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

BACHELOR OF SCIENCE THESIS

OVER-APPROXIMATIVE REDUCTION OF POLYTOPES IN THE CONTEXT OF HYBRID SYSTEMS REACHABILITY ANALYSIS

Igor Nicolai Bongartz

Examiners:
Prof. Dr. Erika Ábrahám

Additional Advisor:
Prof. Dr. Jürgen Giesl

Aachen,
February 29, 2016

Abstract

Flowpipe-based reachability analysis for hybrid systems allows to over-approximate the set of reachable states of a given hybrid automaton to verify safety of a modeled system. Execution time and memory consumption of this analysis are influenced by the complexity of the underlying state sets used. In this work I focus on over-approximative reduction of polytopes as state set representations to ameliorate the execution time and the memory consumption of a reachability analysis algorithm. The introduced modifications show that an improvement is possible but requires careful selection of suitable strategies and parameters for reduction.

Eidesstattliche Versicherung

Name, Vorname	Matrikelnummer (freiwillige Angabe)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum	Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Ort, Datum	Unterschrift

Acknowledgements

I am thankful for the opportunity to write this bachelor thesis. Developing this academic work taught me multiple areas of research. The topic required creativity, formal specification of my ideas, their implementation and the empirical examination of the approach. This was the first time I had to unite all these processes.

I want to thank all the people which supported me during these five months of excessive *research*. First I mention Stefan Schupp. He was my supervisor for this thesis and helped me out in many situations. Next I thank all the people at the i2 who listened to me or explained me some details about hybrid systems or the complex universe of C++. A big thanks goes to Erika Ábrahám, who made this work possible and to my additional advisor Jürgen Giesl.

Furthermore my family has been a great help in several ways. For example the idea to motivate me with a delicious *mousse au cholocat* as reward. Or my aunt Ana and her husband Dan, who despite their own work had the time to read through 40 pages of theoretical *mess* (at this time).

And at the end, thank you Saskia for your daily motivation, advice, warning and everything else.

Contents

Chapter 1

Introduction

»Alice is an architecture student and got her first job. She had to build a house of solid brick in Nehcaa. One problem is that she lives in Engoloc which lies around 1 hour's drive away. Alice was quite nervous about this project so she decided to prepare herself well. The next day she went to a free space and started building the house out of solid bricks. The construction took a long time but the efforts have been worth it. Alice created a beautiful house. She wanted to do it exactly like this in Nehcaa. After she woke up she packed her belongings and went to the house. There she noticed that the house was too big to be moved by hand. She would need a helicopter to move the house from Engoloc to Nehcaa. Angry about her bad planning she called a helicopter service and ordered the house to be carried to the actual construction site.«

We are surrounded by plenty of technical systems which are helpful but safety-critical at the same time. These systems offer an interface for users but are controlled by a technical architecture (e.g. a car or an elevator). The driver decides when the car accelerates but the cause of acceleration is not done by the power of his pure thinking. The acceleration is caused by the engine which can be damaged in several ways such that the safety of the driver is no longer guaranteed. An elevator receives only the desired floor of the building and behaves autonomous from this point on. The possible threats are for example a longer drive (from the ground floor to the fourth floor) or a failure of the control systems, thus a cutoff of the air supply.
These systems can have discrete states (on, off, up, down, go, stop, several levels) and a continuous development of physical components (position, speed, acceleration). Systems with such a structure are specified as *hybrid systems*. In general the safety of technical systems is tested due to multiple repetitions of simulations. But these executions are not sufficient. The system can still be insecure because the testing team stopped an experiment or one repetition before a failure would have been discovered. An alternative is the use of a formal verification method. Due to rules which describe the dynamics of a system one can compute the set of reachable states. Comparing these states with a set of bad states allows a judgment on the system. The application of formal methods requires an abstraction of hybrid systems. Such a specification of a hybrid system is for example a *hybrid automaton* and the computation of reachable states can be referred to as *reachability analysis*. The investigation of whether the system is secure or not is called *safety verification*.

The approach of this work focuses on a more efficient use of the data structures of reachability analysis which might offer an amelioration. The reachability analysis is a *hard* problem and, dependent on the dimension and complexity of the state set, the execution time can be long. The general time complexity class can not be influenced without changing the algorithm.
The result of a reachability analysis of a hybrid system can be large. In 2-dimensional space a result of a simple hybrid system does already require $\approx$ 500 kByte or up to $\approx$ 1,5 MByte. More complex systems cause greater results. It can be of interest, that the size of these results is reduced in context of storage.
In this work I present approaches to accomplish these improvements for a *flowpipe-based* reachability analysis of hybrid systems. To achieve this I study the reduction of polytopes. A reduction P_{red} is an over-approximation of an input polytope P and reduces the amount of data required to represent the reduction. Due to the reduced complexity of P_{red} further operations require less computational effort than with P. Of course an over-approximation influences the reachability analysis in misleading to wrong results. If a bad state is reached by the over-approximative analysis it is possible that a more exact computation of reachable states returns the opposite. Nevertheless if the over-approximative analysis deduces that the system is secure the original hybrid system is secure too.

In the first part (Chapter 2) I define hybrid automata, the reachability analysis, polytopes and two possible representations of polytopes. With this knowledge in mind I present several strategies how to reduce polytopes (Chapter 3). I then analyze the advantages and disadvantages of each strategy. Due to the outcome of my analysis I develop a simple heuristic for the use of the different strategies. In Chapter 4, I analyze the performance of the strategies dependent on specified criteria. Furthermore I embed the reduction of polytopes in a reachability analysis algorithm to achieve the ameliorations I presented in the prior passage. In Chapter 5 I summarize and discuss my results and give an outlook towards possible future improvements. The definition of the polytopes used in this work and results of experiments can be found in the Appendix A.

Chapter 2

Background

To apply formal methods on a given hybrid system, an abstract model of this system is required. Hybrid automata are a popular specification used to describe hybrid systems. In the following I provide required background information on hybrid systems and hybrid automata as well as reachability analysis as an approach towards their verification. Furthermore I give an introduction on convex polytopes as a state set representation for reachability analysis and definitions of set operations required in this context.

2.1 Hybrid Systems

A *hybrid system* combines continuous and discrete behavior. A simple example which I also use later for the experiments is a bouncing ball. If I have a ball in my hand I can let it fall from a predefined height and observe the behavior. First in falls down. After it hits the floor it bounces upwards. At some point the ball reaches its reversal point and falls down again. The height of this point is lower than the initial height the ball was dropped from. Then the procedure repeats until the ball lies on the ground. This small example is already a hybrid system. A common model for a hybrid system is a *hybrid automaton*. It is defined by its initial state, the actions which modify the continuous evolution of variables depending on the time t, an invariant which dictates a certain limit for the variables and *discrete*-transitions. This leads us to the formal definition of hybrid automata.

Definition 2.1. (Hybrid automata [HKPV95]) A hybrid automaton is a tuple $HA = (Loc, Var, Lab, Edge, Act, Inv, Init)$ with following meaning of its components:

- (Loc) The locations are defined by the finite set Loc. For the purpose of visualization they can be displayed as vertices of a graph.

- (Var) The variables are represented by the finite set Var, which describes continuous components of the system. The total number of variables determines the dimension d of a system. All variables can be described by a vector $x \in \mathbb{R}^d$. In combination with a location $l \in Loc$ a state $s \in \Sigma$ is defined as: $s := (l, x)$.

- (Lab) The finite set of labels Lab are assigned to the transitions as a signal caused by a system. They are used to synchronize compositional hybrid au-

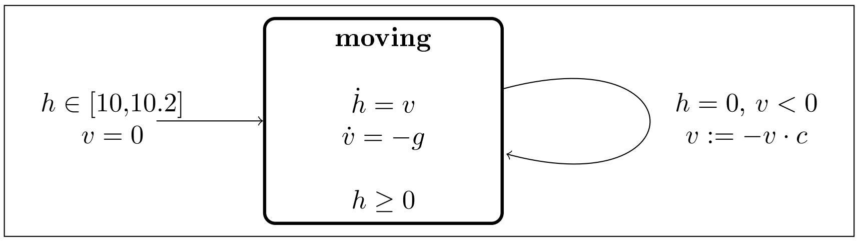


Figure 2.1: The hybrid automaton of a bouncing ball with $g = 9.81$ and $c = 0.75$.

tomata. If several automata are combined and transitions have to react on the same signal they are marked by a label.

- (*Edge*) The elements of the finite set *Edge* correspond to transitions between the vertices of the graph. The definition of an edge is:

$$e = (l, \iota, \mu_\iota, l') \text{ with } l{,}l' \in Loc, \iota \in Lab, \mu_\iota \in 2^{\mathbb{R}^d}$$

  The locations l and l' define the *source* and *target* location. The set $\mu_\iota := (x, x')$ defines the *guard* of the transition with x and the *reset* of all variables with x'. For each location each transition exists as a *stutter*-transition with an empty label $\varepsilon \in Lab$.

- (*Act*) The activities in the finite set *Act* define for each location l how each variable changes over time. An activity $a \in Act$ has the following definition:

$$a := (l{,}\dot{\xi}(t)) \text{ with } l \in Loc,\ \dot{\xi}(t) \in \mathbb{R}^d \times \mathbb{R}^d$$

  The differential equation $\dot{\xi}(t)$ describes the evolution for each variable due to the elapsing of time.

- (*Inv*) The finite conditions of the set *Inv* define valid assignments for variables of Var depending on the location. An element of *Inv* is $i := (l{,}x)$ with $l \in Loc$ and $x \in \mathbb{R}^d$.

- (*Init*) The finite initial state set *Init* defines all initial states of the hybrid system. An element of *Init* is defined as follows:

$$init := (l{,}x) \text{ with } l \in Loc$$

  The vector $x \in \mathbb{R}^d$ represents the initial valuation of all variables.

In a hybrid automaton the set of reachable states Σ is defined as the set of states reachable from an initial state by letting time elapse. A new state can be reached by the composition of two rules: A *time*-transition inside the current location and a *discrete*-transition which can lead to another location. The usage of these transitions is defined by following the rules:

$$\frac{e{=}(l{,}\iota{,}\mu_\iota{,}l')\in Edge \quad (x{,}x')\in\mu_\iota \quad (l'{,}x')\in Inv}{(l{,}x)\xrightarrow{e}(l'{,}x')} \qquad rule_{Discrete}$$

$$\frac{a{=}(l{,}\dot{\xi}(t))\in Act \quad \dot{\xi}(0)+x{=}x \quad \dot{\xi}(t)+x{=}x' \quad t{\geq}0 \quad \forall 0{\leq}t'{\leq}t\text{: } (l{,}\dot{\xi}(t')+x)\in Inv}{(l{,}x)\xrightarrow{t}(l{,}x')} \qquad rule_{Time}$$

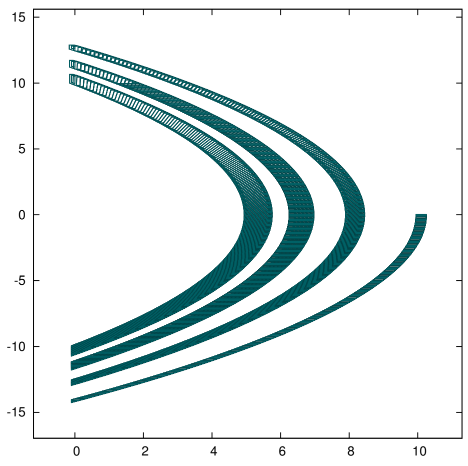


Figure 2.2: The flowpipes generated by a forward reachability analysis with $time_{step} =$ 10 ms and $time_{bound} = 4$ s for the bouncing ball model.

An *execution step* is the combination of a *discrete*- and a *time*-transition. Each state reachable by one or more execution steps from one of the initial states, called *execution path*, is reachable. With the definition of hybrid automata we are able to present a hybrid automaton for this system. A bouncing ball can be described by two variables: its height h and its velocity v. If we drop the ball at the initial height $h \in [10,10.2]$ it has an initial velocity $v = 0$. As long as the ball is falling it is subject to earths gravity, thus h decreases with the differential equation $\dot{h} = v$ while v decreases with $\dot{v} = -g$ and $g = 9{,}81\ m/s^2$. If $h = 0$ is reached our ball hits the floor. Therefore we introduce the invariant $h \geq 0$. If the guard is satisfied the ball bounces. This leads to a reset of the velocity $v = -v \cdot c$ with $0 < c < 1$ to simulate friction and elasticity of the ball. We set the constant to $c = 0.75$. Now v constantly decreases from its maximum and the procedure starts all over again until the ball lost all of its energy [Gue09] [E. 16b]. The hybrid automaton is displayed in Figure 2.1.

2.2 Reachability Analysis of Hybrid Systems

The *reachability analysis* computes the set of reachable states of a hybrid system. The *safety verification* uses the result of reachability analysis to compare the set of reachable sets with a set of bad states. If we start our computation from an initial state we call it *forward* reachability analysis. An alternative is the *backward* reachability analysis. There we start from a specified state (e.g. a bad state) and compute the set of states backwards. In this work I focus on the forward version as used by the C++ library HyPro which is part of an ongoing research project in the group theory of hybrid systems [E. 16a]. If we use reachability analysis with safety verification to our initial bouncing ball example we can say that on the floor lies a cat which has a size of 1. The bad state is reached if the ball drops to a height $h \leq 1$. Does the ball hit the cat? For answering this question we can make use of reachability analysis results.

The analysis evaluates *time*-transitions with a time discretization $time_{step}$ which defines the smallest processing of time for the computations. First an initial reachable set R is defined by all initial states which do not violate their invariants (Line 1). From this point on the algorithm iterates by alternating between *time*- and *discrete*-transitions (Line 5). Step by step R increases. If R does not increase anymore a fixed-point is reached and the analysis is finished (Line 7). If R intersects with the set of bad states the analysis is terminated. Introducing a local timebound allows to enforce termination in case a fixed-point can not be detected. This algorithm can be defined formally as follows:

Definition 2.2. (Forward Reachability Analysis [E. 15]) The structure of the algorithm is defined by Algorithm 1 with *forward time closure* defined as

$$T_l(R) = T_l(s := (l,x)|\ \forall s \in R) = \{(l,x')|\ \forall(l,x),(l,x')\exists t:\ (l,x) \xrightarrow{t} (l,x')\}$$

and the *postcondition*

$$D_e(R) = D_e(s := (l,x)|\ \forall s \in R) = \{(l',x')|\ \forall(l,x),(l',x')\exists e:\ (l,x) \xrightarrow{\iota} (l',x')\}.$$

Algorithm 1 Forward Reachability Analysis

```
1: $R = \{s \in Init | s \in Inv\}$
2: $R = T_l(R)$
3: $R_{new} = R$
4: while ($R_{new} \neq \emptyset$) do
5:     $R_{new} = T_{l'}(D_e(R))$
6:     if ($R_{new} \subseteq R$) then
7:         return $R$
8:     end if
9:     $R = R \cup R_{new}$
10: end while
11: return $R$
```

The reachability analysis makes use of state set representations in order of representing sets of reachable states. For example, an invariant can be encoded as an inequality and a reset as an equation. An alternative we are interested in is the reachability analysis using geometric shapes.
The first segment of a flowpipe is a geometric shape we obtain after an initialization procedure (Figure 2.3). We work with a fixed $time_{step}$ and dependent on the continuous behavior of the system we lose information between the segments. Without the initialization the set of reachable states does not represent the state space of the hybrid system [Gue09]. First we perform a *time*-transition from the initial state *init* (1) and compute the convex hull of these two shapes (2). Next we bloat the convex hull (3) in order to over-approximate the dynamics of the hybrid systems which are represented as curved lines in the Figure. If the system is non-autonomous we apply a second bloating to consider the external influence (4). Every bloating is performed as a *Minkowski*-sum of the current shape and a box.
A *time*-transition is computed as follows (Figure 2.4a): Linear dynamics of the hybrid system are computed as a linear transformation of the segment s and return the segment $s_{\Delta t}$. If the hybrid system contains non-linear dynamics other techniques are used to calculate the set of reachable states. The new segment represents the state

