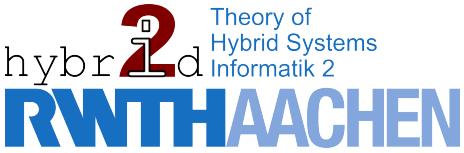

Bachelor of Science Thesis

A zonotope library for hybrid systems reachability analysis.

Maik Glatki

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

Advisor:
Xin Chen

January 15th 2014

Abstract

In reachability analysis, the forward fixed-point approach for solving the reachability problem, a central problem of model checking, has become increasingly popular. Forward fixed-point analysis utilizes over-approximations of state sets to decide whether a system can reach undesirable or desirable states. Efficient state set representations, such as zonotopes, are used to compute certain operations on the state sets. However, while there is a number of specialized libraries available, none of them are free or support the techniques used in our implementation.

In this thesis we introduce an extendable open-source software library for the zonotope representation of (state) sets. Combining an abstract approach to data types with object-oriented extendability, it provides an easy-to-use and reliable tool for efficient state set operations and provides the groundwork for a larger, more extensive library with a wider set of representation types.

Erklärung

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

Maik Glatki
Aachen, den 15. Januar 2014

Contents

Chapter 1

Introduction

Model checking is an important technique to verify software systems. With an increasing number of software systems in everyday life, and software systems controlling machines such as elevators, mobile phones, flood control systems [TWC01] or navigation system, it is more important than ever to ensure the safety of these software system, as a system failure could endanger or impair human lives directly.

Model checking allows to check whether certain properties hold on a modelled system. Model checking, initially a technique to analyze only discrete-time systems, has quickly been adapted for the analysis of hybrid systems [ACH+95].

A central problem in the model checking of software systems is to check, whether a system reaches certain states, i.e., whether the system can reach a set of certain undesired or desired states. This *reachability problem* is central to model checking. It is the question, whether for two given states s, s' in a system, s' is reachable from s.

Reachability analysis is still a major research topic in hybrid systems. One approach is to compute (over-approximations of) state sets in the form of certain *geometric objects* like, e.g., polytopes [Zie95], zonotopes [BEG+95], or ellipsoids [KV00]. For the reachability analysis, we can use a forward fixed-point approach, starting from an (over-approximation of the) initial state set and computing one-step successors iteratively until either a fixed-point is found or a given maximal computation depth is reached. To implement these computations, we need certain operations on the state set. Such operations on state sets are:

- The computation of the union and the intersection of two sets.
- The membership test, i.e., testing whether a given point belongs to a set.
- Linear transformations on a set.
- The emptiness test.

State sets may have an infinite number of elements. Therefore, we need finite representations for them, which can be stored and worked on in the reachability analysis algorithm.

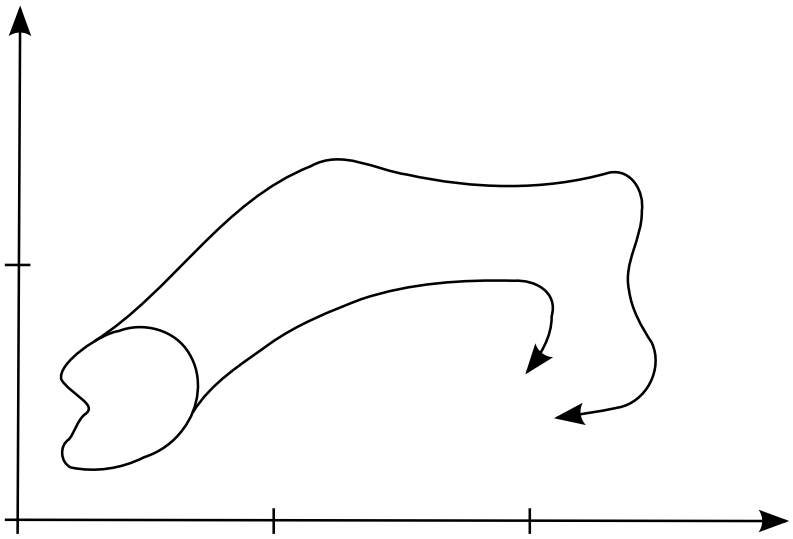

Figure 1.1: Illustration of the initial state set and the reachable state set.

The types of the geometric representations of sets have a large influence on the efficiency and precision of the reachability analysis. Certain geometric representations allow for more efficient analysis, but may have certain drawbacks, e.g., leading to a larger over-approximation. Figure 1.1 and Figure 1.2 illustrate the over-approximation of the reachable sets on a two-dimensional example using rectangles.

In this work, we focus on a certain geometric form to over-approximate state sets: *zonotopes*. Zonotopes [BEG+95] have an elegant definition and can be intuitively defined by a *center point* and the Minkowski sum of finitely many line segments called *generators*. The full definition for zonotopes is given in Chapter 2.

The aim of this work is to provide a transparent, efficient and extendable library for the representation of zonotopes and operations on them. The library is written in C++. Furthermore, the library is released under the GNU General Public License (GPL) [Fre], a free software license and can thus be used by third parties for future research. While there are some libraries for zonotope operations, they either have a non-free licensing system, or are limited to certain data types.

We provide an open-source C++ library that supports the usage of zonotopes in reachability analysis. In addition, the library features an extendable library for matrix computations, used by the zonotope library. The zonotope library features zonotope creation and operations on them, as well as some operations with other geometric objects, most importantly functionality for zonotope-hyperplane intersection. Although some analysis tools use zonotopes, to our best knowledge, there is no open-source C++ library providing these functionalities.

We first introduce all necessary preliminaries and definitions in Chapter 2. In Chapter 3, we discuss a zonotope-hyperplane intersection algorithm which we

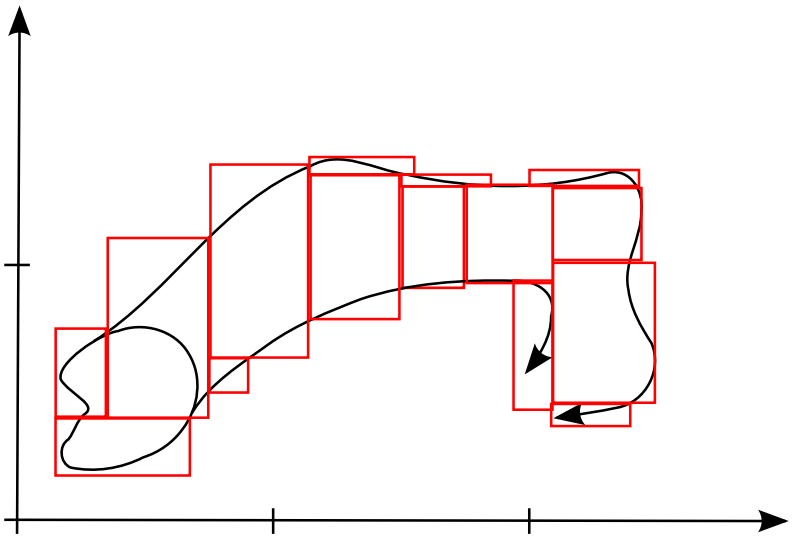

Figure 1.2: An exemplary over-approximation of the reachable state set and the initial state set in Figure 1.1 using boxes.

provided with our library, while details specific to our implementation be presented in Chapter 4, together with some experimental results. In the closing Chapter 5, we will draw conclusions.

Chapter 2

Preliminaries

In this chapter, we discuss the necessary preliminaries to understand this thesis. We formally introduce a number of geometric objects, most importantly hyperplanes, polytopes and zonotopes. We describe the necessary operations on geometric objects. At last, we discuss their implications for the the reachability analysis of continuous systems.

Definitions for sets of points can be arbitrarily complex, which is why some formalization is necessary to define representations for and operations on these sets. As sets considered in the reachability analysis of hybrid systems often have an infinite number of points, (possibly over-approximative) finite representations for these sets are needed. A set of points can be represented (and usually over-approximated) by a wide range of representations.

Furthermore, we need to define efficient operations for such representations, to allow for transformations such as the rotation or shearing of a geometric object. In this chapter, we will discuss these operations.

The geometrical structures mentioned in this thesis are defined for general fields and vector spaces. While our implementation does support these generalizations, we limit ourselves to d-dimensional Euclidean spaces in this chapter, commonly referred to as $\mathbb{R}^d$. For elements in $\mathbb{R}^d$, $\mathbb{R}^{m\times n}$ and similar vector spaces, we will use the normal dot product and matrix multiplication. We define the dot product as follows:

Definition 1. *(Dot product). The* dot product $v \cdot w$*, with* $v,w \in \mathbb{R}^d$ *is defined as:*

$$v \cdot w = \sum_{i=1}^{n} v_i w_i.$$

While other underlying algebraic structures may be used in model checking, the Euclidean space is generally considered sufficient. Different representations have different advantages and disadvantages. The generator representation for zonotopes – the only representation for zonotopes that is discussed in this thesis – allows for easy membership tests, linear transformations and Minkowski sum operations.

