Theory of Hybrid Systems Informatik 2
RWTH AACHEN UNIVERSITY

The present work was submitted to the LuFG Theory of Hybrid Systems

MASTER OF SCIENCE THESIS

VERIFYING AI-CONTROLLED HYBRID SYSTEMS

Ruoran Gabriela Jiang

Examiners:
Prof. Dr. Erika Ábrahám
Prof. Dr. Jürgen Giesl

Additional Advisor:
László Antal

Aachen, 29.03.2023

Abstract

In recent years, *neural networks* have been increasingly utilized in safety-critical and mission-critical systems such as autonomous cars and airborne collision avoidance systems. However, verifying the safety of these systems remains a significant challenge. In this work, we focus on *neural network control systems* (NNCS), where a neural network interacts with a continuous plant, modeled by a simplified hybrid automaton, and aim to make statements about the system's safety by examining the transient behavior of generated trajectories. To achieve this, we employ *reachability analysis*, a conservative approach that over-approximates the set of reachable states. Furthermore, we extend the reachability analysis definition of neural networks to incorporate the staircase function for classification and introduce an iterative approach to minimize over-approximation errors. We demonstrate the effectiveness of our approach on multiple benchmarks and various neural network controllers, reporting quantitative results from experiments. Overall, our work provides a comprehensive and computationally efficient algorithm that enables the reachability analysis of NNCS and facilitates the design of safe neural network controllers.

Keywords— Neural network control systems, safety verification, reachability analysis, flowpipe construction, over-approximative computation, exact computation

Erklärung

Hiermit versichere ich, dass ich die vorgelegte Arbeit selbstständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe. Alle benutzten Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate wurden als solche gekennzeichnet.

Ruoran Gabriela Jiang
Aachen, den 29. März 2023

Acknowledgements

I would like to express my deepest gratitude to my master thesis supervisor, László Antal, for his tireless commitment, guidance and unwavering support throughout the entire process of working on this thesis. His profound understanding of the subject and his willingness to share his knowledge through insightful discussions and explanations have been invaluable in helping me shape my ideas and arguments. I am also grateful for his constructive feedback that helped me to improve the quality of this thesis.

Furthermore, I would like to extend my appreciation to my professor, Professor Ábrahám, who provided me with helpful feedback that steered the thesis in the right direction when I was not sure where I was heading. Her expertise has been instrumental in shaping my approach and methodology.

Contents

Notations

Throughout this work, we will use the following standard notational conventions.

Notation (Set operators)**.** *For sets A,B we define their operators as follows*

- *Set union:* $A \cup B := \{x \mid x \in A \vee x \in B\}$
- *Set intersection:* $A \cap B := \{x \mid x \in A \wedge x \in B\}$
- *Set minus:* $A \setminus B := \{x \mid x \in A \wedge x \notin B\}$
- *Cross product:* $A \times B := \{(a,b) \mid a \in A \wedge b \in B\}$

Notation (Powerset)**.** *For a set A, we denote its powerset by* $2^A = \{B \mid B \subseteq A\}$.

Notation (Number sets)**.** *We denote by*

- $\mathbb{N}$ *the set of natural numbers including 0,*
- $\mathbb{N}_{>0} = \mathbb{N} \setminus \{0\}$ *the set of positive natural numbers,*
- $\mathbb{Z}$ *the set of integers,*
- $\mathbb{Q}$ *the set of rational numbers,*
- $\mathbb{R}$ *the set of real numbers.*

Notation (d-dimensional space of a set)**.** *Let A be a set. Then its d-dimensional space is denoted by* $A^d = \underbrace{A \times \cdots \times A}_{d \text{ times}}$. *If not stated otherwise, we assume* $d \in \mathbb{N}_{>0}$.

Notation (Intervals)**.** *For $S \in \{\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$, and a,$b \in S$, we follow the usual notation where*

- $[a,b] := \{s \in S \mid a \leq s \leq b\}$,
- $[a,b) := \{s \in S \mid a \leq s < b\}$,
- $(a,b] := \{s \in S \mid a < s \leq b\}$,
- $(a,b) := \{s \in S \mid a < s < b\}$

Notation (Set of intervals)**.** *We denote the set of all real-valued intervals by* $\mathbb{I}$.

Notation (Random sample)**.** *For a random variable X and a probability distribution f, we write $X \sim f$ to denote the random variable is being randomly sampled from f.*

Notation (Uniform distribution)**.** *By $\mathcal{U}(a,b)$ we denote the uniform distribution over* $[a,b] \subseteq \mathbb{R}$.

Notation (Tuples)**.** *Let $n \in \mathbb{N}_{>0}$ and let S_i be a set for all $i \in \{1, \ldots, n\}$. Then $s \in S := S_1 \times \cdots \times S_n$ is a tuple. We write $s = (s_1, \ldots, s_n)$.*

Basic Definitions

Before diving into the details of the thesis, it is necessary to establish a common understanding of key terms and concepts. In this chapter, we provide a set of general definitions that are used throughout this work.

Definition 0.0.1 (Vector)**.** *A d-dimensional vector $x \in \mathbb{R}^d$, $d \in \mathbb{N}_{>0}$ is an ordered sequence of d real values $x_1, \ldots, x_d \in \mathbb{R}$. A column vector is ordered vertically*

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$$

whereas a row vector is oriented horizontally and can be written as a tuple $(x_1, \ldots, x_d)$. Transposition $(x_1, \ldots, x_d)^T$ transforms a row vector into a column vector.

Definition 0.0.2 (Standard basis vector)**.** *Let d be the dimension of a real vector space, then we denote its i-th standard basis vector as $e_i \in \mathbb{R}^d$ with $e_i = (0, \ldots, 0, 1, 0, \ldots, 0)^T$.*

Definition 0.0.3 (Matrix)**.** *A (real-valued) matrix $\mathcal{A}$ of dimension $n \times m$ is a collection of $n \cdot m$ real numbers arranged in a rectangular array with n rows and m columns:*

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,m} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,m} \end{bmatrix}.$$

The set of all (real-valued) matrices of dimension $n \times m$ is denoted by $\mathbb{R}^{n \times m}$. The matrix entry at row i and column j in matrix A is referenced by $a_{i,j}$, while we use $a_{i,-}$ to reference the i-th row and consequently $a_{-,j}$ to refer to the j-th column of A.

Definition 0.0.4 (Convex sets)**.** *A set S is* convex *if the line segment between any two points in S lies in S, i.e., for all $s_1, s_2 \in S$, and for all $\alpha \in [0,1]$,*

$$\alpha s_1 + (1 - \alpha) s_2 \in S.$$

We extend this definition to an arbitrary number n of points such that a set S is convex iff any convex combination of points $\alpha_1 s_1 + \cdots + \alpha_n s_n$, for points $s_1, s_2, \ldots, s_n \in S$ and $\alpha_1, \ldots, \alpha_n \geq 0$ with $\sum_{i=1}^{n} \alpha_i = 1$, is also in S.

Definition 0.0.5 (Convex hull)**.** *The* convex hull *of a set S is the set of all convex combinations of points in S:*

$$conv(S) = \left\{ \alpha_1 s_1 + \cdots + \alpha_n s_n \mid s_i \in S, \alpha_i \geq 0, i = 1, \ldots, n, \sum_{i=1}^{n} \alpha_i = 1 \right\}.$$

Definition 0.0.6 (Minkowski sum)**.** *The* Minkowski sum $A \oplus B$ *of two sets* $A, B \subseteq \mathbb{R}^d$ *is defined as the set*

$$\{a + b \mid a \in A \wedge b \in B\}.$$

Definition 0.0.7 (Affine transformation)**.** *For a d-dimensional set* S*, its* affine transformation $A \cdot S + b$*,* $A \in \mathbb{R}^{d \times d}, b \in \mathbb{R}^d$ *is defined as the set*

$$\{A \cdot s + b \mid s \in S\}.$$

Definition 0.0.8 (Emptiness checking)**.** *A set* S *is empty if it holds no elements, that is*

$$\exists S. \forall y. (\neg (y \in S)).$$

Definition 0.0.9 (Halfspace)**.** *A d-dimensional* halfspace *is a convex set*

$$H = \{x \in \mathbb{R}^d \mid c^T \cdot x \leq z\}$$

for some $c \in \mathbb{R}^d$*, called the* normal *of the halfspace, and an offset* $z \in \mathbb{R}$*.*

Definition 0.0.10 (Halfspace intersection)**.** *Let* $H \subseteq \mathbb{R}^d$ *be a halfspace and* $A \subseteq \mathbb{R}^d$ *some set. Then the* halfspace intersection $\cdot \cap h$ *is defined as* $A \cap H$*.*

Definition 0.0.11 (Hausdorff Distance)**.** *The* Hausdorff distance *between two sets* $A, B \subseteq \mathbb{R}^d$ *is defined as*

$$d_{A,B} = \max \left\{ \sup_{a \in A} \inf_{b \in B} d(a,b), \sup_{b \in B} \inf_{a \in A} d(a,b) \right\}$$

for some distance metric $d(\cdot)$*.*

Definition 0.0.12 (Box)**.** *A set* $B \subseteq \mathbb{R}^d$ *is a* box *if there exist intervals* $I_1, \ldots, I_d \subseteq \mathbb{I}^d$ *such that*

$$B = I_1 \times \cdots \times I_d.$$

Definition 0.0.13 (Convex polyhedron)**.** *A set* $\mathcal{P} \subseteq \mathbb{R}^d$ *is a* convex polyhedron *if there are* $n \in \mathbb{N}$ *and* $c_i \in \mathbb{R}^d, s_i \in \mathbb{R}, i = 1, \ldots, n$ *such that*

$$\mathcal{P} = \bigcap_{i=1}^{n} h_i \text{ where } h_i = \left\{ x \in \mathbb{R}^d \mid c_i x \leq s_i \right\}.$$

A convex polyhedron can thus be expressed as an intersection with finitely many halfspaces.

Chapter 1

Introduction

In recent years, neural networks have been introduced to various new domains due to their promising potential, including safety-critical or mission-critical systems such as autonomous cars [WL19] or airborne collision avoidance systems [JLB+16]. As such, data-driven control systems, particularly neural-network-based controllers, have attracted a lot of attention. However, verifying the safety of these systems is a crucial yet challenging problem [KBD+17].

In this work, we consider controlled dynamical systems where the continuous plant model is given as a set of uniquely labeled linear ordinary differential equations (ODEs), and the controller is implemented by a neural network interacting with the plant periodically. Such a system is referred to as a *neural network control system* (NNCS). For this, we are interested in making statements about the safety of such a system by examining the transient behavior of all the generated trajectories. Verifying these questions for hybrid systems can be achieved using the method of *reachability analysis* that aims to determine the states a system can potentially reach. However, since the exact computation of the reachable states is generally infeasible [HKPV98], many approaches focus on a more conservative approach that over-approximates this reachable set, such as by flowpipe construction [FLGD+11]. The result of the analysis thus allows for the following assertion. If the over-approximation does not intersect the set of pre-defined unsafe states, then the actual reachable set also does not intersect them, and the system is thus considered safe. Note that this implication is solely valid in one direction, rendering the method sound but incomplete. On the other hand, the non-linearity of neural network controllers poses a significant challenge, and many approaches are restricted to specific types of neural networks. In this thesis, we apply the existing work by Tran et al. [Tra20] to compute the exact reachable states for the controller with ReLU activation.

The objective of this work is to develop an algorithm that facilitates the reachability analysis of neural network control systems by utilizing flowpipe construction for the plant and exact star-based reachability analysis for the controller. The challenge is to define the interaction between plant and controller and integrate both components into a comprehensive and computationally efficient analysis that allows for iterative over-approximation with a minimal error of the reachable states. To accomplish this task, we introduce the concept of an input star set for the system's state variables to be used in neural network reachability analysis and controller design. Moreover, we extend the reachability analysis definition of neural networks previously specified for

selected piece-wise linear activation functions, such as ReLU, leaky ReLU, and satlin, and also incorporate the staircase function for classification. Finally, to minimize over-approximation errors, we employ an iterative approach that transforms the original initial set rather than the entire flowpipe segment at a specific time point to avoid strong wrapping effects. We demonstrate the effectiveness of our approach on multiple benchmarks, showing its capability in handling dynamical systems with various neural-network controllers and reporting quantitative results from experiments.

1.1 Related Work

In [XTRJ18], the original polyhedron-based approach for ReLU feedforward neural networks was expanded to include the safety verification of neural network controlled systems. Specifically, this approach was used to verify the safety of a ReLU network controller that controls a discrete linear switching plant model. The method calculates the polyhedron-based reachable set of the neural network controller at every control step. It uses the convex hull of all polyhedra in the reachable set as the control input to the plant model. The plant's reachable set is then computed based on this control input. This approach works well for neural network controlled systems that have a small number of neurons and low-dimensional plant models. However, using the convex hull of all polyhedra in the reachable set as the control input to the plant model results in a conservative estimate of the system's safety due to over-approximation error. This over-approximation error can accumulate quickly over time when dealing with a large set of initial states.

More recently, a new simulation-guided approach for safety verification of a neural network controlled system was proposed in [XTYJ20]. The novel idea of this approach is the combination of interval arithmetic and simulation-guided input set partitioning to estimate the neural network's output ranges, which are then used as inputs for the plant model's reachability analysis. The experiments have shown that this approach can efficiently verify the safety of a neural network-based adaptive cruise control system. Moreover, this approach is much less computationally expensive than the maximum sensitivity approach presented in [XTJ18].

Verisig [IWA+19] introduces a novel method to convert a neural network controller with sigmoid activation function (or any other continuously differential activation function) into a nonlinear hybrid automaton by exploiting the fact that the sigmoid can be expressed as a solution to a quadratic differential equation. The resultant hybrid automaton comprises $L + 1$ modes and $2N$ states for a neural network with L layers and N neurons per layer. This hybrid system representation of the neural network controller is then composed in parallel with the plant model to create a single hybrid system representation of the neural network control system. Safety verification of the system properties is performed using hybrid systems verification tools such as Flow* [CÁS13]. The Verisig approach has been successfully applied in verifying the safety properties of the mountain car benchmark and DNN-based quadcopter application. To evaluate the scalability of their approach, the authors compared the reachable set computation times between the Verisig and Flow* approach and the prior mixed-integer linear programming (MILP) approach for DNNs of increasing sizes. The results showed that the network size increases linearly with the Verisig and Flow* reachability time. In contrast, the MILP approach showed an exponential increase in the reachability time, as reported in Figure 6 of the publication [IWA+19].

Huang et al. [HFL+19] propose a new approach called ReachNN for reachability analysis of NNCS with general neural-network controllers. The approach constructs a polynomial approximation based on Bernstein polynomials. It estimates the approximation error bound using two techniques, an a priori theoretical approach and an a posteriori approach based on adaptive sampling. The approach can approximate most neural networks to arbitrary precision and handle dynamic systems with various neural-network controllers, including heterogeneous networks with multiple types of activation functions. The approach achieves comparable or better approximation performance but with longer computation time than state-of-the-art approaches.

ReachNN* [FHC+20] extends the ReachNN tool and employs Bernstein polynomial approximation and knowledge distillation (KD) to retrain an NN controller with a smaller Lipschitz constant while preserving the original network's performance. The extended method accelerates the sampling-based error analysis in ReachNN using GPU-based parallel computing to uniformly sample the input space and evaluate the neural network controller and polynomial approximation. Experimental results demonstrate ReachNN*'s efficiency improvement over the previous prototype by 7x to 422x on a set of benchmarks.

1.2 Thesis Outline

In this thesis, we begin by presenting the fundamental concepts and definitions in Chapter 2. Specifically, in Section 2.1.1, we discuss the formalization of hybrid systems as hybrid automata, and in Section 2.1.2, we cover the established safety verification approach using flowpipe construction. We further introduce artificial neural networks in Section 2.2 and discuss a reachability analysis approach for feedforward neural networks in Section 2.2.1, which constitutes the core concept of neural network control system verification in Section 2.2.2.

Having established the foundations for our work, we proceed by presenting our algorithm in Chapter 3. To this end, we analyze the possible design choices for the final approach. This includes considering the invocation frequency of the neural network in Section 3.1.1, modeling the hybrid system in Section 3.1.2, modeling the controller in Section 3.1.3, and designing the neural network in Section 3.1.4. Finally, we present the complete analysis of our algorithm in Section 3.2.

To assess the algorithm's effectiveness, in Chapter 4, we evaluate the approach on two benchmarks from related literature in Section 4.1 and conduct experiments to quantify the run-time, accuracy, and complexity of the approach in Section 4.2.

Finally, we conclude this work in Chapter 5 by discussing the work's contributions and limitations and by suggesting future directions of work.

Chapter 2

Preliminaries

This chapter begins with a formal introduction of hybrid systems and their associated safety verification problem, and further covers the topic of neural networks and their safety analysis. From there, the preliminary work builds upon these concepts to provide a formal definition of neural network control systems and the methods for verifying their safety.

2.1 Hybrid Systems

Hybrid systems are real-time systems combining both *discrete* and *dynamic* components into a unified system exhibiting a mixed discrete-continuous behavior [Sch19]. Exemplary settings in which hybrid behavior arises are physical processes that are inherently hybrid by nature, such as the canonical example of a ball bouncing off the ground from an initial altitude [ACH+95], as well as digital controllers coupled to a continuous environment. While the controller's state is governed by the discrete switching of its configurations, it interacts with the environment's real-valued and continuously evolving variables. For instance, in aviation systems, the autopilot embodies the discrete controller and acts on quantities such as time, temperature, speed, etc., measured by means of sensors.

Example 2.1.1 (Thermostat)**.** *Consider a thermostat which keeps the temperature x in a room between $17°C$ and $23°C$ by turning the heater on and off according to the temperature it measures. When the heater is turned on (or off), the temperature in the room increases (or falls) according to the differential equation $\dot{x} = K(\rho - x)$ (or $\dot{x} = -Kx$ respectively), where t is the time, K a constant determined by the room and ρ a constant determined by the power of the heater.*

Due to hybrid systems often being embedded in the context of safety-critical applications, such as in autonomous driving or aviation systems, guaranteeing and validating that the hybrid behavior complies with the design specifications of the system is of utmost importance. To verify this safety property, researchers have been seeking to answer an equivalent question: whether a potentially unsafe state (or configuration) is reachable from a set of initial states. This problem is known as the reachability problem for hybrid systems [LTS99]. Solving the reachability problem by formally analyzing a system's behavior thus provides a formal guarantee of its safety.

2.1.1 Hybrid Automata

Hybrid automata [Hen00] formally model systems exhibiting both continuous and discrete dynamics and extend in some respect discrete transition systems. The following definitions are based on notations presented in [ACH+95, Á21, Sch19].