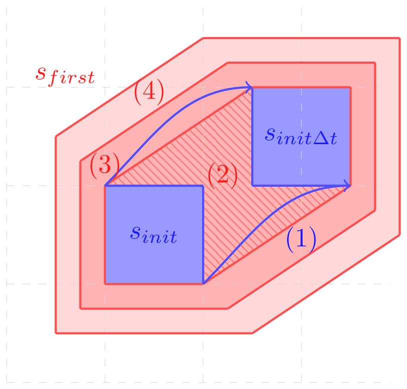
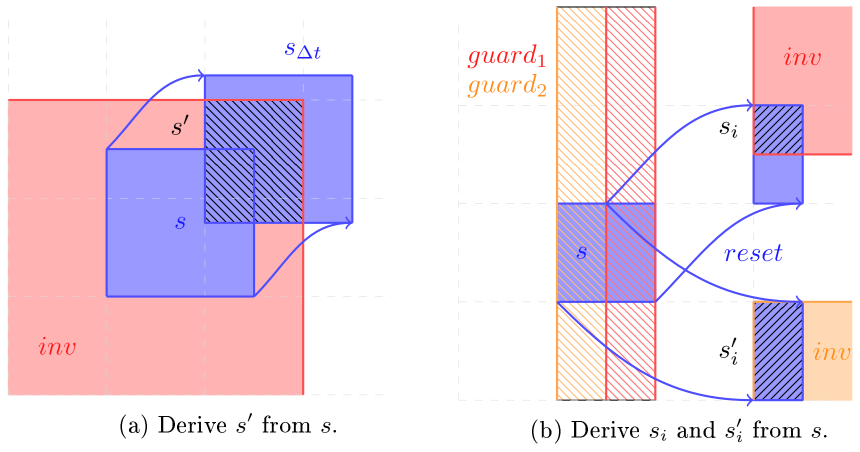


Figure 2.3: The preparation of the first segment s_{first} with $init \in Init$.

reached after a timestep from s. It is possible that the new segment violates the invariant of the current location. Therefore we compute the intersection of $s_{\Delta t}$ and the invariant inv of the location and receive the result of a *time*-transition s'. All segments computed for one location are referred to as a flowpipe. Thus, the result of reachability analysis is a set of flowpipes. Due to the definition of the reachability analysis we alternate between *time*- and *discrete*-transition.
In order to perform a *discrete*-transition we have to consider the guard which enables the discrete jump for a state s, the reset and the invariant of the new location (Figure2.4b). We intersect the computed segments with all guards of outgoing transitions of the current location. If the intersection is not empty we compute a linear reset as a linear transformation. After the reset we intersect the reseted segment with the invariant of the location and obtain its initial set s_i.
The reachable set of the analysis is the set of all computed segments. If any of these segments intersects with the bad states the analysis is terminated. If any new segment is fully contained in an older segment a fixed-point is reached.
The reachability analysis for hybrid automata is undecidable. Over-approximation is necessary for the reachability analysis in order compute the *correct* reachable set of states of a hybrid system. If the reachability analysis concludes that the hybrid automaton is safe this property does also hold for the hybrid system. Every more specific over-approximation of reachable states leads to the same result. However over-approximation causes a drawback. If the analysis returns the insecurity of the systems the hybrid system is not necessarily insecure because the responsible state could have been added due to over-approximation.

2.3 Representations of Polytopes

There are several possible types of state representations (e.g. zonotopes, support-functions, boxes [E. 15]) as which I refer to as geometrical shapes. In this work I consider convex polytopes. Reduction of polytopes is intuitive, it has been done already [Fre08] and polytopes are used by the C++ library HyPro which I use to analyze my approach. Moreover, polytopes are the most complex type of representation and methods using other representations have the same or smaller time complexity. Such that, if we achieve an improvement of reachability analysis the amelioration works for



(a) Derive s' from s. (b) Derive s_i and s'_i from s.

Figure 2.4: The continuous and discrete behavior of a hybrid system are described as *time-* (a) and *discrete*-transitions (b).

other state set representations too.
Before I specify the reduction in the next chapter I define *polytopes* and *operations* on sets of polytopes required for reachability analysis. These definitions prepare the reader to understand the formal definition of reduction and its usage in reachability analysis. I restrict myself to polytopes and not the more general case of polyhedron, because in context of reachability analysis polytopes are more likely than polyhedra and the presented approaches can be modified to work with polyhedra. Polytopes are bounded polyhedra and allows to specify subsets of a d-dimensional space as an intersection of a finite set of half-spaces. Each half-space is represented as a linear constraint.

Definition 2.3. (Polytopes [Zie95]) Polytopes are the bounded convex set in a d-dimensional space specified as the intersection of a finite set of half-spaces $\mathcal{H} = \{h_1, \dots, h_H\}$. Each half-space $h \in \mathcal{H}$ is defined as $h = \{x \in \mathbb{R}^d | n \cdot x \leq o\}$ with the *normal*-vector $n \in \mathbb{R}^d$ and the offset $o \in \mathbb{R}$, such that

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

Each polytope can be represented by its half-spaces. This representation is called the $\mathcal{H}$-representation and the corresponding polytope a $\mathcal{H}$-polytope. The set of points which fulfill the equation $n^T \cdot p = o$ for all $p \in \mathbb{R}^d$ is specified as *hyperplane*. Note that a point and a *normal*-vector determine the offset of a half-space. The common way to implement a $\mathcal{H}$-polytope is to transform the set $\mathcal{H}$ into a matrix $N \in \mathbb{R}^{H \times d}$ and a vector $o \in \mathbb{R}^H$. Every row N_i represents a *normal*-vector n_i and every i-th entry in o stands for the offset o_i of the half-space $h_i \in \mathcal{H}$. The polytope P can be defined by the inequality:

$$P := \{x \in \mathbb{R}^d | N \cdot x \leq o\}.$$

An alternative is the $\mathcal{V}$-representation defined by a finite set of points, called vertices $\mathcal{V}$, which represents the convex hull of a $\mathcal{V}$-polytope P. The number of vertices

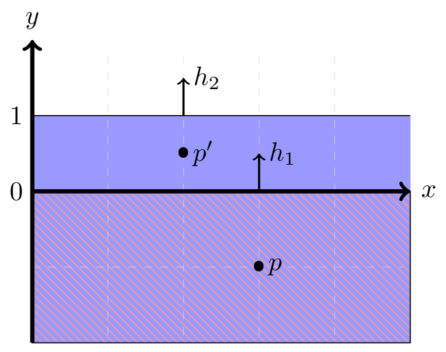


Figure 2.5: The half-spaces h_1 and h_2 with $p \in h_1, h_2$ and $p' \notin h_1, p' \in h_2$.

is V. The corresponding definition of P is:

$$P := \left\{ \sum_{v \in \mathcal{V}} \lambda_v \cdot v | \lambda_v > 0 \; and \sum_{v \in \mathcal{V}} \lambda_v = 1 \right\}.$$

It can happen that some half-spaces are redundant due to the existence of others. Redundancy in this context means, that the half-space does not provide required information to define the polytope. Let us consider following half-spaces h_1 and h_2

$$h_1 : n_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, o_1 = 0 \text{ and } h_2 : n_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, o_2 = 1$$

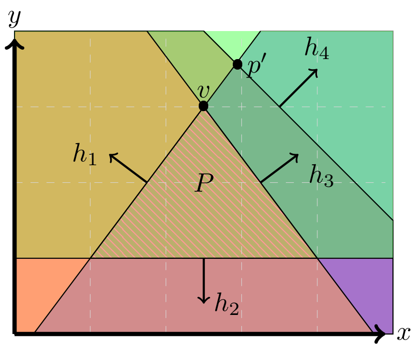
The half-space h_1 defines a subset of h_2, because all points $p \in \mathbb{R}^2$ which satisfy $n_2^T \cdot p \leq 1$ also satisfy $n_1^T \cdot p \leq 0$. But a point $p' \in \mathbb{R}^2$ with $n_1^T \cdot p = 0.5$ is still inside the half-space h_2 but not in the half-space h_1. Figure 2.5 shows these circumstances. Therefore h_2 is redundant if we consider the set $\mathcal{H} = \{h_1, h_2\} = \{h_1\}$. It is possible that a combination of half-spaces causes another half-space to be redundant as shown in Figure 2.6a. Similar to a redundant half-space I define a redundant point p as a point which does not belong to the vertices of a polytope $\mathcal{V}_P$ (Figure 2.6b).

Definition 2.4. (Redundant half-space or point) A half-space h of a set $\mathcal{H}$ is called *redundant* iff $P_{\mathcal{H}/\{h\}} = P_{\mathcal{H}}$ and therefore $\mathcal{H} \subset \{h\}$. A point p of a set $\mathcal{V}$ is called redundant iff $P_{\mathcal{V}/\{p\}} = P_{\mathcal{V}}$.

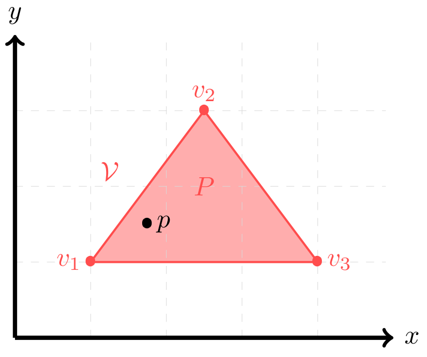
Every non-redundant half-space h of a $\mathcal{H}$-polytope determines a facet f_h which represents the non-redundant section of h. A facet is a $(d-1)$-dimensional object. The volume of $hyperplane_h$ is infinite while the volume of $hyperplane_f$ is finite. The intersection of two affine independent half-spaces is a $(d-2)$-dimensional object usually referred to as ridge r [Zie95]. For $d = 2$ a facet of a polytope is a bounded line and a ridge a point (Figure 2.7).

2.4 Operations on Polytopes

Multiple operations are required in the context of flowpipe-based reachability analysis to manipulate one or more polytopes. The complexity of these operations strongly depends on the used representation type. An *easy* time complexity means that the



(a) The half-space h_4 is redundant.



(b) The point p is redundant.

Figure 2.6: A polytope P is defined as a $\mathcal{H}$- (a) and a $\mathcal{V}$-polytope (b).

execution time is estimated as a polynomial execution time. If a time complexity is *hard* the execution time is estimated as an exponential time in the dimension d. Important for most operations is the conversion of a polytope P between its $\mathcal{H}$- and $\mathcal{V}$-representation, called *facet* and *vertex enumeration problem* [AB95]. This is a *hard* problem [E. 15].
If we want to convert the $\mathcal{H}$-representation of P into its $\mathcal{V}$-representation we have to compute all vertices of P. To achieve this we consider all permutations of d half-spaces $\{h_1, \ldots, h_d\}$ with $h_i \in \mathcal{H}$. Each permutation determines a point p. If the inequality $N \cdot p \leq o$ given by the matrix definition of P holds, p is a vertex v and belongs to $\mathcal{V}$. Otherwise it is a point caused by redundant half-spaces and does not belong to the vertices which determine the convex hull of P (point p' in Figure 2.6a). This conversion can be done in $O(H^d)$ [D. 92].
On the contrary, for the conversion of the $\mathcal{V}$-representation into its $\mathcal{H}$-representation, we have to compute the convex hull of all vertices $v \in \mathcal{V}$ to specify the half-spaces of P. The execution time of this conversion has the same time complexity as the determination of vertices [AB95]. The *convexHull*-algorithm used by C++ library HyPro is implemented according to *Barber et al.* [CB96]. In the following section I define operations on sets of polytopes essential for the flowpipe-based reachability analysis: *Membership*, *union*, *intersection*, *Minkowski-sum* and *linear transformation* [E. 15].

- (Membership) The membership of a point $p \in \mathbb{R}^d$ can be solved in linear time dependent on d for $\mathcal{H}$-polytopes. If $N \cdot p \leq o$ holds for p the point lies inside the polytope.

  If we want to compute the membership for a $\mathcal{V}$-polytope we have to solve following *linear programming problem*:

$$\exists \lambda_{v_1}, \ldots, \lambda_{v_H} : p = \sum_{v_P \in \mathcal{V}} \lambda_{v_P} \cdot v_P,\ \lambda_{v_P} \in [0,1] \text{ and } \sum_{v_P \in \mathcal{V}} \lambda_{v_P} = 1$$

- (Union) A union of two polytopes is in general not convex and therefore no convex polytope anymore. Thus we define the union of two polytopes as their convex hull.

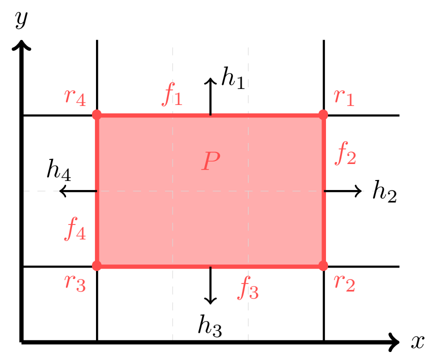


Figure 2.7: Facets describe the relevant part of half-spaces.

The unification of two $\mathcal{V}$-polytopes is the union of all vertices, which can consist of redundant points.

$$P_1 \cup P_2 := \{v | \, \forall v \in \mathcal{V}_{P_1} \cup \mathcal{V}_{P_2}\}$$

The union of two $\mathcal{H}$-polytopes is computationally expensive because the united polytope consists of new half-spaces. One possible algorithm converts the two $\mathcal{H}$-polytopes into their $\mathcal{V}$-representations. After the application of the union for $\mathcal{V}$-polytopes the result is converted back into a $\mathcal{H}$-polytope.

- (Intersection) The intersection of two $\mathcal{H}$-polytopes is the unification of their half-spaces, which can result in redundant half-spaces.

$$P_1 \cap P_2 := \{h | \, \forall h \in \mathcal{H}_{P_1} \cup \mathcal{H}_{P_2}\}$$

If we compute an intersection of two $\mathcal{V}$-polytopes, we follow a similar procedure as for $\mathcal{H}$-polytopes in union. First the $\mathcal{V}$-polytopes are converted into their $\mathcal{H}$-representation. Afterwards we compute the intersection for this representation and transform the result back into a $\mathcal{V}$-polytope.

- (*Minkowski*-sum) The *Minkowski*-sum of two polytopes is defined as the addition of all elements of the two polytopes:

$$P \oplus P' := \{a + b | \, \forall a \in P, b \in P'\}$$

The *Minkowski*-sum of two $\mathcal{V}$-polytopes is a polytope described by the set of points $Points_M$.

$$Points_M := \{v + v' | \, \forall v \in \mathcal{V}_{\mathcal{P}}, \, v' \in \mathcal{V}_{\mathcal{P}'}\}$$

A possible version of the *Minkowski*-sum of two $\mathcal{H}$-polytopes is the *Minkowski*-sum of their $\mathcal{V}$-representations.

- (Linear Transformation) A linear transformation can move, scale and/or rotate a polytope P in the d-dimensional space. In general a linear transformation of a point $p \in \mathbb{R}^d$ is defined as follows:

  $$A \cdot p + b = p' \text{ with } A \in \mathbb{R}^{d \times d}, b \in \mathbb{R}^d$$

  This is easy to realize for $\mathcal{V}$-polytopes. The transformed polytope P' can be defined by the new set of points $\mathcal{V}'$ computed as follows:

  $$\mathcal{V}' := \{v' = Av + b | \; \forall v \in \mathcal{V}_P\}$$

  If we want to transform a $\mathcal{H}$-polytope the only consistent element is the number of half-spaces $\mathcal{H}$. But every half-space can be modified due to the linear transformation. Therefore we apply the linear transformation on the $\mathcal{V}$-representation of the $\mathcal{H}$-polytope.

