Formal Verification of Dynamical Models via Neural Synthesis

Alec Edwards

Worcester College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy in Computer Science

2024

"To light a candle
is to cast a shadow..."
— Ursula K. Le Guin

Acknowledgements

This thesis details the research I conducted during my PhD. This journey has been fraught with obstacles both external and self-imposed which could not have been overcome alone. Here, I hope to acknowledge the support I have received from others that has enabled this work to be completed.

First of all, I am grateful to my supervisor, Alessandro Abate, for his supervision and guidance throughout the course of my PhD. His patience, understanding, and uncompromising support have been a cherished and immutable luxury.

Throughout my PhD I have been fortunate enough to work with many different collaborators, to all of whom I am grateful for their impact on me and my research. In particular, Andrea Peruffo has been a better mentor and friend than I could ask for or deserve, and I'm grateful for our collaborations together so far. Daniele Ahmed's kindness and patience early on in my PhD, and in particular his feedback on good programming practice, has both improved my own ability and the quality of my research. I am thankful to Mirco Giacobbe for many discussions in which ideas could grow. Other collaborators, including Sergiy Bogomolov, Virginie Debauche, Raphael Jungers, Kostiantyn Potomkin, Hashan Punchihewa, Diptarko Roy, Sadegh Soudjani and Paolo Zuliani, have all been a pleasure to work with.

During my PhD I have been a member of the EPSRC CDT in Autonomous Intelligent Machines and Systems (AIMS, EP/S024050/1), which has been a privilege to be a part of. Its student members have been kind, encouraging friends and colleagues

who I have learned a lot from and had a lot of fun with. I am grateful to the CDT's staff, in particular to its administrator Wendy Poole who has been a constant source of support.

I would like to thank my examiners, Rupak Majumdar and Kostas Margellos, for their time and effort in reviewing my thesis. I hope I have addressed their comments and suggestions to their satisfaction, and this work is enhanced by their feedback. I am thankful to my internal assessors: James Worrell, Konstantinos Gatsis and Christoph Haase, who have provided constructive feedback and guidance at significant milestones in my PhD. I would like to thank my college advisor and tutor, Antonis Papachristodoulou, who also first gave me the confidence to pursue a research degree.

Thank you to my parents, Sandra and Clive, for their unyielding devotion to my education and my happiness.

And finally thank you to Becca, who is my light, my rock, and my joy.

Abstract

Dynamical models are a mathematical tool for representing, understanding, and analysing complex systems that are ubiquitous across science and engineering. Complexity and nonlinearity in these systems result in a lack of analytical solutions and challenging, or even intractable, analysis in view of safety-critical applications. Effective methodologies for the verification of behaviours of dynamical models are therefore of great importance, and their effectiveness is often determined by their ability to handle nonlinearity. This thesis studies two alternative approaches to verifying continuous-time nonlinear dynamical models: *abstraction*, which directly disregards nonlinearity, and *indirect* certificate-based approaches, which indirectly handle nonlinearity. The strength of neural networks as universal approximators has seen widespread interest from the formal verification community. Commonly, this is in the context of verifying machine learning models themselves. Conversely, this thesis explores the use of neural networks as a tool for verification, in particular for dynamical models. In particular, we introduce a novel approach for abstracting nonlinear dynamical models via neural networks, and contribute to growing literature which leverages neural networks as certificates for specifications of dynamical models. For the former, we demonstrate that these abstractions, which we call *neural abstractions*, can be used to verify safety properties of nonlinear dynamical models. For the latter, we demonstrate that neural networks may be used in conjunction with neural-based controllers to ensure controlled dynamical models satisfy a range of specifications. Both approaches require a formal synthesis procedure to construct neural networks which satisfy required properties. We utilise counterexample-guided inductive synthesis (CEGIS) to this end, in which neural networks are employed as

approximators alongside solvers for satisfiability modulo theories to provide formal guarantees of correctness. We demonstrate that this synthesis approach is effective for both abstractions and certificates. Through a range of experimental case studies, we demonstrate that the use of neural networks in both abstraction and certificate function synthesis is a flexible, effective approach to verifying nonlinear dynamical models.

Contents

List of Figures

List of Tables

Acronyms

ASM Affine Simplicial Mesh.

CEGIS Counterexample-Guided Inductive Synthesis.

CLF Control Lyapunov Function.

DFA Deterministic Finite Automaton.

HA Hybrid Automaton.

LHA Linear Hybrid Automaton.

LNN Lyapunov Neural Network.

LP Linear Program.

LQR Linear Quadratic Regulator.

LTL Linear Temporal Logic.

MILP Mixed Integer Linear Program.

NN Neural Network.

NP Nondeterministic Polynomial.

ODE Ordinary Differential Equation.

PWA Piecewise Affine.

PWC Piecewise Constant.

RAR Reach Avoid Remain.

ROA Region of Attraction.

RSWA Reach Stay While Avoid.

RWA Reach While Avoid.

SAT Satisfiability.

SMT Satisfiability Modulo Theories.

SOS Sum of Squares.

STC Space Time approximation with Clustering.

SWA Stable While Avoid.

TM Taylor Model.

Chapter 1

Introduction

1.1 Motivation and Research Questions

Nonlinearity is hard. That is not to say that linearity is easy, but for it exists various properties and theorems that enable thorough analysis which often scales well. Nonlinearity does not (in general) abide by such rules. This is in particular true for dynamical models — those of coupled ordinary differential equations which mathematically model dynamical systems ubiquitous in the natural world. Even when linear, these mathematical models can be tricky to handle, by the nature of their structure. When nonlinear, direct integration of the solutions is often intractable. We are left to analyse trajectories and behaviour of the coupled ODEs through more indirect means.

In the case of linear dynamical models, eigenvalues pose a powerful tool for insight into the behaviour [152]. In fact, under certain conditions, the linearisation of a nonlinear model provides valuable insight into the stability of the original model by virtue of its eigenvalues. Linearisation is therefore a common first-step in the analysis of nonlinear dynamical models.

Nonetheless, the ubiquity of nonlinear dynamical systems, and their corresponding

mathematical models, means their analysis is both critical and compelling [103, 152]. Research in dynamical models has developed two standout (at least from the perspective of this thesis) approaches to handle nonlinearity in dynamical models: abstraction (often called *direct* methods) and indirect approaches (in the form of *sufficient* function-based certificates). These approaches are alike in their decision to handle nonlinearity by seeking to disregard it. Abstraction does this directly, certificates indirectly. In this thesis we explore both approaches; we shall briefly introduce them here, but leave a more detailed review to their respective chapters.

Abstraction Abstraction of mathematical models is a well established technique in the field of formal verification [21]. It begins with a *concrete* model, a mathematical model of a system, which is to be abstracted. The abstraction process then involves the construction of an *abstract* model, which is (in principle) a simpler model, but retains certain key properties of the concrete model. This simpler, abstract model should be more amenable to analysis than the concrete model. Approaches to abstraction vary depending on the type of model being abstracted, and the properties of interest. However, the general principle remains the same: superfluous information is discarded in lieu of a simpler model that retains properties of interest. Within formal verification, abstraction was originally studied for finite state models [29, 160], but has since been extended to infinite-state models through the concept of approximate bisimulation [78, 132].

In the case of dynamical models, abstraction is often performed by partitioning the state space of the model into a finite number of regions, and then in each region constructing a simpler (e.g., linear) model that approximates the behaviour of the concrete model in that region [18]. The resulting models are finite state machines, and are commonly modelled as hybrid automata. This abstraction process is therefore known as *hybridisation*; the resulting hybrid automaton may then be analysed

to determine properties of the concrete model. Hybridisation-based approaches suffer from lack of scalability in both the number of dimensions, but also in the number of partitions of the state space. The number of partitions grows exponentially with the number of dimensions, and determining an efficient partitioning of the state space is a difficult task. To counter this, hybridisation approaches may proceed dynamically by constructing the abstraction as the model is analysed (as in [56]), or by splitting partitions to achieve a more accurate abstraction (as in e.g., [61, 69]). These techniques may mitigate some of the aforementioned issues, but they do not completely negate them. It may be easy to add partitions, but it is less clear how to remove them. Furthermore, the shape (or syntax) of the partitions is still fixed a priori. This last point is of particular interest to this thesis, as we shall see in Chapter 3.

Indirect Analysis The second type of approach to the analysis of dynamical models are indirect approaches, which involve the construction of a *certificate* for the model. We shall therefore also refer to these as *certificate-based* approaches. Certificate-based approaches for dynamical models originate from Lyapunov's second method for stability. This involves the construction of a function-based proof of stability for the model, known as a Lyapunov function. A Lyapunov function is a scalar-valued function of the state of the model, and acts as an "energy function" for the model. By construction, a Lyapunov function implies that the energy of the model decreases over time to a minimum (which is the equilibrium point), and hence the model is stable.

Lyapunov functions have historically been constructed by hand, requiring ingenuity and insight into the model. To counter this difficult task, optimisation-based synthesis techniques have grown in interest, none more so than sum-of-squares (SOS) programming [127]. This reduces the synthesis of (polynomial) Lyapunov functions to a (convex) semidefinite program, which can be solved efficiently. Lyapunov func-

tions are generally nonlinear; however, it proves easier to check if a nonlinear function (the Lie derivative of the Lyapunov function along the vector field of the model) is everywhere negative than to directly solve the nonlinear dynamical model. Thus the nonlinearity of the original model is indirectly disregarded in favour of the nonlinearity of the Lyapunov function, for which we must simply prove its sign. This is the essence of indirect approaches for dynamical models.

The success of SOS-based synthesis of Lyapunov functions has led to the emergence of a *certificate synthesis* field of research. This involves both synthesis through alternative methods to SOS, but also the synthesis of certificates for properties other than stability, such as safety [136] and reachability [143]. These certificates do not rely on direct analysis of the model, and instead indirectly prove properties of the models by proving properties of the certificate (and are therefore referred to as *indirect* approaches). Furthermore, a natural extension is the determination of control inputs that ensure the satisfaction of the property of interest. This may be achieved through the synthesis of a feedback controller whose suitability is certified through the synthesis of a certificate, or through the synthesis of a control certificate that is then used to determine the control inputs at any given state that ensure the property of interest is satisfied. The former approach is taken in this thesis, and will be discussed further in Chapter 4.

Towards Neural Synthesis The work in this thesis is facilitated by two key maturing technologies: neural networks, which we leverage as approximators, and satisfiability modulo theories (SMT), which provide correctness guarantees to our abstractions and certificates. The rise of machine learning has seen techniques from it, in particular neural networks, filter throughout research domains. As compact and concise universal function approximators [53, 74, 91] with a highly efficient training mechanism in backpropagation (alongside high performance implementations of this),

neural networks have seen success in a wide range of applications. These advantages make them desirable over universal approximators such as polynomials. However, their rise in popularity has been followed by growing concerns over lack of guarantees for their reliability and robustness. In view of the deployment of these models in safety-critical systems, fields of verification have emerged to address crucial questions regarding their guarantees. This thesis considers a similar but different question: how can the approximation power of neural networks be leveraged to aid classical verification questions of dynamical models? This question still requires proving properties of neural networks, but the focus is on the use of neural networks to aid verification queries for other models, rather than the verification of neural networks themselves.

Meanwhile, within the field of verification, solvers for the satisfiability modulo theories problem – known as SMT-solvers – have seen significant development in recent decades [59, 106, 124]. SMT generalises the boolean satisfiability problem: that of determining an assignment to boolean variables that satisfies a given boolean formula in first-order logic. SMT extends this to further formal theories, enabling satisfiability queries for formulae of integer or real numbers, for example. SMT-solvers enable automated verification tasks that are useful in a wide range of applications.

The growth and combination of these two technologies has seen the emergence of neural synthesis techniques [4, 40, 171]. Here, the approximation power of neural networks may be leveraged while retaining guarantees of correctness through the use of verification, most commonly SMT-solving. This thesis explores the use of such techniques under the framework of counterexample-guided synthesis, which we shall introduce in due course. Notably, SMT is not the only suitable approach for the verification of neural synthesis: mixed integer linear programming is amongst other approaches that have been explored, especially for networks constructed from piecewise affine activation functions [171]. However, in this thesis we focus on the use of SMT-solvers to verify properties of neural networks, which in turn aid the

verification of dynamical models.

Furthermore, we note that bespoke verification techniques for analysing neural networks have been developed. These include simplex-based approaches [100, 99], geometric path-enumeration [25], linear approximation [66], branch-and-bound [37], and abstraction [67]. These approaches are powerful, but are often limited to linear queries over specific classes (commonly piecewise affine) of neural networks. In this work, we instead require queries which involve both nonlinear neural networks and nonlinear dynamical models, and hence require a more general approach.

The verification of dynamical models is a crucial and unsolved task; abstraction and certificate-based methods are two key approaches for tackling this problem. The success of neural networks in other fields leads to research questions regarding their potential and suitability within existing paradigms in formal verification. This thesis tackles this research question, by investigating the use of neural networks both as a means to construct abstractions of dynamical models, and also as certificates for proving properties of dynamical models hold. Neural networks have already been used as certificates for dynamical models, but this thesis explores the use of neural networks as certificates for a wide range of properties alongside the concurrent synthesis of feedback controllers. Meanwhile, the use of neural networks as abstractions of dynamical models is a novel approach.

We shall see that the use of neural networks provides a powerful, general-purpose tool for the verification of dynamical models. Correspondingly, we shall see that the use of SMT-solvers, as well as other verification techniques, enables the proving of queries involving both neural networks and dynamical models. This thesis therefore explores the intersection of these two fields, and contributes to the growing body of work that studies the use of neural networks in formal verification.

1.2 Overview of Thesis

In this section we describe the structure of this thesis. We delegate a detailed discussion on the related literature, contributions and limitations of this thesis to the relevant chapters. Here we provide a brief overview of each chapter, and the relevant contributions.

- Chapter 2 provides the necessary background for the remainder of the thesis. This includes a brief introduction to dynamical and hybrid systems, and the mathematical models that describe them pertinent to this work. We also introduce the counterexample guided inductive synthesis (CEGIS) framework, and subsequently the concepts of SMT-solvers and neural networks, which we leverage as part of this synthesis framework.

- Chapter 3 discusses the abstraction of nonlinear dynamical systems using neural networks. We introduce the novel concept of neural abstraction, and discuss the synthesis of neural abstractions using counterexample guided inductive synthesis. We follow this by describing a methodology for the verification of nonlinear dynamical models using neural abstractions through the casting of abstractions to equivalent abstract models. We present results demonstrating how this approach enables verification of nonlinear dynamical models that do not exhibit local Lipschitz continuity which are difficult to handle using existing approaches.

- Chapter 4 presents a general framework for the control and verification of dynamical models through the concurrent synthesis of neural network based feedback controllers and certificates. We collate, augment and categorise existing certificates across the literature, and provide a unified framework for their synthesis. This culminates in an extensive set of experiments demonstrating the efficacy of this approach at synthesising controllers that guide dynamical models

to satisfy a range of specifications alongside certificates that prove the closed-loop model satisfies the required specification.

- Chapter 5 provides final concluding remarks and discusses suitable directions for future work.

1.2.1 Publications and Contributions

The work in this thesis has either been published or submitted for publication to high quality conference venues and journals. These research publications are listed in this section. I am the lead contributor to each of these three publications, having led the research, implementation and writing of the papers. The first two papers, [6, 63], form the basis of Chapter 3, while the third paper, [65], forms the basis of Chapter 4. For [6, 63], I led the implementation and algorithm development, and developed the theory alongside Mirco Giacobbe. The writing of [6] was a group effort, while [63] was written mostly by myself. For [65], I led the implementation and algorithm development, in collaboration with Andrea Peruffo; the writing of the paper was a group effort led by myself.

- Abate, A., Edwards, A., Giacobbe, M.: Neural abstractions. In: Thirty-Sixth Conference on Neural Information Processing Systems (2022)
  - This paper introduces the concept of neural abstraction, which we define as abstractions of dynamical models consisting of neural networks.
  - It presents the synthesis of neural abstractions consisting of ReLU networks using counterexample guided inductive synthesis.
  - It also presents a methodology for the safety verification using neural abstractions, and benchmarks the approach against the tool Flow*.

- Edwards, A., Giacobbe, M., Abate, A.: On the trade-off between efficiency and precision of neural abstraction. In: Quantitative Evaluation of Systems (2023)
  - This paper extends the work in [6] to the synthesis of neural abstractions which represent piecewise constant functions and sigmoidal functions, and extends the methodology for safety verification to these abstractions.
  - It studies and compares different model semantics against each other in terms of the trade-off between accuracy and computational efficiency.
  - It presents extensive experimental results on a range of benchmarks, significantly improving the robustness of the synthesis approach presented in [6].
- Edwards, A., Peruffo, A., Abate, A.: A General Verification Framework for Dynamical and Control Models via Certificate Synthesis (2024), under review, arXiv:2309.06090
  - This work presents a general framework for the synthesis of certificates for dynamical models, and the concurrent synthesis of feedback controllers and certificates.
  - It presents extensive benchmarks in which the framework is used to synthesise controllers and certificates for a range of dynamical models and specifications.

In addition to these publications, the work in this thesis also relates to the following tool papers, which we discuss separately here.

- 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. HSCC '21 (May 2021)

- Edwards, A., Peruffo, A., Abate, A.: Fossil 2.0: Formal Certificate Synthesis for the Verification and Control of Dynamical Models. In: Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control. HSCC '24, Association for Computing Machinery, New York, NY, USA (2024)

Since they are tool papers, they do not present significant novel research contributions. Instead, they present the implementation of software tools developed as part of the underlying scientific research, and detail how a user may interface, use and extend this tool for their own research or applications. The first of these details the implementation of the tool Fossil [3], whose computation library is used in Chapter 4. The writing of this paper was a group effort, and I supported the writing of the tool and development of the portfolio of benchmarks as part of the team of authors. In turn, the second of these [64] details the implementation of a second major release of this tool which was developed subsequent to the work present in Chapter 4. As such, [64] contains empirical results to evidence the efficacy of the tool, taken from the experiments in Chapter 4 (and [65]). Therefore, the details of these tool papers are not repeated in this thesis as they do not detail the novel scientific contributions of the research.

1.2.1.1 Other Publications by the Author

The following publications by the author are not directly related to the work in this thesis, but are related to the topics discussed. They deal with the verification of mathematical models using neural networks, and so have influenced the work in this thesis by association.

- Abate, A., Edwards, A., Giacobbe, M., Punchihewa, H., Roy, D.: Quantitative verification with neural networks. In: 34th International Conference on Concurrency Theory (CONCUR) (2023)

- Debauche, V., Edwards, A., Abate, A., Jungers, R.M.: Stability analysis of switched linear systems with neural lyapunov functions. Proceedings of the AAAI Conference on Artificial Intelligence (2024)

- Abate, A., Bogomolov, S., Edwards, A., Potomkin, K., Soudjani, S., Zuliani, P.: Safe reach set computation via neural barrier certificates. IFAC **58**(11), 107–114 (2024), 8th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2024

Chapter 2

Preliminaries

This chapter introduces concepts and notations that are used throughout this thesis. We start by introducing the general notation used in this thesis. Then, we introduce models of dynamical and hybrid systems using dynamical models and hybrid automata respectively. We discuss counterexample guided inductive synthesis, and how this work leverages both satisfiability modulo theories (SMT) and neural networks to perform synthesis tasks. We note that salient concepts will be redefined in the relevant chapters consistently with this chapter, and that this chapter is intended to be a reference for the reader.

2.1 General Notation

We denote the set of integers by $\mathbb{Z}$, the set of non-negative integers by $\mathbb{Z}_{\geq 0}$, and the set of real numbers by $\mathbb{R}$. The set of real positive numbers is denoted by $\mathbb{R}_+$, and its extended version as $\overline{\mathbb{R}}_+ = \mathbb{R}_+ \cup \{+\infty\}$. Given a set $S \subset \mathbb{R}^n$, we denote the boundary of S by ∂S, and its interior by $\text{int}(S)$. The complement of S is denoted by $S^{\complement}$.

Throughout this work we discuss the concept of Lipschitz continuity, which is defined as follows.

Definition 2.1. *(Lipschitz Continuity) A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is Lipschitz continuous if there exists a constant $L \in \mathbb{R}_+$ such that for all $x, y \in \mathbb{R}^n$ it holds that*

$$\|f(x) - f(y)\| \leq L\|x - y\|, \tag{2.1}$$

where $\|\cdot\|$ denotes the norm of its argument.

■

2.2 Hybrid Systems Modelling

2.2.1 Dynamical Models

We begin by introducing a model describing dynamical systems, which we will then specialise to the models of interest in this work. We define a dynamical model as a system of nonlinear ordinary differential equations (ODEs), as follows.

Definition 2.2 (Disturbed Control Dynamical Model)**.**

$$\dot{\xi}(t) = f(\xi(t), u(t)) + d, \quad \|d\| \leq \delta, \quad \xi(t_0) = x_0 \in \mathcal{X}_I \subseteq \mathcal{X}. \tag{2.2}$$

■

Here, $t \in \mathbb{R}$ is time (with initial time $t_0 \geq 0$), $\xi(t) = x \in \mathcal{X} \subseteq \mathbb{R}^n$ is the state of the system, $f : \mathcal{X} \times \mathcal{U} \to \mathbb{R}^n$ is a continuous nonlinear vector field describing the model dynamics, $u \in \mathcal{U}$ is the input to the system, and $\mathcal{X}_I \subset \mathcal{X}$ is the set of initial conditions and $\mathcal{X}$ is the state space (sometimes referred to as the domain of interest). A trajectory defined over time horizon $T > t_0$ is a function $\xi\colon [t_0, T] \to \mathbb{R}^n$ that admits derivative at each point in $[t_0, T]$ such that, for all $t \in [t_0, T]$, it holds true that $\xi(t) \in \mathcal{X}$ and $\dot{\xi}(t) = f(\xi(t), u(t)) + d_t$ for some $\|d_t\| < \delta$. Notably, symbol d in Equation (2.2) is interpreted as a nondeterministic disturbance that at any time can take any possible value within the bound provided by δ. We limit our consideration

to models with additive bounded disturbance. As we shall see in Chapter 3, these models are useful for studying error-based abstractions of dynamical models.

We refer to models of this kind simply as dynamical models, and will now introduce three special cases of this model that are of interest to this work.

First, let us consider the case where there is no control input, i.e. $u(t) \equiv 0$, which we refer to as a disturbed dynamical model and study in Chapter 3.

Definition 2.3 (Disturbed Dynamical Model)**.**

$$\dot{\xi}(t) = f(\xi(t)) + d, \quad \|d\| \leq \delta, \quad \xi(t_0) = x_0 \in \mathcal{X}_I \subseteq \mathcal{X}. \tag{2.3}$$

■

Next, we consider the case where the model is not disturbed, i.e. $\delta = 0$, but is still subject to a control input $u(t)$. We refer to these models as control models, and they are studied in Chapter 4.

Definition 2.4 (Control Model)**.**

$$\dot{\xi}(t) = f(\xi(t), u(t)), \quad \xi(t_0) = x_0 \in \mathcal{X}_I \subseteq \mathcal{X}. \tag{2.4}$$

■

Finally, we consider the case where the model is neither disturbed nor subject to a control input, i.e. $\delta = 0$ and $u(t) \equiv 0$. We refer to these models simply as autonomous dynamical models, and they are studied in Chapters 3 and 4.

Definition 2.5 (Autonomous Dynamical Model)**.**

$$\dot{\xi}(t) = f(\xi(t)), \quad \xi(t_0) = x_0 \in \mathcal{X}_I \subseteq \mathcal{X}. \tag{2.5}$$

■

Fig. 2.1 summarises the different dynamical models studied in this work, the corresponding chapters in which they are studied, and the verification approaches

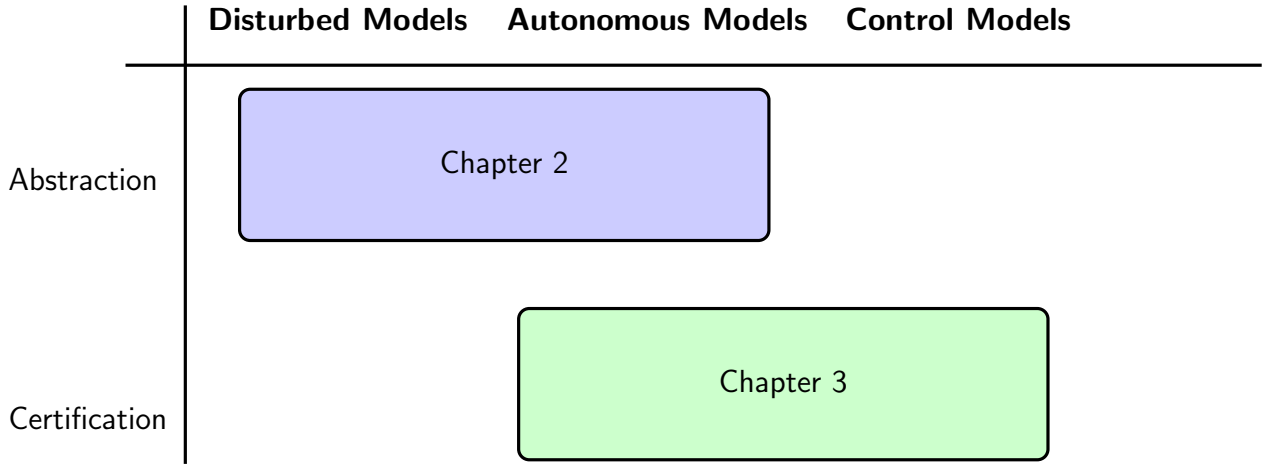


Figure 2.1: Diagram depicting the different dynamical models and verification approaches used in this work, and the corresponding chapters in which they are studied.

relevant to them. We shall comment on the assumptions made on the vector fields f, and the corresponding implications of the solutions of f, in the relevant chapters, since these assumptions are specific to the models studied in each chapter.

2.2.2 Hybrid Automata

Hybrid systems are systems which exhibit both discrete and continuous behaviour, and hybrid automata are a modelling formalism for hybrid systems. Multiple semantics for hybrid automata exist, and we will use the semantics used in [12, 73].

Define a hybrid automaton $\mathcal{H} = (Mod, Var, Lab, Inv, Flow, Trans)$, consisting of a labelled graph encoding the evolution of a finite set of real numbered variables $Var \subset \mathbb{R}^n$, in which the state variables x take their values. Each vertex $m \in Mod$ of the graph is referred to as a mode, and corresponds to a partition within the state space. The state of the automaton is thus given by the pair (m, x), whose evolution in time is described by $Flow(m, x)$. We describe this evolution as the flow, and in each mode the local flow given by $\dot{x} = f_m(x) + d$, where $f_m(x) : \mathbb{R}^n \to \mathbb{R}^n$ is a vector field and d is a nondeterministic disturbance such that $||d|| \leq \delta$. The edges of the graph describing $\mathcal{H}$ are known as transitions. A transition $(m, \alpha, Guard, Asgn, m') \in$

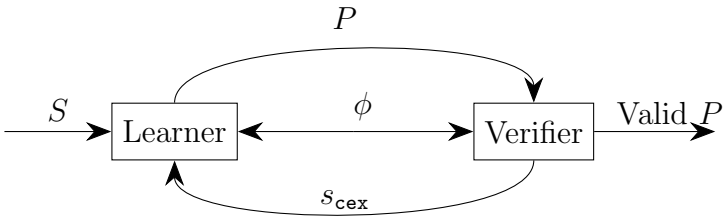


Figure 2.2: Framework for counter-example guided inductive synthesis to synthesise a function (or program) P that satisfies specification ϕ using finite sample set S.

$Trans$ has label $\alpha \in Lab$ and allows the system state to jump from (m, x) to the successor state (m', x'). Transitions are governed by the guard $Guard \subset \mathbb{R}^n$ and assignment $Asgn(x) \subset \mathbb{R}^n$, with a transition being enabled when $x \in Guard$ and $x' \in Asgn(x)$. Moreover, the state may only remain in a mode m if the state x is within the invariant $Inv(m) \subset \mathbb{R}^n$. Finally, transition assignments are of the form $x' = Rx + w$, where $R \in \mathbb{R}^{n \times n}$ and $w \in \mathcal{W}$, with $\mathcal{W} \subset \mathbb{R}^n$ denoting a compact convex set of nondeterministic disturbances (or inputs).

2.3 Counter-example Guided Inductive Synthesis

Counter-example guided inductive synthesis (CEGIS) is a framework for automated synthesis of programs or functions that satisfy a given specification. Its basic operation is depicted in Section 2.3, which shows the interaction between two opposing components as they seek to synthesise a function (or program) P that satisfies a given specification ϕ. The learner component is responsible for *guessing* candidate solutions, which it does inductively. Given a finite sample of data points S, the learner is tasked with finding a function that satisfies the specification ϕ across all of the data points. If it succeeds, this is known as a candidate solution.

Meanwhile the verifier component is responsible for checking whether the candidate solution is valid over the entire domain, rather than just the finite sample of data points. For the candidate solution to be valid, it must satisfy ϕ over the whole

state space, not just over the sample points. This can be achieved using symbolic reasoning, (enumerative) model checking, or using an optimisation-based approach. If the candidate solution is valid, the synthesis procedure terminates having successfully synthesised a function satisfying the specification. Otherwise, the verifier returns a counter-example $s_{\texttt{cex}}$, which is a data point for which the candidate solution does not satisfy the specification. This is added to the sample of data points which the learner uses to inductively synthesise a new candidate solution, and the process repeats.

Notably, in the case of an infinite search space, CEGIS is not complete, i.e., it may not successfully terminate even if a solution exists.

In this work, we shall use neural networks as the learner component, and SMT solvers as the verifier component, and we shall discuss these in the following sections.

2.4 Neural Networks

Denote a neural network $\mathcal{N}$ with input layer $z_0 \in \mathbb{R}^n$, corresponding to the dimension of the dynamical model in (2.2). This is followed by k hidden layers $z_1, \ldots, z_k$ with dimensions $h_1, \ldots, h_k$ respectively, and finally followed by an output layer $z_{k+1} \in \mathbb{R}^d$. In this work, $d \in \{1, m, n\}$, where $d = 1$ corresponds to a scalar-valued certificate, $d = m$ is used for a state-feedback controller with m control variables, and n corresponds to a network approximating a dynamical model.

To each hidden or output layer with index i are associated matrices of weights $W_i \in \mathbb{R}^{h_i \times h_{i-1}}$ and vectors of biases $b_i \in \mathbb{R}^{h_i}$ [114]. Every hidden layer i is associated with an activation function $\sigma_i \colon \mathbb{R} \to \mathbb{R}$. The valuation of hidden layers z_i and output layer z_{k+1} is given by

$$z_i = \sigma_i(W_i \cdot z_{i-1} + b_i), \ i = 1, \ldots, k, \tag{2.6}$$

$$z_{k+1} = W_{k+1} \cdot z_k + b_{k+1}, \tag{2.7}$$

where each σ_i is applied element-wise to its h_i-dimensional argument.

2.5 Satisfiability Modulo Theories

Satisfiability modulo theories (SMT) is the problem of deciding the satisfiability of a first-order formula with respect to some background theory. This generalises the boolean satisfiability (SAT) problem, which involves finding an assignment (or witness) to the variables of a boolean formula such that the formula evaluates to true. Similarly, SMT involves finding assignments (or witnesses) to the variables of a first-order formula ϕ such that the formula is true, but the variables need not be boolean. Instead, the variables may be integers, real numbers or other data types, as long as a background formal theory is defined for the data type. An SMT query over the reals can therefore be expressed as an existentially quantified formula of the form

$$\exists x \in \mathcal{X} \subset \mathbb{R}^n \colon \phi(x), \tag{2.8}$$

where ϕ is a first-order formula over the reals, and x is a vector of real variables that lies in the state space $\mathcal{X}$. It is worth noting that such formulae are often replaced (or used interchangeably) with the equisatisfiable formula without the existential quantifier. In this work we shall consider SMT queries over the real numbers which are nonlinear and may contain transcendental functions, commonly known as nonlinear real arithmetic with transcendental functions (NRA-T).

