TU Kaiserslautern

Computing boundaries of tropical varieties

Master thesis in mathematics

Author
Sebastian Muskalla

Supervisor
Thomas Markwig

April 27, 2015

Contents

Part I.
Introduction

1. Abstract

Tropical geometry is a new field of geometry, which associates algebraic varieties with their tropicalization. The tropicalization of a variety is a degeneration into a semi-linear object, which can be studied using methods from polyhedral theory and combinatorics, but surprisingly preserves some fundamental properties of the variety.

One basic restriction is that we implicitly intersect varieties with the torus before tropicalizing them, i.e. we exclude the boundary where some coordinates are zero. The goal of this thesis is to study how this restriction can be eased. Algorithms to compute the part of the tropicalization in the boundary are presented and applied to the Grassmanian as main example.

The appendix contains implementations of the algorithms for the computer algebra system SINGULAR [Sing].

2. Structure

The initial goal of my research was to study methods how one can compute the boundary of the tropicalization of the Grassmanian and to compare the results of those methods applied to the Grassmanian $G(3, 6)$. Therefore, before introducing tropical varieties, we will start in the second part by defining the Grassmanian. Furthermore, we will study several algorithms to get a finite set of generators for its defining ideal.

In the third part, we introduce tropical varieties, the main objects of tropical geometry. We provide three different definitions: one which gives the best intuitive understanding, one which is best for computational purposes and a purely algebraic variant, which shows why we need the restriction to the torus mentioned in the abstract. In the last section of this part, we state the fundamental theorem, which shows that those three ways are equivalent. We will introduce some concepts from polyhedral theory and cite the structure theorem, which specifies the polyhedral structure of tropical varieties.

The main results of this thesis are contained in the fourth part. We define the boundary via elimination by tropicalizing the part of a variety living inside the hyperplane where some component is zero. On the computational side, this is done by adding a generator to the ideal and intersecting it with a subring, so we will briefly touch on the topic of variable elimination.
A second way to compute the boundary purely relies on projecting certain parts of the polyhedral complex given by the tropical variety. We show that - as in traditional algebraic geometry - there is a correspondence between elimination and projection and conclude that both ways to define the boundary yield the same result if we start with a saturated ideal.

The appendix contains two libraries and other code for the computer algebra system SINGULAR : `grassmanian.lib` consists of algorithms to compute the Pluecker ideal defining the Grassmanian respectively a Groebner basis for it. The two methods to compute the boundaries are implemented in `tropicalboundaries.lib`. Furthermore, procedures to study whether the results of those two methods are equal are provided in `test_tropicalboundaries.c`. Lastly, several examples also mentioned during the thesis are attached.

Part II.
The Grassmanian

3. The Grassmanian

Varieties which parametrize other varieties are objects of interest in algebraic geometry. An easy but very important example is the *Grassmanian* $G(r, m)$. The points of this variety correspond to linear subspaces of dimension r of a vector space K^m. In this chapter, we will study the Grassmanian and algorithms to compute generators of its defining ideal. In later parts of the thesis, the Grassmanian respectively its tropicalization will be used as the main example for the algorithms to compute the boundaries of tropical varieties.

We will always assume that K is some fixed field of characteristic 0 in the following.

3.1 Remark
As usual, we define the affine space $\mathbb{A}_K^n = K^n$ to be the space of points with n components with entries in K (without any vector-space structure). We define the projective space $\mathbb{P}_K^{n-1}$ to be the space we get by identifying scalar multiples in $\mathbb{A}_K^n$:

$$\mathbb{P}_K^{n-1} = \{V \mid V \text{ is a one-dimensional subspace of } K^n\} = \{\langle v \rangle \mid v \in K^n, v \neq 0\}.$$

3.2 Remark
Any r-dimensional subspace of K^m can be represented as the row space of a matrix $A \in K^{r \times m}$ of rank r by writing a set of basis vectors of the subspace into the rows of the matrix. Unfortunately, this representation is not unique: the row spaces of two matrices A and B are equal if and only if A can be obtained from B by applying row operations, i.e.

$$A = G * B$$

for some $G \in \mathrm{Gl}_r(K)$.

To get a unique representation, we map A to vectors containing the minors of size $r \times r$. Any $r \times r$ minor A_I of $A \in K^{r \times m}$ is uniquely determined by the choice of column indices $I \subseteq \{1, ..., n\}$ corresponding to the columns which should be kept in the submatrix. We may write

$$A_I = \det(A * S_I)$$

where $S_I \in K^{r \times m}$ is the matrix selecting the columns of A with indices in I. Its j^{th} column is 1 at some position $i \in I$ and zero otherwise.

By the determinant multiplication theorem,

$$A_I = \det(A * S_I) = \det(G * B * S_I) = \det(G) * \det(B * S_I) = \det(G) * B_I$$

follows. In particular, the minors differ only by a scalar multiple, which is independent from the minor. If we look at the vectors $v(A)$, $v(B)$ of all minors, we have

$$v(A) = \det(G) * v(B)$$

and they define the same point in projective space.

Since any minor of size $r \times r$ of a matrix of size $r \times m$ corresponds to a choice of columns, we will study how to compute such subsets before we will define the Grassmanian formally.

3.3 Definition
Let $r \in \mathbb{N}$ and M be a finite set with $r \leq |M|$. We define the set of subsets of size r:

$$\Lambda(r, M) = \{\alpha \subseteq M, |\alpha| = r\}.$$

By abuse of notation, we write $\Lambda(r, m)$ for $\Lambda\left(r, \{1, ..., m\}\right)$ for $m \in \mathbb{N}, m \geq r$.

3.4 Remark
In the following, we will identify subsets of $\{1, ..., m\}$ of size r with ordered lists, tuples in $\{1, ..., m\}^r$ such that their components are in strictly ascending order. This gives a fixed ordering on the elements of $\alpha \in \Lambda(r, m)$.

Furthermore, we order $\Lambda(r, m)$ itself by looking at the smallest element contained only in one of two sets. For $\alpha, \beta \in \Lambda(r, m)$, we define

$$\alpha < \beta \Leftrightarrow \min\left((\alpha \cup \beta) \backslash (\alpha \cap \beta)\right) \in \alpha.$$

If we represent elements of $\Lambda(r, m)$ as ordered lists as described above, this yields a lexicographic ordering on the tuples.

The following recursive algorithm computes $\Lambda(r, M)$.