Chapter 3

Reduction

»The first job in Nehcaa went fine after her house arrived. However the responsible person was annoyed due to the money spent on the helicopter service. Later Alice got a phone call and received her second job in Nehcaa. Without hesitation she ran to her desk and started brainstorming. Alice thought, that of course the plan-house has to become smaller such that she could carry it to Nehcaa without any additional effort. First she planned to construct only the relevant walls without all the gadgets. Then she stood up and monologized about different lighter material and sat down again. "Maybe I should build a small version of the house of plastic... but it could be easily broken", Alice said. Then suddenly she jumped up and got a bunch of paper. How about drawing the house!«

The size of a $\mathcal{H}$-polytope P is proportional to the number of half-spaces H and the dimension d plus different overheads of the polytope OH_P, a half-space $OH_{halfspace}$ and the number type OH_{Number}

Definition 3.1. (Size of a $\mathcal{H}$-polytope P)

$$size(P) = OH_P + H_P \cdot (OH_{halfspace} + d_P \cdot OH_{Number})$$

The definition will be verified in the next chapter using the software tool MASSIF from VALGRIND [Dev16]. The critical part of a $\mathcal{H}$-polytope is the number of half-spaces and their overhead which is large in high dimensions. Moreover the number of half-spaces of polytopes grows with operations like the intersection (Section 2.4) influencing contrary the execution time of operations.
The reduction of polytopes is similar to the approaches of *Alice* to simplify her plan-house. The new versions require less space and the drawing on paper enables a faster planning. Similarly, a reduction of half-spaces offers two advantages. First, a smaller $\mathcal{H}$-polytope consumes less memory. Second, the operations can be done faster if less half-spaces have to be compared with guards, invariants or bad states.
The reduction of half-spaces leads to a reduction of the polytope P which I refer to as P_{red}. The reduction P_{red} has to be an over-approximation to preserve the correctness of any safety verification algorithm (Figure 3.1).

$$reduce(P) \rightarrow P_{red} \text{ with } P \subseteq P_{red}$$

This condition indicates the disadvantage of reduction. Alike the reachability analysis, the segments have to be over-approximated. But the result of an algorithm using over-

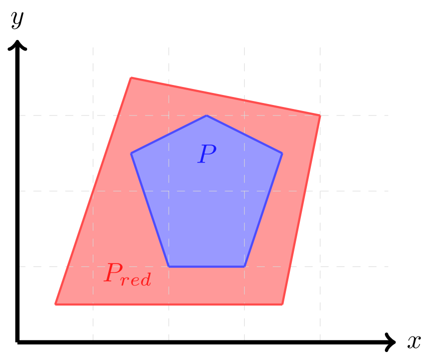


Figure 3.1: A polytope P and a possible over-approximative reduction P_{red}.

approximations is not strictly the result of the original system. Reductions are one additional over-approximation to the already over-approximated set representations. We exchange an improvement in speed and memory consumption against a loss of precision. To compute a reduction the original polytope P has to be considered, regarding its half-spaces $\mathcal{H}$ and vertices $\mathcal{V}$. Half-spaces of P have to be dropped or replaced by new half-spaces to achieve a reduction. This can be done by multiple different strategies.
I focus on reduction in the context of reachability analysis for hybrid systems, thus I define different criteria with following parameters:

- $Time_{red}$: $minimize(\ time(reduce(P))\ )$
- $Memory_{[P \to P_{red}]}$: $minimize\left(\frac{size(P_{red})}{size(P)}\right)$
- $Volume_{[P \to P_{red}]}$: $minimize\left(\frac{volume(P_{red})}{volume(P)}\right)$ with $\frac{volume(P_{red})}{volume(P)} \geq 1$

If we regard only one criterion we get reductions we might not want. As an example, we define an algorithm to compute the smallest P_{red} possible ($Memory_{[P \to P_{red}]}$). This yields a triangle in 2-dimensional space, a tetrahedron in 3-dimensional space and so on (a polytope with the smallest number of half-spaces possible in the dimension). But $volume(P_{red})$ can be much greater than $volume(P)$ and therefore the reduction causes an intersection with a bad state in the context of reachability analysis of hybrid systems. In this case we have to include the criterion $Volume_{[P \to P_{red}]}$. Another point which has to be considered is the context in which the reduction is embedded. If we are interested to store the result of a reachability analysis the criterion $Time_{red}$ is of no interest. But if the reduction is used inside the reachability analysis the algorithm has to be fast. There exists several applications of the reduction algorithm only in the reachability analysis.
Besides hybrid systems the reduction of polytopes can be used as a pure graphical tool to smooth objects too. If operations create polytopes with many half-spaces, an over-approximative reduction can be used as a smoothing tool. Graphical programs are known to consume a lot of memory and the smoothed polytopes requires less.
All these applications share some similar criteria but on the other hand they demand an unique combination.

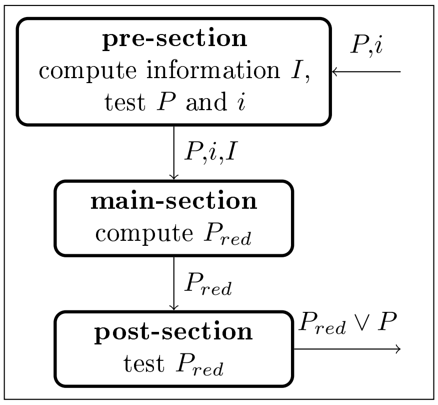


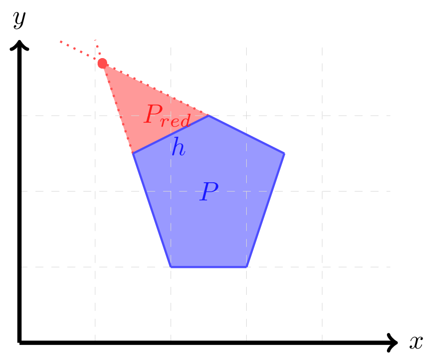
Figure 3.2: The general structure of a reduction for a $\mathcal{H}$-polytope P and input parameters i.

3.1 Strategies

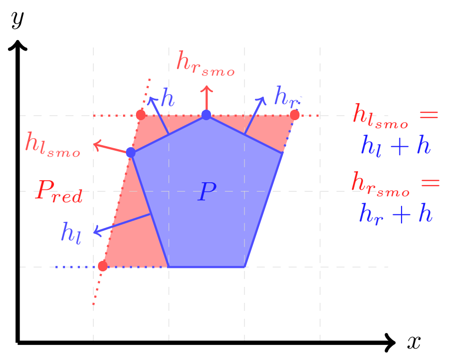
I developed nine strategies which can be assigned to 3 different families. All strategies determine the set S:

- A *drop*-strategy erases a half-space h from P.
  - *drop* performs the basic idea.
  - *dropSmooth* additionally manipulates the neighbors of h.
- A *unite*-strategy computes a new half-space as an average of several half-spaces from P and replaces them.
  - *unite* focuses on two half-spaces a and b.
  - *uniteSmooth* additionally considers the neighbors of a and b.
  - *uniteVertices* uses the vertices of a and b.
  - *uniteWeight* weights a and b with their volume.
- A *directed*-strategy replaces concerned half-spaces by specific directions given as parameters, so that the appearance of P_{red} is influenced.
  - *directedLocal* removes the locally concerned half-spaces.
  - *directedHorizon* removes the globally concerned half-spaces.
  - *directedTemplate* replaces P by an adjusted template polytope.

Each version of reduction can be split up in its *pre*-, *main*- and *post*-section (Figure 3.2). During the *pre*-section the required information of a $\mathcal{H}$-polytope P is computed: The half-spaces $\mathcal{H}$, the vertices $\mathcal{V}$ and a map *membersOfVertices* to know which half-space belongs to which vertex and therefore the neighborhood information of half-spaces. The number of half-spaces is defined as H. While the half-spaces belong to a $\mathcal{H}$-polytope the vertices are computed by the conversion described in Section 2.4 in $O(H^d)$. The map *membersOfVertices* is determined as follows: for each vertex v we deduce all half-spaces $h \in \mathcal{H}$ which fulfill the equality $n_h \cdot v = o_h$. The



(a) Drop the half-space h.



(b) Drop h and smooth the neighbors.

Figure 3.3: The strategies *drop* (a) and *dropSmooth* (b).

corresponding runtime is $O(V^H)$. If no information is needed the *pre*-section is dispensable.
The *main*-section of the reduction uses the informations to modify P in a specified way to obtain P_{red}. The reduced polytope can be unbounded and has to be examined in the *post*-section before the reduction returns a *result*-polytope. The time complexity of each strategy refers to the *pre-* and *main*-section, because depending on the application an unbounded reduced polytope can be wanted.

Characteristics for each strategy are:

- Input parameters
- Number of dropped half-spaces $drop = H_P - H_{P_{red}}$ with $H_{P_{red}} < H_P$ if we disable the option to return a greater reduction
- $time(\ reduce(P)\ )$

3.1.1 Drop a half-space of *P*

The strategy *drop* erases a requested half-space h of P.

$$reduce_{drop}(P, h) \rightarrow P_{red} \text{ with } h \in \mathcal{H}$$

This strategy does not need any preparation in the *pre*-section. In the *main*-section the half-space h is taken from P and we obtain P_{red}. Therefore this reduction can be done in constant time (Figure 3.3a).

Characteristics of $reduce_{drop}$:

$$\begin{aligned} drop_{drop} &\in \{0,1\} \\ time(\ reduce_{drop}\ ) &= O(1) \end{aligned}$$

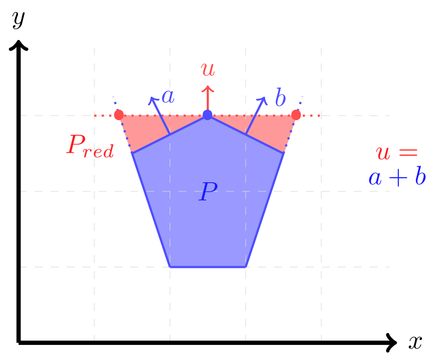


Figure 3.4: A polytope P reduced by *unite* with a and b.

3.1.2 Modification of *Drop*

The strategy *dropSmooth* is a modification of the strategy *drop* (see Section 3.1.1) and replaces each neighboring half-space n of h by a smoothed version n_{smo} additionally to the deletion of h.
During the *pre*-section we collect all neighbors of h by calculating the map *membersOfVertices*. This information is used in the *main*-section to compute all smoothed versions of the neighbors. I compute an addition of two half-spaces n and h as the addition of their offsets and *normal*-vectors. The *normal*-vector of n_{smo} is the average of n and h, therefore closer to the *normal*-vector of h. All neighbors and h are deleted and all smoothed neighbors are inserted (Figure 3.3b).

Characteristics of $reduce_{dropSmooth}$:

$$\begin{aligned} drop_{dropSmooth} &\in \{0,1\} \\ time(\ reduce_{dropSmooth}\ ) &= O(H^d) \end{aligned}$$

3.1.3 Unite half-spaces of P

The strategy *unite* replaces two neighboring half-spaces a and b with a half-space u which represents their average.

$$reduce_{unite}(P,\ a,\ b) \rightarrow P_{red} \text{ with } a, b \in \mathcal{H}$$

During the *pre*-section we verify if a and b are neighbors. In most cases it is impossible to know the neighbor informations of the half-spaces beforehand. If this test holds the reduction proceeds with the *main*-section.
The new *union half-space* u is the addition of a and b (as the addition in Section 3.1.2). Then a and b are replaced by u (Figure 3.4).

Characteristics of $reduce_{unite}$:

$$\begin{aligned} drop_{unite} &\in \{0,1\} \\ time(\ reduce_{unite}\ ) &= O(H^d) \end{aligned}$$

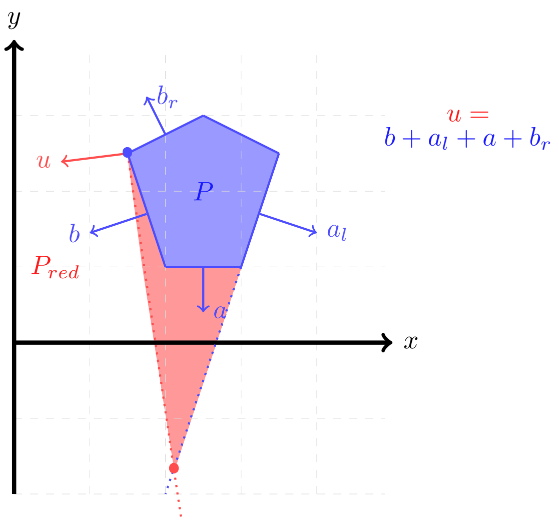


Figure 3.5: A polytope P reduced by *uniteSmooth* with a and b.

3.1.4 First Modification of *Unite*

The strategy *uniteSmooth* extends the strategy *unite* (see Section 3.1.3). Instead of computing the average of the half-spaces a and b this strategy first modifies these half-spaces.
The *normal*-vector of the new *union half-space* $n_{u_{smo}}$ is the addition of $n_{a_{smo}}$ and $n_{b_{smo}}$. Both vectors $n_{a_{smo}}$ and $n_{b_{smo}}$ are determined by the sum of all neighboring normalized *normal*-vectors:

$$n_{x_{smo}} = n_{nei_{x_1}} + n_{nei_{x_2}} + \ldots + n_{nei_{x_{N_x}}}$$

$$\text{with } x \in \{a,b\},\ N_x = |neighbors_x|,\ nei_{x_i} \in \{neighbors_x\} \text{ and } i \in \{1,\ldots,N_x\}$$

The offset $o_{u_{smo}}$ of the new half-space is determined by evaluating every $v \in \mathcal{V}$ against the new plane such that $\forall v :\ n_{u_{smo}}^T \cdot v \leq o_{u_{smo}}$ and $\exists v :\ n_{u_{smo}}^T \cdot v = o_{u_{smo}}$. With v and $n_{u_{smo}}$ I generate u_{smo} as presented in Section 2.3. Then a and b are replaced by u_{smo} (Figure 3.5).

Characteristics of $reduce_{uniteSmooth}$:

$$\begin{aligned} drop_{uniteSmooth} &\in \{0,1\} \\ time(\ reduce_{uniteSmooth}\ ) &= O(H^d) \end{aligned}$$

3.1.5 Second Modification of *Unite*

Contrary to the basic strategy *unite* (see Section 3.1.3) the modification *uniteVertices* does not focus on the directions of a and b. Instead it determines the vertices the half-spaces a and b belong to. A permutation of d vertices determines a hyperplane. The *normal*-vectors of these hyperplanes influence the new half-space.
The new *union half-space* u_{ver} is computed as follows:

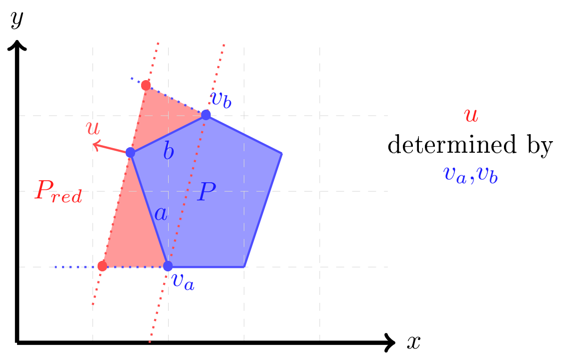


Figure 3.6: A polytope P reduced by *uniteVertices* with a and b.

1. Detect all vertices which belong to the half-spaces a and b, but not to both of them.
2. Create all permutations of d vertices.
3. Loop over all permutations and compute a potential *normal*-vector n_{pot} with the vertices determined by the current permutation. The resulting vector has the correct orientation if the scalar product of n_{pot} and a or b is greater than zero. Otherwise $n_{pot} = -n_{pot}$.
4. Update a vector n as the addition (as in Section 3.1.2) of n and the normalized version of n_{pot}. Then repeat the loop until all permutations are examined.