2.1 Half-spaces and hyperplanes

We start with two very simple geometric representations: Half-spaces and hyperplanes. Weisstein [Wei13] defines an half-space informally as that portion of an n-dimensional space, that is obtained by removing that part lying on one side of an $(n-1)$-dimensional hyperplane. We use the following definitions:

Definition 2. *(Half-space, hyperplane). A* half-space *in $\mathbb{R}^d$ is a set defined by a single affine inequality with $\vec{a} \in \mathbb{R}^d$ and a constant scalar $\gamma \in \mathbb{R}$, with the set being of the form:*

$$H = \{x \in \mathbb{R}^d \mid \vec{a}^\top x \leq \gamma\}.$$

A hyperplane *is defined analogously by using the operator $=$ instead of $\leq$. A hyperplane in a three-dimensional Euclidean space is called a* plane. *A hyperplane in a two-dimensional Euclidean space is called a* line.

A half-space separates the whole underlying vector space – in our case $\mathbb{R}^d$ – into two halves. Hyperplanes are an often used representation for certain types of infinite sets of points. In reachability analysis, simple guards can be represented by hyperplanes.

In addition, the following two classes of geometrical objects are discussed in this thesis:

- *Polytopes* are bounded sets of points constrained by a finite set of half spaces. They are the higher-dimensional generalization of polygons and polyhedra.
- *Zonotopes* are a subclass of polytopes that can intuitively be defined as the Minkowski sum of finitely many line segments.

In the next sections, we discuss polytopes and zonotopes, their representations, and operations on them.

2.2 Polytopes

First, we discuss the different representations for polytopes:
Definition 3. *(Polyhedron). A (convex)* polyhedron *$\mathcal{P}$ is the intersection of a finite set $\mathcal{H} = \{h_1, ..., h_n\}$ of half-spaces, $h_i = \{x \in \mathbb{R}^d \mid \vec{a}_i^\top x \leq \gamma_i\}$ for $i = 1, \cdots, n$:*

$$\mathcal{P} = \bigcap_{h \in \mathcal{H}} h. \tag{2.1}$$

Polytopes *are bounded polyhedra.*

Each d-dimensional polytope can also be given as the convex hull of its vertices $\mathcal{V} \subset \mathbb{R}^d$. Therefore, a polytope can also be represented by a set of points, whose convex hull is the polytope. For polytopes, different operations have different complexities depending on the representation.

Operation	H-polytopes	V-polytopes
convex hull of union	hard	easy
Minkowski sum	hard	easy
linear transformation	easy	easy
intersection	easy	hard
membership test	easy	easy

Table 2.1: We use *hard* to denote that there exists no polynomial-time algorithm, and *easy* to denote that there exists a polynomial-time algorithm in the size of the representation.

A polytope $\mathcal{P} = \bigcap_{i=1}^{n} \{x \in \mathbb{R}^d \mid \vec{a}_i^\top x \leq \gamma_i\}$ can be represented by the pair (A,b), where:

$$A = \begin{pmatrix} \vec{a}_1^T \\ \vdots \\ \vec{a}_n^\top \end{pmatrix}, b = \begin{pmatrix} \gamma_1 \\ \vdots \\ \gamma_n \end{pmatrix}$$

with A being the matrix consisting of the normal vectors of the intersected half-spaces, and b being the vector consisting of the scalar constants of the intersected half-spaces.

We call such an *H-representation* of a polytope an *H-polytope*. Polytopes in the following *H-representation* (vertex representation) are called V-polytopes. A V-polytope consists of the convex hull of a the finite set of vertices $\mathcal{V}$ of a polytope.

$$\mathcal{P} = \{\sum_{v \in \mathcal{V}} \alpha_v v \mid \alpha_v \in \mathbb{R}, \alpha_v \geq 0 \ \forall v \in \mathcal{V}, \textit{ such that } \sum_{v \in \mathcal{V}} \alpha_v = 1\}.$$

For H-polytopes, there exists no polynomial-time algorithm for the computation of the convex hull of the union or the computation of the Minkowski sum.

Similarly, there is no polynomial-time algorithm available to compute the intersection of V-polytopes. The vertex representation for polytopes allows for efficient operations in most aspects, but has no polynomial time algorithm for the intersection of two polytopes. Table 2.1 illustrates the computational complexity aspects of polytopes.

This restriction is of particular interest for this work, as there are proposed polynomial-time algorithms for zonotope intersection with polytopes. Le Guernic's [Gue09] algorithm for the computation of a zonotope-hyperplane intersection computes an over-approximating H-polytope. Thus, we discuss zonotopes and operations on zonotopes in depth in the next section.

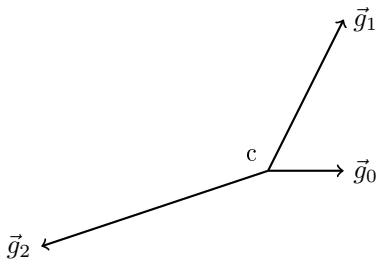

Figure 2.1: Zonotope construction in $\mathbb{R}^2$: The three generators of the zonotope are shown, as well as the center point. The first generator is $\vec{g}_0$.

2.3 Zonotopes

Zonotopes are a sub-class of convex polytopes and a generalization of zonohedrons. Zonohedrons were originally defined and studied by the Russian crystallographer E. S. Fedorov.

A zonotope is defined by a center point c and a set of n generators $G = (\vec{g}_0, \cdots, \vec{g}_{n-1})$. We define the interval set $[-1,1] := \{x \in \mathbb{R} \mid -1 \leq x \leq 1\}$. Then, the zonotope's set of points can be intuitively seen as the Minkowski sum [KS40] of finitely many line segments, represented by the generators:

Definition 4. *(Zonotope). A* zonotope $\mathcal{Z} \subseteq \mathbb{R}^d$ *in the d-dimensional Euclidean space* $\mathbb{R}^d$ *is a set of points, such that there are* $c \in \mathbb{R}^d$ *and* $\vec{g}_0, \ldots, \vec{g}_{n-1} \in \mathbb{R}^d$ *being a finite set of* n generators *with:*

$$\mathcal{Z} = \{x \in \mathbb{R}^d \mid \exists\ a_0, \ldots, a_{n-1} \in [-1,1].(x = c + \sum_{i=0}^{n-1} a_i \vec{g}_i)\}.$$

We call (using the same symbol as for the set) $\mathcal{Z} =< c\ ;\ \vec{g}_0, \cdots, \vec{g}_{n-1} >$ *the* representation *of the zonotope. We call* c *the* center point *and* $G := \{\vec{g}_0, \ldots, \vec{g}_{n-1}\}$ *the* generator set *of* $\mathcal{Z}$*.*

In this work, we generally refer to Definition 4 for zonotopes. If we refer to a zonotope, we generally also refer to the zonotope's representation. Figure 2.1 to 2.5 show the construction of a simple two-dimensional zonotope with three generators in $\mathbb{R}^2$.

Zonotopes have well-known polynomial-time algorithms for the Minkowski sum computation, linear transformation and membership testing. For reachability testing, the following operations on zonotopes, polytopes and hyperplanes are of particular interest:

- The membership test for a zonotope.
- The Minkowski sum of two (n) zonotopes.
- Linear transformation for zonotopes.
- Intersection and (over-approximation of the) union of two zonotopes.

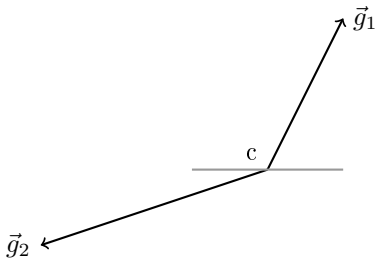

Figure 2.2: The first zonotope, with the representation $< c\ ;\ \vec{g}_0 >$ is shown as a gray line. The generator $\vec{g}_1$ is the second generator used for construction.

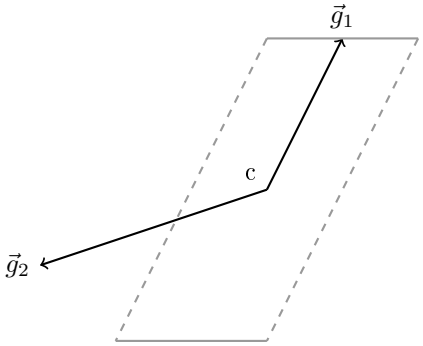

Figure 2.3: The second generator is added to the construction, forming a polygon. The edges constructed using the generator are marked with dashed lines. The grey lines mark the current constructed zonotope.

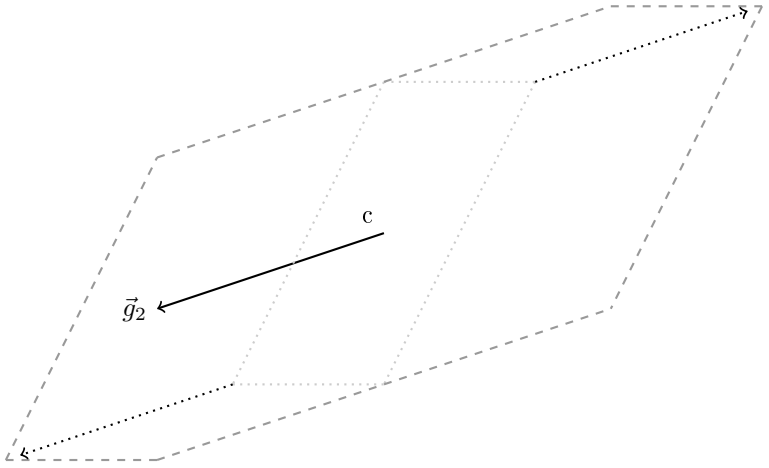

Figure 2.4: The last generator is added to the zonotope, extending the polygon in the directions of the last generator and its inverted counterpart. The dashed lines mark the current constructed zonotope, dotted light grey lines the zonotope from Figure 2.3. The dotted arrow visualizes the direction of the last generator $\vec{g}_3$.

