybrid automata is called *hybridisation*; this process has been widely applied in formal verification [8, 14, 15, 20, 43, 55, 62, 70, 88, 91, 96, 97, 99].

Hybridising involves partitioning the state space and computing a local *overapproximation* of the concrete model within each region of the partition. Common approaches for hybridisation partition the state space by tuning the granularity of rectangular or simplicial meshes, until a desired approximation error is attained. This may yield abstract hybrid automata that are too large in the number of discrete states to be effectively verified. Notably, modern tools for the verification of hybrid automata are designed for models that rarely have over hundred discrete states [7], while arbitrary meshes grow exponentially as the granularity increases. Explosion in discrete states has been mitigated using deductive approaches that construct an appropriate partitioning from the expressions that define the concrete model and, unlike our method, rely on syntactic restrictions [13, 24, 28, 44, 67, 77, 80, 81, 95].

We propose an *inductive approach to abstraction* that combines the tasks of partitioning the state space and overapproximating the dynamics into the single task of training a neural network. We leverage the approximation capability of neural networks with ReLU activation functions to partition the state space into arbitrary polyhedral regions, where each region and local affine approximation correspond to a combinatorial configuration of the neurons. We show that this ultimately enables verifying nonlinear dynamical models using efficient safety verifiers for hybrid automata with affine dynamics (cf. Figure 1).

Our abstraction procedure synthesises abstract models by alternating a *learner*, which proposes candidate abstractions, and a *certifier*, which formally assures (or disproves) their validity, in a counterexample-guided inductive synthesis (CEGIS) loop [105, 106]. First, the learner uses gradient descent to train a neural network that approximates the concrete model over a finite set of sample observations of its dynamics; then, the certifier uses satisfiability modulo theories (SMT) to check the validity of an upper-bound on the approximation error over the entire continuous domain of interest. If the latter disproves the bound, then it produces a counterexample which its added to the set of samples and the loop is repeated. If it certifies the bound, then the procedure returns a neural network approximation and a sound upper-bound on the error. Altogether, neural network and error bound define a neural ODE with bounded additive non-determinism that overapproximates the concrete model, which we call a *neural abstraction*.

We demonstrate the efficacy of our method over multiple dynamical models from a standard benchmark set for the verification of nonlinear systems [63], as well as additional locally non-Lipschitz models, and compare our approach with Flow*, the state-of-the-art verification tool for nonlinear

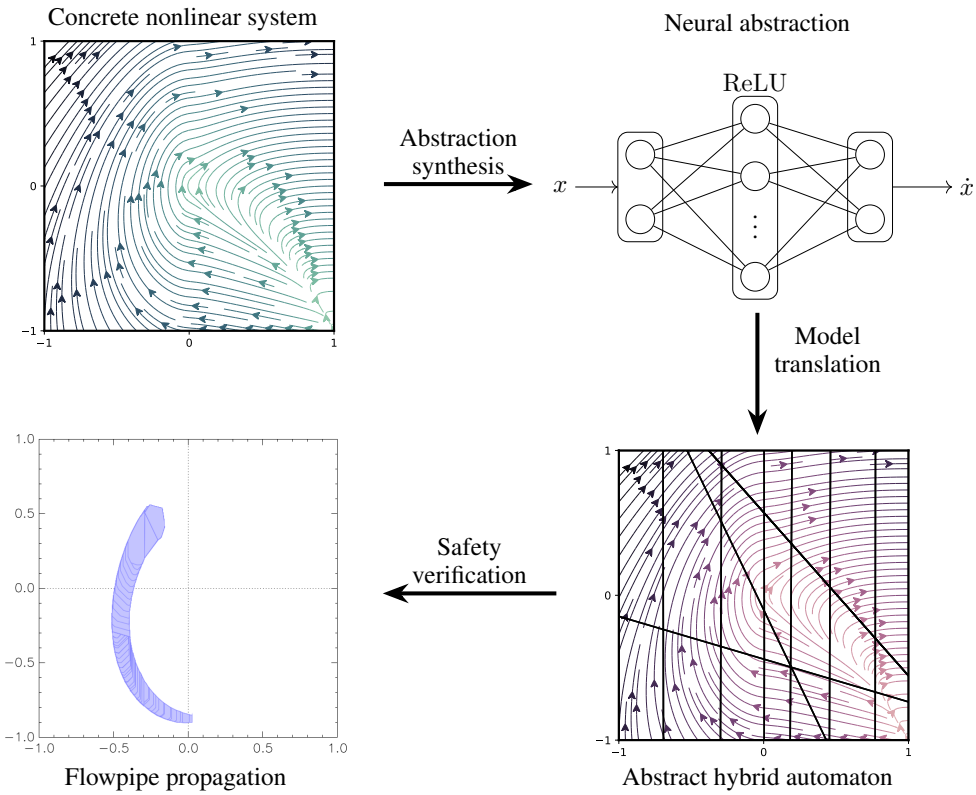

Figure 1: Overview of our workflow on a non-Lipschitz dynamical model (cf. Section 5, NL2). The concrete dynamics are abstracted by a neural ODE with ReLU activation functions and a certified upper-bound on the approximation error. This characterises a polyhedral partitioning and defines a hybrid automaton with affine dynamics and additive non-deterministic drift. Flowpipe propagation is finally performed through a region of non-Lipschitz continuity.

models [32, 33, 35]. We instantiate our approach on top of SpaceEx [59], which is a state-of-the-art tool specialised to linear hybrid models [56, 58, 85]. We evaluate both approaches in safety verification using *flowpipe propagation*, which computes the set of reachable states from a given set of initial states up to a given time horizon. Our experiments demonstrate that our approach performs comparably with Flow* for Lipschitz continuous model, and succeeds with non-Lipschitz models that are out of range for Flow* and violate the working assumptions of many verification tools. These outcomes suggest that neural abstractions are a promising technology, also in view of recent results on direct methods for the safety verification for neural ODEs [65, 66, 92].

We summarise our contributions in the following points:

- we introduce the novel idea of leveraging neural networks to represent abstractions in formal verification, and we instantiate it in safety verification of nonlinear dynamical models;
- we present a CEGIS procedure for the synthesis of neural ODEs that formally overapproximate the dynamics of nonlinear models, which we call neural abstractions;
- we define a translation from neural abstractions defined using ReLU activation functions to hybrid automata with affine dynamics and additive non-determinism;
- we implement our approach[2] and demonstrate its comparable performance w.r.t. the state-of-the-art tool Flow* in safety verification of Lipschitz-continuous models, and even superior efficacy on models that do not exhibit local Lipschitz continuity.

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

[2]The code is available at https://github.com/aleccedwards/neural-abstractions-nips22.

2 Neural Abstractions of Dynamical Models

We study the formal verification question of whether an n-dimensional, continuous-time, autonomous dynamical model with possibly uncertain (bounded) disturbances, considered within a region of interest, is safe with respect to a region of bad states when initialised from a region of initial states.