The *normal*-vector of u_{ver} is n. The correct offset $o_{u_{ver}}$ is computed as in Section 3.1.4. Then a and b are replaced by u_{ver} (Figure 3.6).

Characteristics of $reduce_{uniteVertices}$:

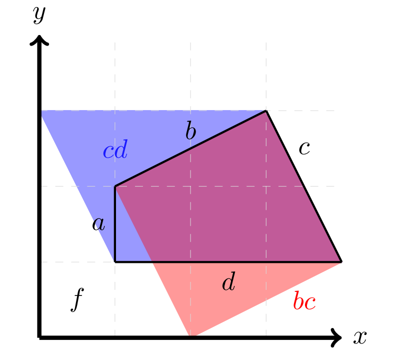
$$\begin{aligned} drop_{uniteVertices} &\in \{0,1\} \\ time(\ reduce_{uniteVertices}\ ) &= O(V^d) \end{aligned}$$

3.1.6 Third Modification of *Unite*

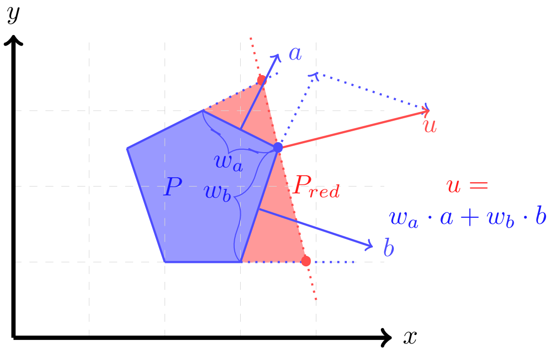
The strategy *uniteWeight* uses a weighting of the half-spaces a and b dependent of the volume of the facets f_a and f_b while the basic strategy *unite* 3.1.3 executes the pure addition.
The *normal*-vector $n_{u_{wei}}$ of the *union half-space* u_{wei} is the weighted addition of the normalized *normal*-vectors of a and b.

$$n_{u_{wei}} = w_a \cdot n_a + w_b \cdot n_b \text{ with } w_a,w_b \in \mathbb{Q}$$

Both weights represent the volume of the facet f_a and f_b of their half-spaces. In 2-dimensional space the volume is defined as follows: A facet f has two ridges which are at the same time two vertices V_1 and V_2 of P. The volume of f is the norm of $V_1 - V_2$. If the dimension of P is higher the weights are computed by an approximative method (Figure 3.7a). The norm of a crossproduct of two vectors represents the volume of



(a) Determine the average of norms.



(b) Compute the weighted average of a and b.

Figure 3.7: Volume-approximation of a 2-dimensional facet f with the ridges a,b,c,d (a) and the reduction with *uniteWeight* of a polytope P (b).

the spanned parallelotope. I determine the norms of all neighboring ridges of a facet. The average of these norms represents an approximative volume of the facet. For each facet I loop over all permutations of d neighboring vertices $\{v_1, \ldots ,v_d\}$. All permutation vertices determine one hyperplane but a corresponding *normal*-vector n with a particular norm. The vector n is computed as the cross-product of all *difference*-vectors $df_i = v_i - v_1 \; \forall i \in \{2, \ldots ,d\}$. The norms of these cross-products are added and normalized by dividing through the total number of cross-products. Finally we achieve the approximative volume $size_f$ of a facet f. These sizes are used as weights $w_a = size_{f_a}$ and $w_b = size_{f_b}$. The offset $o_{u_{wei}}$ is computed as in Section 3.1.4. Then a and b are replaced by u_{wei} (Figure 3.7b).

Characteristics of $reduce_{uniteWeight}$:

$$\begin{aligned} drop_{uniteWeight} &\in \{0,1\} \\ time(\, reduce_{uniteWeight}\,) &= O(2 \cdot V^d) \end{aligned}$$

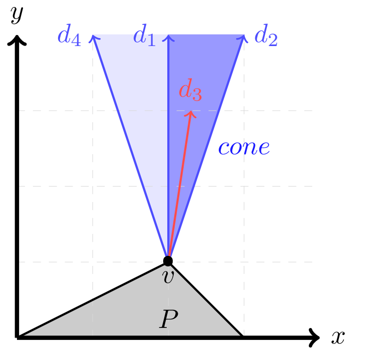
3.1.7 Replace Half-spaces of P by a Template Polyhedron

The strategy *directedLocal* receives a set of directions, computes the correct half-spaces and erases all *concerned* half-spaces of P.

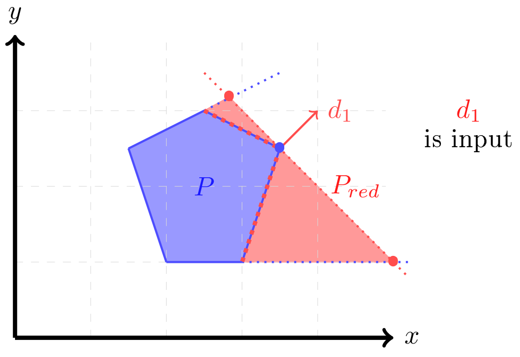
$$reduce_{directedLocal}(P, directions) \rightarrow P_{red}$$

with $directions = \{d_1, \ldots ,d_D\}$, $\forall d_i \in \mathbb{R}^d$ and D the number of directions given.

In the *pre*-section the vertices $\mathcal{V}$ and *membersOfVertices* are computed. Optionally we can test if $H_{P_{red}} \geq H_P$ holds and return P as the result. The final number of half-spaces $H_{P_{red}}$ depends on the P and *directions* and not simply on H and D. Even if $D \geq H$ holds, several directions can be redundant and $H_{P_{red}}$ is still smaller than H.
The *main*-section consists of a loop through all given directions:



(a) The cone defined by d_1 and d_2 is extended by d_4 but not by d_3.



(b) The direction d_1 is used as input and the dotted facets are locally concerned.

Figure 3.8: Usage of a cone to deduce redundant directions (a) and the reduction with *directedLocal* of a polytope P (b).

1. Find the correct vertex v_{d_i} for the current direction $d_i \in directions$ such that $d_i^T \cdot v_{d_i} = o_{d_i}$ and $d_i^T \cdot v \leq o_{d_i}$ for all other $v \in \mathcal{V}$.
2. Loop through all members of v_{d_i} in *membersOfVertices* and erase them from P.
3. Insert the new half-space into P.

The *directed*-strategies have one disadvantage against the other families. If more than d new directions are mapped to a vertex v the strategies create redundant half-spaces. They can be removed while reducing. Therefore a cone $cone_v$ is created for each vertex v of d half-spaces (Figure 3.8a). For details on polyhedral cones I refer the reader to Ziegler [Zie95]. If a new direction d_{new} is mapped to v we can test if d_{new} is inside the cone or not. Following a failed test we update the cone and remove the newly contained half-space. An alternative is to use the method *removeRedundantPlanes*(P_{red}) from C++ library HyPro after the reduction. This function examines a $\mathcal{H}$-polytope $P_\mathcal{H}$ while testing $P_{\mathcal{H}/\{h\}}$ for all $h \in \mathcal{H}$. If $P_{\mathcal{H}/\{h\}} = P_\mathcal{H}$ the half-space h is redundant. A renouncement of this test increases the speed of $reduce_s(P)$ and the size of P_{red} for strategies using template directions (Figure 3.8b).

Characteristics of $reduce_{directedLocal}$:

$$\begin{aligned} drop_{directedLocal} &\in \{drop_{begin}, \ldots, drop_{end}\} \\ time(\ reduce_{directedLocal}\ ) &= O(H^d) \end{aligned}$$

To determine a polytope in d-dimensional space we need at least $d+1$ half-spaces. Therefore we set $drop_{end} = H - d + 1$ if $D > H - d + 1$ and otherwise $drop_{end} = D$ due to the removing of only locally concerned half-spaces. If the option to return P if $H_{P_{red}} > H$ is enabled the bound $drop_{begin}$ is 0 and otherwise $H - D$ such that it can be negative.

3.1.8 First Modification of *Directed*

The strategy *directedHorizon* is defined as *directedLocal* 3.1.7 with one difference: Instead of collecting only members of v_{d_i}, all half-spaces $h \in \mathcal{H}$ with $h^T \cdot d_i > 0$ are

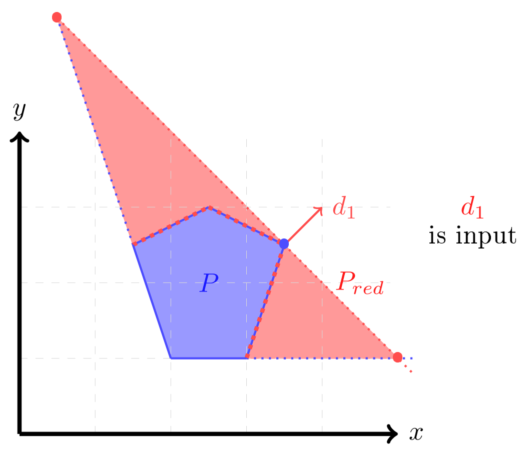


Figure 3.9: A polytope P reduced by *directedHorizon* with direction d_1.

erased from P. They determine the horizon of the direction $d_i \in directions$ (Figure 3.9).

Characteristics of $reduce_{directedHorizon}$:

$$\begin{aligned} drop_{directedHorizon} &\in \{drop_{begin}, \quad \ldots \quad , H-d+1\} \\ time(\ reduce_{directedHorizon}\ ) &= O(H^d) \end{aligned}$$

If the option to return P if $H_{P_{red}} > H$ is enabled the bound $drop_{begin}$ is 0 and otherwise $H - D$ such that it can be negative.

3.1.9 Second Modification of *Directed*

The strategy *directedTemplate* works as the strategy *directedLocal* (see Section 3.1.7). But instead of receiving a set of directions as input this strategy needs only an assignment for a constant t.
The strategy uses an automatic generation of directions: I compute t vectors starting with the vector v by rotating t times with an angle of $360°/t$. The computed set of vectors describes an uniformly distributed set of directions in two dimensions. These vectors are filled into empty d-dimensional vectors at the positions of two dimensions such that all dimension pairs are covered. For example, if we are in 3-dimensional space and specify $t = 7$ we compute a heptagon for the dimensions (1,2), (1,3) and (2,3). With this vector generation it is possible to create polytopes in each dimension. Therefore I combine every generated direction with the offset $o = 1$ and the result is a bounded *uniform* polytope T.

The structure of this reduction is the same as in Section 3.1.7. The reduction P_{red} is the template polytope T with diverse offsets and P does not require to be modified. The *post*-section is skipped, because P_{red} is per definition bounded (Figure 3.10).

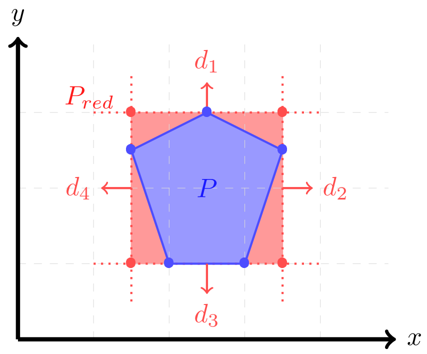


Figure 3.10: A polytope P reduced by *directedTemplate* with $T_{[d=2,t=4]}$.

Characteristics of $reduce_{directedTemplate}$:

$$\begin{aligned} drop_{directedTemplate} &\in \{0, H - D\} \\ time(\ reduce_{directedTemplate}\ ) &= O(H^d) \end{aligned}$$

If the option to return P if $H_{P_{red}} > H$ is enabled $drop_{directedTemplate}$ is 0 and otherwise $H - D$ such that it can be negative.

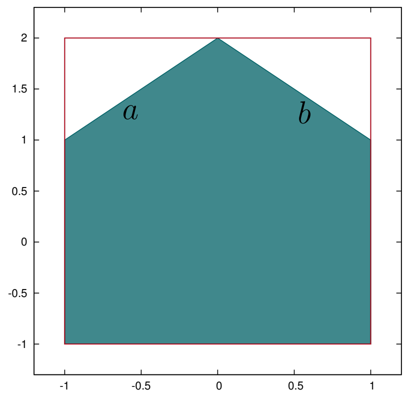
3.1.10 Tests in *Post*-Section

Certain properties of P_{red} have to be checked in the *post*-section of the reduction. The reduction P_{red} can be unbounded. If the test $isBounded(P_{red})$ does not hold P is returned. To test this property I use linear optimization in the direction of removed half-spaces. If the maximization returns unboundedness, P_{red} is unbounded itself. The strategy *directedTemplate* returns a bounded reduced polytope P_{red} due to its uniform distributed directions. If this strategy is used the test $isBounded(P_{red})$ is omitted.
During the development of the strategies I tested if P is contained in P_{red}. If this test does not hold, the implementation of the strategy is broken, due to the definition of P_{red}. Each strategy computes an over-approximation, but numerical accuracy can lead to wrong computations in case floating point arithmetic is used.
If all tests are passed the reduction returns P_{red} and otherwise P.

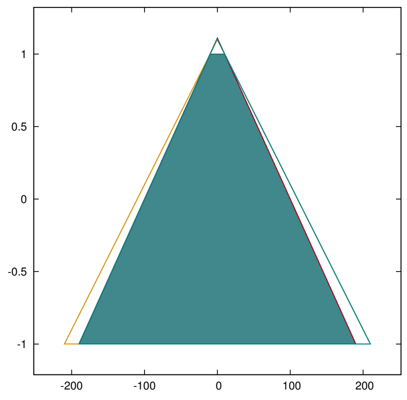
3.2 Comparisons

Every strategy focuses on different details of the input polytope P. But all of them share functionality such that the behavior is alike in specific situations. The *unite*-strategies return the same P_{red} if we reduce P_{unite}. Another example are the strategies *directedLocal* and *directedHorizon*, which compute only one P_{red} if we reduce $P_{directed}$. On the other hand, for every strategy a polytope P exists which results in a unique reduction P_{red}. Furthermore on certain polytopes one strategy s can be the best regarding the criterion $Volume_{[P \to P_{red}]}$.

In this passage I examine relations of the developed strategies. For this purpose I



(a) The *unite*-strategies (red) behave similar.



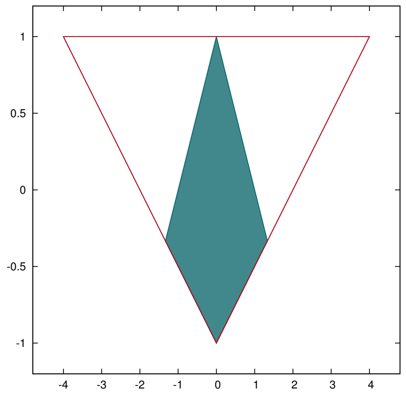
(b) The strategies *uniteWeight* and *dropSmooth* are close to *drop*.

Figure 3.11: Possible reductions of the polytopes P_{unite} (a) and P_{equal} (b).

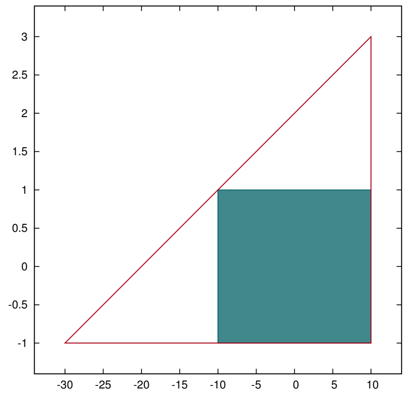
use several representative polytopes to display the similarities and differences. The polytopes are defined in the Appendix A.1. First I present the similarities between the strategies.