Definition 2.1.1 (Syntax of hybrid automata)**.** *A d-dimensional* hybrid automaton $\mathcal{H}$ *is described by a tuple* (*Loc*, *Var*, *Con*, *Lab*, *Edge*, *Act*, *Inv*, *Init*) *where*

- *Loc is a finite set of* locations *or* control modes*,*
- $Var = \{x_0, \ldots, x_{d-1}\}$ *is a finite ordered set of real-valued* variables*,*
- $Con : Loc \to 2^{Var}$ *assigns a set of* controlled variables *to each location,*
- *Lab is a finite set of* labels *including the* stutter label $\tau \in Lab$*,*
- $Edge \subseteq Loc \times Lab \times 2^{V^2} \times Loc$ *is a finite set of* edges *(or* transitions*) including a* τ*-transition* (ℓ, τ, Id, ℓ) *for each location* $\ell \in Loc$ *with* $Id = \{(\nu, \nu') \in V_{Var}^2 \mid \forall x \in Con(\ell).\nu'(x) = \nu(x)\}$*, and where all edges with label* τ *are* τ*-transitions,*
- *Act is a function assigning a set of* activities *or* flows $f : \mathbb{R}_{\geq 0} \to V$ *to each location which are time-invariant meaning that* $f \in Act(\ell)$ *implies* $(f + t) \in Act(\ell)$ *where* $(f + t)(t') = f(t + t')$ *for all* $t' \in \mathbb{R}_{\geq 0}$*,*
- *Inv is a function assigning an* invariant $Inv(\ell) \subseteq V$ *to each location* $\ell \in Loc$*, and*
- $Init \subseteq \Sigma$ *is a set of* initial states*.*

where V *denotes the set of all valuations* $\nu : Var \to \mathbb{R}$*, and* $\Sigma = Loc \times V$ *the state space of* $\mathcal{H}$*.*

Definition 2.1.2 (Operational semantics of hybrid automata)**.** *The operational semantics of a hybrid automaton* $\mathcal{H} = (Loc, Var, Con, Lab, Edge, Act, Inv, Init)$ *is given by two rules: one for discrete instantaneous steps and one for continuous time steps.*

1. *Discrete step semantics*

$$\frac{(\ell, a, (\nu, \nu'), \ell') \in Edge \quad \nu' \subseteq Inv(\ell')}{(\ell, \nu) \xrightarrow{a} (\ell', \nu')} Rule_{discrete}$$

2. *Time step semantics*

$$\frac{f \in Act(\ell) \quad f(0) = \nu \quad f(t) = \nu' \quad t \geq 0 \quad f([0, t]) \subseteq Inv(\ell)}{(\ell, \nu) \xrightarrow{t} (\ell, \nu')} Rule_{time}$$

An execution step $\to = \xrightarrow{a} \cup \xrightarrow{t}$ *of* $\mathcal{H}$ *is either a* discrete *or a* time step*, and we give its transitive closure by* $\to^*$*.*

For a transition $(\ell, a, \mu, \ell') \in \mathit{Edge}$, the transition relation $\mu \in V \times V$ thus specifies that a discrete step can be taken iff $(\nu, \nu') \in \mu$. The corresponding *guard* $\{\nu \in V \mid \exists \nu' \in V.\, (\nu, \nu') \in \mu\}$, often modeled by first-order logic formulae, *enables* a transition if it evaluates to true. Hence the operational semantics of hybrid automata induces a state transition system that can be visualized graphically. For better readability, the trivial τ-transitions can be omitted.

Example 2.1.2 (Thermostat)**.** *Assume the thermostat from Example 2.1.1. The corresponding hybrid automaton, omitting τ-transitions, is depicted in Figure 2.1.*

The system has two locations ℓ_{on} and ℓ_{off}: in location ℓ_{on}, the heater is turned on, whereas in location ℓ_{off}, the heater is off. The transition relations are specified by the guards; control thus may change locations from ℓ_{on} to ℓ_{off} or vice versa, whenever the temperature is above $22°C$ or below $18°C$, respectively. The flows are represented by the differential equations and the location invariants by logical formulae. Initially, the heater is on and the temperature is $20°C$. Thus formally, the automaton is defined as

- $\mathit{Loc} = \{\ell_{on}, \ell_{off}\}$,
- $\mathit{Var} = \{x\}$,
- $\mathit{Con}(\ell_{on}) = \mathit{Con}(\ell_{off}) = \{x\}$
- $\mathit{Lab} = \{\tau, a\}$
- $\mathit{Edge} =$

$$\begin{aligned}\{ &\left(\ell_{on}, a, \left\{(\nu, \nu') \in V^2 \mid \nu(x) \geq 22 \wedge \nu'(x) = \nu(x)\right\}, \ell_{off}\right), \\ &\left(\ell_{off}, a, \left\{(\nu, \nu') \in V^2 \mid \nu(x) \leq 18 \wedge \nu'(x) = \nu(x)\right\}, \ell_{on}\right), \\ &\left(\ell_{on}, \tau, \left\{(\nu, \nu') \in V^2 \mid \nu = \nu'\right\}, \ell_{on}\right), \\ &\left(\ell_{off}, \tau, \left\{(\nu, \nu') \in V^2 \mid \nu = \nu'\right\}, \ell_{off}\right)\}\end{aligned}$$

- $\mathit{Act}(\ell_{on}) = \left\{f : \mathbb{R}_{\geq 0} \to V \mid \forall t \in \mathbb{R}_{\geq 0}.\, f(t)(x) = 20e^{-Kt} + \rho\left(1 - e^{-Kt}\right)\right\}$,
  $\mathit{Act}(\ell_{off}) = \left\{f : \mathbb{R}_{\geq 0} \to V \mid \forall t \in \mathbb{R}_{\geq 0}.\, f(t)(x) = 20e^{-Kt}\right\}$,
- $\mathit{Inv}(\ell_{on}) = \{\nu \in V \mid \nu(x) \leq 23\}$,
  $\mathit{Inv}(\ell_{off}) = \{\nu \in V \mid \nu(x) \geq 17\}$,
- $\mathit{Init} = \{(\ell_{on}, \nu) \in \Sigma \mid \nu(x) = 20\}$.

We note that the synchronization label a would only play a role in parallel composition of this hybrid automaton with another one. Considered in isolation, the label a can be omitted for simplicity.

Definition 2.1.3 (Path)**.** *A* path *(or* run *or* execution*) π of a hybrid automaton $\mathcal{H}$ is a (possibly infinite) sequence*

$$\sigma_0 \to \sigma_1 \to \sigma_2 \ldots$$

of states $\sigma_i \in \Sigma$ for $i \geq 0$ starting in an initial state $\sigma_0 = (\ell_0, \nu_0)$ with $(\ell_0, \nu_0) \in \mathit{Init}$ and $\nu_0 \in \mathit{Inv}(\ell_0)$. A state σ of $\mathcal{H}$ is reachable *iff there is a run of $\mathcal{H}$ starting in an initial state of $\mathcal{H}$ and leading to σ.*

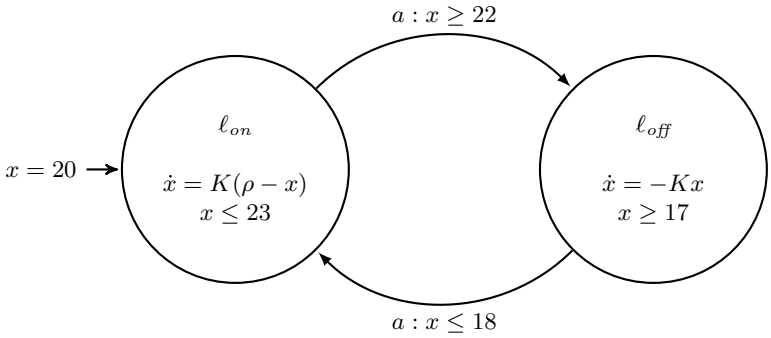


Figure 2.1: The hybrid automaton model of the thermostat.

Hybrid automata can be categorized into subclasses of varying degrees of expressivity. We provide a brief overview over the most popular classes by increasing expressivity.

- *Timed automata* introduce time as the only continuous component. The variables are clocks continuously evolving at a constant rate, i.e. the flow is restricted to the derivative $\dot{x} = 1$. Guards and invariants are conjunctions of constraints comparing variables to constants, edges can only reset variables to 0.
- *Rectangular automata* extend the flows of timed automata to rectangular sets such that $\dot{x} \in [a,b]$ and variables $x \in Var$ can be reset on jumps with $x \in [a,b]$ where $a,b \in \mathbb{N}$.
- *Linear hybrid automata I* (LHA I) further allow for linear expressions over the model variables in the invariants and guards, i.e. constraints are of the form $Ax \sim b$ with $\sim \in \{<, \leq, =, \geq, >\}$ and resets may use affine mappings $Ax + b$ where $A \in \mathbb{R}^{d \times d}, b \in \mathbb{R}^d$.
- *Linear hybrid automata II* (LHA II) additionally support linear expressions in the flows. Dynamics can now be modeled by systems of linear ODEs.

Moving forward, if not stated otherwise, we assume the model class of LHA II.

2.1.2 Verification of Hybrid Systems

As previously mentioned, formal safety verification of hybrid systems can be reduced to computing the set of reachable states from a set of initial states. In this respect, a system is considered safe if the intersection of a set $\bar{S}$ of pre-specified unsafe (or bad) states with the set $\mathcal{R}(I)$ of reachable states from an input set I is empty.

Definition 2.1.4 (Reachable Set)**.** *Let $I \subseteq \Sigma$ be a set of states, then the* reachable set $(I \mapsto^*) \subseteq \Sigma$ *of I is the set $\mathcal{R}$ of all states reachable from states in I, i.e.*

$$\mathcal{R}(I) = \{\sigma \in \Sigma \mid \exists \sigma' \in I . \sigma' \to^* \sigma\}$$

Definition 2.1.5 (Forward Reachability Analysis)**.** *Let $\mathcal{H}$ be a hybrid system, $I \subseteq \Sigma$ the initial state set, $\bar{S} \subseteq \Sigma$ the set of bad states. Then $\mathcal{H}$ is safe iff $\mathcal{R}(I) \cap \bar{S} = \emptyset$.*

Algorithm 1 General reachability algorithm for hybrid systems [Á21]

Input: Set of initial states $Init \subseteq \Sigma$ of a hybrid system model $\mathcal{H}$
Output: Set of reachable states $\mathcal{R}$

1: $\mathcal{R} \leftarrow Init_{\mathcal{H}}$;
2: $\mathcal{R}_{new} \leftarrow \mathcal{R}$;
3: **while** $\mathcal{R}_{\text{new}} \neq \emptyset \wedge \neg termination\_cond$ **do**
4: let $stateset \in \mathcal{R}_{new}$; ▷ Pick unprocessed state set
5: $\mathcal{R}_{new} \leftarrow \mathcal{R}_{new} \backslash \{stateset\}$;
6: $\mathcal{R}' \leftarrow$ computeFlowPipe($stateset$); ▷ Get set of state sets covering flowpipe
7: computeJumpSuccessors($\mathcal{R}'$); ▷ Add jump successors to $\mathcal{R}, \mathcal{R}_{new}$
8: **end while**
9: **return** $\mathcal{R}$

Since the reachability problem for general hybrid automata is undecidable [HKPV98], many approaches rely on conservative approximations, i.e. *over-approximating* the reachable state set with $\mathcal{R} \subseteq \mathcal{R}'$. However, results only provide conclusive answers in the case no bad states are reachable.

From among the various reachability analysis techniques, we focus on *flowpipe-construction-based* approaches that assume a time horizon within which the set of trajectories (*flowpipes*) of a given system is estimated. The general algorithm for flowpipe-based reachability analysis is presented in Algorithm 1 and is borrowed from [Á21, SÁMK17] where detailed explanations can be found. The rough idea is to expand the set of initial states by the successor iteratively states reachable within a single flow and a single jump (Line 6). The crucial step of the over-approximation lies in computing the flowpipes to obtain the time-successors for an initial set within a time horizon T by equally discretising T into δ-sized intervals $[i \cdot \delta, (i+1) \cdot \delta]$, $i = 0, \ldots, T-1$. Hence in the following, we introduce the construction of flowpipes for autonomous linear hybrid systems specified by linear hybrid automata.

Flowpipe Construction

Given an LHA II $\mathcal{H}$, the flow in each location is a system of linear differential equations

$$\dot{x}(t) = Ax(t) \tag{2.1}$$

over d-dimensional variables $x \in Var$ where $A \in \mathbb{R}^{d \times d}$ is the flow matrix at time point t. In a *non-autonomous system*, the dynamics are extended by a time-dependent function $u(t)$ representing external inputs influencing the system evolution, thus resulting in dynamics of the form

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{2.2}$$

with $B \in \mathbb{R}^{d \times d}$. Usually, it is assumed that $u(t) \in \mathbb{R}^{d \times 1}$ is from a bounded domain $\mathcal{U}$.

Solving Equation 2.1 with

$$x(t) = e^{tA}x(0) \tag{2.3}$$

leads to the state at time t that is reachable from the initial state $x(0)$ at time point $t = 0$ by following the flow specified by A. Observe how the factor e^{tA} depends on the current location ℓ for which time successors need to be computed and the time t. Equation 2.3 can directly be extended to the set of variable valuations N such that

$$N_t = e^{tA}N_0. \tag{2.4}$$

While Equation 2.4 allows computing the set of reachable states at specific time points t, it is not yet equipped to make statements about the set of reachable states over a time interval.

To overcome this issue, methods that over-approximate the error α between an approximation Ω of the set of reachable states for the time interval $[0, \delta]$ and the actual set of reachable states

$$\mathcal{R}_{[0,\delta]} = \left\{(\ell, \nu) \mid \nu = e^{tA}x_0, t \in [0, \delta], x_0 \in N_0\right\}$$

have been proposed.

Girard et al. [Gir05] propose to over-approximate the error α by approximating the Hausdorff distance (see Definition 0.0.11) between $\Omega' = Conv(N_0 \cap N_\delta)$ and the actual reachable set of states $\mathcal{R}_{[0,\delta]}$.

For an initial state $x \in N_0$, and the state $r = e^{\delta A}x$ that is reached from x at time point δ, we consider their connecting line segment

$$\left\{ s_x(t) = x + \frac{t}{\delta}(r - x) \mid t \in [0, \delta] \right\}.$$

The convex hull of N_0 and N_δ is then the union of all line segments $s_x(t)$ for all $x \in N_0$. From there, the error between a line segment and the actual trajectory $\zeta_x(t) = e^{tA}x$ is evaluated as

$$\|\zeta_x(t) - s_x(t)\| = \left\| e^{tA}x - x - \frac{t}{\delta}\left(e^{\delta A} - I\right)x \right\|.$$

Using Taylor's theorem of the second degree, this error is approximated such that

$$\left\| e^{tA}x - x - \frac{t}{\delta}\left(e^{\delta A} - I\right)x \right\| \leq \underbrace{\left(e^{\delta\|A\|} - 1 - \delta\|A\|\right)\|x\|}_{\alpha}.$$

This upper bound α on the error allows us to safely over-approximate the set of reachable states $R_{[0,\delta]}$ for the time interval [0,δ] by bloating the convex hull with a ball $\mathcal{B}_\alpha$ of radius α such that

$$\Omega_0 = conv(X_0 \cup e^{\delta A}X_0) \oplus \mathcal{B}_\alpha$$

where the operator $\oplus$ denotes the Minkwoski-sum (Definition 0.0.6). This approach results in a uniform bloating. In later works, non-uniform bloating [LG09] stating $\Omega_0 = conv(X_0 \cup (e^{\delta A}X_0 \oplus \mathcal{B}_{(\alpha')}))$ has been proposed that creates smoother flowpipes and smaller over-approximation than uniform bloating.

From the first flowpipe segment that over-approximates the trajectory in the time $[0, \delta]$ we can compute the further segments Ω_i using the same time step size δ and thereby equally discretizing the time horizon T. For an autonomous hybrid system, we obtain a sequence of segments Ω_i from the recurrence relation

$$\Omega_{i+1} = e^{\delta A}\Omega_i \tag{2.5}$$

for a location ℓ with flow matrix A. By construction, repeated application of $e^{\delta A}$ produces a sequence of segments that safely overapproximates the interval $[i\delta, (i+1)\delta]$. For non-autonomous hybrid systems, an additional bloating step is needed in each iteration. We refer to [LG09] for a detailed description.

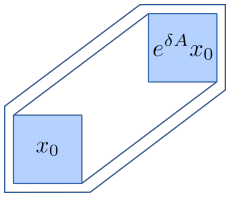


Figure 2.2: Uniform bloating

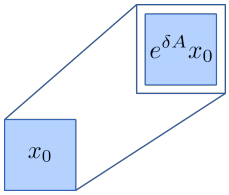


Figure 2.3: Improved bloating

Figure 2.4: Construction of the first segment of a flowpipe, reproduced from [Á21].

State Set Representations

Flowpipe-construction-based reachability analysis as presented in Algorithm 1 calls for an efficient state set representation (datatype) during computation for performing set operations (union, intersection, linear transformation, Minkowski sum, etc.), and touches on the general problem of trading off between computational complexity and accuracy. On that account, a plethora of representations has been introduced, mainly divided into *geometric* representations (e.g., boxes, oriented rectangular hulls, convex polyhedra, template polyhedra, orthogonal polyhedra, zonotopes, ellipsoids) or *symbolic* representations (e.g., support functions or Taylor models). In particular, as this thesis works extensively with convex polytopes, we refer to Definition 0.0.13.

Further assuming closedness leads us to *convex polytopes* that have two widely used representations (see Figure 2.5). An $\mathcal{H}$*-representation* (halfspace representation) defines a polytope by its facets and is given by a pair (C,s) with a matrix $C \in \mathbb{R}^{n\times d}$, a vector $s \in \mathbb{R}^n$ such that $\mathcal{P} = \bigcap_{i=1}^{n}\{x \mid C_{i,-}x \leq s_i\}$, i.e. each row $C_{i,-}$ of C is the normal vector and s_i the offset to the i-th facet of the polytope. A $\mathcal{V}$*-representation* (vertex representation) stores a polytope $\mathcal{P}$ by its vertices and is given by a set V of d-dimensional points such that its convex hull (Definition 0.0.5) specifies the polytope, i.e. $\mathcal{P} = conv(V)$. The underlying polytope representation determines the computational effort for different set operations.

Example 2.1.3 ($\mathcal{H}$-polytope representation)**.** *Assume a 2-dimensional convex polytope $\mathcal{P}$, i.e. a polygon, as depicted in Figure 2.5. Then its $\mathcal{H}$-representation (Figure 2.5a) is given by (C, s) where*

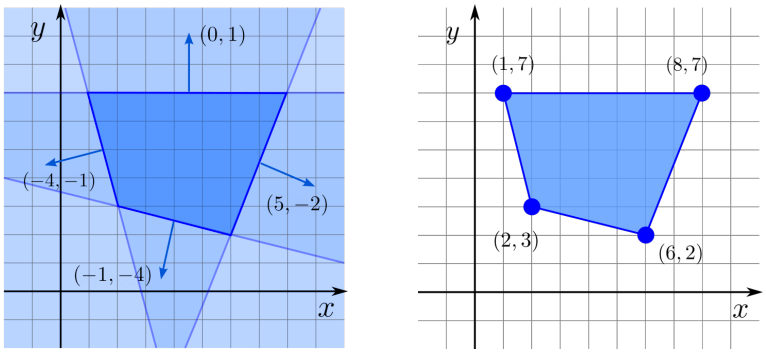
$$C = \begin{pmatrix} 0 & 1 \\ 5 & -2 \\ -1 & -4 \\ -4 & -1 \end{pmatrix}, \quad s = \begin{pmatrix} 7 \\ 26 \\ -14 \\ -11 \end{pmatrix}.$$

We can equivalently use the $\mathcal{V}$-polytope representation to denote the exact same set.

Example 2.1.4 ($\mathcal{V}$-polytope representation)**.** *Suppose the same convex polytope as in Example 2.1.3. Then for the $\mathcal{V}$-representation (Figure 2.5b), we obtain $\mathcal{P} = conv(V)$ where*

$$V = \{(1,7), (8,7), (2,3), (6,2)\}.$$

Given that our work heavily uses the state representation of boxes, it is necessary to separately consider this particular class of convex polytopes. Boxes, defined as a sequence $(I_1, \ldots, I_d)$ of intervals, are considered one of the simplest and most efficient



(a) $\mathcal{H}$-representation (b) $\mathcal{V}$-representation

Figure 2.5: Example of convex polytope representations for $d = 2$.