Definition 1 (Dynamical Model)**.** *A dynamical model $\mathcal{F}$ defined over a region of interest $\mathcal{X} \subseteq \mathbb{R}^n$, consists of a nonlinear function $f\colon \mathbb{R}^n \to \mathbb{R}^n$ and a possibly null disturbance radius $\delta \geq 0$. 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 the norm operator to be given the same semantics across the paper). A trajectory of $\mathcal{F}$ defined over time horizon $T > 0$ is a function $\xi\colon [0,T] \to \mathbb{R}^n$ that admits derivative at each point in $[0,T]$ such that, for all $t \in [0,T]$, it holds true that $\xi(t) \in \mathcal{X}$ and $\dot{\xi}(t) = f(\xi(t)) + d_t$ for some $\|d_t\| < \delta$. Notably, symbol d in Equation (1) *is interpreted as a non-deterministic disturbance that at any time can take any possible value within the bound provided by δ.*

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 a trajectory ξ defined over time horizon T is initialised if $\xi(0) \in \mathcal{X}_0$; additionally, we say that it is safe if $\xi(t) \notin \mathcal{X}_B$ for all $t \in [0,T]$; dually, we say that it is unsafe if $\xi(t) \in \mathcal{X}_B$ for some $t \in [0,T]$. The safety verification question for consists of determining whether all initialised trajectories are safe. If this is the case, then we say that the model is safe with respect to $\mathcal{X}_0$ and $\mathcal{X}_B$. If there exist at least one initialised trajectory that is unsafe, then we say that the model is unsafe.

We tackle safety verification by abstraction, that is, we construct an abstract dynamical model that captures all behaviours of the concrete nonlinear model. This implies that, if the abstract model is safe then the concrete model is necessarily safe too, and we can thus apply a verification procedure over the abstraction to determine whether the concrete model is safe. Notably, the converse may not hold: lack of safety of the abstract model does not carry over to the concrete model, because our abstraction is an overapproximation. We ultimately obtain a sound (but not complete) safety verification procedure. Our approach synthesises an abstract dynamical model defined in terms a feed-forward neural network with ReLU activation functions and endowed with a bounded non-deterministic disturbance. This can be seen as a neural ODEs [31] augmented with an additive non-deterministic drift that ensures the abstract model to overapproximate the concrete model. To the best of our knowledge, this is the first work to consider neural ODEs with non-deterministic semantics.

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. 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 2 (Neural Abstraction)**.** *Let $\mathcal{F}$ be a dynamical model given by function $f\colon \mathbb{R}^n \to \mathbb{R}^n$ and disturbance radius $\delta \geq 0$ and let $\mathcal{X} \subseteq \mathbb{R}^n$ be a region of interest. A feed-forward neural network $\mathcal{N}\colon \mathbb{R}^n \to \mathbb{R}^n$ defines a neural abstraction of $\mathcal{F}$ with error bound $\epsilon > 0$ over $\mathcal{X}$, if it holds true that*

$$\forall x \in \mathcal{X}\colon \|f(x) - \mathcal{N}(x)\| \leq \epsilon - \delta. \tag{3}$$

Then, the neural abstraction consists of the dynamical model $\mathcal{A}$ defined by $\mathcal{N}$ and disturbance ϵ, whose dynamics are given by the following neural ODE with bounded additive disturbances:

$$\dot{x} = \mathcal{N}(x) + d, \quad \|d\| \leq \epsilon, \quad x \in \mathcal{X}. \tag{4}$$

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

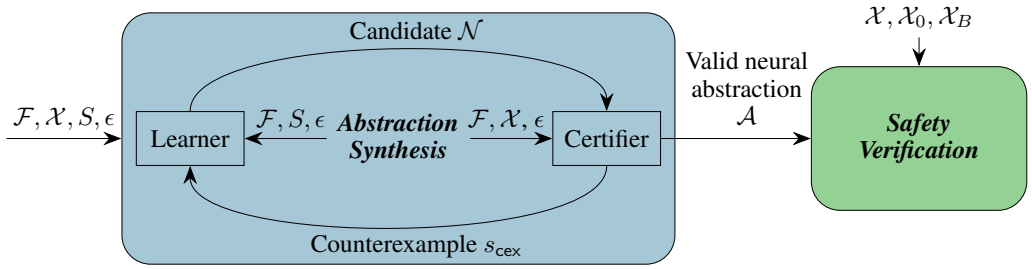

Figure 2: Architecture for the safety verification of nonlinear dynamical models using neural abstractions. The inputs to our architecture are a concrete model $\mathcal{F}$ and its domain of interest $\mathcal{X}$, a finite set of initial datapoints S, a desired approximation error ϵ, and regions of initial $\mathcal{X}_0$ and bad states $\mathcal{X}_B$.

Proof of Theorem 1. Let ξ be a trajectory of $\mathcal{F}$ and T be the time horizon over which ξ is defined. Then, let $t \in [0, T]$. By definition of trajectory we have that (i) $\xi(t) \in \mathcal{X}$ and there exists d_t s.t. (ii) $\|d_t\| \leq \delta$ and (iii) $\dot{\xi}(t) = f(\xi(t)) + d_t$. By (i) and condition (3) we have that $\|f(\xi(t)) - \mathcal{N}(\xi(t))\| + \delta \leq \epsilon$. Then, by (ii) we have that $\|f(\xi(t)) - \mathcal{N}(\xi(t))\| + \|d_t\| \leq \epsilon$ which, by triangle inequality, implies that $\|f(\xi(t)) + d_t - \mathcal{N}(\xi(t))\| \leq \epsilon$. Using (iii), we rewrite it into $\|\dot{\xi}(t) - \mathcal{N}(\xi(t))\| \leq \epsilon$. Finally, we define $d'_t = \dot{\xi}(t) - \mathcal{N}(\xi(t))$. As a result, we have that $\|d'_t\| \leq \epsilon$ and $\dot{\xi}(t) = \mathcal{N}(\xi(t)) + d'_t$ which, together with (i), show that ξ is a trajectory of $\mathcal{A}$. □

Corollary 1. *Let $\mathcal{X}_0 \subset \mathcal{X}$ be a region of initial states and $\mathcal{X}_B \subset \mathcal{X}$ and region of bad states. It holds true that if $\mathcal{A}$ is safe with respect to $\mathcal{X}_0$ and $\mathcal{X}_B$ then also $\mathcal{F}$ is safe with respect to $\mathcal{X}_0$ and $\mathcal{X}_B$.*

Proof of Corollary 1. By Theorem 1, if there exists an initialised trajectory of $\mathcal{F}$ that is unsafe, then the same is an initialised trajectory of $\mathcal{A}$ that is unsafe. The statement follows by contraposition. □

Remark 1 (Existence of Neural Abstractions)**.** *Let $\mathcal{F}$ be a dynamical model defined by function f and disturbance radius $\delta \geq 0$, and let $\mathcal{X} \subseteq \mathbb{R}^n$ be a domain of interest. A neural abstraction of $\mathcal{F}$ with arbitrary error bound $\epsilon > 0$ over $\mathcal{X}$ exists if a neural network that approximates f with error bound $\epsilon - \delta$ (cf. condition* (3)*) exists over the same domain. In this work, we do not prescribe conditions on either width or depth of the network to ensure existence of a neural abstraction. Such conditions are given by universal approximation theorems for neural networks with ReLU activation functions, which have been developed in seminal work in machine learning [23, 39, 60, 71, 87, 90].*

Altogether, we define the neural abstraction of a non-linear dynamical system $\mathcal{F}$ as a neural ODE with an 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, Theorem 1 does not require f to be Lipschitz continuous: the soundness of a neural abstraction relies on condition (3), whose certification we offload to an SMT solver (cf. Section 3.2). The resulting neural abstraction is to a hybrid automaton with affine dynamics and non-deterministic disturbance (cf. Section 4), which does not rely on the Picard-Lindelof theorem to ensure uniqueness or existence of a solutions.

