Theory of Hybrid Systems Informatik 2

RWTH AACHEN UNIVERSITY

Diese Arbeit wurde vorgelegt am LuFG Theorie hybrider Systeme

BACHELORARBEIT

IMPLEMENTIERUNG VON SPARSE POLYNOMIAL ZONOTOPES IN HYPRO

Hao Tran

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

Zusätzlicher Berater:
Jozsef Kovacs

Aachen, Datum
27.11.2023

Abstract

Hybrid systems are systems with continuous and discrete behaviour that we encounter every day, e.g. in the form of modern cars and trams. Since their safety is of great importance, we want to verify whether a system can reach undesirable states. For their *formal verification*, we will use algorithms which perform various operations on state sets, hence efficient and accurate state set representations are needed. In this work, we introduce *Sparse Polynomial Zonotopes* as a new representation. They are able to model nonconvex sets while still allowing efficient computations on them, making them a suitable option for *reachability analysis*. Furthermore, they allow for a tighter enclosure and thus more precise result compared to other commonly used convex set representations. Additionally, they are quite flexible, supporting the conversion to other simpler representations and vice versa. We demonstrate their performance by comparing their runtime on benchmarks as well as the tightness of the enclosure with other state set representations. Overall, this work gives an overview of this novel representation and its practicality in reachability analysis.

Keywords— Hybrid Systems, Formal Verification, Reachability Analysis, Flowpipe Construction, State Set Representations

Contents

Notations

The following notations will be used in this work [KA21]. We use $\mathbb{R}$ to annotate the set of real numbers and $\mathbb{N}$ for the set of natural numbers, both sets include the element 0.. As usual, we will denote matrices by uppercase letters and vectors by lowercase letters (e.g. $A \in \mathbb{R}^{n \times m}$, $v \in \mathbb{R}^n$), while sets will be represented by calligraphic letters (e.g. $\mathcal{H}$). The *cardinality of a set* is denoted by $|\cdot|$. Given a vector $v \in \mathbb{R}^n$, we refer to its i-th entry with $v_{(i)}$, for a matrix $A \in \mathbb{R}^{n \times m}$, $A_{(i,\cdot)}$ refers to the i-th row, $A_{(\cdot,j)}$ to the j-th column and $A_{(i,j)}$ to the j-th entry of the i-th row. Given a set of indices $\mathcal{H} = \{h_1, \ldots, h_{|\mathcal{H}|}\}$ with $1 \leq h_i \leq m$, $\forall i \in \{1, \ldots, |\mathcal{H}|\}$, we use $A_{(\cdot,\mathcal{H})}$ for $\begin{bmatrix} A_{(\cdot,h_1)} & \cdots & A_{(\cdot,h_{|\mathcal{H}|})} \end{bmatrix}$. Given two matrices A and B, we denote the concatenation with $[A\ B]$, appending B to the right of A, and $\begin{bmatrix} A \\ B \end{bmatrix}$, appending B below A. We will use $\mathbf{0}^{(n,m)} \in \mathbb{R}^{n \times m}$ and $\mathbf{1}^{(n,m)} \in \mathbb{R}^{n \times m}$ to denote the matrices containing only zeros and ones respectively. $I_n \in \mathbb{R}^{n \times n}$ represents the identity matrix of dimension n and $\begin{bmatrix} \end{bmatrix}$ is the empty matrix. We define the left-multiplication of a matrix $M \in \mathbb{R}^{n \times m}$ with a set $\mathcal{S} \subset \mathbb{R}^n$ as $M \otimes \mathcal{S} = \{Ms \mid s \in \mathcal{S}\}$. The Minkowski addition of two sets $\mathcal{S}_1 \subset \mathbb{R}^n$ and $\mathcal{S}_2 \subset \mathbb{R}^n$ is defined as $\mathcal{S}_1 \oplus \mathcal{S}_2 = \{s_1 + s_2 \mid s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2\}$. Additionally, the Cartesian product of two sets $\mathcal{S}_1 \in \mathbb{R}^n$ and $\mathcal{S}_2 \in \mathbb{R}^m$ is $\mathcal{S}_1 \times \mathcal{S}_2 = \{\begin{bmatrix} s_1 & s_2 \end{bmatrix}^T \mid s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2\}$. Let $\mathcal{I}$ denote the set of all intervals [a,b] where a,$b \in \mathbb{R}$, $a \leq b$. A n-dimensional interval is defined as $\mathcal{I}^n = I_1 \times \cdots \times I_n$ where $I_n \in \mathcal{I}$.

Chapter 1

Introduction

[SNA17, Sect. 1] In computer science, *hybrid systems* are systems which combine both continuous and discrete behaviour. Examples are physical systems where a digital controller is interacting with a dynamic environment, e.g. a controller regulating the temperature of a room or a controller adjusting the brightness of a lamp according to the sunlight, as well as modern cars and public transport where controllers are integrated to perform various tasks in order for the machine to work properly. There are also hybrid systems without digital controllers, a well known example being a ball dropped from a certain height and bouncing off the ground. Since there are hybrid systems whose malfunctioning pose a major risk, such as a nuclear power plant whose reactor core temperature is adjusted automatically or any system that is in direct contact with people, their verification for safety is of great interest. We often model these systems with *hybrid automata*. *Formal verification* is an approach to ensure that a system meets its specified requirements. In formal verification, we try to check with *reachability analysis* whether a system can reach a set of undesirable states. Multiple methods for reachability analysis of hybrid automata have been developed, but we will focus on *flowpipe-construction-based* approaches in this work. Given a set of initial states, the analysis iteratively computes an overapproximation of the successor states for a fixed time step. The reachable states are then overapproximated by a single set on which we perform various operations such as the *Minkowski addition* or the *convex hull operation*. Since we want to apply this method on real cases and it thus should be practical for actual use, the computation time of the reachability analysis is of great importance. Furthermore, we also want our result to be as accurate as possible. Hence, we try to reduce the error by overapproximation and achieve a more precise solution, thereby minimizing the chance of an incorrect result. Due to this, we are in the need of an accurate *state set representation* on which should also be able to perform operations efficiently. In this work, we will introduce *Sparse Polynomial Zonotopes* [KA21] as a new nonconvex representation, SPZs for short. We begin with the formal definition of SPZs as well as various basic operations on them such as the multiplication with a matrix, while also proving their computational efficiency being at most polynomial with respect to the dimension of the system. We then demonstrate how flexible SPZs are by showing that we can convert them to other commonly used representations, e.g. zonotopes [Gir05], and vice versa with mostly efficient time complexity. Afterwards, we elaborate on reachability analysis, defining hybrid automata as a model of hybrid systems and going into detail how we can

compute an overapproximation of the reachable states in a system through a flowpipe-construction based approach. Following that, we present a way to obtain an even more accurate result in the reachability analysis with SPZs by the *splitting algorithm* [SV22, p. 501] with which we can reduce the overapproximation error through conversion to zonotopes. After that, we make a comparison with the other state set representations, showing how SPZs behave in terms of accuracy and efficiency. Finally, we summarize the results of our work and evaluate the practicality of Sparse Polynomial Zonotopes.

1.1 HyPro

[SNA17, Sec. 3] As previously mentioned, there are several other state set representations for reachability analysis. To aid their time-consuming implementation, the open-source C++ library `HyPro` [SÁMK17] has been developed and published at https://github.com/hypro/hypro. It includes a collection of representations which have already been implemented such as zonotopes and convex polytopes [Zie12]. Besides being able to perform various operations on them, conversions between these representations are also supported. Furthermore, `HyPro` features a flowpipe-construction based reachability analysis algorithm. Several operations are performed on the state sets during the analysis, hence the requirement for the representations to support these. As a result, almost all of the state set representations share a *unified interface*, allowing the user to use multiple representations in a single algorithm. The main task of this work is the extension of `HyPro` by Sparse Polynomial Zonotopes as presented in the following. Some operations used in the reachability analysis are not supported by SPZs, therefore we need to be able to convert them to another representation where we can perform them.

1.2 Related Work

Initially, polynomial zonotopes were proposed as a representation because of their nonconvex property, being able to tightly enclose the reachable sets for nonlinear systems in comparison to the convex representations used in most previous works [Alt13].

Besides for reachability analysis of nonlinear hybrid systems, polynomial zonotopes have also recently been used for the reachability of nonlinear systems with uncertain parameters [LKB23]. This led to a more accurate result in comparison to zonotopes. Additionally, in recent works, polynomial zonotopes have been used in many other fields such as the reachability analysis of neural network controlled systems [KSAB23], safe real-time motion planning and controlling [MHZ+23] and safe reinforcement learning via action projection [KKW+23].

Furthermore, we can also use polynomial zonotopes for reachability with Koopman linearized models obtained from trajectory data when the model of the dynamic system is not given [SV22, p. 490]. Since this approach leads to a nonlinear transformation through which convex initial sets can become complex nonconvex sets, we can use polynomial zonotopes in order to obtain a tighter enclosure.

Another known set representation are polytopes in vertex representation which are closed under important operations such as the linear map, Minkowski sum, Cartesian product, convex hull and intersection [GKKZ03], making them a good choice for

reachability analysis. Without the removal of redundant points, computing the cartesian product and convex hull of V-polytopes is straightforward. Nevertheless, the amount of vertices typically grows exponentially with respect to the dimension, hence linear map and Minkowski sum have exponential time complexity, while computing the intersection is NP-hard [Tiw08].

Zonotopes are a popular subclass of polytopes [Zie12], allowing an exact and efficient computation of the linear map, Minkowski sum and cartesian product. Thus, they are well suited for the use in reachability analysis [Zie12, Gir05].

A special case of zonotopes are multi-dimensional intervals which allow for range bounding by interval arithmetic when dealing with nonlinear functions [JKDW01] and can certainly be used for reachability as well [ERNF11, RN11].

Chapter 2

Preliminaries

2.1 Sparse Polynomial Zonotopes

Definition 2.1.1. *(Sparse Polynomial Zonotope [KA21, Def. 1])*

Given a generator matrix of dependent generators $G \in \mathbb{R}^{n \times h}$, a generator matrix of independent generators $G_I \in \mathbb{R}^{n \times q}$, and an exponent matrix $E \in \mathbb{N}^{p \times h}$, an SPZ is defined as

$$\mathcal{SPZ} = \left\{ \sum_{i=1}^{h} \left(\prod_{k=1}^{p} \alpha_k^{E_{(k,i)}} \right) G_{(\cdot,i)} + \sum_{j=1}^{q} \beta_j G_{I(\cdot,j)} \;\middle|\; \alpha_k, \beta_j \in [-1,1] \right\}. \tag{2.1}$$

For a compact notation, we introduce the shorthand $\mathcal{SPZ} = \langle G, G_I, E \rangle_{\mathcal{SPZ}}$. The α_k scalars are called *dependent factors* because a change in their value can affect the multiplication with multiple generators. Due to these factors, SPZs are nonconvex in general. The scalars β_j only affect multiplication with a single generator. We call them *independent factors*. The term $\alpha_1^{E_{(1,i)}} \cdot \dots \cdot \alpha_p^{E_{(p,i)}} \cdot G_{(\cdot,i)}$ is referred to as a monomial and $\alpha_1^{E_{(1,i)}} \cdot \dots \cdot \alpha_p^{E_{(p,i)}}$ as the variable part of the monomial.

Note that we can transform any independent generator into a dependent one, without changing the set, by simply moving it to the dependent generator matrix and adding a new dependent factor whose exponent is 1 for that generator and 0 for all others. We keep the independent generators for computational reasons, since our computations on them are fast, though overapproximative, while computations on the dependent generators are more expensive but exact. As we can see by the dimensions of the generator matrices G and G_I, h is the number of dependent generators, q is the number of independent generators and n is the space dimension of which the generators are members of. We can tell by the exponent matrix E that the number of dependent factors is p while q also denotes the number of independent factors. The order ρ of an SPZ is defined as $\rho = \frac{h+q}{n}$. To compute the computational complexity for operations later on, we introduce

$$h = c_h n, \; p = c_p n, \; q = c_q n. \tag{2.2}$$

The factors c_h, c_p, $c_q \in \mathbb{R}_{\geq 0}$ are constants to signify that the given values are in the order of the dimension. We justify this assumption by limiting the order ρ and keeping

it below a desired value. When we index a matrix, all of its components are denoted with the same index, e.g. p_i, h_i etc. belong to $\mathcal{SPZ}_i$. The original paper [KA21] also presented a way to keep track of the dependencies between dependent factors from *different* SPZs by introducing an identifier vector *id* for each SPZ. This allows the computation of a tighter result for the addition of two SPZs. Since it doesn't introduce any useful information and it's not necessary for our work, we exclude it.

For better comprehension, we will introduce the following example:

Example 2.1.1. *Let*

$$\mathcal{SPZ} = \left\langle \begin{bmatrix} 2 & -1 & 0 & -1 & 1 \\ -1 & 1 & 2 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \right\rangle_{\mathcal{SPZ}}. \tag{2.3}$$

$\mathcal{SPZ}$ has five dependent generators, namely

$$\begin{bmatrix} 2 \\ -1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

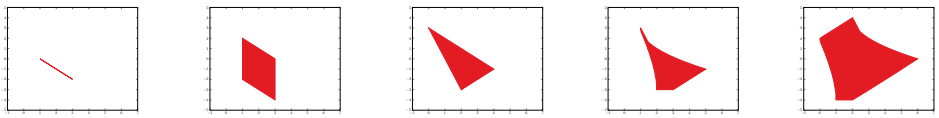
but just one independent generator:

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Each of the columns of the exponent matrix refer to one of the dependent generators. Since there are two rows, we can conclude that there are also only two dependent factors, α_1 and α_2. Thus, (2.3) defines the set

$$\mathcal{SPZ} = \left\{ \begin{bmatrix} 2 \\ -1 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix} \alpha_1 + \begin{bmatrix} 0 \\ 2 \end{bmatrix} \alpha_2 + \begin{bmatrix} -1 \\ 1 \end{bmatrix} \alpha_1\alpha_2 + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \alpha_1^2 + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \beta_1 \;\middle|\; \alpha_1,\alpha_2,\beta_1 \in [-1,1] \right\}. \tag{2.4}$$

An SPZ is constructed generator by generator. This process is visuallized on figure



(a) (b) (c) (d) (e)

Figure 2.1: The SPZ in (2.3) is constructed generator by generator.