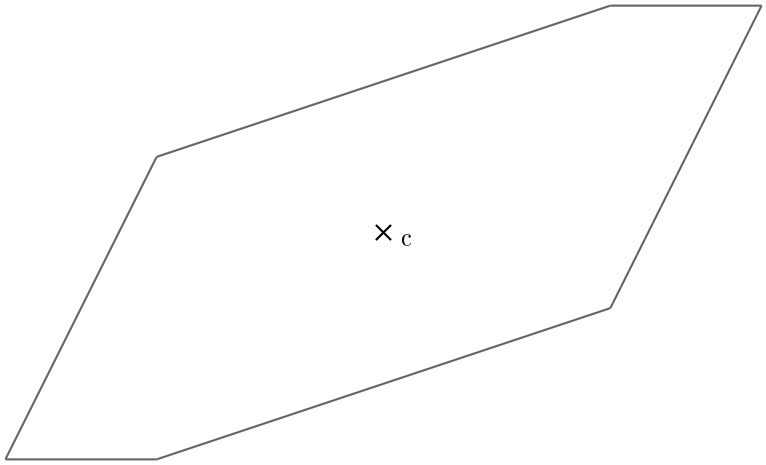

Figure 2.5: The constructed zonotope, with all prior constructed parts and generators omitted.

- Test for emptiness.
- Intersection of a zonotope with a hyperplane as an approximation for the zonotope-zonotope intersection.

In our work, we implemented several operations on zonotopes, including all of the operations mentioned above.

Zonotope membership The membership test for zonotopes can be computed using an algorithm with time complexity $\Omega(n^2)$ for n being the number of generators. The number of vertices of a zonotope can be quadratic in relation to the number of generators. However, there are algorithms for intersection of a zonotope with a point with only $\mathcal{O}(n \log n)$ running time [GNZ03].

According to Definition 4, a point $v \in \mathbb{R}^d$ has *membership* in a zonotope with the representation $< c\ ;\ \vec{g}_0, \cdots, \vec{g}_{n-1} >$ contained in $\mathbb{R}^d$, if there exist $a_0, \cdots, a_{n-1} \in [-1,1]$ such that:

$$c + \sum_{n=0}^{n-1} \vec{g}_i a_i = v.$$

The *membership problem* for zonotopes can be efficiently solved using the simplex algorithm [DT97], utilizing that the zonotopes generators represent a system of linear inequalities that can be solved.

Minkowski sum of zonotopes The Minkowski sum of two zonotopes is informally defined as the addition of the two center points and the generator sets. The result can be used to, e.g., model the effect of disturbances or uncertainties by bloating a set with another set, applying the Minkowski sum operation.

The *Minkowski sum* of *two sets* $A, B \subseteq \mathbb{R}^d$ is defined as:

$$A \oplus B = \{a + b \mid a \in A, b \in B\}$$

Similarly, the *Minkowski sum* $Z_0 \oplus Z_1$ of *two zonotopes* Z_0, Z_1 with their respective center points c_0, c_1, generators $\vec{g}_{01}, \cdots, \vec{g}_{0i}$ and $\vec{g}_{11}, \cdots, \vec{g}_{1j}$ is represented as follows:

$$Z_0 \oplus Z_1 =< c_0 + c_1 \ ; \ \vec{g}_{01}, \cdots, \vec{g}_{0i}, \vec{g}_{11}, \cdots, \vec{g}_{1j} >$$

with the new center points $c_0 + c_1$ components being the sum of the of components of c_0 and c_1.

The Minkowski sum computation for zonotopes is simple and can be done in linear time complexity with respect to the number of generators. Linear transformations are also often used and are also useful for generating testing examples, as discussed in Chapter 4.

Linear transformations Of particular interest for this work are linear transformations in the Euclidean space, i.e., linear transformations from $\mathbb{R}^m$ to $\mathbb{R}^n$. A common representation for linear transformations are transformation matrices. Transformation matrices for linear transformations between $\mathbb{R}^m$ and $\mathbb{R}^n$ are always $m \times n$-matrices.

Definition 5. *(Linear transformation). A* linear transformation *with respect to a matrix* $A \in \mathbb{R}^{m \times n}$ *is a function* $f : \mathbb{R}^n \to \mathbb{R}^m$, $f(x) = Ax$. *We call* A *the linear transformation matrix.*

Linear transformations, and especially automorphic linear transformations are particularly simple to apply to zonotopes. Let Z be a zonotope with the representation $Z =< c \ ; \ \vec{g}_0, \cdots, \vec{g}_{n-1} >$. The linear *transformation* f *of the zonotope* Z with the corresponding transformation matrix A is a linear transformation on the zonotopes center point c and generators $\vec{g}_0, \cdots, \vec{g}_{n-1}$:

$$f(Z) =< Ac \ ; \ A\vec{g}_0, \cdots, A\vec{g}_{n-1} > .$$

2.4 The reachability algorithm using zonotopes

For systems with combined discrete-continuous behavior modeled as hybrid automata, the reachability problem can be solved utilizing geometric representations.

Using geometric objects, it is possible to conduct forward reachability computation, i.e., the extension of the initial state set until no new states are found or an unsafe state is found reachable. A similar approach is the backward reachability computation: By starting the search at a (usually unsafe) set of states, we search backwards to see if the reachable set intersects with the initial set of states.

While these techniques are very powerful, and allow for incomplete analysis, the resulting computations on the exact sets are usually very expensive, and

have strong limitations on the operations on the sets. Thus, we use over-approximating representation of the reachable sets.

The use of efficient set representations allows for a fast computation of reachable sets, thus allowing to prove a systems safety: If, with forward search, the unsafe set is not in the over-approximation of the reachable set, the system is provably safe. Algorithm 2.1 illustrates the basic reachability algorithm in pseudo-code.

Algorithm 2.1: Basic reachability algorithm

```
1  Input: A set of initial states I, a target set T.
2  finite set L of directions.
3  Output: true if T is reachable, false if T is not reachable from I.
4  R ← I                                ◁ Current reachable set
5  DO
6    IF R intersects T THEN
7      RETURN true                      ◁ Target set is reachable.
8    ELSE
9      R ← R ∪ Post(R)                  ◁ Extend the reachable set.
10   END IF
11 UNTIL Post(R) ⊆ R
12 RETURN false                         ◁ Target set not reachable.
```

Thus, using *efficient state set representations*, such as zonotopes, to conduct forward and backward reachability analysis, we can speed up the analysis of continuous systems. In this thesis, we use zonotopes to over-approximate the reachable sets.

While zonotopes are very efficient concerning most operations, they are not closed under the *intersection* operation, as the intersection of a hyperplane and a zonotope may not be a zonotope itself, but a polytope. However, checking the emptiness of the intersection of a zonotope and a hyperplane can be done in polynomial time regarding the number of generators of the zonotope [Gue09].

The method we are focusing on for zonotope-hyperplane intersection is the one proposed by Le Guernic [Gue09], which computes an over-approximation. The algorithm accepts a set of points represented by a zonotope, a hyperplane and a finite set $\mathcal{L}$ of directions, and computes and over-approximated polytope of the hyperplane-zonotope intersection with $|\mathcal{L}|$ affine inequalities. The algorithm approximates the intersection by projecting the zonotope and the hyperplane into $|\mathcal{L}|$ different two-dimensional vector spaces spanned by the hyperplane normal vector and a vector $\vec{l}$ from the set of directions. In this two-dimensional vector space the intersection of the two-dimensional representation of the zonotope with the two-dimensional-representation of the hyperplane is computed.

This can be done in time complexity $\mathcal{O}(d \log n)$ [Gue09]. The resulting intersections can then be used to compute an over-approximation of the intersection in the original d-dimensional vector space represented by a set of constraints to the d-dimensional polytope that is an over-approximated intersection. In the next chapter we discuss Guernic's proposal for this intersection of hyperplanes with d-dimensional zonotopes.

Chapter 3

Zonotope-hyperplane intersection

Hybrid automata typically have guards whose satisfaction sets are half-spaces or hyperplanes. Therefore, if state sets are represented by zonotopes, the computation of the intersection of a zonotope and a hyperplane plays an important role in the reachability analysis.

In this chapter, we discuss an algorithm for the over-approximation of a zonotope-hyperplane intersection proposed by Le Guernic [Gue09]. In the following section, we outline the algorithm for the zonotope-hyperplane intersection. Following this brief outline, we separately discuss each part of the intersection algorithm in the Sections 3.2 to 3.4.

3.1 Outline

Guernic's algorithm computes an over-approximation of the intersection of a zonotope with a hyperplane, i.e., an over-approximation of $\mathcal{G} \cap \mathcal{Z}$ for a hyperplane $\mathcal{G}$ and a zonotope $\mathcal{Z}$. The computation of an over-approximation is necessary, as the intersection of a hyperplane and a zonotope is in general not a zonotope.

We use Le Guernic's proposal [Gue09] for the efficient computation of an over-approximation of the intersection.
The algorithm computes this over-approximation by intersecting projections of the zonotope and the hyperplane in a variety of two-dimensional vector spaces. From the intersections in the two-dimensional vector spaces, we derive half-spaces in the original (Euclidean) space $\mathbb{R}^d$.