3 Formal Synthesis of Neural Abstractions

Our approach to abstraction synthesis follows two phases—a *learning* phase and a *certification* phase—that alternate each other in a CEGIS loop [1, 3, 40, 75, 98, 105, 106] (cf. Figure 2, left). Our learning phase trains the parameters of a neural network $\mathcal{N}$ to approximate the system dynamics over a finite set of samples $S \subset \mathcal{X}$ of the domain of interest. Learning uses gradient descent algorithms, which can possibly scale to large amounts of samples. Then, our certification phase either confirms the validity of condition (3) or produces a counterexample which we use to sample additional states and repeat the loop. Certification is based on SMT solving, which reasons symbolically over the continuous domain $\mathcal{X}$ and assures soundness. As a consequence, when certification confirms condition (3) formally valid, then as per Theorem 1 our neural abstraction $\mathcal{A}$ is a sound overapproximation of the concrete model $\mathcal{F}$ and is thus passed to safety verification (cf. Figure 2, right).

Neural networks have been used in the past as representations of formal certificates for the correctness of systems such as Lyapunov neural networks, neural barrier certificates, neural ranking functions

and supermartingales [1, 2, 4, 29, 30, 41, 64, 86, 94, 108, 114, 115]. In the present work, we use neural networks for the first time as abstractions, and we instantiate this idea in safety verification of nonlinear models. We shall now present the components of our abstraction synthesis procedure: learner (cf. Section 3.1) and certifier (cf. Section 3.2).

3.1 Learning Phase

As with many 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,

$$\mathcal{L} = \sum_{s \in S} \|f(s) - \mathcal{N}(s)\|_2, \tag{5}$$

where $\|\cdot\|_2$ represents the $2-\texttt{norm}$ of its input, and $S \subset \mathcal{X}$ is a finite set of data points that are sampled from the domain of interest. 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 S.

The main inputs to the learning procedure are the vector field f of the concrete dynamical model, an initial set of points S sampled uniformly from the domain of interest $\mathcal{X}$. Additional parameters include the hyper-parameters for the learning scheme such as the learning rate, and a 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. If a target error smaller than $\epsilon - \delta$ is provided, this is when all points in the data set S satisfy the specification (3) and certification subsequently check that this generalises over the entire $\mathcal{X}$. If an alternative loss-based stopping criterion is provided, then an error bound on the approximation is estimated using the maximum approximation error over the data set S for use in certification. This estimated bound is conservative, i.e., greater than the maximum, to allow for successful certification to be more likely.

After learning, the network $\mathcal{N}$ is translated to symbolic form and passed to the certification block, which checks condition (3) as described in Section 3.2. The certifier either determines condition (3) valid, and thus the CEGIS loop terminates, or computes a counterexample that falsifies the condition. The counterexample is returned to the learning procedure and augmented by sampling for additional points nearby in order to maximise the efficiency of learning and the overall synthesis.

3.2 Certification Phase

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 the upper bound $\epsilon - \delta$, as per the specification in condition (3). Therefore, the certifier is provided with the negation of the specification, namely

$$\exists x \colon \underbrace{x \in \mathcal{X} \wedge \|f(x) - \mathcal{N}(x)\| > \epsilon - \delta}_{\phi}. \tag{6}$$

The certifier seeks an assignment s_{cex} of the variable x such that the quantifier-free formula ϕ is *satisfiable*, namely that the specified bound 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 s_{cex} forms the counterexample that is provided back to the learner (the machine learning procedure from Section 3.1). Alternatively, if no assignment is found then specification (3) is proven valid; network $\mathcal{N}$ and error bound ϵ are then passed to the safety verification procedure (cf. Section 4).

Certification of the accuracy of the neural abstractions is performed by an SMT solver. Several options exist for the selection of the SMT solver, with the requirement that the solver should reason over quantifier-free nonlinear real arithmetic formulae [54, 61]. This is because the vector field f may contain nonlinear terms. In our experiments, we employ dReal [61], which supports polynomial and non-polynomial terms such as transcendental functions like trigonometric or exponential ones.

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

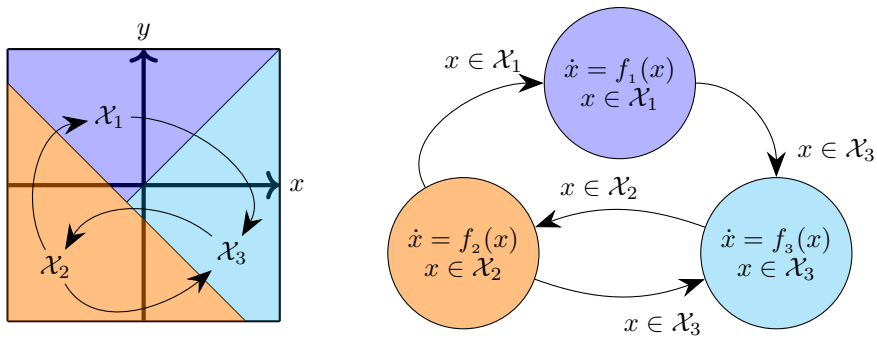

Figure 3: A hybrid automaton corresponding to a state-space partitioning. Each of the three discrete modes corresponds to a unique partition $\mathcal{X}_i$ and vector field $f_i(x)$. Discrete transitions are denoted by the edges of the directed graph with a transition between two modes if the corresponding partitions $\mathcal{X}_i$ and $\mathcal{X}_j$ are adjacent and a trajectory from f_i 'crosses' the corresponding partition.

4 Safety Verification of Neural Abstractions

Neural abstractions are dynamical models expressed in terms of neural ODEs with additive disturbances (cf. Equation 4). Corollary 1 ensures the fact for which concluding that a neural abstraction is safe suffices to assert that the concrete dynamical model is also safe. Consequently, once a neural ODE is formally proven to be an abstraction for the concrete dynamical model, which is entirely delegated to our synthesis procedure (cf. Section 3), our definition of neural abstractions enables any procedure for the safety verification of neural ODEs with disturbances to be a valid safety verification procedure for the corresponding dynamical model.