2.1, where (a) shows the set obtained by the first two dependent generators, subfigures (b) to (d) show the resulting sets by adding the remaining dependent generators one by one (from left to right) and (e) adding the independent generator and thus illustrating the final set.

2.2 Basic Operations on SPZs

The following section presents some basic operations on Sparse Polynomial Zonotopes such as multiplication, addition and so on. It is crucial to be able to perform those

operations to seamlessly integrate SPZs into our HyPro library since when using them as a state set representation, our algorithms rely on them supporting said operations. We will also see that most operations can be done easily and more importantly efficiently, which is an essential criterion for comparison with other set representations. Thus, all operations are implemented in HyPro as presented in this section.

Some of the following set operations may result in an SPZ with monomials sharing an identical variable part. It means that the exponent matrix would have multiple identical columns. Since this could be represented more efficiently by aggregating the corresponding dependent generators, we introduce the following operation.

Proposition 2.2.1. *(Compact [KA21, Prop. 2])*

Given an SPZ $\mathcal{SPZ} = \langle G, G_I, E \rangle_{\mathcal{SPZ}}$, the operation `compact` returns a compressed representation of the set $\mathcal{SPZ}$:

$$\texttt{compact}(\mathcal{SPZ}) = \langle \overline{G}, G_I, \overline{E} \rangle_{\mathcal{SPZ}} \tag{2.5}$$

with

$$\begin{aligned} \overline{E} &= \texttt{uniqueColumns}(E) \in \mathbb{N}_0^{p \times k} \\ \mathcal{H}_j &= \{i \mid \overline{E}_{(l,j)} = E_{(l,i)} \forall l \in \{1, \dots, p\}\}, \\ \overline{G} &= \left[\sum_{i \in \mathcal{H}_1} G_{(\cdot,i)} \cdots \sum_{i \in \mathcal{H}_k} G_{(\cdot,i)} \right]. \end{aligned}$$

The operation `uniqueColumns` removes identical matrix columns until all columns are unique.

Proof. Given an SPZ $\mathcal{SPZ} = \langle G, G_I, E \rangle$ such that $E = \begin{bmatrix} e & e \end{bmatrix}$, meaning the exponent matrix consists of two identical columns $e \in \mathbb{N}_0^p$, the following holds:

$$\begin{aligned} &\left\{ \left(\prod_{k=1}^{p} \alpha_k^{e_{(k)}} \right) G_{(\cdot,1)} + \left(\prod_{k=1}^{p} \alpha_k^{e_{(k)}} \right) G_{(\cdot,2)} \;\middle|\; \alpha_k \in [-1,1] \right\} \\ &= \left\{ \left(\prod_{k=1}^{p} \alpha_k^{e_{(k)}} \right) \left(G_{(\cdot,1)} + G_{(\cdot,2)} \right) \;\middle|\; \alpha_k \in [-1,1] \right\}. \end{aligned}$$

Therefore, combining the generators for monomials with same variable part, thus summing up the dependent generators with same exponents, does not change the set. Hence, $\texttt{compact}(\mathcal{SPZ}) \equiv \mathcal{SPZ}$. Since the number of unique columns k of exponent matrix E is less than or equal to the number of overall columns h, the new matrices $\overline{E}$ and $\overline{G}$ are smaller or as big as the initial matrices E and G, thus resulting in a compressed representation of $\mathcal{SPZ}$. □

Complexity: We can implement the operation `uniqueColumns` efficiently by first sorting the columns of matrix E with worst-case complexity $\mathcal{O}(ph \log h)$ ([Knu98, Chapter 5]) since $k \leq h$ and each column has p entries and then comparing neighboring columns for equality, by which we end up with a complexity of $\mathcal{O}(ph \log h)$. Similiarly, for the construction of matrices $\mathcal{H}_j$, we will also compare the columns of the sorted matrix for equality, and since we have to compute $\mathcal{H}_1$ to $\mathcal{H}_k$, we get a complexity of $\mathcal{O}(hp)$, and since all other operations have at most quadratic complexity, the overall computational complexity is $\mathcal{O}(ph \log h)$, thus $\mathcal{O}(n^2 \log n)$ using (2.2).

We define the left-multiplication with a matrix as follows:

Proposition 2.2.2. *(Multiplication [KA21, Prop. 8])*

Given an SPZ $\mathcal{SPZ} = \langle G, G_I, E \rangle_{\mathcal{SPZ}} \subset \mathbb{R}^n$ and a matrix $M \in \mathbb{R}^{m \times n}$, the left-multiplication is defined as

$$M \otimes \mathcal{SPZ} = \langle MG, MG_I, E \rangle_{\mathcal{SPZ}}. \tag{2.6}$$

Proof. Inserting the definition of SPZs in (2.1) into the definition of the left-multiplication of a matrix with a set in Section directly results in (2.6). □

Complexity: We only have to consider the complexity of both matrix multiplications. Since $G \in \mathbb{R}^{n \times h}$ and $G_I \in \mathbb{R}^{n \times q}$, the computational complexity is $\mathcal{O}(mnh) + \mathcal{O}(mnq) = \mathcal{O}(mn^2)$ using (2.2).

Proposition 2.2.3. *(Minkowski Addition [KA21, Prop. 9])*

Given two SPZs, $\mathcal{SPZ}_1 = \langle G_1, G_{I,1}, E_1 \rangle_{\mathcal{SPZ}}$ and $\mathcal{SPZ}_2 = \langle G_2, G_{I,2}, E_2 \rangle_{\mathcal{SPZ}}$, their Minkowski sum is defined as

$$\mathcal{SPZ}_1 \oplus \mathcal{SPZ}_2 = \left\langle \begin{bmatrix} G_1 & G_2 \end{bmatrix}, \begin{bmatrix} G_{I_1} & G_{I_2} \end{bmatrix}, \begin{bmatrix} E_1 & \mathbf{0}^{(p_1,h_2)} \\ \mathbf{0}^{(p_2,h_1)} & E_2 \end{bmatrix} \right\rangle_{\mathcal{SPZ}} \tag{2.7}$$

Proof. Inserting the definition of SPZs in (2.1) into the definition of the Minkowski sum in Section directly results in (2.7). □

Complexity: Since we only have to concatenate matrices for the construction of the SPZ, its computational complexity is $\mathcal{O}(1)$.

The Minkowski addition of two SPZs requires the construction of a bigger exponent matrix. This is not necessary if the dependent factors of both SPZs are exactly the same, e.g. we want to compute the Minkowski sum of an SPZ with itself (or a multiple of itself). In this case, we can define the exact addition as:

Proposition 2.2.4. *(Exact Addition [KA21, Prop. 10])*

Given two SPZs, $\mathcal{SPZ}_1 = \langle G_1, G_{I,1}, E_1 \rangle_{\mathcal{SPZ}}$ and $\mathcal{SPZ}_2 = \langle G_2, G_{I,2}, E_2 \rangle_{\mathcal{SPZ}}$ with identical dependent factors, the exact addition is defined as

$$\mathcal{SPZ}_1 \boxplus \mathcal{SPZ}_2 = \langle \begin{bmatrix} G_1 & G_2 \end{bmatrix}, \begin{bmatrix} G_{I,1} & G_{I,2} \end{bmatrix}, \begin{bmatrix} E_1 & E_2 \end{bmatrix} \rangle_{\mathcal{SPZ}} \tag{2.8}$$

Proof. Identical to the Minkowski addition in Prop. 2.2.3. □

Complexity: Identical to the Minkowski addition in Prop. 2.2.3.

Before moving on to the SPZs, let us first define the linear combination in general.

Definition 2.2.1. *(Linear combination [KA21, Def. 7])*

Given two sets $\mathcal{S}_1 \subset \mathbb{R}^n$ and $\mathcal{S}_2 \subset \mathbb{R}^n$, the linear combination of them is defined as

$$\texttt{comb}(\mathcal{S}_1,\mathcal{S}_2) = \{0.5(1+\lambda)s_1 + 0.5(1-\lambda)s_2 \mid s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2, \lambda \in [-1,1]\}. \tag{2.9}$$

Note that in the case that $\mathcal{S}_1$ and $\mathcal{S}_2$ are both convex sets, their linear combination will be the same as the known convex hull, which we will denote with $\texttt{conv}(\mathcal{S}_1, \mathcal{S}_2)$. The paper [KA21] presents the linear combination of SPZs with and without independent generators.

Proposition 2.2.5. *(Linear combination without independent Generators [KA21, Prop. 14])*

Given two SPZs, $\mathcal{SPZ}_1 = \langle G_1, [\,], E_1 \rangle_{\mathcal{SPZ}}$ and $\mathcal{SPZ}_2 = \langle G_2, [\,], E_2 \rangle_{\mathcal{SPZ}}$, their linear combination is computed as

$$\begin{aligned} &\text{comb}(\mathcal{SPZ}_1, \mathcal{SPZ}_2) = \langle \overline{G}, [\,], \overline{E} \rangle_{\mathcal{SPZ}} \\ &\text{with} \\ &\overline{G} = 0.5 \begin{bmatrix} G_1 & G_1 & G_2 & -G_2 \end{bmatrix} \\ &\overline{E} = \begin{bmatrix} E_1 & E_1 & \mathbf{0}^{(p_1,h_2)} & \mathbf{0}^{(p_1,h_2)} \\ \mathbf{0}^{(p_2,h_1)} & \mathbf{0}^{(p_2,h_1)} & E_2 & E_2 \\ \mathbf{0}^{(1,h_1)} & \mathbf{1}^{(1,h_1)} & \mathbf{0}^{(1,h_2)} & \mathbf{1}^{(1,h_2)} \end{bmatrix} \end{aligned} \tag{2.10}$$

Proof. Since according to 2.2.1 we will add an SPZ to the multiple of itself, we can use the exact addition as presented in 2.2.4. Thus, we can write 2.2.1 as

$$\begin{aligned} &\texttt{comb}(\mathcal{SPZ}_1, \mathcal{SPZ}_2) = \\ &\qquad \{0.5(\mathcal{SPZ}_1 \boxplus \lambda\, \mathcal{SPZ}_1) \oplus 0.5(\mathcal{SPZ}_2 \boxplus (-\lambda)\, \mathcal{SPZ}_2) \mid \lambda \in [-1,1]\}. \end{aligned}$$

Now by introducing an additional dependent factor $\alpha_{p_1+p_2+1} \in [-1,1]$, we can substitute λ and thus with the definition of the minkowski and exact addition, we get the equations in 2.10. Since $\lambda \in [-1,1]$ and $\alpha_{p_1+p_2+1} \in [-1,1]$, this substitution does not change the set. □

We demonstrate the enclosure by a simple example.

Example 2.2.1. *Let*

$$\begin{aligned} \mathcal{SPZ}_1 &= \left\langle \begin{bmatrix} 2 & -1 & 0 \\ -1 & 1 & 2 \end{bmatrix}, [\,], \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right\rangle_{\mathcal{SPZ}} \\ \mathcal{SPZ}_2 &= \left\langle \begin{bmatrix} 1 & 3 & 5 \\ 2 & 1 & 5 \end{bmatrix}, [\,], \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \right\rangle_{\mathcal{SPZ}}. \end{aligned}$$

Computing $\overline{G}$ and $\overline{E}$ leads us to

$$\begin{aligned} \overline{G} &= \begin{bmatrix} 1 & -0.5 & 0 & 1 & -0.5 & 0 & 0.5 & 1.5 & 2.5 & -0.5 & -1.5 & -2.5 \\ -0.5 & 0.5 & 1 & -0.5 & 0.5 & 1 & 1 & 0.5 & 2.5 & -1 & -0.5 & -2.5 \end{bmatrix} \\ \overline{E} &= \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \end{aligned}$$

With both matrices, we can now construct the linear combination of $\mathcal{SPZ}_1$ and $\mathcal{SPZ}_2$ as shown in figure 2.2.

For computational reasons, we will return an overapproximation of the linear combination when SPZs with independent generators are given.

Proposition 2.2.6. *(Linear combination with independent Generators [KA21, Prop. 15])*

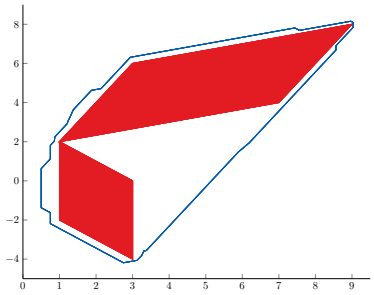


Figure 2.2: Plotting of the linear combination of $\mathcal{SPZ}_1$ and $\mathcal{SPZ}_2$.