These half-spaces are boundaries of a polytope over-approximating the intersection. By combining these half-spaces into a polytope as described in Section 3.4, we obtain an over-approximation of the intersection in the original vector space.

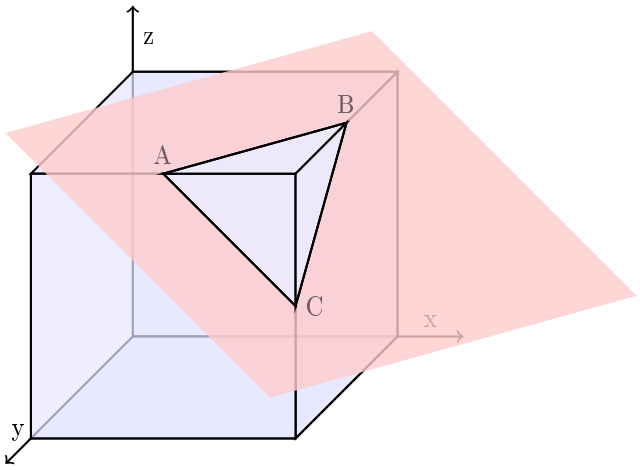

Figure 3.1: An exemplary zonotope in $\mathbb{R}^3$ with an intersecting hyperplane marked in red. The hyperplane-zonotope intersection is bounded by the vertices A,B and C.

The vector spaces have spanning sets consisting of the hyperplane's normal vector and an additional second vector. We discuss the choice of the second vector and the projection in Section 3.2. The over-approximation in the two-dimensional vector space is computed using a divide-and-conquer algorithm, which is discussed in Section 3.3.

In this chapter, we use the simple example in Figure 3.1 to illustrates the intersection algorithm. It shows a zonotope in the Euclidean space with its representation having the center point $(1,\ 1,\ 1)^\top \in \mathbb{R}^3$ and the generator set:

$$g_0, g_1, g_2 = (1,0,0)^\top, (0,1,0)^\top, (0,0,1)^\top \in \mathbb{R}^3.$$

It is intersected with a hyperplane with a normal vector $h = (1,\ 1,\ 1)^\top$ and a scalar constant $\gamma = \frac{5}{3}$. The resulting zonotope-hyperplane intersection has the following vertices:

$$A = \begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix}, B = \begin{pmatrix} 2 \\ 1 \\ 2 \end{pmatrix}, C = \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}$$

The vertices A, B, C are on the edge of the zonotope and can be defined by using all but one generator with a scalar multiplication with 1, as there are no linear dependent generators. They fulfill the hyperplane's parametric form equation:

$$x - \frac{5}{3} + y - \frac{5}{3} + z - \frac{5}{3} = 0$$

This example intersection is a triangle, and thus has no corresponding zonotope representation, but can be represented precisely by a polytope with four affine equations, one for the hyperplane, and three for the half-space boundaries.

Algorithm 3.1: Guernic's original proposal

```
Input: A set S , a hyperplane G = {x ∈ ℝ^d | h⃗^⊤ x = γ} and a
finite set L of directions.
Output: an over−approximation of the polytope representing S ∩ G.
IF S ∩ G ≠ ∅ THEN
  FOR l IN L DO
    S_2d ← DIM_REDUCE(S, h, l);              ◁ Projection to ℝ^2.
    L_γ ← {(x,y) ∈ ℝ^2 | x = γ}
    ρ_l ← BOUND_INTERSECT_2D(S_2d, L_γ)      ◁ Intersection in ℝ^2.
  END FOR
END IF
RETURN {x ∈ ℝ^d | ∀l ∈ L, l^⊤ · x ≤ ρ_l}    ◁ Combining the polytope.
```

Algorithm 3.1 shows the original proposal for the intersection algorithm, which we demonstrate on the example.

We discuss the projection and the choice of the base vectors in the following section, and the intersection computation in Section 3.3.

3.2 Projecting the zonotope and the hyperplane

Since the intersection is computed in two-dimensional vector spaces, the zonotope and the intersecting hyperplane needs to be projected into a two-dimensional vector space. The projection of the set DIM_REDUCE$(\mathcal{S}, h, l)$ in Algorithm 3.1 is a linear projection. A projection into the two-dimensional space has a linear time complexity in the dimension size.

The *two-dimensional projection* $\rho_{h,l}$ is defined as $\rho_{h,l} : \mathbb{R}^d \to \mathbb{R}^2$ with the normal vector h of the hyperplane $\mathcal{G} = \{x \in \mathbb{R}^d \mid h^\top x = \gamma\}$ and a second vector $l \in \mathbb{R}^d$ is a linear projection, such that:

$$\rho_{h,l}(x) = (x \cdot h,\, x \cdot l)\ \forall x \in \mathbb{R}^d.$$

The dimensional projection in Algorithm 3.1 DIM_REDUCE$(\mathcal{S}, h, l)$ is the two-dimensional projection $\rho_{h,l}$ of the set $\mathcal{S}$ from $\mathbb{R}^d$ to $\mathbb{R}^2$.

For each projection, we need to choose a pair of vectors – the base vectors for the target vector space of the projection. The first base vector of the projected space is the hyperplane's normal vector, and thus does not need to be computed. We call the set of second base vectors chosen for the algorithm *directions*.

A direction can be chosen utilizing several different strategies, which influence the quality of the over-approximation. These strategies can, e.g., utilize *base vectors*, i.e., selecting the base vectors of the original vector space as a base vector for the two-dimensional vector space; choosing these vectors as base vectors is not always possible, as we will discuss later in this section.

Other strategies utilize *random* direction generation or *generator-related* choice, i.e., selecting the directions perpendicular to the zonotope generators, which

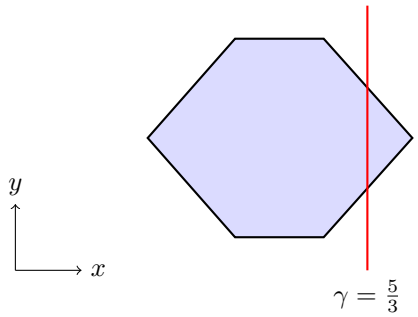

Figure 3.2: A projection of the zonotope and the hyperplane from Figure 3.1 into a two-dimensional vector space. The hyperplane's scalar constant is represented by the red vertical line.

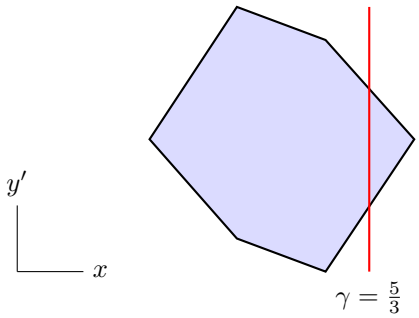

Figure 3.3: A projection similar to Figure 3.2, with a different second vector y'.

may result in the intersection algorithm to compute half-spaces that contain the zonotope's edges.

The choice of directions has a large impact on the quality of the resulting intersection. As a rule, using more directions (and thus, more half-spaces) does not necessarily have to, but in practice often does improve the precision of the result.

Figure 3.2 shows the zonotope from the example in Figure 3.1 projected into a two-dimensional vector space spanned by the hyperplane normal vector $(1, 1, 1)^\top$ and a vector $l_1 = (1, 1, -1)^\top$. Figure 3.3 show the zonotope and the hyperplane projected with a different second base vector.

Therefore the directions – the normal vectors of the derived half-spaces – have a large impact on the tightness of the computed over-approximation of the intersection.

Every direction must be linear independent to the hyperplane normal vector. In many scenarios, a vector perpendicular to the zonotopes surface, i.e., perpendicular to a generator of the zonotope, can results in a tight over-approximation of the zonotope:

Definition 6. *(Perpendicular orientation). A vector* $\vec{v} \in \mathbb{R}^d$ *is* perpendicular, *write* $\vec{v} \perp \vec{w}$, *to a vector* $\vec{w} \in \mathbb{R}^d$, *iff* $\vec{v} \cdot \vec{w} = 0$.

With Guernic's intersection algorithm we can compute the intersection of the projected zonotope and the projected hyperplane in polynomial time [Gue09], which we discuss in the next section. As discussed in Section 3.4, we can improve the precision of the approximation by increasing the number of directions.

Furthermore, the number of direction needs to be high enough, that the resulting half-spaces form a convex polyhedron in the d-dimensional original vector space. In the next section, we discuss the computation of the intersection in the vector spaces generated in the part of the algorithm discussed here.

3.3 Divide-and-conquer intersection algorithm

The intersection computation of two-dimensional zonotopes and hyperplanes, i.e., a line in a two-dimensional vector space, is the central part of the algorithm.
This two-dimensional intersection problem has been solved for two-dimensional polytopes (polygons) and hyperplanes, and can be solved for zonotopes and hyperplanes in a particularly elegant way. Algorithm 3.2 solves the problem of intersecting a line with a two-dimensional zonotope in polynomial-time.

In this discussion of the algorithm, we define the *y-direction* or the direction along the *y-axis* as the second dimension of the vector space, and the *x-direction*, or the direction along the *x-axis* as the first dimension of the vector space.

We will refer to elements in the zonotope as *points*. Analogous, we define a point lying in the positive x-direction from a given point as *right of* the given point and a point lying in the negative x-direction from the given point as being *left of* the given point. For the y-axis, we say a vector points *upward* if it has a positive second component, and *downward* if it has a negative second component.

For points, we may refer to their valuation in the first dimension as their *x-value* or *first value*, and to their valuation in the second dimension as their *y-value* or *second value*.