Safety verification approaches for dynamical systems controlled by neural networks solve a similar problem [17,51,72,74,103,110,111,113], yet with a subtle difference: neural network controllers take control actions at discrete points in time. Instead, neural ODEs characterise dynamics over continuous time. Some procedures for the direct verification of neural ODEs have been introduced very recently, and this currently an area under active development [65, 66, 92]. Yet, existing approaches do not consider the case of a neural ODE with a non-deterministic drift. Therefore, in order to obtain a verification procedure for neural abstractions, we build upon the observation that a neural ODEs with ReLU activation functions and non-deterministic drift defines a hybrid automaton with affine dynamics.

Hybrid automata (cf. Figure 3) model the interaction between continuous dynamical systems and finite-state transition systems [68, 112]. A hybrid automaton consists of a finite set of variables and a finite graph, whose vertices we call discrete modes and edges we call discrete transitions. Every mode is associated with an invariant condition and a flow condition over the variables, which determine the continuous dynamics of the systems on the specific mode. Every discrete transition is associated with a guard condition, which determines the effect on discrete transitions between modes. While we refer the reader to seminal work for a general definition of hybrid automata [68], we present a translation from neural abstractions to hybrid automata.

4.1 Translation of Neural Abstractions Into Hybrid Automata

We begin with the observation that each neuron within a given hidden layer of a neural network with ReLU activation functions induces a hyperplane in the vector space associated with the previous layer This hyperplane results in two half-spaces, one corresponding to the neuron being active and one to it being inactive. For the jth neuron in the ith layer, these two halfspaces are respectively the two parts of the hyperplane given by

$$\{y_{i-1} \mid W_{i,j} y_{i-1} + b_{i,j} = 0\}, \tag{7}$$

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 (cf. Section 2). Therefore, every combinatorial configuration of the neural network defines an intersection of halfspaces that defines a polyhedral region in the vector space of the input neurons. Moreover, every

configuration also defines a linear function from input to output neurons. The space of configurations thus defines a partitioning of the input space, where each region is associated with an affine function. A neural abstraction casts into a hybrid automaton, where every mode is determined by a configuration of the hidden neurons and each of these configurations induces a system of affine ODEs (cf. Figure 3).

Discrete Modes We represent a configuration of a neural network as a sequence $C = (c_1, \ldots, c_k)$ of Boolean vectors $c_1 \in \{0,1\}^{h_1}, \ldots, c_k \in \{0,1\}^{h_k}$, where k denotes the number of hidden layers and $h_1, \ldots, h_k$ denote the number neurons in each of them (cf. Section 2). Every vector c_i represents the configuration of the neurons at the ith hidden later, and the jth element of c_i represent the activation status of the jth neuron at the ith later, which equals to 1 is the neuron is active and 0 if it is inactive. Every mode of the hybrid automaton corresponds to exactly one configuration of neurons.

Invariant Conditions We define the invariant of each mode as a restriction of the domain of interest to a region $\mathcal{X}_C \subseteq \mathcal{X}$, which denotes the maximal set of states that enables configuration C. To construct $\mathcal{X}_C$, we define a higher-dimensional polyhedron on the space of valuation of the neurons that enable configuration C, i.e.,

$$\mathcal{Y}_C = \left\{ (y_0, \ldots, y_k) \;\middle|\; \begin{array}{c} \wedge_{i=1}^{k} y_i = \operatorname{diag}(c_i)(W_i y_{i-1} + b_i) \wedge \\ \operatorname{diag}(2c_i - 1)(W_i y_{i-1} + b_i) \geq 0 \end{array} \right\}. \tag{8}$$

Note that $\operatorname{diag}(v)$ denotes the square diagonal matrix whose diagonal takes its coefficients from vector v; in our case, this results in a square diagonal matrix whose coefficients are either 0 or 1. Then, we project $\mathcal{Y}_C$ onto the input neurons y_0, denoted $\mathcal{Y}_C \restriction_{y_0}$. Since the input neurons y_0 are equivalent to the state variables of the dynamical model, the invariant condition of mode C results in

$$\mathcal{X}_C = (\mathcal{Y}_C \restriction_{y_0}) \cap \mathcal{X}. \tag{9}$$

A projection can be computed using the Fourier-Motzkin algorithm or by projecting the vertices of the polyhedron in a double description method. However, even though this is effective in our experiments, it has worst-case exponential time complexity. A polynomial time construction can be obtained by propagating halfspaces backwards along the network, similarly to methods used in abstraction-refinement [27, 57]. We outline the alternative construction in Appendix C.1.

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

$$\dot{x} = A_C x + b_C + d, \quad \|d\| \leq \epsilon, \quad x \in \mathcal{X}_C. \tag{10}$$

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:

$$A_C = W_{k+1} \textstyle\prod_{i=1}^{k} \operatorname{diag}(c_i) W_i, \tag{11}$$

$$b_C = b_{k+1} + \textstyle\sum_{i=1}^{k} (W_{k+1} \prod_{j=i+1}^{k} \operatorname{diag}(c_j) W_j) \operatorname{diag}(c_i) b_i. \tag{12}$$

Discrete Transitions and Guard Conditions A discrete transition exists between any two given modes if the two polyhedra that define their invariant conditions share a facet and the dynamics pass through at some point along the facet. This can be checked by considering the sign of the Lie derivative between the dynamics and the corresponding facet, that is, the inner product between the dynamics and the normal vector to the facet. In practice, we take a faster but more conservative approach by considering that a transition exists between two modes when the corresponding polyhedral regions share at least a vertex. The guard condition of a discrete transition is simply the invariant of the destination mode.

4.2 Enumeration of Feasible Modes

A given configuration C exists in the hybrid automaton if and only if the corresponding set $\mathcal{X}_C$, which is 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 $H = h_1 + \cdots + h_k$ is the total number of hidden 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.

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. 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 perform a depth-first tree search 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 C.2. This approach is inspired by that presented in [21], which similarly enumerates through the path of neurons using sets to determine the output range of a network.

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* [33] (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 subsequent 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 (available under GPLv3) 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 [36]. We ask them to perform safety verification for a given pair of initial and bad states, on a collection of different 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, and construct a hybrid automaton as described in Section 4 to perform flowpipe computations using SpaceEx. We present the results in Table 1. In the table, we show the Taylor model order (TM) and time step used within Flow*, as well as the structure of the neural networks 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 $[h_1, h_2]$. We note that while Flow*, much like SpaceEx, can perform flowpipe computation on the constructed hybrid automaton, it is not specialised to linear models like SpaceEx is and in practice struggles with the number of modes.

Notably, Flow* is unable to handle the two models that do not exhibit local Lipschitz continuity. Flow* constructs Taylor models that incorporate the derivatives of the dynamics: as expected, unbounded derivatives will cause issues for this approach. Meanwhile, Ariadne [22] a is an alternative 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. Instead, our abstraction method works directly on the dynamics themselves, rather than their derivatives, in order to construct simpler, abstract models that are amenable to be verified. By formally quantifying how different an abstract model is through the approximation error, we are able to formally perform safety verification on such challenging concrete models.

Table 1: Comparison of safety verification between Flow* and the combination of Neural Abstractions plus SpaceEx. Here, T: time horizon, *TM*: Taylor model order, 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 unobtainable.

