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

BACHELOR OF SCIENCE THESIS

A NOVEL REDUCTION METHOD FOR STAR SETS AND ITS APPLICATION IN NEURAL NETWORK VERIFICATION

Vahe Vkhkryan

Communicated by
Prof. Dr. Erika Ábrahám

Examiners:
Prof. Dr. Erika Ábrahám
Prof. Dr. Thomas Noll

Additional Advisor:
László Antal

Aachen, 07.03.2024

Abstract

The star set representation method serves as a powerful technique for working with state sets, demonstrating remarkable efficiency in computing system reachability. This study represents a significant advancement in the field by aiming to streamline star set representation through the elimination of redundancies. Drawing upon principles from linear constraint systems theory, including variable elimination (quantifier elimination) techniques, we develop and implement a novel dimension reduction approach.

We apply our method to compute the reachable set of feedforward neural networks (FNNs). Our dimension reduction approach simplifies the representation of state sets by eliminating redundancies and retaining only essential information. This simplification enhances efficiency and interpretability, facilitating a more effective exploration of the network's behavior and properties.

We evaluated the effectiveness of our method on three benchmarks, including two realistic scenarios involving ACAS Xu and drone networks, as well as a smaller benchmark - thermostat network.

Keywords: Dimension reduction, star sets, Fourier-Motzkin variable elimination, FMplex, star-based reachability, over-approximate analysis, neural networks, safety and robustness verification.

Contents

Definitions

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

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

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 and vice versa.

Definition 0.0.2 (Null vector)**.** *A vector is called a null vector (0-vector) if all entries of the vector are 0.*

$$\mathbf{x} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

Definition 0.0.3 (Unit vector)**.** *A vector $\mathbf{e_i}$ is called the $i-$th identity vector in $\mathbb{R}^m$, if it is a null vector, with a 1 at the $i-$th row.*

$$\mathbf{e_i} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

Definition 0.0.4 (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:*

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

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 $\mathcal{A}$ is referenced by $a_{i,j}$, while we use $\mathcal{A}_{i,-}$ to reference the i-th row and similarly $\mathcal{A}_{-,j}$ to refer to the j-th column of $\mathcal{A}$.

Definition 0.0.5 (0-column matrix)**.** *A matrix $\mathcal{A} \in \mathbb{R}^{n \times m}$ is called 0-column if for at least one column $\mathcal{A}_{-,j}$ holds: $\forall i \;\; 1 \leq i \leq n; \;\; a_{i,j} = 0$:*

$$\mathcal{A} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,j-1} & 0 & a_{1,j+1} & \cdots & a_{1,m} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,j-1} & 0 & a_{n,j+1} & \cdots & a_{n,m} \end{bmatrix}$$

Definition 0.0.6 (Identity matrix)**.** *We define the identity matrix $\mathcal{I}_m \in \mathbb{R}^{m\times m}$ of size m as:*

$$\mathcal{I}_m = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}.$$

The $i-$th column in the identity matrix corresponds to the $i-$th unit vector.

Definition 0.0.7 (Linear dependence)**.** *Linear dependence is a concept that describes the relationship between vectors. In a vector space, a set of vectors is said to be linearly dependent if one or more of the vectors in the set can be written as a linear combination of the others.*
Formally, let $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_n}$ be vectors in a vector space V. These vectors are linearly dependent if there exist real valued scalars (coefficients) $c_1, c_2, \ldots, c_n$, at least one non-zero, such that:

$$c_1\mathbf{v_1} + c_2\mathbf{v_2} + \ldots, +c_n\mathbf{v_n} = 0$$

If no such non-zero coefficients exist, then the vectors are called linearly independent.

Definition 0.0.8 (Rank of a matrix)**.** *The rank of a matrix is a fundamental concept that measures the maximum number of linearly independent columns (or rows) in the matrix. It provides information about the dimension of the column space (or equivalently, the row space) of the matrix.*
For a matrix $\mathcal{A} \in \mathbb{R}^{n\times m}$, where n is the number of rows and m is the number columns, the rank of $\mathcal{A}$ (denoted rank($\mathcal{A}$)) is defined as the maximum number of linearly independent columns (or rows) in $\mathcal{A}$.

Definition 0.0.9 (Kernel or null space of a matrix)**.** *For a matrix $\mathcal{A} \in \mathbb{R}^{n\times m}$ the kernel or null space, denoted as ker($\mathcal{A}$) or Null($\mathcal{A}$), is the set of all vectors $\mathbf{x}$ such that $\mathcal{A}\mathbf{x} = \mathbf{0}$, where $\mathbf{0}$ is the zero vector.*

$$Null(\mathcal{A}) = ker(\mathcal{A}) = \{\mathbf{x} \mid \mathcal{A}\mathbf{x} = \mathbf{0}\}$$

In this thesis, when saying kernel of a matrix, we will refer to the basis of the kernel.

Definition 0.0.10 (Image or column space of a matrix)**.** *The image of a matrix, also known as the column space, refers to the set of all possible linear combinations of the columns of the matrix. More formally, for a matrix $\mathcal{A} \in \mathbb{R}^{n\times m}$, the image of $\mathcal{A}$, denoted by Im($\mathcal{A}$), is the subspace of $\mathbb{R}^m$ spanned by the columns of $\mathcal{A}$.*
Mathematically, the image of $\mathcal{A}$ is defined as:

$$Im(\mathcal{A}) = \{\mathbf{y} \in \mathbb{R}^m : \mathbf{y} = \mathcal{A}\mathbf{x} \text{ for some } \mathbf{x} \in \mathbb{R}^n\}$$

We define the basis for Im($\mathcal{A}$) as a set of vectors $\{\mathbf{v_1}, \mathbf{v_2}, \cdots, \mathbf{v_r}\}$ such that:

1. *Each vector in the set is in the image space of $\mathcal{A}$, i.e., $\mathbf{v}_i \in Im(\mathcal{A})$.*

2. *The set of vectors is linearly independent.*
3. *The set of vectors spans the image space, meaning that any vector in the image space can be expressed as a linear combination of the vectors in the basis.*

Mathematically, if $\{v_1, v_2, \ldots, v_r\}$ *form a basis for* $Im(\mathcal{A})$*, then any vector* $\mathbf{y}$ *in the image space can be expressed as:*

$$\mathbf{y} = c_1\mathbf{v_1} + c_2\mathbf{v_2} + \ldots + c_r\mathbf{v_r}$$

where $c_1, c_2, \ldots, c_r$ *are scalars.*

In this thesis, when saying image of a matrix, we will refer to the basis of the image.

Definition 0.0.11 (Nullity of a matrix)**.** *For a matrix* $\mathcal{A} \in \mathbb{R}^{n\times m}$*, the nullity (denoted as* $nullity(\mathcal{A})$*) is defined as the dimension (the number of independent vectors) of its null space.*

$$nullity(\mathcal{A}) = dim(Null(\mathcal{A}))$$

The nullity of a matrix is related to its rank by the Rank-Nullity Theorem, which states that the sum of the rank and nullity of a matrix is equal to the number of columns of the matrix.

$$rank(\mathcal{A}) + nullity(\mathcal{A}) = m$$

Chapter 1

Introduction

Artificial neural networks (ANNs) [HC18] have surged in popularity over the past few decades, finding applications in diverse fields such as gaming [SHM+16], healthcare [DKML19, KTS+18], autonomous vehicles [BDTD+16], and many more. Among the various types of ANNs are feedforward neural networks (FNNs) [SKP97, Saz06], convolutional neural networks [LBBH98], recurrent neural networks [MJ01], and graph neural networks [ZCH+20], each tailored for specific tasks ranging from pattern recognition, image and video processing to social network analysis.

Despite their widespread adoption, ANNs are often regarded as "black box" mechanisms, wherein inputs yield outputs without transparent intermediate processes [LJB+22]. Consequently, questions regarding the reliability and trustworthiness of neural networks arise, particularly in safety-critical contexts. Can we depend on neural networks to produce accurate outputs consistently, even in unfamiliar scenarios? Can we ensure that their behavior adheres to desired properties, such as robustness, fairness, and safety?

Formal verification offers means to address these concerns by providing mathematical assurances regarding a model's adherence to specified properties. In this thesis, we focus on the formal verification of FNNs via reachability analysis using a specific state-set representation. The state set that we use is called the star set, first introduced in [DV16] and generalized in [BD17].

Tran et al. proposed in [TMLM+19] two ways of computing the reachable set of neural networks: an exact method ensuring soundness and completeness, and an over-approximate method, which while not complete, remains sound and is more scalable to bigger network architectures. Our emphasis lies on the over-approximate analysis, intending to develop a practical and efficient method for simplifying the representation of star sets. To achieve this, we leverage established techniques from theory, specifically focusing on Fourier-Motzkin variable elimination [Dan63, CD07, KL91] and its recent extension FMplex.

The objective of this thesis is to simplify the representation of star sets while preserving all essential information. We propose a novel method for reducing the dimensions of star sets, which has been integrated as an extension of the open-source C++ library HyPro [SÁMK17], facilitating the evaluation of our method's efficiency. Our approach leverages advanced mathematical and computational tools such as the Eigen library for linear algebra operation [GJ+10], GLPK used for the linear optimizations [GAD+13]. Exact calculations are provided by the GNU MP library's

mpq_class rational data type [Gra96].

1.1 Thesis Overview

In the Definitions Chapter, we provide essential definitions, notations, and theorems related to our thesis. Section 2.2 introduces the star set representation, while Section 2.3 elucidates the application of Fourier-Motzkin variable elimination to linear inequality systems. In Chapter 3, we present the central contribution of this thesis, presenting the reduction of star sets. Section 3.2 illustrates the application of FM elimination to star sets, while Section 3.3 establishes precondition for star set reduction and verifies the correctness of our method.

Sections 4.1-4.3 offer a concise overview of neural networks and their reachability analysis, culminating in Section 4.4, where we demonstrate the application of our dimension reduction method tailored to reachability analysis of FNNs. Chapter 5 evaluates our method on three benchmarks regarding the running time, the number of returned constraints, and the number of redundant constraints, while Chapter 6 discusses its performance, limitations, and avenues for future improvement.

1.2 Related Work

Within the domain of formal verification, Xiang et al. (2018) discuss various techniques for the verification of neural networks, providing valuable insights into the challenges and approaches within this field [XTJ18]. Tran et al. (2019) propose methods for computing the reachable set of neural networks, offering insights into ensuring soundness and completeness with exact and scalability with over-approximate analysis [TMLM$^+$19].

The study by Guennebaud and Jacob (2010) introduces Eigen, a C++ library for linear algebra operations, which serves as a foundational component in our implementation [GJ$^+$10]. Furthermore, Nalbach et al. (2023) present FMplex, a variant of Fourier-Motzkin variable elimination, offering potential enhancements for our dimension reduction method [NPÁK23]. Lastly, HyPro, a tool designed for analyzing hybrid systems and extended to also calculate neural network's reachability, which We employ in evaluating the efficiency of our method [SÁMK17].

Yang et al. compare four techniques for zonotope order reduction (same as dimension reduction) [Com03, Gir05, ASB10, SRMB16] aiming to achieve a similar result for zonotopes as our method does for star sets [YWLG17]. Sadraddini et al. proposed another method for zonotope order reduction in [ST19]. The key difference between these methods compared to our approach is that they all try to over-approximate the original zonotope Z^{orig} and get a zonotope Z^{red} described by less generator, such that $Z^{red} \supseteq Z^{orig}$ holds. Raghuraman et al. discuss the problem of possible dimension reduction while removing the redundant generators (with desired numerical precision) from a zonotope representation [RK22].

Chapter 2

Preliminaries

2.1 Set representations

A set, a fundamental concept in mathematics, represents a collection of distinct objects, treated as an entity in itself. These objects, ranging from numbers and symbols to elements of other mathematical structures, are referred to as the members or elements of the set. Sets provide a powerful and abstract framework for organizing and analyzing mathematical ideas and relationships. This thesis focuses on state sets and their representation. A comprehensive overview of the existing state set representation methods is presented in [Sch19]. Below, we will give a short introduction to some of them.

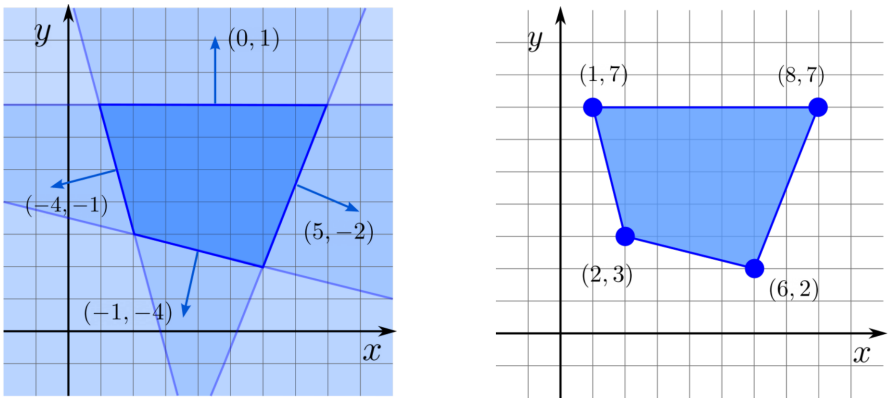
Various methods exist for representing state sets, often employed in geometric or computational contexts. Some of them are $\mathcal{H}$-polytopes (half-space polytopes), $\mathcal{V}$-polytopes (vertex polytopes), star sets, etc.

An $\mathcal{H}$-polytope is represented as the intersection of half-spaces (Figure 2.1a). A half-space represents the region on one side of a hyperplane. For instance, in two dimensions, the $\mathcal{H}$-polytope defined by $x \geq 0$ and $y \geq 0$ corresponds to the first quadrant. $\mathcal{H}$-polytopes offer a natural representation for the intersection of half-spaces, making them suitable for expressing linear constraints in optimization problems. Many efficient algorithms exist for working with $\mathcal{H}$-polytopes, especially in the context of linear programming. However, managing and visualizing $\mathcal{H}$-polytopes becomes increasingly challenging with higher dimensions.

An $\mathcal{V}$-polytope is defined by its vertices (corner points) (Figure 2.1b), constituting the convex hull of a finite set of points. Intuitively, the convex hull of points is the shape formed by the outermost boundary points of the given set, resulting in a convex polygon or polyhedron. However, determining and representing all vertices can be computationally demanding, particularly in high-dimensional spaces.

2.2 Star set representation

This work presents a state set representation called (generalized) star sets, sometimes called AH-polytopes [TMLM$^+$19]. Star sets offer precise and efficient computation with convex sets. They are very efficient in affine mapping operations (Proposition 2.2.2) and intersections with half-spaces (Proposition 2.2.3), which we are going to



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

Figure 2.1: Different representations of the same convex set (redrawn from [Jia23])

use later in Chapter 4 of this thesis.

In the following, we will present the formal definition of a star set and show some useful properties of it. This section is mainly based on the work of Hana Masara [Mas23].

Definition 2.2.1 (Generalized star set)**.** *A generalized (n, m)-dimensional star set Θ is a tuple $\langle \mathbf{c}, \mathcal{V}, P \rangle$ where $\mathbf{c} = (c_1, c_2, \ldots, c_n)^T \in \mathbb{R}^n$ is the center, $\mathcal{V} = (\mathbf{v^1}, \mathbf{v^2}, \ldots, \mathbf{v^m}) \in \mathbb{R}^{n \times m}$ is the generator (basis) matrix, $\forall i \ 1 \leq i \leq m : \ \mathbf{v^i} = (v_1^i, v_2^i, \ldots, v_n^i)^T \in \mathbb{R}^n$ are the generator (basis) vectors that all m together create the generator matrix, and $P : \mathbb{R}^m \to \{\bot, \top\}$ is a predicate. The set represented by the star is given as:*

$$[\![\Theta]\!] = \left\{ \mathbf{x} \mid \mathbf{x} = \mathbf{c} + \sum_{i=1}^{m} \alpha_i \mathbf{v^i} \text{ such that } P(\alpha_1, ..., \alpha_m) = \top \right\}$$

Sometimes we will refer to both the tuple Θ and the set of states $[\![\Theta]\!]$ as Θ. In this work, we restrict the predicate to be a conjunction of linear constraints, $P(\boldsymbol{\alpha}) \triangleq \mathcal{C}\boldsymbol{\alpha} \leq \mathbf{d}$ where, for p linear constraints, $\mathcal{C} \in \mathbb{R}^{p \times m}$ is the matrix of the coefficients, $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_m)^T \in \mathbb{R}^m$, and $\mathbf{d} = (d_1, d_2, \ldots, d_p)^T \in \mathbb{R}^p$ is the limit vector. In this thesis, we will use P referring to the predicate (function with boolean output) and $P(\boldsymbol{\alpha})$ referring the the set of solutions (it is a real-valued convex set).