3.5 Algorithm

Input:

$r \in \mathbb{N}$,

M finite set with $r \leq |M|$

Output:

$\Lambda(r, M)$

```
1 if r = 0 then
2 |   return {∅}
3 end
4 if M = ∅ then
5 |   return {}
6 end
7 Choose e ∈ M
8 M' := M\{e}
9 return {{e} ∪ α | α ∈ Λ(r − 1, M')} ∪ Λ(r, M')
```

Proof of termination:

The size of M decreases with each recursive call. Since $|M|$ is always a natural number and the recursion base catches $|M| = 0$, this process has to stop. □

Proof of soundness:

We prove soundness using induction over the cardinality of M. $M = \emptyset$ has only itself as subset of size 0 and no subsets of any other size. This coincides with the value returned by the algorithm. For any set M of size larger than 0, we may fix an arbitrary element $e \in M$. Any subset α' of size r is either a subset not containing e, and thus an element of $\Lambda\left(r, M\backslash\{e\}\right)$, or it is a subset containing e. In the latter case, we may write it as $\alpha' = \{e\} \cup \alpha$ where $\alpha \in \Lambda\left(r-1, M\backslash\{e\}\right)$. This coincides with the value returned by the algorithm, if we assume that the recursive calls return the correct result using induction. □

3.6 Remark

If we implement the algorithm above using ordered lists to represent sets, we can get the ordering on $\Lambda(r, m)$ mentioned in remark 3.4 by always choosing the smallest element of M as e in the algorithm.

3.7 Example

We compute the subsets of size 2 of $\{1,2,3,4\}$ as in the algorithm:

$$\begin{aligned}\Lambda(2,4) &= \Lambda\left(2,\{1,2,3,4\}\right)\\ &= \{\{1,a\} \mid a\in\{2,3,4\}\}\cup\Lambda\left(2,\{2,3,4\}\right)\\ &= \{\{1,a\} \mid a\in\{2,3,4\}\}\cup\{\{2a\} \mid a\in\{3,4\}\}\cup\Lambda\left(2,\{3,4\}\right)\\ &= \{\{1,a\} \mid a\in\{2,3,4\}\}\cup\{\{2,a\} \mid a\in\{3,4\}\}\cup\{3,4\}\cup\Lambda\left(2,\{4\}\right)\\ &= \{\{1,a\} \mid a\in\{2,3,4\}\}\cup\{\{2,a\} \mid a\in\{3,4\}\}\cup\{3,4\}\cup\{\}\cup\{\}\\ &= \{\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\}\}.\end{aligned}$$

Note that the sets in the last line are ordered with respect to the order on $\Lambda(2,4)$ introduced in remark 3.4.

3.8 Lemma

Let r, M be as above. Then

$$|\Lambda(r,M)| = \binom{|M|}{r}.$$

Proof:

If $r = 0$, then $\Lambda(r,M) = \{\emptyset\}$ and $|\Lambda(r,M)| = 1 = \binom{|M|}{0}$.

If $r > 0$ and $M = \emptyset$, then $\Lambda(r,M) = \{\}$ and $|\Lambda(r,M)| = 0 = \binom{0}{r}$.

If $r > 0$ and M is non-empty, we may fix some element and write - using the same notation as in algorithm 3.5

$$\Lambda(r,M) = \left\{\{e\}\cup\alpha \mid \alpha\in\Lambda\left(r-1,M'\right)\right\}\cup\Lambda\left(r,M'\right).$$

Since all unions in this formula are disjoint, we get

$$|\Lambda(r,M)| = |\Lambda\left(r-1,M'\right)| + |\Lambda\left(r,M'\right)|.$$

By induction on $|M|$, we obtain

$$|\Lambda(r,M)| = \binom{|M|-1}{r-1} + \binom{|M|-1}{r} = \binom{|M|}{r},$$

where the last equality follows by applying a well-known formula for binomial coefficients shifted by one. □

We now have defined the prerequisites to define the Grassmanian using minors of matrices as briefly mentioned in remark 3.2.

3.9 Definition

Given a matrix $A\in K^{r\times m}$ of rank $r\leq m$ and $I\in\Lambda(r,m)$, we may write A_I for the minor of the $r\times r$-submatrix we get by deleting all columns but those with indices in I.

We define $v(A)$ to be the vector of minors in $K^{\binom{m}{r}}$ with $v(A)_I = A_I$ (where we use the order on $\Lambda(r,m)$ from remark 3.4 to identify each subset I with a natural number in $\left\{1, ..., \binom{r}{m}\right\}$). Note that $v(A) \neq 0$ since A has rank r.

The *Grassmanian* $G(r,m)$ is the projective variety in $\mathbb{P}_K^{\binom{m}{r}-1}$ formed by the vectors $v(A)$:

$$G(r,m) = \left\{ \langle v(A) \rangle \mid A \in K^{r \times m}, \text{rank}(A) = r \right\}.$$

Unfortunately, this set description is not very useful for computational purposes. We will now construct an ideal such that $G(r,m)$ is its vanishing set, and we start by defining the ring it lives in.

3.10 Definition
The *Pluecker coordinate ring* for $r, m \in \mathbb{N}$, $r \leq m$ is the polynomial ring

$$K[\underline{p}] = K[\, p_I \mid I \in \Lambda(r,m) \,].$$

We call the variables p_I *Pluecker coordinates*.

3.11 Example
For $r = 2$, $m = 4$, we have

$$K[\underline{p}] = K[p_{\{1,2\}}, p_{\{1,3\}}, p_{\{1,4\}}, p_{\{2,3\}}, p_{\{2,4\}}, p_{\{3,4\}}]$$

as computed in example 3.7.

3.12 Definition
We call a polynomial $f \in K[\underline{p}]$ a *relation among the $r \times r$-minors of $r \times m$-matrices*, if $f(v(A)) = 0$ for all $A \in K^{r \times m}$ of rank r. We define the *Pluecker ideal* $I_{r,m}$ to be the ideal of all such relations:

$$I_{r,m} = \left\{ f \in K[\underline{p}] \mid f \text{ is a relation among the } r \times r\text{-minors of } r \times m\text{-matrices} \right\}.$$

Using the insertion homomorphism, it is easy to see that $I_{r,m}$ is indeed an ideal and one can show that it is prime and the Grassmanian is its vanishing set.

This still does not provide us with a finite description, since in general, there are infinitely many of these relations and it is not clear how to find them. In the following, we will study a combinatorial way to get a finite set of generators for $I_{r,m}$.

3.13 Definition

Let $I \in \Lambda(r-1, m)$ and $J \in \Lambda(r+1, m)$ be subsets for r, m as above.
We define the *Pluecker relation* $P_{I,J}$ to be the following quadric polynomial in $K[\underline{p}]$:

$$P_{I,J} = \sum_{j \in J} \operatorname{sgn}(j, I, J)\, p_{I+j}\, p_{J-j},$$

where

- $\operatorname{sgn}(j, I, J) = (-1)^k$ with $k = |\{j' \in J, j < j'\}| + |\{i' \in I, i' < j\}|$,
- $p_{J-j} = p_{J \setminus \{j\}}$,
- $p_{I+j} = \begin{cases} 0 & , j \in I, \\ p_{I \cup \{j\}} & , \text{else.} \end{cases}$

3.14 Remark

Note that if we represent subsets as ordered lists as described in remark 3.4, then

$$\operatorname{sgn}(j, I, J) = (-1)^{(r+1-pos_J)+(pos_I-1)} = (-1)^{r-pos_J+pos_I}$$

where pos_J is the index of the component of J which stores j and pos_I is the index of the component of $I \cup \{j\}$ where we inserted j. Since we know the index of j in J and we may modify our procedure for insertion to also return the position, this can be used to avoid computing the sign explicitly in an algorithm to compute $P_{I,J}$.