Solving SMT problems is the task of an SMT solver. An SMT solver is a program that takes as input a first-order formula and a background theory, and outputs either a satisfying assignment to the variables or that no such assignment exists. Crucially, as a decision procedure, SMT is *sound*. This means that if a satisfying assignment exists then it will be found. This enables the use of SMT solvers in verification tasks, where the goal is to prove that a given formula holds. To achieve this, the negation

of the formula, $\neg\phi(x)$ is given to an SMT-solver. A witness to the negation of the formula is a counter-example to the original formula $\phi(x)$. Since SMT is sound, if it returns that no witness to the negation exists, then no counter-example exists for the original formula, and thus ϕ must be true. In this way, SMT solvers are invaluable tools for algorithms such as CEGIS, and they make a common choice for the verifier component. Different SMT solvers support different background theories, and the choice of SMT solver depends on the specific verification task as hand. The SMT solvers used in this work are discussed in the relevant chapters.

Chapter 3

Formal Neural Abstractions of Dynamical Models

3.1 Introduction

Abstraction is a fundamental process used to advance understanding and knowledge. Constructing abstractions requires separating important details from those that are unimportant, in order to enable more manageable analysis or computation of that which was abstracted. This enables the drawing of conclusions that would otherwise be unobtainable (whether analytically or computationally). While the features and the properties of abstractions vary between fields and applications, in the context of the analysis and verification of mathematical models of dynamical systems abstractions take on a formal role: namely, abstractions must retain all behaviours of the concrete model that they abstract, while at the same time ought to be easier to analyse or perform computations on. These requirements allow for certain formal specifications, such as safety, to be formally provable over the abstractions and, additionally, for these very specifications to hold true for the concrete (original) dynamical model (which was the object of abstraction).

Dynamical models describe processes which are ubiquitous in science and engineering and whose complex behaviour is subject to safety-critical constraints. The verification of these behaviours is therefore crucial, and safety verification of nonlinear continuous-time dynamical models is a mature field. Analysis of such models can be performed using Taylor models [44, 45, 46, 47], simulations [68], reduction to first-order logic [106], Koopman linearisation [22], and, as in this work, via abstraction [9, 125, 16, 149, 151]. In this work we focus on the challenging setup of verification of nonlinear models with vector fields that may not be locally Lipschitz continuous, which violate the working assumptions of existing verification tools. We tackle this problem via abstraction, specifically abstraction via hybridisation.

Hybridisation is the abstraction of a (nonlinear) vector field to a hybrid automaton, and has been extensively utilised for the analysis of nonlinear models [17, 56, 57, 23, 89, 105, 35, 102, 110, 77, 14], even with dynamics as simple as piecewise constant ones [18, 134]. However, typically these approaches rely on a fixed partitioning of the state space (often based on a grid) that is chosen a priori, which can limit their scalability both with dimension and number of partitions. Hybridisation schemes commonly rely on error-based over-approximation to ensure soundness, though this idea is not exclusive to methodologies that partition the state space [11], and has been formalised by means of the notions of (approximate) bisimulation relations [132, 115, 133].

Properties which make an abstraction 'good' cannot be universally declared, and instead are deemed to be 'useful' in relation to given properties that are the object of analysis. Abstractions that are 'simpler' in structure may be easier to analyse, but might be coarser and thus lead to less refined proofs, or even no proof at all. In contrast, a more refined abstraction, i.e., one that contains fewer additional behaviours relative to the concrete model, might be complex and challenging to analyse. Indeed, for any given problem (a dynamical model and a property), it is unclear at the out-

set where the optimal middle ground between simplicity and precision lies, leading to abstraction techniques that begin with coarser ones and refine them iteratively [50, 147, 33, 66]. This is known as counterexample-guided abstraction refinement (CEGAR).

The trade-off between this relationship is the topic of this work: we study how abstractions constructed from neural networks of different nature and precision can be useful for the formal verification of safety properties of complex (e.g., nonlinear) dynamical models. By their very nature, neural networks are flexible and can be used to construct abstractions of different shapes and expressivities, ranging from abstractions with piecewise constant dynamics to abstractions with piecewise affine dynamics and to abstractions which are themselves nonlinear. These *neural abstractions* consist of a neural network interpreted as an ODE (a neural ODE) [43], alongside a formal (bounded) error between abstract and concrete models. This enables safety verification of models with challenging dynamical behaviours, such as non-linear dynamics including non locally-Lipschitz vector fields.

Piecewise constant and nonlinear (Lipschitz continuous) models are well studied model semantics with mature tools for their analyses, each with different advantages. In this work, we construct neural-based dynamical models which follow these semantics and cast them as formal abstractions, and show their potential use cases. For our piecewise affine abstractions, this cast depends on the crucial observation that a ReLU neural network can be interpreted as a linear hybrid automaton, and we describe how this cast is performed efficiently in practice.

While piecewise constant and piecewise affine are clearly simpler templates to abstract nonlinear ODEs, it is perhaps initially unclear how nonlinear abstractions can be considered useful. However, not all nonlinear models are alike in complexity: models with complex functional composition or with the presence of functions that are not Lipschitz continuous will be abstracted to models which are still nonlinear,

but free of these constraints. The potential of nonlinear neural abstractions is further bolstered by recent advances in developing literature on reachability analysis for neural ODEs [118, 82, 83]. Neural ODEs are leveraged in this work as part of our construction of neural abstractions.

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 or models based on support vector machines [138]. Consequently, there is much interest in verifying models with continuous-time dynamics and neural network controllers that take actions at discrete time steps, often referred to as neural network control systems [163, 92, 62, 95, 154]. Whilst in this work we focus on continuous-time models, we emphasise that verification of discrete-time models and of models with neural controllers is also studied [20, 162, 170].

3.1.1 Contributions

We summarise our main contributions in the following points:

- we introduce the novel idea of leveraging neural networks to represent abstractions in formal verification, and we utilise 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 constructed from three differ-

ent activation functions, and cast equivalent, analysable models (Section 3.4, Section 3.5) for use in safety verification;

- we implement our overall procedure[1] under different abstract models, investigating the quality of our synthesis procedure and the resulting abstractions with regards to precision, construction time, and computation efficiency of analysis;

- we demonstrate the ability of our approach to tackle challenging verification of models that do not exhibit local Lipschitz continuity, higher-dimensional models, and complex neural ODEs, which is significant in light of the growing field of research around neural ODEs (Section 3.6).

3.1.2 Outline

The remainder of this chapter is structured as follows. In Section 3.2, we introduce the necessary preliminary material for this work, including the notions of dynamical models (and their safety verification), hybrid automata and neural networks. In Section 3.3, we introduce the concept of neural abstractions for dynamical models and present the key theoretical results of this work. In Section 3.4, we present our synthesis procedure for neural abstractions based on CEGIS, and in Section 3.5 we describe how we cast these abstractions into equivalent models that are amenable to analysis by existing verification tools. In Section 3.6, we present the results of our experimental evaluation, we discuss threats to generality in Section 3.7, and in Section 3.8 we provide conclusions for this chapter.

[1]The code may be found at the following repositories: https://github.com/aleccedwards/neural-abstractions-nips22; https://github.com/aleccedwards/qest-na.

3.2 Preliminaries

In this work we introduce the concept of neural abstractions of dynamical models. We must begin by first defining the necessary preliminary material to do so, notably the concepts of dynamical models, linear hybrid automata and neural networks.

3.2.1 Dynamical Models and Safety Verification

This work studies continuous-time autonomous dynamical models which consist of coupled ODEs affected by an uncertain disturbance [152, 103]. Denote a disturbed dynamical model as

$$\mathcal{F} : \dot{\xi}(t) = f(\xi(t)) + d, \quad \|d\| \leq \delta, \quad \xi(t_0) = x_0 \in \mathcal{X}_I \subseteq \mathcal{X}. \tag{(2.3)}$$

where $f : \mathbb{R}^n \to \mathbb{R}^n$ is a continuous nonlinear vector field, and the state vector $\xi(t) = x$ lies within some bounded Euclidean domain of interest $\mathcal{X} \subset \mathbb{R}^n$, and where $\|\cdot\|$ denotes a norm operator (unless explicitly stated, we assume all norms are given the same semantics across the chapter).

More formally, we denote the dynamical model in Eq. (2.3) as $\mathcal{F}$. A trajectory of $\mathcal{F}$ defined over time horizon $T > t_0$ is a function $\xi\colon [t_0, T] \to \mathbb{R}^n$ that admits derivative at each point in $[t_0, T]$ such that, for all $t \in [t_0, T]$, it holds true that $\xi(t) \in \mathcal{X}$ and $\dot{\xi}(t) = f(\xi(t)) + d_t$ for some $\|d_t\| < \delta$. Since f is continuous, this is sufficient to guarantee existence of solutions to the corresponding coupled ODEs, due to the Cauchy-Peano theorem, [52, p. 6]. The disturbance d is a signal with range $\delta > 0$, and is interpreted as a nondeterministic disturbance, which at any time can take any possible value within the disturbance bound provided by δ. This concept has also been referred to as nondeterministic input [17, 72], but here we use the term disturbance to avoid confusion with control input.

We study the following formal verification question: whether a continuous-time,

autonomous dynamical model with possibly uncertain bounded disturbances considered within a region of interest is safe over a given time horizon with respect to a region of initial states. Let the sets $\mathcal{X}_I \subset \mathcal{X}$ be a region of initial states and $\mathcal{X}_U \subset \mathcal{X}$ be a region of unsafe states. We say that a trajectory ξ defined over time horizon T is initialised if $\xi(t_0) \in \mathcal{X}_I$; additionally, we say that it is safe if $\xi(t) \notin \mathcal{X}_U$ for all $t \in [t_0, T]$; dually, we say that it is unsafe if $\xi(t) \in \mathcal{X}_U$ for some $t \in [t_0, T]$. The safety verification question for dynamical model $\mathcal{F}$ 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 $\mathcal{X}_I$ and $\mathcal{X}_U$. If there exist at least one initialised trajectory that is unsafe, then we say that the model is unsafe.

We tackle safety verification by means of formal abstractions, followed by reachability analysis via flowpipe (i.e., reach sets) propagation. We construct an abstract dynamical system that captures all behaviours (trajectories, executions) of the original system (it *simulates* the concrete model). We then calculate (sound over-approximations of) the flowpipes of these abstractions: i.e., we find (sound over-approximations of) the reachable states from the initial set $\mathcal{X}_I$. If these states do not intersect with the unsafe states, we can conclude the abstraction is safe. Since we deal with over-approximation, if the abstract model is safe, then the concrete model is necessarily safe too [17]. 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. Since the final check involving set intersection is quite straightforward, the crux of the safety verification problem is the computation of first, the abstract model, and second, the reachable states via flow-pipe propagation - both steps will be the focus of our experimental benchmarks.

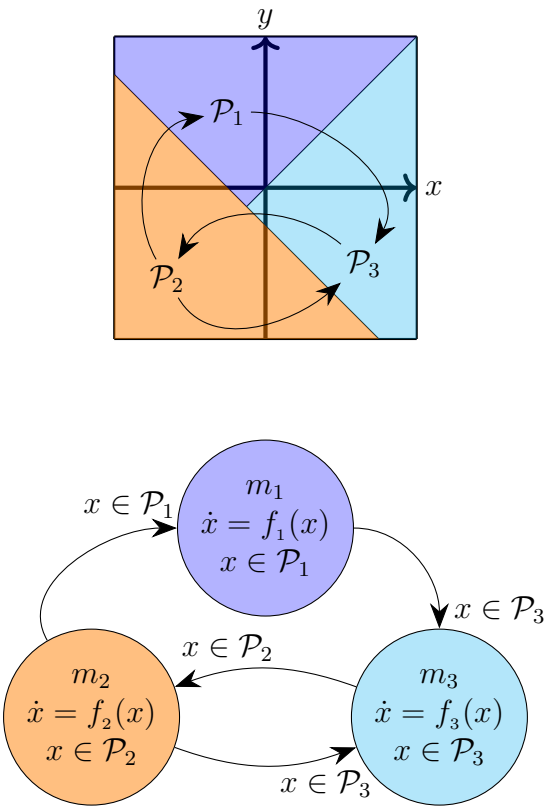


Figure 3.1: A hybrid automaton corresponding to a state-space partitioning. Each of the three discrete modes corresponds to a unique partition $\mathcal{P}_i$ and vector field $f_i(x)$. Discrete transitions are denoted by the edges of the directed graph with a transition between two modes if the corresponding partitions $\mathcal{P}_i$ and $\mathcal{P}_j$ are adjacent and a trajectory from f_i 'crosses' the corresponding partition.

3.2.2 Linear Hybrid Automata

Hybrid automata are useful models of systems comprising both continuous and discrete dynamics [87, 164]. They consist of a collection of (discrete) modes, each with their own dynamics over continuous states, rules for transitioning amongst discrete modes, and consequently to re-initialise the continuous dynamics into a new mode. Different semantics exist for hybrid automata; in this work we use the same semantics as SpaceEx [12, 73], a state-of-the-art tool used for analysis of linear hybrid automata.

The present work considers linear hybrid automata induced by a state space partitioning, in which the invariant, flow and guard conditions are described by

combinations of linear inequalities. Sometimes these are referred to as affine hybrid automata if these conditions are affine, but we will use the term linear hybrid automata. We define these formally as follows. Define a hybrid automaton $\mathcal{H} = (Mod, Var, Lab, Inv, Flow, Trans)$, consisting of a labelled graph encoding the evolution of a finite set of continuous variables $Var \subset \mathbb{R}^n$, in which the state variables x take their values. Each vertex $m \in Mod$ of the graph is referred to as a mode, and corresponds to a partition within the state space. The state of the automaton is thus given by the pair (m, x), whose evolution in time is described by $Flow(m, x)$. We describe this evolution as the flow, and in each mode the local flow given by $\dot{x} = f_m(x) + d$, where $f_m(x) : \mathbb{R}^n \to \mathbb{R}^n$ is affine in x and d is a nondeterministic disturbance such that $||d|| \leq \delta$. The edges of the graph describing $\mathcal{H}$ are known as transitions. A transition $(m, \alpha, Guard, m') \in Trans$ has label $\alpha \in Lab$ and allows the system state to jump from (m, x) to the successor state (m', x) instantaneously. We note that only the mode changes during a transition, with x being mapped via the identity function. Transitions are governed by $Guard \subset \mathbb{R}^n$, with a transition being enabled when $x \in Guard$. Moreover, the state may only remain in a mode m if the continuous state is within the invariant $Inv(m) \subset \mathbb{R}^n$. For the purposes of hybrid automata induced by state-space partitioning in this work, both $Guard$ and $Inv(m)$ are closed convex polyhedra $\mathcal{P} = \{x | \bigwedge_i \langle a_i, x \rangle \leq b_i\}$, where $a_i \in \mathbb{R}^n$ and $b_i \in \mathbb{R}$. We denote by $\mathcal{P}_m$ the polyhedron that defines the invariant for mode m.

We depict an example partitioning-induced hybrid automaton $\mathcal{H}$ in Fig. 3.1. The automaton consists of three modes, m_1, m_2, m_3, induced by the partitioning of the state space, which form the nodes of a graph. These modes have polyhedral invariants $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ respectively. The transitions between the modes are governed by the polyhedral guards, and if a transition exists then it is represented by an edge in the graph. The local flow in each mode is given by the vector field $f_i(x)$, which is affine in x.

3.2.3 Neural Networks

Denote a neural network $\mathcal{N}$ with input layer $z_0 \in \mathbb{R}^n$, corresponding to the dimension of the dynamical model in (2.3). This is followed by k hidden layers $z_1, \ldots, z_k$ with dimensions $h_1, \ldots, h_k$ respectively, and finally followed by an output layer $z_{k+1} \in \mathbb{R}^n$.

To each hidden or output layer with index i are associated matrices of weights $W_i \in \mathbb{R}^{h_i \times h_{i-1}}$ and a vectors of biases $b_i \in \mathbb{R}^{h_i}$.

Every hidden layer i is associated with an activation function $\sigma_i \colon \mathbb{R} \to \mathbb{R}$. The valuation of hidden layers z_i and output layer z_{k+1} is given by

$$z_i = \sigma_i(W_i \cdot z_{i-1} + b_i), \ i = 1, \ldots, k, \tag{(2.6)}$$

$$z_{k+1} = W_{k+1} \cdot z_k + b_{k+1}, \tag{(2.7)}$$

where each σ_i is applied element-wise to its h_i-dimensional argument.

We remark that the choice of activation functions clearly determines the features of a neural network: later we shall discuss three alternatives, and how these affect abstract models built on these templates.

3.3 Neural Abstractions of Dynamical Models

We are now ready to introduce the concept of neural abstractions of dynamical models. Our approach synthesises an abstract dynamical model defined in terms of a feed-forward neural network interpreted as an ODE endowed with a bounded nondeterministic disturbance. This can be seen as a neural ODE [43] augmented with an additive nondeterministic drift. We define this formally as follows.

Definition 3.1 (Neural Abstraction)**.** *Let $\mathcal{F}$ be a dynamical model given by continuous 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

$$\forall x \in \mathcal{X} \colon \|f(x) - \mathcal{N}(x)\| \leq \epsilon - \delta. \tag{3.1}$$

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:

$$\dot{\xi}(t) = \mathcal{N}(\xi(t)) + d, \quad \|d\| \leq \epsilon, \quad \xi(t_0) = x_0 \in \mathcal{X}_I \subseteq \mathcal{X}. \tag{3.2}$$

■

Next, we provide the key theoretical results of this work, in which we show first that a neural abstraction is a correct over-approximation of the concrete model. This result is a direct consequence of the definition of neural abstraction, and is therefore not surprising. However, it is crucial to note that this result does not make any assumptions of the local Lipschitz continuity of the vector field f.

Furthermore, we state an important consequence of the correctness of neural abstractions: they enable a *sound* safety verification procedure of the concrete model. If the concrete model is unsafe, then the neural abstraction will not be safe (or alternatively, if the neural abstraction is safe, then so is the concrete model). However, the converse does not hold: if the neural abstraction is unsafe, then the concrete model may still be safe. This is due to the nature of abstraction by over-approximation, which may introduce additional unsafe behaviours that are not present in the concrete model.

Theorem 3.2 (Correctness of Neural Abstractions)**.** *If $\mathcal{A}$ is a neural abstraction of a dynamical system $\mathcal{F}$ with continuous vector field f and disturbance radius $\delta \geq 0$, over a bounded region of interest $\mathcal{X} \subseteq \mathbb{R}^n$, then every trajectory of $\mathcal{F}$ is also a trajectory*

of $\mathcal{A}$. ■

Proof of Theorem 3.2. Let ξ be a trajectory of $\mathcal{F}$ and T be the time horizon over which ξ is defined. Then, let $t \in [t_0, T]$. By the definition of a 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 (3.1) 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), shows that ξ is a trajectory of $\mathcal{A}$. □

Corollary 3.3. *Let* $\mathcal{X}_I \subset \mathcal{X}$ *be a region of initial states and* $\mathcal{X}_U \subset \mathcal{X}$ *and region of unsafe states. It holds true that if* $\mathcal{A}$ *is safe with respect to* $\mathcal{X}_I$ *and* $\mathcal{X}_U$ *then also* $\mathcal{F}$ *is safe with respect to* $\mathcal{X}_I$ *and* $\mathcal{X}_U$. ■

Proof of Corollary 3.3. By Theorem 3.2, 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 3.4 (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 (3.1)) 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 [112, 28, 53, 74, 91, 109].* ■

Altogether, we define the neural abstraction $\mathcal{A}$ of a non-linear dynamical system $\mathcal{F}$ as a neural ODE with an additive disturbance, and note that it approximates the dynamics of $\mathcal{F}$, while also accounting for the approximation error. Notably, this approximation error is effectively 'tightened' by the disturbance radius δ of the concrete model.

We note again the assumptions placed on the vector field f. While f is assumed to be continuous, f is not required to be locally Lipschitz continuous. This is because the precision of a neural abstraction relies on the condition described by (3.1), which is certified by an SMT solver (see Section 3.4). As previously discussed, continuity of f is a sufficient condition for the existence of solutions to the nonlinear vectory field f, as per the Cauchy-Peano existence theorem [52, p. 6]. Not assuming local Lipschitz continuity of f means that we cannot guarantee *uniqueness* of solutions to the dynamical model; since our abstractions themselves overapproximate they also do not guarantee unique solutions either. Importantly, any trajectory of the concrete model is also a trajectory of the neural abstraction, and therefore the safety of the neural abstraction implies the safety of the concrete model. Later, we shall show how Theorem 3.2 may be proved using an SMT-solver, enabling the automated synthesis of correct neural abstractions.

3.3.1 Semantics of Neural Abstractions: Influence of Activation Functions

The choice of activation function σ for a neural abstraction is significant, as it determines the structure of the final abstraction, which in turn may impact its quality and utility. Furthermore, the choice of activation function determines the underlying semantics of the abstract model, which is a crucial factor when seeking to leverage techniques for safety verification, or otherwise analyse the abstract model. In this work, we consider three different activation functions, and describe the semantics of

the resulting neural abstractions. We note that this is not an exhaustive list, and that other activation functions could be considered in future work. Alternative activation functions, or even network architectures, could be considered in future work for abstractions of unexplored use cases.

3.3.1.1 Piecewise Affine Neural Abstractions

We first consider perhaps the most natural choice of activation function: the ReLU function. ReLU is widely used and studied in the machine learning literature, and results in piecewise affine functions, which are well studied in the formal verification literature. The ReLU function is defined as

$$\text{ReLU}(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{3.3}$$

Consider a neural network $\mathcal{N}$ consisting entirely of ReLU activation functions. Each neuron (i.e., each ReLU function in the network) induces a hyperplane in the corresponding vector space of its corresponding layer, resulting in two half-spaces: one of which has a linear output ('on') and the other zero output ('off'). For the jth neuron in the ith layer, this hyperplane is given by

$$W_{i,j} z_{i-1} + b_{i,j} = 0, \tag{3.4}$$

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.

Any fixed configuration of ReLU functions as 'on' or 'off' can be therefore be interpreted as the intersection of a finite number of half-spaces, which itself is a convex polyhedron. Later, we will show how this can be used to construct a linear hybrid automaton that is equivalent to the neural abstraction, by constructing the labelled graph for $\mathcal{H}$. This involves calculating the appropriate modes (vertices) —

which requires the computation of the convex polyhedral invariants and locally affine flows — and transitions (edges) — whose guards are equivalent to the invariant of the mode being transitioned to.

This interpretation of ReLU functions as half-spaces motivates our choice of ReLU as the activation function for our neural abstractions. We may interpret the piecewise affine function as partitioning the state space into a finite number of polyhedra, each of which is associated with a different affine vector field. There is no limitation on the shape of these polyhedra, and they may be arbitrarily complex as determined by the structure of the neural network in terms of width and depth. The number of polyhedra is determined by the number of neurons in the network. If the network has $H = h_1 + \cdots + h_k$ neurons, then the state space will be partitioned into at most 2^H polyhedra. In practice, the number of polyhedra will be significantly fewer than this, as not all polyhedra induced by the ReLU functions will intersect with the state space $\mathcal{X}$.

3.3.1.2 Piecewise Constant Neural Abstractions

Approximations of nonlinear vector fields need not be piecewise affine. Indeed, piecewise constant approximations are also possible, and can be useful in certain contexts. For instance, while such abstractions might be less precise (due to their less expressive nature), they are also less complex, and thus more amenable to analysis. We construct piecewise constant approximations using the unit-step function, also known as the Heaviside function. This function is defined as

$$\mathrm{H}(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{3.5}$$

It is only necessary for the final hidden layer to use piecewise constant activation functions to obtain a piecewise constant output, given that all other hidden layers

using a ReLU activation function. In other words, to obtain a piecewise constant template we set $\sigma_i(x) = \mathrm{ReLU}(x)$, $i = 1, \ldots, k-1$ and $\sigma_k = \mathrm{H}(x)$. A neural network with piecewise constant output, rather than a piecewise affine output, is also equivalent to a linear hybrid automaton, namely one with constant flows in each mode (rather than affine flows).

3.3.1.3 Nonlinear Neural Abstractions

Finally, we consider a case where the neural abstraction is itself nonlinear. Nonlinear neural abstractions will be seen as "regularised" alternatives of the original nonlinear vector fields. Moreover, by choosing an activation function which is Lipschitz continuous, we may abstract nonlinear vector fields which are not Lipschitz continuous to a model which is. For these nonlinear templates we utilise the Sigmoid function σ_{sig} – resulting in 'sigmoidal' nonlinear templates. We shall refer to neural abstractions constructed from networks of sigmoidal functions as nonlinear neural abstractions or sigmoidal neural abstractions. σ_{sig} serves as a smooth approximation to a step function,

$$\sigma_{\mathrm{sig}}(x) = \frac{1}{1 + \mathrm{e}^{-x}}. \tag{3.6}$$

We could also consider other activation functions, such as the tanh function, which is also a smooth approximation to a step function, and we expect it would yield similar results. However, we do not consider this in this work. Our choice of studying these kinds of nonlinear neural abstractions is motivated by growing literature on the reachability analysis of sigmoid-based neural networks [96, 49, 95].

3.4 Formal Synthesis of Neural Abstractions

In this section we discuss a framework for constructing neural abstractions of different expressivities using inductive training procedures coupled with symbolic reasoning.

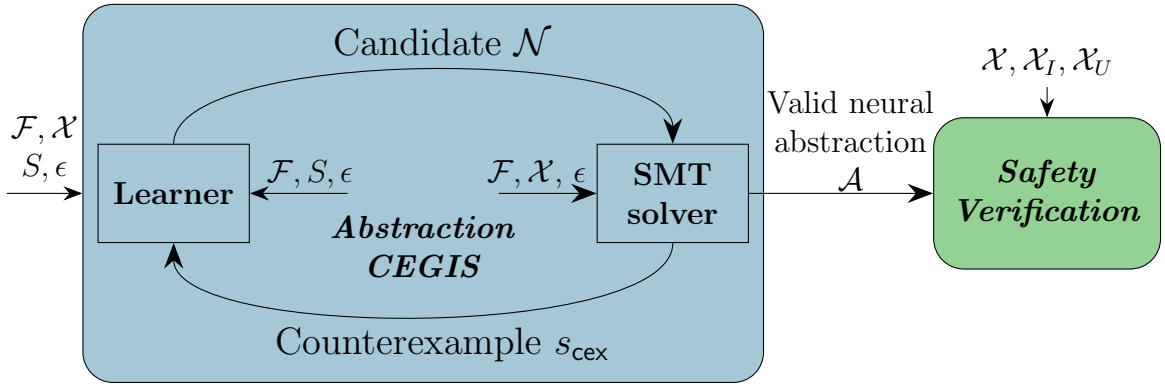


Figure 3.2: Architecture for the safety verification of nonlinear dynamical models using neural abstractions. The inputs to our architecture are a concrete model $\mathcal{F}$ and its domain of interest $\mathcal{X}$, a finite set of initial data points S sampled from the domain of interest $\mathcal{X}$, a desired approximation error ϵ, and regions of initial $\mathcal{X}_I$ and unsafe states $\mathcal{X}_U$.

We leverage an iterative synthesis procedure known as counterexample-guided inductive synthesis (CEGIS, [158]), which has been shown to be useful for inductively constructing functions that satisfy desired specifications.

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. Often the certifier component is known as a *verifier*, but in this chapter we use the term *certifier* to avoid confusion with the safety verification procedure. 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. We observe that in the case of an infinite search space, CEGIS is incomplete and is therefore not guaranteed to terminate. In practice, we show that our CEGIS loop terminates robustly and efficiently when applied to the synthesis of neural abstractions.

As previously described, the learner in this procedure is a neural network; the certifying 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 [76], which is advantageous in that it can be used to query satisfiability over non-polynomial formulae, including nonlinear functions with transcendental terms. This is necessary both for sigmoidal neural abstractions, and also for the abstraction of models containing trigonometric or exponential terms. We note that dReal is a δ-complete SMT-solver: while this guarantees dReal will always find a counterexample if one exists, it may return also spurious counterexamples within a δ-perturbation of the original formula [76]. This implies that our procedure may not terminate successfully, even when the candidate is valid.

We shall now present a methodology to synthesise (learn and certify) neural abstractions, considering each component of the CEGIS loop depicted in Fig. 3.2 in turn. In Section 3.4.1 we present the learning component and in Section 3.4.2 we present the SMT-solver based component used to certify the neural abstractions.

3.4.1 Procedures for Learning Candidate Neural Abstractions

We now present the learning component of the CEGIS loop, which is responsible for guessing a candidate neural network that satisfies the desired conditions. We detail two approaches for learning neural abstractions: one for piecewise affine and nonlinear templates (which lend themselves to gradient-based training), and one for piecewise constant templates (which do not).

For either approach, we sample a finite set of data points S from the domain of interest $\mathcal{X}$. The inputs to the learning procedure are therefore the concrete vector field f of the concrete dynamical model and the set of points S sampled uniformly from the domain of interest $\mathcal{X}$. For both gradient-based and gradient-free training, achieving a low error bound ϵ is the primary goal as it results in a more precise abstraction, though results in increased computation time for the learning procedure.

Gradient-based Training Training neural abstractions for the piecewise affine and the nonlinear templates can be done using gradient descent algorithms (for which we use PyTorch, [129]), which depend on the loss functions used. We seek to minimise the maximum error over the domain; however, this would result in a non-differentiable loss function, which is undesirable for gradient descent. Therefore, we train via a proxy: the mean squared error,

$$\mathcal{L}(s) = \frac{1}{|S|} \sum_{s \in S} \|f(s) - \mathcal{N}(s)\|_2. \tag{3.7}$$

Here $\|\cdot\|_2$ represents the l^2-norm of its input, and S represents a finite set of data points s 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 S.

Unlike regression tasks common in machine learning, we are not concerned about overfitting to the underlying model. In fact, we want to overfit to the model to ensure that the neural abstraction is as precise as possible. However, it is still undesirable to overfit to the data. Furthermore, a model with smaller values for the weights and biases that fits as well as a model with larger values is preferred, as it is a simpler abstraction. For regularisation, we rely on the built in weight decay of AdamW.

Gradient-free Training The step-function $\mathrm{H}(x)$ (cf. Eq. (3.5)) has zero gradient almost everywhere: this makes training with gradient descent-based approaches unsuitable. We therefore rely on gradient-free methods for training neural networks, which notably scale less well and converge less quickly. Several options exist as suitable choices: here we use a particle swarm optimisation approach, specifically with PySwarms [123].

Gradient-based training approaches benefit from differentiable loss functions as they ensure gradient calculations are always possible. However, in the case of neural

abstraction synthesis, this means we do not train based our true objective, but via a proxy. With a non-gradient based approach there are no longer advantages to a differentiable loss function, meaning we can minimise a loss function that better represents our desired specification (cf. Eq. (3.1)) based on the maximum error over samples. In line with this, we also utilise the l^∞-norm, namely

$$\mathcal{L}'(s) = \max_{s \in S} \|f(s) - \mathcal{N}(s)\|_\infty. \tag{3.8}$$

3.4.2 Certifying Candidate Neural Abstractions

Regardless of the template used to generate the abstraction, the procedure for certifying its quality is identical. The network $\mathcal{N}$ is translated to symbolic form and passed to the SMT-solver block, which checks condition (3.1) as described in Section 3.4.2. This involves using an SMT solver to check that at no point in the entire domain of interest $\mathcal{X}$ is the maximum error between the neural network and the concrete model greater than some given ϵ.

The upper bound ϵ on the abstraction error is first estimated empirically using the finite data set S by considering the maximum approximation error over S. We may make this estimated error more conservative by inflating it, to help ensure the check on the validity of the abstraction is successful. SMT problems are in general NP-hard and computationally expensive, so it can be beneficial to provide queries that are simpler to solve or are more likely to be successful. A less conservative estimate of the error bound may result in longer verification times, or even failure to verify the abstraction. Conversely, small error bounds will result in a more precise abstraction, but may be more difficult to verify.

Next, the SMT solver is asked to find an assignment cex of x to the negation of

the specification in Eq. (3.1). This is given by the following formula:

$$\phi = \exists x \in \mathcal{X} : \ \|f(x) - \mathcal{N}(x)\| > \epsilon - \delta. \tag{3.9}$$

If any such assignment s_{cex} is found then the error bound ϵ does not hold over the entire domain $\mathcal{X}$, and a valid abstraction has not been constructed. Instead, the assignment is treated as a counterexample and is added to the data set S, and further training of the network continues by returning to the *learning* phase of the CEGIS loop. The counterexample is augmented by sampling for additional points nearby in order to maximise the efficiency of learning and the overall synthesis. In (3.9), the δ term is the disturbance radius of the concrete model, incorporating any existing disturbances in the concrete model.