In this work, if nothing for a star set is specified, we will refer to this setup and call it a general star set.

$$\mathbf{c} = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} \quad \mathcal{V} = \begin{bmatrix} v_1^1 & \cdots & v_1^m \\ \vdots & \ddots & \vdots \\ v_n^1 & \cdots & v_n^m \end{bmatrix} \quad \mathcal{C} = \begin{bmatrix} C_{11} & \cdots & C_{1m} \\ \vdots & \ddots & \vdots \\ C_{p1} & \cdots & C_{pm} \end{bmatrix} \quad \mathbf{d} = \begin{bmatrix} d_1 \\ \vdots \\ d_p \end{bmatrix} \quad \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_m \end{bmatrix}$$

Example 2.2.1. *Let $\Theta = \langle \mathbf{c}, \mathcal{V}, P \rangle$ (Figure 2.3), $P \triangleq (\mathcal{C}\boldsymbol{\alpha} \leq \mathbf{d})$ (Figure 2.2), where:*

$$\mathbf{c} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad \mathcal{V} = \begin{bmatrix} 1 & 0 \\ 0 & -2 \end{bmatrix} \quad \mathcal{C} = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \quad \mathbf{d} = \begin{bmatrix} 3 \\ 2 \\ 1 \\ 1 \end{bmatrix}$$

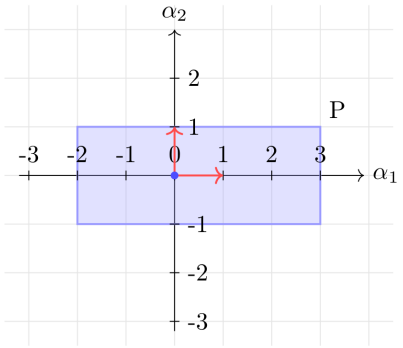


Figure 2.2: The predicate P of the star set in Example 2.2.1

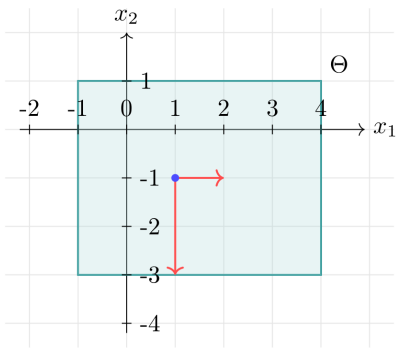


Figure 2.3: The star set in Example 2.2.1

In addition, an equivalent definition to the star set would be the following set:

$$[\![\Theta]\!] = \{(x_1, x_2) \mid -1 \leq x_1 \leq 4 \wedge -3 \leq x_2 \leq 1\}$$

Definition 2.2.2 (Dimension of the star set)**.** *The dimension of a star set is a tuple (n, m), where n is the number of rows and m is the number of columns in the generator matrix $\mathcal{V}$ of the star set.*

Definition 2.2.3 (0-column star set)**.** *The general star set $\Theta = \langle \mathbf{c}, \mathcal{V}, P \rangle$ is called 0-column star set if $\mathcal{V}$ is a 0-column matrix. To visualize this, without loss of generality, we assume that there is only one 0-column in the matrix and it is the first one: $\mathbf{v}^1 = \mathbf{0}$.*

$$\mathbf{c} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \quad \mathcal{V} = \begin{bmatrix} 0 & v_1^2 & \cdots & v_1^m \\ 0 & v_2^2 & \cdots & v_2^m \\ \vdots & \vdots & \ddots & \vdots \\ 0 & v_n^2 & \cdots & v_n^m \end{bmatrix} \quad \mathcal{C} = \begin{bmatrix} C_{11} & \cdots & C_{1m} \\ \vdots & \ddots & \vdots \\ C_{p1} & \cdots & C_{pm} \end{bmatrix} \quad \mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_p \end{bmatrix} \quad \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix}$$

In the upcoming propositions, we show the representational power and efficiency concerning certain operations of star sets. The proofs of all upcoming propositions can be found in Appendix A

Proposition 2.2.1. *Any convex polyhedron $\mathscr{P} \triangleq \{\mathbf{x} \mid \mathcal{C}\mathbf{x} \leq \mathbf{d}, \quad \mathbf{x} \in \mathbb{R}^n\}$ can be represented as a star.*

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

$$\bar{\Theta} = \langle \bar{\mathbf{c}}, \bar{\mathcal{V}}, \bar{P} \rangle, \quad \bar{\mathbf{c}} = \mathcal{W}\mathbf{c} + \mathbf{b}, \quad \bar{\mathcal{V}} = (\mathcal{W}\mathbf{v^1}, \cdots, \mathcal{W}\mathbf{v^m}), \quad \bar{P} \equiv P$$

The use of matrix multiplications to the basis and center and one addition of vectors, as well as preserving the predicate in the affine mapping of star sets, means that the complexity of the affine mapping of a star is constant. However, the fact that there is no known polynomial algorithm for the affine mapping of $\mathcal{H}$-polytopes indicates that the time complexity of affine mapping of $\mathcal{H}$-polytopes is exponential. In conclusion, star sets are an efficient alternative compared to $\mathcal{H}$-polytopes as mentioned earlier. Rotation is one of the many linear affine mappings and further, we will focus on it.

Example 2.2.2. *Let $\Theta = \langle \mathbf{c}, \mathcal{V}, P \rangle$ be the same as in Example 2.2.1, additionally consider for the affine mapping of the star Θ. The affine mapping matrix $\mathcal{W}$ and the offset vector $\mathbf{b}$:*

$$\mathcal{W} = \begin{bmatrix} \cos(45°) & -\sin(45°) \\ \sin(45°) & \cos(45°) \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -\frac{1}{2} \\ -\frac{1}{2} \end{bmatrix}$$

This affine mapping matrix $\mathcal{W}$ describes a 45° rotation.

$$\mathcal{W}\mathbf{v^1} = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix} = \bar{\mathbf{v}}^{\mathbf{1}}$$

$$\mathcal{W}\mathbf{v^2} = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} 0 \\ -2 \end{bmatrix} = \begin{bmatrix} \sqrt{2} \\ -\sqrt{2} \end{bmatrix} = \bar{\mathbf{v}}^{\mathbf{2}}$$

$$\bar{\mathcal{V}} = \begin{bmatrix} \bar{\mathbf{v}}^{\mathbf{1}} & \bar{\mathbf{v}}^{\mathbf{2}} \end{bmatrix}$$

$$\bar{\mathbf{c}} = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} -\frac{1}{2} \\ -\frac{1}{2} \end{bmatrix} = \begin{bmatrix} \frac{2\sqrt{2}-1}{2} \\ -\frac{1}{2} \end{bmatrix}$$

The resulting star set is defined as $\Theta' = \langle \bar{\mathbf{c}}, \bar{\mathcal{V}}, P \rangle$, where:

$$\textit{the basis } \bar{\mathcal{V}} = \begin{bmatrix} \frac{\sqrt{2}}{2} & \sqrt{2} \\ \frac{\sqrt{2}}{2} & -\sqrt{2} \end{bmatrix} \textit{ and the center } \bar{\mathbf{c}} = \begin{bmatrix} \frac{2\sqrt{2}-1}{2} \\ -\frac{1}{2} \end{bmatrix}$$

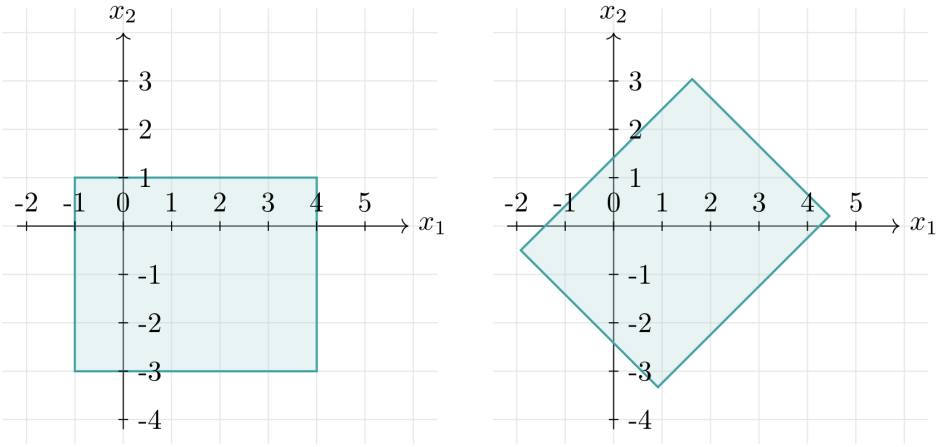


Figure 2.4: The result of affine mapping in Example 2.2.2

Proposition 2.2.3 (Star and half-space Intersection)**.** *The intersection of a star set $\Theta = \langle \mathbf{c}, \mathcal{V}, P \rangle$ and a half-space $\mathscr{H} = \{\mathbf{x} \mid \mathbf{H}^T\mathbf{x} \leq g\}$ is another star*

$$\bar{\Theta} = \Theta \cap \mathscr{H} = \langle \mathbf{c}, \mathcal{V}, \bar{P} \rangle \quad \textit{with} \quad \bar{P} = P \wedge P',$$

where $P'(\boldsymbol{\alpha}) \triangleq (\mathbf{H}^T\mathcal{V})\boldsymbol{\alpha} \leq g - \mathbf{H}^T\mathbf{c}$, and $\mathcal{V} = [\mathbf{v^1}, \mathbf{v^2}, ..., \mathbf{v^m}]$.

Intersection with half-spaces is very efficient when using star sets and they outperform the $\mathcal{V}$-Polytopes [Sch19].

Example 2.2.3. *Let* $\Theta = (\mathbf{c}, \mathcal{V}, P)$ *be the same as in Example 2.2.2 where*

$$\textit{the basis } \mathcal{V} = \begin{bmatrix} \frac{\sqrt{2}}{2} & \sqrt{2} \\ \frac{\sqrt{2}}{2} & -\sqrt{2} \end{bmatrix} \textit{ and the center } \mathbf{c} = \begin{bmatrix} \frac{2\sqrt{2}-1}{2} \\ -\frac{1}{2} \end{bmatrix},$$

also the predicate $P(\boldsymbol{\alpha}) \triangleq \mathcal{C}\boldsymbol{\alpha} \leq \mathbf{d}$ *where* $\mathcal{C} = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix}$ *and* $\mathbf{d} = \begin{bmatrix} 3 \\ 2 \\ 1 \\ 1 \end{bmatrix}$.

Additionally, let half-space $\mathscr{H}$ *be defined as:*

$$\mathscr{H} \triangleq \{\mathbf{x} \mid \mathbf{H}^T\mathbf{x} \leq g\} \quad \textit{with} \quad \mathbf{x} \in \mathbb{R}^2, \quad \mathbf{H} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad g = 3.$$

$P'(\alpha)$ *is defined by this inequality:*

$$[1,1] \begin{bmatrix} \frac{\sqrt{2}}{2} & \sqrt{2} \\ \frac{\sqrt{2}}{2} & -\sqrt{2} \end{bmatrix} \alpha \leq 3 - [1,1] \begin{bmatrix} \frac{2\sqrt{2}-1}{2} \\ -\frac{1}{2} \end{bmatrix}$$

By solving the inequality we get:

$$[\sqrt{2}, 0]\ \alpha \leq 4 - \sqrt{2}$$

The resulting star set becomes $\Theta' = \langle \mathbf{c}, \mathcal{V}, \bar{P} \rangle$*, where*

$$\bar{P}(\boldsymbol{\alpha}) \triangleq \bar{\mathcal{C}}\bar{\boldsymbol{\alpha}} \leq \bar{\mathbf{d}} \quad \textit{where} \quad \bar{\mathcal{C}} = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \\ \sqrt{2} & 0 \end{bmatrix}, \quad \bar{\mathbf{d}} = \begin{bmatrix} 3 \\ 2 \\ 1 \\ 1 \\ 4 - \sqrt{2} \end{bmatrix}.$$

Taking a closer look we can see, that $\bar{P}(\boldsymbol{\alpha})$ *is the same as* $\bar{P}_{reduced}(\boldsymbol{\alpha}) \triangleq \bar{\mathcal{C}}_{reduced} \leq \bar{\mathbf{d}}_{reduced}$*, with*

$$\bar{\mathcal{C}}_{reduced} = \begin{bmatrix} \sqrt{2} & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix}, \quad \bar{\mathbf{d}}_{reduced} = \begin{bmatrix} 4 - \sqrt{2} \\ 2 \\ 1 \\ 1 \end{bmatrix}$$

2.3 Variable elimination in a system of linear inequalities

The objective of eliminating a variable α (a set of variables can also be considered) from a system of linear inequalities is to derive another system of linear inequalities that does not contain the variable α (or without the set of variables that have been eliminated), while preserving equisatisfiability with the original system (i.e., the two systems are equisatisfiable) [Cha93]. Consequently, the resulting system is satisfiable (unsatisfiable) if and only if the original system is satisfiable (unsatisfiable). A key advantage of variable elimination is the reduction in the number of variables in the equisatisfiable system compared to the original one. While various methods address variable elimination problems, this thesis primarily focuses on the Fourier-Motzkin variable elimination method and its recent extension FMplex.

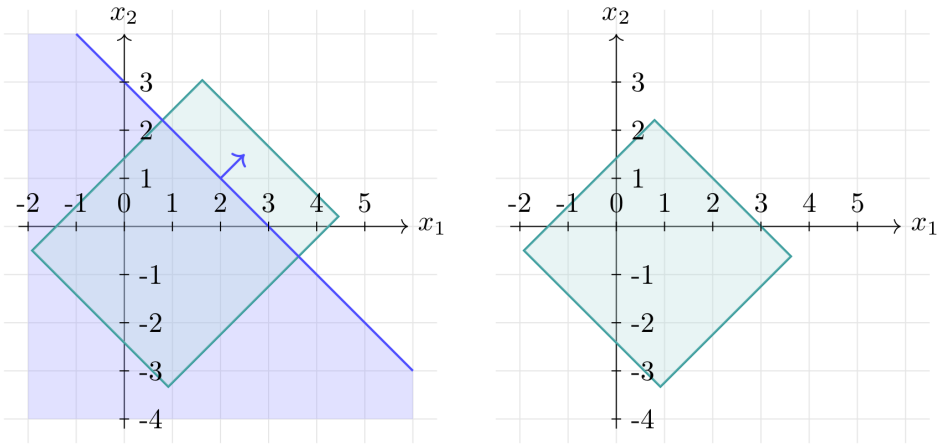


Figure 2.5: The illustration of the intersection of a star set with a half-space in Example 2.2.3

2.3.1 Fourier-Motzkin variable elimination

Fourier–Motzkin variable elimination (FM elimination) [Dan63], sometimes referred to as quantifier elimination or polytope projection [Sch98], is an algorithm utilized for eliminating variables from a system of linear inequalities.

Let P be a system of p linear inequalities over n variables $\alpha_1, \ldots, \alpha_n$. Consider P to be in the normal form: the inequalities have $\leq$ sign, all variables are on the left-hand side and the right-hand side is the real-valued limit (i here corresponds to the $i-th$ inequality in the system):

$$\sum_{j=1}^{n} C_{ij} \cdot \alpha_j \leq d_i$$

Without loss of generality, we assume α_p is the variable to be eliminated. We can rewrite the inequalities from the perspective of α_p:

$$C_{ip} \cdot \alpha_p \leq d_i - \sum_{j=1}^{n-1} C_{ij} \cdot \alpha_j \tag{2.1}$$

The linear inequalities in the system can be grouped into three classes depending on the sign (positive, negative, or null) of the coefficient C_{ip} for α_p.

$$C_{ip} = 0: \quad \Rightarrow \qquad \text{sets no bound on } \alpha_p. \tag{2.2}$$

$$C_{ip} > 0: \quad \Rightarrow \quad \alpha_p \leq \frac{d_i}{C_{ip}} - \frac{\sum_{j=1}^{n-1} C_{ij} \cdot \alpha_j}{C_{ip}}: \qquad \text{sets upper bound on } \alpha_p. \tag{2.3}$$

$$C_{ip} < 0: \quad \Rightarrow \quad \alpha_p \geq \frac{d_i}{C_{ip}} - \frac{\sum_{j=1}^{n-1} C_{ij} \cdot \alpha_j}{C_{ip}}: \qquad \text{sets lower bound on } \alpha_p. \tag{2.4}$$

The inequalities that set no bounds on α_p can be ignored because α_p does not appear in these inequalities. Consequently, eliminating α_p will not affect these inequalities. We denote this list of inequalities by N.

The remaining inequalities set either lower or upper bounds on α_p. Let U be the list of the right-hand sides of the inequalities of the form 2.3 and L be the list of the right-hand sides of the inequalities of the form 2.4. To eliminate α_p we generate new inequalities pairing each lower bound with each upper bound. Formally,

$$\forall l \in L, \forall u \in U : \ l \leq u,$$

and we call the list containing these newly generated inequalities E. The inequality system returned by FM is $N \cup E$. The returned system is equisatisfiable to the original system and does not contain variable α_p.

Let us examine why are both systems equisatisfiable. By construction, we substitute the variable α_p with a set of new inequalities that ensure that each lower bound is less or equal to each upper bound. If these inequalities hold for all pairs, we can deduce the existence of a value between the lower and upper bounds for α_p that satisfies the original inequalities. This assertion remains true due to the domain of variables under consideration being $\mathbb{R}$, which is dense. This denseness property signifies that for any two numbers, there always exists another number lying between them.

By iteratively applying this elimination process to all variables $\alpha_i : 1 \leq i \leq n$ we ultimately arrive at a system where no variables remain. With only constant values, we can easily determine whether any inequality is violated. If a violation is found, FM returns UNSAT; conversely, if all inequalities are upheld, FM returns SAT.