3.15 Example

We compute two relations for $r = 2, m = 4$:

a) Let $I = \{2\}, J = \{1, 3, 4\}$. Then we get the Pluecker relation

$$\begin{aligned} P_{I,J} &= \operatorname{sgn}(1, I, J)\, p_{\{1,2\}} p_{\{3,4\}} \\ &+ \operatorname{sgn}(3, I, J)\, p_{\{2,3\}} p_{\{1,4\}} \\ &+ \operatorname{sgn}(4, I, J)\, p_{\{2,4\}} p_{\{1,3\}} \\ &= (-1)^{2-1+1}\, p_{\{1,2\}} p_{\{3,4\}} \\ &+ (-1)^{2-2+2}\, p_{\{2,3\}} p_{\{1,4\}} \\ &+ (-1)^{2-3+2}\, p_{\{2,4\}} p_{\{1,3\}} \\ &= p_{\{1,2\}} p_{\{3,4\}} + p_{\{2,3\}} p_{\{1,4\}} - p_{\{2,4\}} p_{\{1,3\}} \\ &= p_{\{1,4\}} p_{\{2,3\}} - p_{\{1,3\}} p_{\{2,4\}} + p_{\{1,2\}} p_{\{3,4\}}. \end{aligned}$$

b) Let $I = \{2\}, J = \{1,2,4\}$. Then we get the Pluecker relation

$$\begin{aligned} P_{I,J} &= \operatorname{sgn}(1, I, J)\, p_{\{1,2\}} p_{\{2,4\}} \\ &+ \operatorname{sgn}(3, I, J)\, p_{\{2\}+2} p_{\{1,4\}} \\ &+ \operatorname{sgn}(4, I, J)\, p_{\{2,4\}} p_{\{1,2\}} \\ &= (-1)^{2-1+1}\, p_{\{1,2\}} p_{\{2,4\}} \\ &+ 0 \\ &+ (-1)^{2-3+2}\, p_{\{2,4\}} p_{\{1,2\}} \\ &= p_{\{1,2\}} p_{\{2,4\}} - p_{\{2,4\}} p_{\{1,2\}} \\ &= 0. \end{aligned}$$

3.16 Theorem

The Pluecker ideal is the ideal generated by the Pluecker relations:

$$I_{r,m} = \langle\, P_{I,J} \mid I \in \Lambda(r-1, m), J \in \Lambda(r+1, m) \,\rangle.$$

Proof:

See [MS05, p. 277] for a proof. □

3.17 Corollary

The Pluecker ideal $I_{r,m}$ is homogeneous.

Proof:

The set of all $P_{I,J}$ as above is a set of homogeneous generators, since all non-zero elements have degree 2. □

3.18 Example

Let $r = 2$, $m = 4$. In example 3.15, we have computed

$$f = P_{\{2\},\{1,3,4\}} = p_{\{1,4\}} p_{\{2,3\}} - p_{\{1,3\}} p_{\{2,4\}} + p_{\{1,2\}} p_{\{3,4\}}.$$

If we compute all relations, we notice that we have $P_{I,J} \in \{0, f, -f\}$ for all suitable I and J. We conclude by theorem 3.16 that the Pluecker ideal defining the Grassmanian $G(2,4)$ is the principal ideal

$$I_{2,4} = \langle\, p_{\{1,4\}} p_{\{2,3\}} - p_{\{1,3\}} p_{\{2,4\}} + p_{\{1,2\}} p_{\{3,4\}} \,\rangle.$$

By theorem 3.16, it should be obvious how to derive pseudo-code to compute generators for $I_{r,m}$ from the definitions and you can find an implementation of the algorithms under the names `plueckerRelation` respectively `grassmanianGenerators` for the computer algebra system SINGULAR in the appendix of this thesis in `grassmanian.lib`.

One should note that the set of Pluecker relations is in general not a Groebner basis for $I_{r,m}$. Since most algorithms in computer algebra require ideals to be given via a Groebner basis, we will study another algorithm to compute generators for $I_{r,m}$ that do form a

Groebner basis with respect to a special order. It may be faster to use this second algorithm instead of first calling `grassmanianGenerators` and then applying a Groebner basis computation to its result.

3.19 Definition
Let r, m be as above. We define the map [.] as follows

$$\begin{array}{rccl} [.]: & \{1,...,m\}^r & \to & K[\underline{p}] \\ & v & \mapsto & [v] \quad = \begin{cases} 0 & , v \text{ contains duplicate entries,} \\ \operatorname{sgn}(\sigma_v) * p_{set(v)} & , \text{else,} \end{cases} \end{array}$$

where $set(v) \in \Lambda(r,m)$ is the set containing the components of v and σ_v is the unique permutation which sorts the components of v in ascending order.

We omit brackets and commas when using this map. For example, we may write [1 3 2] instead of [(1,3,2)].

3.20 Remark
The definition above is justified by the following intuition: Given a matrix $A \in K^{r\times m}$, any vector $v \in \{1,...,m\}^r$ defines a matrix $A_v \in K^{r\times r}$ which has the v_i^{th} column of A as its i^{th} column for all $i \in \{1,...,r\}$. If this vector has duplicate entries, the columns are linearly dependent and the determinant of A_v is zero. Only if the components of v are sorted in ascending order, A_v is actually a submatrix of A, but since the determinant is alternating, we can always find a submatrix A' with $\det(A') = \pm\det(A_v)$ by swapping columns, where the sign is the sign of the permutation used to sort the column indices as above.

3.21 Definition
Let $t, s \in \mathbb{N}$, $s \leq t$. Given $\tau \in \Lambda(s,t)$, we write $\overline{\tau}$ for its complement, the unique subset in $\Lambda(t-s,t)$ which contains exactly the elements of $\{1,...,t\}$ missing in τ.

We write $\operatorname{sgn}(\tau,\overline{\tau})$ for the sign of the permutation $\sigma \in \mathbb{S}_t$ with

$$\sigma_i = \begin{cases} \tau_i & , i \leq s, \\ \overline{\tau}_{i-s} & , \text{else.} \end{cases}$$

3.22 Definition
Let r, m be as above, $s \in \{1,...,r\}$ and let $\alpha \in \Lambda(s-1,m)$, $\beta \in \Lambda(r+1,m)$, $\gamma \in \Lambda(r-s,m)$ be subsets. The *van der Waerden syzygy* $[[\alpha\dot{\beta}\gamma]]$ is a quadric polynomial in $K[\underline{p}]$ defined as

$$[[\alpha\dot{\beta}\gamma]] = \sum_{\tau\in\Lambda(s,r+1)} \operatorname{sgn}(\tau,\overline{\tau}) * [\alpha_1...\alpha_{s-1}\beta_{\overline{\tau}_1}...\beta_{\overline{\tau}_{r+1-s}}] * [\beta_{\tau_1}...\beta_{\tau_s}\gamma_1...\gamma_{r-s}].$$

If $\alpha_{s-1} < \beta_{s+1}$ and $\beta_s < \gamma_1$, we call $[[\alpha\dot{\beta}\gamma]]$ a *straightening syzygy*.