On the other hand, if no assignment is found, then the specification in Eq. (3.1) is valid and the synthesis is complete, returning a sound neural abstraction. We note that feed-forward neural networks with σ_{sig} activation functions are simply elementary functions, and networks with ReLU or step-function activations can be interpreted as linear combinations of formulae in first-order logic. These insights enable the encoding of such networks in SMT-formulae by means of symbolic evaluation.

3.5 Safety Verification using Neural Abstractions

Neural abstractions are dynamical models expressed in terms of neural ODEs with additive disturbances (cf. Eq. (3.2)). Corollary 3.3 ensures that performing safety verification using a neural abstractions amounts to verifying a neural ODE with additive bounded disturbance. 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 3.4), 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 [170, 92, 62, 162, 163, 95, 20, 154], yet with a subtle difference: neural network controllers take control actions at discrete points in time. Instead, neural ODEs characterise dynamics over continuous time.

The bottleneck of this safety verification problem is the computation of the reachable states via flow-pipe propagation, which is in general hard for non-linear dynamics. Recent literature has demonstrated advancements in performing reachability analysis on neural ODEs [82, 83, 118]. Neural abstractions are a new concept consisting of neural ODEs with bounded additive disturbances, and therefore no bespoke methods exist for their reachability analysis. We expect to see developing techniques to enable this analysis as tools such as GoTube [83] continue to advance, and neural abstractions mature as a technology. In the meantime, we may instead rely on constructing equivalent models that are amenable to existing methods. By equivalent, we mean that the trajectories generated by both models are identical. In particular, we leverage hybrid automata as equivalent models,

For piecewise affine neural abstractions, we seek to construct a hybrid automaton with affine dynamics and invariants defined by polyhedra. We shall detail this cast and how it can be performed efficiently in the sequel. The obtained models can be analysed more efficiently using the space-time clustering approximation (STC) algorithm [72], which is implemented in the verification tool SpaceEx [73].

Meanwhile, for piecewise constant models we note the similarity between $\mathrm{H}(x)$ and $\mathrm{ReLU}(x)$, neural abstractions with piecewise constant activation functions also induce a linear hybrid automaton, except that now the flows are given by a constant term. Verification of such models is the specialty of the tool PHAVer [70], which itself implements a bespoke version of the symbolic model checking algorithm introduced by [13] and [88].

Finally, we consider nonlinear neural abstractions with σ_{sig} activation functions. We observe that the output of these networks can simply be interpreted as 'regularised' nonlinear dynamical models consisting of elementary functions, hence these abstractions are nonlinear ODEs with bounded additive nondeterminism. Casting a neural ODE as a nonlinear ODE involves evaluating the network symbolically. Reachability analysis of these kinds of models can be performed using the mature tool Flow* [45], which performs flowpipe (i.e., reach sets) propagation of nonlinear models using Taylor approximations [44], and hence is dependent on local Lipschitz continuity. This means that by constructing a neural abstraction with nonlinear templates of a concrete model that is not locally Lipschitz continuous, we enable safety verification (via Flow*) of that otherwise intractable concrete model.

We shall now present the procedure for constructing the equivalent abstract models for each of the three templates considered in this work: piecewise affine, piecewise constant, and nonlinear.

3.5.1 Casting Piecewise Affine Neural Abstractions as 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, described in Eq. (3.4) results in two half-spaces, one corresponding to the neuron being active (linear) and one to it being inactive (zero). Therefore, every combinatorial configuration of the neural network defines an intersection of half-spaces 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 piecewise affine neural abstraction may be cast 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. Fig. 3.1).

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 of neurons in each of them (cf. Section 3.2). Each c_i represents the configuration of the neurons at the ith hidden later, and the jth element of c_i represents the activation status of the jth neuron at the ith layer, which equals to 1 if the neuron is active and 0 if it is inactive. Every mode of the hybrid automaton corresponds to exactly one configuration of neurons.

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 $\mathcal{X}$. We will leverage this fact to define the invariant condition of the mode m.

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

$$\mathcal{Z}_m = \left\{ (z_0, \ldots, z_k) \;\middle|\; \begin{array}{c} \wedge_{i=1}^{k} z_i = \mathrm{diag}(c_i)(W_i z_{i-1} + b_i) \wedge \\ \mathrm{diag}(2c_i - 1)(W_i z_{i-1} + b_i) \geq 0 \end{array} \right\}. \tag{3.10}$$

Note that $\mathrm{diag}(v)$ 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 $\mathcal{Z}_m$ onto the input neurons z_0, de-

noted $\mathcal{Z}_m \restriction_{z_0}$. Since the input neurons z_0 are equivalent to the state variables of the dynamical model, the invariant condition of mode m results in

$$\mathcal{P}_m = (\mathcal{Z}_m \restriction_{z_0}) \cap \mathcal{X}. \tag{3.11}$$

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 half-spaces backwards along the network, similarly to methods used in abstraction-refinement [34, 71]. This alternative construction is outlined in [6] (extended version, Appendix C).

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

$$\dot{x} = A_m x + b_m + d, \quad \|d\| \leq \epsilon, \quad x \in \mathcal{P}_m. \tag{3.12}$$

The matrix $A_m \in \mathbb{R}^{n \times n}$ and the vector of drifts $b_m \in \mathbb{R}^n$ 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:

$$A_m = W_{k+1} \textstyle\prod_{i=1}^{k} \operatorname{diag}(c_i) W_i, \tag{3.13}$$

$$b_m = b_{k+1} + \textstyle\sum_{i=1}^{k} (W_{k+1} \prod_{j=i+1}^{k} \operatorname{diag}(c_j) W_j) \operatorname{diag}(c_i) b_i. \tag{3.14}$$

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. We find that taking a more precise approach, by considering the Lie derivative, does not significantly improve the performance of the resulting verification procedure and is not worth the additional computational cost of computing the Lie derivative.

The guard condition of a discrete transition is simply the invariant of the destination mode, described by the linear inequalities that define the polyhedral invariant.

3.5.1.1 Enumeration of Feasible Modes

A given configuration C exists in the hybrid automaton if and only if the corresponding set $\mathcal{P}_m$, which is a convex polyhedron in $\mathbb{R}^n$, is non-empty; 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 two approaches that work well in practice to determine all valid neuron configurations.

3.5.1.2 Optimisation Approach

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 half-space intersected with the domain of interest $\mathcal{X}$. If only one LP is feasible, we can fix the neuron to this mode, i.e., from this point onward only consider the intersection with the half-space 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. As a depth-first tree search for an implicit binary graph to depth H, this approach has a time complexity of $O(2^H)$. In practice we find that we can prune many branches early, and the number of linear programs solved is significantly fewer than 2^H.

We illustrate this tree in Figure 3.3, which depicts an example search for a neural network with a single hidden layer consisting of three neurons. The tree illustrates the construction of $\mathcal{P}_m$ 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.

This approach is inspired by that presented in [25], which similarly enumerates through the path of neurons using sets to determine the output range of a network.

3.5.1.3 SMT-based Approach

The second approach relies on the observation given any point in the state space, it is easy to evaluate the configuration C of the corresponding polyhedron $\mathcal{P}_m$ it lies in. Furthermore, any given configuration C and corresponding polyhedron $\mathcal{P}_m$

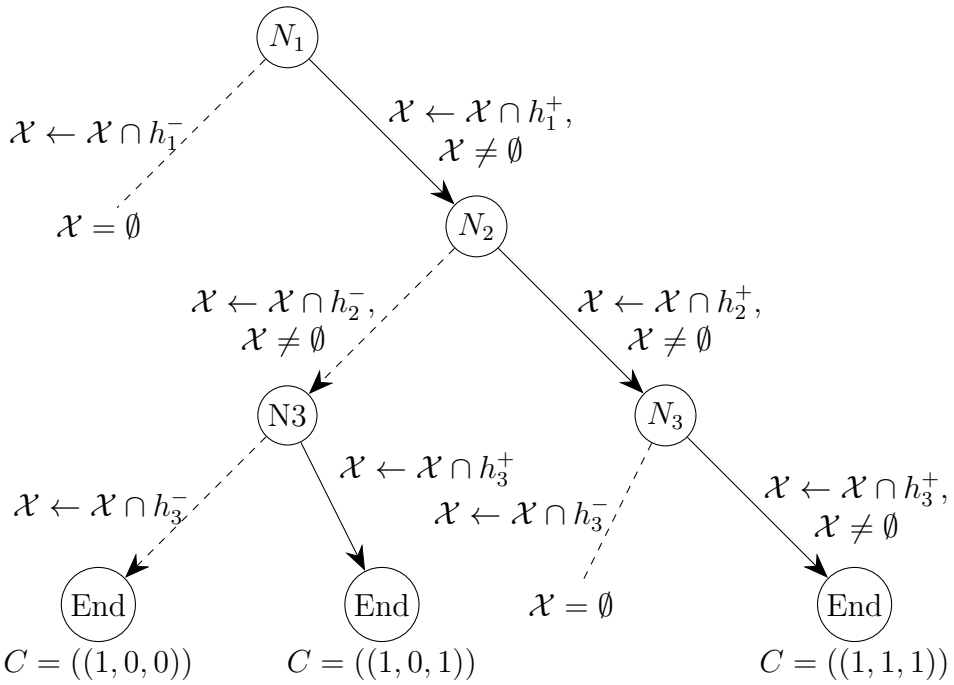


Figure 3.3: Example Tree search to determine the active configurations for a neural network consisting of a single hidden layer with 3 neurons. Here, h_i^+ denotes the positive half-space $\{x : w_i x + b_i \geq 0\}$ and h_i^- denotes the negative half-space $\{x : w_i x + b_i \leq 0\}$ of the i^{th} neuron; w_i represents the i^{th} row of the weight matrix corresponding to the hidden layer, and b_i represents the i^{th} 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.

is a convex polyhedron in $\mathbb{R}^n$, and therefore can be represented by a set of linear inequalities. Therefore, the problem of finding a new point that does not belong to any known configuration may be expressed as an existentially quantified linear formula. We may therefore pose this query to an SMT solver that can reason over linear real arithmetic to find a given configuration. However, our task is to exhaustively find all active configurations, and next we shall detail a procedure to do so. This procedure is inspired by AllSat [120, 84], a known approach within SMT literature to find all satisfying assignments to a formula.

The algorithm takes as input a feed forward ReLU network $\mathcal{N}$ (specifically its weight matrices W and biases b) and a domain of interest $\mathcal{X}$. It initialises an empty list of active configurations. As described previously, a configuration C is a nested list of 1s and 0s corresponding to the network layers, depending on whether a given neuron is enabled or disabled in that configuration. First, we ask an SMT solver (which acts as an oracle) for a point in the state space. Then, we calculate the configuration this corresponds to (depending on whether the neuron evaluation from Eq. (3.4) is positive or negative for each neuron), and add this to the list of active configurations. We ask the SMT solver for another point with a new constraint: that it does not lie in the polyhedron corresponding to any known active configurations. This amounts to taking the conjunction of the previous formula provided to the SMT solver with the linear conditions defining the newly found polyhedron. This step repeats until the SMT solver returns `unsat`, at which point all active configurations have been found. We provide pseudocode for this procedure in Algorithm 1.

3.5.2 Casting Piecewise Constant Neural Abstractions as Hybrid Automata

We recall that by construction, a piecewise constant neural abstraction is similar in structure to that of a piecewise affine network: they consist of a fully connected feed-

Algorithm 1 Pseudocode for SMT-based approach to enumerate all active configurations.

Require: $\mathcal{N}$ (List of neurons), $\mathcal{X}$ (Domain of interest)
Ensure: $configs$ (List of active configurations)

$configs \leftarrow \emptyset$
$S \leftarrow$ InitializeSolver()
S.addConstraint($\mathcal{X}$)
function CalculateConfiguration($point, \mathcal{N}$)
    $C \leftarrow \emptyset$
    for $i = 1$ to $length(\mathcal{N})$ **do** ▷ For each neuron in the network
        $point \leftarrow$ Evaluate($\mathcal{N}[i], point$) ▷ Using corresponding row of W_i and b_i
        $C \leftarrow C \cup \{point > 0\}$ ▷ $W_{i,j}x + b > 0$ (1 if true, 0 if false)
    return C
while S.isSat() **do**
    $p \leftarrow S$.getModel()
    $c \leftarrow$ CalculateConfiguration($p, \mathcal{N}$)
    $configs \leftarrow configs \cup \{c\}$
    S.addConstraint($\neg c$)
    return $configs$

forward neural network, using ReLU functions within the hidden layers. However, the final hidden layer in a piecewise constant neural abstraction consists of unit-step functions H. The input to this function is still a linear combination of the previous layers evaluation, but now the output is simply 1. Since the final output of the network, given by Eq. (2.7), is a linear combination of the final hidden layer, the output of a piecewise constant neural abstraction is a piecewise constant function.

Due to the similar structure of piecewise constant neural abstractions to piecewise affine, we observe that these abstractions are also equivalent to a linear hybrid automaton. Indeed, the evaluation of the final layer $k + 1$ does not appear when calculating the discrete modes, invariant conditions, transitions, or guards for the piecewise affine case. Therefore, calculating these for a piecewise constant neural abstraction is equivalent. The key difference is when calculating the flow conditions, which we shall now discuss.

Flow Conditions The flow describing the dynamics of each model may now be seen as constant dynamics with bounded disturbance:

$$\dot{x} = b_m + d, \quad \|d\| \leq \epsilon, \quad x \in \mathcal{X}_P. \tag{3.15}$$

As before, $b_m \in \mathbb{R}^n$, is now only dependent on the output layer:

$$b_m = W_{k+1}\mathbf{vec}(c_k) + b_{k+1}, \tag{3.16}$$

where $\mathbf{vec}(\cdot)$ denotes the vector which takes its values from its input. This follows from observing that, from the definition of H, the output of the penultimate layer z_k is equal to c_k.

3.5.3 Casting Nonlinear Neural Abstractions as Nonlinear ODEs with Bounded Disturbance

At the time of writing, no tool exists for the reachability analysis of neural ODEs with bounded additive disturbance. However, tools such as Flow* are able to perform reachability analysis of nonlinear ODEs with bounded additive disturbance. We recall that a nonlinear neural abstraction is a neural ODE with bounded additive nondeterministic disturbance. The output of a neural network with σ_{sig} activation functions is simply a nonlinear (elementary) function of its input. Therefore, a neural abstraction with σ_{sig} activation functions is equivalent to a dynamical model as defined by Eq. (2.3).

In order to perform safety verification on these models using Flow*, the network must be evaluated symbolically, and the resulting nonlinear ODE must be expressed in terms of the state variables of the concrete model. This is simple to do, given that, once the abstraction is constructed, we know the weight matrices and bias vectors that define the network, as well as the activation function of each hidden layer.

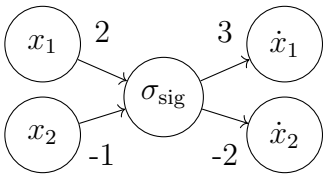


Figure 3.4: An example sigmoidal neural network, without bias. The output of the network is a nonlinear function of the input, which can be interpreted as a nonlinear dynamical model.

We demonstrate this in practice with a small example. Consider the small network shown in Fig. 3.4, which consists of a network with two input neurons, two output neurons and a single hidden layer of one neuron. For simplicity, we omit the bias terms. The output of this network is given by

$$\begin{cases} \dot{x}_1 = 3\sigma_{\text{sig}}(2x_1 - x_2), \\ \dot{x}_2 = -2\sigma_{\text{sig}}(2x_1 - x_2), \end{cases} \tag{3.17}$$

where σ_{sig} denotes the sigmoid function, as in Eq. (3.6).

We note that this casting is not exclusive to neural abstractions with σ_{sig} activation functions; networks with any (elementary) activation function may be cast as a (elementary) nonlinear dynamical model. Furthermore, unlike the piecewise affine and piecewise constant cases, this cast does not result in a model that is 'simpler' than the original neural abstraction, only one that is amenable to verification using Flow*.

3.5.4 Refining the Precision of the Abstraction

Here, we detail a procedure that enables the refinement of the abstraction error bound ϵ for a given piecewise affine or piecewise constant neural abstraction that has been cast to an equivalent hybrid automaton. Recall that the abstraction error given by

Eq. (3.9) is global, that is, it holds across the entire domain $\mathcal{X}$. In reality, the true abstraction error is likely smaller than ϵ throughout much of the domain.

As demonstrated, piecewise constant and -affine neural networks induce a hybrid automaton over $\mathcal{X}$; this automaton is realised as part of the proposed framework, after a valid abstraction has been synthesised. Since this automaton induces a polyhedral partitioning of the state-space, we can consider these polyhedra in turn to determine a local abstraction error for each mode. Consider a given mode m, with flow f_m and polyhedral invariant $\mathcal{P}_m$. As described in Section 3.2.2, the local flow f_m describes the evolution of x while it remains within the invariant given by $\mathcal{P}_m$. We can therefore rewrite Eq. (3.9) for each mode m as

$$\phi_m = \exists x \in \mathcal{P}_m : \ \|f(x) - f(x)_m\| > \epsilon_m - \delta, \tag{3.18}$$

where $\epsilon_m \leq \epsilon$ is a candidate upper bound on the abstraction error estimated empirically over the set $S_m \coloneqq \{s \in S | s \in \mathcal{P}_m\}$.

We provide ϕ_m to an SMT solver as before: if no satisfying assignment is found then the candidate error bound holds for the mode. Note that this does not affect the overall correctness of our abstractions: if a satisfying assignment is found, the original upper bound on abstraction error still holds. We will later present results on the efficacy of this procedure.

3.5.5 Summary of the Safety Verification Procedure

So far we have defined the concept of a *neural abstraction* with regards to a dynamical model, and presented a methodology for their formal synthesis. As a demonstrative use-case of these mathematical objects, we embed them into a framework for safety verification of the concrete models. In addition to synthesis, this involves the construction of equivalent abstract models for each of the three templates presented in

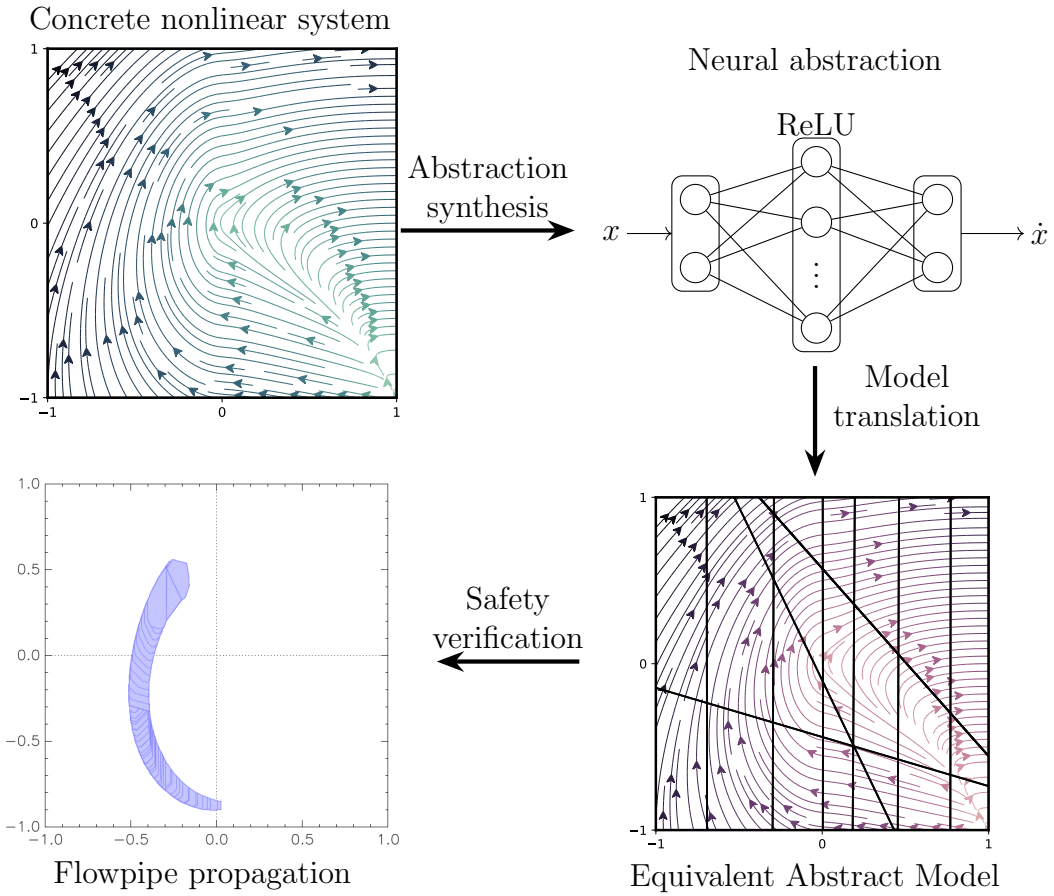


Figure 3.5: Overview of our workflow for safety verification on a non-Lipschitz dynamical model (cf. Section 3.6, NL2). The concrete dynamics are abstracted by a neural ODE with ReLU activation functions and a certified upper-bound on the approximation error. This characterises a polyhedral partitioning and defines a hybrid automaton with affine dynamics and additive nondeterministic drift. Flowpipe propagation is finally performed through a region of non-Lipschitz continuity.

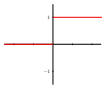
	Piecewise constant	**Piecewise affine**	**Nonlinear**
Concrete models	Nonlinear, non-Lipschitz	Nonlinear, non-Lipschitz	Nonlinear, non-Lipschitz
Activation function	 H	 ReLU	 Sigmoid
Training procedure	Particle swarm	Gradient descent	Gradient descent
Loss function	$\max_{s \in S} l^{\infty}(s)$	$\frac{1}{\lvert S \rvert} \sum_{s \in S} l^{2}(s)$	$\frac{1}{\lvert S \rvert} \sum_{s \in S} l^{2}(s)$
Equivalent abstract model	LHA I with disturbance	LHA II with disturbance	Nonlinear ODE with disturbance
Safety verification technology	Symbolic model checking	STC algorithm	Taylor model based flowpipe propagation
Safety verification tool	PHAVer	SpaceEx	Flow*

Figure 3.6: Overview of neural abstraction templates presented in this work, alongside details of their synthesis procedure, interpretation, and corresponding state-of-the-art verification technologies. Here, LHA I refers to *linear hybrid automata* with polyhedral guards and invariants and constant flows, whereas LHA II refers to the same object but now with affine flows.

this work. These equivalent models are themselves well-studied mathematical objects that are amenable to safety verification via flowpipe propagation, i.e., overapproximation of the reachable states of the system. We summarise this framework in Fig. 3.5, using piecewise-affine based neural abstraction as an example.

Each template, i.e., piecewise affine, piecewise constant, and nonlinear, relies on a different activation function, and hence induces a different type of abstract model. In turn, these different model semantics depend on different verification technologies and corresponding tools. We summarise these relationships in Fig. 3.6.

3.6 Experimental Evaluation

We now present a thorough experimental evaluation studying neural abstractions, our proposed methodology for their synthesis, and the safety verification of nonlinear dynamical models using neural abstractions. The main results of this work in Section 3.6.1 compare the performance of the three different templates presented in this work, comparing the salient features of the constructed abstractions (in terms of error and number of modes) and the computation performance of the corresponding flowpipe propagation.

We also present three supplementary sets of results, two of which act as a baseline comparison for one of the abstraction templates, piecewise affine. These results are as follows:

- Section 3.6.2 compares the performance of neural abstractions to affine abstractions based on a simplicial partitioning of the domain of interest in terms of abstraction error and the number of modes required to achieve a given error bound.

- Section 3.6.3 considers several safety verification benchmarks and compares the

performance of safety verification involving neural abstraction to the state-of-the-art tool Flow* [45].

- Section 3.6.4 evaluates the efficacy of the error refinement scheme presented in Section 3.5.4.

3.6.1 Performance of Different Neural Abstraction Templates

The experiments presented in this section investigate the trade-off between efficiency and precision of neural abstractions. We consider a number of benchmarks and study how the expressivity of neural abstractions, which depends on their activation functions, varies across their already-demonstrated niche, namely models that are not locally Lipschitz continuous. We provide the first examination of an additional context for which neural abstraction is suitable: that of abstracting neural ODEs.

3.6.1.1 Non-Lipschitz Models

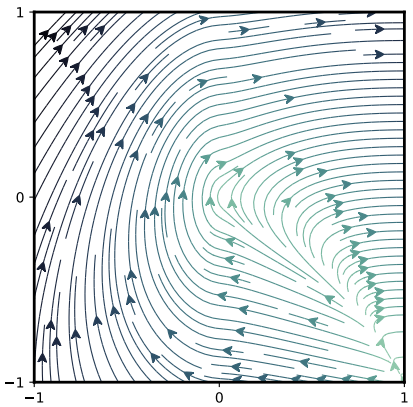
The first set of benchmarks we consider *do not* exhibit local Lipschitz continuity, meaning they violate the working assumptions of state of the art safety verification tools such as Flow* and GoTube [83], and are in general challenging to analyse computationally. Three of the benchmarks are taken from [6], and are shown in the first three rows of Table 3.1. We include two higher dimensional benchmarks, *Water Tank 4D* and *Water Tank 6D*, to demonstrate the ability of our approach to scale to higher-dimensional models with up to six continuous variables. These two additional models are also not locally-Lipschitz continuous. The equations of the dynamics of these models are in Appendix A.1.

For each benchmark, we synthesise an abstraction using one of the proposed templates: piecewise constant, piecewise affine and sigmoidal, and perform flowpipe (i.e., reach sets) propagation over a 1-second time horizon using the appropriate tool[2]. We

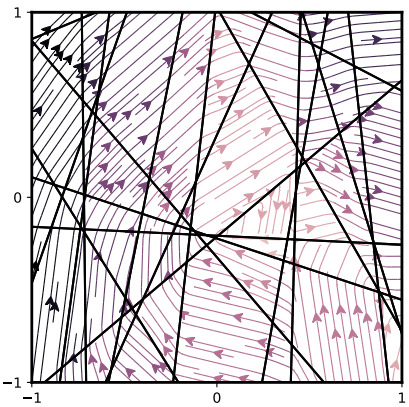
[2]For the *Water Tank 6D*, we only use the piecewise affine template, as this abstraction performs

Table 3.1: Table showing the properties of different abstractions with different templates over a series of benchmarks, and the total time spent for synthesis and flowpipe propagation. Here, 'PWC' denotes piecewise constant, 'PWA' denotes piecewise affine, 'Sig.' denotes nonlinear sigmoidal; W: network architecture (number of neurons per layer); ϵ: error bound, $||.||_1$ denotes the 1-norm; M: number of modes in abstraction; T: total computation time; μ denotes average over 10 repeated runs.

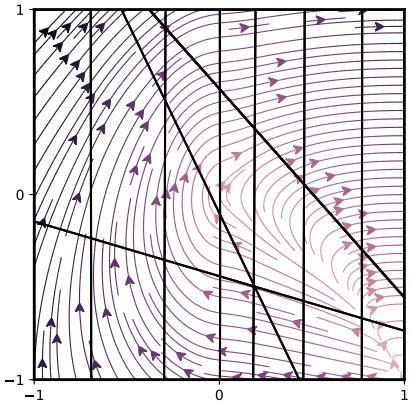
			$\|\|\epsilon\|\|_1$			M			T		
			min	μ	max	min	μ	max	min	μ	max
Benchmark	**Template**	W									
Water Tank	**PWC**	[**15**]	0.16	0.22	0.27	4	5.30	6	6.32	7.32	7.91
	PWA	[**12**]	0.08	0.09	0.10	6	7.00	8	14.55	70.37	390.90
	Sig.	[**4**]	0.07	0.07	0.07	1	1.00	1	15.53	18.31	21.53
Non-Lipschitz1	**PWC**	[**20**]	0.97	1.21	1.47	14	26.50	45	13.84	16.07	20.56
	PWA	[**10**]	0.10	0.12	0.14	6	14.00	20	24.58	51.94	92.40
	Sig.	[**4**]	0.05	0.08	0.10	1	1.00	1	76.24	97.20	105.60
Non-Lipschitz2	**PWC**	[**24**]	1.58	1.97	2.25	55	73.40	94	25.81	32.97	41.99
	PWA	[**12 10**]	0.11	0.12	0.13	11	23.80	62	55.94	86.47	163.16
	Sig.	[**10**]	0.06	0.08	0.10	1	1.00	1	309.82	368.36	451.04
Water Tank 4D	**PWC**	[**25**]	2.47	2.57	2.69	1	1.50	4	173.09	232.33	400.26
	PWA	[**12**]	0.80	0.80	0.80	4	8.10	18	9.14	21.49	50.70
	Sig.	[**7**]	0.35	0.35	0.35	1	1.00	1	100.08	133.00	317.80
Water Tank 6D	**PWA**	[**16**]	1.30	1.30	1.30	11	16.00	25	76.86	426.16	1659.32
NODE1	**PWC**	[**20**]	1.59	1.83	2.12	33	63.10	86	36.23	70.03	104.95
	PWA	[**5**]	0.20	0.20	0.20	4	5.80	7	17.20	17.92	18.65
	Sig.	[**3**]	0.20	0.20	0.20	1	1.00	1	120.38	126.51	137.63



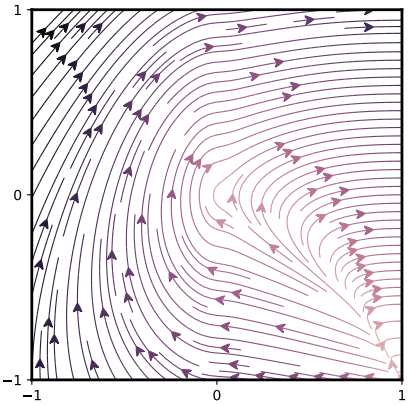
(a) Locally non-Lipschitz, nonlinear concrete dynamical model.



(b) Neural PWC abstraction with associated polyhedral partitioning.

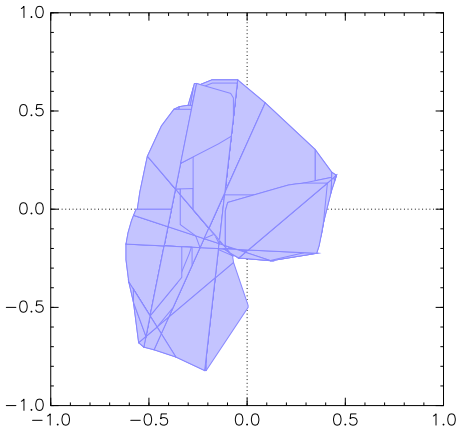


(c) Neural PWA abstraction with associated polyhedral partitioning.

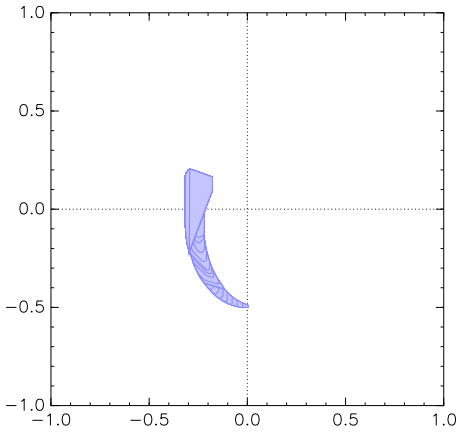


(d) Neural nonlinear (sigmoidal) abstraction.

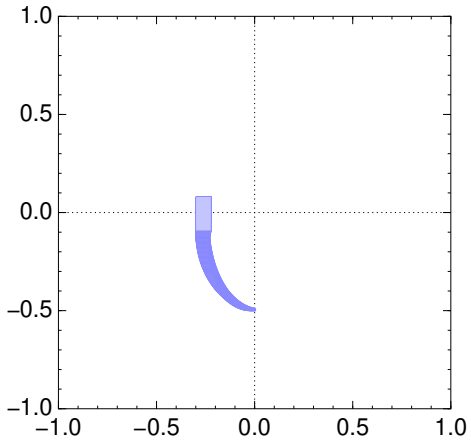
Figure 3.7: Visualisation of neural abstractions (underlying flow and relevant partitioning) from piecewise constant (PWC, b), piecewise affine (PWA, c) and sigmoidal (d) templates of a concrete model (a) that does not exhibit local Lipschitz continuity (Non-Lipschitz 2 model).