3.3.1 Sorting the generators

To start the divide-and-conquer-algorithm on the generator set, the (now two-dimensional) zonotope generators need to be sorted. This is done in two steps: First, all generators that are pointing *upwards* are mirrored by inverting their values. This is necessary for the next step and does not change the zonotopes shape. Algorithm 3.2 shows this in the lines 5 to 10. Figure 3.4 and 3.5 illustrate the mirroring of the generator set.

Following the mirroring, the generator set is sorted in a clockwise order. While an arbitrary trigonometrical sorting is sufficient, it is necessary to decide for

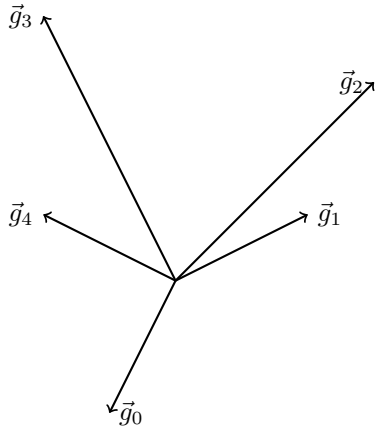

Figure 3.4: An unsorted example generator set of a zonotope into a two-dimensional vector space.

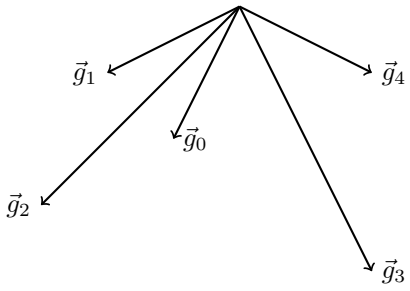

Figure 3.5: The generator set from Figure 3.4 with the generators mirrored downwards.

Algorithm 3.2: The original algorithm

```
1  Input: A two-dimensional zonotope Z =< c ; g_1, ..., g_r > and a line
2  L_γ = {(x, y) : x = γ} such that Z ∩ L_γ ≠ ∅.
3  Output: ρ_{Z∩L_γ}((0,1)^⊤).
4     P ← c                                          ◁ current position P = (x_P, y_P)
5  FOR i FROM 1 TO r DO
6     IF y_{g_i} > 0 or (y_{g_i} = 0 and x_{g_i} > 0) THEN        ◁ g_i = (x_{g_i}, y_{g_i})
7        g_i ← −g_i                  ◁ Ensure all generators are pointing downward
8     END IF
9     P ← P − g_i                    ◁ Drives P toward the highest vertex of Z
10 END FOR
11 g_1, ..., g_r ← SORT(g_1, ..., g_r)        ◁ Sort the generators counter-clockwise
12 IF x_P < γ THEN
13    G ← {g_1, ..., g_r} ∩ {R^+ × R}                    ◁ Look right
14 ELSE
15    G ← {g_1, ..., g_r} ∩ {R^− × R}                    ◁ Look left
16 END IF
17 s ← Σ_{g∈G} 2g                                        ◁ Initial vantage point
18 WHILE |G| > 1 DO
19    (G_1, G_2) = SPLIT_PIVOTING(G,s)                   ◁ Split set along s
20    s_1 ← Σ_{g∈G_1} 2g
21    IF [P; P + s_1] ∩ L_γ ≠ ∅ THEN                     ◁ Check intersection
22       G ← G_1
23       s ← s_1                                         ◁ Use g_1 to compute s
24    ELSE
25       G ← G_2
26       s ← s − s_1                                     ◁ New vantage point
27       P ← P + s_1
28    END IF
29 END WHILE
30 (x, y) ← [P; P + s_1] ∩ L_γ                           ◁ Compute intersection
31 RETURN y
```

ascending or descending order. The polar angle of the generators can be utilized as a sorting key; in our example (and in our implementation) we sort the generators in ascending order.

The sorting is complexity-wise the most complex part of the algorithm, which can be done in time (with respect to n generators) $\mathcal{O}(d \log n)$. The rest of the algorithm has a linear time complexity (to the number and size of the generators). Thus, even for zonotopes with a large number of generators, this algorithm is very efficient. Guernic's original algorithm in Algorithm 3.2 refers to the sorting in line 11. In the next section we discuss the computation of the intersection with the hyperplane in a two-dimensional Euclidean space.

3.3.2 Vantage point iteration

This part of the algorithm is the divide-and-conquer type strategy for the generators set splitting. Following the sorting, we compute a highest vertex in the zonotope, i.e., a point in the zonotope for which no point with a higher y-value exists. This is done by starting with the center point of the zonotope and then subtracting all the now downward-oriented generators from the center point. Line 9 in Algorithm 3.2 illustrates this. This computation can easily be inte-

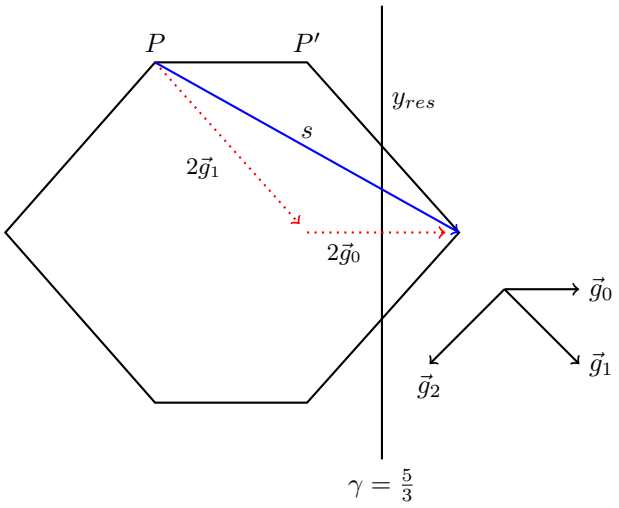

Figure 3.6: P is the starting point the vantage point iteration. P' is the vantage point after one iteration step. The generator set is already sorted. The starting set (visualized as a dotted arrow) G is $\{\vec{g}_0, \vec{g}_1\}$, after the first step G is $\{\vec{g}_1\}$. y_{res} is the computed intersection point.

grated with the generator mirroring and sorting, and thus does not influence the time complexity of the algorithm.

We call this first computed topmost point of the zonotope the *vantage point*, the starting point of the iteration. In the next step, we check if the hyperplane – which is represented in the two-dimensional space as a vertical line – is left or right of the vantage point. Then we omit all generators not pointing in that direction from our working set (line 12 to 16 in the code listing). The remaining generators form the *working set*.

Following the computation of the first vantage point, the (sorted) working set is then split until only the generator that forms the edge intersecting with the hyperplane remains. This is done by splitting the set and moving the vantage point forward (i.e., along the generators), until the intersection can be computed using the vantage point and the single remaining generator in the working set. Figure 3.6 illustrates this on the example from Figure 3.2.

Depending on the selection of the pivot element, this divide-and-conquer algorithm to find the edge intersecting the line can be in linear time, if each generator in the set is only visited once.

A good strategy for the pivot element selection of this divide-and-conquer-algorithm is to split the working set into two subsets G_1, G_2, such that all vectors in G_1 are above the intersecting vector s, and all vectors in G_2 are below or co-linear to s. After the set of generators is sorted, the vantage point iteration on the two-dimensional zonotope is conducted. Figure 3.7 to 3.9 illustrate this iteration on a more complex example.

The final intersection computation to return the y value of the intersected line is trivial. With $P := (P_x, P_y)$ (as referred to in line 30 of Algorithm 3.2) being

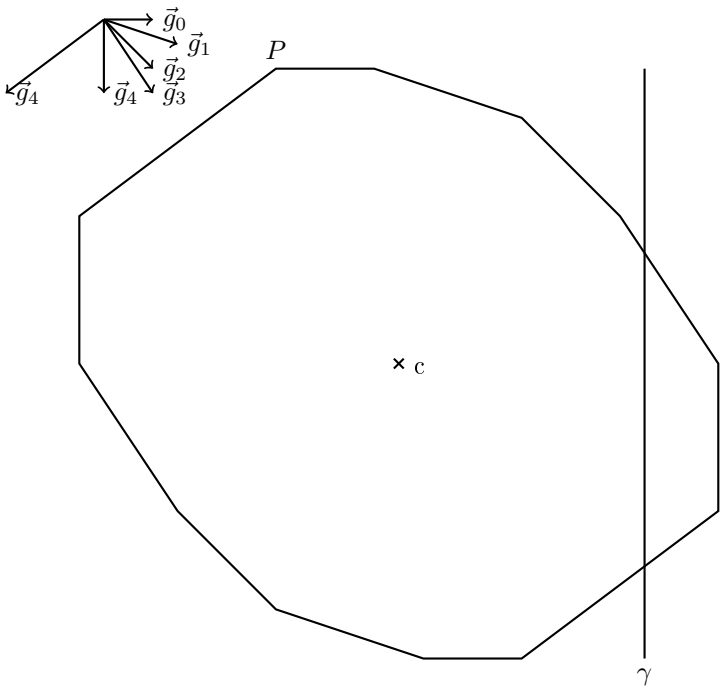

Figure 3.7: A more complex example for the vantage point iteration. The generator set G contains all generators of the zonotope.

the current vantage point, $s := (s_x, s_y)$ being the set intersecting the vertical line with l being its x coordinate, the y value of the intersection is computed as follows:

$$y_{res} = P_y + s_x \frac{l - P_x}{s_x}.$$

The result of this computation multiplied with the original computed projection vector and the projected vector constitutes to an half space, with the choosen direction vector being the normal vector of the half-space. After projecting the point back into the original vector space, every point in direction of the computed projection is not in the intersection. In the following section, we discuss this technique in detail.