3.23 Example

We compute two van der Waerden syzygies for $r = 2, m = 4$.

a) Let $\alpha = \{2\}, \beta = \{1,3,4\}, \gamma = \{\}$. This implies $s = 2$.

$$\begin{aligned}
[[\alpha\dot{\beta}\gamma]] &= [[2\dot{1}\dot{3}4]] \\
&= \operatorname{sgn}(\{1,2\},\{3\})\,[24]\,[13] \\
&+ \operatorname{sgn}(\{1,3\},\{2\})\,[23]\,[14] \\
&+ \operatorname{sgn}(\{2,3\},\{1\})\,[21]\,[34] \\
&= (-1)^0\, p_{\{2,4\}}\, p_{\{1,3\}} \\
&+ (-1)^1\, p_{\{2,3\}}\, p_{\{1,4\}} \\
&- (-1)^2\, p_{\{1,2\}}\, p_{\{3,4\}} \\
&= p_{\{2,4\}}\, p_{\{1,3\}} - p_{\{2,3\}}\, p_{\{1,4\}} - p_{\{1,2\}}\, p_{\{3,4\}} \\
&= -\left(p_{\{1,4\}}p_{\{2,3\}} - p_{\{1,3\}}p_{\{2,4\}} + p_{\{1,2\}}p_{\{3,4\}}\right).
\end{aligned}$$

Note that this is equal to $-P_{\alpha,\beta}$ as computed in example 3.15, which is no coincidence, because the Pluecker relations are special cases of van der Waerden syzygies.

b) Let $\alpha = \{2\}, \beta = \{1,2,4\}, \gamma = \{\}$. This again implies $s = 2$.

$$\begin{aligned}
[[\alpha\dot{\beta}\gamma]] &= [[2\dot{1}\dot{2}4]] \\
&= \operatorname{sgn}(\{1,2\},\{3\})\,[24]\,[12] \\
&+ \operatorname{sgn}(\{1,3\},\{2\})\,[22]\,[24] \\
&+ \operatorname{sgn}(\{2,3\},\{1\})\,[21]\,[14] \\
&= (-1)^0\, p_{\{2,4\}}\, p_{\{1,2\}} \\
&+ (-1)^1 * 0 * p_{\{2,4\}} \\
&- (-1)^2\, p_{\{1,2\}}\, p_{\{1,4\}} \\
&= p_{\{2,4\}}\, p_{\{1,2\}} - p_{\{2,4\}}\, p_{\{1,2\}} \\
&= 0.
\end{aligned}$$

3.24 Definition

Let $>_{dp}$ be the usual degree reverse lexicographical ordering on $K[\underline{p}]$ defined by

$$\underline{p}^A >_{dp} \underline{p}^B :\Leftrightarrow \deg(\underline{p}^A) > \deg(\underline{p}^B)$$
or (degrees equal and rightmost non-zero entry of $A - B$ is negative).

In the context of the Pluecker coordinate ring, $>_{dp}$ is called *tableaux order* since we may write monomials $\underline{p}^A$ as tables called *tableaux* of size $\deg(\underline{p}^A) \times r$ by writing the sets I corresponding to the variables p_I occurring in $\underline{p}^A$ into the rows of the tableau (as tuples and sorted lexicographically as mentioned in remark 3.4). Then we have that $\underline{p}^A > \underline{p}^B$ if either the tableau A for $\underline{p}^A$ has more rows than the tableau $\underline{p}^B$ or the row-count is equal and there are indices i, j such that the first $i-1$ rows are equal and the entries of the i^{th} row coincide up to j with $A_{i,j} > B_{i,j}$.

3.25 Example

The quadric monomials

$$p_{\{1,4\}}p_{\{2,3\}}, \quad p_{\{1,3\}}\,p_{\{2,4\}}, \quad p_{\{1,2\}}\,p_{\{3,4\}}$$

in $K[\underline{p}]$ for $r = 2$, $m = 4$ corresponds to the following three tableaux of size 2×2

$$\begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}, \quad \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}, \quad \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}.$$

By comparing the first rows, we see that the terms of the polynomial

$$p_{\{1,4\}}p_{\{2,3\}} - p_{\{1,3\}}p_{\{2,4\}} + p_{\{1,2\}}p_{\{3,4\}}$$

are ordered with respect to the tableaux order $>_{dp}$.

3.26 Theorem

Let $S_{r,m}$ be the set of *straightening syzygies*, i.e.

$$S_{r,m} = \left\{ [[\alpha\dot{\beta}\gamma]] \;\middle|\; \begin{array}{c} s \in \{1, ..., r\}, \\ \alpha \in \Lambda(s-1, m), \beta \in \Lambda(r+1, m), \gamma \in \Lambda(r-s, m), \\ \alpha_{s-1} < \beta_{s+1}, \beta_s < \gamma_1 \end{array} \right\}$$

and let

$$S^*_{r,m} = \left\{ [[\alpha\dot{\beta}\gamma]] \in S_{r,m} \mid \alpha_i \leq \beta_i \text{ for } i \in \{1, ..., s-1\} \right\}.$$

Both $S_{r,m}$ and $S^*_{r,m}$ are Groebner bases for $I_{r,m}$ with respect to the tableaux order $>_{dp}$.

Proof:

See [Stu93, p. 81f]. □

Again, the last theorem gives an obvious way to compute a Groebner basis for $I_{r,m}$: Iterate through all possibilities for s, α, β and γ, check if the conditions in the definition of $S_{r,m}$ and $S^*_{r,m}$ are fulfilled and compute $[[\alpha\dot{\beta}\gamma]]$ in this case. However, this will lead to many unnecessary computations, since only a fraction of all possibilities actually satisfy the conditions. The next theorem shows how we can compute the subsets leading to syzygies in $S^*_{r,m}$ in a more efficient way.