(a) Flowpipe propagation of neural PWC model.



(b) Flowpipe propagation of neural PWA model.



(c) Flowpipe propagation of neural sigmoidal model.

Figure 3.8: Flowpipe (i.e., reach sets) propagation for the Non-Lipschitz 2 model using neural abstraction templates and corresponding verification tools. Flowpipe propagation is performed over the same initial set over the same time horizon (1 second).

Table 3.2: Breakdown of the total computation time shown in Table 3.1. Here, learning time T_L; T_C: certifying time; T_f: time to compute flowpipe (i.e., reach sets) over-approximation over horizon of 1 sec, using corresponding verification tool (cf. Fig. 3.6).

		T_L			T_C			T_f		
Benchmark	**Template**	min	μ	max	min	μ	max	min	μ	max
Water-tank	**PWC**	6.16	7.20	7.78	0.02	0.03	0.03	0.01	0.01	0.02
	PWA	14.43	70.18	390.64	0.02	0.05	0.08	0.05	0.08	0.12
	Sig.	10.99	13.90	16.79	0.00	0.01	0.02	3.90	4.40	5.13
Non-Lipschitz1	**PWC**	12.94	14.42	16.91	0.19	0.44	0.95	0.16	0.74	3.22
	PWA	22.65	43.64	79.17	0.56	7.21	20.23	0.10	0.99	4.97
	Sig.	19.47	26.63	36.88	0.61	1.60	5.17	46.01	68.97	83.91
Non-Lipschitz2	**PWC**	11.25	13.75	15.22	1.27	1.86	2.58	8.51	15.97	22.27
	PWA	28.48	54.20	136.79	12.93	23.03	38.76	0.45	4.41	8.49
	Sig.	30.73	65.35	127.13	12.76	37.17	58.26	257.82	265.84	275.55
Water-tank-4d	**PWC**	172.06	215.69	306.95	0.38	0.58	0.94	0.20	15.84	92.13
	PWA	6.37	12.63	21.94	2.07	8.61	40.64	0.09	0.13	0.17
	Sig.	42.67	46.00	51.54	25.98	57.09	245.52	29.56	29.91	30.92
Water-tank-6d	**PWA**	38.92	101.84	360.93	16.64	323.45	1619.26	0.18	0.26	0.34
NODE1	**PWC**	6.04	7.61	8.26	28.97	60.21	95.12	0.11	1.64	7.15
	PWA	3.12	4.24	4.89	12.67	13.60	14.49	0.03	0.04	0.04
	Sig.	18.38	26.35	33.16	20.82	23.33	26.79	72.75	76.83	82.47

leverage our procedure's dependence on random seeding and its low computational cost: for each experiment we initialise multiple runs (namely, four) in parallel. When the first of these returns successfully, we discard the remaining and record the result. For statistical and reproducibility reasons, we repeat each experiment 10 times, and present the mean (μ), max and min over these runs.

We present the salient features of the abstractions in Table 3.1. In particular, we present the architecture of the neural network (the number of hidden layers and neurons within them); the size of the resulting abstraction in terms of the number of modes M (nonlinear abstractions consist of a single mode) and the 1-norm of the error bound ϵ – a vector representing the error bound in each dimension – for each abstraction. We note that the ϵ presented is the *global* upper bound to the abstraction error, and does not account for any error refinement achieved by the procedure

best in the similar but smaller *Water Tank 4D* experiments.

detailed in Section 3.5.4. In practice the mode-wise error is often much lower than the global upper bound (see Section 3.6.4 for results on this), which enables us to study higher dimensional models and employ the less expressive piecewise constant templates successfully. The notable exception to this is for *Water Tank 4D*, for which piecewise constant abstractions regularly consist of a very small number of modes, despite a relatively high error: this indicates that the gradient-free learning procedure performs less well for this higher dimensional model, and suggests that neural-based piecewise constant templates are more suitable to smaller dimensional models.

Piecewise constant abstractions perform least well in terms of achieved error bound and in general require larger networks (cf. column W in Table 3.1) and more modes (col. M) in the resulting hybrid automaton: this is unsurprising, particularly when compared to the higher-order piecewise affine abstractions. Meanwhile, piecewise affine and sigmoidal templates perform more comparably to each other, though sigmoidal templates do achieve slightly better error bounds while using significantly smaller neural networks.

We also present a breakdown on the time spent within each stage of synthesis –learning (T_L) and certifying (T_C) – in Table 3.2. We prioritise learning speed for the piecewise constant abstractions rather than comprehensive learning, seeking to ensure learning times that are comparable to the other two templates. This is also because we expect safety verification for piecewise constant dynamics to be quite fast. This means that better errors might be achievable for piecewise constant abstractions, however with significantly greater learning times.

We do not perform explicit safety verification using abstract models here, as selecting regions of unsafe states is arbitrary given that all the abstractions depend on an error bound. Instead, as anticipated earlier, we perform flowpipe (i.e., reach sets) propagation for each abstraction over a time horizon of 1 second, and present the obtained computation time (T_f) in Table 3.2. This column highlights the afore-

mentioned trade-off between abstraction templates: flowpipes can be calculated for piecewise constant abstractions (in general) significantly faster than for the other two in lower dimensions. However, for higher-dimensional or more complex models, the flowpipe computation is much slower: this is due to the abstraction error (ϵ) being greater, making flowpipe computation more difficult. Meanwhile, despite the improved accuracy, the sigmoidal abstractions require more computation for flowpipe propagation due to the increase in model complexity (non-linearity).

The flowpipe propagation is run with a 500 second timeout, and may not terminate successfully - e.g., Flow* is sometimes unable to compute the whole flowpipe. Across all experiments (including repetitions), the flowpipe propagation is unsuccessful *only three times*: twice for the sigmoidal templates for *Non-Lipschitz 2*, and once for the piecewise constant templates for *WaterTank 4D*. We also set an overall timeout of 1800 seconds on the whole procedure (including abstraction synthesis and flowpipe propagation). *Only four* experiments failed to complete before this timeout: once for the piecewise constant template for *Water Tank 4D*, and three times for the *Water Tank 6D* experiments. We emphasise that these outcomes highlight the robustness and 'practical completeness' of the end-to-end procedure. The numerical outcomes shown in the tables exclude those few experiments that time out.

The abstractions for the *Non-Lipschitz 2* model are illustrated in Fig. 3.7, with the corresponding polyhedral partitioning (for piecewise constant and -affine abstractions) and obtained (locally) approximated vector fields. The results of the corresponding flowpipe (i.e., reach sets) propagation are then depicted in Fig. 3.8. This figure illustrates further the sorts of safety verification tasks that can be completed using each abstraction type: challenging verification tasks, requiring enhanced precision (ϵ), should be attempted using piecewise affine or nonlinear templates, whereas simpler tasks might be formally verified more efficiently through safe piecewise constant neural templates.

3.6.1.2 Abstraction of Neural ODEs

The results presented in Table 3.1 include a model *(NODE1)* that is Lipschitz continuous, but whose dynamics are described by a neural network. These kinds of models are known as neural ODEs (NODEs), and have become a widely studied tool across machine learning and verification [43]. Here, we discuss abstracting neural ODEs as an additional use-case for neural abstractions.

Neural ODEs are commonly trained to approximate real physical models based on sample data. However, it is likely that these models are over-parameterised, making their reachability analysis difficult. For the experiment, we train a neural ODE consisting of a feed-forward network with three hidden layers of hyperbolic tangent activations, based on data from an underlying two-dimensional model. Then, as with the previous experiments, we construct abstractions of this neural ODE with three templates and perform flowpipe propagation. The longest computation time is for the sigmoidal template at 138 s, and the best performing average computation time is for piecewise affine abstractions at 17.9s. In contrast, we provide the concrete neural ODE to Flow* with the same settings, which takes 205 s. It is clear that neural abstractions can be beneficial in improving computational efficiency for flowpipe propagation of neural ODEs at the cost of the error bound ϵ. We consider this to be an alternative approach to reducing the Taylor model order with Flow* for improving computational efficiency, but instead of a one-time analysis, it results in a model which is permanently easier to analyse.

Finally, we note that we do not use state-of-the-art tools for neural ODE reachability analysis here, such as GoTube [83] or NNV [118]. These tools cannot perform reachability analysis on neural abstractions, as they do not account for the nondeterminism introduced by the abstraction error. Thus, in order to make a fair comparison we use Flow*, since to the best of our knowledge, no tool specialised to neural ODEs can also handle nonlinear neural abstractions directly. Our results are promising in

light of research interest in neural ODEs, and warrant further interest in the reachability analysis of neural abstractions.

3.6.2 Comparison to Affine Abstractions based on a Simplicial Partitioning

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 [18], which describes a type of hybridisation-based abstractions that are similar to those constructed in this work. The approach relies first on partitioning the state space using a simplicial mesh grid, and 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 [18].

In Table 3.3 we compare between abstractions constructed using an affine simplicial mesh and piecewise affine neural abstractions for three dynamical models with bounded second order derivatives (see Appendix A.1). These abstractions both have the same semantics (i.e., first order approximations), and may therefore be compared in terms of the approximation error and the number of partitions.

We run the procedure described in Section 3.4 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 3.3 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 our 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.

3.6.3 Safety Verification Comparison to Flow*

In this section, we present some supplementary empirical results in which piecewise affine neural abstractions are benchmarked against state-of-the-art safety verification tool Flow*. We focus on piecewise affine abstractions as they are the most explainable and best performing abstraction template in the main results presented in Section 3.6.1. We consider 6 dynamical models within a full (bounded time) safety verification framework (including computation of intersection with unsafe states). Three of these models do not exhibit local Lipschitz continuity, and are therefore impossible for Flow* to analyse. The remaining three models are Lipschitz continuous.

Table 3.3: 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

We benchmark the results obtained by the safety verification algorithm proposed in Section 3.5 against Flow* [45] (available under GPL), which, as previously discussed, is a mature tool for computing reachable regions of dynamical models. 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 (2.3), 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 [48, 24]. Here, we ask them to

perform safety verification for a given pair of initial and unsafe states, on a collection of different nonlinear models. As described in Section 3.3, the task of safety verification consists of ensuring that no trajectory starting within the set of initial states enters the set of unsafe states, within a given time horizon. Choosing the initial and unsafe states is arbitrary, so we are also interested in the time taken to perform the verification step and whether the tool is able to complete the verification within a given time limit (without the error 'blowing up').

Our setup is as follows. Firstly, for a given benchmark model we define a finite time horizon T, a region of initial states $\mathcal{X}_I$ and a region of unsafe states $\mathcal{X}_U$. Then, we run flowpipe computations with Flow* using high-order Taylor models. Similarly we run the procedure described in Section 3.4, and construct a hybrid automaton as described in Section 3.5 to perform flowpipe computations using SpaceEx. We present the results in Table 3.4. 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 required for high precision.

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 [26] 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

Table 3.4: 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*: blow-up in the error before T is reached, and -: no results obtainable.

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**

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.

Notice that we additionally outperform Flow* on a Lipschitz-continuous model (*Exponential* in Table 3.4), 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., 1^{st} order models), neural abstractions are able to compete with, and even outperform, methods that use much higher order functions (10^{th} and 30^{th} in the benchmarks) for approximation.

3.6.4 Evaluation of Error Refinement Scheme

Finally, we present results investigating the effect of the error refinement scheme proposed in Section 3.5.4, over a single benchmark (*Non-Lipschitz 2*) for 10 repeated runs and no parallelism (i.e., each run consists of only a single attempt) with and without the error-refinement scheme. We consider first a piecewise affine template, using a network architecture of two hidden layers with 14 and 12 neurons respectively, and a piecewise constant template with a single hidden layer of 20 neurons. These

Table 3.5: Breakdown of the timings shown in Table 3.4. Shown are the timings in the constituent component shown in Figure 3.2: time spent during learning, time spent during certifying 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

results are shown in Table 3.6, where we report the *mean* (μ) certifying time $\bar{T}_C$ and flowpipe propagation time $\bar{T}_f$. We consider any time spent in SMT-solving to be certifying time. The time spent in the learner, T_L is not reported as this is not impacted by the refinement scheme.

It is clear that for both templates, the error refinement provides a significant decrease in flowpipe propagation time, at the cost of a minor increase in certifying time. In addition to this, it significantly increases the usability of piecewise affine abstractions for SpaceEx with twice as many successfully terminating. We note that runs are considered unsuccessful if they do not terminate successfully from SpaceEx, or within a 500 second timeout, as in previous experiments. Finally, we present the average error bound over all modes in an abstraction, averaged over all successful runs.

From the reported outcomes, it is clear that the error refinement is able to significantly refine the error for modes in the abstraction, resulting in a more precise abstractions and thus more accurate flowpipe propagation. This error refinement technique is promising, as it can enable the approach to scale to higher-dimensional and more complex models more easily.

Table 3.6: The impact of error refinement on certifying time T_C, flowpipe propagation time T_f, and mean abstraction error over modes $||\epsilon||_1$, as well as the effect on success rate (out of 10 repeats) of flowpipe propagation for the Non-Lipschitz 2 benchmark for piecewise affine (pwa) and piecewise constant models. The results shown for T_C, T_f and $||\epsilon||_1$ are averaged over all successful runs, which we denote using bar notation (rather than μ as above).

Template	**Error Refinement**	$\bar{T}_C$	$\bar{T}_f$	$\|\|\bar{\epsilon}\|\|_1$	Success Rate
PWC	**Without**	0.92	16.45	1.89	1.00
	With	1.27	7.87	1.20	1.00
PWA	**Without**	14.54	54.17	0.14	0.30
	With	16.46	39.67	0.07	0.60

3.7 Limitations

3.7.1 Zeno Behaviour

The interaction between the discrete and continuous dynamics of hybrid automata leads to a phenomenon known as Zeno behaviour, characterised by an infinite number of discrete transitions in a finite amount of time. This is a well known pathological behaviour of hybrid systems modelling; often, approaches to hybrid automata verification exclude this behaviour for easy of analysis. Some Zeno behaviour is possible to exclude under assumptions of Lipschitz continuity.

As with similar approaches which construct a mesh-based partitioning of the state space [18], we assume that our abstractions do not exhibit Zeno behaviour. This is a reasonable assumption in practice, but we cannot exclude the possibility of Zeno behaviour in our piecewise affine and piecewise constant abstractions. In practice, it may be possible to reduce, or even eliminate, Zeno behaviour through regularisation during the learning process via a penalty term in the loss function. However, this is an open question for future work.

Finally, we note that the existence of Zeno behaviour is only problematic if it impacts our ability to analyse the model, and hence depends on the verification technology used. For example, SpaceEx is able to enclose some Zeno behaviour, but

not all.

3.7.2 Issues of Scale

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 at least NP-hard (and in the presence of exponential and trigonometric functions, undecidable) - which can struggle with complex formulae with many variables. The SMT-based certifying step requires the largest amount of time (cf. Table 3.2), indicating that improvements in the verification of neural networks can lead to a large performance increase for our abstractions. We discuss the second limitation in the next section.

3.7.3 Abstraction Complexity

We are limited in terms of the complexity of our abstractions by SpaceEx. While SpaceEx is a highly efficient implementation of LGG [108], 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 [10, 32, 155, 26].

3.8 Conclusions

This chapter introduces the novel concept of using neural networks not just as an approximation of dynamical models, but as formal abstractions of dynamical models. We have proposed a novel technique that leverages the approximation capabilities of neural networks with different activation functions to synthesise these formal neural

abstractions of dynamical models. Depending on the choice of activation function, these abstractions may be cast as models with different, but well studied, semantics – namely piecewise constant, piecewise affine and nonlinear (sigmoidal) dynamics.

By combining machine learning and SMT solving algorithms in a CEGIS loop, we present a workflow for the construction of these abstractions, including templating, learning and certifying the abstraction. Our abstractions consist of neural ODEs with additive nondeterminism that overapproximate the concrete nonlinear models. This guarantees the following property, in which safety of the abstract model carries over to the concrete model. In order to perform safety verification effectively, we propose procedures which cast neural abstractions to well studied mathematical objects, namely linear hybrid automata and nonlinear ODEs. These objects are amenable to verification using existing tools, such as SpaceEx, Phaver and Flow*. Using these existing tools, we show the advantages of abstractions of different semantics with regards to their precision and ease to analyse, which indicates their suitability for different verification tasks. The purpose of this is not to advocate for a single abstraction template, but to highlight the trade-offs between different templates, and to highlight the utility of neural abstractions as a flexible tool for safety verification.

As expected, we find that piecewise constant neural abstractions, while the fastest to analyse, are the least precise, and suitable for simpler verification tasks. Conversely, nonlinear neural abstractions are the most precise, but are the most difficult to analyse. Piecewise affine neural abstractions are a good compromise between the two, and are suitable for a wide range of verification tasks and study.

We demonstrate that piecewise affine abstractions constructed from neural networks are more accurate than affine abstractions based on a simplicial partitioning of the domain, for similar numbers of partitions. In doing so, we demonstrate that by reducing the problems of how to partition a state space and how to approximate the dynamics in each partition to a single problem of learning parameters of a neural

network, we are able to construct abstractions that are more accurate and require fewer partitions. We have demonstrated 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.

Finally, we remark that our experiments are limited to low-dimensional models and scalability remains an open challenge. This 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. However, our results are promising in light of growing interest in neural ODEs; future work in this area should consider extending tools specialised in reachability of neural ODEs [82, 83, 117] to neural abstractions by incorporating the appropriate nondeterminism, enabling study of higher-dimensional models and comparison to suitable baselines (such as Flow* and techniques in optimal control). Obtaining scalability to higher dimensions will also require a synergy of efficient SMT solvers for neural networks and safety verification of neural ODEs with bounded nondeterminism, which are both novel and actively researched questions in formal verification [101, 93, 111, 163].

Chapter 4

Certificate-based Verification of Dynamical Models using Neural Networks

4.1 Introduction

The analysis of the behaviour of continuous-time dynamical systems focuses on a wide range of properties, which are themselves suitable for an even wider range of applications. Arguably the most common property is (asymptotic) stability, which considers the (asymptotic) convergence of trajectories to an equilibrium point [152]. Stability is a fundamental property for dynamical systems, and is therefore of great interest to the formal verification community. Furthermore, (unbounded time) safety properties, namely the avoidance of an unsafe region of the state space [31], and their dual, reachability, that is the hitting of a target region of the state space in finite time [86], are also of great interest.

As we shall see, with the combination and the slight modification of these basic properties (we shall alternatively denote as specifications or requirements), an engi-

neer can in practice design a broad range of desired dynamical behaviours for any model at hand.

Given a model of a dynamical system, a spectrum of properties can be investigated from different perspectives and with diverse approaches: either analytical (e.g., via local linearisation and eigenvalues computation) [152], or computational ones (e.g., via dynamic flow propagation or via reach-set computation). In general, non-linearity in dynamical models is difficult to deal with: on account of this, approaches that are *indirect* or *sufficient* can be successful: a proof that the system actually fulfils a given requirement can be offered in the form of a *certificate*: the onus is thus to find, or to synthesise, a real-valued function defined over the state space with proper characteristics. A celebrated instance of indirect methods is the synthesis of Lyapunov functions [113], whereby, historically, one ought to hand-craft a function, often based on intuition and on physical properties of the underlying dynamical model. These functions act as "energy functions" for the underlying model, and show that the energy of the model necessarily decreases over time to a minimum value, which is the equilibrium point of the model. This in turn implies asymptotic stability of the equilibrium point. However, handcrafting Lyapunov functions is a non-trivial task, and often requires ingenuity and understanding of the model at hand.

In recent years numerical optimisation methods have automated the synthesis of Lyapunov functions, employing *templates*, i.e. candidate functions where only the coefficients (parameters) ought to be determined. Commonly, the choice falls onto polynomial templates framed as sum-of-squares (SOS) convex problems [127, 128, 79], which are known to admit globally optimal solutions. Indeed, the success of sum-of-squares methods has led to the development SOS techniques to synthesise barrier certificates for certifying safety properties [135, 136] However, these techniques operate solely on models with polynomial dynamics and various convexity assumptions. Alternative formulations include linear programs [30, 150, 141], and semi-algebraic

systems [157, 156], all of which leverage structural requirements on the dynamical models at hand.

Despite the usefulness of the mentioned synthesis approaches, they are often numerically sensitive and generally unsound, and thus undesirable for robust solutions and safety-critical applications [2, 36, 104]. Consequently, in recent years interest has grown in approaches for synthesis that can yield *provably-correct* certificates for a range of properties, much in the same line of research as *correct-by-design* control synthesis [160, 29]. A powerful technique to reason formally about correctness involves SMT-solving [27]. SAT-modulo-theory (SMT) extends satisfiability (SAT) solving, namely finding satisfying binary assignments to variables that make Boolean functions evaluate to true, to richer theories, such as linear or non-linear arithmetic over real numbers.

SMT has been used for synthesis tasks such as formal control synthesis, [94], but is commonly used as part of a counter-example guided inductive synthesis (CEGIS) [158] framework. This framework involves guessing a candidate solution using induction-based techniques, and then checking the validity of the candidate using an SMT solver. Approaches based on CEGIS have also been utilised for the synthesis of controllers [1] and certificates for dynamical models. Such techniques have been used first for certifying stability of dynamical models using polynomial Lyapunov functions and later extended to more general reach-avoid requirements [8], [98], [142], [143].

Related to SMT-based solutions, approaches that formulate synthesis problems as mixed-integer linear programs (MILP) have also been used to synthesise provably-correct Lyapunov functions [54] [55] for stability analysis, encompassing linear matrix inequalities for uncertain systems [119], and barrier certificates for safety [172], [41, 42]. Notably, mixed-integer problems also encompass candidates in the form of neural networks with ReLU activation functions, and may employ an optimisation engine like Gurobi [85] to certify the soundness of the proposed functions [173].

4.1.1 Related Work

The flexibility of neural networks has permeated throughout fields, including their use for synthesis of Lyapunov-like functions. For instance, early usage of neural networks for Lyapunov functions may be found in [145, 126], and later [97]. These describe generally unsound procedures for gradient descent-based training of a Lyapunov neural network (LNN).

In response to a need for soundness guarantees in safety-critical applications, [98] leverages counter-example guided inductive synthesis (CEGIS) to synthesise Lyapunov functions and barrier certificates for nonlinear continuous-time switched-mode systems. These certificates are verified using SMT-solvers, namely Z3 [59] and dReal [76]. Similarly, synthesis of certificates and controllers for reach-avoid properties in switched systems are studied in [143, 142].

A recent wave of research has seen the use of neural networks as part of the CEGIS framework for the sound synthesis of Lyapunov functions [8, 40, 4, 148, 80, 81] and for barrier certificates in e.g. [130, 171, 139]. The growth of certificate learning for Lyapunov and barrier certificates has seen the development of the software tool [4] and survey work [58] in this area. Within the CEGIS framework for certificate synthesis, the choice of SMT solver depends on the models under consideration and desired certificate template. Z3 [59] handles polynomial functions, dReal [76] and iSat3 [153] enable analysis of non-polynomial functions as well as polynomials.

Meanwhile, more complex reach-avoid-stay properties have continued to receive attention, though from a polynomial perspective. In particular [167, 166] where bespoke genetic algorithms are leveraged to generate *reach-while-stay* and *reach-and-stay-while-stay* functions, alongside controllers, for hybrid systems. Meanwhile reach-avoid-stay properties and corresponding certificates have been well studied from a theoretical perspective [122, 121]. In this chapter, we collate certificates for these more complex properties, and categorise them in order to unify them within simpler

certificates for stability, reachability and safety. We also generalise the concept of reach-avoid-stay property by allowing the *stay* set to be different from the *reach* set.

4.1.2 Contributions

The contributions of this chapter are summarised as follows:

- we collate and add to existing certificates from across the literature for nonlinear continuous-time dynamical models;
- we taxonomically categorise the properties these certificates certify into a simplified and general framework, which we newly describe through notions *arrive, avoid* and *remain*;
- we describe a unified algorithm to synthesise controllers to guide dynamical models to satisfy a specification, concurrently with certificates to prove they satisfy these specifications;
- we implement our framework[1] on top of the computational library Fossil [3], offering a new prototype software tool that can verify general *arrive, avoid* and *remain* properties for nonlinear control models using controllers and certificates based on neural networks.

4.1.3 Organisation

This chapter is organised as follows. Section 4.2 provides relevant background information and notation used across this work. In Section 4.3, we describe a range of properties for dynamical models and certificates which prove that they hold. Next, we describe a unified and computationally correct algorithm for synthesising these certificates in Section 4.4. We present experimental results for a prototype tool to

[1] Available at https://github.com/oxford-oxcav/fossil.

synthesise these certificates in Section 4.5, before discussing the limitations of our framework in Section 4.6 and providing concluding remarks in Section 4.7.

4.2 Preliminaries

4.2.1 Dynamical Models

We denote the set of positive real numbers as $\mathbb{R}_+$, and its extended version as $\overline{\mathbb{R}}_+ = \mathbb{R}_+ \cup \{+\infty\}$. A function is said to be of class $\mathcal{C}^1$ if its first derivative exists and is continuous. Given a scalar valued function V, we denote its β level set as $\Omega_\beta(V) = \{x \in \mathcal{X} \mid V(x) = \beta\}$, and its β sublevel set as $\Omega_\beta^-(V) = \{x \in \mathcal{X} \mid V(x) \leq \beta\}$. Let us consider models described by

$$\dot{\xi}(t) = f(\xi(t), u(t)), \quad \xi(t_0) = x_0 \in \mathcal{X}_I \subseteq \mathcal{X}, \tag{(2.4)}$$

where $\xi(t) = x \in \mathcal{X} \subseteq \mathbb{R}^n$ is the state of the system which takes value in the state space $\mathcal{X}$ and $f : \mathcal{X} \times \mathcal{U} \to \mathbb{R}^n$ is a Lipschitz-continuous vector field describing the model dynamics. We refer to these models as control models, and emphasise that in this chapter, we assume f to be Lipschitz continuous. We denote a trajectory over a time horizon $T \in \overline{\mathbb{R}}_+$ as $\xi(t) : [t_0, T] \to \mathbb{R}^n$, where $\xi(t)$ admits a time derivative everywhere, and such that $\dot{\xi}(t) = f(\xi(t), u(t))$ and $\xi(t) \in \mathcal{X}$, namely the trajectory is a solution of the model in (2.4). Finally, $u(t) \in \mathcal{U} \subseteq \mathbb{R}^m$ is the input and $\mathcal{X}_I$ denotes the set of initial conditions. Once a state-feedback controller $u(t) = k(\xi(t))$ has been specified, we may interpret the dynamics described by (2.4) as those of a closed-loop model, as follows

$$\dot{\xi}(t) = f(\xi(t)), \quad \xi(t_0) = x_0 \in \mathcal{X}_I \subseteq \mathcal{X}. \tag{(2.5)}$$

We refer to models of this kind simply as autonomous dynamical models. As before, we assume f to be Lipschitz continuous. This ensures the existence and uniqueness

of solutions, as given by the Picard-Lindelöf theorem [52, p. 12]. We denote with x^* an equilibrium point of (2.5), namely where $f(x^*) = 0$.

4.2.2 Systems and Properties

The goal of this work is to find a feedback controller, namely a signal $k(x) = u(t)$ in time, such that the dynamics above satisfy some desired temporal requirements (e.g., safety), or to show that given closed-loop (autonomous) dynamics are endowed with some given property (e.g., asymptotic stability).

We interpret properties of dynamical models in (2.5) in terms of their trajectories, and of binary relations that these trajectories have with given sets within the state space $\mathcal{X}$. To this end, we now introduce the notations and the semantics of the sets characterising properties of dynamical models. We denote by $\mathcal{X}_U$ an unsafe set, indicating a region of the state space where the system's trajectories should avoid; notably, it is sometimes simpler to define a set $\mathcal{X}_S$ of safe states, and consider $\mathcal{X}_U = \mathcal{X} \setminus \mathcal{X}_S$. $\mathcal{X}_G$ represents a goal set, indicating the region that the system's trajectories should enter; and $\mathcal{X}_F$ represents a final set, indicating a set where the system's trajectories should remain for all times after arriving at the goal set. These sets will be employed in the next section for the definition of properties, as depicted in Fig. 4.1. Implicitly, we consider that the unsafe and final sets are disjoint, i.e. $\mathcal{X}_U \cap \mathcal{X}_F = \emptyset$ and that the goal set is contained within the final set, i.e. $\mathcal{X}_G \subset \mathcal{X}_F$. Often we will assume sets to be compact; the motivation for assuming sets to be compact (rather than closed) is mainly to ease automated synthesis and verification. Additional topological properties of sets will be clarified later, within formal statements. Given a set S in a domain $\mathcal{X}$, we denote by $S^{\complement}$ its complement, i.e. $\mathcal{X} \setminus S$, and by $\text{int}(S)$ its interior, namely the set without its border, i.e. $\text{int}(S) = S \setminus \partial S$.

We consider a set S to be (forward) *invariant* if at some initial time t_0, $\xi(t_0) \in S$ implies that for all $t > t_0$, $\xi(t) \in S$ [31]. We consider a set S_A to be attracting [31]

with *region of attraction* S_B if for some initial time t_0 and any initial state $\xi(t_0) \in S_B$, the trajectory $\xi(t)$ converges to S_A as $t \to \infty$, i.e. if $\lim_{t\to\infty} dist(\xi(t), S_A) = 0$.

4.2.3 Neural Networks

Denote a neural network $\mathcal{N}$ with input layer $z_0 \in \mathbb{R}^n$, corresponding to the dimension of the dynamical model in (2.4). This is followed by k hidden layers $z_1, \ldots, z_k$ with dimensions $h_1, \ldots, h_k$ respectively, and finally followed by an output layer $z_{k+1} \in \mathbb{R}^d$. In this chapter, $d \in \{1, m\}$, where $d = 1$ corresponds to a scalar-valued certificate, whereas $d = m$ is used for a state-feedback controller with m control variables.

To each hidden or output layer with index i are associated matrices of weights $W_i \in \mathbb{R}^{h_i \times h_{i-1}}$ and vectors of biases $b_i \in \mathbb{R}^{h_i}$ [114]. Every hidden layer i is associated with an activation function $\sigma_i \colon \mathbb{R} \to \mathbb{R}$. The valuation of output and hidden layers is given by

$$z_i = \sigma_i(W_i \cdot z_{i-1} + b_i), \ i = 1, \ldots, k, \tag{(2.6)}$$

$$z_{k+1} = W_{k+1} \cdot z_k + b_{k+1}, \tag{(2.7)}$$

where each σ_i is applied element-wise to its h_i-dimensional argument.

4.3 Properties and Certificates

We present a number of properties (or requirements) for dynamical models defined over their trajectories, alongside definitions of corresponding certificates, whose existence serve as *sufficient conditions* for the satisfaction of the desired properties. We focus on continuous-time models; while the presented properties may be seamlessly applied to discrete time models, the corresponding certificates would not necessarily align with the presentation of the work, hence we omit their discussion as outside

the scope of this work. Several properties (and corresponding certificates) are ubiquitous across the control theory literature, whilst others have been more recently introduced or inherited from analogues in formal verification. Further, we include a practical variant on Lyapunov functions to allow for "constrained" local stability and an original certificate called Reach-Avoid-Remain, and suggest that more can be obtained in a modular, composable fashion. We emphasise that while many of these properties are similar to each other, they are subtly, and importantly, different and distinguished. Later in the section we provide a summary and clarification on their distinguishing features, which may also be seen in Fig. 4.1.

4.3.1 Stability

Stability is the most-studied property of dynamical models, and many definitions of this property exist. Stability is most commonly characterised in a Lyapunov (asymptotic) sense [152], namely in terms of the distance of a trajectory from the equilibrium point. Conversely, in order to achieve a consistent characterisation of properties in terms of set containment, here we characterise stability as follows:

$$\exists \mathcal{X}_I : \forall \xi(t_0) \in \mathcal{X}_I, \exists T \in \overline{\mathbb{R}}_+, \forall \tau \geq T,\ \xi(\tau) \in \{x^*\}, \tag{4.1}$$

where $\mathcal{X}_I$ has non empty interior. In words, there exists some initial set $\mathcal{X}_I$ such that for all trajectories initialised in $\mathcal{X}_I$, there exists a time instant T (possibly at infinity) when the trajectory reaches the equilibrium state x^* and remains there for all times after T. A model can be proven to satisfy this property using a Lyapunov function, which we introduce next.