Given two SPZs, $\mathcal{PZ}_1 = \langle G_1, G_{I,1}, E_1 \rangle_{\mathcal{PZ}}$ and $\mathcal{PZ}_2 = \langle G_2, G_{I,2}, E_2 \rangle_{\mathcal{PZ}}$, we over-approximate their linear combination by

$$\begin{aligned} \texttt{comb}(\mathcal{PZ}_1, \mathcal{PZ}_2) &\subseteq \langle \overline{G}, \overline{G}_I, \overline{E} \rangle_{\mathcal{PZ}} \\ &\text{with} \\ \langle \overline{G}, [\,], \overline{E} \rangle_{\mathcal{PZ}} &= \texttt{comb}(\overline{PZ}_1, \overline{PZ}_2) \\ \langle \mathbf{0}^{(n,1)}, \overline{G}_I \rangle_Z &\supseteq \texttt{conv}(\langle \mathbf{0}^{(n,1)}, G_{I,1} \rangle_Z, \mathbf{0}^{(n,1)}, G_{I,2} \rangle_Z) \\ \overline{\mathcal{PZ}}_1 &= \langle G_1, [\,], E_1 \rangle_{\mathcal{PZ}} \\ \overline{\mathcal{PZ}}_2 &= \langle G_2, [\,], E_2 \rangle_{\mathcal{PZ}}. \end{aligned} \tag{2.11}$$

Since zonotopes are not closed under the convex hull operation, we use the overapproximation of two zonotopes $\mathcal{Z}_1$ and $\mathcal{Z}_2$ as presented in [Alt10, Eq 2.2] such that

$$\langle \mathbf{0}^{(n,1)}, \overline{G}_I \rangle_Z \supseteq \text{conv}(\langle \mathbf{0}^{(n,1)}, G_{I,1} \rangle_Z, \mathbf{0}^{(n,1)}, G_{I,2} \rangle_Z)$$

with

$$\begin{aligned} \overline{G}_I &= \begin{cases} \left[\hat{G}_1 \quad G_{i,1(\cdot,\{q_2+1,\ldots,q_1\})}\right], q_1 \geq q_2 \\ \left[\hat{G}_1 \quad G_{i,1(\cdot,\{q_2+1,\ldots,q_1\})}\right], q_1 \geq q_2 \end{cases} \\ \hat{G}_1 &= \frac{1}{2}\left[G_{I,1(\cdot,\mathcal{K}_2)} + G_{I,2} \quad G_{I,1(\cdot,\mathcal{K}_2)} - G_{I,2}\right] \\ \hat{G}_2 &= \frac{1}{2}\left[G_{I,2(\cdot,\mathcal{K}_1)} + G_{I,1} \quad -G_{I,2(\cdot,\mathcal{K}_1)} + G_{I,1}\right] \end{aligned}$$

where $\mathcal{K}_1 = \{1, \ldots, q_1\}$ and $\mathcal{K}_2 = \{1, \ldots, q_2\}$. The operation $\texttt{comb}(\mathcal{PZ}_1, \mathcal{PZ}_2)$ refers to the convex hull as presentend in Prop. 2.2.5.

Proof. First, we see that given sets $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4 \subset \mathbb{R}^n$, we can derive by definition

that

$$
\begin{aligned}
&\texttt{comb}(\mathcal{S}_1 \oplus \mathcal{S}_2, \mathcal{S}_3 \oplus \mathcal{S}_4)\\
=&\{0.5(1+\lambda)(s_1+s_2)+0.5(1-\lambda)(s_3+s_4) \mid s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2, s_3 \in \mathcal{S}_3, s_4 \in \mathcal{S}_4, \lambda \in [-1,1]\}\\
=&\{\underbrace{0.5(1+\lambda)s_1+0.5(1-\lambda)s_3}_{\texttt{comb}\,(\mathcal{S}_1,\mathcal{S}_3)}+\underbrace{0.5(1+\lambda)s_2+0.5(1-\lambda)s_4}_{\texttt{comb}\,(\mathcal{S}_2,\mathcal{S}_4)} \mid\\
&\quad s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2, s_3 \in \mathcal{S}_3, s_4 \in \mathcal{S}_4, \lambda \in [-1,1]\}\\
\subseteq&\{0.5(1+\lambda)s_1+0.5(1-\lambda)s_3 \mid s_1 \in \mathcal{S}_1, s_3 \in \mathcal{S}_3, \lambda \in [-1,1]\} \oplus\\
&\quad \{0.5(1+\lambda)s_2+0.5(1-\lambda)s_4 \mid s_2 \in \mathcal{S}_2, s_4 \in \mathcal{S}_4, \lambda \in [-1,1]\}\\
=&\,\texttt{comb}\,(\mathcal{S}_1,\mathcal{S}_3) \oplus \texttt{comb}\,(\mathcal{S}_2,\mathcal{S}_4).
\end{aligned}
$$

By insertion into the definitions, we can see that given an SPZ $\mathcal{SPZ} = \langle G, G_I, E\rangle_{\mathcal{SPZ}}$, we are able to split it up into

$$\mathcal{SPZ} = \langle G, [\,], E\rangle_{\mathcal{SPZ}} \oplus \langle \mathbf{0}^{(\mathbf{n},\mathbf{1})}, G_I\rangle_{\mathcal{Z}}.$$

We will denote the resulting SPZ and zonotope with $\overline{\mathcal{SPZ}}$ and $\overline{\mathcal{Z}}$. Thus with our previous result, we can compute the linear combination of two SPZs $\mathcal{SPZ}_1 = \langle G_1, G_{I,1}, E_1\rangle_{\mathcal{SPZ}}$ and $\mathcal{SPZ}_2 = \langle G_2, G_{I,2}, E_2\rangle_{\mathcal{SPZ}}$ with

$$
\begin{aligned}
&\texttt{comb}\,(\mathcal{SPZ}_1, \mathcal{SPZ}_2)\\
&= \texttt{comb}\,(\overline{\mathcal{SPZ}}_1 \oplus \overline{\mathcal{Z}}_1, \overline{\mathcal{SPZ}}_2 \oplus \overline{\mathcal{Z}}_2)\\
&\subseteq \texttt{comb}\,(\overline{\mathcal{SPZ}}_1, \overline{\mathcal{SPZ}}_2) \oplus \texttt{conv}\,(\overline{\mathcal{Z}}_1, \overline{\mathcal{Z}}_2)\,.
\end{aligned}
$$

Since we overapproximate the convex hull of the zonotopes, our result in 2.2.6 is also an overapproximation of the linear combination of two SPZs. □

We see that we can easily obtain the convex hull of two sparse polynomial zonotopes using the linear combination:

$$\texttt{conv}(\mathcal{SPZ}_1, \mathcal{SPZ}_2) = \texttt{comb}(\texttt{comb}(\mathcal{SPZ}_1, \mathcal{SPZ}_1), \texttt{comb}(\mathcal{SPZ}_2, \mathcal{SPZ}_2))$$

Since some operations may increase the number of generators and thus the order of an SPZ, it is necessary to reduce the order for the sake of the computational complexity. Therefore, we propose the following operation based on the order reduction of zonotopes [KSA17].

Proposition 2.2.7. *(Reduce [KA21, Prop. 16])*

Given an SPZ $\mathcal{SPZ} = \langle G, G_I, E\rangle_{\mathcal{SPZ}}$ and a desired order $\rho_d \geq 1+\frac{1}{n}$, the operation `reduce` returns an enclosure of $\mathcal{SPZ}$ with a smaller or equal order to ρ_d:

$$
\begin{aligned}
\texttt{reduce}(\mathcal{SPZ}, \rho_d) &= \left\langle \begin{bmatrix} c_z & G_{(\cdot,\hat{\mathcal{K}})} \end{bmatrix}, \begin{bmatrix} G_{I(\cdot,\hat{\mathcal{H}})} & G_z \end{bmatrix}, \begin{bmatrix} \mathbf{0}^{(p,1)} & E_{(\cdot,\hat{\mathcal{K}})} \end{bmatrix} \right\rangle_{SPZ}\\
\text{with } \langle c_Z, G_z\rangle_Z &= \texttt{reduce}(\mathcal{Z},1)\\
\mathcal{Z} &= \texttt{zono}(\langle G_{(\cdot,\mathcal{K})}, G_{I(\cdot,\mathcal{H})}, E_{(\cdot,\mathcal{K})}\rangle_{SPZ})
\end{aligned}
$$

We define

$$a := \max(0, \min(h+q, \lceil h+q-n(\rho_d-1)+1\rceil)).$$

For the reduction, we choose the smallest generators:

$$\begin{aligned}
\mathcal{K} &= \begin{cases} \emptyset & \text{, if a} = 0 \\ \left\{ i \mid \|G_{(\cdot,i)}\|_2 \leq \|\hat{G}_{(\cdot,d_{(a)})}\|_2 \right\}, & \text{otherwise} \end{cases} \\
\mathcal{H} &= \begin{cases} \emptyset & \text{, if a} = 0 \\ \left\{ i \mid \|G_{I(\cdot,i)}\|_2 \leq \|\hat{G}_{(\cdot,d_{(a)})}\|_2 \right\}, & \text{otherwise} \end{cases} \\
\hat{\mathcal{K}} &= \{1,\ldots,h\} \backslash \mathcal{K} \\
\hat{\mathcal{H}} &= \{1,\ldots,q\} \backslash \mathcal{H} \\
\text{with } & \|\hat{G}_{(\cdot,d_{(1)})}\|_2 \leq \cdots \leq \|\hat{G}_{(\cdot,d_{(h+q)})}\|_2,
\end{aligned}$$

where $\hat{G} = \begin{bmatrix} G & G_I \end{bmatrix}$ and $d \in \mathbb{N}^{h+q}$ with $d > 0$ is a vector of indices where every entry represents a longer generator than the previous one.

Proof. By the definition of a, it holds that $|\hat{\mathcal{K}}|+|\hat{\mathcal{H}}|+n+1 \leq \rho_d n$ for $\rho_d \geq 1+\frac{1}{n}$. Since `zono` and `reduce` are both overapproximative, it follows that $\texttt{reduce}(\mathcal{SPZ}, \rho_d) \supseteq \mathcal{SPZ}$ and $\texttt{reduce}(\texttt{zono}(\cdot))$ is overapproximative as well. □

Complexity: It is necessary to sort the dependent generators for the vector d which has a complexity of $\mathcal{O}(n(h+q))+\mathcal{O}((h+q)\log(h+q))$ and thus $\mathcal{O}(n^2)$ using equation 2.2. In the worst case, all dependent generators are reduced by which the conversion to a zonotope has a complexity of $\mathcal{O}(ph) + \mathcal{O}(nh)$, which is again $\mathcal{O}(n^2)$ using 2.2. Hence, we end up with the total computation complexity of $\mathcal{O}(n^2) + \mathcal{O}(\texttt{reduce})$ where $\mathcal{O}(\texttt{reduce})$ denotes the complexity of the zonotope reduction.

2.3 Conversion from other Set Representations

While operations on Sparse Polynomial Zonotopes might be slower compared to other set representations, they do allow for a tighter and more accurate representation. Since it could be benificial to sacrifice some time needed for computations for a more accurate result, it is important to have the ability to convert simpler set representations to SPZs. In this section, we will show the conversion of other known set representations to SPZs.

2.3.1 Zonotope and Interval

Definition 2.3.1. *(Zonotope [Gir05, Def. 1])*

Given a center $c \in \mathbb{R}^n$ and a generator matrix $G \in \mathbb{R}^{n \times l}$, a Zonotope $\mathcal{Z}$ is a set such that

$$\mathcal{Z} = \left\{ c + \sum_{i=1}^{h} \alpha_i G_{(\cdot,i)} \;\middle|\; \alpha_i \in [-1,1] \right\}. \tag{2.12}$$

For a compact notation we introduce the shorthand $\mathcal{Z} = \langle c,G \rangle_{\mathcal{Z}}$. To act as an intermediate step, we introduce the conversion from an interval to a zonotope.

Proposition 2.3.1. *(Conversion from Interval to Zonotope [Alt10, Prop. 2.1])*

Given an interval $\mathcal{I} = [l, u] \subset \mathbb{R}^n$, an equivalent zonotope $\mathcal{Z} = \langle c,G \rangle_\mathcal{Z}$ with $G = [g_1 \dots g_n]$ is given by

$$c = \frac{1}{2}(l+u), g_i^{(j)} = \begin{cases} \frac{1}{2}(u_i - l_i), & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \tag{2.13}$$

where $g_i^{(j)}$ denotes the j-th entry of the i-th generator.

Proof. By inserting c and G into the definition of a zonotope 2.12, we can see that we obtain an equivalent zonotope. □

Proposition 2.3.2. *(Conversion Zonotope [KA21, Prop. 3])*

Given a zonotope $\mathcal{Z} = \langle c,G \rangle_\mathcal{Z}$, an equivalent SPZ is given by

$$\mathcal{Z} = \left\langle \begin{bmatrix} c & G \end{bmatrix}, [\,], \begin{bmatrix} \mathbf{0}^{(l,1)} & I_l \end{bmatrix} \right\rangle_{\mathcal{SPZ}}$$

As shown earlier, we can convert any interval to a zonotope, so the conversion to an SPZ is straight-forward.

Proof. By inserting $G = \begin{bmatrix} c & G_\mathcal{Z} \end{bmatrix}, G_I = [\,]$ and $E = \begin{bmatrix} \mathbf{0}^{(n,1)} & I_l \end{bmatrix}$ into the definition of an SPZ 2.1, we can see that we obtain an equivalent SPZ. □

Complexity: Since this conversion only involves concatenations of matrices, the computational complexity is $\mathcal{O}(1)$.

2.3.2 Polytope

Definition 2.3.2. *(V-Representation of a Polytope [KA19, Def. 1])*

Given a set $V = \{v_1, \dots, v_r\}, v_i \in \mathbb{R}$, the vertex representation of a Polytope $\mathcal{P}$ is defined as

$$\mathcal{P} = \left\{ \sum_{i=1}^{r} \beta_i v_i \;\middle|\; \beta_i \geq 0, \sum_{i=1}^{r} \beta_i = 1 \right\}. \tag{2.14}$$

For a compact notation of the vertex representation, we introduce the shorthand $P = \langle [v_1, \dots, v_r] \rangle_{\mathcal{P}_v}$.

Theorem 2.3.3. *(Conversion Polytope [KA21, Theorem 1])*

Every polytope in V-representation can be equivalently represented as an SPZ.

Proof. According to Definition 2.3.2, we can represent every bounded polytope $\mathcal{P} = \langle [v_1, \dots, v_r] \rangle_{\mathcal{P}_v}$ by the convex hull of its vertices. We can also easily represent every vertex v_i as an SPZ: $v_i = \langle v_i, [\,], 0 \rangle_{\mathcal{SPZ}}$. Because SPZs without independent generators are closed under the convex hull operation, we can iteratively compute the convex hull of all vertices by which we obtain an SPZ equivalent to the polytope $\mathcal{P}$. □

2.4 Enclosure by other Set Representations

As mentioned in the previous section, other set representations might be faster to compute on which makes it desirable to be able to convert SPZs to previously shown representations. Furthermore, some operations are not even supported by SPZs, making those conversions a necessity. In this section, we will show the enclosing of SPZs by the other previously shown set representations. To demonstrate the enclosure, we will use the SPZ 2.3 as an example.