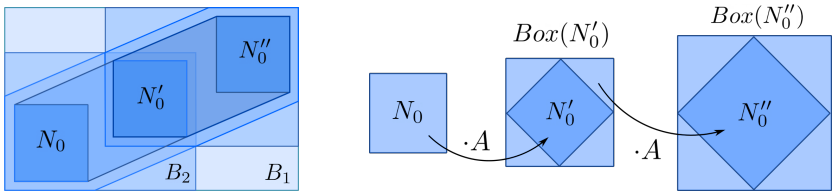
state set representations for reachability analysis of hybrid systems [Sch19] (refer to Definition 0.0.12). Nonetheless, it is important to acknowledge that their use can often result in considerable over-approximations, particularly if the closure of state set representations for all set operations is required, as is the case in HyPro. When boxes are used to compute set operations, the result of each operation is approximated by its bounding box, which can introduce additional over-approximation. These cumulative errors are commonly referred to as the *wrapping effect* [LG09]. Although some of these effects can be alleviated by using a smaller step size δ, others caused by specific linear affine transformations cannot be mitigated. From Figure 2.6a, we see that $B_1 = Box(conv(N_0 \cup N_0' \cup N_0'') \oplus \mathcal{B}_\alpha)$ has a significantly larger error than $B_2 = Box(conv(N_0 \cup N_0') \oplus \mathcal{B}_{\alpha_1}) \cup Box(conv(N_0' \cup N_0'') \oplus \mathcal{B}_{\alpha_2})$ with the intermediate step. However, some transformations such as shearing or rotations inherently introduce large errors over the entire time horizon independent of the step size. We visualize the latter using the transformation by the matrix A in Figure 2.6b.

In approximation problems, constructing flowpipes involves a balance between accuracy and time complexity. The number of flowpipes constructed grows exponentially with each jump, which presents a significant computational burden. To reduce this cost, one can over-approximate several sets with a single set, although this comes at the cost of precision.

Clustering is one method used to achieve such over-approximation. In this method, sets are grouped into clusters and each group is over-approximated by a single set. When all considered sets are aggregated into a single group and thus over-approximated by a single set, it is referred to as *aggregation*. Figure 2.7 provides a visual representation of this process.

2.2 Neural Networks

Alongside the increasingly widespread implementations of neural networks in safety-critical applications, for instance in autonomous cyber-physical systems (CPS) such as self-driving cars, emerges a need for methods proving the safety of AI systems. However, due to the complex and intransparent prediction processes of neural networks, verifying their safety requires dedicated formal techniques different from those for ordinary hybrid systems. The difficulty in proving properties about neural networks is



(a) Step-size dependent (b) Inherent over-approximation errors

Figure 2.6: Wrapping effects in the box representation.

caused by the presence of non-linear characteristics, rendering the problem non-convex such that verifying simple properties is already NP-complete [KBD+17]. Slight perturbances in the input can already make their behavior unpredictable [SZS+14].

The objective of this section is to present a safety verification algorithm for neural network control systems (NNCS) which in the general sense is a plant being regulated by a neural network controller where both components interact in a cyclic manner. We assume the controller to be a fully-connected *feedforward neural network* (FNN), the simplest class of neural networks, restricted to *Rectified Linear Unit* (ReLU) activation layers, and the plant dynamics to be modeled by a hybrid automaton. An illustration of a general NNCS is depicted in Figure 2.9 which we elaborate on in Section 2.2.2. This section is guided by [Tra20].

Taking a step back, we need to first analyze reachability in neural networks before we can establish a method for computing reachable states in NNCS for safety verification. Thus foremost, we briefly introduce FNNs which are composed of a number of interconnected neurons divided into layers. Each neuron responds to the weighted inputs it receives from the neurons of the previous layer, processes them and passes a computed output to the next layer. The action of a neuron is specified by its activation function, such that the output of the i-th neuron is given as

$$y_i = \phi\left(\sum_{j=1}^{n} w_{ij}x_j + b_i\right) \tag{2.6}$$

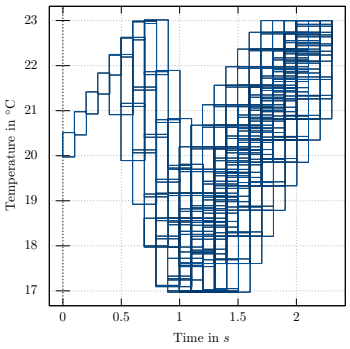
where x_j is the j-th input neuron, w_{ij} the weight from the j-th output, b_i is the bias, and $\phi(\cdot)$ is the (non-linear) activation function. In this thesis, $\phi(x) = ReLU(x) = \max(x,0)$. It is worth noting that the methods introduced in the following can be extended to support any other kind of piece-wise linear activation function.

2.2.1 Reachability Analysis of Neural Networks

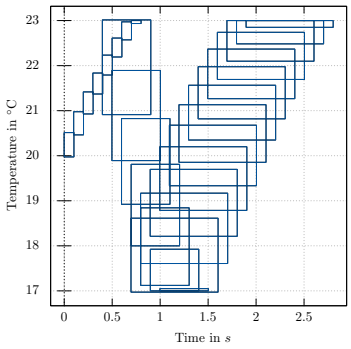
Following the same rationale as for general hybrid systems (refer Definition 2.1.5), safety verification of NNCS can be computed by validating that no bad states can be reached from an initial input set. For this, a notion of reachability in neural networks is required.

Definition 2.2.1 (Reachable set of an FNN)**.** *Let $I = \{x \in \mathbb{R}^d \mid Ax \leq b\}$ with $A \in \mathbb{R}^{n\times d}, b \in \mathbb{R}^n$ be a bounded convex polyhedron input set and $\mathcal{N} = \{L_1, \cdots, L_k\}$ a k-layer FNN. Then the* reachable set $\mathcal{N}(I) = \mathcal{R}_{L_k}$ *of the neural network $\mathcal{N}$ given the input set I is defined inductively by*

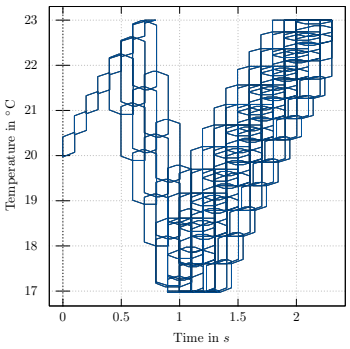
$$\mathcal{R}_{L_0} := I$$
$$\mathcal{R}_{L_i} := \left\{y_i \mid y_i = \phi_i\left(W_i y_{i-1} + b_i\right), y_{i-1} \in \mathcal{R}_{L_{i-1}}\right\}, \quad i = 1, \ldots, k$$



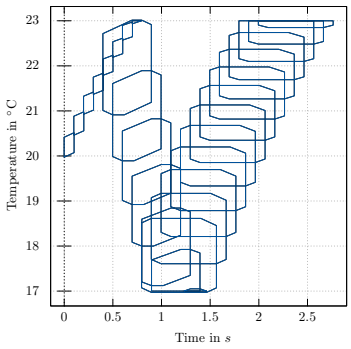
(a) Box, no aggregation



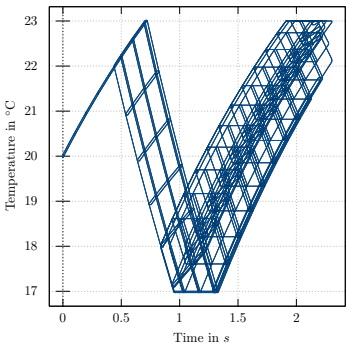
(b) Box, with aggregation



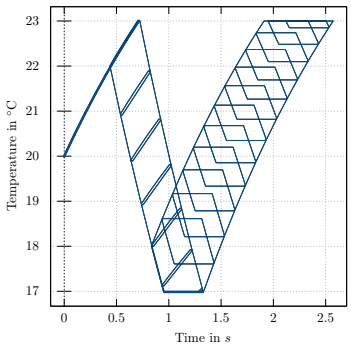
(c) $\mathcal{H}$-polytope, no aggregation



(d) $\mathcal{H}$-polytope, with aggregation



(e) $\mathcal{V}$-polytope, no aggregation



(f) $\mathcal{V}$-polytope, with aggregation

Figure 2.7: Flowpipe construction using different state set representations with and without aggregation for the thermostat example with a global time horizon $T = 2.1$ and time step $\delta = 0.1$.

where W_i, b_i and ϕ_i are the weight matrix, bias vector and activation function of the i-th layer L_i, respectively.

Hence a k-layer FNN $\mathcal{N}$ is considered safe, i.e. $\mathcal{N}(I) \models S$ iff $\mathcal{R}_{L_k} \cap \neg S = \emptyset$ for a safety specification S and an input set I. For the reachability analysis of FNNs with ReLU activation functions, we distinguish between two approaches: an exact and complete analysis (Section 2.2.1), and an over-approximate analysis (Section 2.2.1), that both require an adequate and efficient datatype for computation. This work presents a geometric approach using *star set* representations which provide desirable properties for performing set operations as star sets are scalable and fast, highly accurate and allow for the generation of counterexamples [BD17].

Star Set Representations

Definition 2.2.2 (Generalized star set)**.** *A* generalized star set *is a tuple $\Theta = \langle c, V, P\rangle$ where $c \in \mathbb{R}^d$ is the center, $V = \{v_1, \ldots, v_m\} \subseteq \mathbb{R}^d$ a set of basis vectors, and $P : \mathbb{R}^m \to \{\top, \bot\}$ a predicate. The set of states represented by star sets is given by*

$$[\![\Theta]\!] = \left\{ x \in \mathbb{R}^d \mid x = c + \sum_{i=1}^{m} (\alpha_i v_i) \;\; \textit{such that } P(\alpha_1, \cdots, \alpha_m) = \top \right\}. \tag{2.7}$$

Sometimes we will refer to both the tuple Θ and the set of states $[\![\Theta]\!]$ as Θ. In this work, we restrict the predicates to be a conjunction of linear constraints, $P(\alpha) \triangleq C\alpha \leq s$ where, for p linear constraints, $C \in \mathbb{R}^{p\times m}, \alpha$ is the vector of m-variables i.e., $\alpha = [\alpha_1, \ldots, \alpha_m]^T$, and $s \in \mathbb{R}^{p\times 1}$.

Next, we show the universal representation power of stars sets. The proofs of the following findings are collectively appended in Chapter A.

Proposition 2.2.1. *Any bounded convex polyhedron $\mathcal{P} = \left\{x \mid Cx \leq s, x \in \mathbb{R}^d\right\}$ can be represented as a star.*

Star sets are especially efficient with respect to computing the usually expensive operations of affine mappings and halfspace intersections for which they outperform polytope-based representations [TML+19]. The proofs are included in the supplementary materials, and we refer interested readers to the work by Tran et al. [Tra20] for a comprehensive explanation of the proofs.

Proposition 2.2.2 (Affine mapping of a star)**.** *Given a star set $\Theta = \langle c, V, P\rangle$, an affine mapping of the star Θ with the affine mapping matrix W and offset vector b defined by $\bar{\Theta} = \{y \mid y = Wx + b, x \in \Theta\}$ is another star such that*

$$\bar{\Theta} = \langle \bar{c}, \bar{V}, \bar{P}\rangle, \quad \bar{c} = Wc + b, \quad \bar{V} = \{Wv_1, Wv_2, \ldots, Wv_m\}, \quad \bar{P} \equiv P.$$

Proposition 2.2.3 (Star and halfspace intersection)**.** *The intersection of a star $\Theta = \langle c, V, P\rangle$ and a halfspace $H = \{x \mid Hx \leq g\}$ is another star such that*

$$\bar{\Theta} = \Theta \cap H = \langle c, V, \bar{P}\rangle$$

where $\bar{P} = P \wedge P'$ with

$$P'(\alpha) = (H \times V)\,\alpha \leq g - H \times c, \quad V = \begin{pmatrix} v_1 & v_2 & \cdots & v_m \end{pmatrix}.$$

Exact and Complete Analysis

By Proposition 2.2.1, we can assume the input set I for a k-layer FNN $\mathcal{N}$ is represented by a star set such that $I = \langle c, V, P\rangle$. Since star sets are closed under affine transformations by Proposition 2.2.2, computing the reachable set by Definition 2.2.1 for layer l with n neurons, an input star set Θ, yields a sequence of n stepReLU operations $R_l = ReLU_n\left(ReLU_{n-1}\left(\cdots ReLU_1\left(\Theta\right)\cdots\right)\right)$, potentially multiple star sets.

Deconstructing the stepReLU operation of the i-th neuron in the l-th layer, that is $ReLU_i(\cdot)$, results in the following procedure for an FNN as presented in Algorithm 2. Given the output star set $\Theta = \langle c, V, P\rangle$ of the preceding layer as input, it is partitioned into subsets $\Theta_1 = \Theta \cap x_i \geq 0, \quad \Theta_2 = \Theta \wedge x_i < 0$. Let $\Theta_1 = \langle c, V, P_1\rangle$ and $\Theta_2 = \langle c, V, P_2\rangle$. By definition, for $x = [x_1, \ldots, x_n]^T \in \Theta_1$, it holds that $ReLU(x_i) = x_i$, hence there is no change in x after applying the activation function. In contrast, for $x' = [x'_1, \ldots, x'_n]^T \in \Theta_2$, we have $ReLU(x'_i) = 0$, thus yielding a new vector $x'' = [x'_1, \ldots, x'_{i-1}, 0, x'_{i+1}, \ldots, x'_n]$ which is equivalent to mapping Θ_2 by the identity matrix with the i-th entry set to zero. Finally, the stepReLU operation of the i-th neuron for an input star set Θ results in $ReLU_i(\Theta) = \langle c, V, P_1\rangle \cup \langle Mc, MV, P_2\rangle$.

To reduce the number of computations, it is helpful to determine the ranges of all states in the input set Θ at the i-th neuron beforehand. If $x_i \geq 0$ (or $x_i \leq 0$) for all states $x \in \Theta$, then $ReLU_i(\Theta) = \Theta$ (or $ReLU_i(\Theta) = M\Theta$, respectively) as presented in Line 18 and Line 20 in Algorithm 2.

Lemma 2.2.4 (Worst-case complexity of number of stars)**.** *The worst-case complexity of the number of stars in the reachable set of an FNN with N neurons is $\mathcal{O}(2^N)$.*

Lemma 2.2.5. *The worst-case complexity of the number of constraints of a star in the reachable set of an FNN with N neurons is $\mathcal{O}(N)$.*

Theorem 2.2.6 (Verification complexity)**.** *Let $\mathcal{N}$ be an FNN with N neurons, Θ a star set in the input star set with p linear constraints and m variables in the predicate, S a safety specification with s linear constraints. In the worst case, verifying the safety of the neural network, i.e., checking $\mathcal{N}(\Theta) \stackrel{?}{\models} S$ is equivalent to solving up to 2^N feasibility problems with at most $N+p+s$ linear constraints and m variables each.*

Theorem 2.2.7 (Safety)**.** *Let $\mathcal{N}$ be an FNN, $\Theta = \langle c, V, P\rangle$ be a star input set, $\mathcal{N}(\Theta) = \bigcup_{i=1}^{k} \Theta_i$, $\Theta_i = \langle c_i, V_i, P_i\rangle$ be the reachable set of the neural network, and S be safety specification. Denote $\bar{\Theta}_i = \Theta_i \cap \neg S = \langle c_i, V_i, \bar{P}_i\rangle$, $i = 1, \ldots, k$. The neural network is safe iff $\bar{P} = \emptyset$ for all i.*

Over-approximate Analysis

Due to the exponentially growing number of stars over the number of layers by Lemma 2.2.4, the increase in computation costs limits the scalability of the exact analysis. Instead of decomposing the input star set for each neuron, the following approximation rule constructs only a single star each layer. For an output $y_i = ReLU(x_i)$, let

$$\begin{cases} y_i = x_i & \text{if } l_i \geq 0, \\ y_i = 0 & \text{if } u_i \leq 0, \\ y_i \geq 0,\ y_i \leq \frac{u_i(x_i - l_i)}{u_i - l_i},\ y_i \geq x_i & \text{if } l_i < 0 \text{ and } u_i > 0 \end{cases} \tag{2.8}$$

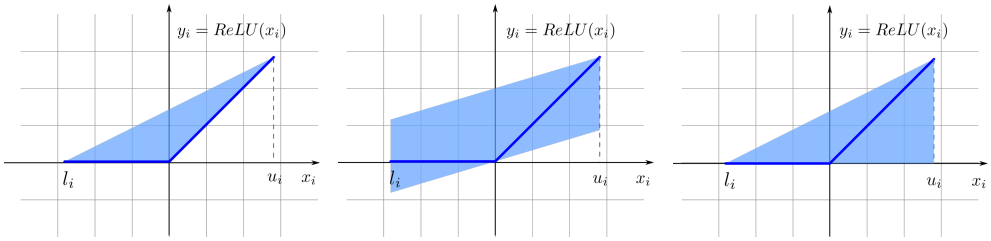
Algorithm 2 Star-based exact reachability analysis for one layer [Tra20].

Input Input star set $I = [\Theta_1 \ \cdots \ \Theta_N]$
Output Exact reachable set $\mathcal{R}$

1: **procedure** LAYERREACH(I, W, b)
2: $\mathcal{R} \leftarrow \emptyset$;
3: **for** $j = 1 : N$ **do**
4: $I_1 \leftarrow W * \Theta_j + b = \langle Wc_j + b, WV_j, P_j \rangle$;
5: $\mathcal{R}_1 \leftarrow I_1$;
6: **for** $i = 1 : n$ **do**
7: $[l_i, u_i] \leftarrow I_1.range(i)$; ▷ $l_i \leq x_i \leq u_i, x_i \in I_1[i]$
8: $\mathcal{R}_1 \leftarrow$ STEPRELU($\mathcal{R}_1, i, l_i, u_i$);
9: **end for**
10: $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_1$;
11: **end for**
12: **return** $\mathcal{R}$;
13: **end procedure**
14: **procedure** STEPRELU($\tilde{I}, i, l_i, u_i$)
15: $\tilde{\mathcal{R}} \leftarrow \emptyset$, $\tilde{I} = [\tilde{\Theta}_1 \ \cdots \ \tilde{\Theta}_k]$; ▷ Intermediate representations
16: **for** $j = 1 : k$ **do**
17: $\mathcal{R}_1 \leftarrow \emptyset$, $M \leftarrow [e_1 \ e_2 \ \cdots e_{i-1} \ 0 \ e_{i+1} \cdots e_n]$;
18: **if** $l_i \geq 0$ **then**
19: $\mathcal{R}_1 \leftarrow \tilde{\Theta}_j = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j \rangle$; ▷ No changes
20: **else if** $u_i \leq 0$ **then**
21: $\mathcal{R}_1 \leftarrow M * \tilde{\Theta}_j = \left\langle M\tilde{c}_j, M\tilde{V}_j, \tilde{P}_j \right\rangle$; ▷ Zero out i-th entry
22: **else** ▷ $l_i < 0 \wedge u_i > 0$
23: $\tilde{\Theta}'_j \leftarrow \tilde{\Theta}_j \wedge x[i] \geq 0 = \left\langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j \right\rangle$;
24: $\tilde{\Theta}''_j \leftarrow \tilde{\Theta}_j \wedge x[i] < 0 = \left\langle \tilde{c}_j, \tilde{V}_j, \tilde{P}''_j \right\rangle$;
25: $\mathcal{R}_1 \leftarrow \tilde{\Theta}'_j \cup M * \tilde{\Theta}''_j$; ▷ Decompose Θ_j
26: **end if**
27: $\tilde{\mathcal{R}} \leftarrow \tilde{\mathcal{R}} \cup \mathcal{R}_1$;
28: **end for**
29: **return** $\tilde{\mathcal{R}}$;
30: **end procedure**

where l_i and u_i are the lower and upper bounds of x_i.