Certificate 1 (Lyapunov Function)**.** *Given a model f with unique equilibrium point $x^* \in \text{int}(\mathcal{X})$, consider a function $V : \mathcal{X} \subset \mathbb{R}^n \to \mathbb{R}, V \in \mathcal{C}^1$. V is a Lyapunov function*

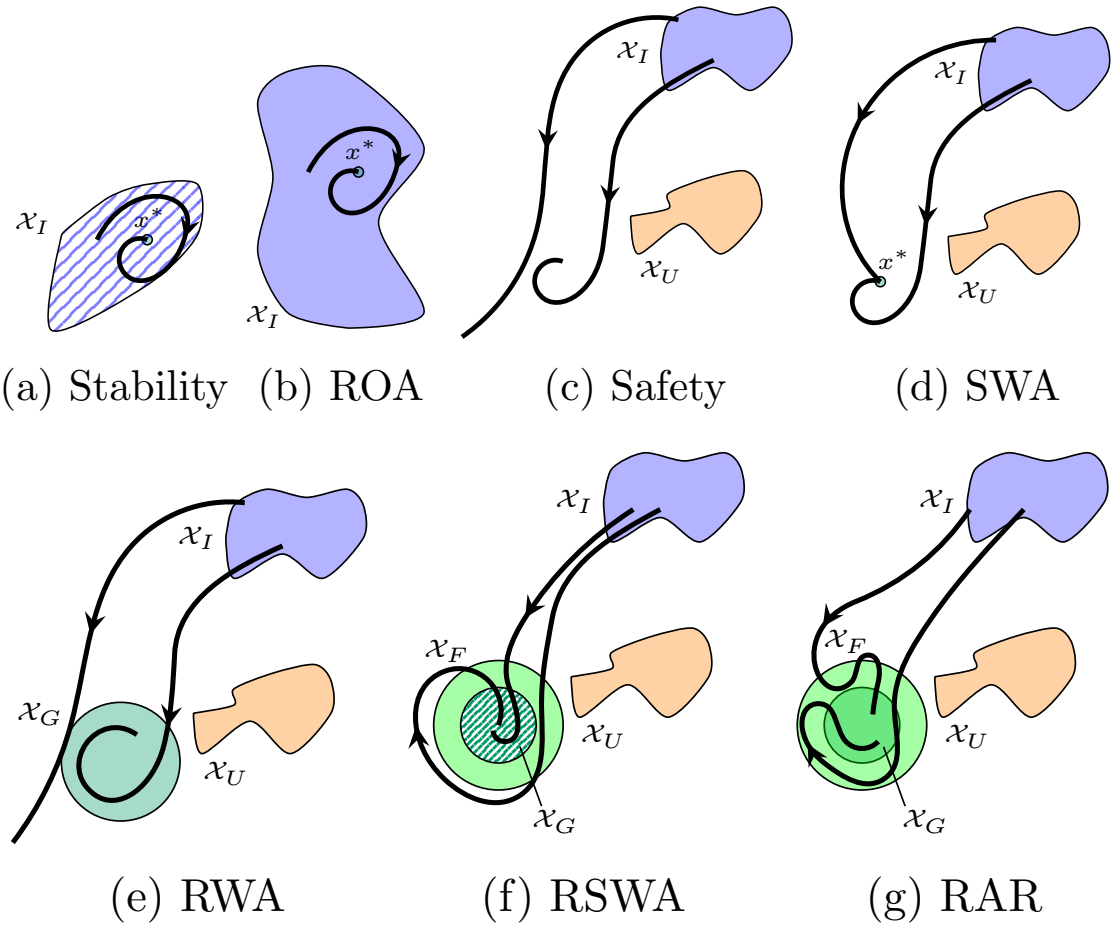


Figure 4.1: Pictorial depiction of relevant properties in this work. Here, $\mathcal{X}_I$ is the initial set, $\mathcal{X}_U$ the unsafe set ($\mathcal{X}_S$ is its safe complement), $\mathcal{X}_G$ the goal/target set, $\mathcal{X}_F$ the final set. (The entire state space is $\mathcal{X}$.) Here, a dashed background denotes that the corresponding set's existence is implied by the corresponding certificate, but that it is not explicitly defined in the property.

if:

$$V(x^*) = 0, \tag{4.2a}$$

$$V(x) > 0 \quad \forall x \in \mathcal{X} \setminus \{x^*\}, \tag{4.2b}$$

$$\dot{V}(x) = \langle \nabla V(x), f(x) \rangle < 0 \quad \forall x \in \mathcal{X} \setminus \{x^*\}. \tag{4.2c}$$

Theorem 4.1 (Stability)**.** *Given a model* (2.5)*, if a Lyapunov function exists, then* (4.1) *holds for some set of initial conditions* $\mathcal{X}_I$*.* ■

Proof. (Proof of Thm. 4.1) We state without proof from Lyapunov theory [152] that the conditions in (4.2) imply that $f(x)$ is asymptotically stable, and that all sub-level sets of $V(x)$ fully contained within $\mathcal{X}$ are forward invariant. Since V is continuous and $\mathcal{X}$ is non-empty, then there exists some β such that the sub-level set $\Omega_\beta^-(V) = \{x \in \mathcal{X} \mid V(x) \leq \beta\}$ is fully contained within $\mathcal{X}$ and $x^* \in \Omega_\beta$. Since x^* is asymptotically stable, all trajectories in Ω_β converge towards it, and (4.1) holds for the initial set Ω_β. □

The structure of a Lyapunov function motivates our characterisation of stability. A Lyapunov function is a scalar function that is positive definite, and provides insights through its sub-level sets. Specifically, the sub-levels sets of a Lyapunov function are forward invariant, and the largest sub-level set over which the Lyapunov conditions (4.2) hold is the largen proven region of attraction for the equilibrium point. This corresponds to the initial set $\mathcal{X}_I$ in (4.1). Let us clarify the issue of finding $\mathcal{X}_I$ in the next section.

4.3.2 Region of Attraction

Thm. 4.1 proves the existence of some region of the state space in which initialised trajectories will converge asymptotically towards the origin – this region is known as

a *region of attraction* (ROA). In general, Lyapunov functions merely prove the existence of a region of attraction within the state space, without additional information about its size or shape, though these attributes are in turn governed by the template used for the Lyapunov function. Lyapunov functions may prove global asymptotic stability under additional conditions, but these can be difficult to synthesise and verify automatically for some models. Here, we offer a certificate that acts as a "middle ground" between local and global asymptotic stability.

Proving that all trajectories initialised within a given $\mathcal{X}_I$ are stable amounts to proving that $\mathcal{X}_I$ is contained wholly within a sub-level set of a Lyapunov function (which must also lie in $\mathcal{X}$), and that the Lyapunov conditions in (4.2) hold over this entire sub-level set. This is treated in the following equation and corollary.

First, we modify (4.1) to now require a specified set of initial states $\mathcal{X}_I$, which should be a region of attraction for an equilibrium point, as follows:

$$\forall \xi(t_0) \in \mathcal{X}_I, \exists T \in \overline{\mathbb{R}}_+, \forall \tau \geq T,\ \xi(\tau) \in \{x^*\}. \tag{4.3}$$

As with the stability property, T is allowed to be infinity. We can certify that an autonomous model satisfies this property using the following certificate.

Certificate 2 (ROA Certificate)**.** *Let a dynamical model f be given with unique equilibrium point $x^* \in \text{int}(\mathcal{X})$. A Lyapunov function V is an ROA certificate if there exists a β such that $\mathcal{X}_I \subset \{x \in \mathcal{X} : V(x) \leq \beta\}$ and that the conditions in (4.2) hold over the set $\{x \in \mathcal{X} : V(x) \leq \beta\}$.* ■

Alternatively, a Lyapunov function can be interpreted as a proof that a region of attraction (here $\mathcal{X}_I$) exists within some larger set ($\mathcal{X}$ here), whereas a ROA certificate proves the converse: that a given initial set $\mathcal{X}_I$ lies within a larger region of attraction, which is defined by a sublevel set of V. This certificate offers a more practical guarantee over classical Lyapunov functions: all trajectories initialised within

the pre-defined set $\mathcal{X}_I$ indeed converge (asymptotically) to x^*.

Corollary 4.2 (Region of Attraction)**.** *Given a model* (2.5)*, a bounded set of initial conditions* $\mathcal{X}_I$*, and a ROA certificate, then* (4.3) *holds.* ■

Proof. (Proof of Cor. 4.2) Let the β sub-level of the ROA certificate V be $\Omega_\beta^-(V) \coloneqq \{x \in \mathcal{X} \mid V(x) \leq \beta\}$. By construction, $\mathcal{X}_I \subset \Omega_\beta$, and the conditions of (4.2) hold. The proof then follows directly from Thm. 4.1. □

4.3.3 Safety

Safety is another fundamental property we can require from dynamical models. It involves the *avoidance* of some unsafe region:

namely, that no trajectory starting from $\mathcal{X}_I$ may enter the unsafe set $\mathcal{X}_U$[2]; formally

$$\forall \xi(t_0) \in \mathcal{X}_I, \forall t \in \overline{\mathbb{R}}_+, t \geq t_0, \xi(t) \in {\mathcal{X}_U}^{\complement}. \tag{4.4}$$

For continuous-time models, safety over an unbounded time horizon can be proved via barrier certificates [135, 136].

Certificate 3 (Barrier Certificate)**.** *Consider a dynamical model* f*, a compact unsafe set* $\mathcal{X}_U$ *and compact initial set* $\mathcal{X}_I$*. A function* $B : \mathcal{X} \subset \mathbb{R}^n \to \mathbb{R}, B \in \mathcal{C}^1$*, is a barrier certificate if the following holds:*

$$B(x) \leq 0 \ \forall x \in \mathcal{X}_I, \tag{4.5a}$$

$$B(x) > 0 \ \forall x \in \mathcal{X}_U, \tag{4.5b}$$

$$\dot{B}(x) < 0 \ \forall x \in \{x \mid B(x) = 0\}. \tag{4.5c}$$

Many characterisations of barrier certificates exist, to cover different applications

[2]We shall later draw connections between the concept of safety and the dual notion of (unconstrained) *reachability* in Section 4.3.8.

or to abide by additional constraints. These include reciprocal [15], high-order (zeroing) [161], or barrier conditions with modifications on the Lie derivative (4.5c) [135]. For the last case, the condition on the Lie derivative is modified to be only non-positive, but over the entire domain $\mathcal{X}$ rather than just the zero level set of B. These certificates are alike in that they certify safety properties. However, the provided, more conventional barrier certificate formulation is most suitable for this work in terms of simplicity and usability.

Theorem 4.3 (Safety)**.** *Given a model* (2.5)*, a compact domain* $\mathcal{X}$*, a compact initial set* $\mathcal{X}_I \subset \mathcal{X}$ *and an unsafe set* $\mathcal{X}_U$*, alongside a barrier certificate, then* (4.4) *holds.* ■

Barrier certificate theory leverages the concept of *forward invariance*, established by Nagumo's theorem (also known as the Bony Brezis theorem), which we state here without proof. For a full proof, see [31].

Theorem 4.4 (Nagumo's Theorem)**.** *Consider the system* $\dot{\xi}(t) = f(\xi(t))$*, where* f *is Lipschitz continuous such that for each initial condition* $\xi(t_0) \in \mathcal{X}_I$ *it admits a unique solution. Let* $\Omega \in \mathcal{X}$ *be a closed set.* Ω *is positively invariant if and only if for every exterior normal vector* v *at point* x *on the border of* $\partial\Omega$*, the inner product satisfies* $\langle f(x), v\rangle \leq 0$.

We may now prove Thm. 4.3.

Proof. (Proof of Thm. 4.3) By definition we have that $\mathcal{X}_I \cap \mathcal{X}_U = \emptyset$. The set $\Omega_0^-(B) = \{x \in \mathcal{X} \mid \dot{B}(x) \leq 0\}$ defines a closed set for which (4.5c) ensures that $f(x)$ points inwards along its border. It follows from Theorem 4.4 that Ω_0 is an forward invariant set which contains $\mathcal{X}_I$, and that at all trajectories initialised in $\mathcal{X}_I$ remain within Ω_0 for all time $t > t_0$. (4.5b) ensures that $\Omega_0 \cap \mathcal{X}_U = \emptyset$, and hence the specification in (4.4) holds. □

4.3.4 Stable While Avoid

It is natural to extend the aforementioned notions of stability and safety towards a combination of both, whereby all relevant trajectories converge towards an equilibrium point, while also avoiding a given unsafe set. Such a property is formally described as follows:

$$\forall \xi(t_0) \in \mathcal{X}_I, \exists T \in \overline{\mathbb{R}}_+, \forall t \in [t_0, T),\ \xi(t) \in {\mathcal{X}_U}^{\complement} \wedge \forall \tau \geq T, \xi(\tau) \in \{x^*\}. \tag{4.6}$$

Since (4.6) is simply the conjunction of a ROA property and a safety property (with T allowed to be infinity as before), certifying this is equivalent to concurrently certifying the model is safe and that the initial set lies within a region of attraction: this task can be thus formally tackled by the following corollary.

Corollary 4.5 (Stable while avoid (SWA))**.** *Given a model* (2.5) *with unique equilibrium point $x^* \in \mathcal{X}$, a compact domain $\mathcal{X}$, a compact initial set $\mathcal{X}_I \subset \mathcal{X}$ and an unsafe set $\mathcal{X}_U$ alongside a ROA certificate V and barrier certificate B, then* (4.6) *holds.* ■

Proof. (Proof of Cor. 4.5) We note that the specification described by (4.6) is the conjunction of those in (4.3) and (4.4). Therefore, the proof follows directly from Cor. 4.2 and Thm. 4.3. □

Notice that it is alternatively possible to combine the conditions of (4.2) and (4.5) into that of a *single* certificate for stability and safety. Such a function is sometimes referred to as a Lyapunov-Barrier certificate [169, 146]. However, in this work we choose to use two separate functions, as this makes synthesis easier and more modular.

4.3.5 Reach While Avoid

Let us now set asymptotic stability aside, and instead study properties over a finite time horizon: namely, we require that trajectories enter a non-singleton set in finite time. These are reachability-like properties, which require trajectories to reach a region known as a goal set. A reach-while-avoid (RWA) property requires a non-singleton goal set to be reached within a finite time horizon T, while avoiding an unsafe region; in formal terms,

$$\begin{aligned} \forall \xi(t_0) \in \mathcal{X}_I, \exists T \in \mathbb{R}, \forall t \in [t_0, T] : \\ \xi(t) \in \mathcal{X}_U{}^{\complement} \wedge \xi(T) \in \mathcal{X}_G. \end{aligned} \tag{4.7}$$

Next, we introduce a RWA certificate to guarantee that this property holds for a model under consideration.

Certificate 4 (RWA)**.** *Define an unsafe set $\mathcal{X}_U = \mathcal{X} \setminus \mathcal{X}_S$, where $\mathcal{X}_S$ is a compact safe set, a compact initial set $\mathcal{X}_I \subset \text{int}(\mathcal{X}_S)$, and a compact goal set $\mathcal{X}_G \subset \text{int}(\mathcal{X}_S)$ with non-empty interior. A reach-while-avoid (RWA) certificate [166] is a function $V : \mathbb{R}^n \to \mathbb{R}, V \in \mathcal{C}^1$ if there exists $\gamma \in \mathbb{R}_+$, such that*

$$V(x) \leq 0 \quad \forall x \in \mathcal{X}_I, \tag{4.8a}$$

$$V(x) > 0 \quad \forall x \in \partial\mathcal{X}_S, \tag{4.8b}$$

$$\dot{V}(x) \leq -\gamma \quad \forall x \in \{x \in \mathcal{X}_S \mid V(x) \leq 0\} \setminus \mathcal{X}_G. \tag{4.8c}$$

Theorem 4.6 (Reach-While-Avoid)**.** *Given a model (2.5) and a RWA Certificate corresponding to the given sets of interest, then (4.7) holds.* ■

Proof. (Proof of Thm. 4.6 (from [167])) Let $A = \{x \in \mathcal{X}_S : V(x) \leq 0\}$. For $\xi(t_0) \in \mathcal{X}_I$, it follows from (4.8a) and the definition of A that $\xi(t_0) \in A$. From (4.8c), for all

$\xi(t_k) \in A \setminus \mathcal{X}_G$, $\dot{V}(\xi(t_k)) < -\gamma$. Using $\forall x \in A, V(x) \leq 0$ and the comparison principle [103], it follows that $\forall t \in [t_k, t_k+h], \forall \xi(t_k) \in A \setminus \mathcal{X}_G : V(\xi(t)) \leq V(\xi(t_k)) - \gamma h \leq -\gamma h$. Therefore, $\xi(t_k) \in A \setminus \mathcal{X}_G$ implies $\forall t \in [t_k, t_k+h], V(\xi(t)$ will decrease and thus cannot reach $\partial\mathcal{X}_S$, since by (4.8b) $\forall x \in \partial\mathcal{X}_S : V(x) > 0$. Since $\mathcal{X}_S$ is compact and $V(x)$ is continuous, $\exists e \in \mathbb{R}$ s.t. $e = \inf_{x \in \mathcal{X}_S \setminus \mathcal{X}_G} V(x)$ and the sublevel sets of $V(x)$ are compact. It follows that $V(x)$ is lower bounded on $A \setminus \mathcal{X}_G \subset \mathcal{X}_S \setminus \mathcal{X}_G$, so $V(\xi(t))$ will decrease until in finite time $\xi(t)$ leaves $A \setminus \mathcal{X}_G$ and may only enter $\mathcal{X}_G$. □

Remark 4.7 (Unconstrained Reachability)**.** *Unconstrained reachability can be defined as a special case of RWA, where we set $\mathcal{X}_U = \emptyset$ (i.e. $\mathcal{X}_S = \mathcal{X}$). Hence, a certificate can be provided accordingly, as special instance of Certificate 4.* ■

We emphasise that a RWA certificate does not prove that trajectories will *remain* within the goal set for all time, or that trajectories shall avoid the unsafe set for all (unbounded) time, nor does the specification in (4.7) indeed encode these requirements. In fact, since the Lie derivative condition (4.8c) does not hold across the goal set $\mathcal{X}_G$, it is possible for trajectories to leave the goal set after entering it, and thereafter enter the unsafe set $\mathcal{X}_U$. This alternative, more restrictive scenario is addressed by the next certificate.

4.3.6 Reach-and-Stay While Avoid

A reach-and-stay while avoid (RSWA) property is similar to the RWA property, consisting of RWA with an additional requirement that trajectories will eventually remain within a given final set indefinitely, after reaching some subset of it, which we refer to as a goal set. Though this goal set is not explicitly defined in the specification, it is implicitly defined by the certificate to be desirable to reach as after reaching it, trajectories must remain within the final set for all time. Formally, trajectories

should satisfy the property

$$\exists \mathcal{X}_G : \forall \xi(t_0) \in \mathcal{X}_I, \exists T \in \mathbb{R}, \forall t \in [t_0, T], \xi(t) \in {\mathcal{X}_U}^{\complement} \\ \wedge \xi(T) \in \mathcal{X}_G \wedge \forall \tau \geq T : \xi(\tau) \in \mathcal{X}_F. \quad (4.9)$$

Notably, this property does not require that trajectories reach the final set and remain within it, and may enter and leave the final set as long as they eventually remain within the final set. However, in finite time trajectories must reach some subset of the final set, known as a goal set, after which point they must remain within the final set for all time.

Certificate 5 (RSWA)**.** *Define an unsafe set $\mathcal{X}_U = \mathcal{X} \setminus \mathcal{X}_S$, where $\mathcal{X}_S$ is a compact safe set, then define a compact initial set $\mathcal{X}_I \subset \text{int}(\mathcal{X}_S)$, and a compact final set $\mathcal{X}_F \subset \text{int}(\mathcal{X}_S)$. A RSWA [166] certificate is a function $V : \mathbb{R}^n \to \mathbb{R}$, $V \in \mathcal{C}^1$, if there exists $\gamma \in \mathbb{R}_+$ such that the following is satisfied:*

$$V(x) \leq 0 \quad \forall x \in \mathcal{X}_I, \quad (4.10a)$$

$$V(x) > 0 \quad \forall x \in \partial\mathcal{X}_S, \quad (4.10b)$$

$$\dot{V}(x) \leq -\gamma \quad \forall x \in \{x \in \mathcal{X}_S \mid V(x) \leq 0\} \setminus \mathcal{X}_F, \quad (4.10c)$$

$$V(x) > \beta \quad \forall x \in \partial\mathcal{X}_F, \quad (4.10d)$$

$$\dot{V}(x) \leq -\gamma \quad \forall x \in \mathcal{X}_F \setminus \mathit{int}(\{x \in \mathcal{X}_S \mid V(x) \leq \beta\}), \quad (4.10e)$$

for some constant $\beta \in \mathbb{R}$.

Theorem 4.8 (Reach-and-stay while avoid)**.** *Given a model (2.5) and a certificate corresponding to the given sets of interest, then (4.9) holds.* ■

Proof. (Proof of Thm. 4.8 (from [167])) Let $B = \{x \in \mathcal{X}_S : V(x) \leq \beta\}$. From Thm. 4.6, there exists a time $T \geq t_0$ such that $\xi(T) \in \mathcal{X}_F$. Using a similar argument as

before, we conclude that $\forall \xi(T) \in \mathcal{X}_F, \xi(t)$ with $t \geq T$ enters in finite time $\mathcal{X}_F \cap B$. Since B is a sub-level set of the compact set S and continuous V, then B is also compact and so is $\mathcal{X}_F \cap B$. From (4.10e) we have $\forall x \in \partial(\mathcal{X}_F \cap B) : \dot{V}(x) \leq -\gamma$. Combining with (4.10d), we have that all states $\xi(t) \in \partial(\mathcal{X}_F \cap B)$ cannot reach $\partial\mathcal{X}_F$ and $V(\xi(t))$ decreases, meaning these trajectories remain in $\mathcal{X}_F \cap B$. Therefore, $\mathcal{X}_F \cap B$ is forward invariant. Since $\mathcal{X}_F \subset \text{int}(\mathcal{X}_S)$, we have that (4.9) holds. □

The sub-level set of V given by β defines an invariant set contained with the final set, and the certificate ensures that trajectories reach this set in finite time without entering an unsafe region. In turn, this implies that trajectories remain within (a subset of) the final set for all time. Note that the specification described in (4.9) - and the corresponding certificate - permit trajectories to enter and leave the final set, as long as trajectories eventually enter a goal set and do not leave the final set afterwards.

4.3.7 Reach, Avoid and Remain

The final property we consider is again similar to the previous *Reach and Stay While Avoid* property, but, as with the ROA certificate, we seek to remove the existential quantifier over the goal set from (4.9). This means that the Reach Avoid Remain (RAR) property requires that trajectories remain within a final set after reaching a goal set, but for specified goal and final sets. We express this formally, as follows:

$$\forall \xi(t_0) \in \mathcal{X}_I, \exists T \in \mathbb{R}, \forall t \in [t_0, T] : \\ \xi(t) \in \mathcal{X}_U{}^{\complement} \wedge \xi(T) \in \mathcal{X}_G \wedge \forall \tau \geq T : \xi(\tau) \in \mathcal{X}_F. \quad (4.11)$$

Certificate 6 (RAR). *Define an unsafe set $\mathcal{X}_U = \mathcal{X} \setminus \mathcal{X}_S$, where $\mathcal{X}_S$ is a compact safe set, a compact initial set $\mathcal{X}_I \subset \text{int}(\mathcal{X}_S)$, a compact final $\mathcal{X}_F \subset \text{int}(\mathcal{X}_S)$, and a compact goal set $\mathcal{X}_G \subset \text{int}(\mathcal{X}_F)$ with non-empty interior. Let $V : \mathbb{R}^n \to \mathbb{R}$ be a RWA*

certificate, and a function $B : \mathbb{R}^n \to \mathbb{R}$, $B \in \mathcal{C}^1$, *such that:*

$$B(x) \leq 0 \ \forall x \in \mathcal{X}_G, \tag{4.12a}$$

$$B(x) > 0 \ \forall x \in \partial\mathcal{X}_F, \tag{4.12b}$$

$$\dot{B}(x) < 0 \ \forall x \in \{x \mid B(x) = 0\}. \tag{4.12c}$$

The pair (V, B) *define a Reach-Avoid-Remain certificate.* ■

As with the Stable-While-Avoid certificate, we choose to formulate this certificate as a pair of separate functions, rather than collapsing the conditions to a single function. This choice, which of course does not affect the soundness of the approach, renders synthesis practically easier and more modular.

Theorem 4.9 (Reach-Avoid-Remain)**.** *Given a model* (2.5) *and a certificate pair* V, B *satisfying the conditions in Certificate 6, then* (4.11) *holds.* ■

Proof. (Proof of 4.9) This proof follows directly from Theorem 4.3 and Theorem 4.6. □

We note that here, the certificate B is similar to a barrier certificate as defined in (4.5), though with $\mathcal{X}_G$ as the initial set and $\partial\mathcal{X}_F$ as the unsafe set. We have restated the function in this context for clarity. This function B is used to ensure that there exists an invariant set contained within the final set, which itself contains the goal set, such that $\mathcal{X}_G \subset \Omega_0^-(B) \subset \mathcal{X}_F$. As before, this ensures that trajectories remain within the final set, no matter where they first enter the goal set. Since the function V ensures that trajectories enter the goal set in finite time (while avoiding the unsafe set), our specification is satisfied.

4.3.8 Summary and Classification of Properties

So far, we have presented a number of different properties that a dynamical model may conform to. These properties, and the certificates that sufficiently prove them to hold, can be complex and subtly different. However, we observe the following similarities between them:

- All certificates rely on a set on initial conditions $\mathcal{X}_I$. In the case of a Lyapunov function, this set is implicitly defined a-posteriori to the synthesis of the certificate.
- $\mathcal{X}_U$ denotes a region trajectories should *avoid* (and thus relate to a safety requirement).
- Either $\mathcal{X}_G$ or $\{x^*\}$ denote a region which trajectories should enter or *arrive* at. We leverage this notion to encompass both finite time reachability and asymptotic stability, and note that it can be thought as the dual of the *avoid* category.
- Either $\mathcal{X}_F$ or $\{x^*\}$ denote a region which trajectories should eventually *remain* in, for all time, as soon as they have *arrived* in a goal set. This notion is thus related to both (forward) set invariance and asymptotically stable equilibria.

Based on these analogies, we introduce three labels *avoid*, *arrive* and *remain*, and assign them to each certificate, in order to clarify their purpose and differences. We portray these relationships in Fig. 4.2, where the label *arrive* encompasses both stability and finite time reachability.

We note that this classification does not fully distinguish between some certificates, which we clarify here. Firstly, as previously mentioned, a Lyapunov function and a ROA certificate differ in that for the latter an initial set is explicitly specified a priori to synthesis. Meanwhile, the SWA, RSWA and RAR certificates satisfy all three

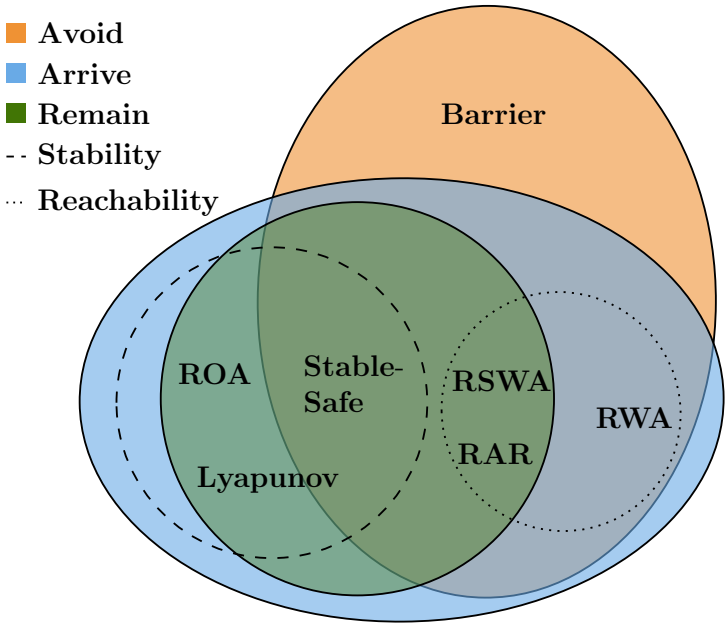


Figure 4.2: Euler Diagram depicting the semantic labels, *arrive, avoid*, and *remain*, associated with each certificate in this work. By the dashed line, we group properties that exhibit asymptotic stability. By the dotted line, we denote properties that exhibit (finite time) reachability.

labels. Stable while Avoid is easy to distinguish, as it handles asymptotic stability, whereas the others treat finite time reachability. Finally, we differentiate the RSWA and RAR certificates as follows. A RSWA certificate proves that there exists a goal set, contained within a given final set, that trajectories will reach and afterwards never leave the final set. Meanwhile a RAR certificate explicitly defines both the goal set and final set associated with this, allowing for a more elaborate specification.

4.3.9 Certificates for Control Models

Much work in the literature of certificate synthesis refers to, e.g., *control* Lyapunov functions and *control* barrier certificates. In this work, we consider such terms to concern certificates that refer to a model expressed in terms of both state x and control input u, as in (2.4). Existing approaches often specify a control set, over which the specifications quantify existentially. In other words, they seek certificates for which

there always exists a valid control action that allows for the property to hold. A control Lyapunov function therefore proves that there always exists a suitable control input such that the Lyapunov conditions hold, and hence the system is (asymptotically) stabilisable. This approach entails that, after a valid control-certificate is synthesised, control actions can be determined, e.g. by solving an optimisation program over the input space and the found certificate.

We emphasise that this is *not* the approach taken in this work. Instead, we synthesise a feedback control law for the model described by (2.4), and "apply" this state feedback to obtain a closed-loop model of the form of (2.5), for which we synthesise a certificate. Whilst we synthesise the control law *concurrently* with the certificate, we do not refer to these as "control certificates", as in the literature. Hence, we verify properties for control models using both a controller and a certificate, and in particular do not delegate the controller synthesis a-posteriori.

4.4 Synthesis of Certificates and Controllers

Following the introduction of specifications that are salient for verification purposes, and certificates whose existence prove that a given model satisfies these conditions, we describe next a unified, efficient, and sound algorithm for the synthesis of these certificates, for both dynamical and controlled models. The objective of the synthesis task is twofold: we seek a feedback controller $k(x)$ (with a chosen fixed template) that ensures a given control model satisfies a desired specification, and concurrently we synthesise a certificate $C(x)$ that serves as a proof that the specification holds. As such, allow us to use $C(x)$ to refer to each function in a possible pair separately. Call with θ_u the parameters of the state feedback law $k(x)$ and θ_c the parameters of the certificate function $C(x)$. The synthesis task therefore amounts to finding values

for the parameters θ_u and θ_c such that the certificate conditions hold, i.e.,

$$\exists\ \theta_u, \theta_C\ \forall x \in \mathcal{X} :\ \phi(C(x)), \tag{4.13}$$

where $\phi(C(x))$ denotes the conditions, related to a given specification (as formalised in the previous section), that the certificate ought to satisfy. Our procedure is based on counter-example guided inductive synthesis (CEGIS) [158], an established approach to formally solving such exists-forall problems. CEGIS consists of two opposing components: a *leaner* and a *verifier* (cf. Figure 4.4), which we detail in turn.

4.4.1 Learner

The learner fulfils the twofold task of designing both a stabilising control law (when required) and the desired certificate, as exemplified in Fig. 4.3.

In this work, we seek *feedback* control laws and employ neural networks as a general template for these functions: neural architectures allow for a wide variety of control laws, as determined by the choice of their activation functions and of the number of neurons.

Our candidate controller is thus simply the output of a neural network, contributing to the closed-loop dynamics, and which is then employed within the certificate synthesis procedure.