Example 2.3.1. *Consider the system of linear inequalities in the normal form below and perform an iterative elimination of* α_1, α_3.

$$\begin{aligned} \alpha_1 - \alpha_2 &\leq 0 \\ \alpha_1 - \alpha_3 &\leq 0 \\ -\alpha_1 + \alpha_2 + 2\alpha_3 &\leq 0 \\ -\alpha_3 &\leq -1 \end{aligned}$$

In the first iteration, we focus on eliminating α_1. *To achieve this, we rewrite the inequalities with respect to* α_1 *as shown in Formula 2.1:*

$$\alpha_1 - \alpha_2 \leq 0 \quad \Leftrightarrow \quad \alpha_1 \leq \alpha_2 \tag{2.5}$$

$$\alpha_1 - \alpha_3 \leq 0 \quad \Leftrightarrow \quad \alpha_1 \leq \alpha_3 \tag{2.6}$$

$$-\alpha_1 + \alpha_2 + 2\alpha_3 \leq 0 \quad \Leftrightarrow \quad \alpha_1 \geq \alpha_2 + 2\alpha_3 \tag{2.7}$$

$$-\alpha_3 \leq -1 \tag{2.8}$$

The Inequality 2.8 sets no bound on α_1. *So we ignore it for this iteration. The Inequalities 2.5 and 2.6 set an upper and 2.7 a lower bound on* α_1. *To eliminate* α_1, *we generate all pairs of lower and upper bounds of* α_1.

$$\alpha_2 + 2\alpha_3 \leq \alpha_2 \tag{2.9}$$

$$\alpha_2 + 2\alpha_3 \leq \alpha_3 \tag{2.10}$$

$$-\alpha_3 \leq -1 \tag{2.11}$$

The next step is to eliminate the α_3. *To do this, we first simplify and then rewrite the current system of inequalities regarding the* α_3.

$$2\alpha_3 \leq 0 \quad \Leftrightarrow \quad \alpha_3 \leq 0 \tag{2.12}$$

$$\alpha_2 + \alpha_3 \leq 0 \quad \Leftrightarrow \quad \alpha_3 \leq -\alpha_2 \tag{2.13}$$

$$-\alpha_3 \leq -1 \quad \Leftrightarrow \quad \alpha_3 \geq 1 \tag{2.14}$$

The Inequalities 2.12 and 2.13 set an upper and 2.14 a lower bound on α_3. To eliminate α_3, generate all pairs of lower and upper bounds of α_3.

$$1 \leq 0 \tag{2.15}$$
$$1 \leq -\alpha_2 \tag{2.16}$$

The Inequality 2.15 is not satisfiable, thus the system is unsatisfiable and FM returns UNSAT.

As shown in Example 2.3.1, each iteration yields a system with one fewer variable than the previous system. However, the number of inequalities increases after each iteration (though some of them might be redundant). Let us consider a scenario where the original system has p inequalities, with $p/2$ of them defining a lower bound and $p/2$ defining an upper bound on the variable to be eliminated. In this case, we obtain $p/2 \cdot p/2 = p^2/4$ new inequalities. In the worst-case scenario, the number of inequalities grows double-exponentially with each iteration: in the $i-th$ iteration, the number of inequalities becomes $4(\frac{p}{4})^{2^i}$ [Ábr23].

However, in practice, the algorithm's performance is often not double-exponential due to several factors. These include the non-equal distribution of lower and upper bounds and the removal of redundant constraints after each iteration. Redundant constraints, also known as redundant inequalities, are those that are *implied* by other inequalities in the system. Mathematically, an inequality is redundant if it can be expressed as a non-negative linear combination of other inequalities in the system. For example, consider the inequality system below:

$$\begin{cases} x & \leq 2 \\ y & \leq 1 \\ x+y & \leq 4 \end{cases}$$

In the system above $x + y \leq 4$ is a redundant constraint by taking the linear combination of the other two with coefficients one.

However, identifying redundant constraints is often computationally inefficient. It requires weighing the computational cost against the potential benefits of removing these constraints. This trade-off lies between the time invested in eliminating redundant constraints and the resulting system with fewer constraints for subsequent calculations.

Chapter 3

Starset dimension reduction

3.1 The need for dimension reduction

In this section, we will be focusing on the dimension reduction of a star set (Definition 2.2.2). Specifically, we will concentrate on the second parameter of the dimension tuple (n, m), namely on m. Consequently, unless explicitly stated otherwise, m will represent the dimension of the star set. Furthermore, the notion of dimension reduction refers to the downsizing of the parameter m.

Intuitively, a star set representation can be envisioned as the affine transformation of the solution set of the predicate (denoted as $P(\boldsymbol{\alpha})$). The generator matrix corresponds to the linear transformation, while the center shifts the result (as proven in Proposition 2.2.2 the result is also a star set). Now, consider a (n, m) dimensional star set, where $n < m$. In this case, the star set representation is an arbitrary n-dimensional convex polytope, which is obtained via projecting a higher, $m-$dimensional polytope: $P(\boldsymbol{\alpha})$.

In other words, we describe the n-dimensional set by embedding it within a higher m-dimensional star set. This suggests that in the predicate, there are more variables than would suffice to represent the set. Consequently, there exists some redundant information in the higher dimension (specifically the $m - n$ variables projected out during compression from a higher-dimensional space m to a lower-dimensional space n). It is evident that the minimum number of variables in the predicate required to represent a n-dimensional set is n and any additional variable in the predicate can be replaced by the linear combination of existing ones.

This work aims to propose an algorithmic solution, that explains, in which cases and how the star set dimension reduction can be done. The benefits after the reduction are: firstly, fewer variables in the predicate means faster computations when it comes to solving LP instances (for example when calculating the vertices, finding bounds, etc.) and secondly, the model becomes more interpretable, as irrelevant dimensions are removed, leaving only those essential for characterizing the set.

3.2 Handling 0-column generator matrix

In some scenarios, we are going to have a 0-column generator matrix $\mathcal{V}$ (see Definition 0.0.5). This is a special case and we will see, that having some 0-column generator

matrix is very convenient. First, assume that the generator matrix is given in such a form that it is a 0-column matrix, and let us understand why having a 0-column star set is beneficial.

Consider the 0-column star set $\Theta = \langle \mathbf{c}, \mathcal{V}, P \rangle, P \triangleq \mathcal{C}\boldsymbol{\alpha} \leq \mathbf{d}$:

$$\mathbf{c} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \mathcal{V} = \begin{bmatrix} 0 & v_1^2 & \cdots & v_1^m \\ 0 & v_2^2 & \cdots & v_2^m \\ \vdots & \vdots & \ddots & \vdots \\ 0 & v_n^2 & \cdots & v_n^m \end{bmatrix} \mathcal{C} = \begin{bmatrix} C_{11} & \cdots & C_{1m} \\ \vdots & \ddots & \vdots \\ C_{p1} & \cdots & C_{pm} \end{bmatrix} \mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_p \end{bmatrix} \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix}$$

The elements of the set $\mathbf{x} \in \mathbb{R}^n$ that the star represents are expressed as:

$$\begin{aligned} x_1 &= c_1 + v_1^1 \cdot \alpha_1 + v_1^2 \cdot \alpha_2 + \cdots + v_1^m \cdot \alpha_m \\ x_2 &= c_2 + v_2^1 \cdot \alpha_1 + v_2^2 \cdot \alpha_2 + \cdots + v_2^m \cdot \alpha_m \\ &\vdots \\ x_n &= c_n + v_n^1 \cdot \alpha_1 + v_n^2 \cdot \alpha_2 + \cdots + v_n^m \cdot \alpha_m \end{aligned}$$

$\mathbf{v^1}$ is a null vector, thus, the equations can be written in this form:

$$\begin{aligned} x_1 &= c_1 + 0 \cdot \alpha_1 + v_1^2 \cdot \alpha_2 + \cdots + v_1^m \cdot \alpha_m = c_1 + v_1^2 \cdot \alpha_2 + \cdots + v_1^m \cdot \alpha_m \\ x_2 &= c_2 + 0 \cdot \alpha_1 + v_2^2 \cdot \alpha_2 + \cdots + v_2^m \cdot \alpha_m = c_2 + v_2^2 \cdot \alpha_2 + \cdots + v_2^m \cdot \alpha_m \\ &\vdots \\ x_n &= c_n + 0 \cdot \alpha_1 + v_n^2 \cdot \alpha_2 + \cdots + v_n^m \cdot \alpha_m = c_n + v_n^2 \cdot \alpha_2 + \cdots + v_n^m \cdot \alpha_m \end{aligned}$$

This means that α_1 does not directly "contribute" to the value of any $x_i : 1 \leq i \leq n$. Consequently, the intuition arises that there should be a method for representing the same set Θ without the predicate variable α_1. This means that in the star set that we are searching for, the variable α_1 has to be removed. However, eliminating α_1 cannot be done arbitrarily while maintaining everything the same as they are. The predicate must be adjusted accordingly because even though α_1 does not directly impact x_i, it still influences $(\alpha_2, \alpha_3, \cdots, \alpha_m)$ through constraints involving both α_1 and the other α-s. This influence entails internal changes to $P(\boldsymbol{\alpha})$. Below will be shown, how α_1 affects the values of other α-s.

Question 1. *Can the variable corresponding to the 0-column be thrown away? To answer this question, let us explicitly write the inequalities in the system* $\mathcal{C}\boldsymbol{\alpha} \leq \mathbf{d}$.

$$\begin{aligned} C_{11} \cdot \alpha_1 + C_{12} \cdot \alpha_2 + \cdots + C_{1m} \cdot \alpha_m &\leq d_1 \\ C_{21} \cdot \alpha_1 + C_{22} \cdot \alpha_2 + \cdots + C_{2m} \cdot \alpha_m &\leq d_2 \\ &\vdots \\ C_{p1} \cdot \alpha_1 + C_{p2} \cdot \alpha_2 + \cdots + C_{pm} \cdot \alpha_m &\leq d_p \end{aligned}$$

Furthermore, we can conclude that in any of the inequalities $i \in \{1, \cdots, p\}$ *and any* $\alpha_j \in \{\alpha_2, \cdots, \alpha_m\}$ *holds:*

$$C_{ij} \cdot \alpha_j \leq d_i - \mathbf{C_{i1}}\boldsymbol{\alpha_1} - \sum_{k=2, k\neq j}^{m} C_{ik}\alpha_k \tag{3.1}$$

As shown in 3.1, α_1 *still has an impact on the satisfying valuations of the other* α*-s, thus on* $P(\boldsymbol{\alpha})$*.*

This emphasizes the necessity of implementing the requisite changes to the predicate before eliminating α_i corresponding to the 0-column. It is worth to note, that if C_{i1} is zero, then it does not influence $P(\boldsymbol{\alpha})$. Thus, the question arises: Is there a method to obtain a system $P'(\boldsymbol{\alpha})$, equisatisfiable to the original system $P(\boldsymbol{\alpha})$ with $C_{i1} = 0 : \forall i \in \{1, \cdots, p\}$. The answer is yes, and the techniques that yield such equisatisfiable systems are known as variable (quantifier) elimination algorithms. One such algorithm is described in Section 2.3.1: *Fourier Motzkin variable elimination (FM elimination).* This method effectively nullifies the influence of α_i (the variable to be eliminated) on the other α-s, by setting all the coefficients of α_i to zero in all constraints within the system. In other words, FM elimination projects out the dimension of the variable we aim to eliminate from the solution of the original system.

Proposition 3.2.1. *Applying Fourier Motzkin variable elimination on a given system of linear inequalities* $\mathcal{C}\boldsymbol{\alpha} \leq \mathbf{d}$ *to eliminate* α_i *from the system, we get an equisatisfiable system* $\mathcal{C}'\boldsymbol{\alpha} \leq \mathbf{d}'$*, where all coefficients of* α_i *are set to zero.*

Proof. As the variable α_i is not present in the resulting system $\mathcal{C}'\boldsymbol{\alpha} \leq \mathbf{d}'$ anymore (because we eliminated α_i), the coefficients of α_i are zero. □

After applying FM elimination on the predicate $\mathcal{C}\boldsymbol{\alpha} \leq \mathbf{d}$, we get an equisatisfiable system of linear inequalities $\mathcal{C}'\boldsymbol{\alpha} \leq \mathbf{d}'$, where all coefficients of the variable α_i are zero. At this stage, proceed to remove the 0-column from both the constraint matrix $\mathcal{C}'$ and the generator matrix $\mathcal{V}$. We will name them ${\mathcal{C}'}_{elim}$ and ${\mathcal{V}'}_{elim}$ respectively. The star set equivalent to the original one is denoted as $\Theta' = \langle \mathbf{c}, {\mathcal{V}'}_{elim}, P' \rangle$, where $P' \triangleq {\mathcal{C}'}_{elim}\boldsymbol{\alpha'} \leq \mathbf{d}'$.

For better understanding, let us look at an example.

Example 3.2.1 (0-column in the generator matrix)**.** $\mathbf{x} \in [\![\Theta]\!]$ *and star set* $\Theta = \langle \mathbf{c}, \mathcal{V}, P \rangle, P \triangleq \mathcal{C}\boldsymbol{\alpha} \leq \mathbf{d}$*, where:*

$$\mathbf{c} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \mathcal{V} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \mathcal{C} = \begin{bmatrix} -1 & -1 & 0 \\ -1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & -1 \\ \frac{\sqrt{2}-6}{10} & 0 & 1 \end{bmatrix}, \mathbf{d} = \begin{bmatrix} \sqrt{2}+1 \\ \sqrt{2} \\ 2\sqrt{2}-1 \\ \sqrt{2} \\ 0 \\ 0 \\ \frac{11\sqrt{2}+2}{20} \end{bmatrix}, \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}$$

These parameters describe the following star set, graphically depicted in Figure 3.1:

$$x_1 = c_1 + v_1^1 \cdot \alpha_1 + v_1^2 \cdot \alpha_2 + v_1^3 \cdot \alpha_3 = 0 + 0 \cdot \alpha_1 + 0 \cdot \alpha_2 + 1 \cdot \alpha_3$$
$$x_2 = c_2 + v_2^1 \cdot \alpha_1 + v_2^2 \cdot \alpha_2 + v_2^3 \cdot \alpha_3 = 0 + 0 \cdot \alpha_1 + 1 \cdot \alpha_2 + 0 \cdot \alpha_3$$

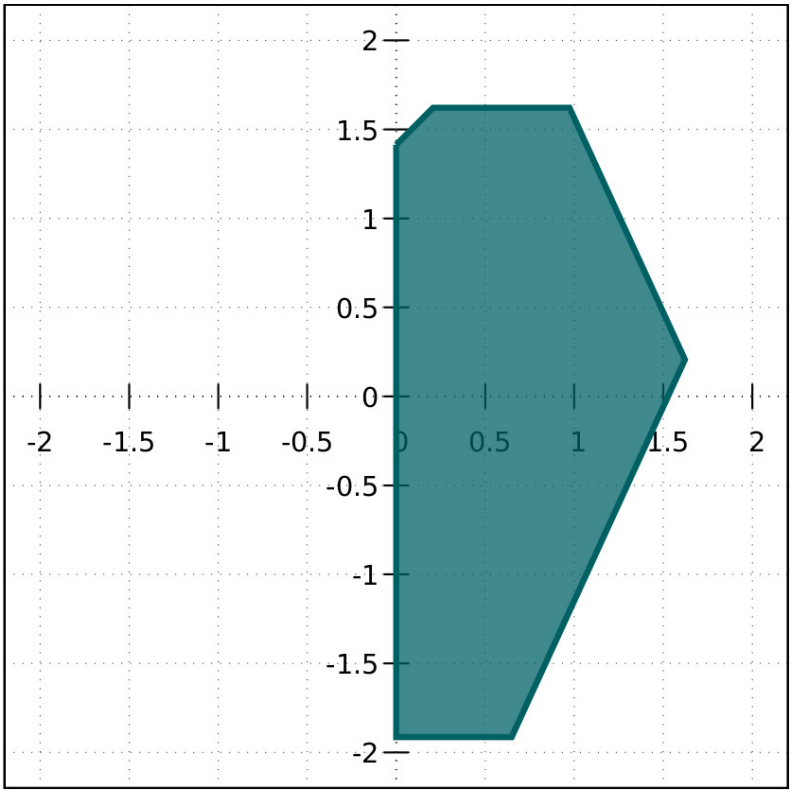


Figure 3.1: The illustration of the set presented in Example 3.2.1

In this example, we observe that there is a 0-column (α_1) in the generator matrix. As described earlier, we use the FM elimination introduced in Subsection 2.3.1 to eliminate α_1. In the first step, we rewrite the inequalities with respect to α_1:

$$-\alpha_1 - \alpha_2 \leq \sqrt{2}+1 \qquad -\alpha_2 - \sqrt{2} - 1 \leq \alpha_1 \tag{3.2}$$

$$-\alpha_1 + \alpha_2 \leq \sqrt{2} \qquad \alpha_2 - \sqrt{2} \leq \alpha_1 \tag{3.3}$$

$$\alpha_1 + \alpha_2 \leq 2\sqrt{2} - 1 \qquad \alpha_1 \leq -\alpha_2 + 2\sqrt{2} - 1 \tag{3.4}$$

$$\alpha_1 - \alpha_2 \leq \sqrt{2} \qquad \alpha_1 \leq \sqrt{2}\alpha_2 \tag{3.5}$$

$$-\alpha_3 \leq 0 \qquad -\alpha_3 \leq 0 \tag{3.6}$$

$$\alpha_1 - \alpha_3 \leq 0 \qquad \alpha_1 \leq \alpha_3 \tag{3.7}$$

$$\frac{\sqrt{2}-6}{10}\alpha_1 + \alpha_3 \leq \frac{11\sqrt{2}+2}{20} \qquad \frac{5(\sqrt{2}-6)}{17}\alpha_3 - \frac{2\sqrt{2}+1}{2} \leq \alpha_1 \tag{3.8}$$