Similar to the exact approach, the over-approximate reachable set of a layer with n neurons can be computed by executing a sequence of n *approximate-stepReLU* operations as outlined in Algorithm 3. Given an input star set $I = \langle c, V, P \rangle$, the algorithm maps the input set according to the layer's weight and bias and carries out the *approximate-stepReLU* operations on the resulting affine mapping. After first determining the lower and upper bound l_i, u_i of the state variable $x[i]$ at the i-th neuron, the operation distinguishes three cases. If the lower bound is not negative (Line 13), no changes are applied to the input. If the the upper bound is not positive (Line 15), the input set with the i-th state variable set to zero is returned as the intermediate reachable set. Notice that these two cases are analogous to the procedure as in the exact approach. Otherwise, in case the lower bound is negative and the



(a) Star (b) Zonotope (c) Abstract domain

Figure 2.8: Over-approximation of the ReLU function by different state set representations [Tra20]. The dark blue line denotes the exact set and the light blue area the approximate set.

upper bound positive (Line 17), for an input set's predicate P with $P(\alpha) \triangleq C\alpha \leq d$, $\alpha = [\alpha_1, \ldots, \alpha_m]^T$, we introduce a new variable α_{m+1} that encodes the over-approximation of the activation function at the i-th neuron according to Equation 2.8. To capture the predicates of the over-approximate output set by $P(\alpha') \triangleq C'\alpha' \leq d'$ where $\alpha' = [\alpha_1, \ldots, \alpha_m, \alpha_{m+1}]^T$, we thus define

$$C' = \begin{pmatrix} C & 0 \\ 0_{1\times m} & -1 \\ V_{i,-} & -1 \\ \frac{-u_i}{u_i - l_i} V_{i,-} & 1 \end{pmatrix}, \quad d' = \begin{pmatrix} d \\ 0 \\ -c_i \\ \frac{u_i}{u_i - l_i}(c_i - l_i) \end{pmatrix}$$

such that $C' \in \mathbb{R}^{(p+3)\times(m+1)}$, $d' \in \mathbb{R}^{p+3}$. Here, $V_{i,-}$ denotes the i-th row of $V = [v_1, \ldots, v_m]$ and $0_{1\times m} \in \mathbb{R}^{1\times m}$ a row vector of zeroes. Indeed, α_{m+1} captures the predicates correctly by only introducing one more variable and three more linear constraints in the predicate for the intermediate reachable set. This observation leads to the following lemma.

Lemma 2.2.8. *The worst-case complexity of the number of variables and constraints in the predicate in the over-approximate reachable set of an FNN with N neurons is $N + m_0$ and $3N + n_0$ respectively for an input set with m_0 variables and n_0 constraints in the predicate.*

To put the over-approximation rule from Equation 2.8 into perspective, over-approximation of the ReLU function by star sets is less conservative than by zonotope-based [SGM+18] or abstract domain based [SGPV19] representations as illustrated in Figure 2.8.

2.2.2 Neural Network Control System Verification

Definition 2.2.3 (Neural network control system [HFC+21])**.** *A* neural network control system *(NNCS) is denoted by a tuple $(X, U, F, \mathcal{N}, \delta, \mathcal{X}_0)$ where*

- *$X = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}$ is the set of finitely many real-valued state variables,*
- *$U = \{u_1, \ldots, u_m\} \subseteq \mathbb{R}$ the set of finitely many real-valued control variables (or inputs),*
- *$F : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ the ODE defining the continuous dynamics of the plant,*

Algorithm 3 Star-based over-approximate reachability analysis for one layer [Tra20].

Input Input star set $I = [\Theta]$
Output Over-approximate reachable set $\mathcal{R}$

procedure APPROXLAYERREACH(I, W, b)
  $I_1 \leftarrow W * I_0 + b = \langle Wc + b, WV, P \rangle$;
  $I' \leftarrow I_1$;
  for i = 1:n **do**
    $I' \leftarrow$ APPROXSTEPRELU(I', i);
  end for
  $\mathcal{R}_1 \leftarrow I'$;
end procedure
procedure APPROXSTEPRELU($\tilde{I}, i$)
  $\tilde{I} \leftarrow \tilde{\Theta} = \langle \tilde{c}, \tilde{V}, \tilde{P} \rangle$;
  $[l_i, u_i] \leftarrow \tilde{\Theta}.range(i)$; ▷ $l_i \leq x[i] \leq u_i$
  $M \leftarrow [e_1\ e_2\ \cdots e_{i-1}\ 0\ e_{i+1} \cdots e_n]$;
  if $l_i \geq 0$ **then**
    $\tilde{\mathcal{R}} \leftarrow \tilde{\Theta} = \langle c, V, P \rangle$;
  else if $u_i \leq 0$ **then**
    $\tilde{\mathcal{R}} \leftarrow M * \tilde{\Theta} = \langle Mc, MV, P \rangle$;
  else if $l_i < 0 \wedge u_i > 0$ **then**
    $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d}$, $\alpha = [\alpha_1, \ldots, \alpha_m]^T$;
    $\alpha' \leftarrow [\alpha_1, \ldots, \alpha_m, \alpha_{m+1}]^T$; ▷ New variable α_{m+1}
    $C_1 \leftarrow [0\ \cdots\ 0\ -1]$, $d_1 \leftarrow 0$; ▷ $\alpha_{m+1} \geq 0 \Leftrightarrow C_1\alpha' \leq d_1$
    $C_2 \leftarrow [\tilde{V}_{i,-}\ -1]$, $d_2 \leftarrow -\tilde{c}[i]$; ▷ $\alpha_{m+1} \geq x[i] \Leftrightarrow C_2\alpha' \leq d_2$
    $C_3 \leftarrow [\frac{-u_i}{u_i - l_i}\tilde{V}_{i,-}\ 1]$, $d_3 \leftarrow \frac{u_i}{u_i - l_i}(\tilde{c}[i] - l_i)$ ▷ $\alpha_{m+1} \leq \frac{u_i(x[i]-l_i)}{u_i-l_i} \Leftrightarrow C_3\alpha' \leq d_3$;
    $C_0 \leftarrow [\tilde{C}\ 0_{m\times 1}], d_0 \leftarrow \tilde{d}$;
    $C' \leftarrow [C_0^T\ C_1^T\ C_2^T\ C_3^T]^T, d' \leftarrow [d_0^T\ d_1\ d_2\ d_3]^T$;
    $P'(\alpha') \triangleq C'\alpha' \leq d'$;
    $c' \leftarrow M\tilde{c}$, $V' \leftarrow M\tilde{V}$, $V' \leftarrow [V'\ e_i]$; ▷ $y[i] = ReLU(x[i]) = \alpha_{m+1}$
    $\tilde{\mathcal{R}} \leftarrow \langle c', V', P' \rangle$;
  end if
end procedure

- $\mathcal{N} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ *denotes the input-output mapping defined by the neural network controller with input dimension* n_x *and output dimension* n_y*,*
- $\delta \in \mathbb{R}_{>0}$ *a positive number denoting the control step size, and*
- $\mathcal{X}_0 \subset \mathbb{R}^n$ *is the set of the initial values of the state variables.*

Given an NNCS and an initial state $x(0) \in \mathcal{X}_0$, the controller senses the system state at the beginning of every control step $t_k = k\delta$ for $k = 0, 1, \ldots$, and updates the control inputs to $u_k = \mathcal{N}(x(t_k))$. The system's dynamics in that control step is governed by the ODE $\dot{x}(t_k) = f(x_k, u_k) = Ax_k + Bu_k$.

Definition 2.2.4 (Execution of an NNCS)**.** *An* execution *(or* trajectory*) of an NNCS is given by a flowmap function* $g : X \times \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ *such that the system state at any time* $t \geq 0$ *from any intial state* $x(0) \in \mathcal{X}_0$ *is the function output* $g(x(0), t)$*.*

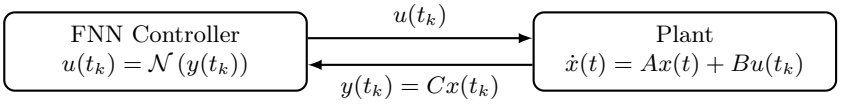


Figure 2.9: Architecture of a closed-loop NNCS at discrete time steps $t_k = k\delta$ for a control step size δ and $k = 0, 1, \ldots$. The plant is modeled by a (simplified) hybrid automaton and the controller by an FNN.

Algorithm 4 Reachability analysis for an NNCS.

Input: NNCS $(X, U, F, \mathcal{N}, \delta, \mathcal{X}_0)$, number of time steps K
Output: Set of reachable states $\mathcal{R}$ over time interval $[0, K\delta]$

1: $\mathcal{R} \leftarrow \emptyset$;
2: $\mathcal{X}_i \leftarrow \mathcal{X}_0$;
3: **for** $i = 1$ to K **do**
4: $\quad \mathcal{U}_i \leftarrow$ computeExactReachableSet$(\mathcal{N}(\mathcal{X}_i))$;
5: $\quad$ let $x(0) \in \mathcal{X}_i, u(0) \in \mathcal{U}_i$; ▷ Pick unprocessed state set
6: $\quad \mathcal{R}_i \leftarrow$ computeFlowPipe$(x(0), u(0))$;
7: $\quad \mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_i$;
8: $\quad \mathcal{X}_i \leftarrow$ getRelevantSets$(\mathcal{R}_i)$; ▷ Select last flowpipe in $\mathcal{R}_i$
9: **end for**
10: **return** $\mathcal{R}$

Definition 2.2.5 (Reachability of an NNCS)**.** *A state* $x' \in \mathbb{R}^n$ *is* reachable *if there exists* $x(0) \in \mathcal{X}_0$ *and* $t \geq 0$ *such that* $x' = g(x(0), t)$.

In the following, assume a general NNCS, i.e., a continuous plant governed by a neural network controller as depicted in Figure 2.9. For an NNCS with an FNN controller $\mathcal{N}$, and a linear plant P with the initial states $x(t_0) \in \mathcal{X}_0$, the task is to verify whether or not the state of the plant satisfies a safety property in a bounded number of time steps K, i.e. whether

$$\forall x(t_0) \in \mathcal{X}_0 \rightarrow g(x(t_0), t_k) \models S(g(x(t_0), t_k)), \quad \forall 0 \leq k \leq K$$

where g is a nonlinear transformation function, S a linear predicate over the transformed state variables $g(x(t_0), t_k)$ defining the safety requirements of the system.

As pointed out in [Tra20], minimizing the approximation error and the time complexity in reachable set computation is crucial to ensure the applicability of a reachability analysis approach in safety verification of practical NNCS.

Reachability Analysis of Neural Network Control Systems

From the architecture of a general NNCS as depicted in Figure 2.9, the reachability analysis can be outlined. First, from the initial set of states $\mathcal{X}_0$ of the plant, the controller $\mathcal{N}$ receives the output $y(t_0) \in \mathcal{Y}_0$ of the plant sampled at time step $k = 0$ as input from which it computes the control set $U_0 = \mathcal{N}(\mathcal{Y}_0, V_0)$. The control set is updated to $U = U_0$ and applied to the plant to compute the next state $\mathcal{X}_1 = \dot{\mathcal{X}}(t_0) = AX_0 + BU$. Iterative execution of the cycle yields the reachable sets $\mathcal{X}_k$ of the plant for time steps $0 \leq k \leq K$. The reachability analysis is now twofold: for the

controller, we need to compute an exact control set U, and for the linear plant, the exact reachable set of states $\mathcal{X}_k$.

First, using the previously presented star-based reachability algorithm from Section 2.2.1, for each time step k we can compute the exact control set $U_k = \mathcal{N}(C\mathcal{X}_k, V_k) = \bigcup_{j=1}^{l} \tilde{\Theta}_j$ as a union of stars for layer l.

Second, we obtain a union of stars for the exact reachable set of the plant $\mathcal{X}_{k+1} = A\mathcal{X}_k + BU_k$. Since the state set $\mathcal{X}_k = \langle c, V, P \rangle$ and the control set U_k are both defined based on a unique predicate variable vector α, for any star in the control set, its predicate P contains all linear constraints of the state set. Consequently, only a subset of the state set can induce an individual control set $\tilde{\Theta}_j$ where the predicates coincide. Therefore, the next state corresponding to the individual control set $\tilde{\Theta}_j = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j \rangle$ is $\mathcal{X}_{k+1} = \langle Ac + B\tilde{c}_j, AV + B\tilde{V}_j, \tilde{P}_j \rangle$ such that the exact next state set of the plant is $\mathcal{X}_{k+1} = \bigcup_{j=1}^{L} \mathcal{X}_{k+1}^{j}$. At this point, we note that the star-based reachability algorithm can be extended to non-linear plants as well. For further information, we refer to [Tra20].

Combining both observations, we can obtain the exact reachable set of an NNCS. In practice however, the exponentially increasing number of state sets in each cycle introduce infeasible computation times. Hence, we compute the interval hull of a set of stars after each step which can be achieved efficiently by solving a set of LP optimization problems.

Safety Verification of Neural Network Control Systems

Let $\bar{S}(z_k) = Hz_k \leq h$ define the unsafe region of an NNCS where $z_k = g(x_k)$ is a nonlinear transformation of the system's states x_k by a nonlinear function g. The task becomes to answer $Z_k \cap \bar{S} \stackrel{?}{=} \emptyset$ for all time steps $k = 0, \ldots, K$ and the transformed reachable set $Z_k = \{z_k \mid z_k = g(x_k),\ x_k \in \mathcal{X}_k\}$. Again, instead of computing the expensive exact transformed reachable set Z_k, we compute the tightest over-approximation $\tilde{Z}_k \supseteq Z_k$ with respect to interval bounds, thus by solving the nonlinear optimization problem

$$\tilde{Z}_k = \min_{z_k \in Z_k} \max_{\bar{z}_k \in Z_k} [\underline{z}_k, \bar{z}_k] \,.$$

The above considerations yield Algorithm 5.

Algorithm 5 Safety verification for an NNCS.

Input: Reachable set of NNCS R, transformation function g, unsafe region $\bar{S}$
Output: $safe = True$ iff no unsafe state reachable

```
safe ← True;
for k = 1 : length(R) do
    R_k ← R{1,k};
    z_k ← min(g(x_k)), z̄_k ← max(g(x_k)), x_k ∈ X_k;
    Z̃_k ← [z_k, z̄_k];
    if Z_k ∩ S̄ ≠ ∅ then                    ▷ Some unsafe state is reachable
        safe ← False;
        break;
    end if
end for
return safe;
```

Chapter 3

Reachability Analysis for Neural Network Controlled Hybrid Systems

In the previous chapter, we first laid the foundations for modeling and verifying hybrid systems, and further established an algorithm for star-based reachability analysis of neural networks that exploits the piece-wise linear definition of ReLUs. The subsequently presented flowpipe-construction based algorithm for safety verification of NNCS (Algorithm 5) provides a high-level framework for combining both approaches into one closed-loop system where in a cyclic manner, the neural network controller processes incoming measurements from the plant and generates control inputs to regulate the plant. However, this raises the question on how to exactly relate properties of both systems with each other to reason about the overall NNCS.

We shall now proceed as follows. Our goal is to develop and evaluate techniques for verifying the safety of NNCS in hybrid systems modeled by discrete interactions between a feedforward neural network with ReLU activation and a hybrid automaton. Hereby, we focus on non-autonomous hybrid systems with external inputs $u(t)$ and want to use the star-based reachability analysis proposed by Tran et al. [Tra20] on the side of the controller and flowpipe construction on the side of the plant. Further, we want to explore how to handle the additional uncertainty and complexity introduced by the neural network.

3.1 Conceptualization

In this section, we present an approach to address the challenge of verifying hybrid systems that are too complex to be modeled as hybrid automata. As outlined in the previous chapters, with the increasing application of neural networks as controllers in a wide range of domains, their integration into hybrid systems has gained significant attention as well. But to model a hybrid system as an NNCS, a suitable controller needs to be devised to ensure the system's safety. Further driven by the necessity to effectively benchmark and validate our proposed methods, we design a neural network controller based on an existing hybrid automaton and train the controller to accurately approximate the original automaton's behavior. In the NNCS setting, the controller

now governs the system's dynamics, and we thereby reduce the underlying hybrid automaton to a simple plant that merely consists of a finite set of flows.

Therefore we introduce an alternative framework where the controller predicts the control mode for the next time step based on the present state of the system. Subsequently, the reachability analysis algorithm constructs flowpipes from the controller input such that all potential trajectories are over-approximated during the reachability analysis. This consideration allows us to verify whether a system is safe in the presence of all admissible disturbances such as input noise and uncertainties in the system parameters. The resulting alternating process aims to provide a more comprehensive analysis of the system's behavior.

We implemented our approach using the hybrid system tool HyPro [SÁMK17] for easy integration with standard hybrid system models.

Moving forward, we will conduct a detailed analysis of the design choices made in this work to provide a clear justification for the approach taken.

3.1.1 Invocation of Neural Network

Interaction

In the context of control systems, the interaction between a controller and a plant can be broadly categorized into two main forms: continuous and discrete. In the case of continuous interaction, the neural network can access the state of the plant at any given time, allowing it to modify the control input to the plant at any time. Conversely, for discrete interaction, the neural network receives the state of the plant at each time step delta and modifies the control input accordingly using the actuators. The interaction frequency may vary based on the system's requirements and may be either regular or irregular.

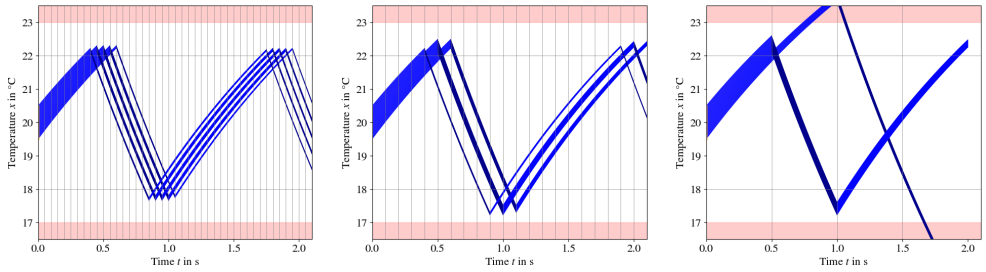
On the implementation level, continuous interaction is achieved through the use of a callback function that is triggered each time a new segment is generated during the flowpipe construction process, allowing the verification process to cover the real continuous interaction between the two. However, in practice, the verification approach at hand requires extensive computation. Specifically, the neural network reachability analysis is executed for each flowpipe segment, making the process too costly. As a consequence, our approach is restricted to handling discrete-time interactions that arise at either irregular or regular time intervals $\delta_1, \delta_2, \dots$ where either $\delta_i \neq \delta_j$ for some $i \neq j$ or $\delta_i = \delta_j$ for all $i \neq j$, respectively.

In the following sections, we make the assumption of a regular interaction frequency with a time interval of δ. Yet, we acknowledge that the formalism of our proposed algorithm is easily adaptable to accommodate irregular interactions.

Step Size

The step size, denoted as δ, is a crucial hyperparameter that requires careful selection to balance safety and computational effort in the control system. The step size must be small enough to maintain the stability of the closed-loop system and accurately capture the discrete changes in the plant. However, it must also be large enough to minimize computational cost and latency in real-time implementation. Finding the right step size involves trade-offs between these conflicting considerations.

From Figure 3.1, we can see that selecting a step size δ of too large size directly affects the safety of the system due to delayed feedback to the controller in Figure 3.1c.



(a) $\delta = 0.05$ (b) $\delta = 0.1$ (c) $\delta = 0.5$

Figure 3.1: Sampled trajectories simulating the NNCS behavior of the thermostat with initial interval $x \in [19.5, 20.5]$ and different step sizes δ. The vertical dashed lines denote the invocation of the neural network controller every δ time and the dashed horizontal lines the temperature threshold of switching the state. Entering the red region denotes a violation of the safety condition.

On the other hand, if chosen very small as in Figure 3.1a, the computation becomes expensive. Thus it seems sensible to choose a middle ground (see Figure 3.1b) between both extremes.

3.1.2 Modeling the Hybrid System

After removing the guards and the invariants of the original hybrid automaton $\mathcal{H}$, we consider the system as a simple plant that merely covers all the possible flows of the system where each flow is uniquely labeled.

Definition 3.1.1 (Reduced hybrid automaton)**.** *Let $\mathcal{H}$ be a hybrid automaton as specified in Definition 2.1.1 with (Loc, Var, Con, Lab, Edge, Act, Inv, Init). We transform it into a reduced representation*

$$\mathcal{H}' = (Loc', Var', Lab', Act', Init')$$

such that

$$Loc' = \{\ell\}, \quad Var' = Var, \quad Lab' : Act \to \{1, \ldots, m\} \times Var,$$
$$Act'(\ell) = \bigcup_i Act(\ell_i), \quad Init' \subseteq \Sigma'$$

where $Loc = \{\ell_1, \ldots, \ell_m\}$, $m \in \mathbb{N}$ and $\Sigma' = Act'(\ell) \times V$.