3.4 Combining the half-spaces

Algorithm 3.2 returns the scalar y-value p of the intersection in the (two-dimensional) vector space spanned by the hyperplane normal vector and the direction $\vec{l}$. For every direction $\vec{l}$, we construct half-space $h_{\vec{l}}$ as follows:

$$h_{\vec{l}} := \{x \in \mathbb{R}^d \mid \vec{l}^{\top} x \leq p\vec{l}\}$$

The intersection polytope can be constructed by combining all computed half spaces into an $\mathcal{H}$-polytope in the original d-dimensional vector space. By com-

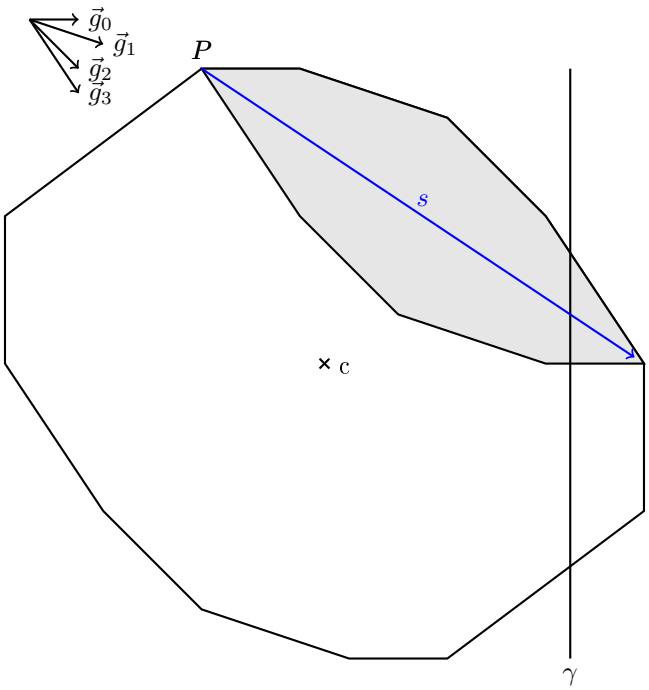

Figure 3.8: The initial generator set is reduced to the generators pointing towards the intersection.

puting the intersection of the half spaces generated by the set of directions, one can compute an over-approximation I of the actual intersection:

$$I = \bigcap_{\vec{l} \in \mathcal{L}} h_{\vec{l}}$$

With more directions (and thus, more computed intersections) the precision of the over-approximation can be increased. Figure 3.11 and 3.12 illustrates the half-space intersections forming a polytope. The choice of the directions in the example resulted in a very precise over-approximation of the intersection; in fact, it is a precise computation of the intersection. While always possible, an exact computation of the intersection is highly unlikely for any but the most simple zonotopes.

In this chapter we described and discussed Guernic's algorithm for the computation of over-approximated hyperplane-zonotope intersections. In the following chapter, we discuss the structure and the features of our library, and evaluate it with some small experiments.

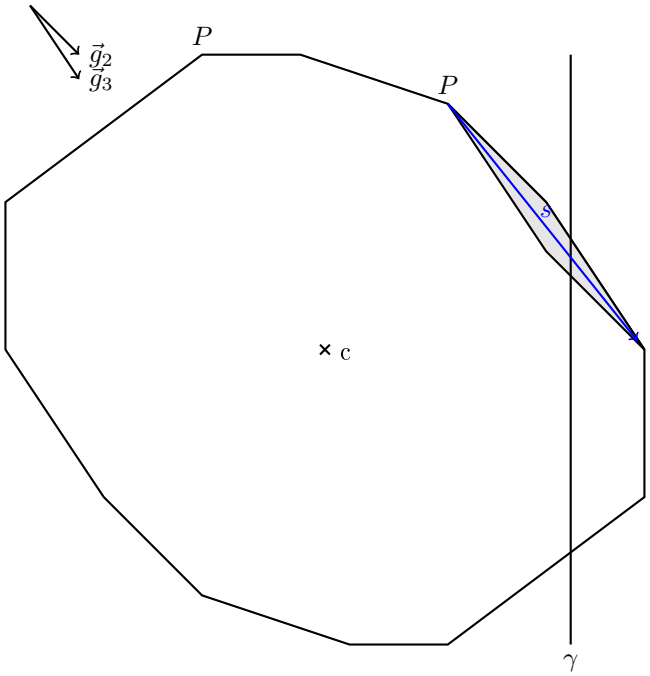

Figure 3.9: Further splitting of the generator set results in the working set containing only two generators, the second one resulting in the intersecting edge.

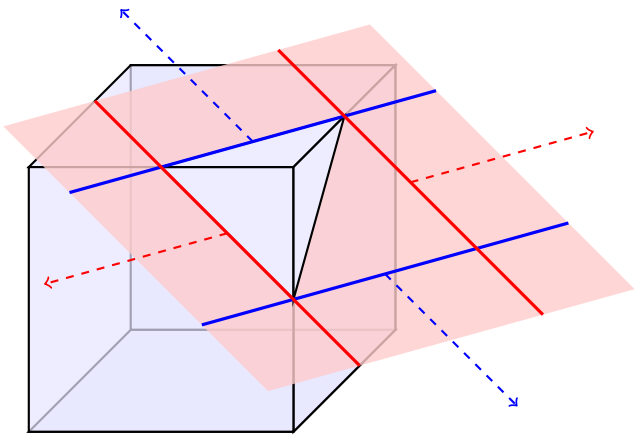

Figure 3.10: The exemplary zonotope from Figure 3.1, which added indicators for four half-spaces. The intersection with the form of an triangle is over-approximated by a rectangle.

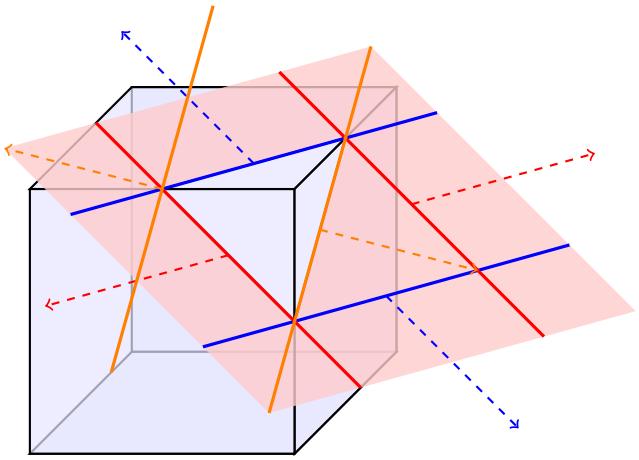

Figure 3.11: The over-approximation from Figure 3.10 with an additional half-space.

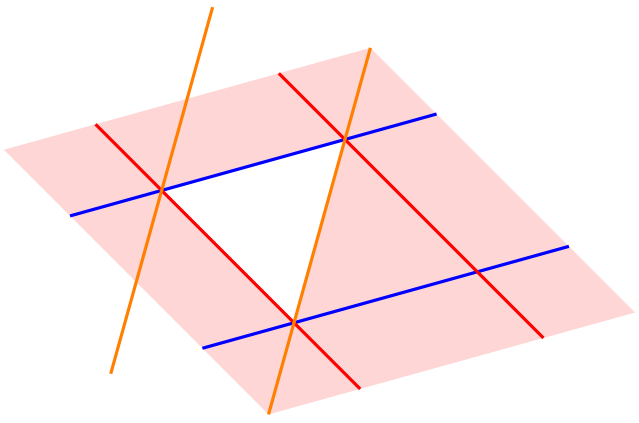

Figure 3.12: The computed over-approximation from Figure 3.11 of the intersection, in this case exactly the intersection.

Chapter 4

Evaluation

In this chapter, we evaluate our work on the topic. First we discuss our zonotope library, second we describe the experimental evaluation of our zonotope library, explain the methodology of our testing and discuss the results in Section 4.1. Following that, we discuss the structure of our implementation in Section 4.2.

Preliminarily to this thesis, we identified the need for a free library providing a geometric representations of reachable sets. The zonotope representation of state sets, while having other interesting properties, allows for efficient computation of the over-approximation of the intersection with a hyperplane. The hyperplane representation is often used to represent a hyperplanar guard, and is thus very useful for forward reachability analysis.

As mentioned in the earlier chapters, existing libraries have unacceptable limitations and can thus not provide these functionalities. To meet the demand for a such an implementation, we implemented a library for zonotope functionalities, with a focus on the hyperplane-zonotope intersection described in Chapter 3.

This free library can be used in other projects, such as HyPro [M+13], as long as they are GPL compliant. The library is easy to adapt, comes fully documented and is thoroughly tested. In the next section, we discuss the experiments we conducted to evaluate the libraries functionalities.

4.1 Experiments

To test our implementation, we conducted a number of experiments. The purpose of the experiments is to ensure that the implementation of the intersection algorithm described in Chapter 4 conducts a correct over-approximation. In this section, we describe the methodology of the experiments and discuss two of the experiments.