- (P_{unite} for the *unite*-strategies):
  The four strategies differ if the two half-spaces a and b have different sizes in $d = 2$ or if they have different shapes for $d > 2$. Moreover if the neighbors of a and b are not *symmetric* the *uniteSmooth*-strategy returns different reductions than the other *unite*-strategies. Symmetric for two half-spaces a and b of a polytope P means that for every neighbor n_a of a with the scalar product SP_{a,n_a} a neighbor n_b exists with $SP_{b,n_b} = SP_{a,n_a}$.
  The behavior is equivalent because the information taken by the extended *unite*-strategies results in the same addition: The strategy *unite* adds a and b. The first modification *uniteSmooth* adds a_{smo} and b_{smo}. The scalar products $SP_{a,a_{smo}}$ and $SP_{b,b_{smo}}$ are the same and therefore the modification has no influence. The result of *uniteWeights* is similar because the weights are identical and non-relevant. In 2-dimensional space the *normal*-vector deduced by the strategy *uniteVertices* is explicit and represents the missing facet f_c of an isosceles triangle determined by the facets f_a and f_b. Therefore f_c has the same angle to f_a and to f_b. The addition of a and b represents exactly this facet f_c. Every single explanation is meant for P_{unite} and the reduction as done in Figure **??**. In this case it is better to apply always the fastest strategy *unite*.

- (P_{equal} for the strategies *drop*, *dropSmooth* and *uniteWeight*):
  Computing the average of the half-spaces a and b by the strategy *uniteWeight* requires the volume of their facets. If the proportion of volumes is extreme the reduction is close to the result of *drop* erasing the half-space with the smaller facet. Another similarity can be found between *drop* and *dropSmooth*. If the scalar products SP_{h,h_l} and SP_{h,h_r} are close to 1 the effect of modifying the neighbors is negligible. In this case it is better to use the fastest strategy *drop*



(a) The *directed*-strategies (red) behave similar.



(b) *uniteVertices* (red) is equal to *uniteWeight*.

Figure 3.12: Possible reductions of the polytopes $P_{directed}$ (a) and $P_{uniteExtended}$ (b).

(Figure 3.11b).

- ($P_{directed}$ for the strategies *directedLocal* and *directedHorizon*):
As I described in Section 3.1.8 these two strategies differ only in the fact that *directedHorizon* removes more half-spaces than *directedLocal*. If there are no additional half-spaces which fulfill the criterion, these two strategies are similar (Figure 3.12a).

- ($P_{uniteExtended}$ for the strategies *uniteVertices* and *uniteWeight*):
Even if both strategies focus on different details they return the same results (Figure 3.13, 3.12b). The explanation for this behavior is that the strategies compute the facet f_c of an triangle determined by f_a and f_b. In other words, let us consider a triangle ABC with f_a the facet between the vertices AB, f_b between BC and f_c between CA. The strategy *uniteVertices* computes f_c. The *normal*-vector of f_c can be represented as a 90° rotation of $C - A$. The strategy *uniteWeight* adds the normalized *normal*-vectors n_a and n_b depending on their weights. To stay consistent with the triangle example, the weighted normalized *normal*-vectors n'_a, n'_b of a and b are the 90° rotations of $A - B$ and $B - C$, in the correct directions. The addition of n'_a and n'_b corresponds to the 90° rotation of the addition of $A - B$ and $B - C$ which is the *normal*-vector of the facet f_c determined by *uniteVertices*. Only the number of cross products, which have to be calculated and therefore the runtime, differs. In higher dimensions *uniteVertices* is faster. The strategy *uniteWeight* computes the size of two facets and therefore (worst case) $O(2 \cdot V^d)$ crossproducts for $d > 2$ while *uniteVertices* has to deal with $O(V^d)$ crossproducts. The similarity can only be found in 2-dimensional space because the half-space u of *uniteVertices* is explicit and the volume of facets is computed precisely. In higher dimensions both computations refer to approximative methods.

Next I examine the differences between the strategies. I do not consider the *directed*-strategies because they depend on the given directions. Once the directions are determined the strategies behave similarly to the *unite*-strategies. Moreover they

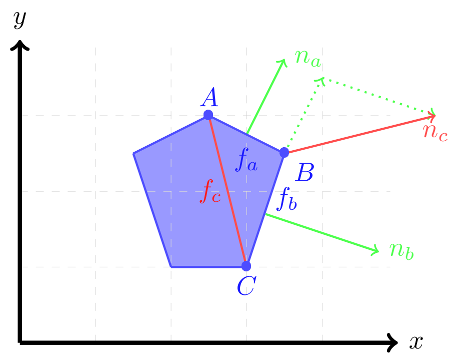


Figure 3.13: The addition n_{a+b} returns f_c rotated by 90° to the left.

can reduce more than one half-space at once. This yields a greater volume increase. I declare a strategy as the best strategy causing the smallest volume increase for an exemplary polytope due to the empirical results of the experiment in Section 4.1.

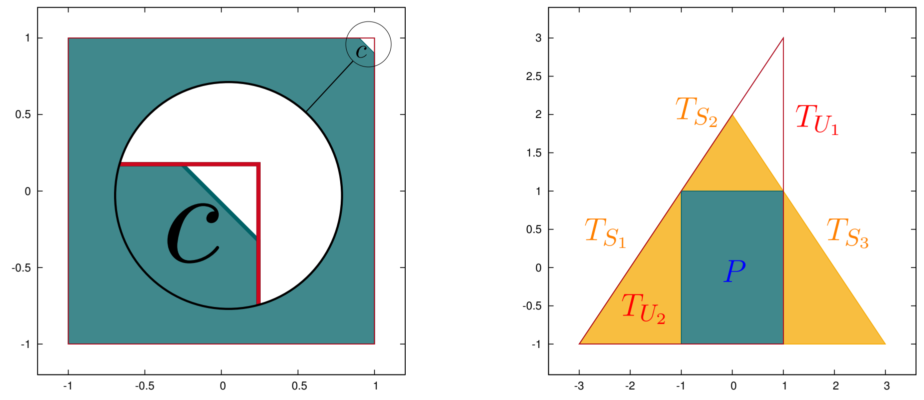
- ($P_{dropBest}$) The polytope $P_{dropBest}$ is a square with a small corner missing. The size of the *corner*-facet f_c of the half-space c is small compared to the other sizes (Figure 3.14a). A drop of c returns the square and the *volume* increase is small compared to the actual size of $P_{dropBest}$.
  Every other strategy computes with more than one half-space and returns a bigger reduction.

- ($P_{uniteBest}$) The reduction $P_{redBest}$ of $P_{uniteBest}$ can be computed by all *unite*-strategies. The polytope $P_{uniteBest}$ is a regular square. The result $P_{redBest}$ is a triangle which consists of $P_{uniteBest}$ in addition to two triangles T_{U_1} and T_{U_2} (Figure 3.14b). These triangles have in total the volume of the square. If we consider the *drop*-strategies we notice, that the right angle between all half-spaces prohibits the use of the strategy *drop*. The *dropSmooth* strategy works but computes always a triangle with three sub triangles T_{D_1}, T_{D_2} and T_{D_3} besides the original square. The two greater sub triangles have in total the volume of the square and the third the volume of the quarter square. Therefore the *unite*-strategies are better than *dropSmooth* for $P_{uniteBest}$.

- ($P_{uniteVerticesBest} = P_{uniteWeightBest}$) As explained before these strategies behave the same. This polytope is a rectangle with a greater side. The strategies *unite* and *uniteSmooth* do not consider the size of the facets and therefore cause a huge volume increase. The two remaining *unite*-strategies compute a polytope $P_{redBest}$ which is a triangle the volume of which is double the volume of the original rectangle as for $P_{uniteBest}$. Additionally, for the same reason the *drop*-strategy can not be used for this polytope and the strategy *dropSmooth* computes a greater polytope too (Figure 3.12b).

3.3 Heuristic for Polytope Reduction

In this section I present a heuristic *Heu* to decide which strategy along with which half-space or half-spaces can be used for a *good* reduction. Good in this case means



(a) The strategy *drop* performs the best reduction.

(b) The *unite*-strategies (red) are better than *dropSmooth* (yellow).

Figure 3.14: Representative polytopes $P_{dropBest}$ (a) and $P_{uniteBest}$ (b).

that the decision leads to P_{red} with a small volume increase. The heuristic Heu choses between the *drop*- and *unite*-strategies, because the *directed*-strategies depend on the directions which are given as input. There exists infinitely many possible directions. If these strategies are excluded, there is only a finite number of possible parameters due to the finite half-spaces which define a polytope. I am interested in a fast and non-perfect procedure in order to reduce computation time at the cost of precision.

$$Heu(P) \to P_{red}$$

My definition of Heu is influenced by *Frehse* [Fre08] and examples P_{sBest} presented in the prior Section. The heuristic Heu has two types of measures: the scalar product $SP_{x,y}$ of two neighbor half-spaces x and y; and $size_f$, the volume of a facet f.
I explain my intuition for the weights:

- The volume increase for the strategy *drop* depends on the size of the facet f_h of the half-space h I want to drop and the scalar product of h with the neighbors. A small $size_{f_h}$ and great scalar products cause a small volume increase.
- The strategy *dropSmooth* is a good choice if the size $size_{f_h}$ is big while the sizes of the neighbor facets are small. Equivalent to the strategy *drop* if the scalar products are close to 1 (i.e. if the *normal*-vectors are similar) the volume increase is smaller.
- If the scalar products are close to 1, especially the scalar product $SP_{a,b}$ the average of the half-spaces a and b causes a good reduction.

Of course this intuition is not perfect. But it represents the basic advantages and disadvantages of the strategies, which depend all on different parts of P. The heuristic Heu has to examine strategies and input parameters in order to deduce a good decision.

Definition 3.2. (Heuristic Heu) In the beginning Heu computes $\mathcal{V}$, *membersOfVertices* and *membersOfHalfspaces*. Next the size of each facet (complexity: $O(H \cdot V^d)$)

and the scalar product (of normalized *normal*-vectors) of all neighbor-pairs (complexity: $O(H^2)$) are determined and saved in *sizesOfFacets* and *scalarproductsOfHalfspaces*. The sizes are computed as the weights of the strategy *uniteWeights* in Section 3.1.6. The initialization of sizes dominates the time complexity. The scalar products $SP_{x,y}$ are added by one: $SP'_{x,y} = SP_{x,y} + 1$. This modification ensures that the weights are positive.
After this initiation-process H computes the best decision for the strategies: *drop*, *dropSmooth*, *unite* and *uniteVertices*. These are the formulas for the weights:

$$\begin{aligned} weight_{drop_h} &= \frac{scalarproducts}{size_{f_h}}, \\ score_{dropSmooth_h} &= \frac{size_{f_h} \cdot scalarproducts}{sizes} \end{aligned}$$

with $scalarproducts = \prod_{xy} SP_{x,y}$ for all half-spaces x,y where $x = h$ or $y = h$, $sizes = \prod_x sizesOfFacets(x)$ for all facets of neighbors x and f_h the facet of the half-space we want to drop.

$$weight_{unite_{ab}} = 2 \cdot SP_{a,b} + scalarproducts$$

with $scalarproducts = \prod_{xy} SP_{xy}$ for all half-spaces x,y where only one of them is the half-space a or b.

After computing the $weight_{unite_{ab}}$ the heuristic Heu compares the sizes $size_{f_a}$ and $size_{f_b}$. If the smaller size $size_{min}$ is less than $x \cdot size_{max}$ I chose as result for the *unite*-strategies *uniteVertices*, otherwise the strategy *unite*. In contrast to *unite* the strategies *uniteVertices* and *uniteWeight* consider the size of facets. Moreover, *uniteVertices* is faster than *uniteWeight*.
Besides the weights the indices of the most promising half-spaces are remembered. All weights are compared and the strategy with the highest weight is taken as the strategy to reduce P with. Finally the reduction P_{red} is the result of Heu.

The heuristic Heu can be modified in multiple ways. The weights and its components can be combined with offsets and constants. For example the constant x used to decide which *unite*-strategy is not definite. During my experiment I define $x = 0.8$. If x is close to 1 Heu chooses *uniteVertices* in most cases which can result in a slower but more precise reduction (if the *unite*-strategies are chosen).

Chapter 4

Experimental Results

»Equipped with her new ideas Alice started the planning of her house. Right as she was about to start she paused again. When should the house be planned on paper? Maybe after she finished constructing the house of solid brick. Then she could do a precise plan and take the plans with her. But this would need even more time as the last time. On the other hand the responsible person would not be upset again. Or an even better solution: I construct a skeleton of the house and from this point on I develop the house on paper. But in this case I have to be as general as possible. A theory-house made on paper could be stable while the real construction would collapse!«

In this chapter I examine how reduction of polytopes influences flowpipe-based reachability analysis. Does reduction of segments provides efficient modifications for reachability analysis algorithm? In the beginning, I reason about the size calculation of a $\mathcal{H}$-polytope (Definition 3.1). Then I apply reduction using the previously developed heuristic Heu on representative polytopes examining $Time_{red}$ and $Volume_{P \Rightarrow P_{red}}$. The second experiment concerns the storage of the set of reachable states R from a flowpipe-based reachability analysis of hybrid systems. I use the model of a bouncing ball as defined in Section 2.1. Next, I apply reduction to the reachability analysis itself in order to ameliorate its memory consumption. Instead of saving each segment I save a reduction of a set of segments. The last experiment focuses on reducing the execution time of reachability analysis. As a more detailed object (e.g. a polytope with many half-spaces) usually causes longer execution times for operations, I examine reduction of objects regarding on the overall execution time for reachability analysis.

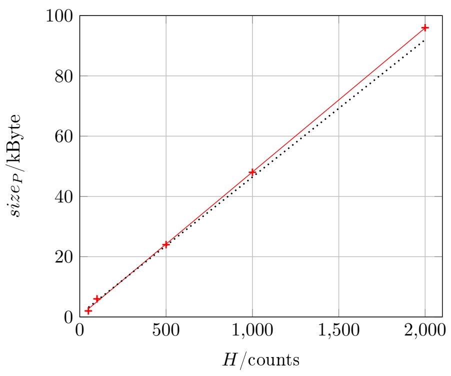
(Verification of Definition 3.1): To confirm my definition for the size of a $\mathcal{H}$-polytope I create a template-polytope T depending on the parameters t and d as in Section 3.1.9. I make use of the *heap profiler* called MASSIF from the software VALGRIND to analyze the actual memory consumption of the systems heap [Dev16]. The tool distinguishes between parts of the executable. Thus I can deduce the actual size of the $\mathcal{H}$-polytope. It is possible to run the program with different combinations of t and d to compare the data (see Table 4.1a) with the developed function.

From our tests we can deduce a linear development if we consider the size of polytopes as a function of the number of half-spaces Heu for every dimension $d \in \{2,3,4\}$

Table 4.1: Sizes of template-polytopes

t \ d	2	3	4
50	2	12	24
100	6	24	48
500	24	96	192
1000	48	192	384
2000	96	384	768

(a) The sizes (in kByte) of template-polytopes depend on t and d.



(b) Sizes of 2-dimensional polytopes dependent of H (MASSIF is solid, function (see Definition 3.1) is dotted).

Figure 4.1: Sizes of polytopes determined by MASSIF (a) and plotted (b).

(Figures 4.1b, 4.2a and 4.2b). The dotted line represents the function defined in 3.1 and is close to the values determined by MASSIF (values and linear regression as solid line). The developing of a function for the size of a H-polytope is important for the next experiments. The use of MASSIF has a huge drawback. The execution time of a program increases and an analysis of the actual time of the algorithm is impossible. A possible analysis is a sequential execution. First we execute the program itself to measure the time and then apply MASSIF to determine the size of the polytope.