Example 3.1.1 (Reduced thermostat hybrid automaton)**.** *Following Definition 3.1.1, we transform the hybrid automaton from Example 2.1.2 into a reduced hybrid automaton. Formally, the automaton becomes*

- $Loc' = \{\ell\}$,
- $Var' = \{x\}$,
- $Lab'(Act(\ell_{on})) = (1,x)$, $Lab'(Act(\ell_{off})) = (2,x)$,

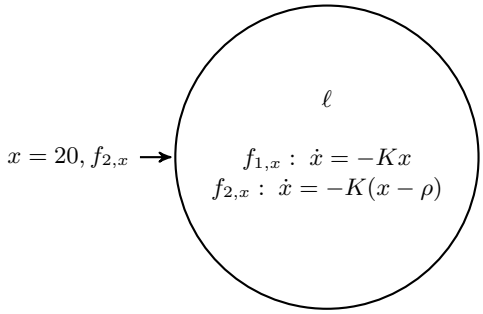


Figure 3.2: Modeling the hybrid automaton as a set of ODEs.

- $Act'(\ell) =$

$$\{f_{1,x} : \mathbb{R}_{\geq 0} \to V \ \textit{s.t.}\ f_{1,x}(t)(x) = 20e^{-Kt} + h\left(1 - e^{-Kt}\right),$$
$$f_{2,x} : \mathbb{R}_{\geq 0} \to V \ \textit{s.t.}\ f_{2,x}(t)(x) = 20e^{-Kt}\},$$

- $Init' = \{(f_{1,x}, \nu) \in \Sigma' \mid \nu(x) = 20\}$.

The resulting automaton is illustrated in Figure 3.2.

3.1.3 Modeling the Neural Network Controller

When modeling the neural network controller, the question of the type of output it should produce arises. Essentially, there are two broad categories of options to consider. The first is training the network to produce a continuous value from an uncountable but usually bounded set, and the second is training it to produce a discrete value from a finite and countable set.

In the first case, the output value could be akin to that of a PID controller and represent a variable in the hybrid system's flow function, which is the most common approach in the literature. If this approach is taken, it is expected that the system will eventually converge to a stable state. However, this approach is not applicable to all use-cases.

In the second case, the network could be treated as a traditional multi-class classifier that predicts the flow being applied to the system. For example, if there are m flows, the controller could output confidence scores for each flow in an m-dimensional vector using a softmax activation function. However, this type of output can be difficult to interpret during reachability analysis, so this approach is not pursued.

Alternatively, the neural network's output could be modeled as a discrete value for a variable in the hybrid system's flow function. Note how the flow of non-autonomous systems is defined as

$$\dot{x}(t) = Ax(t) + Bu(t)$$

according to Equation 2.2. Let $Loc = \{\ell_m : m \leq n\}$ for an $n \in \mathbb{N}$ be a finite and countable set of locations, such that we can uniquely denote a location by some index $m \in \mathbb{N}$. Further, let only a single variable $x \in Var$ be of interest for the practical applicability of plotting a two-dimensional reachability plot. If we now assume the

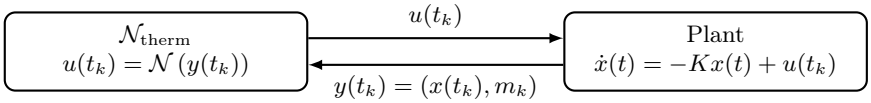


Figure 3.3: Feedback loop on the example of the thermostat at discrete time steps t_k where the controller senses the state $x(t_k)$ and the current mode m_k of the plant, evaluates the input and outputs a control $u(t_k) \in \{0, Kh\}$.

flows of the variable x of a given hybrid automaton $\mathcal{H}$ can be distinctly differentiated by a constant such that

$$\dot{x}(t) = Ax(t) + h_i$$

for $h_i \in \mathbb{R}$, then we can model the external inputs from the controller as $u(t) = h_i$.

The approach that has been outlined lastly is the one that we will be adopting and incorporating into the subsequent modeling of the neural network controller. Subsequently, we will take the neural network to be a classifier and thus the outputs h_i as its classes indicating the control input of the hybrid system.

Example 3.1.2 (Modeling the output for the thermostat controller)**.** *The two flows are given as $\dot{x} = -Kx$ and $\dot{x} = K(\rho - x) = -Kx + K\rho$ for the thermostat being turned off or on respectively. Thus the neural network is designed to predict 0 or $K\rho = 15$ as illustrated in Figure 3.3.*

3.1.4 Designing the Neural Network

Given a hybrid automaton $\mathcal{H}$ with d variables $Var = \{x_0, \ldots, x_{d-1}\}$ and a countable set of locations $Loc = \{\ell_m : m \leq n\}$, we define the input dimension for the neural network to be $d + 1$ such that the input to the neural network is a real-valued tuple $(x_0, \ldots, x_{d-1}, m)$ of the variables and the current flow denoted by an index $m \in \mathbb{N}$. The hidden layers are straightforward and restricted to ReLU activation functions to satisfy the requirements of the exact star-based reachability approach used during the analysis.

For the output layer, we implement a classification layer using step functions, formally a function that can be written as a finite linear combination of indicator functions of intervals.

Definition 3.1.2 (Step function)**.** *A* step function $f_{step} : \mathbb{R} \to \mathbb{R}$ *is defined as*

$$f_{step}(x) = \sum_{i=1}^{n} \alpha_i \chi_{A_i}(x) \tag{3.1}$$

where $\alpha_i \in \mathbb{R}$ are real coefficients, $n > 0$, χ_{A_i} is the indicator function such that

$$\chi_{A_i}(x) = \begin{cases} 1, & \text{if } x \in A_i, \\ 0, & \text{otherwise} \end{cases}$$

for each interval A_i, $i = 1, \ldots, n$.

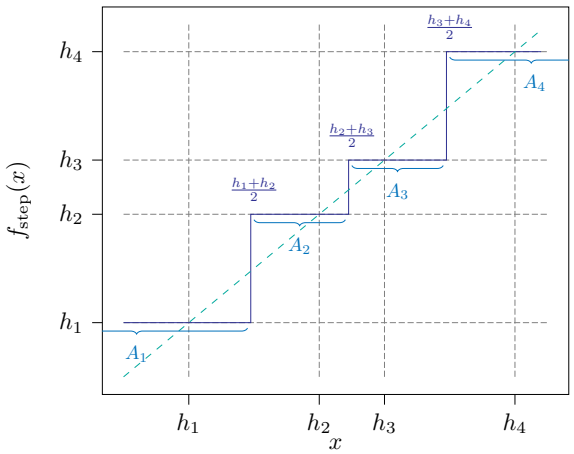


Figure 3.4: Step function for ordered classes h_1, h_2, h_3, h_4 and such that the corresponding intervals A_i with $i = 1, \ldots, 4$ are as defined in Equation 3.2.

Moving forward, we assume $A_i \subset \mathbb{R}$. For ordered target outputs $h_i \in \mathbb{R}$ with $i = 1, \ldots, n$ and $h_{i-1} < h_i$, the neural network employs the step function with intervals

$$A_i = \begin{cases} (-\infty, \frac{h_i+h_{i+1}}{2}), & \text{if } i = 1, \\ [\frac{h_i+h_{i-1}}{2}, \infty), & \text{if } i = n, \\ [\frac{h_{i-1}+h_i}{2}, \frac{h_i+h_{i+1}}{2}), & \text{otherwise} \end{cases} \tag{3.2}$$

and $\alpha_i = h_i$ for classifying the intermediate output value. This definition simply rounds a number x to the closest class h_i as depicted in Figure 3.4.

Example 3.1.3 (Classifying the output for the thermostat)**.** *For inputs $x \in \mathbb{R}$ and $m \in \{0,1\}$, the desired input-output mapping of the neural network $\mathcal{N}_{th}$ is*

$$\mathcal{N}_{th}(x,m) = \begin{cases} 0, & \text{if } x \geq 22, \\ 15, & \text{if } x \leq 18, \\ m \cdot 15, & \text{otherwise.} \end{cases}$$

This can equally be achieved by using the step function as the last layer where $A_1 = (-\infty, 7.5), A_2 = [7.5, \infty)$ and $\alpha_1 = 0, \alpha_2 = 15$. Let $\mathcal{N}'_{th}$ denote $\mathcal{N}_{th}$ without the final classification layer, such that for an intermediate output $x' = \mathcal{N}'_{th}(x,m)$ we get

$$f_{step}(x') = 0 \cdot \chi_{A_1}(x') + 15 \cdot \chi_{A_2}(x') = \begin{cases} 0, & \text{if } x' < 7.5, \\ 15, & \text{if } x' \geq 7.5. \end{cases}$$

Overall, we can now define a specific neural network controller and train it to achieve the desired behavior.

Example 3.1.4 (Neural network architecture for the thermostat)**.** *In the context of the thermostat example, we devised a straightforward feedforward neural network consisting of three layers. The input layer comprises of two neurons to represent temperature $x \in \mathbb{R}$ and mode (on or off), denoted by $m \in \{0, 1\}$. The network further*

comprises two hidden layers, each having ten neurons. Finally, the output layer of the network generates a single prediction for $Kh \in \{0, 15\}$.

The hyperparameters used in the network implementation are as follows: the network was trained on 15000 samples, with a 80/20 train-validation split. The batch size was set to 32 and the number of epochs was set to 300. The mean squared error (MSE) loss was used as the loss function, and the learning rate was set to 0.0012. The Adam optimizer was used with weight decay of 1e-5. The accuracy of the network was tested on 3750 samples, and it was found to be 98.40%.

3.2 NNCS Analysis

As we advance, we take it as given that the two components of the NNCS, specifically the trained neural network and the plant, are as previously defined. We will proceed by describing the interaction between them and how the algorithm operates during the safety verification. Note that in the subsequent, the neural network is employed for the exact star-based method of the reachability algorithm as laid out in the Preliminaries in Algorithm 2.

3.2.1 Input Star Definition

At time t_{k-1}, the plant constructs flowpipes for δ time from an initial set $\mathcal{I}_{k-1}$. The reachable set $\mathcal{R}_k$ with state variables $x_0, \ldots, x_{d-1}$ and the mode m_k are used to create an input star set $\Theta_{I_k} = \langle c, V, P \rangle$ such that

$$c = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}, \quad V = \{e_1, \ldots, e_{d'}\}, \quad P(\alpha) \triangleq C\alpha \leq s \tag{3.3}$$

where $d' = d + 1$ is the dimension of the input to the neural network, the center c is a d'-dimensional zero vector, V is the d'-dimensional standard basis, and the constraints $C \in \mathbb{R}^{2d' \times d'}$ are defined as

$$C_{ij} = \begin{cases} -1, & \text{for } i = 2j \\ 1, & \text{for } i = 2j + 1 \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

where $0 \leq j < d'$. We specify the limits $s \in \mathbb{R}^{2d'}$ by requiring

$$s_{2i+1} = -x_{i_l} \tag{3.5}$$

$$s_{2i+2} = x_{i_u} \tag{3.6}$$

for $0 \leq i < d$, and $s_{2d+1} = -m_k$ and $s_{2d+2} = m_k$. Since the reachable set $\mathcal{R}_k$ is a set, it potentially covers an interval $[x_{i_l}, x_{i_u}]$ in each dimension $i \leq d'$. The intuition behind this definition is to convert the reachable set into an equivalent star set that satisfies the same conditions for each state variable while also including the system's mode. Consequently, each element $y \in \Theta$ in this star set satisfies $y_i \geq x_{i_l}$, $y_i \leq x_{i_u}$ and $y_{d'} = m_k$.

We note that the aforementioned definition is applicable only when the initial set is represented as a d-dimensional interval, and it has to be adjusted accordingly to

also hold for other types of representations such as polytopes. It is worth mentioning that HyPro, at the implementation level, defines the initial set exclusively in terms of intervals, and thus we can safely make this assumption.

Example 3.2.1 (Input star set for the thermostat)**.** *Let $x_{k_l}, x_{k_u} \in \mathbb{R}$ be the lower and upper bounds of the temperature, and $m_k \in \{0,1\}$ the mode the system is in at time point t_k. Then the input star set for the controller $\Theta_I = \langle c, V, P \rangle$ with $P(\alpha) \triangleq C\alpha \leq s$ is defined as*

$$c = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad V = \{e_1, e_2\}, \quad C = \begin{bmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{bmatrix}, \quad s = \begin{bmatrix} -x_{k_l} \\ x_{k_u} \\ -m_k \\ m_k \end{bmatrix}.$$

3.2.2 Neural Network Reachability Analysis

Let X be the set of all inputs to the neural network controller $\mathcal{N}$ and $Y = \mathcal{N}(X)$ the set of all outputs. As defined above, the input star set $\Theta_I \in X$ is then passed to the controller that performs the exact star-based neural network reachability analysis (Algorithm 2 and Algorithm 6). By definition, the output is again a set of stars such that for $Y_i \in Y$, $Y_i = \langle c, V, P \rangle$. Assuming the neural network gets a d'-dimensional input point during run-time, it now receives a d'-dimensional input set with $2d'$ constraints during the analysis to handle sets spanning intervals in each feature dimension. Analogously, the output covers an interval instead of a single number.

More specifically, given the ordered target classes $h_1, \ldots, h_n$ during run-time, for each output star $Y_i \in Y$ during the analysis, we obtain a set of intervals $\mathcal{J}_i = \{J_{i_1}, \ldots J_{i_{n'}}\}$ with trivial intervals $J_{i_j} = [h_{i_j}, h_{i_j}]$ and $i_j = i_{j-1} + 1 \leq n$. To account for the possibility that $Y_i \cap Y_j = \emptyset$ for some output stars Y_i, Y_j, the subsequent flowpipes have to over-approximate all the possible trajectories that are induced. That means that if there are predictions for classes h_{j-1} and h_{j+1}, but not for h_j, we need to consider the trajectory induced by the mode h_j as well. Formally, let $\mathcal{J} = \bigcup_i \mathcal{J}_i$, then we obtain all the predicted modes as $Y' = \{h_{i'} : [h_{i'}, h_{i'}] \in \mathcal{J}\}$ which by definition are each uniquely associated with respective flows $\dot{x} = Ax(t) + h_{i'}$. To take all the possible trajectories into account, we derive the indices of the minimum and maximum value of the predicted classes

$$i_{min} = \arg\min_{i'} \{h_{i'} : h_{i'} \in Y'\}, \quad i_{max} = \arg\max_{i'} \{h_{i'} : h_{i'} \in Y'\} \tag{3.7}$$

to also pass on the intermediate classes h_i with $i_{min} < i < i_{max}$ for inferring the respective flows. This process is embedded in the exact star-based reachability analysis for the staircase function in Algorithm 6. Overall, this leads to the potential branching of the created flowpipes over time.

3.2.3 Flowpipe Construction Using Control Input

The following objective at hand is to specify from the predicted modes of the controller the input to the plant to perform flowpipe construction. Recall that flowpipe construction is rooted in the idea of successively applying an affine transformation, defined by the flow at the current location, on an initial set at time t_1 until some specified time point t_2, considering the convex hull of both states and bloating this

Algorithm 6 Star-based exact reachability analysis for one layer with step function.

Input Input star set $I = [\Theta_1 \cdots \Theta_N]$, ordered classes $[h_1 \cdots h_m]$ such that intervals $[A_1 \cdots A_m]$ are as defined in Equation 3.2

Output Exact reachable set $\mathcal{R}$

procedure LAYERREACH(I, W, b)
  $\mathcal{R} \leftarrow \emptyset$;
  for $j = 1 : N$ **do**
    $I_1 \leftarrow W * \Theta_j + b = \langle Wc_j + b, WV_j, P_j \rangle$;
    $\mathcal{R}_1 \leftarrow I_1$;
    for $i = 1 : n$ **do**
      $[l_i, u_i] \leftarrow I_1.range(i)$; ▷ $l_i \leq x_i \leq u_i, x_i \in I_1[i]$
      $\mathcal{R}_1 \leftarrow$ STEPSTAIRCASE($\mathcal{R}_1, i, l_i, u_i$);
    end for
    $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_1$;
  end for
  return $\mathcal{R}$;
end procedure
procedure STEPSTAIRCASE($\tilde{I}, i, l_i, u_i$)
  $\tilde{\mathcal{R}} \leftarrow \emptyset$;
  $\tilde{I} = [\tilde{\Theta}_1 \ \cdots \ \tilde{\Theta}_k]$;
  $M \leftarrow [e_1 \ e_2 \ \cdots e_{i-1} \ 0 \ e_{i+1} \cdots e_n]$; ▷ Intermediate representations
  for $j = 1 : k$ **do**
    $\mathcal{R}_1 \leftarrow \emptyset$;
    $\tilde{\Theta}_j = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j, \rangle$;
    $i_{min} \leftarrow \arg\min_{i'} \{h_{i'} \mid l_i \in A_{i'}, i' \leq m\}$; ▷ Index of smallest class
    $i_{max} \leftarrow \arg\max_{i'} \{h_{i'} \mid u_i \in A_{i'}, i' \leq m\}$; ▷ Index of greatest class
    for $j' = i_{min} : i_{max}$ **do**
      $v_1 \leftarrow h_{j'} \cdot e_i$;
      $\tilde{\Theta}'_{j'} \leftarrow \langle M\tilde{c}_j + v_1, M\tilde{V}_j, \tilde{P}_j, \rangle$;
      $\mathcal{R}_1 \leftarrow \mathcal{R}_1 \cup \tilde{\Theta}'_{j'}$;
    end for
    $\tilde{\mathcal{R}} \leftarrow \tilde{\mathcal{R}} \cup \mathcal{R}_1$;
  end for
  return $\tilde{\mathcal{R}}$;
end procedure

hull by a ball of the approximated error to finally obtain the reachable set $\mathcal{R}_{[t_1,t_2]}$. Accordingly, it is necessary to determine the initial set for the next flowpipe construction when passing the control output to the plant. We propose to record the initial sets for each flowpipe by inference from the initial set $\mathcal{I}_0$ of the entire analysis. Therefore we can establish the following recurrence relation between the initial set for the current time point t_k and the previous time point t_{k-1}.

Definition 3.2.1 (Initial set of NNCS analysis)**.** *Let $(X, U, F, \mathcal{N}, \delta, \mathcal{X}_0)$ be an NNCS and $T = K\delta$ the time horizon of the analysis. Then the initial set $\mathcal{I}_k$ at time point*

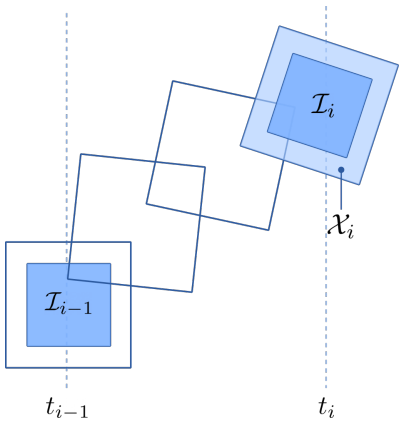


Figure 3.5: For the current set information $\mathcal{S}_i = \{(\mathcal{X}_i, F, \mathcal{I}_{i-1})\}$ at time t_i, the old initial set from which the segment $\mathcal{X}_i$ was constructed is transformed by the flow F to obtain $\mathcal{I}_i$. The dashed lines denote times of invocation of the neural network.

$t_k = k\delta$ is defined inductively by

$$\begin{aligned} \mathcal{I}_0 &:= \mathcal{X}_0 \\ \mathcal{I}_k &:= e^{\delta A_{k-1}} \mathcal{I}_{k-1}, \quad k = 1, \ldots, K \end{aligned}$$

where A_{k-1} is the flow matrix of the flow $\dot{x} = A_{k-1}x$ over the time interval $[t_{k-1}, t_k]$.