Model	T	**Flow***				**Neural Abstractions**			
		TM	δ	Safety Ver.	t	W	M	Safety Ver.	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**
Water Tank	2.0	-	-	No	-	[12]	6	Yes	**49**
Non-Lipschitz 1	1.4	-	-	No	-	[10]	12	Yes	**19**
Non-Lipschitz 2	1.5	-	-	No	-	[12, 10]	32	Yes	**59**

Notice that we additionally outperform Flow* on 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. We highlight 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 to the size of the utilised 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. The certification step requires the largest amount of time (cf. Appendix B), indicating that improvements in the verification of neural networks can lead to a large performance increase for our abstractions.

Secondly, we are limited in terms of the complexity of our abstractions by SpaceEx. While SpaceEx is a highly efficient implementation of LGG [85], the presence of a large number of discrete modes poses a significant computational challenge. It future work, we hope to investigate the balance between abstraction complexity and accuracy. The efficacy of neural abstraction on further tools for hybrid automata with affine dynamics also remains to be investigated [6, 22, 26, 104].

6 Conclusion

We have proposed a novel technique that leverages the approximation capabilities of neural networks with ReLU activation functions to synthesise formal abstractions of dynamical models. By combining machine learning and SMT solving algorthms in a CEGIS loop, our method computes abstract neural ODEs with non-determinism that overapproximate the concrete models and therefore satisfy the property for which safety of the abstract model carries over to the concrete model. Then, we cast these neural ODEs into hybrid automata with affine dynamics. We have used SpaceEx to verify the abstraction, which has shown that our method is not only comparable to Flow* in safety verification on existing nonlinear benchmarks, but also shows superior effectiveness on models that do not exhibit local Lipschitz continuity, which is a hard problem in formal verification. Yet, our experiments are limited to low-dimensional models and scalability remains an open challenge. Our approach has advanced the state of the art in terms of expressivity, which is the first step toward obtaining a general and efficient verifier based on neural abstraction. Obtaining scalability to higher dimensions will require a synergy of efficient SMT solvers for neural networks and safety verification of neural ODEs, which are both novel and actively researched questions in formal verification [65,66,73,76,89,92,111].

Acknowledgements

We thank the anonymous reviewers for their helpful suggestions. Alec was supported by the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems (EP/S024050/1).

References

[1] 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: HSCC. pp. 24:1–24:11. ACM (2021)

[2] Abate, A., Ahmed, D., Giacobbe, M., Peruffo, A.: Formal synthesis of Lyapunov neural networks. IEEE Control. Syst. Lett. **5**(3), 773–778 (2021)

[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] Abate, A., Giacobbe, M., Roy, D.: Learning probabilistic termination proofs. In: CAV (2). Lecture Notes in Computer Science, vol. 12760, pp. 3–26. Springer (2021)

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

[6] Althoff, M.: An introduction to CORA 2015. In: ARCH@CPSWeek. EPiC Series in Computing, vol. 34, pp. 120–151. EasyChair (2015)

[7] Althoff, M., Ábrahám, E., Forets, M., Frehse, G., Freire, D., Schilling, C., Schupp, S., Wetzlinger, M.: ARCH-COMP21 category report: Continuous and hybrid systems with linear continuous dynamics. In: ARCH@ADHS. EPiC Series in Computing, vol. 80, pp. 1–31. EasyChair (2021)

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

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

[10] Alur, R.: Principles of cyber-physical systems. MIT press (2015)

[11] Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)

[12] 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)

[13] 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)

[14] 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)

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

[16] Aylward, E.M., Parrilo, P.A., Slotine, J.J.E.: Stability and robustness analysis of nonlinear systems via contraction metrics and SOS programming. Autom. **44**, 2163–2170 (2008)

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

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

[19] Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008)

[20] 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)

[21] Bak, S., Tran, H., Hobbs, K., Johnson, T.T.: Improved geometric path enumeration for verifying relu neural networks. In: CAV (1). Lecture Notes in Computer Science, vol. 12224, pp. 66–96. Springer (2020)

[22] Balluchi, A., Casagrande, A., Collins, P., Ferrari, A., Villa, T., Sangiovanni-Vincentelli, A.: Ariadne: A framework for reachability analysis of hybrid automata. In: MTNS. pp. 1267–1269 (2006)

[23] Barron, A.R.: Approximation and estimation bounds for artificial neural networks. Machine Learning **14**(1), 115–133 (1994)

[24] Batt, G., Belta, C., Weiss, R.: Temporal logic analysis of gene networks under parameter uncertainty. IEEE Trans. Autom. Control. **53**, 215–229 (2008)

[25] Ben Sassi, M.A., 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)

[26] Bogomolov, S., Forets, M., Frehse, G., Potomkin, K., Schilling, C.: Juliareach: a toolbox for set-based reachability. In: HSCC. pp. 39–44. ACM (2019)

[27] Bogomolov, S., Frehse, G., Giacobbe, M., Henzinger, T.A.: Counterexample-guided refinement of template polyhedra. In: TACAS (1). Lecture Notes in Computer Science, vol. 10205, pp. 589–606 (2017)

[28] 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)

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

[30] Chen, S., Fazlyab, M., Morari, M., Pappas, G.J., Preciado, V.M.: Learning Lyapunov functions for hybrid systems. In: HSCC. pp. 13:1–13:11. ACM (2021)

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

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

[33] 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)

[34] 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)

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

[36] Chen, X., Sankaranarayanan, S., Ábrahám, E.: Flow* 1.2: More effective to play with hybrid systems. In: ARCH@CPSWeek. EPiC Series in Computing, vol. 34, pp. 152–159. EasyChair (2015)

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

[38] Clarke, E.M., Grumberg, O., Kroening, D., Peled, D.A., Veith, H.: Model checking, 2nd Edition. MIT Press (2018)

[39] Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems **2**(4), 303–314 (1989)

[40] Dai, H., Landry, B., Pavone, M., Tedrake, R.: Counter-example guided synthesis of neural network Lyapunov functions for piecewise linear systems. In: CDC. pp. 1274–1281. IEEE (2020)

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

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

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

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

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

[46] Derler, P., Lee, E.A., Sangiovanni-Vincentelli, A.L.: Modeling cyber-physical systems. Proc. IEEE **100**(1), 13–28 (2012)

[47] Diacu, F.: The solution of the n-body problem. Tech. rep. (1995)

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

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

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

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

[52] Eyink, G.L., Drivas, T.D.: Spontaneous stochasticity and anomalous dissipation for burgers equation. Journal of Statistical Physics **158**(2), 386–432 (2015)

[53] 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)

[54] Fränzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. J. Satisf. Boolean Model. Comput. **1**(3-4), 209–236 (2007)

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

[56] Frehse, G.: Reachability of hybrid systems in space-time. In: EMSOFT. pp. 41–50. IEEE (2015)

[57] Frehse, G., Giacobbe, M., Henzinger, T.A.: Space-time interpolants. In: CAV (1). Lecture Notes in Computer Science, vol. 10981, pp. 468–486. Springer (2018)