Our experiments use linear transformations on randomly generated zonotopes and randomly generated hyperplanes. We start by generating a zonotope $Z \subset \mathbb{R}^d$ utilizing the zonotope representation described in Chapter 2. The zonotope Z is generated with varying number of randomly generated generators. In most

tests, the corresponding polytope has fifteen or more faces. Similarly, we generate a hyperplane intersecting the generated zonotope, which randomly generated normal and hyperplane vectors.

The values of the components of the generated vectors are generated using random number generation. We use random number generation with a discrete uniform distribution, i.e., we decide on a number of decimal places for the generated numbers, e.g., two, and only generate numbers with decimal places up to that number.

In the next step, we choose a linear transformation that transforms the zonotope along the surface of the hyperplane, i.e., in a way that the zonotope is transformed, but still intersects the hyperplane. This can be done, for example, by rotating the zonotope (and, more importantly the center point) along the hyperplane's surface. Another technique is to apply the linear transformation only to the generators of the zonotope, not to the center point. For convenience, the center point can be shifted along the hyperplane to allow for an intersection free plotting of the transformed zonotopes.

We use the latter transformation technique in the exemplary experiments shown in the next section. This linear transformation is then applied a number of times to the zonotope, each time computing the over-approximated intersection algorithm described in Chapter 3, and plot the transformed zonotopes along with the over-approximated intersections.

By plotting the computed results, we can easily check the quality of the computed over-approximated intersection. With the help of a viewer for three-dimensional objects, such as the one in MATLAB, one can easily verify that the computed intersection is indeed a correct over-approximation. We exemplarily discuss two simplified experiments, with – for convenience – smaller generator sets and and rounded values in the vectors components. The full series of experiments can be found in the User's Manual [Gla14].

Exemplary experiments To illustrate our testing process, we present two simplified examples of our testing (i.e., with rounded numbers and a small number of generators). Both zonotopes use the hyperplane represented by using normal vector $(1,1,1)^\top \in \mathbb{R}^3$ with a scalar value 1.

These examples use linear transformations which are combinations of rotations, shear transformations (a shear transformation is a linear mapping that displaces each point in a fixed direction, proportional to its distance from a line – the eigenvector of the transformation) and scaling transformations, which are applied several times to the generators of the experiments zonotope.

The zonotope in our first example has the following representation:

$$< \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \; ; \; \{ \begin{pmatrix} 1.1 \\ 0 \\ -3.61 \end{pmatrix}, \begin{pmatrix} 2.39 \\ -1.61 \\ 3.6 \end{pmatrix}, \begin{pmatrix} 3 \\ 0.25 \\ -3.9 \end{pmatrix}, \begin{pmatrix} -3.6 \\ 3.4 \\ 1.6 \end{pmatrix}, \begin{pmatrix} -3.4 \\ 2.1 \\ 2.9 \end{pmatrix} \} >$$

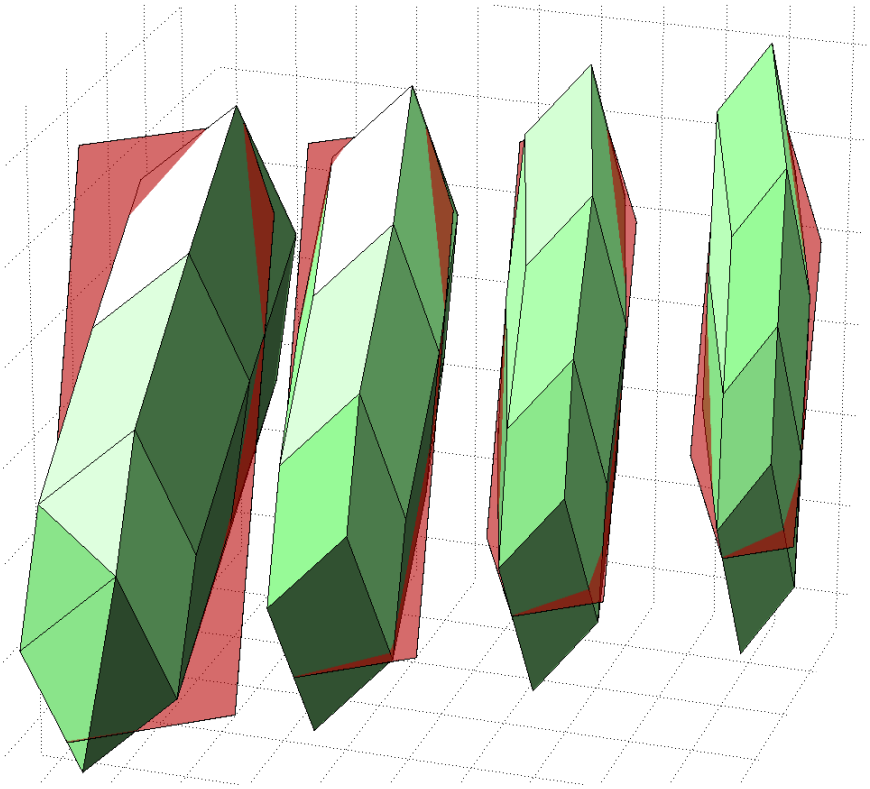

Figure 4.1: A first exemplary experiment using a rotation and a shear transformation on the zonotopes generators. The computed over-approximation is marked in red. After each transformation, the zonotope is shifted along the hyperplanes surface for convenience.

The transformation matrix used in the experiment, a combined shearing and rotation, is the following:

$$\begin{pmatrix} 1.36 & 0.2 & 0 \\ -0.2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Figure 4.1 shows plotted version of the over-approximated zonotope-hyperplane intersections computed.

The second, smaller experiment, uses a slightly simpler zonotope with four generators:

$$< \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \; ; \; \{ \begin{pmatrix} 1.1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0.31 \\ -1.60 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0.25 \\ -1.9 \end{pmatrix}, \begin{pmatrix} -1.42 \\ 1.2 \\ 0 \end{pmatrix} \} >$$

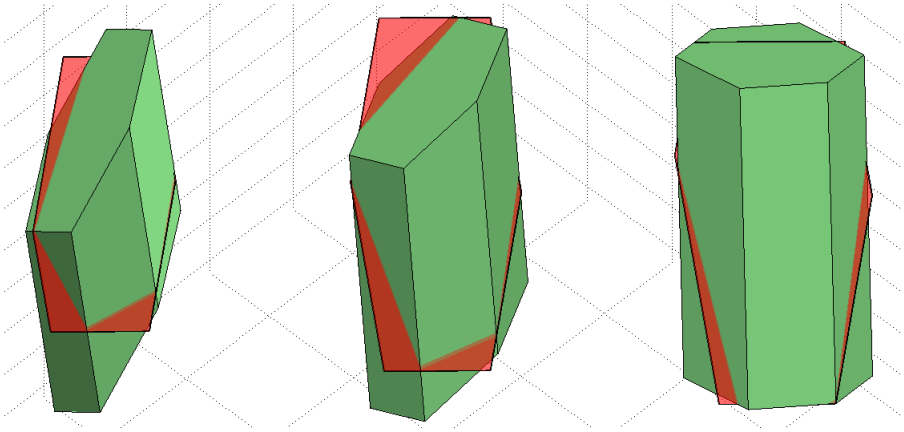

Figure 4.2: The second, smaller, exemplary experiment using a combined rotation and scaling transformation. Similarly to Figure 4.1, the computed overapproximation is marked in red.

The linear transformation matrix corresponding to this example is a combined rotation and scaling matrix:

$$\begin{pmatrix} 1.82 & 0 & 0.22 \\ 0 & 0.85 & 0 \\ -0.27 & 0 & 0.82 \end{pmatrix}$$

Figure 4.2 illustrates this examplary experiment.

These experiments allow for quick and reliable evaluation of the correctness of the intersection. The intersection algorithm is very efficient, as the sorting of the generator set the most time consuming part. The rest of the algorithm has a linear time complexity.

Testing Environment Our testing system runs on a 32-bit Intel Core 2 Duo 1.66 GHz CPU with 1 GB memory. The details of the testing system are described in detail in the User's Manual [Gla14]. None of our experiments took more than a second to run, and the first experiment described took an average 0.324 seconds to run, including writing the results to the hard disk.

In the User's manual, there are twenty documented experiments, with increasing numbers of generators. It is also very easy to create additional experiments. In the next section, we discuss the structure of our implementation.

4.2 Implementation

For the implementation of the zonotope library, it was necessary to implement a library providing matrix functionalities. While there are matrix libraries available, such as the GNU Scientific Library (GSL) [G+09], they have limitations,

such as GSL not allowing for arbitrary precision computations. The library is available for download on the HyPro webpage [M+13].

4.2.1 Matrix implementation

Current matrix libraries are either not free software or, in the case of the GSL library, do not support arbitrary precision data types. Thus, we needed to implement a library for matrix creation and operations that is not limited to finite precision data types or specific data types at all. While the matrix is currently limited to operations necessary for the given application, it can easily be extended.

Unlike the GSL library, our library provides an object-oriented matrix implementation. It allows some matrix operations, i.e., operations on the matrix itself and memory-stable operations, within the matrix class scope.

Our class provides built-in functionalities for several commonly used matrix functionalities, such as changing of matrix entries, linear transformations, multiplying with matrices, scalars and vectors as well as row and column swapping. Likewise, we implement structures for often used column and row vectors which provide similar operations.

The matrix library can be extended with additional functionalities, if necessary. The class-based approach allows for easy extending of objects, e.g., by inheritance of objects. With the template-based approach to multiple data types, it is easy to use own high-precision types like GNU MPFR [FHL+07] or special purpose data types. Thus, the matrix can be used to implement the zonotope representation.