As you can see, in Inequality 3.6 α_1 is not present (Inequality 3.6 sets no bound on α_1), so for the next step, it can be ignored. The Inequalities 3.2, 3.3, and 3.8 set a lower and 3.4, 3.5, and 3.7 an upper bound on α_1. We generate new inequalities by

building all possible combinations of lower and upper bounds:

$$\begin{aligned}
-\alpha_2 - \sqrt{2} - 1 &\leq 2\sqrt{2} - 1 - \alpha_2 \\
-\alpha_2 - \sqrt{2} - 1 &\leq \sqrt{2} + \alpha_2 \\
-\alpha_2 - \sqrt{2} - 1 &\leq \alpha_3 \\
\alpha_2 - \sqrt{2} &\leq 2\sqrt{2} - 1 - \alpha_2 \\
\alpha_2 - \sqrt{2} &\leq \alpha_2 + \sqrt{2} \\
\alpha_2 - \sqrt{2} &\leq \alpha_3 \\
\frac{5(\sqrt{2}-6)}{17}\alpha_3 - \frac{2\sqrt{2}+1}{2} &\leq 2\sqrt{2} - 1 - \alpha_2 \\
\frac{5(\sqrt{2}-6)}{17}\alpha_3 - \frac{2\sqrt{2}+1}{2} &\leq \sqrt{2} + \alpha_2 \\
\frac{5(\sqrt{2}-6)}{17}\alpha_3 - \frac{2\sqrt{2}+1}{2} &\leq \alpha_3
\end{aligned}$$

Now let us remove the redundant inequalities, simplify the remaining ones, and bring back Inequality 3.6.

$$\begin{aligned}
-\alpha_3 &\leq 0 \\
-2\alpha_2 &\leq 2\sqrt{2} + 1 \\
2\alpha_2 &\leq 3\sqrt{2} - 1 \\
\alpha_2 - \alpha_3 &\leq \sqrt{2} \\
\alpha_2 + \frac{5(\sqrt{2}+6)}{17}\alpha_3 &\leq \frac{6\sqrt{2}-1}{2} \\
-\alpha_2 + \frac{5(\sqrt{2}+6)}{17}\alpha_3 &\leq \frac{4\sqrt{2}+1}{2}
\end{aligned}$$

In the resulting set of inequalities, we see, that there is no α_1. This means, that for our star set, the generator matrix can be modified and the 0-column can be removed. The resulting star set is $\Theta' = \langle \mathbf{c}', \mathcal{V}', P' \rangle$, with $\mathbf{c}' = \mathbf{c}$, $P' \triangleq \mathcal{C}'_{elim}\boldsymbol{\alpha} \leq \mathbf{d}'$ and:

$$\mathcal{V}' = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \mathcal{C}'_{elim} = \begin{bmatrix} 0 & -1 \\ -2 & 0 \\ 2 & 0 \\ 1 & -1 \\ 1 & \frac{5(\sqrt{2}+6)}{17} \\ -1 & \frac{5(\sqrt{2}+6)}{17} \end{bmatrix}, \mathbf{d}' = \begin{bmatrix} 0 \\ 2\sqrt{2}+1 \\ 3\sqrt{2}-1 \\ \sqrt{2} \\ \frac{6\sqrt{2}-1}{2} \\ \frac{4\sqrt{2}+1}{2} \end{bmatrix}$$

Plotting the resulting star set, we see that the resulting set is the same as it was in Figure 3.1, before doing the elimination. This means that we could successfully remove one variable ($alpha_1$) from the star set while not changing the set itself.

In consequence, we can conclude that **if the generator is a 0-column matrix,** FM elimination can be directly applied to reduce the dimension of the star set. Furthermore, if after one iteration of FM elimination, there is another 0-column in

the generator matrix, we can iterate this procedure and successively eliminate α-s, corresponding to the zero columns in the generator, to reduce the dimension of the star set.

3.3 Transforming into 0-column star set

As shown in the previous Section 3.2, we can eliminate a column (or multiple columns) from the generator matrix if this is a 0-column matrix. For this, we apply the FM elimination on the variable, which corresponds to the 0-column. However, this is not a general instance, and in most cases, the generator matrix is unlikely to be a 0-column matrix.

In this section, we focus on addressing the scenario wherein the generator is not a 0-column matrix. Specifically, we explore whether we can implement any modifications to the star set to eliminate certain variables. In some cases yes. In the following, we present the requisite condition and the adjustments applied to the star set to bring it into a form where elimination can be done.

As mentioned, the ideal form of the generator is a 0-column matrix. Even when the matrix lacks 0 columns, we can still generate some through elementary column transformations. The number of possible 0-columns is strongly related to the dimension of the column space (also known as the rank of the matrix, Definition 0.0.8). If the rank of the matrix is less than the number of its columns (indicating a non-full column rank), then some columns are linearly dependent. This redundancy implies that linearly dependent columns can be expressed as linear combinations of other columns in the matrix. Therefore, these redundant columns can be eliminated without losing any information, simplifying the representation of the matrix.

The proposed method for reducing the dimension of a star set involves generating as many 0-columns in the generator matrix as possible and subsequently eliminating them using FM elimination, as demonstrated in Section 3.2.

Consider a general star set $\Theta = \langle \mathbf{c}, \mathcal{V}, P \rangle, P \triangleq (\mathcal{C}\boldsymbol{\alpha} \leq \mathbf{d})$ and the generator matrix $\mathcal{V} \in \mathbb{R}^{n \times m}$ and the condition $n < m$. As proven in Section A of the appendix of this thesis (Proposition A.0.1) $\mathcal{V}$ has at least $m - n$ linearly dependent columns. This means the rank of this matrix is bounded by the number of rows (since the matrix has fewer rows than columns). Bounded, because there can still exist rows that are linearly dependent from other rows. Therefore, it is the same as $\text{rank}(\mathcal{V}) \leq n$.

Using the fact of linear dependency between the generator vectors and employing elementary column operations, we can transform the generator matrix $\mathcal{V}$ into a 0-column matrix $\mathcal{V}'$. The number of 0-columns in $\mathcal{V}'$ precisely equals $m - \text{rank}(\mathcal{V})$. To get a star set Θ' representing the same set as Θ, we have to adapt the predicate P as well. Below we show that such transformations for the generator matrix $\mathcal{V}$ and predicate P exist. To do this, we will move to the initial description of a star set and show step by step how $\mathcal{V}'$ and P' can be obtained.

Proposition 3.3.1 (Star set transformation)**.** *Any general star set* $\Theta = \langle \mathbf{c}, \mathcal{V}, P \rangle,\ P \triangleq (\mathcal{C}\boldsymbol{\alpha} \leq \mathbf{d})$ *can be transformed to another general star set* $\Theta' = \langle \mathbf{c}', \mathcal{V}', P' \rangle,\ P' \triangleq (\mathcal{C}'\boldsymbol{\alpha}' \leq \mathbf{d}')$*, such that* $\mathcal{V} \neq \mathcal{V}', \mathcal{C} \neq \mathcal{C}'$ *and* $\Theta \equiv \Theta'$*. For the transformation, we use a manually designed invertible matrix* $\mathcal{G}$*, such that* $\mathcal{V}\mathcal{G} = \mathcal{V}'$*.*

Proof. Using the definition of the star set and applying some transformations, we get:

$$\begin{aligned}\Theta &= \{\mathbf{x} = \mathcal{V}\boldsymbol{\alpha} + \mathbf{c} \quad | \quad \mathcal{C}\boldsymbol{\alpha} \leq \mathbf{d}\} \qquad (3.9)\\ &= \{\mathbf{x} = \underbrace{\mathcal{V}\mathcal{G}}_{=\mathcal{V}'}\mathcal{G}^{-1}\boldsymbol{\alpha} + \mathbf{c} \quad | \quad \mathcal{C}\boldsymbol{\alpha} \leq \mathbf{d}\}\\ &= \{\mathbf{x} = \mathcal{V}'\underbrace{\mathcal{G}^{-1}\boldsymbol{\alpha}}_{=\boldsymbol{\alpha}'} + \mathbf{c} \quad | \quad \mathcal{C}\boldsymbol{\alpha} \leq \mathbf{d}\}\\ &= \{\mathbf{x} = \mathcal{V}'\boldsymbol{\alpha}' + \mathbf{c} \quad | \quad \mathcal{C}\boldsymbol{\alpha} \leq \mathbf{d}\}. \qquad (3.10)\end{aligned}$$

By multiplying the equation $\mathcal{G}^{-1}\boldsymbol{\alpha} = \boldsymbol{\alpha}'$ by $\mathcal{G}$ from the left side, we obtain:

$$\underbrace{\mathcal{G}\mathcal{G}^{-1}}_{I_m}\boldsymbol{\alpha} = \mathcal{G}\boldsymbol{\alpha}' \Rightarrow \boldsymbol{\alpha} = \mathcal{G}\boldsymbol{\alpha}'. \qquad (3.11)$$

Hence, with (3.10) and (3.11):

$$\begin{aligned}\Theta &= \{\mathbf{x} = \mathcal{V}'\boldsymbol{\alpha}' + \mathbf{c} \quad | \quad \underbrace{\mathcal{C}\mathcal{G}}_{=\mathcal{C}'}\boldsymbol{\alpha}' \leq \mathbf{d}\}\\ &= \{\mathbf{x} = \mathcal{V}'\boldsymbol{\alpha}' + \mathbf{c} \quad | \quad \mathcal{C}'\boldsymbol{\alpha}' \leq \mathbf{d}\} = \Theta'. \qquad (3.12)\end{aligned}$$

□

The proof suggests that two parameters stay the same: $\mathbf{c}' = \mathbf{c}$ *and* $\mathbf{d}' = \mathbf{d}$.
In this thesis, the matrix $\mathcal{G}$ *will be referred to as the transformation matrix.*

The founded equation shows that with the help of transformation matrix $\mathcal{G}$ we get the same star set but different generator $\mathcal{V}$ and constraint matrix $\mathcal{C}$. However, we should be careful while choosing the matrix $\mathcal{G}$. In most cases, there are infinitely many $\mathcal{G}$-s that satisfy the equation $\mathcal{V}\mathcal{G} = \mathcal{V}'$. The only restriction is that $\mathcal{G}$ must be invertible, although the inverse of $\mathcal{G}$ does not need to be calculated. This careful consideration of the transformation matrix is essential for the success and validity of the star set transformation process.

Lemma 3.3.2. *For any star set transformation from* $\Theta = \langle \mathbf{c}, \mathcal{V}, P\rangle,\ P \triangleq (\mathcal{C}\boldsymbol{\alpha} \leq \mathbf{d})$ *to* $\Theta' = \langle \mathbf{c}, \mathcal{V}', P'\rangle,\ P' \triangleq (\mathcal{C}'\boldsymbol{\alpha}' \leq \mathbf{d})$ *with* $\mathcal{V}, \mathcal{V}' \in \mathbb{R}^{n\times m}$ *and* $\Theta \equiv \Theta'$ *holds: if rank*$(\mathcal{V}) = r < m$*, then there exists an invertible transformation matrix* $\mathcal{G} \in \mathbb{R}^{m\times m}$*, such that* $\mathcal{V}\mathcal{G} = \mathcal{V}'$ *and* $\mathcal{V}'$ *has exactly* $\boldsymbol{r}$ *linear independent columns from* $\mathcal{V}$ *and* $\boldsymbol{m}$**-**$\boldsymbol{r}$ *zero-columns.*

Proof. From the rank-nullity theorem (Definition 0.0.11) follows, that

$$nullity(\mathcal{V}) = m - r, \qquad (3.13)$$

where $r = \text{rank}(\mathcal{V})$. Let J be the set of the indices of the linear dependent columns of $\mathcal{V}, M := \{1, \ldots, m\}, I = M \setminus J$. Then $|J| = m - r$. Let $\mathbf{k_j} \in \mathbb{R}^m, j \in J$ be linearly independent vectors forming a basis for the null space (also called kernel, Definition 0.0.11) of $\mathcal{V}$, which means:

$$\mathcal{V}\mathbf{k_j} = \mathbf{0},\ j \in J. \qquad (3.14)$$

From Equation 3.13 follows, that any set of $m - r$ linear independent vectors from the kernel will form a base for it. We can construct the transformation matrix $\mathcal{G} = [\,\mathbf{g_1}, \ldots, \mathbf{g_m}\,],\ \mathbf{g_i} \in \mathbb{R}^m,\ 1 \leq i \leq m$ as follows:

$$\mathbf{g_i} := \begin{cases}\mathbf{k_i}, & i \in J,\\ \mathbf{e_i}, & i \in I,\end{cases} \qquad (3.15)$$

where $i \in M$ and $\mathbf{e_i}$ is the i-th unit vector (Definition 0.0.3) in $\mathbb{R}^m$. Then, with Equations 3.14 and 3.15:

$$\mathcal{V}\mathcal{G} = \mathcal{V}',$$

where $\mathcal{V}'$ is the matrix having exactly the r linear independent columns from $\mathcal{V}$ and $m - r$ zero-columns.

Next, we want to show that the transformation matrix $\mathcal{G}$ is invertible.

For this, we need to prove that the columns of $\mathcal{G}$ are linearly independent. That means:

$$\alpha_j, \beta_i \in \mathbb{R},\ j \in J, i \in I : \sum_{j\in J} \alpha_j \mathbf{k_j} + \sum_{i\in I} \beta_i \mathbf{e_i} = \mathbf{0} \implies \alpha_j = 0,\ \beta_i = 0\ \forall j \in J, i \in I. \tag{3.16}$$

Assume, the linear combination in Equation 3.16 equals $\mathbf{0}$. By multiplying the linear combination by the matrix $\mathcal{V}$ we obtain:

$$\mathcal{V}\left(\sum_{j\in J} \alpha_j \mathbf{k_j} + \sum_{i\in I} \beta_i \mathbf{e_i}\right) = \mathbf{0}$$

$$\iff \sum_{j\in J} \alpha_j \underbrace{\mathcal{V}\mathbf{k_j}}_{=\mathbf{0}} + \sum_{i\in I} \beta_i \mathcal{V}\mathbf{e_i} = \mathbf{0} \implies \sum_{i\in I} \beta_i \mathcal{V}\mathbf{e_i} = \mathbf{0}.$$

I is the set of indices of linearly independent columns of $\mathcal{V}$ and $\mathcal{V}\mathbf{e_i}$ is the $i-th$ column of $\mathcal{V}$. Therefore, $\beta_i = 0\ \forall i \in I$. That means:

$$\sum_{j\in J} \alpha_j \mathbf{k_j} + \sum_{i\in I} \beta_i \mathbf{e_i} = \sum_{j\in J} \alpha_j \mathbf{k_j} = \mathbf{0}$$

From the linear independence of the vectors $\mathbf{k_j}, j \in J$ follows, that:

$$\alpha_j = 0 :\ \forall j \in J.$$

□

In particular, we get the generator $\mathcal{V}'$ that has only the columns, which form the column space of the original generator matrix $\mathcal{V}$ (the **image** of $\mathcal{V}$, Definition 0.0.10).

3.4 Dimension reduction method

In this section, we want to summarize all the steps described in Sections 3.2 and 3.3 and formulate a clear, step-by-step algorithm and apply this on an example.