4.1 Reduction for Polytopes

In this experiment I apply each strategy presented in Section 3.1 except of the *directed*-strategies and the heuristic Heu on representative polytopes in the dimension 2 to 5. While the heuristic Heu returns a final P_{red} the single strategies have to be applied on every half-space h or every neighbor-pair of half-spaces (a,b). Therefore I loop over all half-spaces and all pairs of half-spaces. Instead of returning all result polytopes I display the best reduction for each strategy corresponding to the criterion $Volume_{P \Rightarrow P_{red}}$.
The volume of a polytope $volume(P)$ is calculated by a sampling based approximation. Depending on the boundaries $d_{i_{min}}$ and $d_{i_{max}}$ of each dimension d_i with $i \in \{1, \dots, d\}$ I create a grid of $d-$dimensional points. Each point represents a $d-$dimensional cube of a specific volume:

$$volume_{Cube} = \prod_{i=1}^{d} \frac{(d_{i_{max}} - d_{i_{min}})}{r}.$$

A high resolution r leads to a more precise grid but increases the execution time of the approximation. If a point is contained in the polytope the volume of the cube is added to the volume of the polytope P.
In the beginning I compute the absolute size $volume(P)$, such that I can calculate the relative volume increase for every reduction P_{red}.

I show the results of one polytope in each dimension (Table 4.1). The remaining results are added in the Appendix in Section A.2. The formal definition of each polytope can be found in Appendix in Section A.1.

Table 4.1: Reduction of $P_{uniteBest}$, P_{3dCube}, P_{4dCube} and P_{5dCube}

	$P_{uniteBest}$			P_{3dCube}		
strategy	$V_{[r=400]}/\%$	i	$time$/ms	$V_{[r=100]}/\%$	i	$time$/ms
drop	×	×	**0.07**	×	×	**0.07**
dropSmooth	125	0	0.23	365	1	0.52
unite	**100**	(2,1)	0.23	**110**	(5,1)	0.49
uniteSmooth	**100**	(2,1)	0.22	**110**	(5,1)	0.52
uniteVertices	**100**	(2,1)	0.27	**110**	(5,1)	0.71
uniteWeight	**100**	(2,1)	0.23	**110**	(5,1)	0.75
$Heu_{[x=0.8]}$	**100**	$_{unite}$(1,0)	0.22+0.23	**110**	$_{unite}$(2,0)	1.4+0.51

	P_{4dCube}			P_{5dCube}		
strategy	$V_{[r=25]}/\%$	i	$time$/ms	$V_{[r=12]}/\%$	i	$time$/ms
drop	×	×	**0.07**	×	×	**0.07**
dropSmooth	1100	1	1.7	3700	0	7.5
unite	**130**	(3,1)	1.64	**85**	(2,0)	7.4
uniteSmooth	**130**	(3,1)	1.77	**85**	(2,0)	7.5
uniteVertices	**130**	(3,1)	5.84	**85**	(2,0)	439
uniteWeight	**130**	(3,1)	8.78	**85**	(2,0)	807
$Heu_{[x=0.8]}$	**130**	$_{unite}$(2,0)	33+1.7	**85**	$_{unite}$(2,0)	4133+7

V is the relative volume increase approximated with the resolution r, the column i determines the input parameters and $time$ represents the execution time of the reduction. For H the time is defined as $time_{heuristic} + time_{reduction}$. A "×" means that for all i the reduction is unbounded.

My first observation is that while the dimension d and the number of half-spaces H increase linearly, the execution time augments much faster. Only the execution time of the strategy *drop* rises slowly. The strategy itself has the time complexity $O(1)$ but the test *isBounded*(P_{red}) depends on the polytope P_{red}. The strategies *dropSmooth*, *unite* and *uniteSmooth* have a similar execution time. The strategies *uniteVertices* and *uniteWeight* are always slower. The measured execution times correspond to the time complexities of the strategies. The heuristic Heu has the worst performance. In Figure 4.3 I plot the number of half-spaces in relation to the execution time and the dimensionality of the respective objects. The bottleneck for the calculation of Heu is the computation of sizes. If we remove this part the value $time$ decreases from ≥ 3 h to 69 s for $P_{T_{[d=5,t=6]}}$. The remaining time of 69 s is dominated by the computation of the $\mathcal{V}$-representation of the polytope with 48 half-spaces. The knowledge about the vertices of the polytope P is indispensable for the heuristic and the reduction.
The results of the heuristic Heu are not satisfiable (Heu returns a P_{red} worse than $P_{redBest}$ in 7 of 14 cases). But this was not my aim in the first place as I pointed out in Section 3.3. The heuristic Heu has to be fast which is not satisfied as well

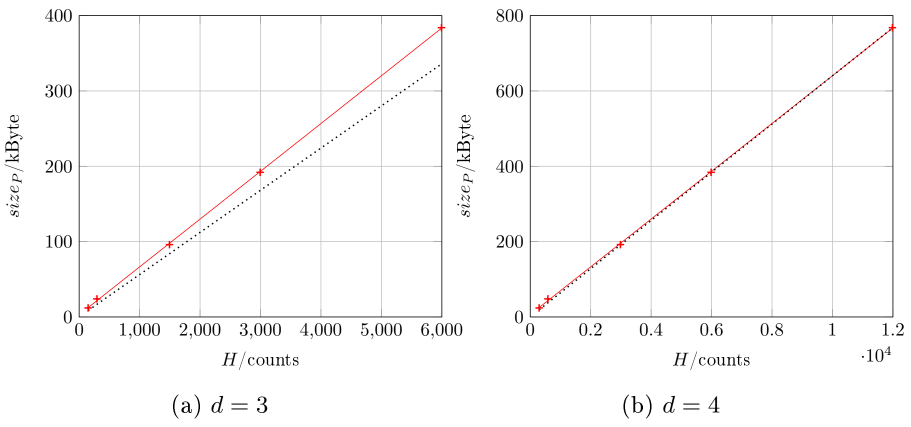


Figure 4.2: Sizes of d-dimensional polytopes dependent of H (MASSIF is solid, function (see Definition 3.1) is dotted).

and therefore it is not usable in general because of its bad execution time in higher dimensions. An example for an examined hybrid system, the ***10 vehicle platoon*** is 31 dimensional and its initial set consists of $2 \cdot 31 = 62$ half-spaces [X. 15], as every variable is bounded by an interval initially and therefore needs two half-spaces. During a reachability analysis the number of half-spaces of the computed segments usually grows. Thus, a fast algorithm to compute or approximate the volume of a facet is needed to create efficient heuristics.

4.2 Efficient Storing of a Set of $\mathcal{H}$-Polytopes

The result of reachability analysis is the set of reachable states R and consists of several flowpipes. If a segment of one flowpipe is contained in another segment the hybrid system has a fixed-point. Each segment is represented as $\mathcal{H}$-polytopes in our setup. The size of a flowpipe is the sum of sizes of all segments. Depending on the timestep, timebound and hybrid system itself the result R can be meticulous and large as the house of solid brick developed by *Alice*. In this experiment I test a composition of algorithms to decrease the size of a result R which has no fixed-point. This idea is similar to the first approach of *Alice*. I use a combination of a convexHull-algorithm and the reduction of polytopes to create a reduced version R_{red} of R. The size of the new result is smaller and its flowpipes still do not intersect in order not to create an additional fixed-point.

I use the hybrid system of a bouncing ball model defined in Section 2.1. The result is $R_{bouncingBall}$. We use double precision for our computation, and the constants $time_{step} = 10$ ms and $time_{bound} = 4$ s. I iterate over the flowpipes of $R_{bouncingBall}$ and create the convex hull of up to c segments (the last convex hull which is computed on a flowpipe is a cluster of $c_{last} \leq c$ segments). Each convex hull is reduced by the strategy $directedTemplate_{[d=2,t]}$ with the variable t. This strategy is one of the strategies which can decrease the number of half-spaces by more than one in a single reduction step. Moreover it depends only on one variable and the reduction is bounded by definition. The c segments are replaced by the reduced convex hull. I

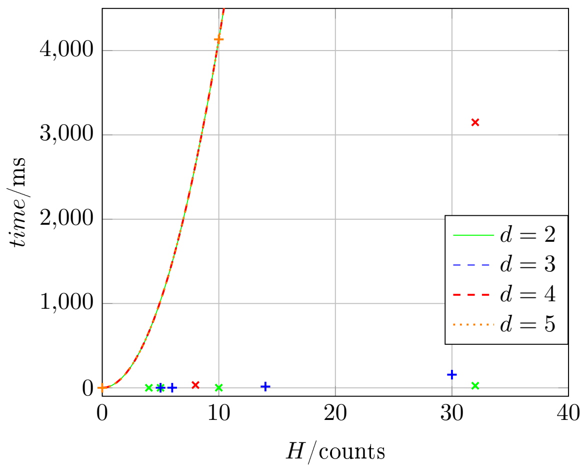


Figure 4.3: Time of heuristic Heu approximated by a polynomial curve in dependency of the number of half-spaces H.

define a condition for R_{red}, which is violated if the flowpipes of R_{red} intersect and the flowpipes of R do not.

(c_{max} and t_{min} for R) If the convex hulls of two or more flowpipes intersect R has an upper bound c_{max} such that for every $c > c_{max}$ the flowpipes of the reduction R_{red} intersect. The upper bound for c_{max} is the highest number of segments of a flowpipe. If c_{max} exists R can have a lower bound for the variable t defined as t_{min} such that for every $t < t_{min}$ the flowpipes of the result R_{red} intersect. These constants restrict the amount of possible assignments.

The obtained reduction R_{red} can violate the condition depending on c and t. Therefore I have to test if the resulting flowpipes of R_{red} intersect. Additionally I test if all segments of $R_{bouncingBall}$ are contained in R_{red}. This can be false due to numerical instability. Moreover R_{red} depends on two decisions. If I remove the redundant planes (1) or not (2) and if I allow the reduction to reduce with a set of directions which is greater than the number of half-spaces of the convex hull. I test both options (1) and (2) and expect option (1) to return a smaller R_{red} and (2) to be faster. Furthermore I allow to reduce a polytope even if the number of new half-spaces is greater than the number of original half-spaces. If we input the convex hull of c clustered segments into the method *removingRedunantPlanes* the computation does not terminate (due to problems with the linear optimizer).
If the tests hold we calculate the relative size decrease $size_{decrease} := size(R_{red})/size(R)$. The reduction R_{red} depends on the variables c for the clustering and t for the reduction by $directedTemplate_{[d=2,t]}$. In the case of $c = 1$ no convex hull is computed at all. For a big c and small t the $size_{decrease}$ might be small, but the probability of an intersection of flowpipes is huge. A greater t prevents R_{red} from intersecting, but the size of R_{red} grows.
First I perform the reachability analysis to obtain $R_{bouncingBall}$ with a size of 480656 Byte. Next I compute R_{red} without the reduction of the convex hull. The main part of the experiment consists of the iteration through multiple assignments of c and t

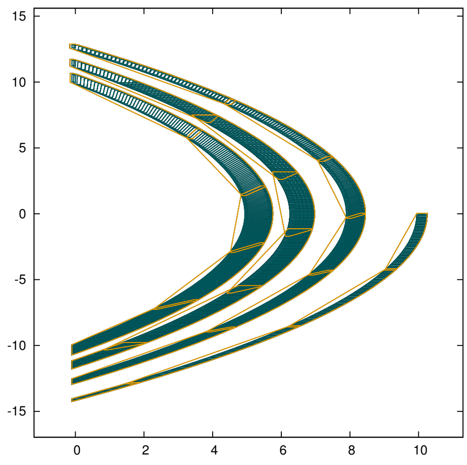


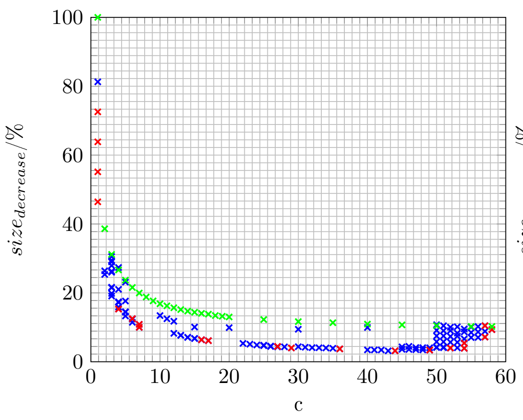
Figure 4.4: The best R_{red} (orange) for $c = 43$ and $t = 50$ with $size_{decrease} = 3.2\%$.

with the options (1) and (2).

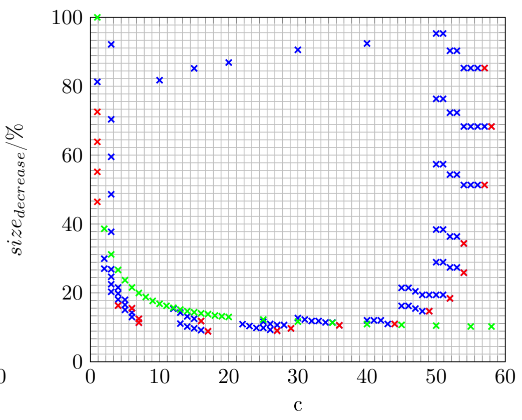
The results are plotted in Figure 4.5a (1) and 4.5b (2) and the single values stand for:

- The green values represent $size_{decrease}$ of the pure convex hulls.
- The blue values stand for successful computations of R_{red} for multiple t.
- The red values belong to a t and show the first value of c which leads to an intersection.

The characteristic constants c_{max} and t_{min} exist and can be set to $c_{max} = 59$ and $t_{min} = 7$. The detailed data of the test is added in the Appendix A.3. The experiment with the setting (1) returns the best value for $c = 43$ and $t = 50$ with $size_{decrease} = 3.2\%$ (Figure 4.4). The best value of setting (2) is $size_{decrease} = 9.2\%$ for $c = 16$ and $t = 15$. The red values of (1) describe an exponential drop with a weak increase in the end, while the development of red values of (2) seems to be subject to quadratic evolving. If we increase c and t while using option (1) the result is below 11% which is a satisfying reduction. Moreover the values (blue and red) of (1) lie below or on the green values. This behavior depends on the removing of redundant planes. Every plane which was unnecessarily added by the reduction is removed and the worst case is that the number of half-spaces left corresponds to the original convex hull. The values (blue and red) created by using option (2) do not show this property. If we increase c and t with option (2) the size of the reduction converges to the size of the original result. For $t = 500$ the size of R_{red} is 85.3% of $size(R_{bouncingBall})$.
The only benefit of option (2) against option (1) is the fact that every iteration of (2) has an execution time less than 4 seconds, while the execution time of some iterations of (1) are as much as 40 seconds. Though, the augmented execution time and $size(R_{red})$ appear only for high c and t, the successful computations of R_{red} for $c \approx 30$ are efficient for both options. Therefore I determine that option (1) is the better algorithm.



(a) Reduction with removing redundant half-spaces.



(b) Reduction without removing redundant half-spaces.

Figure 4.5: The size decrease in dependency of c with only clustering (green), reductions with intersecting flowpipes (red) and without (blue).

4.3 Improvement of Memory Consumption

During a reachability analysis the default algorithm saves every segment as a $\mathcal{H}$-polytope. The single segments are needed to check if a fixed-point is reached. This part of the algorithm can be replaced by storing the reduction of a cluster of several segments. Therefore the fixed-point check has to be executed with the reduction of segments. If the test holds the flowpipe has to be recalculated, otherwise the algorithm proceeds. The recalculation implies a drawback concerning the execution time. However in this experiment I concentrate on the bouncing ball model with no fixed-point. Results of the last experiment in Section 4.2 show, that R_{red} can be reduced to $\approx 5\%$ of the size of $R_{bouncingBall}$. In this experiment the flowpipes are modified with the same procedure as in the last experiment, but with reduction carried out during their computation. I expect similar results concerning $size_{decrease}$. The important aspect of this experiment is the additional time which is required for the convexHull-algorithm and the reduction of polytopes. In the end every user has to decide which values of $time_{increase}$ are bad. I say that a $time_{increase}$ of 200% for a $size_{decrease}$ of 5% is a fair exchange.