For the certificate synthesis, we also employ a neural network as a template for $C(x)$: this allows not only for polynomial templates akin to classical (e.g., SOS-based) certificates, but also for highly non-linear functions, leveraging the expressive power of neural networks. One of the crucial components of a successful training within the learner is the definition of a tailored loss function. We can define a loss function that allows us to train both the controller and the certificate simultaneously (with the same gradient steps), as follows.

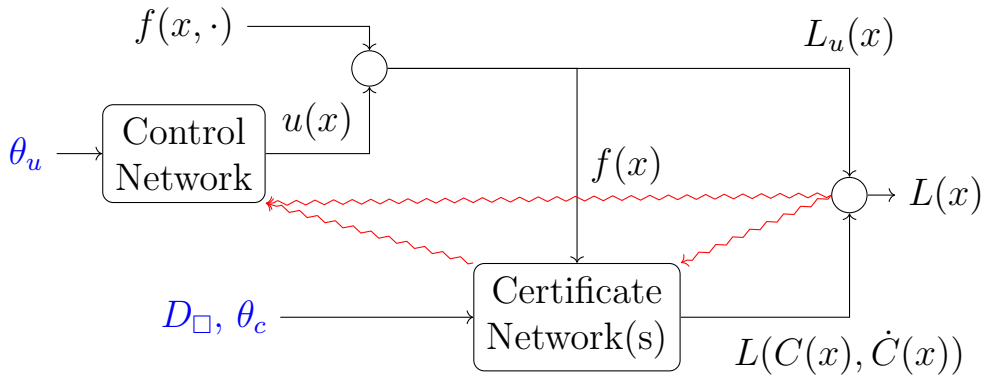


Figure 4.3: Loss calculation block diagram. Blue indicates the inputs required for the loss calculation: namely the sampled data sets $D_\square$, the control and certificate network parameters θ_u and θ_c, respectively. Red arrows represent the back-propagation steps, updating the networks' parameters during training. $f(x,\cdot)$ corresponds to the model in (2.4), while $f(x)$ corresponds to the model in (2.5).

4.4.1.1 Certificate Loss

We note that all of the certificate conditions described in the previous section can be expressed in terms of inequalities, either as

$$C(x) \bowtie c \ \forall x \in \mathcal{X}_\square, \text{ or as } \dot{C}(x) \bowtie c \ \forall x \in \mathcal{X}_\square, \tag{4.14}$$

where $\bowtie \in \{<, >, \geq, \leq\}$, $C(x)$ represents the certificate, $\mathcal{X}_\square$ represents the relevant set (e.g., $\mathcal{X}_I$ or $\mathcal{X}$, $\square \in \{\emptyset, \text{I, U, S, F, G}\}$). and $c \in \mathbb{R}$. We can then construct a loss function based on this observation, as follows. Consider a monotonically increasing function $g(\cdot)$ and suppose we have a finite set of sampled data points over the set $\mathcal{X}_\square$, which we denote as $D_\square$. A general loss function for any of the discussed certificate conditions is thus

$$L_C = \sum_{d \in D_\square} g(p \cdot C(d)), \tag{4.15}$$

where $p = 1$ if $\bowtie = \{\leq, <\}$ and $p = -1$ if $\bowtie = \{\geq, >\}$. Note that this function penalises points in $D_\square$ where the required condition is not satisfied. Suitable choices for function m are leaky-ReLU, which is piecewise linear, and softplus, which is

smooth. Since several of the sets $\mathcal{X}_\square$ are boundaries of sets, or represent level sets, in practice we consider a small "band" around them, in order to encompass a sufficient number of data points in $D_\square$.

Example. Let us consider a barrier certificate $B(x)$ for safety verification (see (4.5)). We create separate sets of finite samples for each set defined by the property (initial, unsafe, state-space), and the resulting loss function is given by

$$L_C = \frac{1}{N_I}\sum_{d\in D_I} g(B(d)) + \frac{1}{N_U}\sum_{d\in D_U} g(-B(d)) + \frac{1}{N}\sum_{d\in Z_B} g(\dot{B}(d)). \tag{4.16}$$

Here $Z_B(d) = \{d \in D : B(d) = 0\}$, where D is the set of N samples over the whole state space, D_I is the set of N_I samples over the initial set and D_U is the set N_U of samples of the unsafe set. As described, we may modify the definition of Z_B to include a band around the zero-level set of $B(x)$, in order to ensure that enough samples are included in the loss function. This band is defined as $Z_B^\tau(d) = \{d \in D : |B(d)| < \tau\}$, where τ is a small positive constant.

■

4.4.1.2 Control Loss

Let us now discuss controlled models. Note that, as described in Figure 4.3, the parameters of the neural network controller appear in the loss function for the certificate, as described previously. Specifically, they manifest themselves in the terms corresponding to conditions on the Lie derivative of the certificate, since these in turn depend on the control-dependent dynamics f. This should encourage the learner to seek a controller that enables a valid certificate.

However, in practice we find that this is insufficient for robust synthesis in the case of *arrive* requirements with non-trivial sets (in other words, not including Lyapunov

and ROA certificates). While our certificates and properties make no assumption on the location of the goal set or of the equilibrium, often an equilibrium point (which, without loss of generality, is the origin) lies within the goal set.

If this is the case, trajectories converging to the origin exhibiting a negative Lie derivative are in turn attracted to the goal set. Since the Lie derivative consists of two components, $f(x)$ and $\nabla C(x)$, it can be helpful to separate these elements within the loss function to encourage better learning. We thus propose the additional term for the loss function:

$$L_u = \frac{1}{N} \sum_{d \in D} \frac{\langle d, f(d) \rangle}{||d|| \cdot ||f(d)||}, \tag{4.17}$$

where $\langle \cdot, \cdot \rangle$ is the inner product of its inputs and $|| \cdot ||$ is the 2-norm of its input, and D is the set of N samples over the state space. This loss is known as the *cosine similarity* of the two vectors d and $f(d)$, and rewards vectors for pointing in opposite directions, but without being influenced by the magnitude. We interpret this loss function as separating $f(x)$ from $\nabla C(x)$ when calculating the Lie derivative, and instead replacing the corresponding certificate with a "default" positive definite function (that of Euclidean distance). This encourages the loss function to specifically focus on the parameters in the feedback law to learn a desirable $f(x)$. As an alternative interpretation, since any vector d is fixed and points away from the origin, this encourages a controller which guides the dynamics to point towards the origin, and hence eventually converge towards it. We demonstrate the efficacy of this loss component in Section 4.5.2.

4.4.1.3 Combined Loss

We combine the two loss functions described above to obtain the final loss function for the learner. We note that the control loss is only relevant for controlled models, and hence we only include it in the loss function when synthesising a controller

concurrently with a certificate. The final loss function is then:

$$L = L_C + \lambda L_u, \tag{4.18}$$

where $\lambda \in \{0, 1\}$. If $\lambda = 0$, then we are synthesising a certificate for an autonomous model (and no feedback controller); if $\lambda = 1$, then we are synthesising a certificate and a controller simultaneously. The value of *lambda* is merely indicative of the task at hand, and does not affect the synthesis procedure other than including the control loss in the overall loss function.

4.4.2 Verifier

We now discuss the dual component of the CEGIS architecture, the verifier, as portrayed is Figure 4.4. The verifier's role is assumed by an SMT solver and its operation is described as follows. Let $\phi(C)$ denote the conditions required for a given certificate C. We seek a point $x \in \mathcal{X}$ that violates any of the constraints $\phi(C(x))$ associated to the certificate. To this end, we express the *negation* of such requirements $\neg\phi$, and formulate a nonlinear constrained problem over real numbers. Formally, we ask an SMT solver to find a witness to

$$\exists x \in \mathcal{X} : \neg\phi(C(x)), \tag{4.19}$$

where any such witness x would be considered a counterexample to the validity of the certificate ϕ.

Example (Cont'd). Consider the negation of barrier certificate conditions ϕ as in

(4.5), namely

$$\begin{aligned}&(x \in \mathcal{X}_I \wedge B(x) > 0) \vee \\ &(x \in \mathcal{X}_U \wedge B(x) \leq 0) \vee \\ &(B(x) = 0 \wedge \dot{B}(x) \geq 0).\end{aligned} \tag{4.20}$$

The verifier searches for an assignment d_{cex} of x of the negation constraints in (4.20). In the context of a barrier certificate, this means the verifier searches for a point d_{cex} where the barrier certificate $B(x)$ is positive in the initial set $\mathcal{X}_I$, or negative in the unsafe set $\mathcal{X}_U$, or where the Lie derivative $\dot{B}(x)$ is non-negative at the boundary of the barrier certificate. This in general requires manipulating non-convex functions and sets, and is therefore handled by an SMT solver. Whenever such x is found, it represents a witness that the candidate $B(x)$ is *not* a valid certificate function. We may augment the corresponding data set $D_{\square}$ based on this counterexample and provide additional information to the learner component at the next iteration. ■

The correctness of our algorithm hinges upon the soundness of the verification engine: we use two solvers, Z3 [59] and dReal [76], both of which are sound over non-linear real arithmetic. Z3 is restricted to polynomial reasoning, whereas dReal can handle non-polynomial expressions, for instance containing trigonometric or exponential terms, thus allowing for more complex models and certificates (via their activation functions). We note that dReal is a δ-complete SMT-solver: while this guarantees dReal will always find a counterexample if one exists, it may return also spurious counterexamples within a δ-perturbation of the original formula [76]. This implies that our procedure may not terminate successfully, even when the candidate is valid. However quite importantly it does not compromise the correctness of certificates we verify successfully.

4.4.3 Enhanced Communication amongst Components

Our CEGIS approach builds on that of the software tool Fossil [3]. As part of its CEGIS loop, Fossil adds two elements to enhance the communication between components.

4.4.3.1 Translator

The translator is tasked with the conversion of the neural networks into a symbolic candidate $C(x)$ and the corresponding $\dot{C}(x)$, ready to be processed by the verifier (see Fig. 4.4).

The efficiency of SMT solvers depends also upon the numerical expressions of the formulae to be verified. Often the training returns numerically ill-conditioned expressions, e.g. unreasonably small coefficients (e.g., in the order of 10^{-8}); these candidates might slow the verification step, and thus the whole procedure. The translator thus rounds the coefficients of the candidate function to a specified precision, in order to help human interpretation and the verification process.

4.4.3.2 Consolidator

Generating counterexamples is, in general, an expensive procedure, and the verification engine returns a single counterexample (d_{cex} in Fig. 4.4); e.g. an instance satisfying (4.20). Naturally, an isolated sample does not provide enough information for the learner to improve the candidate certificate. To overcome this issue, we randomly generate a *cloud* of points around the d_{cex} point, since samples around a counterexample are also likely to invalidate the certificate conditions. Secondly, starting from d_{cex}, we compute the gradient of C (or of $\dot{C}$) thanks to an automatic differentiation feature, and follow the direction that maximises the violation of the certificate constraints. The consolidator then aggregates the original counterexample and the newly generated points (denoted d^{+}_{cex} in Fig. 4.4) with the sample set S for

a new synthesis attempt by the learner.

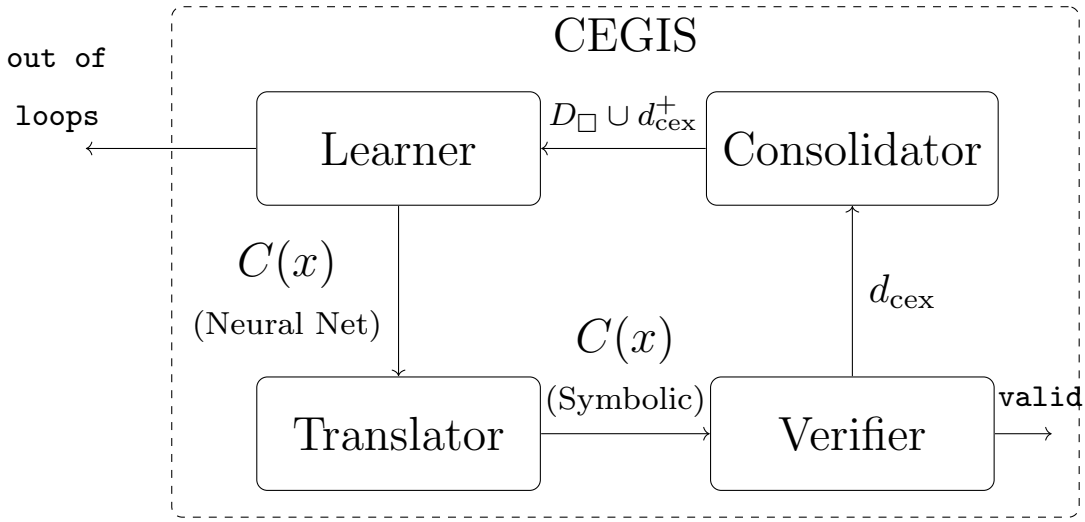


Figure 4.4: Enhanced CEGIS architecture within Fossil.

4.4.4 Comments on Specific Certificates

We add a few remarks next, elaborating on practical synthesis details that are unique to specific certificates.

4.4.4.1 Lyapunov Functions

We assume without loss of generality that the given model has an equilibrium at the origin. We aid synthesis by enforcing the positivity condition by construction: in the case of positive-definite functions this is done by fixing the output layer's weights W_{k+1} to be all positive.

4.4.4.2 ROA

For verification, we estimate the smallest sub-level set that contains $\mathcal{X}_I$ using sample points. Then, we verify that $\mathcal{X}_I$ is contained wholly within this level set and that the Lyapunov conditions hold over the entire sub-level set. Due to dReal's inner workings, when verifying either Lyapunov or ROA certificates, we remove a very small region

around the origin from the verification domain. This is common practice [40] across similar works, and does not affect sub-level sets outside this region. In fact, this notion is studied as ϵ-stability [75]. This caveat does not apply when using Z3.

4.4.4.3 RSWA

The choice of γ for the RSWA and RWA certificates is arbitrary and may be fixed prior to synthesis. Meanwhile, the conditions described in (4.10d, 4.10e) depend on the existence of the parameter β. We must prove a value for this parameter exists such that the relevant conditions hold in order to prove the certificate is valid. After successfully verifying the conditions which do not depend on β, we perform a line search for β to find a suitable value [166]. Given a value for β, we check the conditions in 4.10d, 4.10e using dReal. If 4.10d is violated, we decrease β and repeat the check. If 4.10e is violated, we increase β and repeat the check. This process is repeated until both conditions are satisfied, or until the line search fails. If we do not find a valid β, we return to synthesis and keep training.

4.5 Computational Experiments and Benchmarks

We have implemented the proposed framework based on the computational library of Fossil. We have tested our approach across a large number of case studies, and benchmarked it against the first release of Fossil, showing improved results.

4.5.1 Main Results

Our new prototype tool, which we refer to as Fossil 2.0, is able to verify all properties described in Section 4.3 for continuous-time models, both autonomous and controlled. The extension to all the presented properties in discrete time is matter of future work. We showcase the efficacy of our framework and corresponding tool across 26 bench-

marks, taken from existing literature on certificate synthesis [150, 167, 3, 165]. Note that, in some cases, we have modified these benchmarks to further challenge our approach, for instance by using disjoint, non-convex sets in the specifications. We consider a key strength of our approach to be its flexibility - we are able to perform well on straightforward and challenging benchmarks using certificates that represent both polynomials and more complex non-polynomial functions (as determined by the activation function of the neural network). We reflect this in our selection of benchmarks, including dynamics that are relatively simple and dynamics that involve transcendental and trigonometric functions. Due to the large number of benchmarks, details on the dynamics and sets can be found in Appendix A.2, and in the corresponding code-base, https://github.com/oxford-oxcav/fossil, where additional benchmarks can be also found. The results are reported in Table 4.1, where for each benchmark we outline the number of variables N_s and of control input N_u, the property to be verified (cf. acronyms introduced earlier), the number of neurons in each hidden layer and the corresponding activation functions for these layers. The number of neurons is denoted as a list, e.g. $[n_1, n_2]$ indicates two hidden layers are composed of n_1 and n_2 neurons, respectively. The activation functions for these hidden layers are denoted similarly. As mentioned, a strength of our methodology is its flexibility in terms of the form that certificates may take: we are able to synthesise polynomial certificates as well as non-polynomial certificates that represent more "neural-typical" functions – this is illustrated in the *Activations* column of Table 4.1. By φ_j we denote that the layer represents a polynomial function of order j; σ_{sig} represents the sigmoid function, σ_{t} represents the hyperbolic tangent function, σ_{t^2} is the square of σ_{t} and σ_{soft} is the softplus function.

For almost all benchmarks, we use a linear control function. Our approach can handle more general nonlinear templates, but we emphasise that, as we solve a verification problem, rather than a control problem, we only seek a feedback law such that

the property is satisfied by the closed-loop dynamics, and thus offer no guarantee on the optimality of this controller. Still, we use a nonlinear controller employing σ_t functions for the benchmark number 10 of Table 4.1.

We measure the robustness performance by running each experiment 10 times, where we initialise the network with different weights and a new dataset across separate random seeds. Our procedure is not guaranteed to terminate, so after a maximum number of CEGIS loops we stop it and consider the overall run a failure. In general, we allow 25 CEGIS loops; for the SWA and RAR certificates we allow 100 CEGIS loops as these certificates are composed of two functions.

We consider two metrics to assess the quality of our framework when attempting to synthesise a certificate and controller: how often it returns a successful result, the success rate S, and how long the algorithm takes for these successful runs, the time T. Table 4.1 thus reports S, along with the average, minimum and maximum time for the procedure to terminate, under the T column, denoted μ, min and max, respectively. In brackets, we also denote the amount of time spent during the learning phase of our procedure, with approximately all remaining time spent during the verification phase.

The success rate is consistently close to 100%, with the minimum standing at 40% for a single benchmark , highlighting the robustness of our approach over the presented broad range of complex properties.

4.5.1.1 Visualisations of Certificates

We present visualisations of three certificates from Table 4.1 in Figs. 4.5 to 4.7.

First, Fig. 4.5 shows an example region of attraction certificate in a phase portrait, and corresponds to Benchmark 5 in Table 4.1. This consists of the smallest found sub-level set of the corresponding Lyapunov function which contains the initial set $\mathcal{X}_I$, and that also verified to satisfy the Lyapunov conditions over the entire sub-

	N_s	N_u	Property	Neurons	Activations	T (s)			S (%)
						min	μ	max	
1	2	0	Stability	[6]	$[\varphi_2]$	0.01 ($\approx$ 0.00)	0.16 (0.15)	1.50 (1.48)	100
2	3	0	Stability	[8]	$[\varphi_2]$	0.28 ($\approx$ 0.00)	2.22 (0.45)	12.57 (3.31)	100
3	2	2	Stability	[4]	$[\varphi_2]$	0.07 (0.01)	0.19 (0.02)	0.47 (0.04)	100
4	2	2	Stability	[5]	$[\varphi_2]$	0.09 (0.01)	0.26 (0.02)	0.54 (0.03)	100
5	2	0	ROA	[5]	$[\sigma_{t^2}]$	0.71 (0.02)	1.17 (0.02)	2.59 (0.03)	50
6	3	3	ROA	[8]	$[\varphi_2]$	1.24 (0.02)	39.08 (0.03)	287.89 (0.04)	100
7	2	0	Safety	[15]	$[\sigma_t]$	0.44 (0.35)	3.36 (2.90)	7.61 (7.11)	100
9	8	0	Safety	[10]	$[\varphi_1]$	12.63 (7.71)	51.97 (32.75)	70.59 (44.66)	70
10	3	1	Safety	[15]	$[\sigma_t]$	1.57 (0.19)	11.87 (2.50)	51.08 (7.52)	90
11	3	0	SWA	[6], [5]	$[\varphi_2]$, $[\sigma_t]$	0.19 (0.05)	2.46 (0.100)	12.10 (0.20)	90
12	2	0	SWA	[5], [5, 5]	$[\varphi_2]$, $[\sigma_{sig},\varphi_2]$	0.13 (0.06)	0.27 (0.14)	0.39 (0.20)	100
13	2	1	SWA	[8], [5]	$[\varphi_2]$, $[\varphi_2]$	0.06 (0.03)	0.20 (0.10)	0.58 (0.24)	90
14	3	1	SWA	[10], [8]	$[\varphi_2]$, $[\sigma_t]$	4.06 (0.87)	19.81 (2.73)	103.49 (7.23)	90
15	2	0	RWA	[4]	$[\varphi_2]$	0.14 (0.09)	1.81 (1.75)	4.70 (4.63)	100
16	3	0	RWA	[16]	$[\varphi_2]$	1.36 (0.09)	14.10 (0.14)	72.97 (0.20)	90
17	2	1	RWA	[4, 4]	$[\sigma_{sig},\varphi_2]$	0.59 (0.27)	6.82 (3.32)	20.07 (11.46)	100
18	3	1	RWA	[5]	$[\varphi_2]$	0.46 (0.11)	16.06 (5.81)	72.47 (44.64)	80
19	2	2	RWA	[5]	$[\sigma_{sig}]$	0.69 (0.40)	1.38 (0.94)	2.14 (1.90)	100
20	2	0	RSWA	[4]	$[\varphi_2]$	0.19 (0.03)	1.29 (1.04)	3.79 (3.37)	100
21	3	0	RSWA	[16]	$[\varphi_2]$	4.81 (0.13)	27.14 (0.19)	80.95 (0.25)	100
22	2	0	RSWA	[5, 5]	$[\sigma_{sig},\varphi_2]$	1.52 (0.06)	4.45 (0.19)	10.97 (0.35)	100
23	2	1	RSWA	[8]	$[\varphi_2]$	0.21 (0.05)	0.67 (0.25)	1.19 (0.91)	100
24	2	2	RSWA	[5, 5]	$[\sigma_{sig},\varphi_2]$	0.98 (0.16)	1.23 (0.28)	1.61 (0.46)	100
25	2	0	RAR	[6], [6]	$[\sigma_{soft}]$, $[\varphi_2]$	6.65 (1.08)	24.74 (6.46)	77.80 (15.06)	100
26	2	2	RAR	[6, 6], [6, 6]	$[\sigma_{sig},\varphi_2]$, $[\sigma_{sig},\varphi_2]$	5.13 (1.34)	26.99 (9.90)	101.23 (60.14)	100

Table 4.1: Results of synthesising certificates for all properties presented in this work. The first column indexes the benchmarks. N_s: Number of states, N_u: number of control inputs. We show the *Property* being verified and network structure (*Neurons* and *Activations*). For certificates of two functions, comma-separated lists shows the different structures. Finally, we report success rate (S) and the minimum, mean (μ) and maximum computation time T over successful runs, in seconds. In brackets we show the time spent during the *learning* phase.

level set. This is shown by the larger dashed line. For comparison, we include the sub-level set of a simple Lyapunov function from a stability certificate, shown by the smaller dashed line. We also show the set $\mathcal{X}_I$ in dark blue. This figure demonstrates the utility of the ROA certificate, which is able to verify that trajectories starting in the given $\mathcal{X}_I$ converge to the origin (since they lie within a region of attraction). Moreover, Fig. 4.5 illustrates the low success rate of the corresponding benchmark (40%): this is due to the pathological nature of the dynamics and the non-convexity of the sets involved. This is a well-studied model which is known to be challenging to synthesise a global Lyapunov function and corresponding region of attraction [165], and is why we include it in our benchmark suite.

Fig. 4.6 shows a reach while avoid certificate as a surface plot, and corresponds to Benchmark 17 in Table 4.1. The zero level set is shown as a dashed line. We denote sets as follows: dark blue: $\mathcal{X}_I$; light blue: $\mathcal{X}_S$; green: $\mathcal{X}_G$. This figure illustrates an example non-convex safe set handled by our synthesis methodology.

Finally, Fig. 4.7 shows a RAR certificate, in a phase portrait and corresponding to Benchmark 26 in Table 4.1. As before, we denote sets as follows: dark blue: $\mathcal{X}_I$; light blue: $\mathcal{X}_S$; green: $\mathcal{X}_G$; orange: $\mathcal{X}_F$. The RAR certificate is composed of a RWA function (outer dashed line), which proves that trajectories reach the goal set $\mathcal{X}_G$ in finite time while staying in the safe set, and a barrier component (inner dashed line), which proves that trajectories will remain within the final set $\mathcal{X}_F$ once they reach the goal set $\mathcal{X}_G$.

4.5.2 Control Loss Evaluation

We have presented a novel loss function in (4.17) with the purpose of encouraging trajectories to converge to the origin, which is often desirable in the case of *arrive* conditions, as discussed in Section 4.4.1.2 We evaluate the inclusion of this loss function in two ways: first, we consider the effect on success rate and computation time

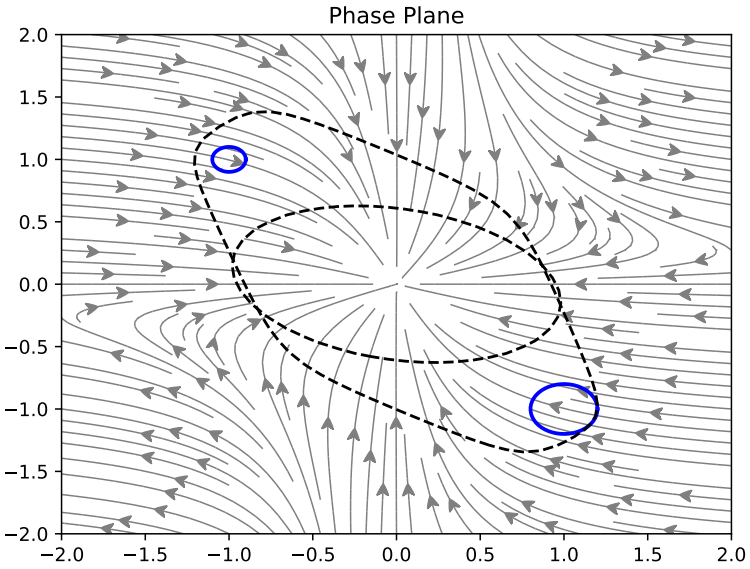


Figure 4.5: ROA certificate (Benchmark 5) showing the largest verified sub-level set and corresponding region of attraction, with the level set (smaller dashed line) of a synthesised Lyapunov function shown for comparison. The set $\mathcal{X}_I$ is shown in dark blue.

	Success (%)	Time (s)
With L_u	**96.67**	**7.14**
Without L_u	80.00	10.64

Table 4.2: Comparison of success rate and computation time for the combined RWA, RSWA and RAR benchmarks with control, with and without the loss term described in (4.17).

relative to not having the term across all control benchmarks for RWA, RSWA and RAR properties, of which there are 6. These results are presented in Table 4.2, where as before we present the success rate (this time over all 60 runs), and the average computation time for successful runs. It is clear that the inclusion of this loss term improves both the robustness and efficiency of the automated synthesis.

Secondly, we compare again these two metrics when instead using an LQR feedback controller. This is the purpose of benchmarks 22 and 24, which are identical except for one key difference: benchmark 22 is equipped with a pre-computed LQR

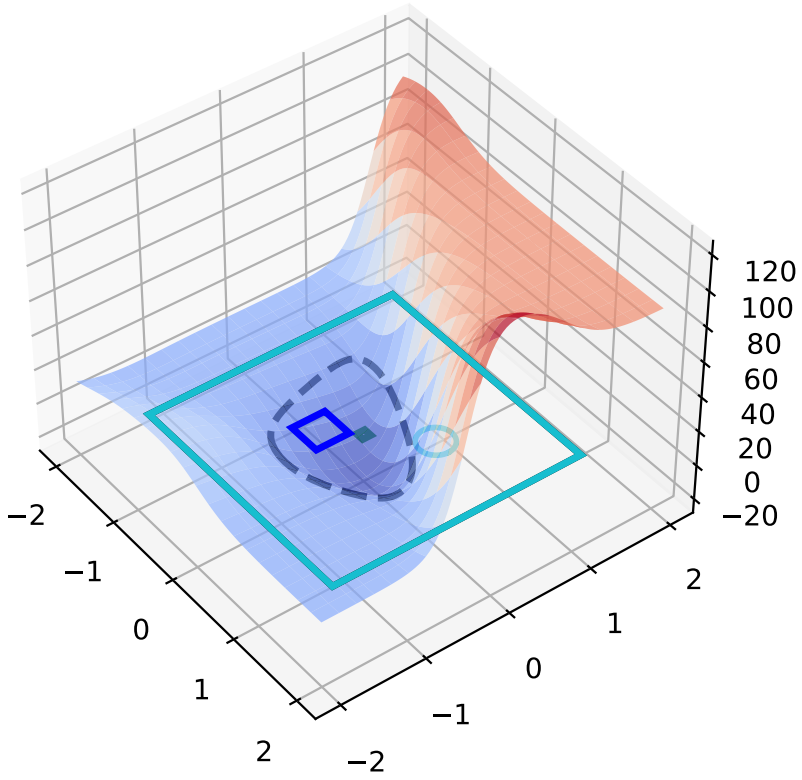


Figure 4.6: Surface plot of a RWA certificate (Benchmark 17) with zero level set as dashed line. We denote sets as follows: dark blue: $\mathcal{X}_I$; light blue: $\mathcal{X}_S$; green: $\mathcal{X}_G$.

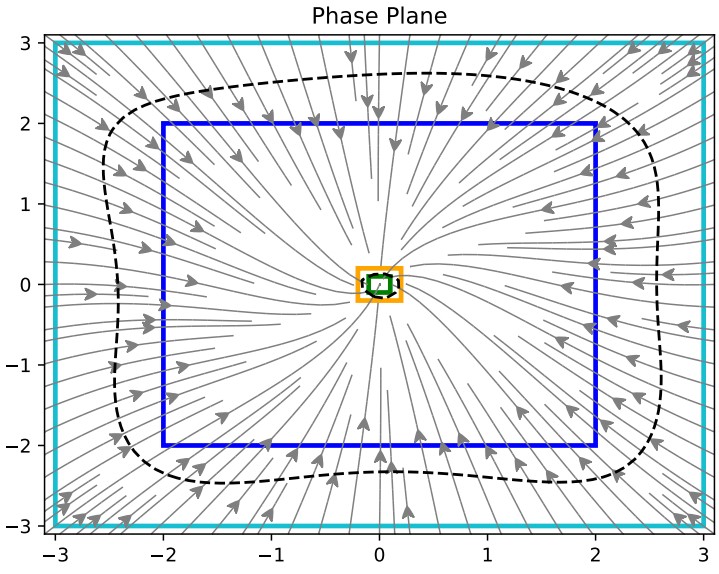


Figure 4.7: RAR certificate (Benchmark 26) with zero level sets for both the RWA function (outer dashed line) and barrier component (inner dashed line). We denote sets as follows: dark blue: $\mathcal{X}_I$; light blue: $\mathcal{X}_S$; green: $\mathcal{X}_G$; orange: $\mathcal{X}_F$.

controller, *de facto* representing an autonomous model, whilst we compute a control law in benchmark 24 using the framework presented in this work. This allows us to compare our framework's ability to learn feedback laws which satisfy properties relative to a common baseline controller, the known LQR. The results for these two benchmarks are very similar, with our control approach performing slightly more efficiently. Note, we do not claim to outperform LQR controllers, as other cost matrices may perform better or worse; further, we do not provide the cost minimisation guarantees - we simply use these benchmarks as a baseline comparison.

4.5.3 Comparison to Fossil 1.0 Baseline

We have built a prototype tool based on our framework, improving on the work initially presented in [3]. This tool compares against competitive state-of-the-art techniques such as SOS-tools, proving to deliver a faster synthesis for stability and safety properties for autonomous models. Table 4.3 collects works providing an automated and sound synthesis of relevant certificates. However, these related works are not proper software tools, and not equipped with a usable interface e.g., to tune network architecture and activation functions or to handle several certificates. We therefore benchmark against Fossil 1.0 as the state-of-the-art for certificate synthesis. These results are presented in Table 4.4.

We employ the benchmarks originally outlined in [3], and for a fair comparison we use the same network structure (width and activations) for both tools. It is clear that the approaches are very similar when synthesising the more straightforward Lyapunov functions. However, we achieve significant improvements in terms of both success rate and synthesis time relative to the baseline, on account of a fine-tuned loss function, an enhanced communication between the CEGIS components, and an overall improved software implementation.