2.4.1 Zonotope

Proposition 2.4.1 (Enclosure by Zonotope [KA21], Prop. 5)**.**

Given an SPZ $\mathcal{SPZ} = \langle G, G_I, E \rangle_{\mathcal{SPZ}}$, an equivalent zonotope $Z = \langle c, G \rangle_Z$ is given by

$$
\begin{aligned}
\mathcal{Z} &= \left\langle \sum_{i \in \mathcal{N}} G_{(\cdot,i)} + 0.5 \sum_{i \in \mathcal{H}} G_{(\cdot,i)}, \begin{bmatrix} 0.5G_{(\cdot,\mathcal{H})} & G_{(\cdot,\mathcal{K})} & G_I \end{bmatrix} \right\rangle_{\mathcal{Z}} \\
&\textit{with} \\
&\mathcal{N} = \{ i \mid E_{(j,i)} = 0 \ \forall j \in \{1, \dots, p\} \} \\
&\mathcal{H} = \left\{ i \;\middle|\; \prod_{j=1}^{p} (1 - (E_{(j,i)} \bmod 2)) = 1 \right\} \setminus \mathcal{N} \\
&\mathcal{K} = \{1, \dots, h\} \setminus (\mathcal{H} \cup \mathcal{N})
\end{aligned}
\tag{2.15}
$$

with $x \bmod y$, x,$y \in \mathbb{R}$ denoting the modulo operation.

Proof. Since by definition the independent generators of an SPZ are equivalent to the generators of a zonotope, they can simply be taken into account by adding them to the generator matrix of the zonotope. All dependent generators with dependent factors where the exponents are all 0, thus forming the set $\mathcal{N}$, represent a constant offset. They remain the same no matter the values of the factors. Thus, adding all of them up into a single dependent generator doesn't change the set. Therefore, we can represent this single generator exactly by using it as the center of the zonotope. All other dependent generators have to be overapproximated. The set $\mathcal{H}$ is formed by the indices of all dependent generators whose factors have only even exponents. Thus, raising their factors to their powers will lead to a nonnegative value, so we can enclose them tighter using

$$
\forall i \in \mathcal{H} : \left(\prod_{k=1}^{p} [-1,1]^{E_{(k,i)}} \right) G_{(\cdot,i)} = [0,1] \cdot G_{(\cdot,i)} = 0.5 G_{(\cdot,i)} + [-1,1] 0.5 \; G_{(\cdot,i)}.
$$

The remaining generators in $\mathcal{K}$ have to be overapproximated by the interval [-1,1]. By inserting (2.15) into the definition of a zonotope, we can see that we have exactly represented or overapproximated the variable parts of all monomials, thus we obtain an overapproximation of the initial set and therefore an enclosure of our SPZ. □

Complexity: The complexity of constructing the set $\mathcal{H}$ is $\mathcal{O}(ph)$. The worst-case for computing the zonotope is to have all exponents even, leading to the summation

of all h dependent generators with n entries. Hence, we end up with the overall complexity of $\mathcal{O}(ph) + \mathcal{O}(nh)$, thus $\mathcal{O}(n^2)$ using 2.2.

Example 2.4.1. *As a reminder, our running example is 2.3.*

We first compute our sets $\mathcal{N}, \mathcal{H}$ and $\mathcal{K}$. Since only the first column of the exponent matrix E has only zeros as entries, its index is the only element in $\mathcal{N}$. Likewise, only the fifth column of E has only even entries with at least one entry nonzero, thus the fifth index is also the only element in $\mathcal{H}$. All other indices of dependent factors are therefore in $\mathcal{K}$. Thus, we obtain

$$\begin{aligned} \mathcal{N} &= \{1\} \\ \mathcal{H} &= \{5\} \\ \mathcal{K} &= \{2,3,4\}. \end{aligned} \tag{2.16}$$

Inserting all generators according to (2.15) results in the zonotope (visualized in figure 2.3)

$$\begin{aligned} \mathcal{Z} &= \left\langle \begin{bmatrix} 2 \\ -1 \end{bmatrix} + 0.5 \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.5 \begin{bmatrix} 1 \\ 0 \end{bmatrix} & \begin{bmatrix} -1 & 0 & -1 \\ 1 & 2 & 1 \end{bmatrix} & \begin{bmatrix} 1 \\ 1 \end{bmatrix} \end{bmatrix} \right\rangle_{\mathcal{Z}} \\ &= \left\langle \begin{bmatrix} 2.5 \\ -1 \end{bmatrix}, \begin{bmatrix} 0.5 & -1 & 0 & -1 & 1 \\ 0 & 1 & 2 & 1 & 1 \end{bmatrix} \right\rangle_{\mathcal{Z}} \end{aligned} \tag{2.17}$$

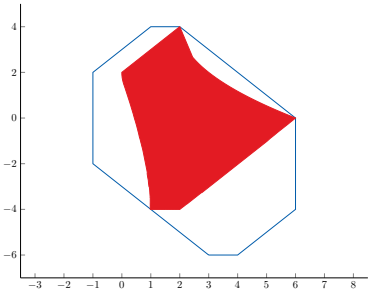


Figure 2.3: Enclosure of the SPZ in (2.3) by a zonotope

2.4.2 V-Polytope

Since it is necessary for the algorithm we use to convert SPZs to V-polytopes, we will introduce another representation of a polytope:

Definition 2.4.1. *(Z-Representation of a Polytope [KA19, Def. 4])*

Given a center $c \in \mathbb{R}^n$ and a generator matrix $G \in \mathbb{R}^{n \times h}$, the Z-representation of a polytope $\mathcal{P}$ is defined as

$$\mathcal{P} = \left\{ c + \sum_{i=1}^{h} \left(\prod_{k=1}^{m_i} \alpha_{\varepsilon_{(i,k)}} \right) G_{(\cdot,i)} \;\middle|\; \alpha_{\varepsilon_{(i,k)}} \in [-1,1] \right\} \tag{2.18}$$

with $I = (e_1,\ldots,e_h), \forall i \in \{1,\ldots,h\} : e_i \in \mathbb{N}^{m_i}$, $e_i \leq p$, and $\forall i \in \{1,\ldots,h\}$, $\forall j,k$: $j \neq k \Rightarrow e_{i(j)} \neq e_{i(k)}$. The tuple I is storing vectors of the indices of the factors, m_i is denoting the length of e_i, p is referring to the total number of factors $\alpha_{I_{(i,k)}}$ and h denotes the number of generators. Because of the constraint $\forall i \in \{1,\ldots,h\}\forall j,k$: $j \neq k \Rightarrow e_{i(j)} \neq e_{i(k)}$, no factor will appear more than once in the product, thus the product will only consist of factors with exponent 0 or 1. Note that every bounded polytope can be represented by the Z-representation (see [[KA19, Theorem 1]), but not every Z-representation is a polytope (see [KA19, Def. 4]). For a compact notation, we will introduce the shorthand $\mathcal{P} = \langle c,G,I\rangle_{\mathcal{P}_z}$.

Proposition 2.4.2. *(Conversion of a polytope from Z-representation to V-representation [KA19, Alg. 2])*

We introduce the algorithm:

Algorithm 1 Conversion from Z-representation to V-representation

Require: Bounded polytope in Z-representation $\mathcal{P} = \langle c,G,I\rangle_{\mathcal{Z}}$
Ensure: V-representation $\mathcal{P} = \langle[v_1,\ldots,v_r]\rangle_{\mathcal{P}}$ of the polytope
1: $\mathcal{I} \leftarrow [-\mathbf{1}^{(p)}, \mathbf{1}^{(p)}]$
2: $\{\hat{\alpha}^{(1)},\ldots,\hat{\alpha}^{(2^p)}\} \leftarrow \texttt{vertices}(\mathcal{I})$
3: $\mathcal{K} \leftarrow \emptyset$
4: **for** $j \leftarrow 1$ to 2^p **do**
5: $\quad v \leftarrow c + \sum_{i=1}^{h}(\prod_{k=1}^{m_i} \hat{\alpha}^{(j)}_{I_{(i,k)}})G_{(\cdot,i)}$
6: $\quad$ **if** $v \notin \mathcal{K}$ **then**
7: $\quad\quad \mathcal{K} \leftarrow \mathcal{K} \cup v$
8: $\quad$ **end if**
9: **end for**
10: $[v_1,\ldots,v_r] \leftarrow \texttt{convexHull}(\mathcal{K})$
11: $\mathcal{P} \leftarrow \langle[v_1,\ldots,v_r]\rangle_{\mathcal{P}}$

The operation `vertices` returns the 2^p vertices of the p-dimensional hypercube $\mathcal{I} = [-\mathbf{1}^{(p,1)}, \mathbf{1}^{(p,1)}] \subset \mathbb{R}^p$, e.g. for $p = 2$,

$$\texttt{vertices}(\mathcal{I}) = \left\{\begin{bmatrix}-1\\-1\end{bmatrix}, \begin{bmatrix}-1\\1\end{bmatrix}, \begin{bmatrix}1\\-1\end{bmatrix}, \begin{bmatrix}1\\1\end{bmatrix}\right\}.$$

For the proof of algorithm 1, we have to show the following proposition:

Proposition 2.4.3. *[KA19, Prop. 5]*

Given a polytope $\mathcal{P} = \langle c,G,I\rangle_{\mathcal{Z}}$ in Z-representation, the polytope vertices are a subset of

$$\mathcal{K} = \left\{ c + \sum_{i=1}^{h}\left(\prod_{k=1}^{m_i} \hat{\alpha}_{I_{(i,k)}}\right)G_{(\cdot,i)} \;\middle|\; \hat{\alpha} = [\hat{\alpha}_1,\ldots,\hat{\alpha}_p]^T \in \texttt{vertices}(\mathcal{I}) \right\}.$$

Proof. We have to show that for every vertex $v^{(j)}$ of the polytope, there is a corresponding vertex $\hat{\alpha}^{(j)}$ such that

$$v^{(j)} = c + \sum_{i=1}^{h}\left(\prod_{k=1}^{m_i} \hat{\alpha}^{(j)}_{I_{(i,k)}}\right)G_{(\cdot,i)}.$$

In [AP01, Chapter 7.2] it is shown that for each vertex $v^{(j)}$ of a polytope $\mathcal{P}$ there exists a vector $d_j \in \mathbb{R}^n$ such that

$$v^{(j)} = \underset{s \in \mathcal{P}}{\operatorname{argmax}}\, d_J^T s.$$

Combining both equations, we receive

$$\begin{aligned} v^{(j)} &= c + \sum_{i=1}^{h} \left(\prod_{k=1}^{m_i} \alpha^*_{I_{(i,k)}} \right) G_{(\cdot,i)} \\ \text{with } \; [\alpha_1^*, \ldots, \alpha_p^*]^T &= \underset{[\alpha_1,\ldots,\alpha_p]^T \in \mathcal{I}}{\operatorname{argmax}} \underbrace{d_j^T \left(c + \sum_{i=1}^{h} \left(\prod_{k=1}^{m_i} \alpha_{I_{(i,k)}} \right) G_{(\cdot,i)} \right)}_{f(\alpha_1,\ldots,\alpha_p)}. \end{aligned}$$

Thus, we must proof that the point $\alpha^* = [\alpha_1^*, \ldots, \alpha_p^*]^T$ where the maximum of the function $f(\cdot)$ is reached within the domain $[\alpha_1, \ldots, \alpha_p]^T \in \mathcal{I}$ is one of the vertices of the hypercube $\mathcal{I}$. The function $f(\cdot)$ does not contain a polynomial exponent which is greater than 1, hence it holds that its partial derivate with respect to the variable α_i does not depend on α_i:

$$\forall i \in \{1, \ldots, p\} : \frac{\delta f(\alpha_1, \ldots, \alpha_p)}{\delta \alpha_i} = g_i(\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_p),$$

for some function g_i. Therefore, $f(\cdot)$ reaches its extremum on the domain $\alpha_i \in [-1, 1]$ at either $\alpha_i^* = -1$ or $\alpha_i^* = 1$, and since this holds for all α_i, the function $f(\cdot)$ reaches its maximum within the domain $[\alpha_1, \ldots, \alpha_p]^T \in \mathcal{I}$ at the point $\alpha^* = [\alpha_1^*, \ldots, \alpha_p^*]^T$ with $\alpha_j^* \in \{-1, 1\}, j = 1, \ldots, p$, which we can see is a vertex of the hypercube $\mathcal{I}$. □

Algorithm 1 is based on proposition 2.4.3. Inside of the for-loop from line 4 to 9, the potential polytope vertices are computed according to said proposition. If the candidate is not already inside of the set $\mathcal{K}$, it will be added in line 7. After all iterations of the loop. the convex hull of all computed vertices is computed which results in the desired V-representation of the given polytope.