In practice, this means that at time point t_k once the controller outputs the predicted modes, the former initial set for each flowpipe from the previous time point t_{k-1} gets propagated by an affine transformation for δ time. These affine transformations correspond to the flows of the predicted modes. The advantage of adopting this definition is that it effectively minimizes the wrapping effects. This is due to the fact that our approach involves propagating only the original initial sets, as opposed to considering the entire flowpipe segments at a jump, which is the conventional method employed in flowpipe construction.

3.2.4 Algorithm Implementation

Having thoroughly discussed the various components of the NNCS analysis, we will proceed to integrate them into a comprehensive algorithm, as depicted in Algorithm 7.

Suppose we have an NNCS $(X, U, F, \mathcal{N}, \delta, \mathcal{X}_0)$ where the hybrid system is modeled by a reduced hybrid automaton $\mathcal{H}' = (Loc', Var', Con', Act', Init')$ as proposed in Definition 3.1.1 with initial set $\mathcal{I}_0$, and a global time horizon $T = K\delta$.

The first step involves the execution of the flowpipe construction on the initial set $\mathcal{I}_0$ with the initial flow F_0 for δ time. This results in the over-approximate computation of the reachable sets $\mathcal{R}_{[0,\delta]} = \{\nu \mid \nu = e^{tA}x_0, t \in [0, \delta], x_0 \in N_0\}$, A being the flow matrix of F_0. For simplicity, the location is omitted as the reduced hybrid automaton consists of only a single location. Then, the relevant sets, i.e. those that intersect the time point δ, are extracted and stored in the variable $\mathcal{X}_i$.

As the flowpipe construction may lead to branching into multiple flowpipes, it is necessary to keep track of the relevant sets, the initial sets from which they were

Algorithm 7 NNCS analysis (discrete).

Input: NNCS $(X, U, F, \mathcal{N}, \delta, \mathcal{X}_0)$, reduced HA $\mathcal{H}$ with initial set $\mathcal{I}_0$
Output: Set of reachable states $\mathcal{R}$ over time interval $[0, K\delta]$

```
1: procedure NNCSREACH(I_0, K)
2:     R ← COMPUTEFLOWPIPE(I_0, δ);
3:     X_i ← GETRELEVANTSETS(R, δ);                ▷ Get sets that intersect time δ
4:     S_{i-1} ← {(X_i, F_0, I_0)};                ▷ Current branch: set, initial flow, initial set
5:     for i = 1 to K do
6:         S_i ← ∅;
7:         for (X_j, F_j, I_j) ∈ S_{i-1} do
8:             (X', F', I') ← COMPUTESETFLOWMAPS(X_j, F_j, I_j);
9:             S_i ← S_i ∪ {(X', F', I')};
10:        end for
11:        for j = 1 to |S_i| do
12:            R_j ← COMPUTEFLOWPIPE(I_j, δ);
13:            R ← R ∪ R_j;
14:        end for
15:        S_{i-1} ← S_i;
16:        X_i ← GETRELEVANTSETS(R, (i + 1)δ);
17:    end for
18:    return R
19: end procedure
20: procedure COMPUTESETFLOWMAPS(X, F, I)
21:    I ← MAKEINPUTSTAR(X, F);                    ▷ As defined in Section 3.2.1
22:    R ← NNFORWARDANALYSIS(I);                   ▷ Uses Algorithm2
23:    S = ∅;
24:    I' ← AFFINETRANSFORMATION(I, F, δ);         ▷ Using Definition 3.2.1
25:    for Θ_i ∈ R do
26:        F' ← Ax + Θ_i.vertices_1;               ▷ Θ_i.vertices_1 is the control input
27:        S ← S ∪ (X, F', I');
28:    end for
29:    return S
30: end procedure
```

generated, and the corresponding flows that transformed them. To this end, we maintain this information in a set of triples

$$\mathcal{S}_{i-1} = \{(\mathcal{X}_i, F, \mathcal{I}) \mid \mathcal{X}_i \subseteq V', F \in \mathit{Act}'(\ell), \mathcal{I} \in \mathit{Init}'\}.$$

We visualize the set information triple in Figure 3.5 and elaborate on its function further below.

Proceeding to the central loop iterating over the K time steps, it can be divided into two parts. In the first part, we calculate all conceivable system behaviors, which correspond to possible combinations of flow and state. In the second part, we perform flowpipe construction for each combination generated from the previous iteration and accumulate the resulting reachable states.

The key computation is carried out by the method *computeSetFlowMaps* in Line 8 mapping reachable sets of the hybrid system to flows through the application of a neural network controller to generate control inputs for the plant, thus establishing a

mapping between the sets and the corresponding flows. Starting from the set information triple $\mathcal{S}_{i-1}$, the method first computes an input star on which it applies the neural network reachability analysis, resulting in a reachable set $\mathcal{R} = \{\Theta_1, \ldots, \Theta_n\}$ of stars. Then, the flow matrix of the flow F from the previous time step applies an affine transformation on the initial set $\mathcal{I}$ by δ time to obtain $\mathcal{I}'$. For each reachable star set Θ_i, the new flow F' is inferred from the predicted mode h_j, the interval bounds $[h_j, h_j]$ of the one-dimensional output star Θ_i, by extracting the output star set's vertices. Since by definition of the controller predictions h and a flow being described as $\dot{x} = Kx + h$, we obtain the corresponding flow matrix. The set of set information triplets is extended accordingly by the updated triple $(\mathcal{X}, F', \mathcal{I}')$. Overall, this method allows for the effective manipulation of the system's behavior through the selection of appropriate control inputs.

For each element in the set information set $\mathcal{S}_i$, we construct a new flowpipe for δ time from the corresponding initial sets and add them to the reachable sets. Finally, we update the set information and the sets intersecting the following time step $(i+1)\delta$. This procedure is repeated until the global time horizon of T is reached.

We now show that the sequence Ω_k, the set of reachable states using the discrete NNCS analysis, covers the set of actual reachable states $\mathcal{R}$.

Theorem 3.2.1. *Let $\mathfrak{N} = (X, U, F, \mathcal{N}, \delta, \mathcal{X}_0)$ be an NNCS. Then*

$$\mathcal{R}_{[0,\delta]}(\mathcal{X}_0) \subseteq \Omega_{[0,\delta]}(\mathcal{X}_0, \mathcal{U}). \tag{3.8}$$

Proof. For the entire analysis to satisfy Equation 3.8, it suffices to show that all building elements satisfy the statement for interval $[0, \delta]$. We provide the proof by reviewing Algorithm 7 for one iteration. The validity of the statement is straightforward to see for Line 2, as shown in Section 2.1.2, flowpipe construction is already a valid over-approximation. Let $\bar{\mathcal{R}}_{[0,\delta]}$ be the result of flowpipe construction. For the first controller invocation at time δ, the reachable set $\mathcal{X}_i \subseteq \bar{\mathcal{R}}_{[0,\delta]}$ with $\mathcal{X}_i \cap \mathcal{R}_{[\delta,\delta]}$ is used to obtain the current state variable valuations for creating the input star Θ_I as outlined in Section 3.2.1, which by design encompasses the reachable set at time δ. By definition of the exact star-based reachability algorithm in Algorithm 2, the reachable states $\mathcal{N}(\Theta_I) = \mathcal{R}_{L_k}$ of a k-layer neural network controller are exactly computed. Lastly, it remains to consider the initial set transformation. As defined in Definition 3.2.1, the initial set for the first time step is thus $\mathcal{I}_k = e^{\delta A_0}\mathcal{X}_0$, for the flow matrix A_0 over $[0,\delta]$. Hence $\mathcal{I}_k \subseteq \bar{\mathcal{R}}_{[0,\delta]}$. By design of the controller, the prediction yields a valid flow such that in the next iteration, flowpipe construction can be applied to the transformed initial set. □

As for flowpipe construction, iterative execution of the algorithm results in the over-approximation over the global time horizon $T = K\delta$ such that

$$\mathcal{R}_{[0,T]}(\mathcal{X}_0) \subseteq \Omega_{[0,T]}(\mathcal{X}_0, \mathcal{U}) = \bigcup_{i=1}^{K} \Omega_{[(i-1)\delta, i\delta]}(\mathcal{I}_{i-1}, \mathcal{U}).$$

Now that we know the NNCS analysis is indeed an over-approximation of the true reachable states, it follows directly that Algorithm 7 is sound, meaning that a system is considered safe iff no unsafe states can be reached.

Theorem 3.2.2. *Let $\mathfrak{N} = (X, U, F, \mathcal{N}, \delta, \mathcal{X}_0)$ be an NNCS, $I \subseteq X$ the initial state set, $\bar{S} \subseteq X$ the set of bad states. Then $\mathfrak{N}$ is safe iff $\Omega_{[0,T]}(I) \cap \bar{S} = \emptyset$.*

Chapter 4

Evaluation

4.1 Benchmarks

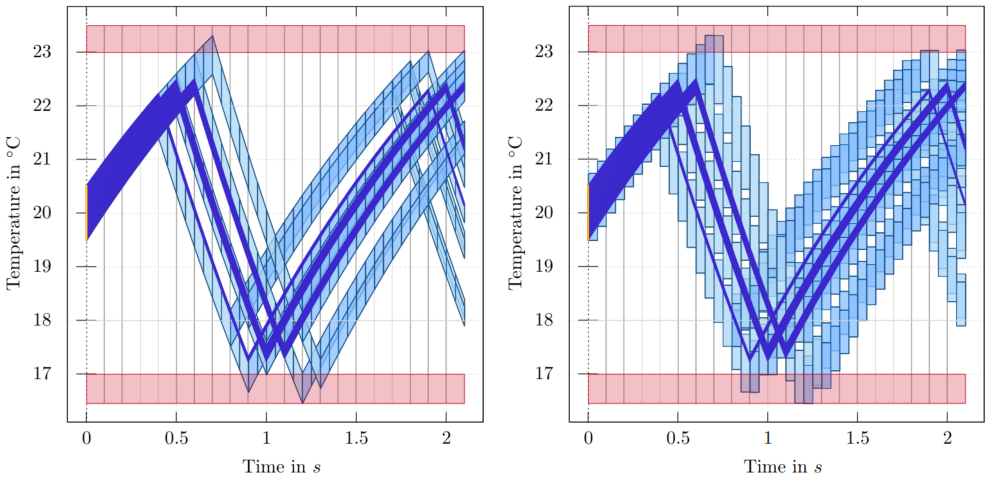
Moving forward, we use two benchmarks to evaluate our proposed approach. The following benchmarks are compiled from related literature and were selected to cover non-autonomous hybrid systems with flows that can be decomposed the same way as described in Section 3.1.3. The first one, the thermostat, was already introduced as a simple running example to illustrate the definitions of our method, while the second benchmark is more sophisticated and exhibits non-deterministic switching between different control modes while still maintaining a relatively small size.

4.1.1 Thermostat

Having laid the theoretical groundwork for the constituent components of our approach in Chapter 3 and demonstrated their effectiveness through complementary examples with the thermostat benchmark, we are now prepared to present the results of our reachable set computation. In this section, we assume that the components have been defined as described in Chapter 3.

Example 4.1.1 (NNCS analysis of the thermostat)**.** *In Figure 3.3, we report on two results of the application of Algorithm 7 on the thermostat example using different state set representations to compute the reachable sets of the NNCS. The system uses both the neural network as specified in Example 3.1.4 as well as the reduced hybrid automaton from Example 3.1.1. For the plots, the controller gets invoked every* $\delta = 0.1$ *time units over a global time horizon* $T = 2.1$*.*

In Figure 3.1, the system is shown to be safe with the choice of step size $\delta = 0.1$ with respect to the trajectories. However, the safety verification results obtained in Figure 4.1 indicate that the lower temperature bound is exceeded between time 0.8 and 1.4, and the system enters the unsafe region. This behavior can be attributed to the introduction of an additional error by the controller. This is because the neural network only makes a binary decision, which is either the thermostat being turned on or off, and does not take into account any qualitative aspects such as the degree by which the temperature is adjusted. Therefore, the full initial set for flowpipe construction is either propagated or not in each time step, leading to potentially oversized flowpipe segments, which can cause such deviations.



(a) $\mathcal{V}$-polytope representation (b) Box representation

Figure 4.1: Result of NNCS analysis on the thermostat example with $\delta = 0.1$ and different representations. The dark blue curves on the figures are sampled trajectories, and the underlying light blue curves are the over-approximated sets by the reachability analysis where the borders accentuate each segment. The initial set with temperature $x \in [19.5,20.5]$ and flow $f_{1,x}$ at time 0 is indicated by an orange line.

We further observe that the result does not suffer from strong wrapping effects, which refers to the over-approximation errors introduced during computation. due to the requirement of closure for all representations. We attribute this phenomenon to mainly two reasons: first, at all times, we are merely propagating the original initial set and second, there are no rotations present in the affine transformations induced by the flows.

4.1.2 Rod Reactor

The second selected benchmark, commonly referred to as the rod reactor, presents an intriguing subject for analysis as it involves multiple state variables and exhibits unique properties despite its relative simplicity in structure. To provide context for the problem, we will initially present a high-level description.

Example 4.1.2 (Rod Reactor [ACH+95, SÁMK17])**.** *The rod reactor system is a simplified model of the reactor core of a nuclear power plant and features fuel rods that heat the surrounding water to power turbines and cooling rods made from a material that inhibits the radioactive reaction. These cooling rods can be lowered in between the fuel rods to regulate the temperature. The system models the reactor temperature depending on the state of two different cooling rods that can be extended or retracted. Since the cooling rods are made from different materials, they have a different influence on the temperature dynamics when used. Without any cooling rods inserted, the temperature x rises according to the ODE $\dot{x} = 0.1x - 50$. If the first rod is inserted, the temperature falls by $\dot{x} = 0.1x - 56$, and by $\dot{x} = 0.1x - 60$ for the second rod. At most one rod can be inserted at a time.*

The controller can decide which cooling rod to use whenever the temperature ex-

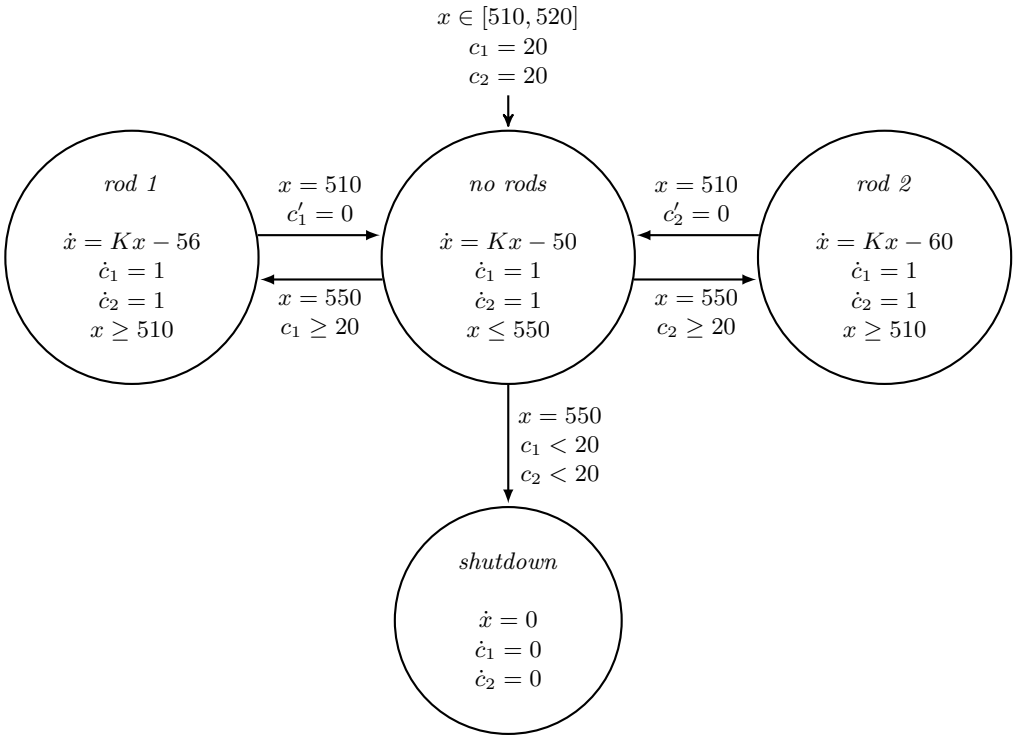


Figure 4.2: The hybrid automaton model for the rod reactor system.

ceeds a fixed boundary of 550 °C but cannot use the same rod for 20 time units after it has been used. It thus controls the coolant temperature in a reactor tank by moving two independent control rods. However, allowing the temperature to rise or fall beyond 600 °C or 500 °C, respectively, leads to a meltdown or shutdown.

The presented description gives rise to four distinct states, depending on whether no rods or either the first or the second rod is inserted, or the system is shut down. The system's unsafe state is represented by the latter configuration. The dynamic constraints further impose restrictions on the system's behavior. Specifically, if the temperature falls below 500 $°C$, it can only decrease. The first rod is effective in cooling down the reactor core if the temperature is below 560 $°C$, while the second rod can achieve the same goal only if the temperature is below 600 $°C$. As a result, they must be retracted before the temperature exceeds 500 $°C$.

The issue of ensuring the safe operation of the reactor cooling system necessitates the development of a suitable control strategy. In practice, designing a safe and effective control strategy would typically require learning through techniques such as reinforcement learning. However, given the scope of this study, we opted to adopt a control strategy previously proposed in the literature, as modelled by a hybrid automaton [Sch19].

Example 4.1.3 (Hybrid automaton for rod reactor)**.** *The goal is to maintain the coolant between temperatures 510 °C and 550 °C. When the temperature reaches its maximum value 550 °C, the tank must be refrigerated with one of the rods. The*

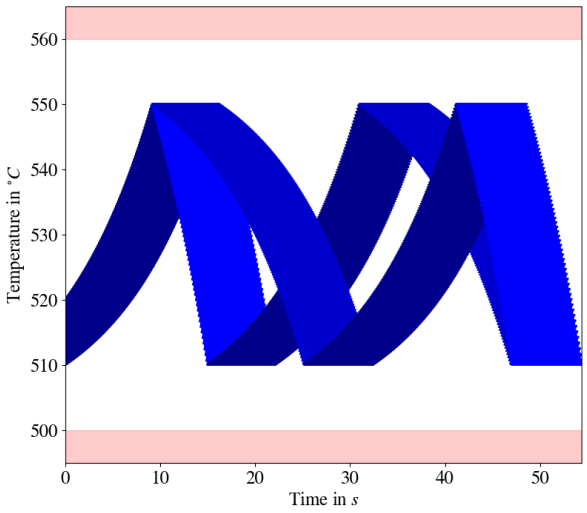


Figure 4.3: Flowpipe construction for the rod reactor example using box representation with a step size of 0.1 and a jump depth of 3, colored by flow.

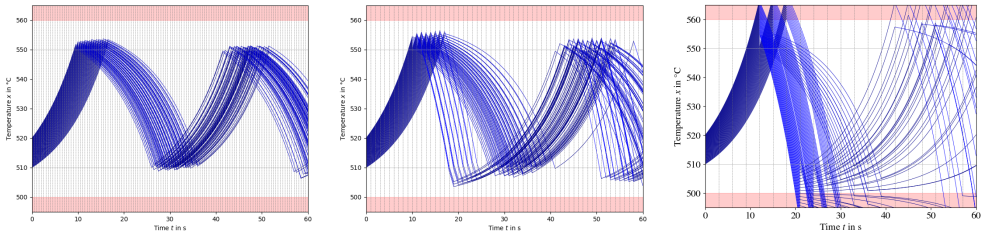
temperature rises at a rate $\dot{x} = Kx - 50$ *and decreases at rates* $\dot{x} = Kx - 56$ *and* $\dot{x} = Kx - 60$ *depending on which rod is being used. A rod can be inserted again only if 20 seconds have elapsed since the last time it was removed. If the temperature of the coolant cannot decrease because there is no available rod, a complete shutdown is required. Figure 4.2 illustrates the hybrid automaton of this example: variable* x *measures the temperature and the values of clocks* c_1 *and* c_2 *represent the times elapsed since the last use of the first and second rod, respectively. In practice, we find* $K = 0.1$ *to guarantee a safe behavior.*

In Figure 4.3, we illustrate the reachable state sets obtained from flowpipe construction to demonstrate that the system under consideration satisfies the safety criteria. The first rod insertion exhibits non-deterministic behavior, while the system's behavior becomes fully deterministic thereafter. Specifically, both rods are inserted in an alternating manner, and the process repeats periodically, ensuring that the system does not enter any unsafe state. Neither the shutdown mode is reached nor are the critical temperatures exceeded. This observation motivates the training of a neural network controller based on the control strategy established by the hybrid automaton depicted in Figure 4.2.