Example 3.4.1 (No 0-column in the generator matrix)**.** *Now let us consider the following star set* $\Theta = \langle \mathbf{c}, \mathcal{V}, P \rangle$ *and* $P \triangleq \mathcal{C}\boldsymbol{\alpha} \leq \mathbf{d}$*, where:*

$$\mathbf{c} = \begin{bmatrix} 0 \\ -\frac{1}{2} \end{bmatrix},\ \mathcal{V} = \begin{bmatrix} 0 & 0 & 1 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \end{bmatrix},\ \mathcal{C} = \begin{bmatrix} -1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & -1 \\ \frac{-3\sqrt{2}+1}{10} & \frac{3\sqrt{2}-1}{10} & 1 \end{bmatrix},\ \mathbf{d} = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 1 \\ 0 \\ 0.5 \\ \frac{3\sqrt{2}-1}{5} \end{bmatrix}$$

Algorithm 1 Dimension reduction method of star set

Input: $\Theta = \langle \mathbf{c}, \mathcal{V}, P \rangle,\ P \triangleq \mathcal{C}\boldsymbol{\alpha} \leq \mathbf{d}$
Output: $\Theta' = \langle \mathbf{c}, \mathcal{V}', P' \rangle,\ P' \triangleq \mathcal{C}'\boldsymbol{\alpha'} \leq \mathbf{d}'$

1: $m = \mathcal{V}.cols()$ ▷ number of columns in $\mathcal{V}$
2: $r = \mathcal{V}.rank()$
3: **if** $r < m$ **then** ▷ if false, method not applicable
4: $\mathcal{V}' = \mathcal{V}.image()$
5: $linDep = \mathcal{V}.getLinDepCols()^*$ ▷ list of column indices that are not in the image
6: $\mathcal{G} = \mathcal{I}_m$ ▷ identity matrix of the size m
7: $\mathcal{K} = \mathcal{V}.kernel()$ ▷ matrix which columns form basis of the kernel of $\mathcal{V}$
8: **for** $i = 1 : linDep.length()$ **do**
9: $\mathcal{G}.col(linDep[i]) = \mathcal{K}.col(i)$ ▷ replace $i-th$ column in $\mathcal{G}$ with current kernel vector
10: **end for**
11: $\mathcal{C}_{FM} = \mathcal{C} * \mathcal{G}$ ▷ applying transformation by $\mathcal{G}$
12: $\mathcal{C}'_{FM}, \mathbf{d}' = FMelimination(\mathcal{C}_{FM}, \mathbf{d}, linDep)$ ▷ iteratively eliminate all variables in linDep from the system and return the final constraints matrix and limits vector
13: $\mathcal{C}' = \mathcal{C}'_{FM}.removeZeroColumns()$
14: OPTIONAL: Remove redundant constraints from $\mathcal{C}'\boldsymbol{\alpha'} \leq \mathbf{d}'$
15: **return** $\Theta' = \langle \mathbf{c}, \mathcal{V}', P' \rangle,\ P' \triangleq \mathcal{C}'\boldsymbol{\alpha'} \leq \mathbf{d}'$
16: **end if**

* If there are some identical columns in $\mathcal{V}$, only one of them is in the image, and the indices of all remaining ones should be added to this list.

This star set represents the same set, as Example 3.2.1 before (graphically Figure 3.1).

As we see, there is no 0-column in the generator matrix, so we cannot directly apply FM elimination and eliminate one of the variables. However, rank$(\mathcal{V}) < 3$*. This means we can apply the steps of Algorithm 1.*

First, we compute the image of $\mathcal{V}$*:*

$$\mathcal{V}' = \begin{bmatrix} 0 & 1 \\ \frac{\sqrt{2}}{2} & 0 \end{bmatrix}$$

Then we store the indices of those columns that are not in the image. Note that the first and second columns are identical, and as stated in the algorithm above, we need to add the index of the second one to the list. We start the indexing with 1*.*

$$linDep = [2]$$

In the next step, we create the identity matrix of size 3*.*

$$\mathcal{G} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Next, we compute the kernel of $\mathcal{V}$*.*

$$\mathcal{K} = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$$

We replace each column in $\mathcal{G}$ which index is in linDep list. In this case, it is only the second column. After replacing, $\mathcal{G}$ looks as follows.

$$\mathcal{G} = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Note that, $\mathcal{VG} = \mathcal{V}''$, such that $\mathcal{V}''$ is identical to $\mathcal{V}$ except it has 0-column at position 2.

As the next step, we prepare the predicate to apply FM elimination: $\mathcal{C}_{FM} = \mathcal{CG}$.

$$\mathcal{C}_{FM} = \begin{bmatrix} -1 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ \frac{\sqrt{2}}{2} & -\sqrt{2} & -1 \\ -\frac{3\sqrt{2}-1}{10} & \frac{3\sqrt{2}-1}{5} & 1 \end{bmatrix}$$

Now we have to apply FM elimination on the system linear inequalities $\mathcal{C}_{FM}\boldsymbol{\alpha} \leq \boldsymbol{d}$ to eliminate the columns whose indices are in the list linDep, namely in this case only the second column.

After applying FM, it returns the following system:

$$\mathcal{C'}_{FM} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0.5 & 0 & \frac{5(3\sqrt{2}+1)}{17} \\ -1 & 0 & 0 \\ 0 & 0 & 0 \\ -0.5 & 0 & \frac{5(3\sqrt{2}+1)}{17} \\ -0.5 & 0 & -\frac{\sqrt{2}}{2} \\ 0.5 & 0 & -\frac{\sqrt{2}}{2} \\ 0 & 0 & \frac{13\sqrt{2}+10}{34} \end{bmatrix}, \quad \boldsymbol{d'} = \begin{bmatrix} 3 \\ 3 \\ 3 \\ 2 \\ 2 \\ 2 \\ \frac{1+2\sqrt{2}}{2\sqrt{2}} \\ \frac{1+2\sqrt{2}}{2\sqrt{2}} \\ \frac{1+2\sqrt{2}}{2\sqrt{2}} \end{bmatrix}$$

As the final step, we need to erase the 0-columns in the $\mathcal{C'}_{FM}$.

$$\mathcal{C'} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0.5 & \frac{5(3\sqrt{2}+1)}{17} \\ -1 & 0 \\ 0 & 0 \\ -0.5 & \frac{5(3\sqrt{2}+1)}{17} \\ -0.5 & -\frac{\sqrt{2}}{2} \\ 0.5 & -\frac{\sqrt{2}}{2} \\ 0 & \frac{13\sqrt{2}+10}{34} \end{bmatrix}$$

If we also consider the optional step of removing the redundant constraints (which are the first, fifth, and seventh in this example), the predicate looks as follows:

$$\mathcal{C}' = \begin{bmatrix} 1 & 0 \\ 0.5 & \frac{5(3\sqrt{2}+1)}{17} \\ -1 & 0 \\ -0.5 & \frac{5(3\sqrt{2}+1)}{17} \\ 0.5 & -\frac{\sqrt{2}}{2} \\ 0 & \frac{13\sqrt{2}+10}{34} \end{bmatrix}, \quad \boldsymbol{d'} = \begin{bmatrix} 3 \\ 3 \\ 2 \\ 2 \\ \frac{1+2\sqrt{2}}{2\sqrt{2}} \\ \frac{1+2\sqrt{2}}{2\sqrt{2}} \end{bmatrix}$$

The resulting star set $\Theta' = \langle \mathbf{c}, \mathcal{V}', P' \rangle$ *with* $P' \triangleq \mathcal{C}'\boldsymbol{\alpha'} \leq \mathbf{d}'$ *describes the same set as the previous* $\Theta = \langle \mathbf{c}, \mathcal{V}, P \rangle$ *with* $P \triangleq \mathcal{C}\boldsymbol{\alpha} \leq \mathbf{d}$*, and but it has* 1 *variable less in the predicate.*

As one can see, the predicate without redundant constraints, has six constraints. In our example, the number of constraints returned by the FM elimination was nine, but only six of them were non-redundant. This is an indicator, that a naive implementation of FM elimination may return many redundant constraints. This aspect is later discussed in Chapter 5.

Chapter 4

Application on neural network verification

In this chapter, we will first give some basic principles of the feedforward neural network (FNN), and then discuss the star-based reachability method. Finally, we will discuss the application of star set dimension reduction within this context.

4.1 Feedforward neural networks (FNNs)

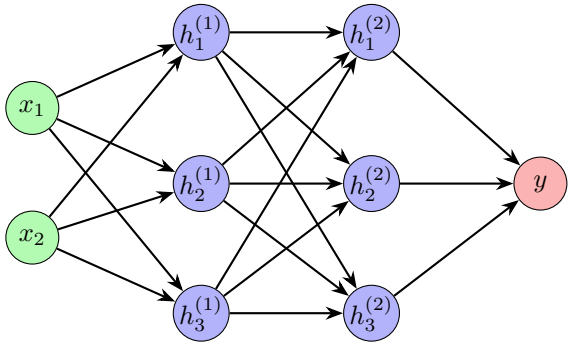


Figure 4.1: Multi-layer FNN with two hidden layers, 3 neurons in each, two neurons in the input layer and one neuron in the output layer

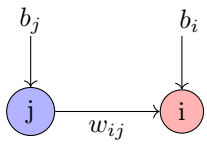


Figure 4.2: Connection from $j-$th neuron to the $i-$th neuron with the weight w_{ij} and the biases of each neuron

Feedforward neural networks can be conceptualized as multi-layer graphs with one input layer, one output layer, and one (single-layered FNN) or many (multi-layer FNN) hidden layers in between (see Figure 4.1). The inputted information (input vector) is propagated through each layer until it reaches the output layer [Saz06].

Each hidden layer has neurons in it, which are connected to the neurons in the previous and next hidden layers (or in the case of the first and last hidden layer to the input and output layer), and each connection has its weight. Given the input vector ($(x_1, x_2)^T$ in the Figure 4.1), FNN will give us the output (y in the Figure 4.1) based on three key components: weight matrix, bias vector, and activation function.

The weight matrix $\mathcal{W}_k$ represents the weighted connections of neurons in two neighboring layers $k-1$ and k. The entry w_{ij} in $\mathcal{W}_k$ describes the weight of the connection from $j-$th to $i-$th neuron (see Figure 4.2).

The second component is the bias vector. We denote $\mathbf{b^k}$ to be the bias vector of the $k-th$ layer (see Figure 4.2).

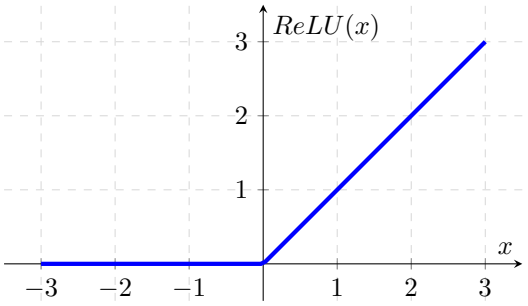


Figure 4.3: ReLU (rectified linear unit)

The third component is the activation function, which is chosen to transmit non-linearity to the neural network. It enables the model to learn and represent complex and sometimes non-interpretable relations between the variables. While various activation functions are employed in practice (such as tanh, sigmoid, softmax, etc.), in this thesis, we are only going to focus on $ReLU$ (rectified linear unit) defined as $ReLU(x) = max(0, x)$ (see Figure 4.3).

Now let us connect all three components, and see, how the output of the $i-th$ neuron (y_i) is calculated given the $n-$dimensional input vector ($\mathbf{x}$). First, we take $\mathcal{W}_k$ with the corresponding column and compute the weighted sum of the input vector. Then, we add the bias value (b_i) and in the last step apply the activation function ($ReLU$):

$$y_i = ReLU(b_i + \sum_{j=1}^{n} w_{ij} x_j)$$

This calculation is done for all neurons and the output becomes the input for the neurons in the next hidden layer (or for the output layer).

4.2 Reachability analysis of FNNs

In general, the goal of the reachability analysis of FNN is to determine the final state (or the final set of states) of the network, given a fixed set of input states [LM17]. Specifically, the aim is to ascertain whether the reachable set of the FNN is safe. Safe means, that the FNN should fulfill some predefined safety specifications. This process, known as safety verification of FNNs [HKWW17], involves assessing whether the FNN behaves safely under all circumstances.

In this thesis, we will use the generalized star set representation introduced in Section 2.2. Leveraging the generalized star representation offers several advantages, including scalability, efficiency, and high accuracy. Additionally, it possesses the capability to generate tangible counterexamples, a feature typically unavailable in alternative reachability approaches [BD17].

Before moving any further, let us formally define the reachable set of FNN.

Definition 4.2.1 (Reachable set of FNN)**.** *Given a bounded convex polyhedron input set defined as* $\mathscr{I} \triangleq \{\mathbf{x} \in \mathbb{R}^n \mid \mathcal{A}\mathbf{x} \leq \mathbf{b}\}$ *with* $\mathcal{A} \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^m$*, and an k-layers feed-forward neural network* $F \triangleq \{L_1, L_2, \cdots, L_k\}$ *with ReLU activation function, the reachable set* $F(\mathscr{I}) = \mathscr{R}_{L_k}$ *of the FNN F corresponding to the input set* $\mathscr{I}$ *is defined incrementally by:*

$$
\begin{aligned}
\mathscr{R}_{L_1} &\triangleq \{\mathbf{y_1} \mid \mathbf{y_1} = ReLU(\mathcal{W}_1\mathbf{x} + \mathbf{b_1}),\ \mathbf{x} \in \mathscr{I}\},\\
\mathscr{R}_{L_2} &\triangleq \{\mathbf{y_2} \mid \mathbf{y_2} = ReLU(\mathcal{W}_2\mathbf{y_1} + \mathbf{b_2}),\ \mathbf{y_1} \in \mathscr{R}_{L_1}\},\\
&\vdots\\
\mathscr{R}_{L_k} &\triangleq \{\mathbf{y_k} \mid \mathbf{y_k} = ReLU(\mathcal{W}_k\mathbf{y_{k-1}} + \mathbf{b_k}),\ \mathbf{y_{k-1}} \in \mathscr{R}_{L_{k-1}}\},
\end{aligned}
$$

where $\mathcal{W}_k$ *and* $\mathbf{b_k}$ *are the weight matrix and bias vector of the* k^{th} *layer* L_k*, respectively. The reachable set* $\mathscr{R}_{L_k}$ *contains all outputs of the FNN corresponding to all input vectors* $\mathbf{x}$ *in the input set* $\mathscr{I}$*. The ReLU function is applied dimensionwise (on each neuron) on each vector* $\mathcal{W}_i\mathbf{y}_i + \mathbf{b}_i$*.*

Definition 4.2.2 (Safety verification of FNN)**.** *Given a k-layers feedforward neural network F, and a safety specification* $\mathscr{S}$ *defined as a set of linear constraints on the neural network outputs* $\mathscr{S} \triangleq \{\mathbf{y}_k \mid \mathcal{C}\mathbf{y}_k \leq d\}$*, the neural network F is called to be safe corresponding to the input set* $\mathscr{I}$*, we write* $F(\mathscr{I}) \models \mathscr{S}$*, if and only if* $R_{L_k} \cap \neg\mathscr{S} = \emptyset$*, where* R_{L_k} *is the reachable set of the neural network with the input set* $\mathscr{I}$*, and* $\neg$ *is the symbol for logical negation. Otherwise, the neural network is called to be unsafe* $F(\mathscr{I}) \not\models \mathscr{S}$*.*

4.3 Methods to solve the reachability problem of FNNs

In this Section, we will introduce two algorithms proposed in [TMLM+19] for solving the reachability problem for FNNs with the *ReLU* activation function.

The first approach is the exact and thus, complete analysis. This algorithm will be discussed to provide insight into the motivation behind the second algorithm. The second method, termed over-approximate analysis, will be the central focus of this thesis. However, our method for reducing the star set's dimension can also be applied to the exact analysis.

4.3.1 Exact and Complete Analysis

As we have shown in Proposition 2.2.1, any bounded convex polyhedron can be represented as a star. Hence, we can assume that the input $\mathscr{I}$ of the FNN is a star set. According to the Definition 4.2.1, the reachable set of the first layer of the FNN $\mathscr{R}_{L_1}$ is computed by applying operations on the input set $\mathscr{I}$. The multiplication of $\mathcal{W}$ and the star set $\mathscr{I}$, along with the addition of the bias vector $\mathbf{b_1}$ is an affine transformation of the star set. The *ReLU* operation on a star set involves intersecting the star set with half-spaces, corresponding to the two cases of the *ReLU* function (projection for the negative case and identity for the positive case). Thus, according to the Propositions 2.2.2 and 2.2.3 $\mathscr{R}_{L_1}$ (which is a union of star sets due to the case

splittings) is also a star set. According to the Definition 4.2.1, the reachable set of the FNN is defined incrementally, layer-by-layer. While we are using the same operations (affine mapping, intersection with half-spaces) for all layers, having $\mathscr{R}_{L_1}$ as a star implies that all other reachable sets of the FNN are also stars.

For a layer L with n neurons in it, the reachable set of this layer is computed by doing a sequence of n chained *exactStepReLU* operations:

$$\mathscr{R}_L = ReLU_n(ReLU_{n-1}(\cdots(ReLU_1(\Theta)))),$$

with $\Theta = \langle \mathbf{c}, \mathcal{V}, P \rangle$ being the input star set to the FNN. Intuitively, this means that each neuron is responsible for the *exactStepReLU* operation on one dimension of the star set. The $i-$th dimension of the star set captures the reachable states (values) of the $i-$th neuron in the layer.

The *exactStepReLU* operation on the $i-$th neuron in the layer, i.e., $exactStepReLU_i(\cdot)$, works as follows. First the input star set $\Theta = \langle \mathbf{c}, \mathcal{V}, P \rangle$ is divided into two subsets: $\Theta = \Theta_1 \cup \Theta_2$, such that $\Theta_1 = \Theta \wedge x_i \geq 0$ and $\Theta_2 = \Theta \wedge x_i < 0$. As Θ_1 and Θ_2 are the results of the intersection of a star set with a half-space, according to the Proposition 2.2.3, Θ_1 and Θ_2 are also star sets. Let us assume Θ_1 and Θ_2 are as follows: $\Theta_1 = \langle \mathbf{c}, \mathcal{V}, P_1 \rangle$ and $\Theta_2 = \langle \mathbf{c}, \mathcal{V}, P_2 \rangle$. This step was a preparation to apply the *exactStepReLU* on the x_i. Since, $x_i < 0$ for Θ_2, applying *ReLU* on the x_i of the vector $\mathbf{x} = [x_1, \cdots, x_{i-1}, x_i, x_{i+1}, \cdots, x_n]^T$ results a new vector $\mathbf{x}' = [x_1, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_n]^T$. This procedure is the same, as mapping the star set Θ_2 by the mapping matrix $\mathcal{M} = [\mathbf{e_1}, \cdots, \mathbf{e_{i-1}}, \mathbf{0}, \mathbf{e_{i+1}}, \cdots, \mathbf{e_n}]$: $\mathbf{x}' = \mathcal{M}\mathbf{x}$. On the other hand, applying *ReLU* on the x_i of the vector $\mathbf{x} \in \Theta_1$ does not change anything (since $x_i \geq 0$ and $ReLU(x_i) = x_i$).