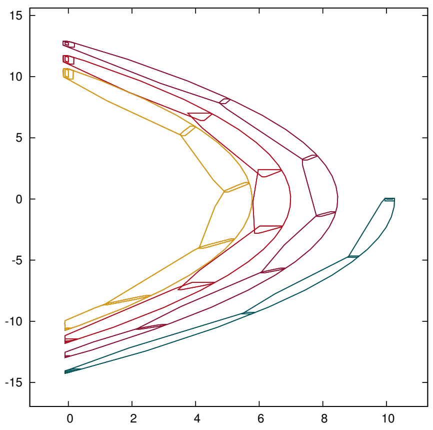
Initially I perform the forward flowpipe-based reachability analysis to determine $size(R_{bouncingBall}) = 480656$ Byte and $time = 2619$ ms. The value for $time$ is the average of 10 executions. Then I apply reduction with the same parameters as in Experiment 4.2 to replace the computed segments by their reduced convex hull. Instead of removing all original segments by their reduction, I remember the segments which are required to compute the *discrete*-transition. If we perform this step with the reduced segments we lose the precision of the non-modified reachability analysis. The result R_{red} has to be verified and if the flowpipes do not intersect the new size and the additional execution time can be analyzed. I perform the modified reachability analysis with multiple assignments of c and t which are influenced by the last experiment (Section 4.2). I am interested in the pairs (t,c_{Good}) and (t,c_{Bad}) with

$c_{Good} < c_{Bad}$. The first pair computes the last R_{red} with no intersection while the second pair returns a R_{red} with intersection depending on a fixed t. Moreover, I execute the experiment again having removed redundant planes (1) and keeping them (2). I expect an time advantage in the modified reachability analysis without removing the redundant planes compared to the analysis which removes them.
The detailed results are put in the Appendix A.4. The best values for $size_{decrease}$ are 11.2% for $t = 25$ and $c = 25$ (2) and 5% for $t = 75$ and $c = 47$ (1). As expected the best computation with option (1) is slower than the best of option (2): 257% > 183% (6.7 s> 4.8 s). But there exist several R_{red} from option (1) which have a better $size_{decrease}$ and a similar $time_{increase}$. For $t = 20$ and $c = 15$ we obtain $time_{increase} = 183\%$ with $size_{decrease} = 8.6\%$ which is less than the best size decrease of option (2). The fastest computation (2) returns a result with $size_{decrease} = 14.2\%$ for $t = 15$, $c = 12$ and $time_{increase} = 167\%$ (4.4 s). For high c and t option (1) has a $time_{increase}$ of up to 1684% (44 s) while the highest value of option (2) is 506% (13 s). Both modifications cause an increasing of execution which renders the method unusable time increase for $t = 500$ and $c = 57$. But the option (2) can be the more reliable choice even if $size_{decrease}$ is not even noticeable at some point. If the flowpipes computed by reachability analysis can be analyzed in advance somehow (for example a reduction dependent on the size of the input polytopes) option (1) can be used with reasonable assignments for c and t.

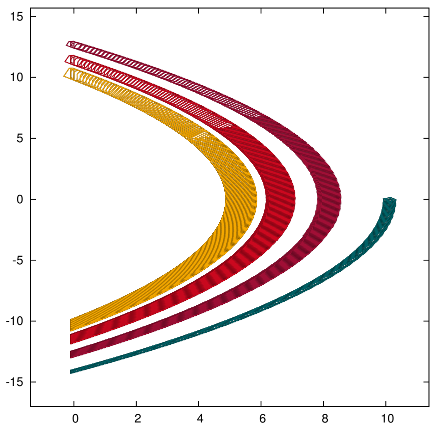
During the experiment I noticed one problem of numerical instability. Depending on the uniformly generated directions computed by the algorithm described in Section 3.1.9 the computations inside the reachability analysis can be precise, lack precision or do not terminate. Therefore I use two different start vectors for the generation of directions: $v = (1{,}0)^T$ and $v^* = (1{,}1)^T$ which can prevent this behavior. In general the generation is done with v but for some t it did not work. The computation did not terminate or the output was distorted. For these situations I chose v^* marked as $t = i^*$ in the tables of this experiment. The empty entries of the tables are caused by the failure of any performed computation. The use of v and v^* should have no influence.

4.4 Improvement of Execution Time

Every segment has to be compared with guards and invariants whose execution time highly depends on the number of half-spaces. A reduction of half-spaces of a segment decreases the computation time of all operations, if the reduction of polytopes itself costs less time. A reduction of a polytope by the strategy *directedTemplate* determines an upper guard for the number of half-spaces H of the reduced polytope. As start vector I choose $v = (1{,}0)^T$ or $v^* = (1{,}1)^T$ as in the last Experiment 4.3.
I manipulate the segments computed by reachability analysis during runtime in order to influence the execution time. After the initialization of the first segment of a flowpipe I perform a reduction with the strategy $directedTemplate_{[d=2,t]}$ depending on the variable t of segments. This idea corresponds to the final version of *Alice's* planning: First the complex object is simplified and then the development proceeds with this simplified version. The modified first segment has half as much half-spaces as before and if the following segments computed by linear transformations stay inside the invariant the new segments keep the initial number of half-spaces. Of course working



(a) Decrease the size to 5% with $c = 47$ and $t = 75$.



(b) Decrease the execution time to 62% with $t = fS/2$.

Figure 4.6: Flowpipes of modified reachability analyses.

with an over-approximation enables the possibility of intersection of flowpipes which do not intersect beforehand.

The setup characteristics for $size_{bouncingBall} = 480656$ Byte and $time = 2619$ ms are the same as in the last experiment. Dependent of this information I determine $size_{Decrease}$ and $time_{Decrease}$. For this experiment I iterate over the natural number $t \in [3{,}20]$. Moreover the experiment is also performed for some t with a different generation of directions. After the result of the last experiment I decided to analyze the influence of the input directions. Therefore I use the vector v and v^* as start vectors for the generation of directions. The results of $size_{Decrease}$ and $time_{Decrease}$ achieved by the use of the vector v^* are marked by brackets (x) beside the other results. I performed the reduction with v^* after the reduction with v and only for the t, which I expected to lead to a time decrease. If a reduction leads to an intersection the result is marked with a "−" beside the value.

The results are added to the Appendix A.5 and show that the improvement of execution time is possible for the bouncing ball model. The best value for $time_{Decrease}$ is 67%. In larger hybrid systems the computation time of reduction by the strategy *directedTemplate* might be even smaller compared to the actual time used by reachability analysis without reduction. Therefore I assume that $time_{Decrease}$ can be decreased further. I mention this assumption because in this example of modifying the reachability analysis the amount of possible t is small. Only 7 of 16 assignments for t lead to a $time_{Decrease} < 100\%$ (Table A.11).
Next I analyze the modified reachability analysis in detail. The average computation time of five executions for the single flowpipes of the original reachability analysis and the modification is presented in Table 4.2.

Table 4.2: Second result of experiment *4.4*

Flowpipe	$time_{orig}$/ms	$time_{t=5}$/ms	$time_{t=6}$/ms	$time_{t=9}$/ms	$time_{t=fS/2}$/ms
1	239	(241_5)	(247_4)	(442_2)	$\mathbf{221}_5$ (240_5)
2	730	(348_7)	(305_6)	(680_2)	545_6 ($\mathbf{302}_6$)
3	440	(431_7)	(437_5)	(810_2)	368_6 ($\mathbf{275}_6$)
4	511	($\mathbf{297}_7$)	(385_6)	(608_3)	306_6 (315_6)
5	448	(270_7)	($\mathbf{227}_5$)	(588_2)	326_6 (270_5)

The execution time presented as ms for every flowpipe. The number next to each *time*-entry is the number of dropped half-spaces *drop*.

This result shows that a small *drop* has barely any positive or even a negative effect on the reachability analysis result, because the calculation requires more time than the offered time advantage. Therefore I added the modification that t is computed dynamically because a fixed t can be a good choice for one example but bad for another. Imagine t is 19 and the average fS for the first example is 20 and for a second example 100. The value *drop* of the first example is $20 - 19 = 1$, while for the second example the number of left half-spaces is 81.
I propose the modification: $t = \lceil fS/2 \rceil$ if $\lceil fS/2 \rceil > dimension$ otherwise I do not perform any reduction. The new version of the reachability analysis returns the best result for the bouncing ball model. For the direction generation with v^* the time improvement $time_{Decrease}$ is 62%. Moreover every reduction causes a size decrease too. In the worst case $size_{Decrease}$ is approx 100%. This happens if t is too big. After removing redundant half-spaces the number of new half-spaces correspond to the number of original half-spaces. If t is too small the over-approximation is not precise anymore and R_{red} violates the condition as for $t = 5$ for this example.

One serious problem is the numerical instability of this method. Some modified reachability analysis setups compute distorted polytopes, thus an intersection of flowpipes and a wrong result. Furthermore the computation can stop completely. The reason has to be the different versions of reductions influenced by the generation of directions. For a quantitative experiment the method is sufficient but such an insecurity requires additional investigation on parameters.

Chapter 5

Conclusion

»Finally Alice's idea worked. The responsible person was not angry anymore and the construction of the house was ok. Time passed and Alice became a promising architect. One day she went back to Engoloc from a site in Nehcaa and got a phone call. A house she built had collapsed. Alice figured out that her plans had been too general. She has to develop a better specification of walls, roofs and floors such that no future house will collapse.«

The general approach of reduction for polytopes works but depends strongly on the parameters. The simple and extended strategies of S are not easily embeddable into greater algorithms. However, the usage of *directedTemplate* where the unknown parameters are mostly known in advance is effective and yields an efficient memory consumption of results of reachability analysis. Moreover the size of the result can be influenced during runtime and I showed the amelioration of the execution time of the an implementation of reachability analysis of hybrid systems on a single example. In order to make a general statement further investigation is required.

5.1 Summary

First I presented the background for this work and the developed strategies along with a heuristic on how to use them. The evaluation of these strategies showed following results: Most strategies are not usable in context of algorithms such as reachability analysis. For any strategy with unknown parameters these have to be set beforehand. This requires an examination of the polytope regarding the volume of facets and the scalar products between all pairs of half-spaces in my case. The execution time of the heuristic was excessive for high dimensions and a high number of half-spaces. Therefore I focused on the use of the strategy $directedTemplate_{[d,t]}$ which depends only on one input parameter t, because the assignment of d is explicit. In my experiments, in order to influence the result of the reachability analysis or the reachability analysis itself, I used a combination of a convexHull-algorithm and the strategy *directedTemplate*. The required parameters are the number of clustered segments and the input parameter for the applied reduction strategy. Extended evaluation of different parameter configuration showed the general applicability of this approach. It is possible to store a result of the reachability analysis of the bouncing ball model occupying only 5% of its original size while preserving the original characteristics. Furthermore the size decrease can be achieved also by manipulating the reachability analysis directly. I replace each single segment by the reduced convex hull of a cluster of segments. During the later evaluation I focused on decreasing the execution time of the algorithm. A dynamic reduction showed the best result by the strategy *directedTemplate* which offered a time decrease of 62% for the bouncing ball model.

5.2 Discussion

It is possible to improve the reachability analysis by reducing the segments of flowpipes. However the experiments demonstrate at the same time, that the best use of these modifications depends on the careful choice of unknown parameters. These decisions influence the result of the analysis. The reachability analysis can require a long execution time. Or the size decrease of the segments of flowpipes leads to a size increase instead. But due to numerical instability while using floating point arithmetic the analysis can return wrong results, or even not terminate such that the execution has to be aborted. This behavior makes a general use of this modification in a reachability analysis of hybrid systems impossible.
Moreover I detected, that the bottleneck of the heuristic *Heu* is the calculation of volumes of facets. In order to address this, a faster and therefore less precise algorithm can be used to calculate or approximate the volume of facets or the heuristic has to rely on other information. Nevertheless I assume that the volume of facets is a major condition for heuristics which consider the volume increase of polytopes.

5.3 Future work

As discussed in the prior passage the crucial part which requires further investigation is the numerical stability and the assignment of parameters of the reachability analysis modified by the reduction of polytopes. Only if this problem is solved can a faster analysis, or a less memory consuming analysis, be developed at all. I present different promising approaches:

1. A fast heuristic which focuses on a small volume increase of polytopes can be used as an alternative to the *directedTemplate* strategy. Moreover such an algorithm can be applied in other areas where complex polytopes are generated and need to be simplified.

2. The resulting flowpipes of flowpipe-based reachability analysis can be reduced by an algorithm which I showed in Section 4.2 and 4.3. However the reduction depends on parameters which have an infinite number of assignments. A generalization of the algorithm requires an automated generation of parameters. Moreover, a consequent reduction with *directedTemplate* enables the renouncement of *normal*-vectors of each segment due to the finite number of possible directions of half-spaces.

3. The reachability analysis can be speed up by the reduction of the first segment of each flowpipe. The number of half-spaces of the modified first segment has to be defined carefully. The removing of too many half-spaces might cause the safety verification to declare a system as insecure. On the other hand if we leave too many half-spaces the time advantage disappears due to reduction time.

Bibliography

[AB95] D. Avis and D. Bremner. How good are convex hull algorithms? In *Proceedings of the Eleventh Annual Symposium on Computational Geometry*, SCG '95, pages 20–28. ACM, 1995.

[CB96] H. Huhdanpaa C. Barber, D. Dobkin. The Quickhull Algorithm for Convex Hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996. ACM.

[D. 92] D. Avis, K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete & Computaitional Geometry*, 8(3):295–313, 1992. Springer New York.

[Dev16] Valgrind^{TM} Developers. *9. Massif: a heap profiler*, 2016. `http://valgrind.org/docs/manual/ms-manual.html`, last checked: 27.1.2016.

[E. 15] E. Ábrahám, X. Chen. *Lectures on Modeling and Analysis of Hybrid Systems*, 2015. `https://ths.rwth-aachen.de/wp-content/uploads/sites/4/teaching/vorlesung_hybride_systeme/handout.pdf`, last checked: 27.1.2016.

[E. 16a] E. Ábrahám, X. Chen, S. Kowalewski, I. B. Makhlouf, S. Schupp, S. Sankaranarayanan. *A Toolbox for the Reachability Analysis of Hybrid Systems using Geometric Approximations (HyPro)*, 2016. `https://ths.rwth-aachen.de/research/projects/hypro/`, last checked: 27.1.2016.

[E. 16b] E. Ábrahám, X. Chen, S. Kowalewski, I. B. Makhlouf, S. Schupp, S. Sankaranarayanan. *Benchmarks of continuous and hybrid systems*, 2016. `https://ths.rwth-aachen.de/research/projects/hypro/benchmarks-of-continuous-and-hybrid-systems/`, last checked: 27.1.2016.

[Fre08] Goran Frehse. PHAVer: Algorithmic Verification of Hybrid Systems past HyTech. *International Journal on Software Tools for Technology Transfer*, 10(3):263–279, 2008. Springer-Verlag.

[Gue09] Colas Le Guernic. *Reachability Analysis of Hybrid Systems with Linear Continuous Dynamics*. PhD thesis, Université de Grenoble, 2009.

[HKPV95] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing*, STOC '95, pages 373–382. ACM, 1995.

[X. 15] X. Chen, S. Schupp, I.B. Makhlouf, E. Ábrahám, G. Frehse, S. Kowalwski. A benchmark suite for hybrid systems reachability analysis. *Lecture Notes in Computer Science*, 9058:408–414, 2015. Springer International Publishing.

[Zie95] G. M. Ziegler. *Lectures on Polytopes.* Springer New York, 1995.