3.27 Theorem

The following sets are equal:

$$LHS = \left\{ (\alpha, \beta, \gamma) \;\middle|\; \begin{array}{c} s \in \{1, ..., r\}, \\ \alpha \in \Lambda(s-1, m), \beta \in \Lambda(r+1, m), \gamma \in \Lambda(r-s, m), \\ \alpha_{s-1} < \beta_{s+1}, \beta_s < \gamma_1, \alpha_i \leq \beta_i \text{ for } i \in \{1, ..., s-1\} \end{array} \right\}$$

$$RHS = \left\{ (\alpha, \beta, \gamma) \;\middle|\; \begin{array}{c} R, S \in \Lambda(r, m), \\ \nu = \nu(R, S) \neq 0, \\ \alpha = R_1 ... R_{\nu-1}, \beta = S_1 ... S_\nu R_\nu ... R_r, \gamma = S_{\nu+1} ... S_r \end{array} \right\}$$

where $\nu(R, S)$ is the first index i such that $R_i > S_i$ or 0 if no such index exists.

Proof:

"$LHS \subseteq RHS$"

Assume s, α, β and γ as in the definition of LHS are given, We may write $\beta = \beta'\beta''$ where $\beta' = \beta_1 ... \beta_s$ and $\beta'' = \beta_{s+1} ... \beta_{r+1}$.

Define $R = \alpha\beta''$ and $S = \beta'\gamma$ and note that by the conditions coming from the definition of straightening syzygy, those are ordered lists and thus represent subsets of size r.

The condition $\alpha_i \leq \beta_i$ for $i \in \{1, ..., s-1\}$ coming from the definition of $S^*_{r,m}$ guarantees $R_i = \alpha_i \leq \beta_i = \beta'_i = S_i$ for $1 \leq i \leq s-1$. Since $R_s = \beta''_1 = \beta_{s+1}$ and $S_s = \beta'_s = \beta_s$ and β was an ordered list, we have $R_s > S_s$ and $s = \nu(S, R)$ is indeed the first index where R is larger than S.

Therefore, the definition of RHS recovers our original α, β and γ for R, S as constructed.

"$LHS \supseteq RHS$"

Let R, S be subsets (respectively ordered lists representing them) as in the definition of RHS. First note that for $s = \nu = \nu(R, S) \in \{1, ..., r\}$, α, β and γ are subsets of the correct size. By the choice of ν, $S_\nu < R_\nu$, so β as in the definition of RHS is an ordered list.

Since R was a subset of $\{1, ..., r\}$ represented as ordered list,

$$\alpha_{s-1} = R_{\nu-1} < R_\nu = \beta_{s+1}$$

holds. Analogously,

$$\beta_s = S_\nu < S_{\nu+1} = \gamma_1,$$

so the van der Waerden-sygygy given by α, β and γ is indeed a straightening syzygy. Furthermore, the choice of ν as the first index i with $R_i > S_i$ immediately gives

$$\alpha_i = R_i \leq S_i = \beta_i \text{ for } 1 \leq i \leq s-1.$$

□

Before we state the resulting algorithm, we note that we can further improve the running time of the algorithm by a factor of about $\frac{1}{2}$ since we only have to consider pairs R, S with $S > R$ with respect to the order introduced in remark 3.4.

3.28 Proposition
With notation as in theorem 3.26, the following set is a Groebner basis with respect to the tableaux order for $I_{r,m}$:

$$S^{**}_{r,m} = \left\{ [[\alpha\dot{\beta}\gamma]] \in S_{r,m} \mid \alpha_i \leq \beta_i \text{ for } i \in \{1, ..., s-1\}, \exists j \in \{1, ..., s-1\} : \alpha_j < \beta_j \right\}.$$

Proof:
The proof of theorem 3.26 in [Stu93] relies on the construction of R, S (and subsequently the resulting α, β and γ) as the $(i-1)^{th}$ and i^{th} row of a tableau such that there is some position s with $R_s > S_s$. Since the rows of a tableau are sorted with respect to the order introduced in remark 3.4, the case $R_i = S_i$ for all $i \in \{1, ..., s-1\}$ and $R_s > S_s$ cannot occur if R has a lower row index than S. Therefore, we can assume the restriction defining $S^{**}_{r,m}$ and the proof remains valid without any change. □

3.29 Proposition
Let $R, S \in \Lambda(r, m)$ be sets with $\nu(R, S) \neq 0$ and let α, β and γ be the lists constructed as in *RHS* in theorem 3.27. The van Der Waerden syzygy $[[\alpha\dot{\beta}\gamma]]$ is in $S^{**}_{r,m}$ if and only if $S > R$.

Proof:
We have already seen, that the van der Waerden syzygies defined by the S, R without the restriction form the set $S^{*}_{r,m}$. Let us show that the sets with $S \geq R$ do not contribute to $S^{**}_{r,m}$.
If $R = S$, there is no position s such that $R_s > S_s$. With notation as in theorem 3.27, we get that $\nu(R, S) = 0$ and we have shown in this theorem that we do not need those pairs to get all van der Waerden syzygies in $S^{*}_{r,m} \supseteq S^{**}_{r,m}$.
Let us assume $S < R$ now. By the definition of the order, the smallest element not contained in both is contained in S. Since $s = \nu(R, S)$ is the first position such that $R_s > S_s$ we conclude that R, S are of the form

$$R = a\,R_s b, \quad S = a\,S_s c$$

for suitable lists a, b, c. As in the theorem, we construct

$$\alpha = a, \quad \beta = a\, S_s R_s b, \quad \gamma = c$$

and we note that $\alpha_i = \beta_i$ for all $i \in \{1, ..., s-1\}$, so the resulting van der Waerden syzygy $[[\alpha\dot{\beta}\gamma]]$ is not in $S^{**}_{r,m}$
Vice versa, if $S > R$, then the smallest position not contained in both is contained in R. Therefore, we conclude that R, S are of the form

$$R = ax\, R_s b, \quad S = ay\, S_s c$$

for suitable lists a, x, y, b, c where x and y are non-empty with $x_1 < y_1$. The resulting lists are

$$\alpha = ax, \quad \beta = ay\, S_s R_s b, \quad \gamma = c$$

and we note that at the position $|a| + 1 < s$, we have $\alpha_{|a|+1} = x_1 < y_1 = \beta_{|a|+1}$, so the syzygy is in $S^{**}_{r,m}$. □

We finish this chapter by stating the algorithm to compute $S^{**}_{r,m}$.

3.30 Algorithm
Input:

$r \in \mathbb{N}$,
$m \in \mathbb{N}$ with $r \leq m$

Output:

$S^{**}_{r,m}$, a Groebner basis for $I_{r,m} \subseteq K[\underline{p}]$ with respect to $>_{dp}$

```
1  G := ∅
2  foreach R ∈ Λ(r, m) do
3      foreach S ∈ Λ(r, m), S > R do
4          s := ν(R, S)
5          if s ≠ 0 then
6              α := R_1 ... R_{s-1}
7              β := S_1 ... S_s R_s ... R_r
8              γ := S_{s+1} ... S_r
9              G := G ∪ {[[αβ̇γ]]}
10         end
11     end
12 end
13 return G
```

where $\nu(R, S)$ is the first index $i \in \{1, ..., r\}$ with $R_i > S_i$ and 0 if no such index exists.