The result of the *exactStepReLU* operation on the $i-$th neuron in a layer L is the union of two star sets: $ReLU_i(\Theta) = \langle \mathbf{c}, \mathcal{V}, P_1 \rangle \cup \langle \mathcal{M}\mathbf{c}, \mathcal{M}\mathcal{V}, P_2 \rangle$.

From the description above, we can discern three cases regarding the ranges of the states in one arbitrary dimension of the input star set:

1. The lower bound is greater than zero,

2. The upper bound is less than zero,

3. The lower bound is smaller than zero, and the upper bound is greater than zero.

The first two cases imply that there is no need to divide the star set into two subsets, which is advantageous as it avoids increasing the number of star sets. This means that knowing the ranges of all states will help boost the efficiency of the exact analysis. We only increase the number of star sets by dividing the input star set into two subsets only when the range of the values of the neuron lies on both sides of zero. Another optimization strategy might involve checking the inclusion of star sets after each *exactStepReLU*. However, this approach was found to be NP-complete [ST19] and therefore, simply keeping both star sets as they are and later treating them as separate star sets is more beneficial [ST19].

Although the exact analysis provides a concrete description of the neural network's behavior, the exponentially increasing number of star sets poses a serious limitation. This makes the exact analysis inefficiently applicable to large neural networks with many hidden layers, limiting its scalability.

Algorithm 2 Star-based over-approximate reachability analysis for one layer

Input: $\mathscr{I} = \Theta = \langle \mathbf{c}, \mathcal{V}, P \rangle, \mathcal{W}, \mathbf{b}$ ▷ input star set, weight matrix, bias vector
Output: $\mathscr{R}$ ▷ over-approximated reachable set

procedure $\mathscr{R}$ =ApproxLayerReach($\mathscr{I}, \mathcal{W}, \mathbf{b}$)
  $\mathscr{I}_1 = \mathcal{W} * \Theta + \mathbf{b} = \langle \mathcal{W}\mathbf{c} + \mathbf{b}, \mathcal{W}\mathcal{V}, P \rangle$
  $\mathscr{I}_n = \mathscr{I}_1$
  for $i = 1 : n$ **do** ▷ n is the number of neurons of the layer
    $\mathscr{I}_n = ApproxStepReLU(\mathscr{I}_n, i)$ ▷ i^{th} approximate-stepReLU operation
  end for
  $\mathscr{R} = \mathscr{I}_n$
end procedure
procedure ApproxStepReLU($\tilde{\mathscr{I}}, i$)
  $\tilde{\mathscr{I}} = \tilde{\Theta} = \langle \tilde{\mathbf{c}}, \tilde{\mathcal{V}}, \tilde{P} \rangle$
  $[l_i, u_i] = \tilde{\Theta}.getRange(i)$ ▷ range of $x[i]$, i.e., $l_i \leq x[i] \leq u_i$
  $\mathcal{M} = \begin{bmatrix} \mathbf{e_1} & \mathbf{e_2} & \cdots & \mathbf{e_{i-1}} & \mathbf{0} & \mathbf{e_{i+1}} & \cdots & \mathbf{e_n} \end{bmatrix}$
  if $l_i \geq 0$ **then** $\tilde{\mathscr{R}} = \tilde{\Theta} = \langle \tilde{\mathbf{c}}, \tilde{\mathcal{V}}, \tilde{P} \rangle$
  if $u_i \leq 0$ **then** $\tilde{\mathscr{R}} = \mathcal{M} * \tilde{\Theta} = \langle \mathcal{M}\tilde{\mathbf{c}}, \mathcal{M}\tilde{\mathcal{V}}, \tilde{P} \rangle$
  if $l_i < 0$ and $u_i > 0$ **then**
    $\tilde{P}(\boldsymbol{\alpha}) \triangleq \tilde{C}\boldsymbol{\alpha} \leq \tilde{\mathbf{d}}, \boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_m]^T$ ▷ input set's predicate with p constraints
    $\boldsymbol{\alpha'} = [\alpha_1, \ldots, \alpha_m, \alpha_{m+1}]^T$ ▷ new variable α_{m+1}
    $\mathbf{C_1} = \begin{bmatrix} 0 & \cdots & 0 & -1 \end{bmatrix} \in \mathbb{R}^{1 \times m+1}, d_1 = 0$ ▷ $\alpha_{m+1} \geq 0 \Leftrightarrow C_1\alpha' \leq d_1$
    $\mathbf{C_2} = \begin{bmatrix} \tilde{\mathcal{V}}[i:] & -1 \end{bmatrix} \in \mathbb{R}^{1 \times m+1}, d_2 = -\tilde{c}[i]$ ▷ $\alpha_{m+1} \geq x[i] \Leftrightarrow C_2\alpha' \leq d_2$
    $\mathbf{C_3} = \begin{bmatrix} \frac{-u_i}{u_i - l_i} \times \tilde{\mathcal{V}}[i:] & 1 \end{bmatrix}, d_3 = \frac{u_i}{u_i - l_i}(\tilde{c}_i - l_i)$ ▷ $\alpha_{m+1} \leq \frac{u_i(x[i]-l_i)}{u_i - l_i} \Leftrightarrow C_3\alpha' \leq d_3$
    $\mathbf{C_0} = \begin{bmatrix} \tilde{C} & \mathbf{0_{p \times 1}} \end{bmatrix}, \mathbf{d_0} = \tilde{\mathbf{d}}$
    $\mathcal{C}' = [\mathbf{C_0};\ \mathbf{C_1};\ \mathbf{C_2};\ \mathbf{C_3}], \quad \mathbf{d}' = [d_0;\ d_1;\ d_2;\ d_3]$
    $P'(\boldsymbol{\alpha'}) \triangleq \mathcal{C}'\boldsymbol{\alpha'} \leq \mathbf{d}'$ ▷ output set's predicate
    $\mathbf{c}' = \mathcal{M}\tilde{\mathbf{c}}, \ \mathcal{V}' = \mathcal{M}\tilde{\mathcal{V}}, \ \mathcal{V}' = [\mathcal{V}'\ \mathbf{e_i}]$ ▷ $y[i] = ReLU(x[i]) = \alpha_{m+1}$
    $\tilde{\mathscr{R}} = \langle \mathbf{c}', \mathcal{V}', P' \rangle$
  end if
end procedure

4.3.2 Over-approximation Analysis

The over-approximate reachability analysis (presented in Algorithm 2) is an alternative to the exact analysis. Although the exact analysis provides a complete view of the reachable sets of an FNN, the downside of it is the exponentially increasing number of star sets making its application on large FNNs computationally unfavorable. The primary benefit of the over-approximate analysis is that we will have to track only one single star set through the whole reachability analysis.

The need to separate the star set into two parts in exact analysis occurs when we would get a non-convex set as the result of the *ReLU*. That happens if the lower bound of a neuron is less than zero and the upper bound is greater than zero. In such scenarios, the exact and over-approximate analyses differ.

For any input x_i the output of $ReLU(x_i) = y_i$ in over-approximate analysis is:

$$\begin{cases} y_i = x_i, & \text{if } l_i \geq 0. \\ y_i = 0, & \text{if } u_i \leq 0. \\ y_i \geq 0, y_i \leq \frac{u_i(x_i - l_i)}{u_i - l_i}, y_i \geq x_i & \text{if } l_i \leq 0 \text{ and } u_i \geq 0 \end{cases} \tag{4.1}$$

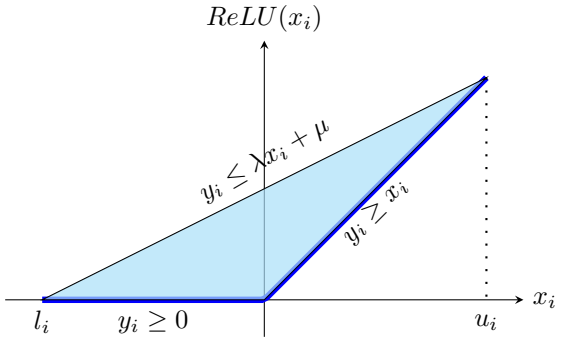


Figure 4.4: Over-approximation of ReLU neuron with $\lambda = \dfrac{u_i}{u_i - l_i}$ and $\mu = -\dfrac{l_i u_i}{x_i - l_i}$ (Redrawn from [TMLM+19])

Figure 4.4 shows the values for x_i in the third case that are restricted in the depicted triangle, which is a convex over-approximation of the $ReLU$ function on the domain $[l_i, u_i]$.

In over-approximate reachability analysis we get the values for $ReLU(x_i)$ with some over-approximation error in the third case. The dark blue lines describe the exact set of the $ReLU$ and the light blue shaded triangle is the over-approximated set. Although this makes the analysis incomplete, it significantly enhances the scalability of the over-approximate analysis for large FNNs compared to the exact analysis.

The over-approximation of a neuron is achieved by adding one new variable (α_{m+1}) to the predicate of the star set (line 17 in the Algorithm 2) and three new inequalities in the predicate (lines $18-20$ in Algorithm 2). These inequalities constrain the newly added variable in the generator to be inside the triangle depicted above. The center and the generator are also correspondingly changed (line 24 in the Algorithm 2).

4.4 Application in the reachability of neural networks

In this section, we will explore how our method can be applied during computing the reachable set of a neural network using the over-approximation method introduced in Section 4.3.2.

As presented in Section 3.3, the condition for the dimension reduction method to work is that the rank of the generator matrix $\mathcal{V}$ must be less than the number of its columns. This condition needs to be checked for the reachable set of each layer. Therefore, for a k-layer FNN, we will perform k checks at the end of each layer and at most k dimension reduction steps during the analysis.

Real-life FNNs are typically much larger than the exemplary FNNs presented in Examples 3.2.1 and 3.4.1. For example, a well-known benchmark like the Airborne

Collision Avoidance System (ACAS) Xu has five input and five output neurons, and six hidden layers with 50 neurons each [LJB+22]. For a star set to represent the reachable set of for example first layer, this means having a generator matrix of size $(50, 5 + j)$, where 5 is the number of original variables and j is the number of over-approximated neurons in the layer.

Dealing with matrices of this size might cause computational overload. That is why a non-naive implementation of the dimension reduction method is necessary. In this section, we will discuss the tricks and improvements made in the implementation to address this challenge.

The structure of the generator matrix $\mathcal{V}$ after one layer can be analyzed to understand its possible configuration. Assume, we had a star set of dimension (n, m) as the input to the layer (meaning that for the generator matrix $\bar{\mathcal{V}}$ of the input star set: $\bar{\mathcal{V}} \in \mathbb{R}^{n \times m}$), and during the layer, we did j over-approximations (case 3 in Formula 4.1), i projections of the negatives (case 2 in Formula 4.1), and $n - i - j$ identity mappings (case 1 in Formula 4.1).

Reordering the rows of the generator matrix yields the following structure, where $\mathcal{I}_j$ corresponds to the identity matrix of size j (Definition 0.0.6) and the $\mathcal{V}_{\text{slice}} \in \mathbb{R}^{(n-j-i) \times m}$ corresponds to the matrix containing some rows of the original $\bar{\mathcal{V}}$ which was the generator matrix of the current layer's input star set:

$$\mathcal{V} = \left[\begin{array}{c|cc} 0 & \mathcal{I}_j & \\ \hline \mathcal{V}_{\text{slice}} & 0 & \\ \hline 0 & 0 & \end{array}\right] \in \mathbb{R}^{n \times (m+j)}$$

The presence of the identity matrix in the generator matrix corresponds j over-approximation steps (the first j rows), the 0-rows correspond to the projection of the negatives (the last i rows), and in the middle part are the $n-i-j$ identity mappings, preserving the original rows from the generator matrix before the $ReLU$ step.

For the first step, we are only interested in the rank of $\mathcal{V}$, and in particular, if $\text{rank}(\mathcal{V}) < m + j$ holds. The last i rows can not contribute to the rank (the rank of that block is zero), so we can ignore them for this step. The rank of the $\mathcal{I}_j$ is fixed and is equal to j. Because of the 0 blocks in $\mathcal{V}$, the number of independent columns in $\mathcal{V}$ is implied by:

1. the number of independent columns in $\mathcal{V}_{\text{slice}}$ ($\text{rank}(\mathcal{V}_{\text{slice}})$),
2. the number of independent columns in the identity matrix $\mathcal{I}_j$ ($\text{rank}(\mathcal{I}_j)$, which is j).

Therefore,

$$\text{rank}(\mathcal{V}) = j + \text{rank}(\mathcal{V}_{\text{slice}}).$$

To sum up, instead of considering the whole matrix $\mathcal{V}$ and checking $\text{rank}(\mathcal{V}) < m + j$, we can just take $\mathcal{V}_{\text{slice}}$, and check if $\text{rank}(\mathcal{V}_{\text{slice}}) < m$. If this does not hold, then the dimension reduction method does not apply to this star set. Otherwise, we continue, and based on the Algorithm 1 we do the steps described in lines $4-10$ while replacing each $\mathcal{V}$ in the algorithm with $\mathcal{V}_{\text{slice}}$. Now we have a transformation matrix $\mathcal{G}^* \in \mathbb{R}^{m \times m}$ that ensures

$$\mathcal{V}_{\text{slice}}\mathcal{G}^* = \mathcal{V}'_{\text{slice}},$$

where $\mathcal{V}'_{\text{slice}}$ is the matrix that has exactly $\text{rank}(\mathcal{V}_{\text{slice}})$ non-zero and $m-\text{rank}(\mathcal{V}_{\text{slice}})$ 0-columns. This is not the same transformation matrix $\mathcal{G} \in \mathbb{R}^{(m+j)\times(m+j)}$, which we would have obtained if we would consider the whole $\mathcal{V}$. This means, now we need to construct back $\mathcal{G}$ having $\mathcal{G}^*$. The dimensions of $\mathcal{G}$ and $\mathcal{G}^*$ show that we are missing only the j over-approximation columns. This means, that $\mathcal{G}$ is constructed by appending unit vectors from the $(m+1)-th$ to $(m+j)-th$ position:

$$\mathcal{G} = \left[\begin{array}{c|c} \mathcal{G}^* & 0 \\ \hline 0 & \mathcal{I}_j \end{array}\right]$$

Now let us prove that this is the right transformation matrix $\mathcal{G}$. Using the block structure of the matrices in the matrix multiplication, we obtain:

$$\mathcal{V}\mathcal{G} = \left[\begin{array}{c|c} 0 & \mathcal{I}_j \\ \hline \mathcal{V}_{\text{slice}} & 0 \\ \hline 0 & 0 \end{array}\right] \left[\begin{array}{c|c} \mathcal{G}^* & 0 \\ \hline 0 & \mathcal{I}_j \end{array}\right] = \left[\begin{array}{c|c} 0 \cdot \mathcal{G}^* + \mathcal{I}_j \cdot 0 & 0 \cdot 0 + \mathcal{I}_j \cdot \mathcal{I}_j \\ \mathcal{G}^* \cdot \mathcal{V}_{\text{slice}} + 0 \cdot 0 & \mathcal{V}_{\text{slice}} \cdot 0 + 0 \cdot \mathcal{I}_j \\ 0 \cdot \mathcal{G}^* + 0 \cdot 0 & 0 \cdot 0 + 0 \cdot \mathcal{I}_j \end{array}\right] =$$
$$\left[\begin{array}{c|c} 0 & \mathcal{I}_j \\ \hline \mathcal{V}'_{\text{slice}} & 0 \\ \hline 0 & 0 \end{array}\right]$$

Now when we have the right transformation matrix $\mathcal{G}$, we can compute $\mathcal{C}\mathcal{G} = \mathcal{C}_{FM}$ and we are ready to apply the FM elimination. After applying the FM elimination, we get the new ${\mathcal{C}'}_{FM}$ and $\mathbf{d}'$. We erase the 0-columns from ${\mathcal{C}'}_{FM}$ and get $\mathcal{C}'$. We can also apply the optional step of removing the redundant constraints and get a final star set $\Theta' = \langle \mathbf{c}, \mathcal{V}', P' \rangle, P' \triangleq \mathcal{C}'\boldsymbol{\alpha}' \leq \mathbf{d}'$. This is the reachable set of the current layer with reduced dimension, which serves as input for the next layer.

Chapter 5

Experimental results

In this section, we show and analyze the performance of the dimension reduction algorithm and compare two variable elimination algorithms. As benchmarks, we used the ACAS Xu[KBD+17], Drones[DGPT24], and Thermostat[Jia23] benchmark sets.