Example 4.1.4 (Neural network for rod reactor)**.** *In Section 3.1.4, we present the design of the neural network's input features to encompass the hybrid automaton's state variables and the current mode, uniquely represented by an index. This allows us to create an input tuple* (x, c_1, c_2, m) *for the temperature, clocks of the first and second rod, and the indexed mode.*

Given that the flows of the state variable x *follow the form* $\dot{x} = Kx - h_i$ *for* $h_1 = 50, h_2 = 56, h_3 = 60$ *and the shutdown mode is represented by* $\dot{x} = 0$*, we construct the neural network to predict the next mode from the set* $\{50, 56, 60, 0\}$.

To perform the analysis, we adopt a specific architecture consisting of a four-layer feedforward neural network with an input dimension of four, hidden layers of sizes 10,



(a) $\delta = 0.5$ (b) $\delta = 1.0$ (c) $\delta = 3.0$

Figure 4.4: Sampled trajectories simulating the NNCS behavior of the rod reactor with initial interval $x \in [510, 520]$ and different step sizes δ. The vertical dashed lines denote the invocation of the neural network controller every δ time and the dashed horizontal lines the temperature threshold of switching the state. Entering a red region denotes a violation of the safety condition.

20, and 10, and an output layer for a one-dimensional output. Classification is defined using Equation 3.2, where $A_1 = (-\infty,25), A_2 = [25,53), A_3 = [53,58), A_4 = [58,\infty)$ *and* $\alpha_1 = 0, \alpha_2 = 50, \alpha_3 = 56, \alpha_4 = 60$.

To train the neural network, we randomly generate 2M samples where $x \sim \mathcal{U}(500,560)$, $c_1, c_2 \sim \mathcal{U}(0,60)$, *and* $m \sim P(0,3)$, *where* $P(x=0) = 0.48, P(x=1) = P(x=2) = 0.24$ *and* $P(x=3) = 0.04$. *We define the target outputs as specified by the hybrid automaton in Example 4.1.3. We set the train-validation split to 80/20 and train the model using the Adam optimizer with a learning rate of* $1e-3$ *and weight decay* $1e-5$ *for 140 epochs. The batch size is fixed as 32 and we employ the MSE loss function. The resulting trained network achieves an accuracy of 92.45%.*

Assuming the neural network is trained to keep the hybrid system behavior safe, we shall now proceed to employ it for safety verification. To this end, we need to define the input to the controller during the reachability analysis which is straightforward by Section 3.2.1.

Example 4.1.5 (Input star set for the rod reactor)**.** *Let* $x_l, x_u, c_{1_l}, c_{1_u}, c_{2_l}, c_{2_u} \in \mathbb{R}$ *be the lower and upper bounds of the temperature, the first and second clock, respectively, and* $m \in \{0,1,2,3\}$ *the mode the system is in at time point* t_k. *Then the input star set for the controller* $\Theta_I = \langle c, V, P \rangle$ *with* $V = \{e_1, e_2, e_3, e_4\}$ *and* $P(\alpha) \triangleq C\alpha \leq s$ *is defined as*

$$c = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad C = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad s = \begin{bmatrix} -x_l \\ x_u \\ -c_{1_l} \\ c_{1_u} \\ -c_{2_l} \\ c_{2_u} \\ -m_k \\ m_k \end{bmatrix}.$$

Now that all the components are defined, it remains to fix the hyperparameters. Following the rationale from Section 3.1.1, we find a controller invocation frequency of $\delta = 1.0$ to be an adequate trade-off between computational effort and safety. We

compare a selection of different step sizes in Figure 4.4 to underline our decision. Lastly, we apply Algorithm 7 to compute the reachable sets of the system and testify the system's safety.

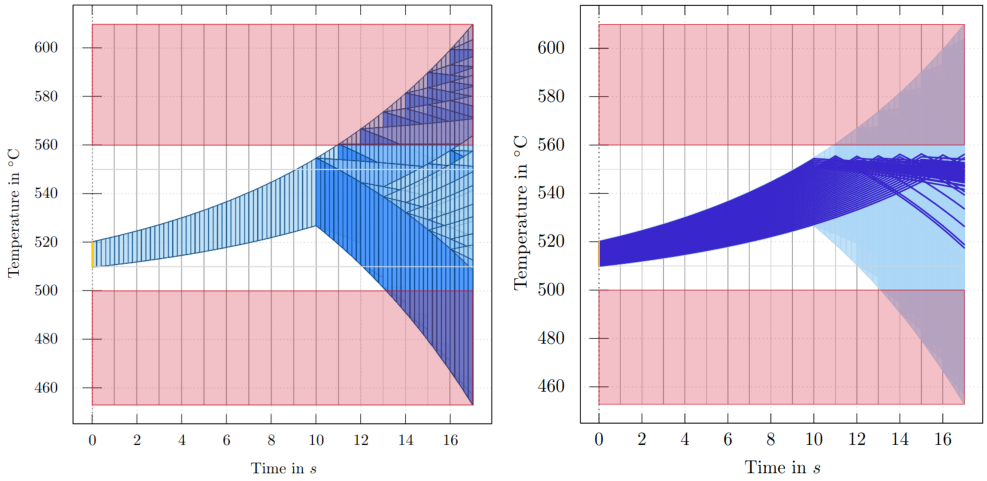
Example 4.1.6 (NNCS analysis on rod reactor)**.** *The resulting reachable states are plotted in Figure 4.5. Although the neural network controller is trained to behave safely, the NNCS reachability analysis alone is inconclusive to make statements about the system's safety. Clearly, the reachable states begin to diverge at the first time point when there is a non-deterministic choice to insert a rod at time 10 and quickly intersect the unsafe regions with* $x \geq 560$ *and* $x \leq 500$*. This observation can be attributed to the following points. The desired decision boundaries as indicated by the horizontal lines at 500 °C and 560 °C correspond to the guards of the original automaton. At each time of invocation of the neural network, one of the decision boundaries is contained in the temperature interval. By definition of our approach, the analysis is tasked to over-approximate all possible trajectories, hence at this point, the predicted control modes require a branching of the original flowpipe into multiple flowpipes with different flows. In this case, it is a branching into inserting the first, the second and no rod, respectively. Additionally, as aforementioned, by the defining temperature flows of each mode, once the unsafe region of the minimum or maximum critical temperature is entered, and the flow is too small or too large, the flowpipes will eventually diverge. For instance, after time step 13 once the flowpipe according to the temperature flow* $\dot{x} = 0.1x - 50$*, where no rod is inserted, reaches the unsafe region below 500 °C, the slope of the flowpipes only gets steeper with time the lower the temperature since* $0.1 * 500 - 50 \leq 0$*.*

Due to the diverging behavior, eventually both unsafe regions are reached. The result is thus inconclusive for the considered time horizon of 17 seconds as the over-approximation of reachable states intersect the bad states although the real trajectories as illustrated in Figure 4.5b model a safe behavior. In a real-time verification, it would thus prove beneficial to restrict the analysis of this system to a smaller time horizon.

4.2 Experimental Results

In this section, we will perform experiments to obtain quantitative results that complement our analysis. To quantify the error introduced by the over-approximation, we measure the surface area of the flowpipe segments and compare the results obtained with different state-set representations. Furthermore, we investigate the run-time of the analysis using different state-set representations. Additionally, we analyze the branching factor over the course of the analysis and the number of flowpipe segments generated during the analysis. All experiments are conducted using floating point numbers.

For this, we present the experimental setup for evaluating the proposed method. We consider the following benchmarks: the thermostat, which was introduced in Example 3.1.4, and the rod reactor, as introduced in Example 4.1.2. The Thermostat is bounded by a time horizon of $T = 2.1$ with an invocation frequency of 0.1, and a flowpipe-wise time step of 0.05. For the Rod Reactor, we set the time horizon to $T = 17$, local time horizons to 1.0, and flowpipe-wise time steps to 0.25.



(a) Reachable states colored by flow

(b) With trajectories (dark blue) overlay

Figure 4.5: Reachable states of the rod reactor example computed by the NNCS analysis for $T = 17$ using $\mathcal{V}$-polytope representation. The initial set with temperature $x \in [510{,}520]$ and flow $f_{1,x}$ at time 0 is indicated by an orange line.

4.2.1 Run-time

To measure the run-time performance, we execute the analysis 100 times using different state-set representations, namely box, $\mathcal{H}$-polytope, and $\mathcal{V}$-polytope. Unfortunately, due to numerical inaccuracies, it is not possible to deploy the analysis for the rod reactor benchmark using the $\mathcal{H}$-polytope representation, therefore in that case, we restrict the results to the other two state-set representations. We report the aggregated results in Table 4.1. Our results indicate that the box representation consistently yields the lowest run-time. Specifically, for the thermostat example, the box representation is 2.9 times faster than the $\mathcal{V}$-polytope representation and 5 times faster than the $\mathcal{H}$-polytope representation. The speedup is even more pronounced in the rod reactor example, where the speedup is by a factor of 13.4 for the $\mathcal{V}$ representation.

The median runtime values, as shown in the results, follow a similar pattern to that of the mean values, with the box representation consistently exhibiting the lowest median runtime and the $\mathcal{H}$-polytope representation manifesting the highest median runtime in the thermostat benchmark. Additionally, the minimum and maximum runtime values reveal that the $\mathcal{H}$-polytope representation has a narrower range of runtime values in comparison to the other two representations in both benchmarks. Moreover, the $\mathcal{H}$-polytope representation demonstrates the smallest standard deviation, implying that the runtimes are relatively consistent in contrast to the other representations, while the $\mathcal{V}$-polytope representation bears the highest standard deviation in both benchmarks, signifying a broader range of runtime values.

The significant differences in run-time between the $\mathcal{H}$- and $\mathcal{V}$-polytopes can be attributed to the fact that in the implementation of the analysis, operations are used where the $\mathcal{V}$-polytope requires less computational efforts than the $\mathcal{H}$-polytope. In Algorithm 7, several instances of such operations are employed. For example, creating the input star (as well as processing the predictions of the output stars of

Representation	Mean	Median	Min	Max	Std
Thermostat					
Box	880.73	864.33	657.49	3433.20	279.67
$\mathcal{H}$-polytope	4426.76	4418.52	3759.03	4980.86	186.12
$\mathcal{V}$-polytope	2544.66	2496.54	1650.08	5090.44	355.09
Rod reactor					
Box	6969.49	6953.31	4877.65	9638.46	474.57
$\mathcal{V}$-polytope	93445.35	91314.90	87657.10	116347.00	4646.71

Table 4.1: Run-time measures in milliseconds.

the neural network reachability analysis) require extracting interval bounds in each dimension of the considered state set, which necessitates an expensive conversion to a $\mathcal{V}$-polytope. The affine transformation of the initial set also involves a conversion. In HyPro, the conversion from $\mathcal{H}$- to $\mathcal{V}$-polytope requires vertex enumeration methods that entail solving $2d$ linear programs to obtain the interval bounds for each variable. Interested readers can find an overview of the computational demands of operations using convex polytopes in Chapter 6 of [Sch19].

4.2.2 Accuracy

As previously stated in Section 2.1.2, the reachability problem for general hybrid automata is undecidable [Hen00]. As a result, assessing the accuracy of our approach cannot be done by a direct comparison with the exact reachable set. Instead, alternative measures are necessary for an indirect evaluation of the accuracy. To this end, we propose quantifying the degree of over-approximation error through a comparison of the total volume of the flowpipe segments projected to a two-dimensional subspace with respect to various state-set representations. To assess the evolution over time, we further examine the area of the projections over the entire time horizon $T = K\delta$ and the associated growth factor. In this section, when we refer to the growth factor, we refer to a time-dependent factor $a_k \in \mathbb{R}$ with $k = 1, \ldots, K$, such that

$$a_k = \begin{cases} 1, & \text{if } k = 1 \\ \frac{b_k}{b_{k-1}}, & \text{if } k = 2, \ldots K \end{cases} \tag{4.1}$$

given an arbitrary sequence of measurements $b_1, \ldots, b_K \in \mathbb{R}$. In case of the area growth factor, we assume the measurements b_k to be the cumulative area until time t_k.

Computation of the area is straightforward due to two restrictions. Firstly, we consider convex polytopes exclusively, and secondly, to compute the surface area of the projected flowpipe segments, we only need to consider two-dimensional convex polytopes, or convex polyhedra. We refer to the Appendix B for the detailed computation of such areas.

We report the results in Table 4.2 and Figure 4.6. The table shows the results of computing the total area for different representations (Box, $\mathcal{H}$-polytope, and $\mathcal{V}$-polytope) over different dimensions for the two benchmarks and allows us to draw conclusions about the accuracy of the analysis with respect to the state-set representations. In both examples, the box representation consistently yields the largest

Projection	Box	$\mathcal{H}$-polytope	$\mathcal{V}$-polytope
Thermostat			
t, x	11.0967	10.4597	6.8272
Rod Reactor			
t, x	8226.88	-	8087.65
t, c_1	49.02	-	5.82292

Table 4.2: Total area of flowpipes computed over different dimension projections.

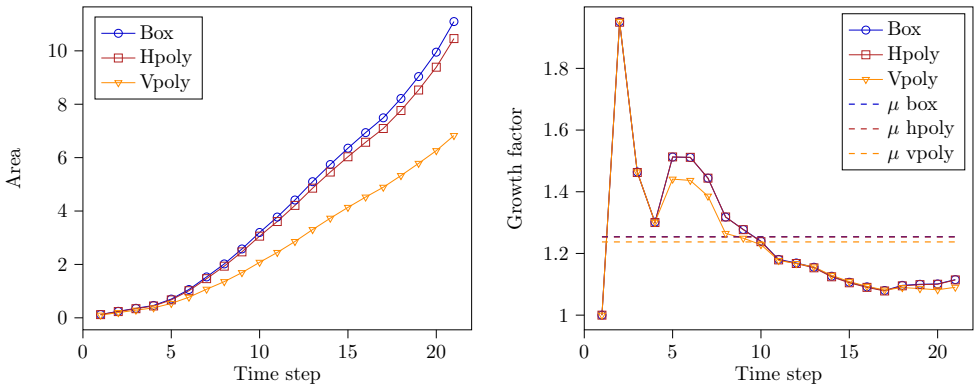
over-approximation. For the thermostat benchmark, the area of the box, reported as 11.0967, is 1.0609 times larger than of the $\mathcal{H}$-polytope with 10.4597 and even 1.625361 times larger than the $\mathcal{V}$-polytope with 6.82722. When considering the evolution of the areas over time, Figure 4.6a reveals that the projected flowpipes using box and $\mathcal{H}$-polytope representation are growing in size faster than $\mathcal{V}$-polytope. This observation is further supported when calculating the area growth factor over time as shown in Figure 4.6b which is smaller between time 4 and 11 for the $\mathcal{V}$-polytope, leading to a mean growth factor (as denoted by μ in the plots) of 1.236 instead of 1.254 for the other two representations.

Regarding the rod reactor benchmark, it involves four state variables, namely the temperature denoted by x, as well as the clocks c_1 and c_2 for the first and second rods, respectively, and the global time t. In order to compute the area over time, we project the four-dimensional space onto the plane spanned by the time axis and each of the state variables separately, resulting in three two-dimensional projections to be analyzed. Again, the boxes yield the most inaccurate over-approximation in all projections. In comparison to the $\mathcal{V}$-polytope, the boxes cover an area 1.01 or 8.42 times its area when considering the subspace of the temperature over time or the clocks over time, respectively. We omit the projection to the global time and the second clock as it is the same as for the first clock by design of the clock dynamics and the same initial conditions. Interestingly, the cumulative area over time in Figure 4.6d for the box and the $\mathcal{V}$-polytope representation are hardly discernible, causing the growth factors of the flowpipe area over time to be virtually identical as plotted in Figure 4.6e, unlike the thermostat benchmark. The resulting mean growth factors are thus 1.4994 and 1.4981 for the box $\mathcal{V}$-polytope, respectively.

The findings presented herein provide compelling evidence for a discernible trade-off between computational efficiency and precision, with the box representation being the least computationally demanding but also the most imprecise. Additionally, the results suggest that utilizing $\mathcal{H}$-polytopes in the analysis may not be beneficial given their excessive run-time and negligible improvement in over-approximation error compared to the box representation. In contrast, despite incurring a significant computational cost, the employment of $\mathcal{V}$-polytopes may be justified by their superior accuracy in approximating the true reachable set.

4.2.3 Complexity

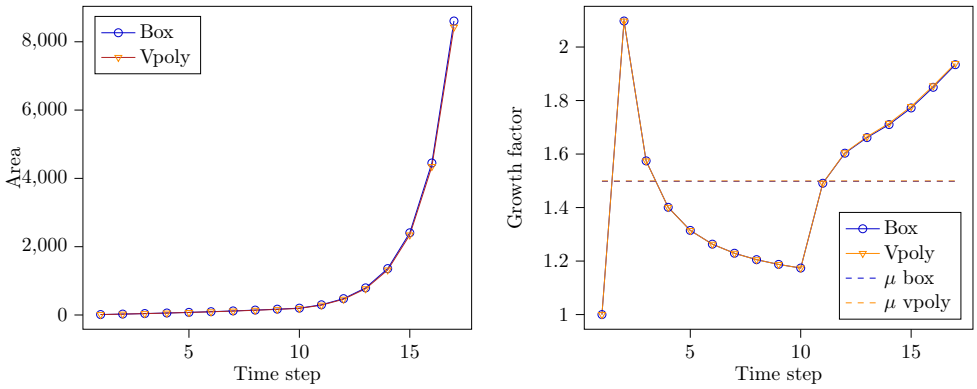
Due to the intricacy of Algorithm 7, a thorough formal analysis of its time and space complexity is beyond the scope of this thesis. However, we propose an alternative approach to gain insights into its theoretical complexity by presenting quantitative



(a) Cumulative area

(b) Area growth factor

(c) Thermostat



(d) Cumulative area

(e) Area growth factor

(f) Rod reactor

Figure 4.6: Area projected to global time and temperature over time (first column), and the growth factors (second column) for each representation (box, $\mathcal{H}$-polytope, $\mathcal{V}$-polytope), respectively. The dashed lines in the combined plot in the second column denote the mean growth factors.

results. We measure the number of branches and the number of flowpipes at each time step, which indirectly allows us to evaluate the complexity of the system being analyzed. Specifically, a larger number of branches may indicate a more complex system with numerous intersections and discontinuities in the dynamics, while a smaller number of branches may indicate a simpler system with more regular dynamics that is easier to analyze. By quantifying the number of branches at each time step, we can indirectly assess the theoretical complexity of the analysis and gain insights into the potential computational cost and difficulty of verifying the system.