Proof of termination:
The set $\Lambda(r,m)$ is finite, $\nu(R,S)$ and $[[\alpha\dot{\beta}\gamma]]$ can be computed in finite time using their definitions. □

Proof of soundness:
By proposition 3.28, the set $S^{**}_{r,m}$ is a Groebner basis for $I_{r,m}$ with respect to the tableaux order $>_{dp}$. We have seen in theorem 3.27 and proposition 3.29 that the set computed by the algorithm is $S^{**}_{r,m}$. □

The SINGULAR library `grassmanian.lib` provides a procedure implementing this algorithm under the name `grassmanianGB`.

3.31 Example
Let $r = 2, m = 4$. The pair of sets $R = \{1,4\}, S = \{2,3\}$ is the only pair such that $S > R$ and $\nu(R,S) \neq 0$. Note that $s = 2$ and the resulting subsets are $\alpha = \{1\}, \beta = \{2,3,4\}$, $\gamma = \{\}$ and the van der Waerden syzygy given by these is

$$[[1\dot{2}\dot{3}\dot{4}]] = p_{\{1,4\}} * p_{\{2,3\}} - p_{\{1,3\}} * p_{\{2,4\}} + p_{\{1,2\}} * p_{\{3,4\}}$$

and it is the only element of $S^{**}_{r,m}$. Since we already computed this polynomial as the generator for $I_{2,4}$ in example 3.18 and any generator of a principal ideal is always a Groebner basis with respect to any ordering, this is rather unsurprising, but for some small numbers like $r = 2, m = 5$, the set of Pluecker relations is a Groebner basis although $I_{r,m}$ is not principal.

3.32 Example
Let $r = 3, m = 6$. The sets $R = \{1,4,6\}, S = \{2,3,5\}$ with $S > R$ and $\nu(R,S) = 2$ get split into $\alpha = \{1\}, \beta = \{2,3,4,6\}, \gamma = \{5\}$. The resulting syzygy

$$\begin{aligned}
[[\alpha\dot{\beta}\gamma]] &= [[1\dot{2}\dot{3}\dot{4}\dot{6}5]] \\
&= p_{\{1,4,5\}} * p_{\{2,3,6\}} - p_{\{1,3,5\}} * p_{\{2,4,6\}} - p_{\{1,3,4\}} * p_{\{2,5,6\}} \\
&+ p_{\{1,2,5\}} * p_{\{3,4,6\}} + p_{\{1,2,4\}} * p_{\{3,5,6\}} - p_{\{1,2,3\}} * p_{\{4,5,6\}}
\end{aligned}$$

does not occur as Pluecker relation $P_{I,J}$, but it is contained in a Groebner basis for $I_{r,m}$ as the syzygy polynomial of the Pluecker relations

$$\begin{aligned}
P_{\{1,5\},\{2,3,4,6\}} &= p_{\{1,5,6\}} * p_{\{2,3,4\}} + p_{\{1,4,5\}} * p_{\{2,3,6\}} \\
&- p_{\{1,3,5\}} * p_{\{2,4,6\}} + p_{\{1,2,5\}} * p_{\{3,4,6\}}
\end{aligned}$$

and

$$\begin{aligned}
P_{\{5,6\},\{1,2,3,4\}} &= -p_{\{1,5,6\}} * p_{\{2,3,4\}} + p_{\{1,3,4\}} * p_{\{2,5,6\}} \\
&- p_{\{1,2,4\}} * p_{\{3,5,6\}} + p_{\{1,2,3\}} * p_{\{4,5,6\}}.
\end{aligned}$$

Part III.
Introduction to tropicalizations

In this part, we will define tropical varieties, the main objects of tropical geometry. The next three chapters will show several approaches to tropicalizations, each emphasizing different aspects and giving a different point of view. In the fourth chapter, we will cite the fundamental theorem, which states that all given definitions are equivalent and we will develop some polyhedral theory to understand the structure of tropical varieties.

We follow the presentation from Bernd Sturmfels' and Diane Maclagan's book [MS15] with two major differences: Firstly, Maclagan and Sturmfels define everything in terms of very affine varieties given by ideals in the Laurent polynomial ring and with arbitrary valuation on the ground field, while we restrict ourselves to polynomials with non-negative exponents and we only consider the trivial valuation. Secondly, we follow the convention used in Anders Jensen PhD thesis [Jen07] and SINGULAR , which means that compared to [MS15], all tropical varieties are reflected across the origin.

4. Tropicalizations using the tropical semiring

The first approach to tropicalizations uses modified algebraic geometry over a set called the tropical semiring. Unlike the two other methods, it provides a good intuition about the structure of tropical varieties, so I chose to present it firstly.

4.1 Definition
We define the *tropical semiring* to be

$$(\mathbb{R} \cup \{-\infty\}, \oplus, \odot),$$

where the binary operations are defined by

$$\begin{aligned} a \oplus b &= \max\{a, b\} \\ a \odot b &= a + b \end{aligned}$$

with the following rules to deal with the special element $-\infty$:

$$\begin{aligned} \max\{-\infty, b\} &= b \\ -\infty + b &= b + (-\infty) = -\infty \end{aligned}$$

for all $b \in \mathbb{R} \cup \{-\infty\}$.

4.2 Remark
Obviously, for the "multiplication" $\odot$ on $\mathbb{R}$, the usual properties of addition hold: 0 is the neutral element and for $a \in \mathbb{R}$, $-a$ is its inverse, but $-\infty$ has no inverse.

Note that $-\infty$ is the neutral element with respect to "addition" via $\oplus$, but no inverse elements exist, hence, we use the name *semiring*.

4.3 Remark
In some part of the literature, the isomorphic semiring

$$(\mathbb{R} \cup \{\infty\}, \min, \odot)$$

is used and also called tropical semiring. As already mentioned in the introduction to this part, this would lead to an inversion of the sign in the definition of tropicalization and finally to a reflection across the origin on the geometric side.

4.4 Notation
We will write $K[\underline{x}]$ instead of $K[x_1, ..., x_n]$ to denote the polynomial ring in n indeterminates over K. If $\alpha \in \mathbb{N}^n$, we may use it as multi-index and write $\underline{x}^\alpha$ instead of $x_1^{\alpha_1}...x_n^{\alpha^n}$.

4.5 Remark
We may evaluate polynomials in the tropical semiring. This will associate a polynomial in $\mathbb{Q}[\underline{x}]$ of the form

$$f = \sum_{\alpha \in \mathbb{N}^n} c_\alpha \underline{x}^\alpha$$

to the following expression over the tropical semi-ring:

$$\max_{\alpha \in \mathbb{N}^n, c_\alpha \neq 0} \{c_\alpha + \alpha^T \underline{x}\}.$$

To get a consistent theory, it is better to replace the coefficients c_α by their valuation. Here, we will only consider the case of the trivial valuation, which maps each element to zero, so we get

$$\text{troppoly}(f) = \max_{\alpha \in \mathbb{N}^n, c_\alpha \neq 0} \{\alpha^T \underline{x}\}.$$

By convention, we set

$$\text{troppoly}(0) = -\infty$$

since $-\infty$ is the neutral element with respect to taking the maximum.