4.2.2 Zonotope implementation

The zonotope implementations allows for creation, import, export and other operations on zonotopes in the generator representation. For many operations, it utilizes the matrix functionalities provided by the matrix library, which is necessary to use the zonotope functionalities.

As it uses the matrix library, the zonotope representation library is independent of certain data types, and can thus utilize arbitrary precision data types. We provide functionalities for several typical zonotope operations, such as:

- General zonotope functionalities such as creating, importing and exporting zonotopes.
- Additional functionalities, such as the membership test, Minkowski sum computation of two zonotopes.
- Linear transformation of zonotopes.
- Computation of over-approximations of the intersection of a zonotopes with a hyperplane by the way of intersecting two-dimensional projections.

- Additional utility functionalities such as plotting, generator sorting according to different sorting criteria, and edge computation. These functionalities are mostly used in tests and examples included in the library.

Some functionalities cannot – for reasons of consistency and memory management – be used inside the zonotope class scope. These functions mostly create new zonotopes or objects, e.g., the Minkowski sum of two zonotopes. To keep the libraries usable for arbitrary data types, these new objects must be initialized out of the zonotopes context.

The object-oriented library can easily be extended with new functionalities, and the code is fully documented. A number of examples provide sample uses and are easy to modify and to adapt. Further examples – which sometimes need additional software – provide visualizations of zonotopes in the Euclidean space, similar to the illustrations in Section 4.1.

4.2.3 Environment and external libraries

Our test system setup consists of a default installation of a Linux Mint 15 system running on a Linux 3.8 kernel. The library is written in C++11, the most recent version standard of the C++ programming language. While the library itself is independent of external libraries and uses only built-in functionalities of the GNU C++ library, some examples and tests use external libraries, however, these are optional. For these external libraries, some constraints apply. In this work, we used the following software:

- We compiled tests and examples using the C++ compiler from the *GNU Compiler Collection* [The13a], version *4.7.3*. Newer versions should work without limitations, and older versions may work without limitations.
- We link against the GNU C++ library [The13b], commonly known as *libstdc++*. We use the version *libstdc++-6-4.7* of the library, which is included in the GNU compiler collection. The GNU C++ library, as well as the GNU C++ compiler are published under the terms of the GNU General Public License (GPL) [Fre], the same license our library is published under.
- For examples and visualizations, we use MATLAB.
  We use version *R2013a* [MAT13], a non-free numerical computing environment by MathWorks Inc. We strictly do not use MATLAB for any computation in our library, although some of the examples make use of MATLAB's visualization functionalities.
- For testing purposes, we use the Multi-Parametric Toolbox (MPT), an open-source collection of algorithms for MATLAB [HKJM13]. We use version *3.0.12* of MPT. MPT is distributed under the GPL [Fre].
- Again used only for testing, but also highly recommended, we use the GNU MPRF Library, a library for multiple-precision floating point computation [FHL+07]. While not necessary for the usage of the library, it is used for some examples and experiments. We used version *3.1.1-1* of the GNU MPFR library.

- In some examples, Christian Schneider's small wrapper library *mpfr::real* [Sch] is used. While strictly optional, the library, which is also distributed under the terms of the GPL provides a simple wrapper to arbitrary-precision computations.

Detailed instructions on how to include and use some of the mentioned software can be found in the documentation of the library. In the following chapter, we will conclude this thesis.

Chapter 5

Conclusion

Recent years have seen a rising popularity of forward reachability analysis of hybrid systems utilizing fixed-point approach. Thus, the need for reliable tools and libraries for solving the reachability problem using geometric state set representations has similarly increased. While a number of libraries is available, these are either not available under a free open-source license, do not allow arbitrary precision computations, or do not implement the zonotopes set representation. To the best of our knowledge, no such open-source library meeting this requirements has been made available yet.

In this thesis, we identified the demand for an implementation of an open-source library for zonotope-related operations in model checking. Based on these demands, we presented an extendable open-source library for operations on and with sets represented by zonotopes. The library allows for all basic operations concerning zonotopes, and further extends these with number of utility functionalities, further extending its functionalities by realizing Le Guernic's efficient algorithm for zonotope-hyperplane intersection by over-approximation in two-dimensional vector-spaces. The implemented matrix library lays out groundwork for the zonotope library, and can be further extended to incorporate a large number of matrix functionalities.
Our library utilizes an object-based approach, achieving reusability and extensibility for future analysis and research. The utility functionalities allow for easy use in evaluation and can easily be adapted for other data formats than the ones already implemented. To assist the evaluation of our library we also conducted a number of experiments, which can be found in Chapter 4. These experiments and examples, which are included in the library, can easily be reproduced and allow for quick and reliable checking of the libraries functionalities.

We are confident, that the libraries implementation of state set representations and the intersection algorithm can support researchers analyzing hybrid systems. Our library with its combination of extensibility and its open-source approach can assist researchers using the forward fixed-point approach in reachability analysis.

5.1 Future work

While proceeding with the project, it quickly became clear that some functionalities – while useful – could not be implemented without exceeding the scope of this bachelor thesis. In particular, adding more functionalities to the matrix library, such as QR decomposition, LR decomposition and efficient algorithms for computing inverted matrices. These functionalities are not necessary for the operations and algorithms described in this work, but are generally considered essential functionalities of a matrix library.

Similarly, the implementation of more state set representations, such as boxes, general polytopes or ellipsoids, would be a logical next step in developing a greater representation library. Some works in this direction have already begun. Furthermore, it might be interesting to utilize inheritance structures to implement future set representations, allowing for seamless usage of different set representations in reachability analysis.

Concerning the intersection algorithm described in Chapter 3, a further analysis of different choosing and generations strategies for the intersection algorithm's directions would be a very interesting topic. This would necessite a complete analysis of the quality of the over-approximations and an in-depth analysis using different data sets for the underlying sets of points, which would exceed the scope of this thesis. Similarly, we would like to conduct a full evaluation on the library's efficiency (in particular the intersection algorithm) in comparison to similar implementations.
Nevertheless, we are confident to have contributed a useful library for research and reachability analysis.

Bibliography

[ACH+95] R. Alur, C .Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[BEG+95] M. Bern, D. Eppstein, L. Guibas, J. Hershberger, S. Suri, and J. Wolter. The centroid of points with approximate weights. In Paul Spirakis, editor, *Algorithms — ESA '95*, volume 979 of *Lecture Notes in Computer Science*, pages 460–472. Springer Berlin Heidelberg, 1995.

[DT97] G. B. Dantzig and M. N. Thapa. *Linear programming 1: Introduction*, volume 1. Springer, 1997.

[FHL+07] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software*, 33(2):13:1–13:15, June 2007.
`http://www.mpfr.org`.

[Fre] Free Software Foundation. GNU General Public License, Version 3.
`http://www.gnu.org/licenses/gpl.html`.

[G+09] M. Galassi et al. The GNU Scientific Library Reference Manual (Third edition), 2009.
`http://www.gnu.org/software/gsl/`.

[Gla14] M. Glatki. A C++ library for the computation with zonotopes: User's manual, January 2014.
`http://www-i2.informatik.rwth-aachen.de/i2/hybrid_research_pub0/`.

[GNZ03] L. J. Guibas, A. Nguyen, and L. Zhang. Zonotopes as bounding volumes. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 803–812, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[Gue09] C. Le Guernic. Reachability analysis of hybrid systems with linear continuous dynamics. *University Joseph Fourier*, 2009.

[HKJM13] M. Herceg, M. Kvasnica, C. N. Jones, and M. Morari. Multi-Parametric Toolbox 3.0. In *Proc. of the European Control Conference*,

pages 502–510, Zürich, Switzerland, July 17–19 2013. http://control.ee.ethz.ch/~mpt.

[KS40] M. Krein and V. Smulian. On regulary convex sets in the space conjugate to a Banach space. In *Annals of Mathematics*, volume 41 of *Second Series*, pages 556–583. Annals of Mathematics, 1940.

[KV00] A. B. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In *Hybrid Systems: Computation and Control*, pages 202–214. Springer, 2000.

[MAT13] MATLAB. *version 8.1.0.604 (R2013a)*. The MathWorks Inc., Natick, Massachusetts, 2013.

[Sch] C. Schneider. A high-level C++ wrapper for the GNU MPFR library. http://chschneider.eu/programming/mpfr_real/.

[The13a] The GCC team. GCC, the GNU Compiler Collection, 2013. http://gcc.gnu.org.

[The13b] The GCC team. GNU Standard C++ Library, 2013.

[TWC01] J. Tretmans, K. Wijbrans, and M. Chaudron. Software engineering with formal methods: the development of a storm surge barrier control system. *Formal Methods in System Design*, pages 195–215, 2001.

[Wei13] E. W. Weisstein. Half-space. *MathWorld – A Wolfram Web Ressource*, 2013. http://mathworld.wolfram.com/Half-Space.html.

[Zie95] G. Ziegler. *Lectures on polytopes*, volume 152. Springer, 1995.

[M+13] E. Ábrahám, X. Chen, I. Ben Makhlouf, S. Kowalewski, and S. Sankaranarayanan and. A toolbox for the reachability analysis of hybrid systems using geometric approximations (HyPro), December 2013. http://www-i2.informatik.rwth-aachen.de/i2/index.php?id=733.