We present the results plotted over time in Figure 4.7, where for each benchmarks, we show the number of branches, the number of flowpipes, and their respective growth factors according to Equation 4.1. The thermostat benchmark exhibits a steady increase in the number of branches with short plateaus until at most 12 branches at

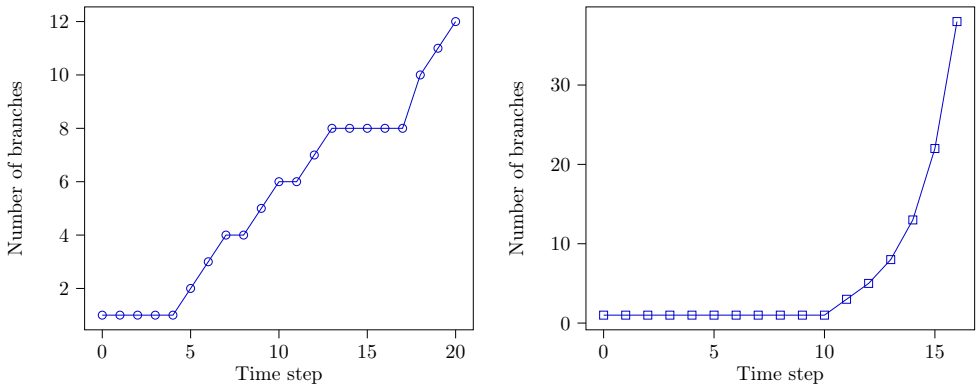
time step 20, whereas the rod reactor example has only one branch for the first ten time steps, after which the number increases heavily at each time point, from 3 to 38 branches in six time steps until the final time step 16. The number of flowpipes in both benchmarks steadily increases, with the thermostat having 129 flowpipes and the rod reactor having 169 flowpipes by the time the global time horizon is reached. However, the slope only begins to rise more strongly after a while for the rod reactor, and it is more extreme than for the thermostat.

For each time step t_k, we further compute the branching factor b_k according to Equation 4.1 over the number of branches. Comparing both benchmarks, the thermostat has a mean branching factor of 1.16 while the rod reactor has a mean of 1.35, denoted by the dashed horizontal lines. The general tendency for the thermostat seems to be a decrease in the branching factor over time, whereas it is the opposite for the rod reactor, where there appears to be an increase in the long run.

Analogously, we compute the growth factor f_k of the number of flowpipes at time t_k and obtain a mean of 1.252 for the thermostat and a mean of 1.397 for the rod reactor, highlighted by the dashed lines. Interestingly, the growth factor of the number of flowpipes in each benchmark appears to follow the evolution of the branching factor, with the thermostat flowpipe growth factor decreasing over time and the rod reactor's increasing.

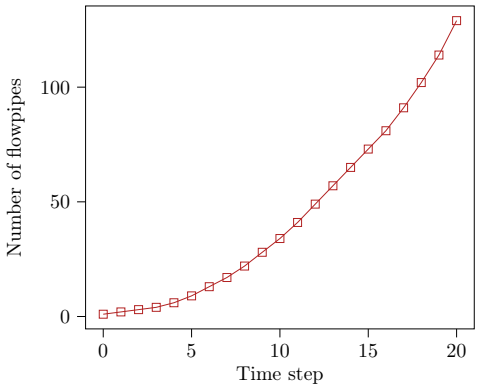
The findings of this study indicate that the complexity of both benchmarks increases with time, as evidenced by the continuous rise in the number of branches and flowpipes. Nevertheless, the rod reactor benchmark displays a more pronounced growth in these quantities compared to the thermostat. In addition, the branching factor and growth factor analyses provide further insights into the complexity of the benchmarks. The mean values of both factors are greater for the rod reactor compared to the thermostat, suggesting that the former is more complex in terms of the number of branches and flowpipes generated over time which indeed aligns with the comparative larger size of the rod reactor benchmark.

While there are no established references for interpreting these numbers, the direct comparison of both benchmarks allows us to infer that the factors are indicative of the complexity of the respective systems. It is worth noting, however, that this metric has limitations as it depends on the hyperparameters set for the benchmarks, such as the global time horizon or step size. Therefore, it is challenging to generalize these findings to unseen benchmarks and make them comparable. Nevertheless, we still believe that the evolution of the number of branches and flowpipes over time is an insightful metric to consider.

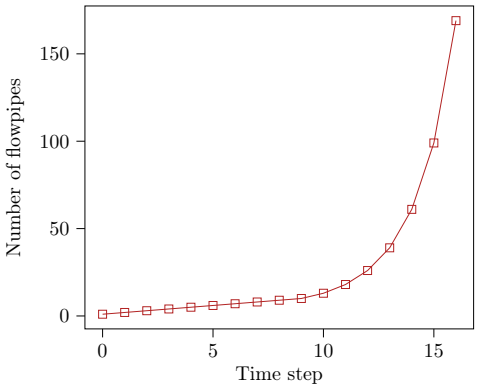


(a) Thermostat, branching

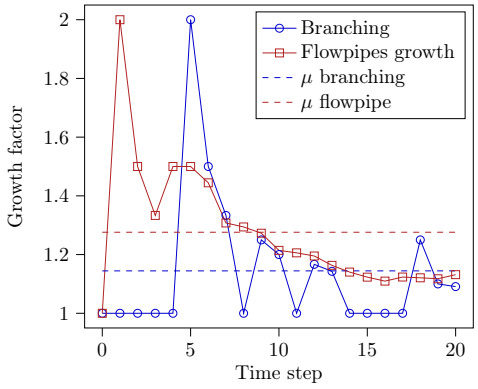
(b) Rod reactor, branching



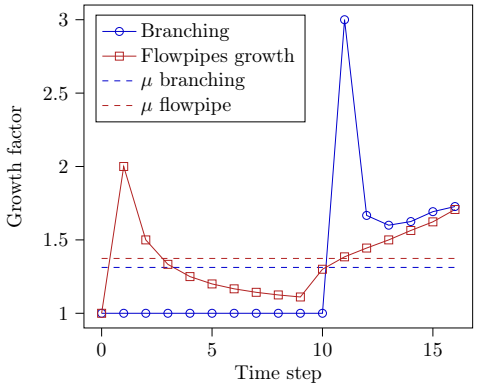
(c) Thermostat, flowpipes



(d) Rod reactor, flowpipes



(e) Thermostat, growth factors



(f) Rod reactor, growth factors

Figure 4.7: Branching (first row), cumulative number of flowpipes (second row) and growth factor of number of flowpipes and branches (third row) over time for the thermostat (left) and rod reactor (right) benchmarks. The dashed lines in the combined plot in the third row denote the mean growth factors.

Chapter 5

Conclusion

In the last chapter, we provide a critical analysis of the results obtained in the previous chapters and discuss their implications, limitations, and potential for future research.

5.1 Discussion

In this work, we developed an algorithm for NNCS safety verification of a hybrid system modeled by a feedback loop between a neural network controller and a simplified hybrid automaton, represented as a set of linear ODEs defining the plant dynamics, which enabled leveraging the established reachability analysis method of flowpipe construction in conjunction with an exact star-based neural network reachability analysis. This not only keeps the over-approximation error minimal but also allows for efficient computation of reachable sets.

The approach is supported by a comprehensive analysis of the possible design choices that led to the final solution and is further shown to be sound for verifying the safety of neural network controlled hybrid systems. Our approach keeps the over-approximation error minimal for the following reasons: first, the controller's reachability analysis is exact and thus does not introduce any additional errors, and second, since we are only propagating the original initial set, we are mitigating accumulated wrapping effects. We assess the algorithm's effectiveness by evaluating it on two benchmarks from related literature and provide quantitative results that offer insights into the method's run-time, accuracy and complexity. The implementation with HyPro [SÁMK17] allows for easy integration of further benchmarks and extension to additional capabilities.

Yet, the current method is limited to fully connected feed-forward neural networks with piece-wise linear activation functions only. Further, since all potential transient trajectories generated from a system are considered with equal confidence, the resulting analysis may return diverging states as seen in the rod reactor benchmark. Motivated by the need to evaluate our approach on some benchmarks and limited by time constraints, we adopted the controller behavior from existing hybrid automata, however, it would be more sensible for the neural network to learn the optimal controls in an unsupervised manner. These restrictions thereby give rise to the following improvements future work may undertake.

5.2 Future work

Currently, the controller in this work only makes definite decisions about letting the plant construct flowpipes according to the predicted mode in the next step. This behavior results in the propagation of the original initial set in its full size for a duration of δ. However, a more nuanced decision-making process could be developed to reduce the over-approximation error and the wrapping effects by incorporating a qualitative measure that considers the controller's confidence, given that it models a safe system. Additionally, backward analysis could be applied from the reachable unsafe states to refine the over-approximation by constructing flowpipes only for the relevant trajectories that lead to unsafe states, which would be a more efficient approach than constructing flowpipes for all potential trajectories.

In addition, the current output model of the controller is constrained by the assumption that the flows of the underlying hybrid automaton can only be distinguished by a constant value that the controller is trained to predict. This definition needs to be expanded to increase the applicability of this approach. For example, dealing with plants with nonlinear dynamics would enable comparisons with existing techniques [IWA+19, ICW+21] that utilize Taylor models as a representation and implement them using FlowStar* [CÁS13]. Moreover, it would be worthwhile to investigate the feasibility of the controller operating as a PID controller in specific cases, with the controller output representing a control input from a finite but uncountable set rather than a single discretized control.

To improve the practicality of the analysis for real-world applications, it may be beneficial to apply a reinforcement learning approach to find the optimal control given a specific state rather than artificially generating controls, as this work did. These extensions and improvements could enhance the capabilities and usefulness of the proposed algorithm.

Further metrics could be developed to enhance the comparability of the algorithm with existing tools. Moreover, the theoretical time and space complexity could be analyzed to provide a better understanding of the algorithm's performance.

The rod reactor benchmark posed numerical challenges that prevented the analysis from being conducted using the $\mathcal{H}$-polytope representation. This issue highlights the need for future work to improve the existing methodology. Furthermore, incorporating alternative state set representations, such as zonotopes and support functions, may be explored in future research, each with its benefits and drawbacks.

Bibliography

[Á21] Erika Ábrahám. Lecture notes in modeling and analysis of hybrid systems, May 2021.

[ACH+95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A Henzinger, P-H Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical computer science*, 138(1):3–34, 1995.

[BD17] Stanley Bak and Parasara Sridhar Duggirala. Simulation-equivalent reachability of large linear systems with inputs. In *International Conference on Computer Aided Verification*, pages 401–420. Springer, 2017.

[Bra86] Bart Braden. The surveyor's area formula. *The College Mathematics Journal*, 17(4):326–337, 1986.

[CÁS13] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25*, pages 258–263. Springer, 2013.

[FHC+20] Jiameng Fan, Chao Huang, Xin Chen, Wenchao Li, and Qi Zhu. Reachnn*: A tool for reachability analysis of neural-network controlled systems. In *Automated Technology for Verification and Analysis: 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19–23, 2020, Proceedings*, pages 537–542. Springer, 2020.

[FLGD+11] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings 23*, pages 379–395. Springer, 2011.

[Gir05] Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *International Workshop on Hybrid Systems: Computation and Control*, pages 291–305. Springer, 2005.

[Hen00] Thomas A Henzinger. The theory of hybrid automata. In *Verification of digital and hybrid systems*, pages 265–292. Springer, 2000.

[HFC+21] Chao Huang, Jiameng Fan, Xin Chen, Wenchao Li, and Qi Zhu. Polar: A polynomial arithmetic framework for verifying neural-network controlled systems. *arXiv:2106.13867*, 2021.

[HFL+19] Chao Huang, Jiameng Fan, Wenchao Li, Xin Chen, and Qi Zhu. Reachnn: Reachability analysis of neural-network controlled systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–22, 2019.

[HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, 1998.

[ICW+21] Radoslav Ivanov, Taylor Carpenter, James Weimer, Rajeev Alur, George Pappas, and Insup Lee. Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I*, pages 249–262. Springer, 2021.

[IWA+19] Radoslav Ivanov, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 169–178, 2019.

[JLB+16] Kyle D Julian, Jessica Lopez, Jeffrey S Brush, Michael P Owen, and Mykel J Kochenderfer. Policy compression for aircraft collision avoidance systems. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–10. IEEE, 2016.

[KBD+17] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International conference on computer aided verification*, pages 97–117. Springer, 2017.

[LG09] Colas Le Guernic. *Reachability analysis of hybrid systems with linear continuous dynamics*. PhD thesis, Université Joseph-Fourier-Grenoble I, 2009.

[LTS99] John Lygeros, Claire Tomlin, and Shankar Sastry. Controllers for reachability specifications for hybrid systems. *Automatica*, 35(3):349–370, 1999.

[SÁMK17] Stefan Schupp, Erika Ábrahám, Ibtissem Ben Makhlouf, and Stefan Kowalewski. Hypro: A C++ library of state set representations for hybrid systems reachability analysis. In *NASA Formal Methods Symposium*, pages 288–294. Springer, 2017.

[Sch19] Stefan Schupp. *State Set Representations and Their Usage in the Reachability Analysis of Hybrid Systems*. PhD thesis, RWTH Aachen University, Aachen, 2019.

[SGM+18] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. *Advances in neural information processing systems*, 31, 2018.

[SGPV19] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019.

[SZS+14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[TML+19] Hoang-Dung Tran, Patrick Musau, Diego Manzanas Lopez, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T Johnson. Parallelizable reachability analysis algorithms for feed-forward neural networks. In *2019 IEEE/ACM 7th International Conference on Formal Methods in Software Engineering (FormaliSE)*, pages 51–60. IEEE, 2019.

[Tra20] Dung Hoang Tran. *Verification of Learning-Enabled Cyber-Physical Systems*. PhD thesis, Vanderbilt University, 2020.

[WL19] Song Wang and Zhixia Li. Exploring the mechanism of crashes with automated vehicles using statistical modeling approaches. *PloS one*, 14(3):e0214550, 2019.

[XTJ18] Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. Output reachable set estimation and verification for multilayer neural networks. *IEEE transactions on neural networks and learning systems*, 29(11):5777–5783, 2018.

[XTRJ18] Weiming Xiang, Hoang-Dung Tran, Joel A Rosenfeld, and Taylor T Johnson. Reachable set estimation and safety verification for piecewise linear systems with neural network controllers. In *2018 Annual American Control Conference (ACC)*, pages 1574–1579. IEEE, 2018.

[XTYJ20] Weiming Xiang, Hoang-Dung Tran, Xiaodong Yang, and Taylor T Johnson. Reachable set estimation for neural network control systems: A simulation-guided approach. *IEEE Transactions on Neural Networks and Learning Systems*, 32(5):1821–1830, 2020.

Appendix A

Supplementary Proofs

In this supplementary proofs section, we present a collection of proofs that were omitted from the preliminary section of the thesis. These proofs provide additional support for the main theorems and lemmas discussed in the thesis. The supplementary proofs section is intended to be read in conjunction with the preliminary section and the main body of the thesis.

Proposition A.0.1 (2.2.1)**.** *Any bounded convex polyhedron* $\mathcal{P} = \left\{x \mid Cx \leq s, x \in \mathbb{R}^d\right\}$ *can be represented as a star.*

Proof. The polyhedron can be encoded by the star set Θ with the center $c = (0,0,\cdots,0)^T$, the basis vectors $V = \{e_1, e_2, \cdots, e_d\}$ in which e_i is the i-th unit vector of $\mathbb{R}^d$, and the predicate $P(\alpha) \triangleq C\alpha \leq s$. □

Proposition A.0.2 (2.2.2)**.** *Given a star set* $\Theta = \langle c, V, P\rangle$*, an affine mapping of the star* Θ *with the affine mapping matrix* W *and offset vector* b *defined by* $\bar{\Theta} = \{y \mid y = Wx + b, x \in \Theta\}$ *is another star such that*

$$\bar{\Theta} = \langle \bar{c}, \bar{V}, \bar{P}\rangle, \quad \bar{c} = Wc + b, \quad \bar{V} = \{Wv_1, Wv_2, \ldots, Wv_m\}, \quad \bar{P} \equiv P.$$

Proof. Based on the definition of a star, we can infer that can be expressed as □

Lemma A.0.3 (2.2.4)**.** *The worst-case complexity of the number of stars in the reachable set of an FNN with* N *neurons is* $\mathcal{O}(2^N)$*.*

Proof. To prove the worst-case complexity of the number of stars in the reachable set of an FNN, we begin with a star input set. Each *stepReLU* operation applied to the input set results in at most two more stars. Therefore, the total number of stars in the worst-case scenario of one layer is 2^{n_L}, where n_L is the number of neurons in the layer. Since the output reachable sets of one layer are the inputs of the next layer, the total number of stars in the reachable set of an FNN with k layers and N neurons, in the worst-case scenario, is $2^{n_{L_1}} \cdot \cdots \cdot 2^{n_{L_k}} = 2^{n_{L_1}+\cdots+n_{L_k}} = 2^N$. Hence, the worst-case complexity of the number of stars in the reachable set of an FNN is $O(2^N)$. □

Lemma A.0.4 (2.2.5)**.** *The worst-case complexity of the number of constraints of a star in the reachable set of an FNN with* N *neurons is* $\mathcal{O}(N)$*.*

Proof. The stepReLU sub-procedure produces one or two stars that have at most one more constraint than the star input set Θ. Hence, for a layer of n neurons, at most n stepReLU operations are performed, leading to star reachable sets that contain at most n more constraints than the input set. Therefore, the number of constraints in a star input set grows linearly over layers, resulting in a worst-case complexity of $O(N)$ for the number of constraints in a star in the reachable set of an N-neuron FNN. □

Theorem A.0.5 (2.2.7)**.** *Let $\mathcal{N}$ be an FNN, $\Theta = \langle c, V, P\rangle$ be a star input set, $\mathcal{N}(\Theta) = \bigcup_{i=1}^{k} \Theta_i$, $\Theta_i = \langle c_i, V_i, P_i\rangle$ be the reachable set of the neural network, and $\mathcal{S}$ be safety specification. Denote $\bar{\Theta}_i = \Theta_i \cap \neg S = \langle c_i, V_i, \bar{P}_i\rangle$, $i = 1, \ldots, k$. The neural network is safe iff $\bar{P} = \emptyset$ for all i.*

Proof. The exact reachable set is a union of stars. It is trivial that the neural network is safe if and only if all stars in the reachable set do not intersect with the unsafe region, i.e., $\bar{\Theta}_i$ is an empty set for all i, or equivalently, the predicate $\bar{P}_i$ is empty for all i. □

Appendix B

Convex Polygon Area Computation

Let $p_1, p_2, \ldots, p_n \in \mathbb{R}^2$ with $p_i = (x_i, y_i)^T$ for $i = 1, \ldots, n, n > 1$ denote the unordered coordinates of a convex polygon. To compute its area, we first want to sort them in a clockwise or anti-clockwise order. Without loss of generality, assume we want to order them in counter-clockwise order. For this, we require an arbitrary reference point that lies inside the polygon. Take

$$p^* = \frac{1}{n} \sum_{i=1}^{n} p_i,$$

which is simply the averaged point of all coordinates.

After establishing a reference point by averaging, which is applicable only for the convex polygon, the vertices can be sorted by the angle formed by a line segment connecting the reference point and each vertex in a counterclockwise direction with respect to the x-axis.

The angle between the line segment connecting two points $p_1, p_2 \in \mathbb{R}^2$ with the positive x-axis in the anti-clockwise direction is computed by

$$\theta(p_1, p_2) = \begin{cases} 2\pi - \arcsin(k) & \text{if } k \geq 0 \text{ and } x_2 \geq x_1 \quad \text{(first quadrant)} \\ \pi + \arcsin(k) & \text{if } k \geq 0 \text{ and } x_2 < x_1 \quad \text{(second quadrant)} \\ \pi - \arcsin(-k) & \text{if } k < 0 \text{ and } x_2 < x_1 \quad \text{(third quadrant)} \\ \arcsin(-k) & \text{if } k < 0 \text{ and } x_2 \geq x_1 \quad \text{(fourth quadrant)} \end{cases} \tag{B.1}$$

where k is the slope of the line connecting p_1 and p_2, calculated as

$$k = (y_2 - y_1)/d(p_1, p_2).$$

The distance function $d(\cdot)$ is assumed to be defined as the Euclidean distance between two points in the Cartesian coordinate system.

Lastly, the points $p_i = (x_i, y_i)^T$ for $i = 1, \ldots, n$ are sorted by their angle $\theta_i(p^*, p_i)$ and the shoelace formula [Bra86]

$$A = \frac{1}{2} \left| \sum_{i=1}^{n-1} x_i y_{i+1} + x_n y_1 - \sum_{i=1}^{n-1} x_{i+1} y_i - x_1 y_n \right| \tag{B.2}$$

returns the area of the convex polygon.