One could try to apply the usual definitions from algebraic geometry to these tropical polynomials, but the condition of "being zero" at some point is not very useful in this setting. The zero with respect to addition is $-\infty$, but as long as $f \neq 0$ and we only evaluate troppoly(f) at real vectors, it will never occur as result.

Thus, we need a better criterion for points to be in the geometric object defined by a tropical polynomial. By the definition of tropical polynomials, it is quite clear that they define piecewise linear functions. Instead of studying their vanishing set, we can study the set of points at which the function is non-linear.

4.6 Definition

Let $f \in \mathbb{Q}[\underline{x}]$, $f \neq 0$ be a polynomial. We may view troppoly(f) as function

$$\begin{array}{rccc} \text{troppoly}(f) & : & \mathbb{R}^n & \to & \mathbb{R} \\ & & w & \mapsto & \max\limits_{\substack{\alpha \in \mathbb{N}^n, \\ c_\alpha \neq 0}} \{\alpha^T w\} \end{array}$$

by evaluating the expression from remark 4.5. As mentioned above, we get indeed troppoly(f)(w) $\in \mathbb{R}$ for any $w \in \mathbb{R}^n$ since we restricted ourselves to rational coefficients and plug in only real vectors.

We define the *tropicalization* Trop (f) of f as the subset of $\mathbb{R}^n$ such that the maximum in troppoly(f) is attained in at least two terms:

$$\text{Trop}\,(f) = \{w \in \mathbb{R}^n \mid \text{the value troppoly}(f)(w) \text{ is attained at least twice}\}.$$

More explicitly

$$\text{Trop}\,(f) = \left\{ \quad w \in \mathbb{R}^n \quad \middle| \quad \begin{array}{c} \exists \alpha, \beta \in \mathbb{N}^n : \text{troppoly}(f)(w) = \alpha^T w = \beta^T w, \\ \alpha \neq \beta, c_\alpha \neq 0, c_\beta \neq 0 \end{array} \right\}.$$

By convention, we define

$$\text{Trop}\,(0) = \mathbb{R}^n.$$

Given an ideal $I \subseteq \mathbb{Q}[\underline{x}]$, we can define its tropicalization as the intersection of the tropicalizations of its elements, just as in traditional algebraic geometry:

$$\text{Trop}\,(I) = \bigcap_{f \in I} \text{Trop}\,(f).$$

4.7 Example

Let $f = x + y \in \mathbb{Q}[x, y]$. We get

$$\text{troppoly}(f) = \max\{x, y\}$$

and therefore

$$\text{Trop}\,(f) = \left\{(w_1, w_2) \in \mathbb{R}^2 \mid w_1 = w_2\right\} = \left\{(w_1, w_1) \in \mathbb{R}^2\right\}.$$

5. Tropicalizations using initial forms

The definitions in the last section give a good intuition of tropicalizations, but they are not very good for computational purposes. The approach in this section provides the foundation to actually compute tropical varieties. We restrict ourselves to polynomial rings with $\mathbb{Q}$ as ground field and we only consider the trivial valuation on $\mathbb{Q}$, which maps each element to zero.

5.1 Definition
Let $w \in \mathbb{R}^n$ be a weight vector. We define the *(weighted) degree with respect to w* (or w-degree) of a monomial in $K[\underline{x}]$ as

$$\deg_w(\underline{x}^\alpha) = w^T\alpha$$

and the (weighted) degree of a non-zero polynomial as usual as the maximum of the degrees of the monomials:

$$\deg_w\left(\sum_{\alpha\in\mathbb{N}^n} c_\alpha \underline{x}^\alpha\right) = \max_{\alpha\in\mathbb{N}^n}\left\{w^T\alpha \mid c_\alpha \neq 0\right\}.$$

We call the sum of the terms with maximal degree the *initial form* of the polynomial

$$\mathrm{in}_w(f) = \mathrm{in}_w\left(\sum_{\alpha\in\mathbb{N}^n} c_\alpha \underline{x}^\alpha\right) = \sum_{\substack{\alpha\in\mathbb{N}^n,\\ \deg_w(\underline{x}^\alpha)=\deg_w(f)}} c_\alpha \underline{x}^\alpha.$$

By convention, we set $\mathrm{in}_w(0) = 0$.

5.2 Example
Unlike leading terms with respect to monomial orderings, initial forms may not be monomial if several terms have the same maximal degree. Let $f = x + y \in \mathbb{Q}[x, y]$ and $w = (1, 1)^T$, then $\mathrm{in}_w(f) = x + y$ since both terms have weighted degree 1.

5.3 Definition
A polynomial $f \in K[\underline{x}]$ is called *homogeneous* with respect to some weight vector $w \in \mathbb{R}^n$ (or w-homogeneous) if $\mathrm{in}_w(f) = f$, that is all terms of f have the same w-degree t.
An ideal $I \subseteq K[\underline{x}]$ is called *w-homogeneous* if there is a set $G \subseteq K[\underline{x}]$ consisting of w-homogeneous polynomials with $I = \langle G\rangle$.
If we just write homogeneous, we mean homogeneous with respect to the weight vector $(1, 1, ..., 1) \in \mathbb{R}^n$. Similarly, the degree of a polynomial is as usual the weighted degree with respect to this vector.

One could generalize most of the theory for ideals which are homogeneous with respect to some strictly positive vector $w \in \mathbb{R}^n_{>0}$. Since the Pluecker ideal $I_{r,m}$, which is our main

example, is homogeneous with respect to $(1, ..., 1)$, we restrict ourselves to this less general definition.

5.4 Definition
Let $w \in \mathbb{R}^n$ be a weight vector and $I \subseteq K[\underline{x}]$ a set of polynomials. We define the *initial ideal* of I with respect to w to be the ideal

$$\mathrm{in}_w(I) = \langle\ \mathrm{in}_w(f)\ \mid f \in I\ \rangle$$

generated by all initial forms.
We say that a set of polynomial is *monomial-free* if it contains no single term, i.e. no polynomial of the form $c_\alpha \underline{x}^\alpha$ with $c_\alpha \neq 0$.
This is equivalent to the condition that the set contains no monomial if the set is closed under the multiplication with units, since K^* is the set of units of $K[\underline{x}]$. In particular, an ideal is monomial-free if it contains no monomial.