We implemented the dimension reduction algorithm in the open-source C++ tool HyPro[1] [SÁMK17] using the Eigen[2] [GJ+10] library for linear algebra. We evaluate our algorithm only during the over-approximative analysis for computing the reachability set of an FNN. The whole analysis is executed using exact number representations (rational numbers represented by the *mpq_ class* of the GNU MP library). The experiments were done using a machine with Ubuntu 22.04.3 LTS operating system, 12GB RAM, 8 cores, and Intel® Core™ i7-1065G7 CPU @ 1.30GHz × 8 processor.

During our evaluation of the dimension reduction method on certain benchmarks, we encountered one issue, namely, applying the FM elimination causes an exponential blow-up in the number of constraints, thus calculating the result is computationally infeasible. In most cases, at the end of each layer, we have to eliminate more than one variable. For this, we would have to iteratively apply FM elimination until there are no variables to be eliminated. However, as shown in Section 2.3.1, the worst-case number of new constraints the algorithm generates for eliminating one variable is double exponential regarding the number of constraints in the original system. The exponential blow-up happens not only in worst-case but also, as our experiments show, that on average the number of newly generated constraints is too big even when only eliminating one variable. If we had applied the FM Elimination to eliminate more than one variable, then in most cases we would have ended up with a huge number of constraints, that are even hard to allocate in the memory.

Based on this problem, we used another variable elimination algorithm, which has the same idea as FM, but it allows us to eliminate more than one variable in one shot. The method is called FMplex [NPÁK23], which is an extension of the FM variable elimination. It uses simplex optimization[Dan63] while generating the constraints and in the implementation, it operates with sparse matrices[YWLG17] (instead of the dense matrices like our basic FM elimination implementation does).

Although FMplex is a faster algorithm and in total produced fewer constraints than the basic implementation of FM elimination, it still takes much more time than the whole analysis of an FNN with the redundant variables in the predicate and the

[1] https://github.com/hypro

[2] https://eigen.tuxfamily.org/index.php?title=Main_Page

generator matrix. For example, on one of the ACAS Xu networks, the elimination of only **three variables** can take up to 2.5 minutes ($N^3_{3,1}$ of Table 5.1). In comparison, the whole reachability analysis of the ACAS Xu benchmark with redundant variables in the predicate and the generator matrix lasts on average only 10.38 seconds (note that this result was achieved by a machine with an overall poorer performance) [Mas23]. Therefore, the dimension reduction method does not improve the overall running time of the whole analysis. For this reason, in this chapter, we are not going to focus on the running time, but on other aspects, such as the number of (redundant) constraints returned by FMplex, and in Chapter 6 we will discuss some suggestions, that might improve FMplex in the future.

5.1 ACAS Xu

To start demonstrating different statistics, we wanted to test, in how many cases the condition for our method (rank of the matrix less than the number of its columns) holds. For this, we examined the ACAS Xu networks for Property 3 [KBD+17] with the over-approximation method. At the end of each layer, we check whether the condition holds and how many variables can be eliminated. The results are summarized in Figure 5.1.

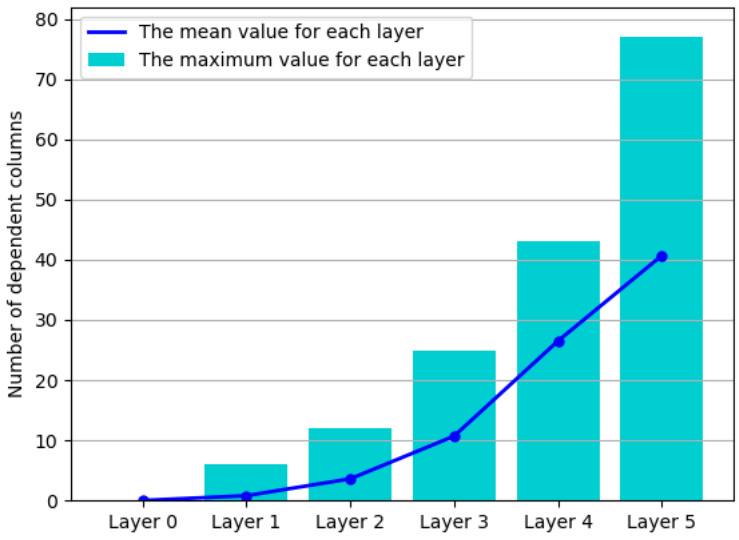


Figure 5.1: The arithmetic means and the maximum number of variables that can be eliminated after each layer when applying the over-approximation method on ACAS Xu benchmarks considering Property 3 (see appendix of [KBD+17]).

Let us discuss the results. As we can see, at the end of the first layer, there is no benchmark, for which the condition rank is smaller than the number of columns in the generator matrix holds. Already starting from the second layer, we observe that there are some benchmarks, for which we can apply the dimension reduction. The small

value of the means indicated that for most benchmarks, the condition still does not hold. The averages for the third and upcoming layers show that for most benchmarks the condition holds. As you can see, the number of variables that can be eliminated grows exponentially. The linear dependency that occurs at any layer (particularly at Layer 1 and Layer 2), will not disappear and even more, increases the chances of getting more dependencies in the upcoming layers. However, we assume that this number would be much lower if we had eliminated the linear dependent after that layer, where it first appears. We could not prove or disprove our thoughts because the number of constraints that we get after eliminating the variables after the first possible layer (even when using FMplex) is huge and it is impossible to continue the analysis with this number of constraints. In Table 5.1, the performance of FMplex on some benchmarks of ACAS Xu is summarized.

$N^p_{x,y}$	Row	Col	LinDep	Time(s)	Constraints
$N^3_{3,1}$	34	13	1	0.022	224
$N^4_{1,9}$	55	20	1	0.408	577
$N^3_{1,2}$	58	21	2	1.333	4797
$N^3_{5,5}$	91	32	2	37.860	16770
$N^3_{2,9}$	34	13	3	1.739	4385
$N^3_{4,1}$	67	24	3	533.361	58984
$N^3_{1,1}$	73	26	3	283	88891
$N^4_{4,7}$	88	31	3	T	-
$N^4_{4,4}$	49	18	4	55.418	54163
$N^3_{2,8}$	55	20	4	114.399	73761
$N^4_{3,7}$	52	19	4	240.006	82496
$N^3_{2,6}$	76	27	4	T	-
$N^4_{4,1}$	82	29	4	T	-
$N^3_{2,7}$	52	19	5	460.166	164203
$N^3_{2,5}$	61	22	5	T	-
$N^3_{2,2}$	64	23	5	T	-
$N^4_{3,4}$	70	25	5	T	-
$N^4_{1,5}$	85	30	5	T	-
$N^3_{3,9}$	34	13	6	103.906	33458
$N^4_{5,1}$	82	29	6	T	-
$N^3_{5,4}$	52	19	7	T	-
$N^4_{2,8}$	79	28	7	T	-

Table 5.1: The performance of FMplex on the ACAS Xu benchmark set. $N_{x,y}$ is the benchmark network and the p is the input property (3 or 4). The Row is the number of constraints that the predicate has originally, Col is the number of variables in the predicate, LinDep is the number of linearly dependent columns, and Constraints is the number of constraints returned after eliminating all linearly dependent columns. T means the method did not terminate within one hour.

As you can see, eliminating more than four variables is challenging for FMplex. When trying to eliminate five or more variables almost on all benchmarks FMplex did not terminate within one hour. However, on $N^3_{3,9}$ it could eliminate six variables in less than two minutes. The reason is the number of constraints and variables in

the original system. It has only 34 constraints over 13 variables. In comparison, the other benchmarks have at least 60 constraints over at least 22 variables, when trying to eliminate five or more variables. It is also interesting to state, that although $N^4_{5,4}$ has less than 60 constraints (52) and less than 22 variables (19), FMplex still did not terminate in one hour when trying to eliminate seven variables.

For all networks, regardless of how many variables were eliminated, we could not continue analyzing the reachable set of the FNN. Even with the smallest number of returned constraints (which happens in benchmark $N^3_{3,1}$), we encounter an error thrown by GLPK when trying to solve LP instances (namely simplex) on these constraint systems resulting from FMPlex: Assertion failed: $-DBL\_MAX <= val$ && $val <= +DBL\_MAX$. This error indicates that when trying to calculate the bounds of the neurons in the next layer using simplex (which has a certain range for the input), we encounter very large numerators and denominators in the fractions represented by mpq_class that exceed the maximum allowed limit for GLPK simplex. These large numbers occur because we are working with exact numbers and due to the precise calculations of FMplex, which require very large numerators and denominators to represent values with high precision. The solution to this problem could be that after each layer we normalize all values. This would also be a costly operation, however, it will allow us to calculate the reachable set of at least the next layer.

The returned number of constraints is also an interesting topic. As we can observe, the number of constraints FMplex returns is exponential regarding the number of original constraints. It is better than the one returned by the basic FM elimination, which is double exponential. However, dealing with exponentially more constraints is highly inefficient. Moving back to our dimension reduction Algorithm 1, we see that at the end we could also remove the redundant constraints and end up with less number of constraints. We tried to apply one of the best linear programming solvers: GLPK [GAD+13]. However, for the ACAS Xu benchmark, detecting which inequalities are redundant and removing them was impossible. We faced again the same error: Assertion failed: $-DBL\_MAX <= val$ && $val <= +DBL\_MAX$. The reason is, that for removing the redundant constraints also the GLPK simplex is invoked, which raises the same error as we had when trying to calculate the bounds of neurons at the next layer. However, we also tried to detect and remove redundancies while computing the reachability with floating points (doubles). Unfortunately, in this case, the program did not terminate in a reasonable time.

The encountered problems above show that we are in a deadlock even if assuming that the normalization of the values after each layer is done: we cannot continue the analysis of the FNN because the number of constraints is so big that we can neither allocate the constraint matrix (bad alloc error, discussed in the next paragraphs) nor remove the redundant constraint.

This means, with the current implementation of FMplex it is impossible to apply the dimension reduction method on the ACAS Xu benchmark.

5.2 Drones

Another benchmark, that we used is the Drones benchmark. This benchmark has eight trained neural networks (denoted AC1, ..., AC8, Table 5.2).

However, the number of layers becomes irrelevant as we could not proceed beyond

Architecture	Network ID	Neurons
Two layers	AC1	32, 16
	AC2	64, 32
	AC3	128, 64
	AC4	256, 128
Three layers	AC5	32, 16, 8
	AC6	64, 32, 16
	AC7	128, 64, 32
	AC8	256, 128, 64

Table 5.2: Drones benchmark networks architecture, reused from [Mas23]

the first layer where our condition was met. Either the number of returned constraints after the dimension reduction method was too large, rendering further analysis impossible (both with FMplex and basic FM), or the program failed to terminate. However, we can present the results of the eliminations where it terminated.

For each neural network, we have two safety properties (denoted $AC1_1$ and $AC1_2$ when applied on the first neural network), that we can check (two different inputs). In total, we have 16 network and input property pairs, but because the computation of the reachable set of two pairs ($AC4_1$ and $AC8_1$) could not be finished within two hours, those two pairs will not analyzed. Note, that the dimension reduction method was not even applied on these pairs. On the pair $AC3_2$ the method is not applicable because the condition never was met. We left out these three networks and input property pairs and compared the behavior of FMplex and the basic FM implementation on the remaining 13 benchmarks. In Table 5.3 we summarized the number of returned constraints and in Tables 5.4a and 5.4b the running time of the methods.

The findings reveal that FMplex successfully solved only four out of 13 instances. In this benchmark, the number of eliminated variables peaked at 6 within one hour. For any number of variables greater than six, FMplex did not terminate in one hour. However, the number of already existing constraints also matters. For instance, while both $AC1_1$ and $AC6_2$ benchmarks required the elimination of six variables, $AC1_1$ entailed double the number of constraints compared to $AC6_2$. Consequently, FMplex managed to tackle the latter but not the former benchmark. Comparing FMplex with basic FM we can say, that Fmplex remarkably outperformed FM. FM could only eliminate at most two variables, which is way beyond the number of variables that should be eliminated. Because of this problem, we decided to work with smaller benchmarks, to be able at least to finish the FNN reachable set analysis by applying the dimension reduction method at the end of each layer.

	Row	Col	LinDep	FMplex	FM1	FM2	FM3
$AC1_1$	69	27	6	T	509	53869	Killed(BA)
$AC1_2$	42	18	11	T	131	2920	Killed(R)
$AC2_1$	105	39	29	T	1004	221277	Killed(BA)
$AC2_2$	33	15	5	20486	153	4688	Killed(R)
$AC3_1$	231	81	51	T	5336	Killed (BA)	-
$AC4_2$	57	23	4	214859	608	86147	Killed(BA)
$AC5_1$	81	31	9	T	564	64572	Killed(BA)
$AC5_2$	30	14	4	4268	129	3153	Killed(R)
$AC6_1$	105	39	29	T	1034	232321	Killed(BA)
$AC6_2$	30	14	6	18932	109	2215	Killed(R)
$AC7_1$	159	57	43	T	2461	Killed(R)	-
$AC7_2$	42	18	9	T	161	4840	Killed(R)
$AC8_2$	49	19	13	T	134	2939	Killed(R)

Table 5.3: The comparison of the performance of both methods, FMplex and basic FM elimination. Row is the number of constraints that the predicate has originally, Col is the number of variables in the predicate, LinDep is the number of linearly dependent columns, under FMplex is the number of constraints after eliminating all linearly dependent columns, under FM1, FM2, and FM3 are the numbers of constraints when trying to iteratively eliminate first, second and third using basic FM. Killed(BA) indicates that we have a bad allocation error, Killed(R) indicates that we ran out of RAM during the computation and T means the method timed out.

	FM1	FM2	FM3
$AC1_1$	0.014	1.886	Killed
$AC1_2$	0.004	0.136	Killed
$AC2_1$	0.030	13.719	Killed
$AC2_2$	0.006	0.231	Killed
$AC3_1$	0.406	Killed	-
$AC4_2$	0.044	8.049	Killed
$AC5_1$	0.015	1.776	Killed
$AC5_2$	0.002	0.065	Killed
$AC6_1$	0.040	15.371	Killed
$AC6_2$	0.003	0.097	Killed
$AC7_1$	0.090	Killed	-
$AC7_2$	0.006	0.266	Killed
$AC8_2$	0.003	0.111	Killed

(a) Time in milliseconds to eliminate one and two variables by iteratively applying FM. The reasons for Killed are the same as presented in the Table 5.3 above

	linDep	FMplex
$AC1_1$	6	T
$AC1_2$	11	T
$AC2_1$	29	T
$AC2_2$	5	42
$AC3_1$	51	T
$AC4_2$	4	408
$AC5_1$	9	T
$AC5_2$	4	7
$AC6_1$	29	T
$AC6_2$	6	76
$AC7_1$	43	T
$AC7_2$	9	T
$AC8_2$	13	T

(b) Time in seconds to eliminate linDep variables using FMplex. T means the method timed out.

Table 5.4: Time comparison of two variable elimination techniques on Drones benchmark.

5.3 Thermostat

The next benchmark that we worked with is the Thermostat benchmark. The neural network of this benchmark consists of two input neurons, ten neurons in each of two hidden layers, and one output neuron. The purpose of focusing on this benchmark is to analyze the number of redundant constraints returned by FMplex. Therefore, we will leave the time measurements for the thermostat benchmark.

Although the benchmark is small, when eliminating two or more variables, we could not find the number of redundancies. Even when eliminating one variable we can see, that it is not always the case that the number of redundant constraints could be calculated (benchmark ID 13). We got again the Assertion error: $-DBL\_MAX <= val \;\&\&\; val <= +DBL\_MAX$. This shows, that the values go beyond the ranges that GLPK can handle. For example, on benchmark 13, we get the number that has 309 digits, which is greater than $DBL\_MAX = 1.7976931348623158e + 308$. In this case, neither the analysis can be continued, nor the redundant constraints can be removed. The reason is that in both cases simplex is invoked, which cannot be executed because of the assertion error. As in the case of elimination of more than one variable we do lots of division and multiplication operations, and the values blow up very quickly. This is why when eliminating more than one variable, on none of the benchmarks the redundancy check could be done. One solution to this problem could be to divide the nominator and denominator of the fraction by the GCD (greatest common divisor). It might only not help in general, but still is an improvement, that can be applied in some cases. There is also another solution, which is more complex and requires a proper implementation. As during the over-approximation step the values of the coefficients in the three new inequalities (lines $17-19$ in Algorithm 2) in the constraint matrix $\mathcal{C}$ are implied by the bounds of the neurons, we can modify those bounds in a way, that the resulting constraints are easier and computationally more beneficial to represent with exact representation. This can be done by rounding down (or just decreasing) the lower bounds and rounding up (or just increasing) the upper bounds. In this way, we over-approximate the values even more, but the analysis remains sound.

Focusing on the cases where the number of redundant constraints could be calculated we can say, that on this benchmark, the number of generated redundant constraints by FMplex is approximately 34%. This number may be way different than what we would expect to see in general cases. The main reason is that 34% that we got, is only based on the cases where only one variable is eliminated. The results of applying the dimension method on this benchmark are summarized in Table 5.5 below.