[58] Frehse, G., Kateja, R., Le Guernic, C.: Flowpipe approximation and clustering in space-time. In: HSCC. pp. 203–212. ACM (2013)

[59] 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: CAV. Lecture Notes in Computer Science, vol. 6806, pp. 379–395. Springer (2011)

[60] Funahashi, K.I.: On the approximate realization of continuous mappings by neural networks. Neural Networks **2**(3), 183–192 (1989)

[61] Gao, S., Kong, S., Clarke, E.M.: dReal: an SMT solver for nonlinear theories over the reals. In: CADE, Lecture Notes in Computer Science, vol. 7898, pp. 208–214. Springer (2013)

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

[63] Geretti, L., Sandretto, J.A.D., Althoff, M., Benet, L., Chapoutot, A., Collins, P., Duggirala, P.S., Forets, M., Kim, E., Linares, U., Sanders, D.P., Schilling, C., Wetzlinger, M.: ARCH-COMP21 category report: Continuous and hybrid systems with nonlinear dynamics. In: ARCH@ADHS. EPiC Series in Computing, vol. 80, pp. 32–54. EasyChair (2021)

[64] Giacobbe, M., Kroening, D., Parsert, J.: Neural termination analysis. In: ESEC/SIGSOFT FSE. ACM (2022), to appear

[65] 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)

[66] 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)

[67] Habets, L.C.G.J.M., Kloetzer, M., Belta, C.: Control of rectangular multi-affine hybrid systems. In: CDC. pp. 2619–2624. IEEE (2006)

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

[69] Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? J. Comput. Syst. Sci. **57**(1), 94–124 (1998)

[70] 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)

[71] Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks **2**(5), 359–366 (1989)

[72] 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)

[73] Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: CAV (1). Lecture Notes in Computer Science, vol. 10426, pp. 3–29. Springer (2017)

[74] 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)

[75] Kapinski, J., Deshmukh, J.V., Sankaranarayanan, S., Aréchiga, N.: Simulation-guided Lyapunov analysis for hybrid dynamical systems. In: HSCC. pp. 133–142. ACM (2014)

[76] Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljic, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.W.: The Marabou framework for verification and analysis of deep neural networks. In: CAV (1). Lecture Notes in Computer Science, vol. 11561, pp. 443–452. Springer (2019)

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

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

[79] 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)

[80] Kloetzer, M., Belta, C.: Reachability analysis of multi-affine systems. In: HSCC. Lecture Notes in Computer Science, vol. 3927, pp. 348–362. Springer (2006)

[81] 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)

[82] 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)

[83] Lafferriere, G., Pappas, G.J., Sastry, S.: O-minimal hybrid systems. Math. Control. Signals Syst. **13**(1), 1–21 (2000)

[84] Lafferriere, G., Pappas, G.J., Yovine, S.: A new class of decidable hybrid systems. In: HSCC. Lecture Notes in Computer Science, vol. 1569, pp. 137–151. Springer (1999)

[85] Le Guernic, C., Girard, A.: Reachability analysis of hybrid systems using support functions. In: CAV, Lecture Notes in Computer Science, vol. 5643, pp. 540–554. Springer (2009)

[86] Lechner, M., Zikelic, D., Chatterjee, K., Henzinger, T.A.: Stability verification in stochastic control systems via neural network supermartingales. In: AAAI. pp. 7326–7336. AAAI Press (2022)

[87] Leshno, M., Lin, V.Y., Pinkus, A., Schocken, S.: Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. Neural Networks **6**(6), 861–867 (1993)

[88] 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)

[89] Liu, C., Arnon, T., Lazarus, C., Strong, C.A., Barrett, C.W., Kochenderfer, M.J.: Algorithms for verifying deep neural networks. Found. Trends Optim. **4**(3-4), 244–404 (2021)

[90] Lu, Z., Pu, H., Wang, F., Hu, Z., Wang, L.: The expressive power of neural networks: A view from the width. In: NIPS. pp. 6231–6239 (2017)

[91] 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)

[92] Manzanas Lopez, D., Musau, P., Hamilton, N., Johnson, T.T.: Reachability analysis of a general class of neural ordinary differential equations. In: FORMATS. Lecture Notes in Computer Science, vol. 13465, pp. 258–277. Springer (2022)

[93] 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)

[94] Peruffo, A., Ahmed, D., Abate, A.: Automated and formal synthesis of neural barrier certificates for dynamical models. In: TACAS (1). Lecture Notes in Computer Science, vol. 12651, pp. 370–388. Springer (2021)

[95] Pola, G., Girard, A., Tabuada, P.: Approximately bisimilar symbolic models for nonlinear control systems. Autom. **44**(10), 2508–2516 (2008)

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

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

[98] Ravanbakhsh, H., Sankaranarayanan, S.: Learning control Lyapunov functions from counterexamples and demonstrations. Auton. Robots **43**(2), 275–307 (2019)

[99] 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)

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

[101] 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)

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

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

[104] Schupp, S., Ábrahám, E., Makhlouf, I.B., Kowalewski, S.: Hypro: A C++ library of state set representations for hybrid systems reachability analysis. In: NFM. Lecture Notes in Computer Science, vol. 10227, pp. 288–294 (2017)

[105] Solar-Lezama, A.: Program Synthesis by Sketching. Ph.D. thesis, University of California at Berkeley, USA (2008)

[106] Solar-Lezama, A., Tancau, L., Bodík, R., Seshia, S.A., Saraswat, V.A.: Combinatorial sketching for finite programs. In: ASPLOS. pp. 404–415. ACM (2006)

[107] Sotomayor, J., Mello, L., Braga, D.: Bifurcation analysis of the watt governor system. Computational and Applied Mathematics **26** (Jul 2006)

[108] Sun, D., Jha, S., Fan, C.: Learning certified control using contraction metric. In: CoRL. Proceedings of Machine Learning Research, vol. 155, pp. 1519–1539. PMLR (2020)

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

[110] Tran, H., Cai, F., Manzanas Lopez, D., 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)

[111] Tran, H., Yang, X., Manzanas Lopez, D., 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)

[112] Van Der Schaft, A.J., Schumacher, H.: An introduction to hybrid dynamical systems, vol. 251. Springer (2007)

[113] 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)

[114] Zhao, H., Zeng, X., Chen, T., Liu, Z.: Synthesizing barrier certificates using neural networks. In: HSCC. pp. 25:1–25:11. ACM (2020)

[115] Zhao, H., Zeng, X., Chen, T., Liu, Z., Woodcock, J.: Learning safe neural network controllers with barrier certificates. Formal Aspects Comput. **33**(3), 437–455 (2021)

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 generation 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]

A Benchmark Nonlinear Dynamical Models

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 $[-1, 1]^n$.