Proposition 2.4.4. *(Enclosure by Polytope [KA21, Prop. 6])*

Given an SPZ $\mathcal{SPZ} = \langle G, G_I, E \rangle_{\mathcal{SPZ}}$, an equivalent polytope $\mathcal{P} = \langle [v_1, \ldots, v_r] \rangle_{\mathcal{P}_v}$ is given by applying algorithm 1 to the following SPZ:

$$\begin{aligned} \overline{\mathcal{SPZ}} &= \langle \begin{bmatrix} c_z & G_{(\cdot,\mathcal{K})} \end{bmatrix}, \begin{bmatrix} G_I & G_z \end{bmatrix}, \begin{bmatrix} \mathbf{0}^{(n,1)} E_{(\cdot,\mathcal{K})} \end{bmatrix} \rangle_{\mathcal{SPZ}} \\ &\text{with} \\ &\quad \mathcal{H} = \{ i \mid \exists j \in \{1, \ldots, p\} : E_{(j,i)} > 1 \} \\ &\quad \mathcal{K} = \{1, \ldots, h\} \setminus \mathcal{H} \\ \langle c_z, G_z \rangle_{\mathcal{Z}} &= \operatorname{zono}(\langle G_{(\cdot,\mathcal{H})}, [\,], E_{(\cdot,\mathcal{H})} \rangle_{\mathcal{SPZ}}) \end{aligned} \tag{2.19}$$

Proof. Algorithm 1 returns a polytope in vertex representation given a polytope in Z-representation. Since a Z-representation does not contain any exponents greater than 1, we need to adjust our SPZ in order to be able to apply the algorithm on it. Hence, we split $\mathcal{SPZ}$ into two parts. First one having only zeros or ones in the

exponent matrix which we construct with $\mathcal{K}$ which contains all indices of generators with entries less than or equal to one in the corresponding column of the exponent matrix, resulting in $\langle G_{(\cdot,\mathcal{K})}, G_I, E_{(\cdot,\mathcal{K})}\rangle_{\mathcal{SPZ}}$. In order to remove exponents greater than one for the remaining part, it is enclosed by a zonotope. Since this leads to an overapproximation, combination of these two parts leads to the SPZ $\overline{\mathcal{SPZ}}$ which satisfies

$$\mathcal{SPZ} \subseteq \overline{\mathcal{SPZ}} \subseteq \langle [v_1, \dots, v_r]\rangle_{P_v}$$

□

Complexity: Computing the sets $\mathcal{H}$ and $\mathcal{K}$ has complexity $\mathcal{O}(ph)$, and computing the enclosure by a zonotope has a complexity of $\mathcal{O}(n^2)$ according to proposition 2.4.1. Since the complexity of algorithm 1 according to [KA19] is $\mathcal{O}((2^p)^{\lfloor n/2\rfloor+1}+4^p(p+n))$. Thus, using 2.2, the overall complexity is $\mathcal{O}(2^{n^2})$. zono denotes the conversion of an SPZ to a zonotope.

For better understanding of the algorithm, we will demonstrate the enclosure by a V-polytope on our example 2.3.

Example 2.4.2. *We first need to compute our sets $\mathcal{H}$ and $\mathcal{K}$. Since only the fifth dependent generator has factors with an exponent greater than 1, its index is the only element in $\mathcal{H}$. Likewise, all other indices make up the set $\mathcal{K}$. Thus, we get*

$$\mathcal{H} = \{5\}$$
$$\mathcal{K} = \{1,2,3,4\}$$

With $\mathcal{H}$, we can now compute the zonotope using Proposition 2.4.1, by which we end up with the zonotope

$$\mathcal{Z} = \left\langle \begin{bmatrix}0.5\\0\end{bmatrix}, \begin{bmatrix}0.5\\0\end{bmatrix} \right\rangle_{\mathcal{Z}}.$$

Now we can also construct our SPZ:

$$\begin{aligned}\overline{\mathcal{SPZ}} &= \left\langle \begin{bmatrix}0.5 & 0.5 & -1 & 0 & -1\\ 0 & 0 & 1 & 2 & 1\end{bmatrix}, \begin{bmatrix}1 & 0.5\\ 1 & 0\end{bmatrix}, \begin{bmatrix}0 & 0 & 1 & 0 & 1\\ 0 & 0 & 0 & 1 & 1\end{bmatrix} \right\rangle_{\mathcal{SPZ}}\\ &= \left\{ \begin{bmatrix}1\\0\end{bmatrix} + \begin{bmatrix}-1\\1\end{bmatrix}\alpha_1 + \begin{bmatrix}0\\2\end{bmatrix}\alpha_2 + \begin{bmatrix}-1\\1\end{bmatrix}\alpha_1\alpha_2 + \begin{bmatrix}1\\1\end{bmatrix}\beta_1 + \begin{bmatrix}0.5\\0\end{bmatrix}\beta_2 \;\middle|\; \alpha_1,\alpha_2,\beta_1,\beta_2 \in [-1,1] \right\}\end{aligned}$$

By using $\begin{bmatrix}1\\0\end{bmatrix}$ as our center and renaming β_1,β_2 to α_3,α_4, we obtain our Z-representation with

$$\mathcal{P} = \left\langle \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}-1 & 0 & -1 & 1 & 0.5\\ 1 & 2 & 1 & 1 & 0\end{bmatrix}, ([1], [2], \begin{bmatrix}1\\2\end{bmatrix}, [3], [4]) \right\rangle.$$

Now we make use of algorithm 1. Since we have exactly 4 different factors, thus $p = 4$, we have to iterate through all of the $2^4 = 16$ possible assignments and compute the vectors. After all iterations, we end up with the set

$$\mathcal{K}_{alg} = \left\{ \begin{bmatrix}1\\-4\end{bmatrix}, \begin{bmatrix}2\\-4\end{bmatrix}, \begin{bmatrix}3\\-2\end{bmatrix}, \begin{bmatrix}4\\-2\end{bmatrix}, \begin{bmatrix}5\\0\end{bmatrix}, \begin{bmatrix}6\\0\end{bmatrix}, \begin{bmatrix}-1\\2\end{bmatrix}, \begin{bmatrix}0\\2\end{bmatrix}, \begin{bmatrix}1\\4\end{bmatrix}, \begin{bmatrix}2\\4\end{bmatrix} \right\}.$$

Finally, computing the convex hull of $\mathcal{K}_{alg}$ leads to the V-representation of our enclosing polytope

$$\mathcal{P} = \left\langle \begin{bmatrix}1\\-4\end{bmatrix}, \begin{bmatrix}2\\-4\end{bmatrix}, \begin{bmatrix}4\\-2\end{bmatrix}, \begin{bmatrix}6\\0\end{bmatrix}, \begin{bmatrix}-1\\2\end{bmatrix}, \begin{bmatrix}1\\4\end{bmatrix}, \begin{bmatrix}2\\4\end{bmatrix} \right\rangle_{\mathcal{P}_v}$$

. The figure 2.4 illustrates the enclosure.

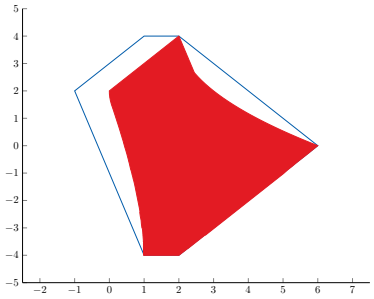


Figure 2.4: Enclosure of the SPZ in 2.3 by a polytope

2.4.3 Interval

We will treat the enclosure by an interval as a special case of a support function.

Definition 2.4.2. *(Support Function [GG08], Def. 1)*

Given a compact convex set $\mathcal{S} \subset \mathbb{R}^n$ and a direction $d \in \mathbb{R}^n$, the support function $s_{\mathcal{S}} : \mathbb{R}^n \to \mathbb{R}$ of $\mathcal{S}$ is defined as

$$s_{\mathcal{S}}(d) = \max_{x \in \mathcal{S}} d^T x. \tag{2.20}$$

Since SPZs are nonconvex in general, their support functions return an overapproximation. In order to compute them, we introduce the range bounding operation. For a function $\mathbb{R}^n \to \mathbb{R}$ and an interval $\mathcal{I} \subset \mathbb{R}^n$, the range bounding operation returns an overapproximation of the exact bounds of f on $\mathcal{I}$:

$$\mathtt{B}(f(x),\ \mathcal{I}) \supseteq \left[\min_{x \in \mathcal{I}} f(x), \max_{x \in \mathcal{I}} f(x)\right]. \tag{2.21}$$

Note that there are multiple methods of approximating the bounds. The tightness of the support function solely depends on the chosen method.

Proposition 2.4.5. *(Support Function for SPZs [KA21], Prop. 7)*

Given an SPZ $\mathcal{SPZ} = \langle G, G_I, E \rangle_{\mathcal{SPZ}}$ and a direction $d \in \mathbb{R}^n$, the support function

$$\begin{aligned} s_{\mathcal{SPZ}}(d) &= u + \sum_{j=1}^{q} |\overline{g}_{I(j)}| \\ &\text{with} \\ \langle \overline{g}, \overline{g_I}, E \rangle_{\mathcal{PZ}} &= d^T \otimes \mathcal{SPZ} \\ [l, u] &= \mathtt{B}\left(w(\alpha_1, \ldots, \alpha_p), [-\mathbf{1}^{(n,1)}, \mathbf{1}^{(n,1)}]\right) \\ w(\alpha_1, \ldots, \alpha_p) &= \sum_{i=1}^{h} \left(\prod_{k=1}^{p} \alpha_k^{E_{(k,i)}} \right) \overline{g}_{(i)} \end{aligned} \tag{2.22}$$

returns an overapproximation. Now the enclosure by an interval is a special case where we evaluate the support function for directions $\mathcal{D} = \{I_{n(\cdot,1)}, \ldots, I_{n(\cdot,n)}, -I_{n(\cdot,1)}, \ldots, -I_{n(\cdot,n)}\}$.

Proof. First, the SPZ is projected onto the direction d. Then, we divide the projection into two parts, one with dependent generators and one part with independent generators:

$$d^T \otimes \mathcal{SPZ} = \underbrace{\left\{ \sum_{i=1}^{h} \left(\prod_{k=1}^{p} \alpha_k^{E_{(k,i)}} \right) \overline{g}_{(i)} \middle| \alpha_k \in [-1,1] \right\}}_{\text{dependent part}} \oplus \underbrace{\left\{ \sum_{j=1}^{q} \beta_j \overline{g}_{I(j)} \middle| \beta_j \in [-1,1] \right\}}_{\text{independent part}}. \tag{2.23}$$

The bounds of the independent part can be calculated exactly with

$$\left\{ \sum_{j=1}^{q} \beta_j \overline{g}_{I(j)} \middle| \beta_j \in [-1,1] \right\} \equiv \left[-\sum_{j=1}^{q} |\overline{g}_{I(j)}|, \sum_{j=1}^{q} |\overline{g}_{I(j)}| \right]. \tag{2.24}$$

On the other hand, the interval $[l, u]$ overapproximates the bounds of the dependent part since the range bounding operation $\mathtt{B}$ returns an overapproxmation. Thus $s_{\mathcal{SPZ}}$ is an overapproximation of $\mathcal{SPZ}$. □

Complexity: Projecting the SPZ onto the direction d has a complexity of $\mathcal{O}(nh) + \mathcal{O}(nq)$ according to proposition 2.2.2. Thus, since all other operations have linear complexity and with 2.2, the overall complexity is $\mathcal{O}(n^2) + \mathcal{O}(\mathtt{B})$.

Example 2.4.3. *Again, we're using example 2.3. Since the dimension is* 2*, we have to compute the bounds of our enclosing interval by the directions*

$$\mathcal{D} = \{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \end{bmatrix} \}.$$

We will only go through the first direction since computation of the others is identical. For our range bounding operation, we use interval arithmetic. First, we project our SPZ onto the direction:

$$\begin{aligned} \overline{g} &= \{2, -1, 0, -1, 1\} \\ \overline{g}_I &= \{1\} \end{aligned}$$

Now we will compute the upper bound u using interval arithmetic:

$$\begin{aligned} [l, u] &= 2 - [-1,1] - [-1,1][-1,1] + [-1,1]^2 \\ &= [1,3] - [-1,1] + [-1,1] \\ &= [0,4] + [-1,1] \\ &= [-1,5] \end{aligned}$$

Thus, $u = 5$. Since $\overline{g}_I = \{1\}$, we obtain the result

$$s_{\mathcal{SPZ}}(\begin{bmatrix} 1 \\ 0 \end{bmatrix}) = 5 + 1 = 6.$$

Likewise, the results of the other directions are

$$s_{\mathcal{SPZ}}(\begin{bmatrix} 0 \\ 1 \end{bmatrix}) = 3,\ s_{\mathcal{SPZ}}(\begin{bmatrix} -1 \\ 0 \end{bmatrix}) = -2,\ s_{\mathcal{SPZ}}(\begin{bmatrix} 0 \\ -1 \end{bmatrix}) = -5.$$

Figure 2.5 illustrates the enclosing interval.

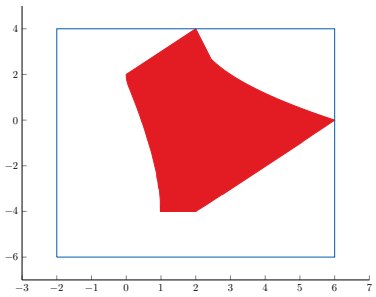


Figure 2.5: Enclosure of the SPZ in 2.3 by an interval

Chapter 3

Reachability Analysis

3.1 Hybrid Systems

[Sch19a, Sec. 3.1] *Discrete systems* are systems with a countable amount of discrete states. We often consider those systems whose state changes we can assume as instantaneous, e.g. a computer processing its instructions, which can often be modeled as a series of discrete steps. Although we can already represent a substantial amount of real-world situations with these, we need an even more expressive solution when we want to analyze the influence of such systems on dynamic environments. In contrast to these, the quantities of *continuous systems*, e.g. physical systems in general, evolve continuously over time. In computer science, *hybrid systems* are systems which show both discrete and continuous behavior, combining the continuous nature of a dynamic system with the discreteness of digital systems. Besides naturally hybrid physical systems, e.g. a ball bouncing off the ground, systems where a digital controller interacts with a dynamic environment are also examples for hybrid systems. Since the safety of such systems is often of interest, we want to verify whether they can reach undesirable states. Therefore, we try to model our system in order to analyze its reachability. More specifically, given a initial set, we want to find out which states are reachable and whether they violate any constraints. Thus, the reachability problem of a hybrid system $\mathcal{H}$ is whether starting from an initial set of states *Init* it is possible to reach another set of bad states P_{bad}.

3.2 Hybrid Automata

Hybrid automata [Hen00] are a popular choice when wanting to abstract from a system with mixed discrete-continuous behavior. They allow us to represent the discrete behavior of a digital controller by states as well as specifying the continuous change of the quantities. Let's consider a bouncing ball example [Sch19a, Sec. 4.2]. The hybrid system corresponding to this example can be seen on figure 3.1. We go through every single component one by one. Note that c is a parameter which needs to be set beforehand.