	Stability	ROA	Safety	SWA	RWA	RSWA	RAR	Control
Fossil 2.0	✓	✓	✓	✓	✓	✓	✓	✓
Fossil 1.0 [3]	✓	✗	✓	✗	✗	✗	✗	✗
F4CS [166],[167]	✓	✗	✓	✗	✓	✓	✗	✓
NLC [40]	✓	✗	✗	✗	✗	✗	✗	✓
[144, 142], [143]	✗	✗	✗	✗	✓	✓	✗	✓
[173]	✗	✗	✓	✗	✗	✗	✗	✗
[171], [172]	✗	✗	✓	✗	✗	✗	✗	✓
[139]	✗	✗	✓	✗	✗	✗	✗	✗
[98]	✓	✗	✓	✗	✗	✗	✗	✗
[54]	✓	✗	✗	✗	✗	✗	✗	✗
[80, 81]	✓	✗	✗	✗	✗	✗	✗	✓

Table 4.3: Comparison of works for automated (and sound) synthesis of certificates for continuous-time dynamical models. We show the properties verified in the respective works, and whether they are also able to verify these properties for control models (either using control certificates or controller and certificate). Similar works are grouped together.

					Fosill 1.0				Fossil 2.0			
Benchmark	N_s	Property	Neurons	Activations	min	μ	max	S	min	μ	max	S
NonPoly0	2	Stability	[5]	$[\varphi_2]$	0.04	0.21	1.58	100	0.01	0.16	1.54	100
Poly2	2	Stability	[5]	$[\varphi_2]$	0.35	11.71	70.39	90	0.08	4.77	6.50	100
Barr1	2	Safety	[10]	$[\varphi_1]$	0.34	1.00	2.72	40	0.02	0.27	0.63	100
Barr3	2	Safety	[10, 10]	$[\sigma_{\text{sig}},\sigma_{\text{sig}}]$	16.80	101.72	334.79	50	3.81	14.14	30.63	100

Table 4.4: Comparison of Fossil 1.0 vs Fossil 2.0 (the present work). Here, we use the same naming scheme for benchmarks as used in Fossil 1.0, rather than indexing by number. See Table 4.1 for details on the columns.

4.6 Discussions on Generality

4.6.1 Asymptotic Convergence to Limit Cycles

Oscillations are important and common phenomena occurring in dynamical systems, typically linked to (stable) limit cycles. Certificates for stability analysis in the presence of limit cycles, or equivalently for the asymptotic convergence to such set, have been so far notably absent from the synthesis framework in this manuscript. We discuss them next, recalling Barbashin-Krasowskii-Lasalle's Principle [103].

Theorem 4.10 (Invariance Principle [103]). *Let $\Omega \subset \mathcal{X}$ be a compact set that is positively invariant with respect to (2.5). Let $V : \mathcal{X} \to \mathbb{R}$ be a continuously differentiable function such that $\dot{V}(x) \leq 0$. Let E be the set of all points in Ω where $V(x) = 0$. Let M be the largest invariant set in E. Then every solution starting in Ω approaches M as $t \to \infty$.* ■

Crucially, the principle states that we can prove asymptotic convergence towards a set thanks to a Lyapunov-like function which is equal to zero *exactly* at the limit cycle. Such certificates have recently been studied from the perspective of disturbed models [122, 121]. Nonetheless, without prior knowledge of the existence and location of a limit cycle it is, in our experience, impractical to automatically synthesise such certificates, as they would need to be strongly templated based on the limit cycle (namely, precisely tailored to that set). Whilst in principle our approach could offer certificates for such properties, in this work we omit certificates requiring such an extensive analysis of the model dynamics.

4.6.2 Sufficiency of the Certificates and Completeness of their Synthesis

The certificates provided in this work are *sufficient* proofs for the corresponding properties. We do not in general provide guarantees that a certificate exists if the property is satisfied, i.e. *necessary* proofs. Such converse results exists for Lyapunov and barrier functions. In particular, a construction method of Lyapunov functions is known for globally exponentially stable models [103], whilst the necessity of barrier certificates is studied in, e.g., [137, 168, 140].

Contextually, whilst *sound*, our counterexample-based inductive synthesis method is not *complete*: whenever it finds a certificate, the desired property formally holds for the model under consideration. On the other hand, if our algorithm fails to find a

certificate, we cannot draw any conclusion about the validity of the property for the given model. We show that in practice our procedure consistently terminates with a successful outcome.

4.6.3 Modularity of the Synthesis and Nested Properties

We have not considered properties describing that of sequential reachability - namely trajectories arriving at a series of target sets before the goal set. We have considered properties obtained by conjunction of requirements, and we shall comment in Section 4.6.5 on instances obtained via disjunction. We could similarly obtain certificates by manipulating via propositional logic, (e.g., conjunction and disjunction, rather than temporally, as suggested previously) requirements and corresponding certificates. Such properties tie into our approach of using certificates in a *modular* fashion, which is a relatively unexplored concept and represents an area of future work.

4.6.4 Issues of Scale

Our CEGIS-based approach consists of a gradient descent based learning phase followed by an SMT dependent verification phase. While gradient descent is known to scale well to higher dimensions, nonlinear real arithmetic SMT does not in general scale well to higher dimensions and is the main computational bottleneck for our approach in higher dimensions. This problem is shared to all methodologies which rely on SMT. Possible mitigations, beyond an improvement of SMT performance, include the use of an alternative verification method, e.g., software as Marabou [100] for ReLU networks, or interval bound propagation based techniques. Issues of scale are not specific to our approach, and are shared by many approaches in the literature, including those reliant on SMT solvers.

4.6.5 Broader Connections and Taxonomy of Properties

In Section 4.3 we have presented a diverse set of certifiable properties in terms of the behaviour of trajectories (that is, of solutions) of a given dynamical model. Furthermore, in Section 4.3.8, we have laid connections across such properties through the notions of *avoid, arrive*, and *remain*. Here, we further frame such requirements in broader contexts, providing a categorisation of these properties within known classes of specifications. This section is written for the benefit of readers with a background in the mentioned areas, or with an interest in a perspective on dynamical models grounded upon formal methods [160, 29] - this section may be otherwise dispensed with, at no loss of understanding of the overall material.

First, we draw connections with formal languages (in particular, regular expressions) and with automata theory (specifically, deterministic finite automata) [90], and in passing we also informally relate to the use of temporal logic for specifications of reactive models [131, 21, 51]. However, these connections can be drawn under an important proviso: that is, in formal methods, properties by and large are expressed over finite alphabets, namely over a finite set of labels (which can be related to our sets – or complement thereof – of interest), but most importantly concern traces in discrete time. Similarly, formal languages and automata deal with finite or countably-infinite strings of (finite) characters.

Safety and Reachability - Language and Semantic Duality We remark that all the presented properties consist of combinations of two fundamental specifications in formal verification [116, 107]. The first is *safety*, which qualitatively concerns trajectories always staying clear from a nominative region in the state space that is deemed to be unsafe. Thus, safety specifications are infinite-horizon requirements, which however can be equivalently defined as requirements admitting finite-horizon counterexamples: namely, they are specifications that are necessarily invalidated by

trajectories that enter the given unsafe set in finite time.

The second specification we refer to as *reachability*, and comprise trajectories reaching a given desirable goal set (which is also known as *reach* or *target* set). Reachability is notably also employed to prove finite time termination of programs, by means of certificates known as *ranking functions* [39] that are similar to Lyapunov or RWA functions. In the area of formal languages, reachability is known to be a *co-safety* property, namely a dual to a safety requirement and, as such, it is a finite-horizon property. Indeed, in formal verification co-safe specifications admit finite-horizon witnesses, that is satisfying trajectories that enter the goal set in finite time. In Section 4.3 we have defined this as *unconstrained* reachability, and as a special instance of the reach-while avoid specification. We could have alternatively introduced it as the logical dual of safety, except that the certificate synthesis would not have followed as seamlessly. We should mention that, more generally, co-safety properties (and, in particular, reachability) are special instances of another broad class of specifications, known as *liveness* properties [21, 51]: in this work we do not deal with general liveness properties, and in particular the synthesis of sufficient certificates for this class is left as future work.

Finite- and Infinite-horizon Requirements We have discussed that safety and reachability properties are dual in the sense that the earlier raises a requirement over infinite-horizon trajectories, asking that *nothing bad ever happens*, whereas the latter requires reasoning about finite-horizon solutions, asking that over a finite time horizon *something good eventually happens* [21]. However note that, whenever safety requirements are added to finite time *reachability* properties, as in the case of the *avoid* constraint in RWA, RSWA, and RAR, these safety requirements ought to hold only over the finite time spans inherited from the reachability requirements (since the goal sets should be safe). Dually, as much as reachability requirements natively encompass

co-safety properties, in this work we have discussed extension of such reachability requirements over *infinite time* horizons: this is quite natural in control theory, namely in the context of asymptotic stability. Similarly, we have not only raised requirements that trajectories reach a goal set within a finite time horizon, which we have simply denoted as *reachability*, but additionally discussed (cf. 4.6.1) that they may approach the set (e.g., a set of equilibria, or a limit cycle) asymptotically, which we have denoted as *asymptotic reachability*.

Automata Theory Next, we qualitatively relate these classes of properties to automata theory. Through their duality, both safety and co-safety properties can be expressed by the same class of finite-state models, namely deterministic finite automata (DFA) [90]. In other words, traces within these two classes of specifications can be "compiled" by a DFA model, which "reads" them when they are started in one of its initial states, and when they terminate upon reaching one of its accepting states. Conversely, liveness properties, which generalise co-safety specifications, require automata of a different nature, such as Büchi or Rabin automata, which specifically accept infinite traces. Summarising this discussion, we can conclude that we can provide sufficient certificates for properties that can be expressed as DFAs: an enticing extension of our work is looking at richer specifications obtained by modularly composing such finite-state models.

Linear-time Properties and Temporal Logic Finally, let us draw connections to temporal logic, a class of modal logic that was natively developed to specify requirements of (models of) reactive programs [131]. Whilst originally employed for finite-state programs evolving over discrete-time steps, temporal logics, such as LTL (linear TL), have been also widely employed in the context of dynamical and control models [160, 29]. Bearing in mind that the above caveat on discrete-time semantics holds also in this context (which for instance renders its *next* operator useless for

our purposes), we can express safety requirements over safe set S as *always* S - in LTL syntax this is expressed as the formula $\Box S$ (or $\mathsf{G}S$) - and, dually, reachability requirements over target set T as *eventually* T - in LTL this is $\diamond T$ (or $\mathsf{F}T$); incidentally, the discussed duality between these requirements is crisply expressed logically as: $\Box S \equiv \neg \diamond \neg S \equiv \neg \diamond S^{\complement}$. Similar discussions follow for finite time properties, which in LTL require selecting a (finite) horizon $\tau \in \mathbb{R}$, thus yielding for example $\Box^{\leq \tau} S$. Reach-avoid can be expressed via the *until* operator U in LTL, say $S \,\mathsf{U} R$, which can be tailored to *unconstrained reachability* as $\diamond T \equiv \mathtt{true}\,\mathsf{U} T$.

Dovetailing to the discussion above, we can thus flexibly and modularly synthesise certificates also for the known weak until W, and even for the release R specifications in LTL. Indeed, recall that [21]

$$S \,\mathsf{W} T = (S \,\mathsf{U} T) \vee \Box S.$$

We can thus obtain a certificate for weak until from either a certificate for reach-avoid or one for safety, respectively. Additionally notice, in particular, that $\Box S \equiv S \,\mathsf{W}\mathtt{false}$. Additionally,

$$T \,\mathsf{R} S = \neg(\neg T \,\mathsf{U} \neg S),$$

and in fact (the next equivalent expression is easier for finding certificates

$$T \,\mathsf{R} S = (\neg T \wedge S) \mathsf{W} (T \wedge S),$$

and so, in particular, $\Box S \equiv \mathtt{false}\,\mathsf{R} S$.

4.7 Concluding Remarks

In this chapter we present a series of specifications for dynamical models which are of interest in control theory and formal verification. We collate certificates, both from

seminal and recent literature, as well as adding a new certificate for a reach-avoid-remain property. These certificates act as sufficient proofs that these properties hold for a given model. To address the subtle differences between these properties (and their corresponding certificates), we provide a taxonomy of the properties described in this chapter based on semantics of associated sets, and describe how these sets are related to trajectories using the notions of *avoid*, *arrive* and *remain*. We associate *avoid* properties with unsafe sets (and safety), *arrive* properties with goal sets (and reachability or stability), and *remain* properties with invariant sets (and equilibria).

We then describe a general framework for the verification and control of dynamical models via certificate (and control) synthesis. The framework seeks to provide a generalised approach to synthesise all of the certificates described in this chapter, alongside feedback control laws which satisfy the corresponding properties. The synthesis procedure is based on a CEGIS loop, exploiting neural networks to provide candidate functions, which are then formally verified with the help of SMT solvers. Our approach is able to efficiently synthesise certificates for a wide range of properties for both autonomous and control models, with varying complexity of dynamics and set structure. Furthermore, our approach is able to synthesise certificates which consist of multiple, modular functions, concurrently with a feedback control law, to prove that the resulting closed loop model satisfies the desired property. All of these functions are neural networks trained using gradient descent. Since these networks can represent functions of simple polynomial semantics, or more complex functions which are highly nonlinear and non-convex, our approach enables significant flexibility in the verification problems our framework can solve.

We construct a bespoke loss function for the training of these networks simultaneously, introducing a new term to encourage the feedback controller to drive trajectories towards the origin. We show empirically that this loss function improves the success rate and total synthesis time for benchmarks with properties involving finite

time reachability to a non-trivial goal set.

We develop a prototype tool based on our framework using the computation library of Fossil 1.0, and correspondingly develop a new version of this tool, Fossil 2.0. We show that our synthesis framework outperforms a state-of-the-art tool Fossil 1.0 for synthesis of Lyapunov and barrier certificates for autonomous models, both in terms of success rate and computation time. We test our framework and corresponding tool on a number of benchmarks, outperforming a state-of-the-art tool for certificate synthesis and showing robustness to initialisation. The results show that our framework is able to synthesise all of the certificates presented in this work with a high success rate and low computation time.

In future work, we hope to explore further the modular synthesis of certificates and leverage this to verify more complex properties, such as those involving sequential reachability. Furthermore, we plan to extend our framework to additional model semantics, including, for example, models with stochasticity.

Chapter 5

Concluding Remarks

In this final chapter, we summarise the contributions, findings and conclusions of this thesis, before ultimately discussing avenues and recommendations for future work.

5.1 Conclusions

It is clear that approaches from the flourishing field of machine learning will continue to filter to other domains, and the field of formal verification is no exception. Whilst formal verification has much to offer to machine learning, the converse is also true. This thesis studies how neural networks can be used in conjunction with two fundamental techniques in formal verification: abstraction and certificate-based approaches. This is studied in the context of nonlinear continuous-time dynamical models, ubiquitous models in engineering and science which can describe countless real-world physical systems. These models are by nature difficult to analyse, and the techniques developed in this thesis contribute to an enormous body of work devoted to this task.

In Chapter 3, we study the novel idea of leveraging a neural network not just as an approximation of a (concrete) dynamical model, but as a formal abstraction. This is achieved by proving that the neural network satisfies some approximation error with

respect to the concrete model, enabling the construction of a formal abstraction with nondeterminism, rooted in the concept of simulation. Constructing an abstraction therefore requires procedure for synthesising functions with formal specifications, for which we develop a novel algorithm based on CEGIS. We propose safety verification as a motivating application for neural abstractions, and present a framework in which neural abstractions are embedded to perform this task. This involves casting neural networks to equivalent models that are more amenable to analysis via existing verification tools.

As part of the experimental evaluation, we compare and contrast the performance, in terms of accuracy and computational efficiency, of neural abstractions whose underlying model semantics vary in terms of their expressiveness. We consider three different model semantics: (i) piecewise constant, (ii) piecewise affine, and (iii) nonlinear (sigmoidal). We demonstrate the effectiveness of our synthesis algorithm and the suitability of neural abstractions for safety verification on a number of benchmarks, showcasing their effectiveness at verifying concrete models which do not exhibit local Lipschitz continuity – a property that creates a challenging verification task and a common obstacle for existing verification tools.

In Chapter 4, we study the problem of proving that a dynamical model satisfies some given specification via synthesis of function-based certificates. Here, we build on existing work in the field of neural certificate synthesis. We gather and augment certificates from the literature, both recent and seminal, and develop a taxonomy to categorise these certificates, using the notions of *arrive, avoid* and *remain*, based on the nature of the properties they certify. These notions relate to how trajectories of the model behave with respect to unsafe, goal and final sets.

We then propose a novel synthesis algorithm based on CEGIS, which is able to concurrently synthesise neural feedback controllers alongside certificates (which themselves may consist of multiple neural networks). This involves constructing a

loss function to guide the certificate synthesis, including an original term so that the feedback controller encourages trajectories to converge to the goal set. The synthesis algorithm enables an approach which seeks a control law for a dynamical model that satisfies a given specification, while simultaneously synthesising a certificate for the model to prove that the specification is satisfied. We demonstrate the effectiveness of this approach on a large number of benchmarks, comparing it to an existing tool for neural certificate synthesis. We find that our approach is able to effectively synthesise controllers and certificates for a wide range of benchmarks, including models or tasks which are simple and those which are more complex, due the the flexibility enabled by the use of neural networks.

Both of the approaches presented in this thesis illustrate a key advantage of neural networks: their ability to universally approximate nonlinear functions (with some caveats of universal approximation theory, such as continuity and Lebesgue measurability). This endows verification approaches rooted in neural networks with a high degree of flexibility, and enables them to be applied to a wide range of dynamical models. This bonus is also itself a drawback: bespoke approaches for specific models may be more effective than a general approach. Furthermore, it is unclear what neural network architecture (in terms of size and activation function) is best for a given task. Given the size and complexity of neural networks and the corresponding challenge of verifying their properties, this is an important question to answer.

5.2 Future Work

Finally, we discuss a number of avenues for future work, both in the context of neural abstractions and neural certificates.

5.2.1 Neural Abstractions

Additional Model Semantics In Chapter 3, we studied the abstraction of continuous-time dynamical models with additive bounded nondeterminism. Using neural networks to abstract alternative model semantics, such as hybrid models or discrete-time models, is an open question. While the synthesis approach for such models might be similar, the suitable use-case and verification applications may differ. Furthermore, we did not consider control models for abstractions as we do in Chapter 4. It is not clear on how control inputs should be considered as part of the abstraction (do they work immediately to counter the disturbance, for example?). This is a topic for future work.

Alternative to SMT Solvers As discussed, a key limitation for synthesising neural abstractions is the issue of scale present in SMT solvers. Approaches for synthesising neural abstractions without SMT solvers would be one way to overcome this limitation. One approach would be to instead only obtain a sample-based abstraction (e.g., via the scenario approach [38]), rather than a formal one. This must then be reflected in the abstraction semantics and subsequent verification approach.

Reachability Analysis Tools The work presented in Chapter 3 is the only work to date that studies reachability analysis of neural ODEs plus additive nondeterministic disturbances. Recently, literature has grown in the study on reachability of neural ODEs themselves [117, 83]. At the time of writing, the tools developed by these complementary works do not support nondeterminism. However, handling nondeterminism is well studied and therefore a natural extension of these works and the work presented in this thesis. This would enable a more efficient approach to the reachability analysis of neural abstractions. In light of the findings in Chapter 3 (specifically, Section 3.6.1.2), in which a neural ODE is abstracted to a neural ab-

straction consisting of a smaller neural network to ease reachability analysis, this is a promising line of future work. More efficient reachability analysis tools would enable the use of neural abstractions in more complex applications, such as for robustness queries of ODEs used as classifiers.

Refinement of Neural Network Structure A fundamental question studied in Chapter 3 is how different neural abstraction semantics compare in terms of their accuracy and computational efficiency. This is important as it is not clear *a priori* which semantics are best suited to a given verification task. An alternative approach to this problem is to use an refinement procedure regarding the structure of the neural network itself. This would involve starting with a neural abstraction based on simple model semantics and a small neural network, and then iteratively refining the abstraction by increasing the expressiveness of the model semantics (via the activation function) and the size of the neural network. This would be done until the abstraction is accurate enough to verify the given specification. Abstraction refinement has been studied in classical verification approaches [50], and even recently in the context of neural networks [67], but not in the context of neural abstractions of dynamical models. It is unclear whether the width or depth of a network should be increased, or if this can be done without needing to restart the synthesis procedure from scratch.

5.2.2 Neural Certificates

Additional Model Semantics The work presented in Chapter 4 considers continuous-time dynamical models. However, certificates are studied in the context of many other model semantics, including discrete-time models, hybrid models, and stochastic models. All of the properties studied in this chapter are of interest in these contexts, and several of the corresponding certificates have direct analogues for some of these model semantics. Lyapunov functions, for instance, are used to prove stability for cer-

tain hybrid models, whereas the similar notion of supermartingales have been used to prove stability for stochastic models. Incorporating these model semantics into the framework presented would be a suitable next step. Furthermore, we have not considered the synthesis of certificates for models with additive disturbances, which are studied in Chapter 3. Under additive disturbance, stability certificates are more challenging to define and synthesise, but are of great interest in practice and would be a natural extension of the work presented here.

Modularity and Sequential Reachability The framework presented in Chapter 4 encompasses a number of properties, and does so by exploring the concept of certificate modularity. However, overall this idea could be further explored, and there are a number of avenues for future work in this area. For instance, properties describing sequential reachability may be described through the conjunction of individual *reach* properties (and certificates). The framework presented in this thesis is not limited in the number of certificates it can synthesise concurrently with a controller, and so this is a natural extension of the work presented here. This has not been implemented in practice.

Controller Constraints and Specifications Currently, the presented framework seeks to synthesise any neural network feedback controller that ensure the dynamics satisfy the required specification. We solve a verification problem, rather than a control problem, so offer no guarantees on the performance of the controller. This is likely to be limiting in practice, as there may be constraints on the controller in terms of actuation limits or cost, or other specifications on the controller in terms of performance. Incorporating actuation limits in the controller is a natural first step towards this and is achievable with many of the presented certificates. Including additional terms in the loss function to encourage the controller to seek some optimality criteria is also an avenue for future work.

Choice of Neural Network Architecture The choice of neural network architecture, both the structure of the hidden layers and the activation functions within the hidden layers, is a key consideration in the design of both controllers and certificates. In practice, this is done by arbitrary choice, and there is no clear guidance on what the best architecture is for a given problem. This is a key question to answer, and one that is not yet answered in the literature. Exploring which architectures are best suited to which problems is a natural first step towards answering this question.

Appendix A

Benchmarks of Dynamical Systems

A.1 Benchmarks for Abstraction Chapter

A.1.1 Benchmarks for Template Comparsion

The benchmarks in this section are used in Sections 3.6.1 and 3.6.4. 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.

Water Tank

$\dot{x} = 1.5 - \sqrt{x}$

$\mathcal{X}_I = [0, 0.01]$

$\mathcal{X} = [0, 2]$

Non-Lipschitz Vector Field 1 (NL1)

$\dot{x} = y$

$\dot{y} = \sqrt{x}$

$\mathcal{X}_I = [0, 0.01] \times [0, 0.01]$

$\mathcal{X} = [0, 1] \times [-1, 1]$

Non-Lipschitz Vector Field 2 (NL2)

$\dot{x} = x^2 + y$

$\dot{y} = \sqrt[3]{x^2} - x$

$\mathcal{X} = [-1, 1]^2$

$\mathcal{X}_I = [-0.005, 0.005] \times [-0.5, -0.49]$

Water Tank 4D

$\dot{x_0} = 0.2 - \sqrt{x_0}$

$\dot{x_1} = -x_1$

$\dot{x_2} = -x_2$

$\dot{x_3} = -0.25(x_0 + x_1 + x_2 + x_3)$

$\mathcal{X} = [0, 1]^4$

$\mathcal{X}_I = [0, 0.01] \times [0.8, 0.81] \times [0.8, 0.81] \times [0.8, 0.81]$

Water Tank 6D

$$\dot{x_0} = 0.2 - \sqrt{x_0}$$

$$\dot{x_1} = -x_1$$

$$\dot{x_2} = -x_2$$

$$\dot{x_3} = -x_3$$

$$\dot{x_4} = -x_4$$

$$\dot{x_5} = -\frac{1}{6}(x_0 + x_1 + x_2 + x_3 + x_4 + x_5)$$

$$\mathcal{X} = [0, 1]^6$$

$$\mathcal{X}_I = [0, 0.01] \times [0.8, 0.81] \times [0.8, 0.81] \times$$
$$[0.8, 0.81] \times [0.7, 0.71] \times [0.65, 0.66]$$

A.1.2 Benchmarks for ASM and Flow* Comparison

For the comparison with Flow* we perform a full safety verification procedure, including the computation of the intersection of the unsafe set with the reachable set. For these results in Section 3.6.3 and the comparison to the affine simplicial mesh abstractions in Section 3.6.2, we also use three Lipschitz continuous vector fields. Below, we present the vector fields and the spatial domains for each of the benchmarks; some of these are the same as the benchmarks in the previous section, but the initial sets are different and they include the unsafe set.

Water Tank

$$\dot{x} = 1.5 - \sqrt{x}$$

$$\mathcal{X}_I = [0, 0.01]$$

$$\mathcal{X}_U = \{x \mid x \geq 2\}$$

Jet Engine [19]

$$\dot{x} = -y - 1.5x^2 - 0.5x^3 - 0.1$$

$$\dot{y} = 3x - y$$

$$\mathcal{X}_I = [0.45, 0.50] \times [-0.60, -0.55]$$

$$\mathcal{X}_U = [0.3, 0.35] \times [0.5, 0.6]$$

Steam Governor [159]

$$\dot{x} = y$$

$$\dot{y} = z^2 \sin(x) \cos(x) - \sin(x) - 3y$$

$$\dot{z} = -(\cos(x) - 1)$$

$$\mathcal{X}_I = [0.70, 0.75] \times [-0.05, 0.05] \times [0.70, 0.75]$$

$$\mathcal{X}_U = [0.5, 0.6] \times [-0.4, -0.3] \times [0.7, 0.8]$$

Exponential

$$\dot{x} = -\sin(\exp(y^3 + 1)) - y^2$$

$$\dot{y} = -x$$

$$\mathcal{X}_I = [0.45, 0.5] \times [0.86, 0.91]$$

$$\mathcal{X}_U = [0.3, 0.4] \times [0.5, 0.6]$$

Non-Lipschitz Vector Field 1 (NL1)

$\dot{x} = y$

$\dot{y} = \sqrt{x}$

$\mathcal{X} = [0, 1] \times [-1, 1]$

$\mathcal{X}_I = [0, 0.05] \times [0, 0.1]$

$\mathcal{X}_U = [0.35, 0.45] \times [0.1, 0.2]$

Non-Lipschitz Vector Field 2 (NL2)

$\dot{x} = x^2 + y$

$\dot{y} = \sqrt[3]{x^2} - x$

$\mathcal{X}_I = [-0.025, 0.025] \times [-0.9, -0.85]$

$\mathcal{X}_U = [-0.05, 0.05] \times [-0.8, -0.7]$

A.2 Benchmarks for Certificates Chapter

Here, we detail the benchmarks presented in this work. These benchmarks are indexed with a number, corresponding to the row index from Table 4.1, along with the property that is being verified for the benchmark. We show the salient sets for the property; note that in some aligned it is simpler to show the safe set $\mathcal{X}_S$, and in other aligned the unsafe set $\mathcal{X}_U$. These are always the complement of each other: $\mathcal{X}_S = \mathcal{X} \setminus \mathcal{X}_U$.

We also use shorthand notation to describe sets of familiar geometric structure:

- Rectangle(lb, ub) denotes a hyper rectangle characterised by its lower bounds (lb) for each dimension, and its upper bounds (ub) in each dimension.

- Sphere(c, r) denotes a hyper sphere with centre c and radius r. The dimension of the hyper-sphere corresponds to the size of c.
- Torus(c, r_i, r_o) denotes the set defined by Sphere$(c, r_o) \setminus$ Sphere(c, r_i); we note that this is not a true torus, but rather a sphere with a spherical hole in the middle.