Water Tank

$$\begin{cases} \dot{x} = 1.5 - \sqrt{x} \\ \mathcal{X}_0 = [0, 0.01] \\ \mathcal{X}_B = \{x | x \geq 2\} \end{cases} \tag{13}$$

Jet Engine [16]

$$\begin{cases} \dot{x} = -y - 1.5x^2 - 0.5x^3 - 0.1, \\ \dot{y} = 3x - y, \\ \mathcal{X}_0 = [0.45, 0.50] \times [-0.60, -0.55] \\ \mathcal{X}_B = [0.3, 0.35] \times [0.5, 0.6] \end{cases} \tag{14}$$

Steam Governor [107]

$$\begin{cases} \dot{x} = y, \\ \dot{y} = z^2 \sin(x) \cos(x) - \sin(x) - 3y, \\ \dot{z} = -(\cos(x) - 1), \\ \mathcal{X}_0 = [0.70, 0.75] \times [-0.05, 0.05] \times [0.70, 0.75] \\ \mathcal{X}_B = [0.5, 0.6] \times [-0.4, -0.3] \times [0.7, 0.8] \end{cases} \tag{15}$$

Exponential

$$\begin{cases} \dot{x} = -\sin(\exp(y^3 + 1)) - y^2 \\ \dot{y} = -x, \\ \mathcal{X}_0 = [0.45, 0.5] \times [0.86, 0.91] \\ \mathcal{X}_B = [0.3, 0.4] \times [0.5, 0.6] \end{cases} \tag{16}$$

Non-Lipschitz Vector Field 1 (NL1)

$$\begin{cases} \dot{x} = y \\ \dot{y} = \sqrt{x} \\ \mathcal{X} = [0, 1] \times [-1, 1], \\ \mathcal{X}_0 = [0, 0.05] \times [0, 0.1] \\ \mathcal{X}_B = [0.35, 0.45] \times [0.1, 0.2] \end{cases} \tag{17}$$

Non-Lipschitz Vector Field 2 (NL2)

$$\begin{cases} \dot{x} = x^2 + y \\ \dot{y} = \sqrt[3]{x^2} - x, \\ \mathcal{X}_0 = [-0.025, 0.025] \times [-0.9, -0.85] \\ \mathcal{X}_B = [-0.05, 0.05] \times [-0.8, -0.7] \end{cases} \tag{18}$$

B Additional Experimental Results and Figures

B.1 Experimental Comparison Against Affine Simplical Meshes

In this section, we present some supplementary empirical results on neural abstractions. Firstly, we note that hybridisation-based abstraction of nonlinear models have been studied previously, such as in [15], which describes a type of hybridisation-based abstractions that is similar to those constructed in this work. The approach relies first on partitioning the state space using a simplicial mesh grid, and

Table 2: A comparison between abstractions constructed using an affine simplicial mesh and neural abstractions. Here, W represents the neural structure used for neural abstraction, N_P: total number of partitions, ϵ: the calculated upper bound on the approximation error, $\bar{N}_P$: average (mean) number of partitions, $\bar{\epsilon}$: average (mean) approximation error bound, ϵ^+ : the maximum approximation error, ϵ^-: the minimum approximation error, Success Ratio: the ratio of repeated experiments that terminated successfully (i.e., an error of 0.5 was reached within the first timeout of 300s). Note, we only include successful experiments when calculating the average, min and max (since no error exists for unsuccessful experiments). All reported errors use the `2-norm`.

Benchmark	Affine Simplicial Mesh		Neural Abstractions					
	N_p	ϵ	W	$\bar{N}_P$	$\bar{\epsilon}$	ϵ^+	ϵ^-	Success Ratio
Jet Engine	8	1.33	[10]	9	0.11	0.22	0.040	1.0
	32	0.33	[10, 10]	27	0.077	0.17	0.040	1.0
	128	0.083	[15, 15]	61	0.058	0.071	0.053	1.0
Steam	24	3.58	[10]	27	0.27	0.37	0.21	1.0
	192	0.89	[20]	236	0.18	0.27	0.15	1.0
Exponential	8	13.7	[10]	9	0.29	0.40	0.22	0.5
	32	3.44	[20]	30	0.19	0.22	0.13	0.9
	128	0.86	[20, 20]	75	0.15	0.22	0.071	1.0

then allowing the dynamics in each mesh to be calculated from an affine interpolation between the vertices of the simplex. This affine simplicial mesh (ASM) based approach constructs abstractions of the same expressivity as neural abstractions (first order approximations) with partitions defined by affine inequalities. An approximation-error bound for ASM can be calculated for systems which have bounded second order derivatives using the model dynamics and the size of each simplex (all simplices are assumed to be the same size), as described in [15]. In Table 2 we compare between abstractions constructed using an affine simplicial mesh and neural abstractions. We run our procedure to synthesise certified abstractions using selected network structures and an initial target error of 0.5. If a successful abstraction is synthesised, we reduce the error by some multiplicative factor and repeat. This iterative procedure continues until no success is reached within a time of 300s. We report the results from 10 repeated experiments over different initial random seeds for neural abstractions, reporting the average (mean), minimum and maximum results obtained. In contrast, we report the approximation-error bound for ASM for different numbers of partitions.

The results reported in Table 2 illustrate that neural abstractions outperform ASM based abstractions in terms of error for similar numbers of partitions. Furthermore, neural abstractions generally require significantly fewer partitions for significantly lower approximation-error bounds. In practice this means neural abstractions will outperform ASM-based abstractions for safety verification both in terms of speed and accuracy. We also note the success ratio of our experiments, i.e., the ratio of all experiments which achieve an approximation-error bound of 0.5 or less. These results suggest that in general or procedure is robust and terminates successfully with high probability for reasonable target errors.

We note that since ASM based abstractions are constructive and are able to deterministically increase the number partitions and consequently reduce the error, for very large numbers of partitions they would achieve lower errors than neural abstractions. However, in practice these abstractions would be too large in complexity to use with SpaceEx for safety verification.

B.2 Computation Run-time Profiling

In Table 3 we show a breakdown of the runtimes of our procedure shown in the main text. In particular, we present the total time spent during learning, certification of the abstraction and finally in safety verification.

Table 3: Breakdown of the timings shown in Table 1. Shown are the timings in the constituent component shown in Figure 2: time spent during learning, time spent during certification of the neural abstraction, and time spent during safety verification. Remaining time is spent in overheads, such as converting from neural network to hybrid automaton.