Hybrid automatons consist of locations or control nodes which represent the discrete part of the system (in this case, we can see that the system consists of only a single location (l)). On the very left, we can see a loose arrow pointing to the node,

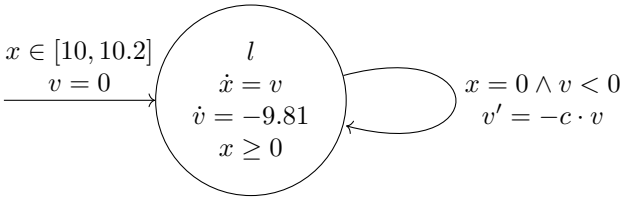


Figure 3.1: Bouncing ball system as a hybrid automaton

indicating it is the starting state. Above the arrow, we see the predicates $x \in [10, 10.2]$ and $v = 0$, which describe the *initial* starting set and we can also see that we got the *variables* x and v. Inside of the node we see l, which simply denotes the location's name, and $\dot{x} = v, \dot{v} = -9.81$, which describe the so called *flow* or *dynamics* of the system with differential equations, and $x \geq 0$, which represents an *invariant* for the loation. While in this location l, our variables change according to the specified flows, but at the same time the variable valuations must satisfy the invariants of the node. If the invariant cannot be satisfied by the current valuations of the state variables, we have to leave the location. On the right we can see a loop of the node with the predicates $x = 0 \wedge v < 0$. In general, an edge between two nodes describe a possible jump from one location to another under the condition that the specified *guards*, in this case the previously mentioned predicates, are satisfied by the current variable valuation. If such a jump is taken, the variables can be updated by the *resets*, in this case $v' = -c \cdot v$, where v' denotes the value after the change.

The automaton represents a bouncing ball dropped from a height between 10 and 10.2 distance units with a starting velocity of 0 distance units per second. Now inside of the node, we can see by $\dot{v}$ that the first derivative of v is -9.81, which represents the acceleration due to gravity on earth. Likewise $\dot{x}$, thus the first derivative of x, denoting the vertical height of the ball, is v, thus the velocity at which the ball falls. As long as the ball doesn't hit the ground, therefore $x > 0$, we stay inside of the node. When $x = 0$, we can and have to take the jump since not only are the guards satisfied, but also to not violate the invariant $x \geq 0$. Afterwards, the velocity will be reset with a factor of $-c$, illustrating the upward movement of the ball with a dampening factor of c. Formally, a hybrid automaton is defined as the following:

Definition 3.2.1. *(Hybrid Automata, [Jia23, Def. 2.1.1])*

A d-dimensional hybrid automaton H is described by a tuple $(Loc, Var, Con, Lab, Edge, Act, Inv, Init)$ where

- Loc is a finite set of locations or control nodes,
- $Var = \{x_0, \ldots, x_{d-1}\}$ is a finite ordered set of real-valued variables where $d-1$ denotes the dimensionality of the system,
- $Con : Loc \to 2^{Var}$ assigns a set of controlled variables to each location,
- Lab is a finite set of labels including the stutter label $\tau \in Lab$,
- $Edge \subseteq Loc \times Lab \times 2^{V^2} \times Loc$ is a finite set of edges (or transitions) including a τ-transition $(l, \tau, \mathrm{Id}, l)$ for each location $l \in Loc$ with $\mathrm{Id} = \{(v, v') \in V^2_{Var} \mid \forall x \in Con(l) : v'(x) = v(x)\}$, and where all edges with label τ are τ-transitions,

- Act is a function assigning a set of activities or flows $f : \mathbb{R}_{\geq 0} \to V$ to each location which are time-invariant meaning that $f \in Act(l)$ implies $(f + t) \in Act(l)$ where $(f + t)(t') = f(t + t')$ for all $t' \in \mathbb{R}_{\geq 0}$,
- Inv is a function assigning an invariant $Inv(l) \subseteq V$ to each location $l \in Loc$,
- $Init \subseteq \sum$ is a set of inital states,

with V denoting the set of all valuations $v : Var \to \mathbb{R}$, and $\sum = Loc \times V$ denoting the state space of H.

Definition 3.2.2. *(Operational semantics of hybrid automata [Jia23, Def 2.1.2])*

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

- Discrete step semantics

$$\frac{(l, a, (v, v'), l') \in Edge \quad v' \subseteq Inv(l')}{(l, v) \to^a (l', v')} \quad Rule_{discrete}$$

- Time step semantics

$$\frac{f \in Act(l) \quad f(0) = v \quad f(t) = v' \quad t \geq 0 \quad f([0,t] \subseteq Inv(l))}{(l, v) \to^t (l, v')} \quad Rule_{time}$$

An execution step $\to = \to^a \cup \to^t$ of $\mathcal{H}$ is either a discrete or a time step, and we give its transitive closure by $\to^*$. Let $(l, a, \mu, l') \in Edge$ be a transition of an automaton, then the transition relation $\mu \in V \times V$ defines that a discrete step can be taken if and only if $(v, v') \in \mu$. The corresponding guard $\{v \in V \mid \exists v' \in V : (v, v') \in \mu\}$, which is often modeled by first-order logic formulas, enables a transition if it evaluates to true.

Example 3.2.1. *Using the example from above, we can now formally define the system described in figure 3.1.*

- $Loc = \{l\}$
- $Var = \{x, v\}$
- $Con(l) = \{x, v\}$
- $Lab = \{\tau\}$
- $Edge = \{(l, \tau, \{(v_{val}, v'_{val}) \in V^2 \mid v_{val}(x) = 0 \wedge v_{val}(v) < 0\}, l)\}$
- $Act(l) = \{f : \mathbb{R}_{\geq 0} \to V \mid \forall t \in \mathbb{R}_{\geq 0} : f(t)(v) = -9.81t, f(t)(x) = -\frac{9.81}{2}t^2 + C\}$ *with* $C \in [10, 10.2]$
- $Inv(l) = \{v_{val} \in V \mid v_{val}(x) \geq 0\}$
- $Init = \{(l,v) \in \Sigma \mid v(x) \in [10, 10.2]\}$

Definition 3.2.3. *(Path [Sch19a, Def. 3.3])*

A path π of a hybrid automaton H is a finite or infinite sequence

$$\sigma_0 \to^{\tau_0} \sigma_1 \to^{e_1} \sigma_2 \to^{\tau_2} \dots$$

of states $\sigma_i \in \Sigma$ for $i \geq 0$, starting in a state $\sigma_0 = (l_0, v_0)$ with $v_0 \in Inv(l_0)$, which are connected through alternating time and discrete steps with $\tau_i \in \mathbb{R}_{\geq 0}, e_i \in Edge$. If a path starts in an initial state of H and thus $(l_0, v_0) \in Init$, we will call the path *initial*. A state σ is reachable in H if there is an initial path in H leading to it.

3.2.1 Classes of Hybrid Automata

Having defined hybrid automata, we can now take a look at several interesting subclasses [Sch19a, Sec. 3.2]. These subclasses restrict the predicates used to specify flows, invariants, guards and resets for the automata. We will shortly present some of them, going from least to most expressive:

- **Timed automata** are a simple subclass of hybrid automata where all variables $x \in Var$ are clocks measuring time. Thus, they all evolve continuously at a constant rate, hence the flow is restricted to the derivative $\dot{x} = 1$. Predicates can only compare variables with constants $c \in \mathbb{N}$ and discrete jumps either don't change the value of a variable or reset it to 0.
- **Rectangular automata** are a more expressive subclass, extending timed automata. They allow derivatives in the form of $\dot{x} \in [a, b]$ for flows and resets in the form of $x \in [a, b]$, a,$b \in \mathbb{N}$ for variables $x \in Var$, therefore only allowing rectangular sets.
- **Linear hybrid automata I** will still restrict flows to rectangular sets, but they allow predicates for invariants and guards in the form of linear expressions over the variables, thus in the form of $Ax \sim b$ with $\sim \in \{<, \leq, =, \geq, >\}$, while also letting resets use affine transformation $Ax + b$, with $A \in \mathbb{R}^{d \times d}, b \in \mathbb{R}^d$.
- **Linear hybrid automata II** extend LHA I by also allowing linear expressions over the variables for the flows with which dynamics of systems with linear ODEs can be specified. We will put our focus on this subclass of hybrid automata in this work.

3.3 Reachability Analysis for Hybrid Systems

As mentioned before, the safety of hybrid systems is of big interest. [Sch19a, Sec. 3.3] We therefore try to compute the set of reachable states in our system and check for a set of undesirable states P_{bad}. In *reachability analysis*, given a hybrid automaton H, we want to verify whether starting from any of its initial states, the set of bad states P_{bad} is reachable. Hence, we want to compute whether $Reach_H \cap P_{bad} = \emptyset$ with $Reach_H$ denoting the set of reachable states in H. While there are several other methods of solving this problem, we will focus on *forward reachability analysis*, an iterative method where starting from an initial set of states, we iteratively compute the successor states until we either reach the set of bad states or fixed-point has been found. The figure 3.3 shows the scheme for a forward reachability analysis algorithm.

To avoid non-termination, we usually restrict the reachability analysis by setting bounds for the time duration and limiting the number of jumps. Here, we will set a

Algorithm 2 Forward reachability analysis algorithm [Sch19a, Alg. 1]

Input: Set of initial states I
Output: Set of reachable states R
$R \leftarrow I$
$R_{new} \leftarrow R$
while $R_{new} \neq \emptyset$ **do**
  $R_{new} \leftarrow \texttt{computeReach}(R_{new}) \backslash R$
  $R \leftarrow R \cup R_{new}$
end while
return R

time horizon T as an upper limit on uninterrupted time evolution without any jumps. In this work, we will take a closer look at *flowpipe-construction-based* methods, a specific kind of forward reachability analysis. In flowpipe construction, successor states are computed by dividing the time horizon into smaller time steps δ. We will also restrict the number of discrete transitions with a jump depth J. For each step, starting from a set of initial states, the set of trajectories, or the *flowpipe*, is overapproximated by a single state set, for which we usually use state set representations such as intervals, zonotopes or polytopes.

3.4 Flowpipe Construction

In this section, we expand upon the flowpipe-construction-based reachability analysis [Jia23, Sec 2.1.2], [Sch19a, Sec. 3.4]. Given a Linear Hybrid Automata II H, each location specifies the dynamics of the system by a system of linear differential equations

$$\dot{x}(t) = Ax(t) \tag{3.1}$$

with $x = (x_0, \ldots, x_{d-1}) \in Var$ where $A \in \mathbb{R}^{d \times d}$ is a coefficient matrix denoting the flow at time t. In a *non-autonomous system*, we extend the dynamics by a time-dependent function $u(t)$, which is usually assumed to be from a bounded domain U. This function $u(t)$ describes the external inputs influencing the system, by which we obtain dynamics in the form of

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{3.2}$$

with $B \in \mathbb{R}^{d \times d}$. The solutions of equation 3.1 are of the form

$$x(t) = e^{tA}x(0), \tag{3.3}$$

which means that the state $x(t)$ is reachable at time point t from the initial state $x(0)$ when following the flow from A. We can see that e^{tA} depends on the time t and the current location l for which the time successors need to be computed. Note that the solution 3.3 can be directly extended to sets of variable valuations N such that

$$N_t = e^{tA}N_0. \tag{3.4}$$

Although it is possible to compute the set of reachable states for a precies time point t with equation 3.4, we cannot compute the set of reachable states for a time interval.

To solve this issue, several methods have been developed to overapproximate the error α between an approximation Ω of the set of reachable states for the time interval $[0, \delta]$ and the actual set of reachable states

$$\mathcal{R}_{[0,\delta]} = \{(l, v) \mid v = e^{tA}x_0, t \in [0, \delta], x_0 \in N_0\}.$$

In [Gir05], an approach was presented by overapproximating the error α through the approximation of the Hausdorff-distance between $\Omega' = \texttt{convHull}(N_0 \cap N_\delta)$ and $\mathcal{R}_{[0,\delta]}$.

Definition 3.4.1. *(Hausdorff Distance [Sch19a, Def. 3.6])*

The *Hausdorff distance* between two sets $A, B \subseteq \mathbb{R}^d$ is defined as

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

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

Given an initial state $x \in \mathbb{N}_0$ and the state $r = e^{\delta A}x$ reached from x at time point δ, their connecting line segment is

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

By computing the union of all line segments for all initial states $x \in N_0$, we obtain the convex hull of the sets N_0 and N_δ. It holds that the error between a line segment and the actual trajectory $\zeta_x(t) = e^{tA}x$ is

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

As shown in [Gir05], this error can be approximated by using Taylor's theorem, cutting off the Taylor-expansion after degree two and overapproxmating the remainder, hence

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

Since α is an upper bound for the error, we can safely overapproximate $R_{[0,\delta]}$ by bloating the convex hull of the two sets N_0 and N_δ with a ball $\mathcal{B}_\alpha$ of radius α:

$$\Omega_0 = \texttt{convHull}(N_0, N_\delta) \oplus \mathcal{B}_\alpha$$

with $\oplus$ denoting the Minkowski sum, by which we obtain a uniform bloating of the set. Having obtained the first flowpipe segment, it is now possible to compute the following segments using the same step size δ. For autonomous linear hybrid systems, we can obtain such a sequence of segments using

$$\Omega_{i+1} = e^{\delta A}\Omega_I$$

for a location l with flow matrix A, where each segment is a safe overapproximation of the time interval $[i\delta, (i+1)\delta]$. For non-autonomous systems, a second bloating step for each segment Ω_i is required, see [LG09].

By this, we are now able to compute the flowpipe segments Ω_i, $i = 0, \ldots, k-1$ for a flow matrix A and a time horizon T using the fixed time step size $\delta = \frac{T}{k}$ and starting from a defined set of initial states (l, N_0)..