ID	Row	Col	LinDep	Constraints	Redundant
1	4	2	1	2	0
2	7	3	1	8	2
3	10	4	1	16	4
4	13	5	1	26	5
5	13	5	1	26	6
6	10	4	1	16	10
7	19	7	1	52	18
8	29	9	1	86	20
9	31	11	1	128	27
10	31	11	1	128	29
11	31	11	1	128	30
12	31	11	1	128	101
13	31	11	1	128	-
14	16	6	2	150	-
15	19	7	2	205	-
16	22	8	2	291	-
17	22	8	3	791	-
18	69	9	4	677463	-
19	28	10	4	4372	-
20	22	8	4	1797	-
21	25	9	4	2871	-
22	28	10	5	8987	-
23	31	11	5	14415	-
24	39	14	7	134571	-

Table 5.5: The performance of FMplex on the Thermostat benchmark. Row is the number of constraints that the predicate has originally, Col is the number of variables in the predicate, LinDep is the number of linearly dependent columns, Constraints is the number of constraints returned after eliminating all linearly dependent columns, Redundant is the number of redundant constraints detected by GLPK solver. - indicates that we got an error: Assertion failed: $-DBL\_MAX <= val \;\&\&\; val <= +DBL\_MAX$.

Chapter 6

Conclusion

6.1 Summary

In Section 3.4, we introduced a method for star set dimension reduction where we decrease the number of variables in the predicate using Forier-Motzkin(FM) variable elimination. For this, in Section 3.2 we showed what form must the star set have to apply the FM elimination. In Section 3.3 we proved under which condition and how any given star set can be transformed into an equivalent form like in Section 3.2. Moreover, in Chapter 5, we showed how often this condition is met on a real-life benchmark - ACAS Xu.

Furthermore, we implemented our method in Hypro - a C++ state set representation library, using the Eigen library for our calculations. Finally, we applied our method for dimension reduction on the state set over-approximate reachability analysis of feedforward neural networks. For this, we extended the implementation of our method to be more efficient while applying it in the reachability analysis of FNNs, taking into account the specific aspects of the over-approximate analysis. We chose three benchmarks to measure the runtime, and number of returned constraints and tried to determine how many of them are redundant. We also compared two variable elimination techniques mentioned above by integrating them into our dimension reduction method.

6.2 Discussion and Future work

Although our method successfully reduces the number of variables in the star set representation without any information loss, it does not necessarily reduce its dimension in the first parameter: the number of constraints. Neither FMplex nor FM could **efficiently** eliminate the desired variables. Moreover, in some cases, the elimination process could not be terminated within a fixed running time. In other cases, the coefficients in the returned system exceeded the threshold of manageable values for further analysis. We suppose this is because of the complex operations done by FMplex internally for dealing with a large number of constraints. These numbers got so big that it was also impossible to detect and remove the redundant constraints.

The problems mentioned above hindered us from getting even one layer further after applying the dimension reduction algorithm (for the realistic benchmarks, such as

Drones and ACAS Xu). As the results after the elimination are already out of control, we have two suggestions that can help to improve the performance if implemented **during** the FMplex procedure:

1. Bind the numbers within certain ranges.

   We have observed that once the numbers exceed a certain threshold, it becomes increasingly challenging (sometimes impossible) to efficiently bind them back. We attempted to apply the corresponding method in HyPro to bind the values of the returned constraints, and we did not succeed. Already before the binding procedure, the numbers exceeded the DBL_MAX threshold. After this, it is impossible to bind them back.

2. Prevent the redundant constraint to appear in the system.

   We tried to apply the removal of the redundant constraints on the result of FMplex using the linear solver GLPK. While we successfully identified redundant constraints in some instances, allowing us to remove them and proceed with fewer constraints, we encountered challenges in detecting redundancies in most cases due to unbounded numerical values.

   Indeed, a proactive approach to avoid adding redundant constraints can be very beneficial. Some heuristics can be applied to remove redundant constraints during FMplex. A practical and straightforward approach can be the following: do not add those constraints to the system that are multiples of already existing constraints during the FMplex. This will not guarantee a redundancy-free system afterward, but at this stage, it can significantly decrease the number of returned constraints.

However, we think that even if the above-listed and also other improvements are done on FMplex, it may still be challenging to achieve significant runtime improvements using dimension reduction in star-based reachability analysis of FNNs. Nevertheless, with continued enhancements and optimizations, there is a possibility of achieving more efficient analysis procedures with the dimension reduction method that can provide valuable insights into the behavior of FNNs.

In particular, even if the analysis cannot be significantly accelerated, it can still yield important information about the network's behavior. Detecting dimensions that violate safety properties, for example, can be crucial for identifying potential vulnerabilities in the network. This information can be used to generate adversarial examples, which can then be used as training data to improve the network's pattern recognition capabilities.

Therefore, while the dimension reduction method may not directly improve runtime efficiency, it can still play a valuable role in enhancing the overall analysis process and improving the robustness of FNNs.

The dimension reduction can also be performed not only at the end of each layer but also after each neuron in a layer. This approach would involve checking for changes in the generator matrix's rank and number of columns after each projection step. Furthermore, this approach might lean towards employing FM over FMplex, as eliminating only one variable tends to be more favorable with FM. However, in certain scenarios, FMplex may outperform FM. Therefore, it is crucial to develop an implementation tailored for efficient single-variable elimination. This alternative approach requires further implementation and evaluation to assess its effectiveness

comprehensively. Unfortunately, due to time limitations, we were unable to implement and evaluate this approach in our experiments.

Besides these applications, where an actual variable elimination is applied, our implementation can be used differently. The ability to transform a star set into an equivalent form efficiently opens up opportunities for various applications where a desired representation of the star set is needed. We proved, that the transformation of a star set returns another star set that is equivalent to the original one.

The experimental results show that the transformation of a star set to an equivalent one is done very efficiently. It took an average of less than a millisecond on a real-life benchmark (ACAS Xu). Moreover, the flexibility in choosing the transformation matrix $\mathcal{G}$ allows for customization based on specific requirements or desired properties of the resulting star set. The only restriction on choosing $\mathcal{G}$ is that it needs to be invertible.

Finally, the implementation of the dimension reduction method as it is now can be used by the developers of FMplex. With the help of the efficient star set transformation operation, they can easily test their improvements and ideas on any star-set-based benchmark.

Bibliography

[Ábr23] Erika Ábrahám. Gauß and Fourier-Motzkin variable elimination. 2023.

[ASB10] Matthias Althoff, Olaf Stursberg, and Martin Buss. Computing reachable sets of hybrid systems using a combination of zonotopes and polytopes. *Nonlinear analysis: hybrid systems*, 4(2):233–249, 2010.

[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.

[BDTD+16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[CD07] Mark Chavira and Adnan Darwiche. Compiling bayesian networks using variable elimination. In *IJCAI*, volume 2443. Citeseer, 2007.

[Cha93] Vijay Chandru. Variable elimination in linear constraints. *The Computer Journal*, 36(5):463–472, 1993.

[Com03] Christophe Combastel. A state bounding observer based on zonotopes. In *2003 European control conference (ECC)*, pages 2589–2594. IEEE, 2003.

[Dan63] George Dantzig. *Linear programming and extensions*. Princeton university press, 1963.

[DGPT24] Stefano Demarchi, Dario Guidotti, Luca Pulina, and Armando Tacchella. Never2: Learning and verification of neural networks. 2024.

[DKML19] Daniel Durstewitz, Georgia Koppe, and Andreas Meyer-Lindenberg. Deep neural networks in psychiatry. *Molecular psychiatry*, 24(11):1583–1598, 2019.

[DV16] Parasara Sridhar Duggirala and Mahesh Viswanathan. Parsimonious, simulation based verification of linear systems. In *International Conference on Computer Aided Verification*, pages 477–494. Springer, 2016.

[GAD+13] Jared Lee Gearhart, Kristin Lynn Adair, Justin David Durfee, Katherine A Jones, Nathaniel Martin, and Richard Joseph Detry. Comparison of open-source linear programming solvers. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2013.

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

[GJ+10] Gaël Guennebaud, Benoit Jacob, et al. Eigen. *URl: http://eigen. tuxfamily. org*, 3(1), 2010.

[Gra96] Torbjörn Granlund. Gnu mp. *The GNU Multiple Precision Arithmetic Library*, 2(2), 1996.

[HC18] Ian Goodfellow Yoshua Bengio Heaton, Jeff and Aaron Courville. Deep learning: The mit press, 2016, 800 pp, isbn: 0262035618. *Genetic Programming and Evolvable Machines*, 19(1-2):305–307, 2018.

[HKWW17] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*, pages 3–29. Springer, 2017.

[Jia23] Ruoran Gabriela Jiang. *Verifying ai-controlled hybrid systems.* PhD thesis, Master's thesis, RWTH Aachen University, Aachen, Germany. Available at https . . . , 2023.

[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 *Computer Aided Verification: 29th International Conference, CAV 2017, Part I 30*, pages 97–117. Springer, 2017.

[KL91] Deepak Kapur and Yagati N Lakshman. *Elimination methods: An introduction.* State University of New York at Albany, Department of Computer Science, 1991.

[KTS+18] Dimitrios Kollias, Athanasios Tagaris, Andreas Stafylopatis, Stefanos Kollias, and Georgios Tagaris. Deep neural architectures for prediction in healthcare. *Complex & Intelligent Systems*, 4:119–131, 2018.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LJB+22] Diego Manzanas Lopez, Taylor T Johnson, Stanley Bak, Hoang-Dung Tran, and Kerianne Hobbs. Evaluation of neural network verification methods for air-to-air collision avoidance. *AIAA Journal of Air Transportation (JAT)*, 2022.

[LM17] Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351*, 2017.

[Mas23] Hana Masara. *Star set based reachability analysis of neural networks with differing layers and activation functions.* PhD thesis, Bachelor's thesis, RWTH Aachen University, Aachen, Germany., 2023.

[MJ01] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5(64-67):2, 2001.

[NPÁK23] Jasper Nalbach, Valentin Promies, Erika Ábrahám, and Paul Kobialka. Fmplex: A novel method for solving linear real arithmetic problems. *arXiv preprint arXiv:2309.03138*, 2023.

[RK22] Vignesh Raghuraman and Justin P Koeln. Set operations and order reductions for constrained zonotopes. *Automatica*, 139:110204, 2022.

[SÁMK17] Stefan Schupp, Erika Ábrahám, Ibtissem Ben Makhlouf, and Stefan Kowalewski. H y p ro: A c++ library of state set representations for hybrid systems reachability analysis. In *NASA Formal Methods: 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings 9*, pages 288–294. Springer, 2017.

[Saz06] Murat H Sazli. A brief review of feed-forward neural networks. *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering*, 50(01), 2006.

[Sch98] Murray Schechter. Integration over a polyhedron: an application of the fourier-motzkin elimination method. *The American Mathematical Monthly*, 105(3):246–251, 1998.

[Sch19] Stefan Schupp. *State set representations and their usage in the reachability analysis of hybrid systems*. PhD thesis, Dissertation, RWTH Aachen University, 2019, 2019.

[SHM+16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[SKP97] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

[SRMB16] Joseph K Scott, Davide M Raimondo, Giuseppe Roberto Marseglia, and Richard D Braatz. Constrained zonotopes: A new tool for set-based estimation and fault detection. *Automatica*, 69:126–136, 2016.

[ST19] Sadra Sadraddini and Russ Tedrake. Linear encodings for polytope containment problems, 2019.

[TMLM+19] Hoang-Dung Tran, Diago Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T Johnson. Star-based reachability analysis of deep neural networks. In *Formal Methods–The Next 30 Years: Third World Congress, FM 2019, Porto, Portugal, October 7–11, 2019, Proceedings 3*, pages 670–686. Springer, 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.

[YWLG17] Di Yan, Tao Wu, Ying Liu, and Yang Gao. An efficient sparse-dense matrix multiplication on a multicore system. In *2017 IEEE 17th International Conference on Communication Technology (ICCT)*, pages 1880–1883. IEEE, 2017.

[ZCH+20] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

Appendix A

Proofs

This section contains additional proofs that were omitted from the thesis. These proofs further support the main theorems and lemmas discussed throughout the thesis. Therefore, the purpose of the supplementary proofs section is to be read alongside the thesis, enhancing the reader's understanding.

Proposition A.0.1. *Given a matrix $\mathcal{A}$ in $\mathbb{R}^{n\times m}$ $n, m \in \mathbb{N} : n < m$. Prove there are at least $m - n$ linearly dependent columns in $\mathcal{A}$.*

Proof. Let $rank(\mathcal{A}) = r$. Since r is the maximum number of linearly independent columns, $r \leq \min(m, n)$, therefore, we have $r \leq n$. Now, consider the nullity of $\mathcal{A}$: nullity($\mathcal{A}$). The nullity is given by the rank-nullity theorem:

$$\text{nullity}(\mathcal{A}) = m - r$$

Since $r \leq n$, we have $m - r \geq m - n$, which implies that nullity($\mathcal{A}$) $\geq m - n$. The nullity of $\mathcal{A}$ represents the number of linearly dependent columns in $\mathcal{A}$. Therefore, we have shown that there are at least $m - n$ linearly dependent columns in $\mathcal{A}$. □

Proposition A.0.2 (Convex polytopes as stars)**.** *Any bounded convex polyhedron $\mathcal{P} = \{\mathbf{x} \mid \mathcal{C}\mathbf{x} \leq \mathbf{d}, \mathbf{x} \in \mathbb{R}^n\}$ can be presented as a star.*

Proof. The star set Θ represents the polyhedron $\mathcal{P}$ with the center $\mathbf{c} = [0 \ \dots \ 0]^T$, the basis vectors $\mathcal{V} = \{\mathbf{e_1}, \dots, \mathbf{e_n}\}$ in which $\mathbf{e_i}$ is the i-th basic vector of $\mathbb{R}^n$, and the predicate $P(\boldsymbol{\alpha}) \triangleq \mathcal{C}\mathbf{x} \leq \mathbf{d}$. □

Proposition A.0.3 (Affine transformation)**.** *Given a star set $\Theta = \langle \mathbf{c}, \mathcal{V}, P \rangle$, an affine mapping of the star Θ with the linear mapping matrix $\mathcal{W}$ and offset vector $\mathbf{b}$ defined by $\bar{\Theta} = \{\mathbf{y} \mid \mathcal{W}\mathbf{x} + \mathbf{b}, \mathbf{x} \in \Theta\}$ is another star such that*

$$\bar{\Theta} = \langle \bar{\mathbf{c}}, \bar{\mathcal{V}}, \bar{P} \rangle, \quad \bar{\mathbf{c}} = \mathcal{W}\mathbf{c} + \mathbf{b}, \quad \bar{\mathcal{V}} = [\mathcal{W}\mathbf{v^1}, \cdots, \mathcal{W}\mathbf{v^m}], \quad \bar{P} \equiv P$$

Proof. By the definition of a star, we have $\bar{\Theta} = \{\mathbf{y} \mid \mathbf{y} = \mathcal{W}\mathbf{c} + \mathbf{b} + \sum_{i=1}^{m}(\alpha_i \mathcal{W}\mathbf{v^i})\}$ so that $P(\alpha_1, \dots, \alpha_m) = \top$ yields that $\bar{\Theta}$ is another star with the center $\bar{\mathbf{c}} = \mathcal{W}\mathbf{c} + \mathbf{b}$, basis vectors $\bar{\mathcal{V}} = \{\mathcal{W}\mathbf{v^1}, \dots, \mathcal{W}\mathbf{v^m}\}$ and the same predicate P as the original star Θ. □

Proposition A.0.4 (Intersection with halfspace)**.** *Given a star set* $\Theta = \langle \mathbf{c}, \mathcal{V}, P \rangle$*, with* $\mathcal{V} \in \mathbb{R}^{n \times m}$ *and a halfspace* $\mathscr{H} = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{H}^T\mathbf{x} \leq g\}$ *with* $\mathbf{H} \in \mathbb{R}^n$ *and* $g \in \mathbb{R}$*. The intersection* $\Theta \cap \mathscr{H}$ *is another star set* $\bar{\Theta}$ *with:*

$$\bar{\Theta} = \langle \mathbf{c}, \mathcal{V}, \bar{P} \rangle \quad \textit{with} \quad \bar{P} = P \wedge P',$$

where $P'(\boldsymbol{\alpha}) \triangleq (\mathbf{H}^T\mathcal{V})\boldsymbol{\alpha} \leq g - \mathbf{H}^T\mathbf{c}$*, and* $\mathcal{V} = [\mathbf{v^1}, \mathbf{v^2}, ..., \mathbf{v^m}]$.

Proof. The resulting star is $\Theta = \{\mathbf{x} \mid \mathbf{x} = \mathbf{c} + \sum_{j=1}^{m} \alpha_j \mathbf{v^j}\}$ s.t. $(\alpha_1, \ldots, \alpha_m)^T \in P \cap \mathbf{H}^T\mathbf{x} \leq g\}$. Since $\mathbf{x} = \mathbf{x} + \sum_{j=1}^{m} \alpha_j \mathbf{v^j}$, the new constraint can be written as $\mathbf{H}^T(\mathbf{c} + \mathcal{V}\boldsymbol{\alpha}) \leq g$, where $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_m]^T$. Consequently, the new predicate is $P \wedge P', P'(\boldsymbol{\alpha}) = (\mathbf{H}^T\mathcal{V})\boldsymbol{\alpha} \leq (g - \mathbf{H}^T\mathbf{c})$.

□