[1] NonPoly0 Stability

$f(x) = [x_0 x_1 - x_0,\ -x_1]$

$\mathcal{X}$: Torus$([0, 0], 1, 0.01)$

Activation: $[\varphi_2]$, Neurons: [6]

[2] Poly1 Stability

$f(x) = \left[-x_0^3 - x_0 x_2^2,\ -x_0^2 x_1 - x_1,\ 3x_0^2 x_2 - 4x_2\right]$

$\mathcal{X}$: Torus$([0.0, 0.0, 0.0], 10.0, 0.1)$

Activation: $[\varphi_2]$, Neurons: [8]

[3] Benchmark1 Stability

$f(x) = [u_0 + x_0 + x_1,\ u_1 - x_0 - x_1]$

$\mathcal{X}$: Torus$([0.0, 0.0], 10.0, 0.1)$

Activation: $[\varphi_2]$, Neurons: [4] CTRLActivation: $[\varphi_1]$, Neurons: [15, 2]

[4] InvertedPendulum Stability

$$f(x) = \left[u_0 + x_1,\ u_1 - \frac{8}{3}x_1 + 19.62 \sin(x_0)\right]$$

$\mathcal{X}$: Torus$([0.0, 0.0], 1, 0.1)$

Activation: $[\varphi_2]$, Neurons: [5] CTRLActivation: $[\varphi_1]$, Neurons: [25, 2]

[5] NonPoly1 ROA

$f(x) = \left[2x_0^2 x_1 - x_0,\ -x_1\right]$

$\mathcal{X}_I$: (Sphere($[-1, 1], 0.1$) $\cup$ Sphere($[1, -1], 0.2$))

Activation: [σ_{soft}], Neurons: [5]

[6] LorenzSystem ROA

$$f(x) = \left[u_0 - 10.0x_0 + 10.0x_1,\ u_1 - x_0x_2 + 28.0x_0 - x_1,\ u_2 + x_0x_1 - \frac{8}{3}x_2\right]$$

$\mathcal{X}_I$: Sphere($[0, 0, 0], 0.3$)

Activation: [φ_2], Neurons: [8] CTRLActivation: [φ_1], Neurons: [8, 3]

[7] Barr2 Safety

$f(x) = \left[x_1 - 1 + e^{-x_0},\ -\sin^2(x_0)\right]$

$\mathcal{X} : x_0 \geq -2 \wedge x_1 \leq 2$

$\mathcal{X}_I$: Sphere($[-0.5, 0.5], 0.4$)

$\mathcal{X}_U$: Sphere($[0.7, -0.7], 0.3$)

Activation: [σ_{t}], Neurons: [15]

[8] ObstacleAvoidance Safety

$$f(x) = \left[\sin(x_2),\ \cos(x_2),\ \frac{3x_0\sin(x_2) + 3x_1\cos(x_2)}{x_0^2 + x_1^2 + 0.5} - \sin(x_2)\right]$$

$\mathcal{X} : x_0 \geq -2 \wedge x_1 \geq -2 \wedge x_2 \geq -1.57 \wedge x_0 \leq 2 \wedge x_1 \leq 2 \wedge x_2 \leq 1.57$

$\mathcal{X}_I : x_0 \geq -0.1 \wedge x_1 \geq -2 \wedge x_2 \geq -0.52 \wedge x_0 \leq 0.1 \wedge x_1 \leq -1.8 \wedge x_2 \leq 0.52$

$\mathcal{X}_U : x_0^2 + x_1^2 \leq 0.04$

Activation: [φ_4], Neurons: [25]

[9] HighOrd8 Safety

$$f(x) = [x_1, x_2, x_3, x_4, x_5, x_6, x_7,$$
$$-576x_0 - 2400x_1 - 4180x_2 - 3980x_3 - 2273x_4 - 800x_5 - 170x_6 - 20x_7]$$
$$\mathcal{X} : \text{Rectangle}([-2.2, -2.2, -2.2, -2.2, -2.2, -2.2, -2.2, -2.2],$$
$$[2.2, 2.2, 2.2, 2.2, 2.2, 2.2, 2.2, 2.2])$$
$$\mathcal{X}_I : \text{Rectangle}([0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9],$$
$$[1.1, 1.1, 1.1, 1.1, 1.1, 1.1, 1.1, 1.1])$$
$$\mathcal{X}_U : \text{Rectangle}([-2.2, -2.2, -2.2, -2.2, -2.2, -2.2, -2.2, -2.2],$$
$$[-1.8, -1.8, -1.8, -1.8, -1.8, -1.8, -1.8, -1.8])$$

Activation: $[\varphi_1]$, Neurons: [10]

[10] CtrlObstacleAvoidance Safety

$$f(x) = [\sin(x_2),\ \cos(x_2),\ u_0 - \sin(x_2)]$$

$$\mathcal{X} : \text{Rectangle}([-10.0, -10.0, -\pi], [10.0, 10.0, \pi])$$

$$\mathcal{X}_I : \text{Rectangle}([4.0, 4.0, \frac{-\pi}{2}], [6.0, 6.0, \frac{\pi}{2}])$$

$$\mathcal{X}_U : \text{Rectangle}([-9, -9, \frac{-\pi}{2}], [-7.0, -6.0, \frac{\pi}{2}])$$

Activation: $[\sigma_t]$, Neurons: [15] CTRLActivation: $[\sigma_t]$, Neurons: [5, 1]

[11] NonPoly3 SWA

$$f(x) = \left[-0.1x_0x_1^3 - 3x_0,\ -x_1 + x_2,\ -x_2\right]$$

$$\mathcal{X} : \text{Torus}([0, 0, 0], 3, 0.01)$$

$$\mathcal{X}_I : \text{Sphere}([-0.9, -0.9, -0.9], 1.0)$$

$$\mathcal{X}_U : (\text{Sphere}([0.4, 0.4, 0.4], 0.2) \cup \text{Sphere}([-0.4, 0.4, 0.4], 0.2))$$

Activation: $[\varphi_2]$, Neurons: [6] AltActivation: $[\sigma_t]$, AltNeurons: [5]

[12] Barr3 SWA

$$f(x) = \left[x_1,\ \frac{1}{3}x_0^3 - x_0 - x_1\right]$$

$\mathcal{X}$: Rectangle($[-3, -2], [2.5, 1]$)

$\mathcal{X}_I$: Rectangle($[0.4, 0.1], [0.8, 0.5]$)

$\mathcal{X}_U$: Sphere($[-1, -1], 0.4$)

Activation: [φ_2], Neurons: [5] AltActivation: [σ_{sig},φ_2], AltNeurons: [5, 5]

[13] SecondOrder SWA

$f(x) = \left[-x_0^3 + x_1,\ u_0\right]$

$\mathcal{X}$: Rectangle($[-1.5, -1.5], [1.5, 1.5]$)

$\mathcal{X}_I$: Rectangle($[-0.5, -0.5], [0.5, 0.5]$)

$\mathcal{X}_U$: Complement(Rectangle($[-1, -1], [1, 1]$))

Activation: [φ_2], Neurons: [8] AltActivation: [φ_2], AltNeurons: [5] CTRLActivation: [φ_1], Neurons: [8, 1]

[14] ThirdOrder SWA

$$f(x) = \left[u_0 - 10x_0 + 10x_1,\ -x_0x_2 + 28x_0 - x_1,\ x_0x_1 - \frac{8}{3}x_2\right]$$

$\mathcal{X}$: Rectangle($[-6, -6, -6], [6, 6, 6]$)

$\mathcal{X}_I$: Rectangle($[-1.2, -1.2, -1.2], [1.2, 1.2, 1.2]$)

$\mathcal{X}_U$: Complement(Rectangle($[-5, -5, -5], [5, 5, 5]$))

Activation: [φ_2], Neurons: [10] AltActivation: [σ_{t}], AltNeurons: [8] CTRLActivation: [φ_1], Neurons: [8, 1]

[15] SecondOrderLQR RWA

$f(x) = \left[-x_0^3 + x_1,\ -1.0x_0 - 1.73x_1\right]$

$\mathcal{X}$: Rectangle($[-1.5, -1.5], [1.5, 1.5]$)

$\mathcal{X}_I$: Rectangle($[-0.5, -0.5], [0.5, 0.5]$)

$\mathcal{X}_S$: Rectangle($[-1, -1], [1, 1]$)

$\mathcal{X}_G$: Rectangle($[-0.05, -0.05], [0.05, 0.05]$)

Activation: [φ_2], Neurons: [4]

[16] ThirdOrderLQR RWA

$$f(x) = \left[-33.71x_0 - 8.49x_1,\ -x_0x_2 + 28x_0 - x_1,\ x_0x_1 - \frac{8}{3}x_2\right]$$

$\mathcal{X}$: Rectangle($[-6, -6, -6], [6, 6, 6]$)

$\mathcal{X}_I$: Rectangle($[-1.2, -1.2, -1.2], [1.2, 1.2, 1.2]$)

$\mathcal{X}_S$: Rectangle($[-5, -5, -5], [5, 5, 5]$)

$\mathcal{X}_G$: Rectangle($[-0.3, -0.3, -0.3], [0.3, 0.3, 0.3]$)

Activation: [φ_2], Neurons: [16]

[17] SecondOrder RWA

$$f(x) = \left[-x_0^3 + x_1,\ u_0\right]$$

$\mathcal{X}$: Rectangle($[-1.5, -1.5], [1.5, 1.5]$)

$\mathcal{X}_I$: Rectangle($[-0.5, -0.5], [-0.1, -0.1]$)

$\mathcal{X}_S$: (Rectangle($[-1.5, -1.5], [1.5, 1.5]$) \ Sphere($[0.5, 0.5], 0.2$))

$\mathcal{X}_G$: Rectangle($[-0.05, -0.05], [0.05, 0.05]$)

Activation: [σ_{sig},φ_2], Neurons: [4, 4] CTRLActivation: [φ_1], Neurons: [8, 1]

[18] ThirdOrder RWA

$$f(x) = \left[u_0 - 10x_0 + 10x_1,\ -x_0x_2 + 28x_0 - x_1,\ x_0x_1 - \frac{8}{3}x_2\right]$$

$\mathcal{X}$: Rectangle($[-6, -6, -6], [6, 6, 6]$)

$\mathcal{X}_I$: Rectangle($[-1.2, -1.2, -1.2], [1.2, 1.2, 1.2]$)

$\mathcal{X}_S$: Rectangle($[-5, -5, -5], [5, 5, 5]$)

$\mathcal{X}_G$: Rectangle($[-0.3, -0.3, -0.3], [0.3, 0.3, 0.3]$)

Activation: [φ_2], Neurons: [5] CTRLActivation: [φ_1], Neurons: [8, 1]

[19] InvertedPendulum RWA

$$f(x) = \left[u_0 + x_1,\ u_1 - \frac{8}{3}x_1 + 19.62\sin(x_0)\right]$$

$\mathcal{X}$: Rectangle($[-3,-3],[3,3]$)

$\mathcal{X}_I$: Rectangle($[-0.6,-0.6],[0.6,0.6]$)

$\mathcal{X}_S$: Rectangle($[-2.5,-2.5],[2.5,2.5]$)

$\mathcal{X}_G$: Rectangle($[-0.01,-0.01],[0.01,0.01]$)

Activation: [σ_{sig}], Neurons: [5] CTRLActivation: [φ_1], Neurons: [8, 2]

[20] SecondOrderLQR RSWA

$$f(x) = \left[-x_0^3 + x_1,\ -1.0x_0 - 1.73x_1\right]$$

$\mathcal{X}$: Rectangle($[-1.5,-1.5],[1.5,1.5]$)

$\mathcal{X}_I$: Rectangle($[-0.5,-0.5],[0.5,0.5]$)

$\mathcal{X}_S$: Rectangle($[-1,-1],[1,1]$)

$\mathcal{X}_F$: Rectangle($[-0.05,-0.05],[0.05,0.05]$)

Activation: [φ_2], Neurons: [4]

[21] ThirdOrderLQR RSWA

$$f(x) = \left[-33.71x_0 - 8.49x_1,\ -x_0x_2 + 28x_0 - x_1,\ x_0x_1 - \frac{8}{3}x_2\right]$$

$\mathcal{X}$: Rectangle($[-6,-6,-6],[6,6,6]$)

$\mathcal{X}_I$: Rectangle($[-1.2,-1.2,-1.2],[1.2,1.2,1.2]$)

$\mathcal{X}_S$: Rectangle($[-5,-5,-5],[5,5,5]$)

$\mathcal{X}_F$: Rectangle($[-0.3,-0.3,-0.3],[0.3,0.3,0.3]$)

Activation: [φ_2], Neurons: [16]

[22] InvertedPendulumLQR RSWA

$f(x) = [-7.21x_0 - 0.34x_1,\ -1.34x_0 - 2.997x_1 + 19.62\sin(x_0)]$

$\mathcal{X}$: Rectangle($[-3, -3], [3, 3]$)

$\mathcal{X}_I$: Rectangle($[-0.6, -0.6], [0.6, 0.6]$)

$\mathcal{X}_S$: Rectangle($[-2.5, -2.5], [2.5, 2.5]$)

$\mathcal{X}_F$: Rectangle($[-0.3, -0.3], [0.3, 0.3]$)

Activation: [σ_{sig},φ_2], Neurons: [5, 5]

[23] SecondOrder RSWA

$f(x) = \left[-x_0^3 + x_1,\ u_0\right]$

$\mathcal{X}$: Rectangle($[-1.5, -1.5], [1.5, 1.5]$)

$\mathcal{X}_I$: Rectangle($[-0.5, -0.5], [0.5, 0.5]$)

$\mathcal{X}_S$: Rectangle($[-1, -1], [1, 1]$)

$\mathcal{X}_F$: Rectangle($[-0.05, -0.05], [0.05, 0.05]$)

Activation: [φ_2], Neurons: [8] CTRLActivation: [φ_1], Neurons: [8, 1]

[24] InvertedPendulum RSWA

$$f(x) = \left[u_0 + x_1,\ u_1 - \frac{8}{3}x_1 + 19.62\sin(x_0)\right]$$

$\mathcal{X}$: Rectangle($[-3, -3], [3, 3]$)

$\mathcal{X}_I$: Rectangle($[-0.6, -0.6], [0.6, 0.6]$)

$\mathcal{X}_S$: Rectangle($[-2.5, -2.5], [2.5, 2.5]$)

$\mathcal{X}_F$: Rectangle($[-0.3, -0.3], [0.3, 0.3]$)

Activation: [σ_{sig},φ_2], Neurons: [5, 5] CTRLActivation: [φ_1], Neurons: [8, 2]

[25] SecondOrderLQR RAR

$f(x) = \left[-x_0^3 + x_1,\ -1.0x_0 - 1.73x_1\right]$

$\mathcal{X}$: Rectangle($[-3.5, -3.5], [3.5, 3.5]$)

$\mathcal{X}_I$: Rectangle($[-2, -2], [2, 2]$)

$\mathcal{X}_S$: Rectangle($[-3, -3], [3, 3]$)

$\mathcal{X}_G$: Rectangle($[-0.1, -0.1], [0.1, 0.1]$)

$\mathcal{X}_F$: Rectangle($[-0.15, -0.15], [0.15, 0.15]$)

Activation: [σ_{soft}], Neurons: [6] AltActivation: [φ_2], AltNeurons: [6]

[26] InvertedPendulum RAR

$$f(x) = \left[u_0 + x_1,\ u_1 - \frac{8}{3}x_1 + 19.62\sin\left(x_0\right)\right]$$

$\mathcal{X}$: Rectangle($[-3.5, -3.5], [3.5, 3.5]$)

$\mathcal{X}_I$: Rectangle($[-2, -2], [2, 2]$)

$\mathcal{X}_S$: Rectangle($[-3, -3], [3, 3]$)

$\mathcal{X}_G$: Rectangle($[-0.1, -0.1], [0.1, 0.1]$)

$\mathcal{X}_F$: Rectangle($[-0.2, -0.2], [0.2, 0.2]$)

Activation: [σ_{sig},φ_2], Neurons: [6, 6] AltActivation: [σ_{sig},φ_2], AltNeurons: [6, 6] CTRLActivation: [φ_1], Neurons: [8, 2]

Bibliography

[1] Abate, A., Bessa, I., Cattaruzza, D., Cordeiro, L., David, C., Kesseli, P., Kroening, D., Polgreen, E.: Automated formal synthesis of provably safe digital controllers for continuous plants. Acta Informatica **57**(3), 223–244 (2020)

[2] Abate, A.: Formal verification of complex systems: Model-based and data-driven methods. MEMOCODE 2017 - 15th ACM-IEEE International Conference on Formal Methods and Models for System Design pp. 91–93 (2017)

[3] 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. HSCC '21 (May 2021)

[4] Abate, A., Ahmed, D., Giacobbe, M., Peruffo, A.: Formal Synthesis of Lyapunov Neural Networks. IEEE Control Systems Letters **5**(3), 773–778 (Jul 2021)

[5] Abate, A., Bogomolov, S., Edwards, A., Potomkin, K., Soudjani, S., Zuliani, P.: Safe reach set computation via neural barrier certificates. IFAC **58**(11), 107–114 (2024), 8th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2024

[6] Abate, A., Edwards, A., Giacobbe, M.: Neural abstractions. In: Thirty-Sixth Conference on Neural Information Processing Systems (2022)

[7] Abate, A., Edwards, A., Giacobbe, M., Punchihewa, H., Roy, D.: Quantitative verification with neural networks. In: 34th International Conference on Concurrency Theory (CONCUR) (2023)

[8] Ahmed, D., Peruffo, A., Abate, A.: Automated and sound synthesis of lyapunov functions with SMT solvers. In: Biere, A., Parker, D. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 97–114. Springer International Publishing, Cham (2020)

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

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

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

[12] Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical Computer Science **138**(1), 3–34 (1995)

[13] Alur, R., Henzinger, T., Ho, P.H.: Automatic symbolic verification of embedded systems. IEEE Transactions on Software Engineering **22**(3), 181–201 (1996)

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

[15] Ames, A.D., Xu, X., Grizzle, J.W., Tabuada, P.: Control Barrier Function Based Quadratic Programs for Safety Critical Systems. IEEE Transactions on Automatic Control **62**(8), 3861–3876 (Aug 2017)

[16] 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)

[17] 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)

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

[19] 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)

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

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

[22] Bak, S., Bogomolov, S., Duggirala, P.S., Gerlach, A.R., Potomkin, K.: Reachability of black-box nonlinear systems after Koopman operator linearization. IFAC-PapersOnLine **54**(5), 253–258 (2021), 7th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2021

[23] 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)

[24] Bak, S., Bogomolov, S., Johnson, T.T.: HYST: a source transformation and translation tool for hybrid automaton models. In: Girard, A., Sankaranarayanan, S. (eds.) Proceedings of the 18th International Conference on Hy-

brid Systems: Computation and Control, HSCC'15, Seattle, WA, USA, April 14-16, 2015. pp. 128–133. ACM (2015)

[25] Bak, S., Tran, H.D., Hobbs, K., Johnson, T.T.: Improved geometric path enumeration for verifying relu neural networks. In: Proceedings of the 32nd International Conference on Computer Aided Verification. Springer (2020)

[26] 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)

[27] Barrett, C., Stump, A., Tinelli, C., et al.: The smt-lib standard: Version 2.0. In: Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, UK). vol. 13, p. 14 (2010)

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

[29] Belta, C., Yordanov, B., Gol, E.A.: Formal methods for discrete-time dynamical systems, vol. 15. Springer (2017)

[30] Ben Sassi, M.A., Sankaranarayanan, S., Chen, X., Ábrahám, E.: Linear relaxations of polynomial positivity for polynomial Lyapunov function synthesis. IMA Journal of Mathematical Control and Information **33**(3), 723–756 (Sep 2016)

[31] Blanchini, F., Miani, S.: Set-Theoretic Methods in Control. Birkhäuser Boston, Boston, MA (2008)

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

[33] Bogomolov, S., Frehse, G., Giacobbe, M., Henzinger, T.A.: Counterexample-Guided Refinement of Template Polyhedra. In: Legay, A., Margaria, T. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, vol. 10205, pp. 589–606. Springer Berlin Heidelberg, Berlin, Heidelberg (2017)

[34] 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)

[35] 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)

[36] Bohrer, R., Tan, Y.K., Mitsch, S., Myreen, M.O., Platzer, A.: Veriphy: verified controller executables from verified cyber-physical system models. In: ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018. pp. 617–630 (2018)

[37] Bunel, R., Lu, J., Turkaslan, I., Torr, P.H.S., Kohli, P., Kumar, M.P.: Branch and bound for piecewise linear neural network verification. Journal of Machine Learning Research **21**, 42:1–42:39 (2019)

[38] Calafiore, G., Campi, M.: The scenario approach to robust control design. IEEE Transactions on Automatic Control (2006)

[39] Chakarov, A., Sankaranarayanan, S.: Probabilistic program analysis with martingales. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification. pp. 511–526. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

[40] Chang, Y.C., Roohi, N., Gao, S.: Neural lyapunov control. Advances in neural information processing systems **32** (2019)

[41] Chen, S., Fazlyab, M., Morari, M., Pappas, G.J., Preciado, V.M.: Learning Lyapunov Functions for Piecewise Affine Systems with Neural Network Controllers. arXiv:2008.06546 [math] (Aug 2020)

[42] 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)

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

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

[45] 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)

[46] 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)

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

[48] 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)

[49] Choi, S.W., Ivashchenko, M., Nguyen, L.V., Tran, H.D.: Reachability analysis of sigmoidal neural networks. ACM Trans. Embed. Comput. Syst. (Oct 2023)

[50] Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-Guided Abstraction Refinement. In: Emerson, E.A., Sistla, A.P. (eds.) Computer Aided Verification. pp. 154–169. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2000)

[51] Clarke, E., Grumberg, O., Kroening, D., Peled, D., Veith, H.: Principles of model checking. MIT Press, 2 edn. (2018)

[52] Coddington, E.A., Levinson, N.: Theory of Ordinary Differential Equations. McGraw-Hill (1955)

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

[54] 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)

[55] 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)

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

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

[58] Dawson, C., Gao, S., Fan, C.: Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods. arXiv preprint arXiv:2202.11762 (2022)

[59] de Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. 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., Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, vol. 4963, pp. 337–340. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)

[60] Debauche, V., Edwards, A., Abate, A., Jungers, R.M.: Stability analysis of switched linear systems with neural lyapunov functions. Proceedings of the AAAI Conference on Artificial Intelligence (2024)

[61] Doyen, L., Henzinger, T.A., Raskin, J.F.: Automatic rectangular refinement of affine hybrid systems. In: Formal Modeling and Analysis of Timed Systems (2005)

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

[63] Edwards, A., Giacobbe, M., Abate, A.: On the trade-off between efficiency and precision of neural abstraction. In: Quantitative Evaluation of Systems (2023)

[64] Edwards, A., Peruffo, A., Abate, A.: Fossil 2.0: Formal Certificate Synthesis for the Verification and Control of Dynamical Models. In: Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control. HSCC '24, Association for Computing Machinery, New York, NY, USA (2024)

[65] Edwards, A., Peruffo, A., Abate, A.: A General Verification Framework for Dynamical and Control Models via Certificate Synthesis (2024), under review, arXiv:2309.06090

[66] Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: Automated Technology for Verification and Analysis (2017)

[67] Elboher, Y.Y., Gottschlich, J., Katz, G.: An Abstraction-Based Framework for Neural Network Verification. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification. vol. 12224, pp. 43–65. Springer International Publishing, Cham (2020)

[68] 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)

[69] Frehse, G., Krogh, B., Rutenbar, R.: Verifying analog oscillator circuits using forward/backward abstraction refinement. In: Proceedings of the Design Automation & Test in Europe Conference (2006)

[70] Frehse, G.: PHAVer: Algorithmic verification of hybrid systems past HyTech. International Journal on Software Tools for Technology Transfer **10**(3), 263–279 (Jun 2008)

[71] 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)

[72] Frehse, G., Kateja, R., Le Guernic, C.: Flowpipe approximation and clustering in space-time. In: Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control - HSCC '13. p. 203. ACM Press, Philadelphia, Pennsylvania, USA (2013)

[73] 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)

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

[75] Gao, S., Kapinski, J., Deshmukh, J., Roohi, N., Solar-Lezama, A., Arechiga, N., Kong, S.: Numerically-robust inductive proof rules for continuous dynamical systems. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification. pp. 137–154. Springer International Publishing, Cham (2019)

[76] 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, Lecture Notes in Computer Science, vol. 7898, pp. 208–214. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

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

[78] Girard, A., Pappas, G.J.: Approximation Metrics for Discrete and Continuous Systems. IEEE Transactions on Automatic Control **52**(5), 782–798 (May 2007)

[79] Goubault, E., Jourdan, J.H., Putot, S., Sankaranarayanan, S.: Finding Non-Polynomial Positive Invariants and Lyapunov Functions for Polynomial Systems through Darboux Polynomials. In: American Control Conference (ACC). Portland, United States (Jun 2014)

[80] Grande, D., Anderlini, E., Peruffo, A., Salavasidis, G.: Augmented neural lyapunov control. IEEE Access (2023)

[81] Grande, D., Fenucci, D., Peruffo, A., Anderlini, E., Phillips, A.B., Giles, T., Salavasidis, G.: Systematic synthesis of passive fault-tolerant augmented neural lyapunov control laws for nonlinear systems. In: 2023 62nd IEEE Conference on Decision and Control (CDC) (Dec 2023)

[82] 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)

[83] 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)

[84] Grumberg, O., Schuster, A., Yadgar, A.: Memory efficient all-solutions SAT solver and its application for reachability analysis. In: Proceedings of the 5th International Conference on Formal Methods in Computer-Aided Design (FMCAD) (Nov 2004)

[85] Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2021)

[86] Henzinger, T.: The theory of hybrid automata. In: Proceedings 11th Annual IEEE Symposium on Logic in Computer Science. pp. 278–292 (Jul 1996)

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

[88] Henzinger, T.A., Ho, P.H., Wong-Toi, H.: HYTECH: A model checker for hybrid systems. International Journal on Software Tools for Technology Transfer **1**(1-2), 110–122 (Dec 1997)

[89] 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)

[90] Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley Publishing (1979)

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

[92] 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)

[93] 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)

[94] Huang, Z., Wang, Y., Mitra, S., Dullerud, G.E., Chaudhuri, S.: Controller synthesis with inductive proofs for piecewise linear systems: An smt-based algorithm. In: 2015 54th IEEE conference on decision and control (CDC). pp. 7434–7439. IEEE (2015)

[95] 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)

[96] Ivanov, R., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig: Verifying safety properties of hybrid systems with neural network controllers. In: HSCC. pp. 169–178. ACM (2019)

[97] Jin, W., Wang, Z., Yang, Z., Mou, S.: Neural Certificates for Safe Control Policies. arXiv:2006.08465 [cs, eess] (Jun 2020)

[98] 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)

[99] Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) Computer Aided Verification. pp. 97–117. Springer International Publishing, Cham (2017)

[100] Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.: The Marabou Framework for Verification and Analysis of Deep Neural Networks. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification. pp. 443–452. Lecture Notes in Computer Science, Springer International Publishing, Cham (2019)

[101] 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)

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

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

[104] Knight, J.C.: Safety critical systems: challenges and directions. In: ICSE. pp. 547–550. ACM (2002)

[105] 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)

[106] 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)

[107] Kupferman, O., Vardi, M.: Model checking of safety properties. Formal Methods in System Design **19** (1999)

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

[109] 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 (Jan 1993)

[110] 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)

[111] 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)

[112] Lu, Z., Pu, H., Wang, F., Hu, Z., Wang, L.: The expressive power of neural networks: A view from the width. In: NIPS. vol. 30, pp. 6231–6239. Curran Associates, Inc. (2017)

[113] LYAPUNOV, A.M.: The general problem of the stability of motion. International Journal of Control **55**(3), 531–534 (Mar 1992)

[114] MacKay, D.J.C.: Information Theory, Inference, and Learning Algorithms. Cambridge University Press, Cambridge, UK ; New York (2003)

[115] 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)

[116] Manna, Z., Pnueli, A.: A hierarchy of temporal properties. In: Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing. p. 377–410. PODC '90 (1990)

[117] 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)

[118] Manzanas Lopez, D., Musau, P., Hamilton, N.P., Johnson, T.T.: Reachability analysis of a general class of neural ordinary differential equations. In: Bogomolov, S., Parker, D. (eds.) Formal Modeling and Analysis of Timed Systems. pp. 258–277. Springer International Publishing, Cham (2022)

[119] Masti, D., Fabiani, F., Gnecco, G., Bemporad, A.: Counter-example guided inductive synthesis of control lyapunov functions for uncertain systems. IEEE Control Systems Letters (2023)

[120] McMillan, K.L.: Applying SAT methods in unbounded symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) Computer Aided Verification (2002)

[121] Meng, Y., Li, Y., Fitzsimmons, M., Liu, J.: Smooth Converse Lyapunov-Barrier Theorems for Asymptotic Stability with Safety Constraints and Reach-Avoid-Stay Specifications (Dec 2021)

[122] Meng, Y., Li, Y., Liu, J.: Control of Nonlinear Systems with Reach-Avoid-Stay Specifications: A Lyapunov-Barrier Approach with an Application to the Moore-Greizer Model. In: 2021 American Control Conference (ACC). pp. 2284–2291 (May 2021)

[123] Miranda, L.J.V.: PySwarms, a research-toolkit for Particle Swarm Optimization in Python. Journal of Open Source Software **3** (2018)

[124] Monniaux, D.: A survey of satisfiability modulo theory. In: Gerdt, V.P., Koepf, W., Seiler, W.M., Vorozhtsov, E.V. (eds.) Computer Algebra in Scientific Computing. pp. 401–425. Springer International Publishing, Cham (2016)

[125] 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)

[126] Noroozi, N., Karimaghaee, P., Safaei, F., Javadi, H.: Generation of lyapunov functions by neural networks. In: Proceedings of the World Congress on Engineering. vol. 2008 (2008)

[127] Papachristodoulou, A., Prajna, S.: On the construction of Lyapunov functions using the sum of squares decomposition. In: Proceedings of the 41st IEEE Conference on Decision and Control, 2002. vol. 3, pp. 3482–3487. IEEE, Las Vegas, NV, USA (2002)

[128] Papachristodoulou, A., Anderson, J., Valmorbida, G., Prajna, S., Seiler, P., Parrilo, P.: SOSTOOLS Version 3.00 Sum of Squares Optimization Toolbox for MATLAB. arXiv:1310.4716 [cs, math] (Oct 2013)

[129] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E.Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: an imperative style, high-performance deep learning library. In: NeurIPS. pp. 8024–8035 (2019)

[130] Peruffo, A., Ahmed, D., Abate, A.: Automated and formal synthesis of neural barrier certificates for dynamical models. In: Groote, J.F., Larsen, K.G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 370–388. Springer International Publishing, Cham (2021)

[131] Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science. pp. 46–57 (1977)

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

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

[134] 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)

[135] Prajna, S., Jadbabaie, A., Pappas, G.: Stochastic safety verification using barrier certificates. In: 2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601). pp. 929–934 Vol.1. IEEE, Nassau, Bahamas (2004)

[136] Prajna, S.: Barrier certificates for nonlinear model validation. Automatica (Journal of IFAC) **42**(1), 117–126 (Jan 2006)

[137] Prajna, S., Rantzer, A.: On the necessity of barrier certificates. IFAC Proceedings Volumes **38**(1), 526–531 (2005)

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

[139] Ratschan, S.: Simulation based computation of certificates for safety of dynamical systems. In: Formal Modeling and Analysis of Timed Systems: 15th International Conference, FORMATS 2017, Berlin, Germany, September 5–7, 2017, Proceedings 15. pp. 303–317. Springer (2017)

[140] Ratschan, S.: Converse Theorems for Safety and Barrier Certificates. IEEE Transactions on Automatic Control **63**(8), 2628–2632 (Aug 2018)

[141] Ratschan, S., She, Z.: Providing a Basin of Attraction to a Target Region of Polynomial Systems by Computation of Lyapunov-Like Functions. SIAM Journal on Control and Optimization **48**(7), 4377–4394 (Jan 2010)

[142] 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. IEEE (Dec 2015)

[143] Ravanbakhsh, H., Sankaranarayanan, S.: Counterexample Guided Synthesis of Switched Controllers for Reach-While-Stay Properties. arXiv:1505.01180 [cs] (Sep 2015)

[144] Ravanbakhsh, H., Sankaranarayanan, S.: Learning control lyapunov functions from counterexamples and demonstrations. Autonomous Robots **43**(2), 275–307 (Feb 2019)

[145] Richards, S.M., Berkenkamp, F., Krause, A.: The Lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems. In: Conference on Robot Learning. pp. 466–476. PMLR (2018)

[146] Romdlony, M.Z., Jayawardhana, B.: Stabilization with guaranteed safety using control lyapunov–barrier function. Automatica **66**, 39–47 (2016)

[147] 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)

[148] Samanipour, P., Poonawala, H.A.: Stability analysis and controller synthesis using single-hidden-layer relu neural networks. IEEE Transactions on Automatic Control pp. 1–12 (2023)

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

[150] Sankaranarayanan, S., Chen, X., Ábrahám, E.: Lyapunov Function Synthesis using Handelman Representations. IFAC Proceedings Volumes **46**(23), 576–581 (2013)

[151] 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)

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

[153] Scheibler, K., Neubauer, F., Mahdi, A., Fränzle, M., Teige, T., Bienmüller, T., Fehrer, D., Becker, B.: Accurate icp-based floating-point reasoning. In: 2016 Formal Methods in Computer-Aided Design (FMCAD). pp. 177–184 (2016)

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

[155] 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)

[156] She, Z., Li, H., Xue, B., Zheng, Z., Xia, B.: Discovering polynomial Lyapunov functions for continuous dynamical systems. Journal of Symbolic Computation **58**, 41–63 (Nov 2013)

[157] She, Z., Xia, B., Xiao, R., Zheng, Z.: A semi-algebraic approach for asymptotic stability analysis. Nonlinear Analysis: Hybrid Systems (2009)

[158] Solar-Lezama, A., Tancau, L., Bodik, R., Seshia, S., Saraswat, V.: Combinatorial sketching for finite programs. SIGOPS Oper. Syst. Rev. **40**(5), 404–415 (Oct 2006)

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

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

[161] Tan, X., Cortez, W.S., Dimarogonas, D.V.: High-order barrier functions: Robustness, safety, and performance-critical control. IEEE Transactions on Automatic Control **67**(6), 3021–3028 (2022)

[162] 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)

[163] 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)

[164] Van Der Schaft, A.J., Schumacher, H.: An introduction to hybrid dynamical systems, Lecture Notes in Control and Information Sciences, vol. 251. Springer, London (2007)

[165] Vannelli, A., Vidyasagar, M.: Maximal Lyapunov functions and domains of attraction for autonomous nonlinear systems. Automatica **21**(1), 69–80 (Jan 1985)

[166] Verdier, C.F., Mazo, M.: Formal synthesis of analytic controllers for sampled-data systems via genetic programming. In: 2018 IEEE Conference on Decision and Control (CDC). pp. 4896–4901 (2018)

[167] Verdier, C.F., Mazo Jr, M.: Formal controller synthesis for hybrid systems using genetic programming. arXiv:2003.14322 [cs, eess] (Sep 2020)

[168] Wisniewski, R., Sloth, C.: Converse barrier certificate theorems. IEEE Transactions on Automatic Control **61**(5), 1356–1361 (2015)

[169] Wu, Z., Albalawi, F., Zhang, Z., Zhang, J., Durand, H., Christofides, P.D.: Control Lyapunov-Barrier function-based model predictive control of nonlinear systems. Automatica **109**, 108508 (Nov 2019)

[170] 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)

[171] Zhao, H., Zeng, X., Chen, T., Liu, Z.: Synthesizing barrier certificates using neural networks. In: Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control. pp. 1–11. HSCC '20, Association for Computing Machinery, New York, NY, USA (Apr 2020)

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

[173] Zhao, Q., Chen, X., Zhang, Y., Sha, M., Yang, Z., Lin, W., Tang, E., Chen, Q., Li, X.: Synthesizing ReLU neural networks with two hidden layers as barrier certificates for hybrid systems. In: Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control. pp. 1–11. Association for Computing Machinery, New York, NY, USA (May 2021)