Since a locations invariant has to be satisfied at all times and $\Omega_i = (l, N_i)$ overapproximates the actual set of reachable states in the corresponding time interval, we can stop computing further segments if $N_i \cap Inv(l) = \emptyset$.

Likewise, for each outgoing transition $e = (l,g,r,l') \in Edge$ of the current location l and for each flowpipe segment $\Omega_i = (l, N_i)$, it must hold that $N_i' = N_i \cap g \neq \emptyset$, thus the jump can be taken if it can be taken by any state in the segment. For all non-empty segments $\Omega_i = (l, N_i')$ with $N_i' \neq \emptyset$, the reset function r is applied to discretely update the variable valuations, by which we obtain

$$\Omega_i'' = (l, N_i'') = (l, r(N_i')).$$

We usually use an affine transformation for linear hybrid systems, thus $N_i'' = A \cdot N_i' + b$ for a matrix A and translation vector b.

Similiarly to before, we will also check whether N_i'' satisfies the invariant from the target location l' by verifying $N_i''' = N_i'' \cap Inv(l') \neq \emptyset$. We then go on with computing the flowpipe segments in l' for all non-empty $\Omega_i''' = (l', N''')$.

The algorithm 3.4 demonstrates how to compute a flowpipe construction for reachability analysis.

Algorithm 3 Bounded flowpipe-construction-based reachability analysis [Sch19a, Alg. 2]

Input: Hybrid automaton

$$H = (Loc, Var, Lab, Flow, Inv, Edge, Init)$$

Output: An over-approximation of the reachable states in H

$Q \leftarrow$ Init
$R \leftarrow \emptyset$
while $Q \neq \emptyset$ **do**
  $\Omega \leftarrow$ `computeFirstSegment(getElement(`Q`))`
  while *not* `timeBoundReached()` **do**
    $\Omega \leftarrow \Omega \cap Inv(l)$
    $R \leftarrow R \cup \Omega$
    if *not* `jumpBoundReached()` **then**
      for $(l,g,r,l') \in Edge$ **do**
        `addElement(`$Q, r(\Omega \cap g) \cap Inv(l')$`)`
      end for
      $\Omega \leftarrow$ `letTimePass(`Ω`)`
    end if
  end while
end while
return R

We realize that for the computation of the reachable states in the hybrid system, we perform many operations on sets of states. Hence, we need an efficient representation for these while still not being too overapproximative, making SPZs a possible solution.

3.5 Splitting Algorithm

During flowpipe construction, we often check whether our current state sets intersection with a halfspace is empty or not (for example the intersection with a guard or the invariant of a given location). Since the operation for intersection checking between an SPZ and a halfspace is not supported, we have to convert them to a different set representation in which this operation is supported. In our case, it is the zonotope state set representation. Because the conversion leads to an overapproximation of the SPZ, the result may also be incorrect in some cases. Thus, for a more precise outcome, we need to have a tighter overapproximation. One way of doing this is to split the SPZ into two smaller SPZs such that their union is equivalent to the inital one, and then convert each of those to a zonotope. This was also implemented in HyPro and it can be turned on and off with a flag. An example for such a splitting can be seen in figure 3.5, where the subfigure 3.2a shows an SPZ where the intersection check with the given half space would return a false value, while 3.2b shows the same SPZ using the splitting algorithm, correctly not intersecting with the half space.

Proposition 3.5.1. *(Split [SV22], p. 501, Prop. 1)*

Given an SPZ $\mathcal{SPZ} = \langle G, G_I, E \rangle_{\mathcal{SPZ}}$ and an index $r \in \{1, \dots, p\}$ of one of the dependent factors, two SPZs $\mathcal{SPZ}_1, \mathcal{SPZ}_2$ satisfying $\mathcal{SPZ}_1 \cup \mathcal{SPZ}_2 = \mathcal{SPZ}$ are given by

$$\mathcal{SPZ}_1 = \left\langle [\hat{G}_1^{(1)} \dots \hat{G}_h^{(1)}], G_I, [\hat{E}_1 \dots \hat{E}_h] \right\rangle_{\mathcal{SPZ}}$$
$$\mathcal{SPZ}_2 = \left\langle [\hat{G}_1^{(2)} \dots \hat{G}_h^{(2)}], G_I, [\hat{E}_1 \dots \hat{E}_h] \right\rangle_{\mathcal{SPZ}}$$

with

$$\hat{E}_i = \begin{bmatrix} E_{(\{1,\dots,r-1\},i)} & E_{(\{1,\dots,r-1\},i)} & \cdots & E_{(\{1,\dots,r-1\},i)} & E_{(\{1,\dots,r-1\},i)} \\ 0 & 1 & \dots & E_{(r,i)} - 1 & E_{(r,i)} \\ E_{(\{r+1,\dots,p\},i)} & E_{(\{r+1,\dots,p\},i)} & \cdots & E_{(\{r+1,\dots,p\},i)} & E_{(\{r+1,\dots,p\},i)} \end{bmatrix}$$
$$\hat{G}_i^{(k)} = \left[b_{i,0}^{(k)} \cdot G_{(\cdot,i)} \dots b_{i,E_{(r,i)}}^{(k)} \cdot G_{(\cdot,i)} \right]$$
$$b_{i,j}^{(1)} = 0.5^{E_{(r,i)}} \binom{E_{(r,i)}}{j}$$
$$b_{i,j}^{(2)} = -0.5^{E_{(r,i)}} (2(E_{(r,i)} \bmod 2) - 1) \binom{E_{(r,i)}}{j}$$

Proof. The splitting is essentially done by substituting the dependent factor α_r with two new dependent factors $\alpha_{r,1}$ and $\alpha_{r,2}$.

$$\begin{aligned} & \{\alpha_r \mid \alpha_r \in [-1,1]\} \\ = & \{0.5(1+\alpha_{r,1}) - 0.5(1+\alpha_{r,2}) \mid \alpha_{r,1}, \alpha_{r,2} \in [-1,1]\} \\ = & \{0.5(1+\alpha_{r,1}) \mid \alpha_{r,1} \in [-1,1]\} \cup \{-0.5(1+\alpha_{r,2}) \mid \alpha_{r,2} \in [-1,1]\} \end{aligned}$$

Now inserting this into the definition of SPZs leads to

$$\begin{aligned}\mathcal{SPZ} &= \left\{ \sum_{i=1}^{h} \left(\prod_{k=1}^{p} \alpha_k^{E_{(k,i)}} \right) G_{(\cdot,i)} + \sum_{j=1}^{q} \beta_j G_{I(\cdot,j)} \;\middle|\; \alpha_k,\beta_j \in [-1,1] \right\} \\ &= \underbrace{\left\{ \sum_{i=1}^{h} \left(\prod_{k=1,k\neq r}^{p} \alpha_k^{E_{(k,i)}} \right) (\frac{1+\alpha_{r,1}}{2})^{E_{(r,i)}} G_{(\cdot,i)} + \sum_{j=1}^{q} \beta_j G_{I(\cdot,j)} \;\middle|\; \alpha_k,\beta_j,\alpha_{r,1} \in [-1,1] \right\}}_{\mathcal{SPZ}_1} \\ &\cup \underbrace{\left\{ \sum_{i=1}^{h} \left(\prod_{k=1,k\neq r}^{p} \alpha_k^{E_{(k,i)}} \right) (\frac{1+\alpha_{r,2}}{-2})^{E_{(r,i)}} G_{(\cdot,i)} + \sum_{j=1}^{q} \beta_j G_{I(\cdot,j)} \;\middle|\; \alpha_k,\beta_j,\alpha_{r,2} \in [-1,1] \right\}}_{\mathcal{SPZ}_2}\end{aligned}$$

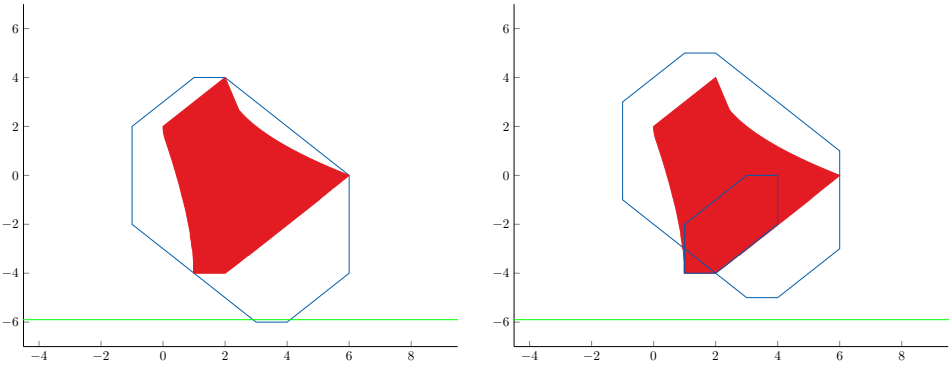
Now with

$$\begin{aligned}(\frac{1+\alpha_{r,1}}{2})^{E_{(r,i)}} &= b_{i,0}^{(1)} + b_{i,1}^{(1)}\alpha_{r,1} + \cdots + b_{i,E_{(r,i)}}^{(1)} \alpha_{r,1}^{E_{(r,i)}} \\ (\frac{1+\alpha_{r,2}}{-2})^{E_{(r,i)}} &= b_{i,0}^{(2)} + b_{i,1}^{(2)}\alpha_{r,2} + \cdots + b_{i,E_{(r,i)}}^{(2)} \alpha_{r,2}^{E_{(r,i)}}\end{aligned}$$

we obtain our equations from above. □

Note that the splitting of an SPZ is not exact, meaning that the resulting SPZs might overlap. One can reduce the amount of overlapping by their choice of which dependent factor to split. To minimize this, we choose the index r which maximizes the following heuristic:

$$\max_{r\in\{1,\ldots,p\}} \sum_{i=1,E_{(r,i)}>1}^{h} (1 - 0.5^{(E_{(r,i)})}) \|G_{(\cdot,i)}\|_2$$



(a) Conversion to zonotope without splitting (b) Conversion to zonotope with splitting

3.6 Experimental Results

In this section, we will present the experiments conducted in order to gather results validating our analysis. We will use the flowpipe construction implemented in HyPro

[Sch19a] to compute the reachability of multiple examples for hybrid systems. Furthermore, we want to emphasize the performance of the newly presented state set representation, thus we will compare SPZs to other widely used representations. Additionally, we will also look at the performance of SPZs when the splitting algorithm is applied, allowing us to determine whether the computational overhead is worth the increased precision. The following benchmarks were used:

- [Sch19a] A thermostat model where the temperature of a room is kept between 17 and 23 degrees celsius. The heater is turned on and off according to the measured temperature, heating the room when it is turned on as given by the differential equation, or letting it cool down. The automaton for this model can be seen in figure 3.3.

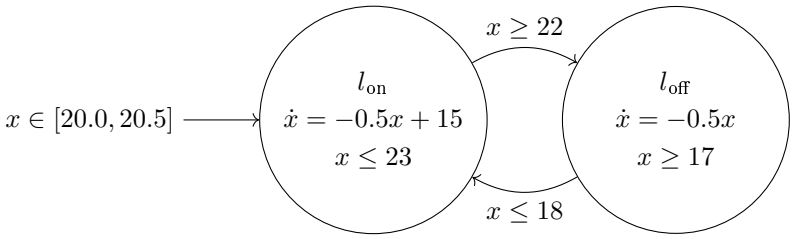


Figure 3.3: Thermostat system

- A rod reactor system [SÁMK17],[ACH+95]. This system models a simple reactor of a nuclear power plant where cooling rods and fuel rods are used to heat water in order to power turbines. The cooling rods can be lowered in between the fuel rods for the purpose of regulating the water temperature. The model describes the reactor temperature in dependency of two different cooling rods that can be lowered individually while also having different cooling properties because they are made out of different materials. Without any cooling rods, the temperature rises with $\dot{x} = 0.1x - 50$. With the first rod lowered, it falls by $\dot{x} = 0.1x - 56$ and with the second one, by $\dot{x} = 0.1x - 60$. Both cooling rods cannot be lowered at the same time. While the controller is able to lower any of the cooling rods as soon as the temperature exceeds 550 degrees celsius, after using one of them, it cannot lower it again for 20 time units. If the temperature reaches 550 degrees celsius but the controller is not able to lower any of the cooling rods, it will lead to a shutdown. The automaton for this system can be seen in figure 3.4.

- The bouncing ball introduced in section 3.2 with a dampening factor of $c = 0.75$ for a step size of 0.01 and a total time of 3.

The constructed flowpipes for the bouncing ball model with different state set representations can be seen in figures 3.5a and 3.5b.

The table 3.1 shows the runtime of different state set representations on the previously mentioned models. The reachability was computed 100 times in order to compensate for fluctuation in runtimes by calculating the average. The following results were carried out on an Intel Core i5-1135 with 8 GiB RAM. Note that when testing SPZs with splitting enabled, a split depth of 1 was set.