Appendix A

Polytopes and Results of Experiments

In this part of my work I present the complete results of my experiments and additional information about the $\mathcal{H}$-polytopes which I used.

A.1 $\mathcal{H}$-Polytopes

In this section I define every polytope used in this work, using the matrix representation (see Definition 2.3).

$$\begin{aligned}
P_{unite} &:= \left\{ x \in \mathbb{R}^d \,\middle|\, \begin{pmatrix} -1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 1 & 0 \\ 0 & -1 \end{pmatrix} \cdot x \leq \begin{pmatrix} 1 \\ 2 \\ 2 \\ 1 \\ 1 \end{pmatrix} \right\} \\
P_{uniteExtended} &:= \left\{ x \in \mathbb{R}^d \,\middle|\, \begin{pmatrix} -1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & -1 \end{pmatrix} \cdot x \leq \begin{pmatrix} 10 \\ 1 \\ 10 \\ 1 \end{pmatrix} \right\} \\
P_{dropBest} &:= \left\{ x \in \mathbb{R}^d \,\middle|\, \begin{pmatrix} -1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \\ 0 & -1 \end{pmatrix} \cdot x \leq \begin{pmatrix} 1 \\ 1 \\ 1.9 \\ 1 \\ 1 \end{pmatrix} \right\} \\
P_{3d} &:= \left\{ x \in \mathbb{R}^d \,\middle|\, \begin{pmatrix} 0 & 0 & -1 \\ 1 & 0 & 0.1 \\ -1 & -1 & 0.1 \\ 0 & 1 & 0.1 \\ 0 & 0 & 1 \end{pmatrix} \cdot x \leq \begin{pmatrix} 0 \\ 3 \\ 3 \\ 3 \\ 2 \end{pmatrix} \right\}
\end{aligned}$$

P_{dCube}	:=	a polytope representing a unit cube in the dimensions 2,3,4 and 5. The polytope P_{2Cube} is called $P_{uniteBest}$
$P_{T_{[d,t]}}$	:=	a polytope created by the algorithm described in Section 3.1.9 with input parameters d, t and $v = (1,0)^T$

A.2 Results of Experiment *4.1*

These are the additional results of the experiment in Section 4.1. I reduced the resulting flowpipes of a forward flowpipe-based reachability analysis in order of size decrease.

Table A.1: Reduction of P_{unite}, $P_{uniteExtended}$, $P_{dropBest}$ and $P_{T_{[d=2,t=10]}}$

	P_{unite}			$P_{uniteExtended}$		
strategy	$V/\%$	i	*time*/ms	$V/\%$	i	*time*/ms
drop	20	1	0.07	×	×	**0.07**
dropSmooth	35	1	0.29	262	1	0.22
unite	20	(2,1)	0.28	507	(1,0)	0.20
uniteSmooth	20	(2,1)	0.29	507	(1,0)	0.23
uniteVertices	20	(2,1)	0.35	**100**	(1,0)	0.27
uniteWeight	20	(2,1)	0.31	**100**	(1,0)	0.23
Heu	20	$_{unite}$(2,1)	0.33+0.31	262	$_{dropSmooth}$ 1	0.22+0.24
	$P_{dropBest}$			$P_{T_{[d=2,t=10]}}$		
drop	**0.2**	2	**0.07**	**2.5**	0	**0.1**
dropSmooth	46	2	0.28	4	0	0.99
unite	23	(2,1)	0.28	3	(1,0)	1.1
uniteSmooth	17	(2,1)	0.29	3	(1,0)	1.0
uniteVertices	2.5	(2,1)	0.35	3	(1,0)	1.25
uniteWeight	2.5	(2,1)	0.30	3	(1,0)	1.39
Heu	**0.2**	$_{drop}$ 2	0.33+0.11	3	$_{unite}$(9,0)	1.5+0.94

V is the relative volume increase approximated with the resolution $r = 400$, the column i determines the input parameters and *time* represents the execution time of the reduction. For H the time is defined as $time_{heuristic} + time_{reduction}$. A "×" means that for all i the reduction is unbounded.

Table A.2: Reduction of $P_{T_{[d=2,t=32]}}$, P_{3d}, $P_{T_{[d=3,t=5]}}$, $P_{T_{[d=3,t=12]}}$, $P_{T_{[d=4,t=8]}}$ and $P_{T_{[d=5,t=3]}}$

	$P_{T_{[d=2,t=32]}}$			P_{3d}		
strategy	$V/\%$	i	*time*/ms	$V/\%$	i	*time*/ms
drop	**0.2**	0	**0.14**	455	4	**0.07**
dropSmooth	**0.2**	0	8.7	130	4	0.37
unite	**0.2**	(1,0)	8.7	130	(4,2)	0.35
uniteSmooth	**0.2**	(1,0)	8.8	**120**	(4,2)	0.38
uniteVertices	**0.2**	(1,0)	9.2	165	(4,2)	0.47
uniteWeight	**0.2**	(1,0)	8.8	150	(4,3)	0.54
Heu	**0.2**	$_{drop}$ 31	25+0.21	130	$_{unite}$(4,3)	0.86+0.37
	$P_{T_{[d=3,t=5]}}$			$P_{T_{[d=3,t=12]}}$		
drop	**2**	0	**0.09**	0.5	0	**0.15**
dropSmooth	17	0	8.6	3	0	101
unite	5	(1,0)	9.3	**1**	(1,0)	101
uniteSmooth	5	(1,0)	8.6	**1**	(1,0)	102
uniteVertices	5	(1,0)	9.0	**1**	(1,0)	105
uniteWeight	5	(1,0)	9.6	**1**	(1,0)	112
Heu	14	$_{unite}$ (11,1)	15+8.5	9	$_{dropSmooth}$ 18	156+158
	$P_{T_{[d=4,t=8]}}$			$P_{T_{[d=5,t=3]}}$		
drop	**5**	0	**0.07**	**15**	5	**0.16**
dropSmooth	15	0	1197	190	6	1742
unite	10	(1,0)	1195	**15**	(4,0)	1740
uniteSmooth	**5**	(1,0)	1196	**15**	(4,0)	1737
uniteVertices	**5**	(1,0)	1257	20	(3,0)	55 s
uniteWeight	**5**	(1,0)	1257	20	(3,0)	48 min
Heu	50	$_{dropSmooth}$ 21	3149+1189			

V is the relative volume increase approximated with the resolution $r = 400$ for $P_{T_{[d=2,t=32]}}$, $r = 25$ for $P_{T_{[d=4,t=8]}}$, $r = 12$ for $P_{T_{[d=5,t=3]}}$ and otherwise $r = 100$, the column i determines the input parameters and *time* represents the execution time of the reduction. For H the time is defined as $time_{heuristic} + time_{reduction}$. A "×" means that for all i the reduction is unbounded. The results of $P_{T_{[d=5,t=3]}}$ do not represent the best results due to excessive computations (timebound was 1 h).

Table A.3: Reduction of $P_{T_{[d=5,t=6]}}$

strategy	*drop*	*dropSmooth*	*unite*	*uniteSmooth*	*uniteVertices*	*uniteWeight*	*Heu*
time/ms	**0.57**	78 s	77 s	77 s	189 s	182 s	≥ 3 h

A.3 Results of Experiment *4.2*

These are the computational results of the experiment in Section 4.2. The variable c stands for the number of clustered segments. The variable t represents the input parameter for the reduction with the template polytope $T_{[d=2,t]}$. The entries contain the percental decrease. If a $\times$ is added to the entry that means this combination causes an intersection. Not every combination was examined. Therefore some entries in the table are empty.

Table A.4: Data (1) of experiment *4.2*

c	1	2	3	4	5	6	7	8	9	10
$size_{Decrease}/\%$	100	38.7	31.2	26.7	23.7	21.6	20	18.8	17.7	16.9
c	15	20	25	30	35	40	45	50	55	58
$size_{Decrease}/\%$	14.4	13.1	12.3	11.7	11.4	11	10.8	10.5	**10.3**	**10.3**

Values of $size_{Decrease}/\%$ of the convex hull computation without any reduction.

Table A.5: Data (2.1) of experiment *4.2*

t c	3	4	5	6	7	8	9	10	15	20	25
1	$\times$46.5	$\times$55.2	$\times$63.9	$\times$72.6	81.3						
2					25.4	26.5					
3					19.1	19.9	21.5	21.7	26.3	26	29
4					$\times$15.3	15.9	17.3	17.5	21.1		
5						13.3	14.4	14.6	17.7		
6						11.5	12.4	$\times$12.6			
7						$\times$10	$\times$10.9				
12										8.3	
13									7.7	7.7	
14									7.1	7.1	
15									6.8	6.8	
16									6.4	$\times$6.4	
17									$\times$6.2		
22											5.4
23											5.1
24											4.9
25											4.9
26											4.6
27											$\times$4.5

Values of $size_{Decrease}/\%$ **with** (1) removing redundant planes

Table A.6: Data (2.2) of experiment *4.2*

t c	30	40	50	75	100	150	200	300	400	500
3	28	30	30.6							
4				27.4						
5				23.2						
10					13.5					
15						10.1				
20							10			
25	4.7									
26	4.5									
27	4.4									
28	4.4									
29	$\times$4.1									
30		4.4						9.5		
31		4.3								
32		4.1								
33		4.1								
34		4.1								
35		4.0								
36		$\times$3.8								
40			3.5						10	
41			3.5							
42			3.5							
43			**3.2**							
44			$\times$3.2							
45				3.7	4.4					
46				3.6	4.5					
47				3.6	4.2					
48				3.5	4.1					
49				$\times$3.4	4.1					
50					4.2	4.3	5.9	7.5	9.2	10.8
51					4.1	4.3	5.8	7.5	9.1	10.5
52					$\times$4	4.1	5.7	7.4	9.1	10.1
53						4.1	5.7	7.5	8.3	10.2
54						$\times$4	$\times$5.5	6.7	8.7	9.6
55								7.1	8.5	10.1
56								7.1	8.7	10.3
57								$\times$7.2	8.9	$\times$10.5
58									$\times$9.4	

Values of $size_{Decrease}/\%$ **with** (1) removing redundant planes

Table A.7: Data (3.1) of experiment *4.2*

t / c	3	4	5	6	7	8	9	10	15	20	25
1	×46.5	×55.2	×63.9	×72.6	81.3						
2					27.1	30					
3					20.4	22.5	24.7	26.9	37.8	48.7	59.6
4					×16.4	18.1	19.9	21.6			
5						15.1	16.6	18			
6						13.1	14.3	×15.6			
7						×11.4	×12.5				
12										15.4	
13									11.1	14.3	
14									10.3	13.2	
15									9.7	12.5	
16									**9.2**	×11.9	
17									×8.9		
22											11
23											10.4
24											9.9
25											9.9
26											9.3
27											×9.1

Values of $size_{Decrease}/\%$ **without** (2) removing redundant planes

Table A.8: Data (3.2) of experiment *4.2*

t / c	30	40	50	75	100	150	200	300	400	500
3	70.4	92.2	114							
4				135.2						
5				112.8						
10					81.8					
15						85.2				
20							86.9			
25	11.7									
26	11									
27	10.7									
28	10.7									
29	$\times$9.7									
30		12.7						90.6		
31		12.3								
32		11.9								
33		11.9								
34		11.5								
35		11.5								
36		$\times$10.6								
40			12.1						92.5	
41			12.1							
42			12.1							
43			11							
44			$\times$11							
45				16.3	21.5					
46				16.3	21.5					
47				15.5	20.5					
48				14.7	19.5					
49				$\times$14.7	19.5					
50					19.5	29	38.4	57.4	76.4	95.4
51					19.5	29	38.4	57.4	76.4	95.4
52					$\times$18.4	27.4	36.5	54.4	72.4	90.3
53						27.4	36.4	54.4	72.4	90.3
54						$\times$25.9	$\times$34.4	51.4	68.4	85.3
55								51.4	68.4	85.3
56								51.4	68.4	85.3
57								$\times$51.4	68.4	$\times$85.3
58									$\times$68.4	

Values of $size_{Decrease}/\%$ **without** (2) removing redundant planes

A.4 Results of Experiment *4.3*

These are the computational results of the experiment in Section 4.3. The variable c_{Good} and c_{Bad} stand for the number of clustered segments. The variable t represents the input parameter for the reduction with the template polytope $T_{[d=2,t]}$. If a * is added, the uniform generated directions are computed with the vector $v^* = (1,1)^T$. The entries of $size_{DecGood}$ and $size_{DecBad}$ indicate the relative size decrease. The entries of $time_{IncGood}$ and $time_{IncBad}$ indicate the relative time increase. The *good* side of $t \in \{3,4,5,6\}$ are empty because such an imprecise reduction causes an intersection of flowpipes even without clustering. Moreover, the row with $t = 8$ is missing due to numerical problems.

Table A.9: Data (1) of experiment *4.3*

t	c_{Good}	$size_{DecGood}/\%$	$time_{IncGood}/\%$	c_{Bad}	$size_{DecBad}/\%$	$time_{IncBad}/\%$
3*				1	47.9	140
4*				1	56.4	155
5				1	64.6	166
6*				1	73.5	178
7*	2	40.9	232	3	27.7	204
9*	10	10.2	180	11	9.4	170
10*	10	10.7	196	11	9.8	180
15	12	9.8	**172**	14	8.9	173
20	15	8.6	183	16	8.7	193
25	25	6.1	190	26	6.3	199
30	28	6	198	29	5.6	193
40	34	5.5	210	36	5.3	216
50	36	5.1	228	37	4.9	211
75	47	**5**	257	48	4.8	236
100	50	5.1	250	51	5.2	263
150	53	5.2	308	54	5.1	319
200	55	6.3	377	57	6.6	488
300	55	8	670	56	8.1	657
400	56	9.7	1029	57	9.6	1094
500	56	11.5	1684	57	14.7	1940

Table A.10: Data (2) of experiment *4.3*

t	c_{Good}	$size_{DecGood}/\%$	$time_{IncGood}/\%$	c_{Bad}	$size_{DecBad}/\%$	$time_{IncBad}/\%$
3*				1	47.9	137
4*				1	56.4	153
5				1	64.6	166
6*				1	73.5	173
7*	2	41.8	226	3	28.4	197
9*	8	13.9	188	9	12.7	194
10*	10	12.2	176	11	11.2	175
15	12	14.2	**167**	14	12.5	167
20	15	14.5	177	16	13.9	190
25	25	**11.2**	183	26	11.2	190
30	28	12.1	199	29	11.7	184
40	34	13.3	198	36	12.8	208
50	36	15.5	201	37	14.4	196
75	47	17.6	226	48	16.8	219
100	50	20.7	221	51	20.7	240
150	53	28.7	242	54	28.7	253
200	55	35.7	260	57	35.7	351
300	55	52.7	323	56	52.7	325
400	56	69.6	382	57	69.6	445
500	56	86.7	506	57	86.7	566

A.5 Results of Experiment *4.4*

These are the computational results of the experiment in Section 4.4.

Table A.11: First result of experiment *4.4*

t	$size_{Decrease}/\%$	$time_{Decrease}/\%$
5	66.9^{-} (**61.7**)	68^{-} (68)
6	71 (69.7)	78.7 (67)
7	79.1	95
8	76.7 (73.7)	80 (77)
9	82.5(90)	97 (127)
10	81.1 (99.8)	92 (149)
11	93.2 (87.3)	125 (113)
12	88.2 (95.4)	95 (116)
13	95	132
14	94.8	132
15	106.7	164
16	85.5 (87.9)	101 (116)
17	102.8	157
18	100.6	138
19	108.2	179
20	100.9	137
$fS/2$	68.4 (62.5)	74 (**62**)

Dependent of t the result R has $size_{Decrease}$ and $time_{Decrease}$ as percentages. For $t \in \{3,4\}$ no results are correct. For $t = 5$ with v the resulting flowpipes intersect. The variable fS is the number of half-spaces of the first segment.