Model	Learner	Certifier	Safety Verification
Jet Engine	19	194	1.8
Steam Governor	42	177	0.5
Exponential	27	278	3.3
Water-tank	48	0.001	0.05
Non-Lipschitz 1	13	0.50	5.5
Non-Lipschitz 2	31	15	5.1

C Improved Translation from Neural Abstractions to Hybrid Automata

C.1 Computing Invariant Conditions

Invariant conditions are computed from the configuration of a neural network denoted as the sequence $C = (c_1, \ldots, c_k)$ of Boolean vectors $c_1 \in \{0,1\}^{h_1}, \ldots, c_k \in \{0,1\}^{h_k}$, where k denotes the number of hidden layers and $h_1, \ldots, h_k$ denote the number neurons in each of them (cf. Section 2). Every vector c_i represents the configuration of the neurons at the ith hidden later, and its jth element $c_{i,j}$ represents the activation status of the jth neuron at the ith layer. Every mode of the hybrid automaton corresponds to exactly one configuration of neurons. In turn, every configuration of neurons C restricts the neural network $\mathcal{N}$ into a linear function. More precisely, we inductively define the linear restriction at the ith hidden layer as follows:

$$\mathcal{N}_C^{(i)}(x) = \mathrm{diag}(c_i)(W_i \mathcal{N}_C^{(i-1)}(x) + b_i), \text{ for } i = 1, \ldots, k, \qquad \mathcal{N}_C^{(0)}(x) = x. \tag{19}$$

We define the invariant of each mode as a restriction of the domain of interest to a region $\mathcal{X}_C \subseteq \mathcal{X}$, which denotes the maximal set of states that enables configuration C. To construct $\mathcal{X}_C$, we begin with the observation that the activation configuration c_i at every ith hidden layer induces a halfspace on the vector space of the previous layer of the neural network. Then, the pre-image of this halfspace backward along the previous layers of the linear restriction of the network characterises a corresponding halfspace on its input neurons. Since the input neurons are equivalent to the state variables of the dynamical model, the halfspace induced by layer i projected onto state variables x is

$$\mathcal{H}_C^{(i)} = \text{pre-image of } \underbrace{\{y_{i-1} \mid \mathrm{diag}(2c_i - 1)(W_i y_{i-1} + b_i) \geq 0\}}_{\text{halfspace induced by } i\text{th layer onto } (i-1)\text{th layer}} \text{ under } \mathcal{N}_C^{(i-1)} \tag{20}$$

The pre-image of a set $\mathcal{Y}$ under a function g is defined as $\{x \mid g(x) \in \mathcal{Y}\}$ and can be generally computed by quantifier elimination or, in the linear case, double description methods. However, these methods have worst-case exponential time complexity. To obtain $\mathcal{X}_C$ efficiently, we can leverage the fact that the pre-image of any halfspace $\{y \mid c^\mathsf{T} y \leq d\}$ under any affine function $g(x) = Ax + b$ equals to the set $\{x \mid c^\mathsf{T} y \leq d \wedge y = Ax + b\}$, which in turn defines the halfspace $\{x \mid c^\mathsf{T} Ax \leq d - c^\mathsf{T} b\}$. Therefore, since $\mathcal{N}_C^{(i-1)}$ is an affine function, every halfspace can be projected backward through the affine functions $\mathcal{N}_C^{(i-1)}, \ldots, \mathcal{N}_C^{(1)}$ using $\mathcal{O}(k)$ linear algebra operations. Finally, the entire invariant condition for configuration C is defined as the following polyhedron:

$$\mathcal{X}_C = \cap \{\mathcal{H}_C^{(i)} \mid i = 1, \ldots, k\} \cap \mathcal{X}. \tag{21}$$

An invariant condition thus results in a polyhedron defined as the intersection of k halfspaces together with the constrains that define the domain of interest. Notably, under the definition in this appendix, the dynamics of mode C given in Equation 10 correspond to the affine dynamical model

$$\dot{x} = \mathcal{N}_C^{(k+1)}(x) + d, \quad \|d\| \leq \epsilon, \quad x \in \mathcal{X}_C, \tag{22}$$

whose dynamics are governed by the affine function

$$\mathcal{N}_C^{(k+1)}(x) = W_{k+1} \mathcal{N}_C^{(k)}(x) + b_{k+1}. \tag{23}$$

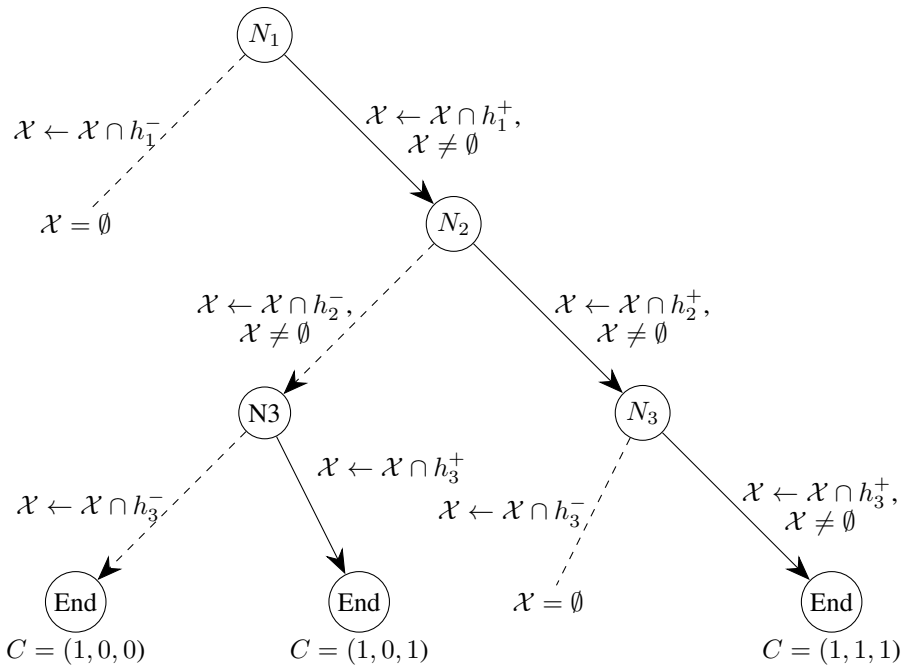

Figure 4: Example Tree search to determine the active configurations for a neural network consisting of a single hidden layer with 3 neurons. Here, h_i^+ denotes the positive half-space ($\{x : w_i x + b_i \geq 0\}$) and h_i^- denotes the negative half-space ($\{x : w_i x + b_i \leq 0\}$) of the ith neuron; w_i represents the ith row of the weight matrix corresponding to the hidden layer, and b_i represents the ith element of the bias vector of the hidden layer. Notably, when the set $\mathcal{X}$ becomes empty, it is no longer necessary to continue along that path. Once we reach the end of the tree, we have an active configuration C, and backtrack to the last node that was not fully explored.

C.2 Enumerating Feasible Modes

Determining whether a mode C exists in the hybrid automaton amounts to determining the linear program (LP) associated to polyhedron $\mathcal{X}_C$ is feasible. Finding all modes therefore consists of solving 2^H linear programs, where $H = h_1 + \cdots + h_k$ is the total number of neurons. This scales exponentially in the number of neurons. Here, we elaborate on the tree search algorithm described in Section 4.2 using a diagram; the purpose of this algorithm is to efficiently determine all active neuron configurations within a bounded domain of interest $\mathcal{X}$.

We consider an example tree in Figure 4, which depicts an example search for a neural network with a single hidden layer consisting of three neurons. The tree illustrates the construction of $\mathcal{X}_C$ through repeated intersections of half-spaces as paths are taken through the tree structure. Nodes represent each neuron, labelled N_i, $i = 1, 2, 3$ and each edge represents one of two possible half-spaces for the neuron it leaves from (ReLU enabled, solid line, and disabled, dashed line). This approach allows us to prune neurons and overall solve significantly fewer linear programs than simply enumerating through all possible configurations.