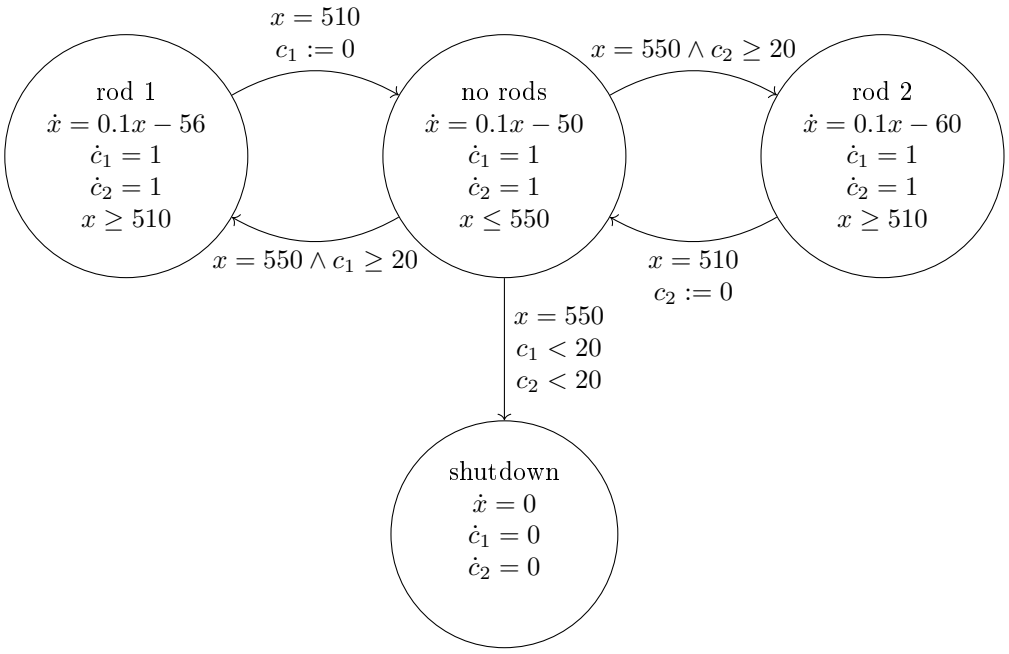


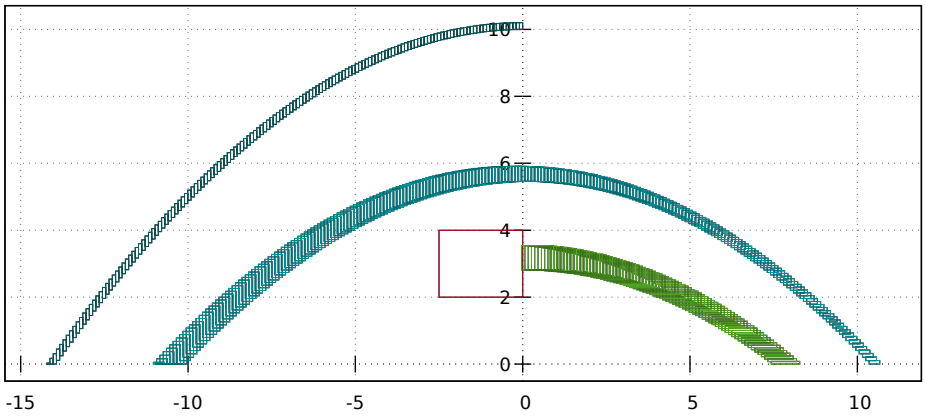
Figure 3.4: Rod reactor system

State Set Representation	Thermostat	Rod Reactor	Bouncing Ball
Interval	0.94ms	17.15ms	3.91ms
Zonotope	7.92ms	149.47ms	23.81ms
SPZ	10.93ms	220.75ms	47.95ms
SPZ with Splitting	22.83ms	871.12ms	179.62ms

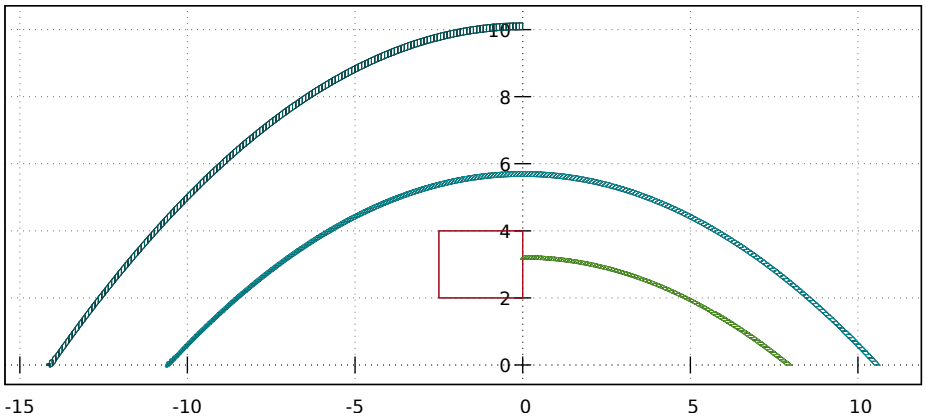
Table 3.1: Benchmark Results

The results show that while computations on intervals are still way faster, SPZs are not that much slower compared to zonotopes. Additionally, we can see that enabling the splitting algorithm makes the runtime of SPZs significantly worse, with a split depth of only 1. While the higher precision does sound good at first, it almost never happens that splitting the SPZ detects a false halfspace intersection since the zonotope enclosures are tight enough in most cases.

While the runtime on SPZs might not be better, they do lead to an improvement in the sense of tightness and thus overapproximation error compared to the others. For instance, figures 3.5a and 3.5b show the flowpipe construction of the bouncing ball model. It is apparent that the computation on intervals lead to a significantly bigger overapproximation error and thus a worse result compared to SPZs (note that the plotting of the SPZs was done by converting them to zonotopes before plotting).



(a) Flowpipe with intervals



(b) Flowpipe with SPZs

Figure 3.5: Flowpipe segments for the bouncing ball model

Chapter 4

Conclusion

4.1 Summary

In this work, we have taken a look at *Sparse Polynomial Zonotopes* as a new state set representation for reachability analysis. We defined them formally and also presented how to perform commonly used operations such as the Minkowski sum or convex hull on them. Furthermore, we have also seen their flexibility, allowing us to convert them to other common representations and vice versa seemlessly. Additionally, we have also analyzed the time complexity of previously mentioned operations, discovering that most of them can be computed efficiently which makes them practical for real use. After that, we introduced reachability analysis and what we need SPZs (and other state set representations) for. We have shortly shown what hybrid systems are and pointed out their characteristics. Then, we have seen that we are able to model them using hybrid automata, defining and elaborating on them as they are the base of further analysis. We explained the reachablity problem for hybrid systems and how to solve it, going further into detail by looking at flowpipe-construction-based methods. We described how we can compute an overapproximative solution to this problem by constructing a flowpipe step-by-step. We realized that we can reduce the overapproximation error for a SPZs caused by the conversion to zonotopes by the splitting of the polynomial zonotope, allowing for an even more precise result at the cost of computational effort. Lastly, we ran experiments on SPZs, analyzing how they compare to other state set representations on various benchmarks.

We can conclude that Sparse Polynomial Zonotopes are very expressive, allowing for a very precise and tight representation of sets. Since they support many basic operations as well as allowing for the conversion to other commonly used representations and vice versa, with most operations having polynomial time complexity, they are very practical and allow for real usage in practice. As we can see in the benchmarks, their performance in reachability analysis have shown that they are still very efficient in comparison to other representations, e.g. their runtime not being much slower compared to zonotopes. We can obtain an even more precise result by the usage of the splitting algorithm, although it does lead to a significant worsening of the runtime, making it rather impractical for use. Thus, in conclusion, Sparse Polynomial Zonotopes are a very expressive and versatile alternative for reachability analysis, although one has to judge whether the more precise result is worth the computational overhead.

4.2 Future work

In the current implementation of HyPro, SPZs are plotted by converting them to zonotopes and then plotting the result. Since the conversion usually leads to an overapproximation error, the resulting graphs are inaccurate. To improve this, one could iteratively use the splitting algorithm as presented in section 3.5, getting a tighter enclosure by splitting the SPZ multiple times. With n splits, we obtain 2^n smaller SPZs which we can then convert to zonotopes for plotting, leading to a more accurate solution. The more splits are done, the smaller each SPZ is and thus the better the resulting plot will be, although the time needed for computation grows exponentially with respect to the amount of splits.

Furthermore, various other operations on Sparse Polynomial Zonotopes have been presented in [Koc22] which are not implemented in `HyPro` yet. For instance, the work shows that it's possible to compute the intersection of two SPZs as well as checking whether an SPZ is contained in another one. Additionally, it is proven that we can also check whether an interval is contained in an SPZ.

On top of that, another possible improvement could be the early stopping of the splitting algorithm when using a splitting depth greater than 1. As of right now, `HyPro` will perform a depth-first search, splitting the SPZ n times and checking for the resulting 2^n SPZs whether their enclosure by zonotopes satisfies a given half-space. It is now clear that as soon as one of them does not satisfy the half-space, the result would be *false* and the algorithm can be terminated. Implementing this behaviour could be benificial for the computational complexity of said method.

Last but not least, one could also take a look at different heuristics for the choice of which dependent factor to split on which might affect and improve the amount of overlapping of the resulting polynomial zonotopes.

Bibliography

[ACH+95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. volume 138, pages 3–34, 02 1995.

[Alt10] Matthias Althoff. Reachability analysis and its application to the safety assessment of autonomous cars. 2010.

[Alt13] Matthias Althoff. Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, HSCC '13, page 173–182, New York, NY, USA, 2013. Association for Computing Machinery.

[AP01] D. Alevras and M.W. Padberg. Linear optimization and extensions: Problems and solutions. Universitext (Berlin. Print). Springer Berlin Heidelberg, 2001.

[ERNF11] Andreas Eggers, Nacim Ramdani, Nedialko Nedialkov, and Martin Fränzle. Improving the sat modulo ode approach to hybrid systems analysis by combining different enclosure methods. volume 14, pages 172–187, 11 2011.

[GG08] Antoine Girard and Colas Le Guernic. Efficient reachability analysis for linear systems using support functions. volume 41, pages 8966–8971, 2008. 17th IFAC World Congress.

[Gir05] Antoine Girard. Reachability of uncertain linear systems using zonotopes. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control*, pages 291–305, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[GKKZ03] B. Grünbaum, V. Kaibel, V. Klee, and G.M. Ziegler. *Convex Polytopes*. Graduate Texts in Mathematics. Springer, 2003.

[Hen00] Thomas A. Henzinger. The theory of hybrid automata. In M. Kemal Inan and Robert P. Kurshan, editors, *Verification of Digital and Hybrid Systems*, pages 265–292, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[Jia23] Ruoran Gabriela Jiang. Verfifying ai-controlled hybrid systems. 03 2023.

[JKDW01] Luc Jaulin, Michel Kieffer, Olivier Didrit, and Eric Walter. *Applied Interval Analysis with Examples in Parameter and State Estimation, Robust Control and Robotics*. 08 2001.

[KA19] Niklas Kochdumper and Matthias Althoff. Representation of polytopes as polynomial zonotopes. 10 2019.

[KA21] Niklas Kochdumper and Matthias Althoff. Sparse polynomial zonotopes: A novel set representation for reachability analysis. volume 66, pages 4043–4058, Sep. 2021.

[KKW+23] Niklas Kochdumper, Hanna Krasowski, Xiao Wang, Stanley Bak, and Matthias Althoff. Provably safe reinforcement learning via action projection using reachability analysis and polynomial zonotopes. *IEEE Open Journal of Control Systems*, 2:79–92, 2023.

[Knu98] D.E. Knuth. The art of computer programming: Sorting and searching, volume 3. Pearson Education, 1998.

[Koc22] Niklas Kochdumper. *Extensions of Polynomial Zonotopes and their Application to Verification of Cyber-Physical Systems (Erweiterungen von Polynomiellen Zonotopen und deren Anwendung für die Verifikation von Cyber-Physischen Systemen)*. PhD thesis, Technical University of Munich, Germany, 2022.

[KSA17] Anna-Kathrin Kopetzki, Bastian Schurmann, and Matthias Althoff. Methods for order reduction of zonotopes. pages 5626–5633, 12 2017.

[KSAB23] Niklas Kochdumper, Christian Schilling, Matthias Althoff, and Stanley Bak. *Open- and Closed-Loop Neural Network Verification Using Polynomial Zonotopes*, page 16–36. Springer Nature Switzerland, 2023.

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

[LKB23] Ertai Luo, Niklas Kochdumper, and Stanley Bak. Reachability analysis for linear systems with uncertain parameters using polynomial zonotopes. pages 1–12, 05 2023.

[MHZ+23] Jonathan Michaux, Patrick Holmes, Bohao Zhang, Che Chen, Baiyue Wang, Shrey Sahgal, Tiancheng Zhang, Sidhartha Dey, Shreyas Kousik, and Ram Vasudevan. Can't touch this: Real-time, safe motion planning and control for manipulators under uncertainty, 2023.

[RN11] Nacim Ramdani and Nedialko Nedialkov. Computing reachable sets for uncertain nonlinear hybrid systems using interval constraint-propagation techniques. *Nonlinear Analysis: Hybrid Systems*, 5:149–162, 05 2011.

[SÁMK17] Stefan Schupp, Erika Ábrahám, Ibtissem Ben Makhlouf, and Stefan Kowalewski. Hypro: A c++ library of state set representations for hybrid systems reachability analysis. In Clark Barrett, Misty Davies, and Temesghen Kahsai, editors, *NASA Formal Methods*, pages 288–294, Cham, 2017. Springer International Publishing.

[Sch19a] Stefan Schupp. State set representations and their usage in the reachability analysis of hybrid systems. pages 1 Online–Ressource (217 Seiten) : Illustrationen, Diagramme, Aachen, 2019. Veröffentlicht auf dem Publikationsserver der RWTH Aachen University; Dissertation, RWTH Aachen University, 2019.

[Sch19b] Stefan Schupp. State set representations and their usage in the reachability analysis of hybrid systems. pages 1 Online–Ressource (217 Seiten) : Illustrationen, Diagramme, Aachen, 2019. Veröffentlicht auf dem Publikationsserver der RWTH Aachen University; Dissertation, RWTH Aachen University, 2019.

[SNA17] Stefan Schupp, Johanna Nellen, and Erika Abraham. Divide and conquer: Variable set separation in hybrid systems reachability analysis. volume 250, page 1–14. Open Publishing Association, July 2017.

[SV22] Sharon Shoham and Yakir Vizel, editors. *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I*, volume 13371 of *Lecture Notes in Computer Science*. Springer, 2022.

[Tiw08] Hans Raj Tiwary. On the hardness of computing intersection, union and minkowski sum of polytopes. *Discrete & Computational Geometry*, 40(3):469–479, October 1 2008.

[Zie12] G.M. Ziegler. *Lectures on Polytopes*. Graduate Texts in Mathematics. Springer New York, 2012.