5.5 Definition
Let $I \subseteq \mathbb{Q}[\underline{x}]$ be an ideal. Then we define its tropicalization as the set of weight vectors such that the corresponding initial ideal contains no monomial:

$$\mathrm{Trop}(I) = \{w \in \mathbb{R}^n \mid \mathrm{in}_w(I) \text{ is monomial-free}\}.$$

5.6 Example
If an ideal $I \subseteq \mathbb{Q}[\underline{x}]$ contains the monomial $\underline{x}^\alpha$, its initial ideal also contains it, since $\mathrm{in}_w(\underline{x}^\alpha) = \underline{x}^\alpha$ with respect to any weight vector. In particular, we have $\mathrm{Trop}(I) = \emptyset$ in this case.

5.7 Remark
In computer algebra systems like SINGULAR, ideals are usually given via a finite set of generators but, as for leading ideals, the initial forms of an arbitrary set of generators do not generate the initial ideal in general.

Note that the initial ideal consists of all linear combinations of initial forms. We will now show that it is sufficient to check that each initial form is not a monomial.

5.8 Lemma
Let $w \in \mathbb{R}^n$ and let $f, g \in K[\underline{x}]$ be polynomials where f is w-homogeneous. Then

$$f * \mathrm{in}_w(g) = \mathrm{in}_w(f * g).$$

Proof:
Since f is w-homogeneous, there is some $t \in \mathbb{R}$ such that all terms of f have w-degree t. Multiplication with f will increase the w-degree of all terms in g by t. The terms of $f * g$ with maximal w-degree are exactly the maximal terms of g multiplied with f. □

5.9 Proposition
Let $I \subseteq \mathbb{Q}[\underline{x}]$ be an ideal and $w \in \mathbb{R}^n$ a weight vector. The initial ideal $\mathrm{in}_w(I)$ is monomial-free if and only if $\{\mathrm{in}_w(I) \mid f \in I\}$ contains no monomial.

Proof:
The set $\{\mathrm{in}_w(I) \mid f \in I\}$ is closed under the multiplication with units, since $\mathrm{in}_w(uf) = u\,\mathrm{in}_w(f)$ for any unit u and any polynomial f. Furthermore, it is clearly a subset of the initial ideal, so one direction is trivial.
Assume some term $c_\alpha \underline{x}^\alpha$ for $c_\alpha \neq 0$ is contained in $\mathrm{in}_w(I)$. Since non-zero scalar factors are units in $\mathbb{Q}[\underline{x}]$, we may assume that the term is normalized, so $c_\alpha = 1$. By the definition of the initial ideal, we get that $\underline{x}^\alpha$ is a finite linear combination of initial forms, i.e.

$$\underline{x}^\alpha = \sum_{g \in I} c_g * \mathrm{in}_w(g)$$

for some $c_g \in \mathbb{Q}[\underline{x}]$. The left hand-side has w-degree $t = w^T\alpha$, so we may as well assume that all terms on the right-hand side have this w-degree, since the rest has to cancel out. The inital forms with respect to w are w-homogeneous by definition, so we can assume that all non-zero c_g are w-homogeneous such that such that $c_g * \mathrm{in}_w(g)$ has w-degree t. By lemma 5.8, we have

$$\underline{x}^\alpha = \sum_{g \in I} \mathrm{in}_w\left(c_g * g\right)$$

and since I is closed under multiplication with ring elements as ideal, we may further simplify this to

$$\underline{x}^\alpha = \sum_{g' \in I} \mathrm{in}_w\left(g'\right)$$

where all non-zero g' have w-degree t. In this case, the sum commutes with taking the initial form and we get

$$\underline{x}^\alpha = \mathrm{in}_w\left(\sum_{g' \in I} g'\right) \in \{\mathrm{in}_w(f) \mid f \in I\},$$

which is an element of the desired form. □

5.10 Remark
We could also define initial forms and tropicalizations in terms of Laurent polynomials. Note that the units in the ring of Laurent polynomials are exactly the terms $c_\alpha \underline{x}^\alpha$ with $c_\alpha \neq 0$, so this would simplify the definition of $\mathrm{Trop}(I)$ to

$$\mathrm{Trop}(I) = \{w \in \mathbb{R}^n \mid \mathrm{in}_w(I) \neq \langle 1 \rangle\}.$$

Furthermore, we could take the coefficient into account when defining initial forms. As mentioned remark 4.5, this corresponds to allowing non-trivial valuations on the ground field. See [MS15] for a more general definition of initial forms.

The proposition above also allows us to deduce easily that the two definitions for Trop (I) we have given so far coincide.

5.11 Proposition
Let $I \subseteq \mathbb{Q}[\underline{x}]$ be an ideal. The following two sets are equal:

a) The set Trop (I) as in definition 4.6 using geometry over the tropical semiring.

b) The set Trop (I) as in definition 5.5 using initial ideals.

Proof:
Let $w \in \mathbb{R}^n$ be some point in the intersection of all Trop (f) for f in I. For all $f \neq 0$, the value troppoly$(f)(w)$ is attained in at least two terms of f, i.e.

$$\text{troppoly}(f) = \max_{\alpha \in \mathbb{N}^n} \left\{ w^T \alpha \mid c_\alpha \neq 0 \right\} = w^T \beta = w^T \gamma$$

for some $\beta \neq \gamma \in \mathbb{N}^n$ with $c_\beta \neq 0 \neq c_\gamma$. But this means that $c_\beta \underline{x}^\beta$ and $c_\gamma \underline{x}^\gamma$ occur as terms in the initial form of f, so it is not monomial-free. The initial form of 0 is monomial-free by definition, so we conclude with proposition 5.9 that w is in the tropicalization Trop (I) as defined in 5.5. Since all transformations in this proof were equivalences, both directions hold. □

5.12 Example
We continue example 4.7. Let $I = \langle f \rangle = \langle x + y \rangle \subseteq \mathbb{Q}[x, y]$. We have already computed Trop $(f) = \left\{ (w_1, w_1) \in \mathbb{R}^2 \right\}$ and we want to verify Trop (f) = Trop (I).
At any point not in Trop (f), the initial form of f is either x or y, so we get Trop $(I) \subseteq$ Trop (f). Any polynomial $g \in I, g \neq 0$ can be written as $g = g'(x + y) = g'x + g'y$ for some $g' \neq 0$ and we get that $\text{in}_{(w_1,w_1)}(g) = \text{in}_{(w_1,w_1)}(g')\, x + \text{in}_{(w_1,w_1)}(g')\, y$ for any $w_1 \in \mathbb{R}$, i.e. the initial form is not a single term. Since we do not consider $\text{in}_w(0) = 0$ to be a single term for any weight vector, we conclude Trop (I) = Trop (f).

We will now study the relation of initial forms to the classical theory of computer algebra. Let us recall some definitions.

5.13 Definition
$>$ is called a *monomial ordering* on $K[\underline{x}]$ if $>$ is a relation on the monomials of $K[\underline{x}]$, which fulfills the following properties:

- For monomials $\underline{x}^\alpha$ and $\underline{x}^\beta$ exactly one of the following statements is true: $\underline{x}^\alpha > \underline{x}^\beta$, $\underline{x}^\alpha < \underline{x}^\beta$, $\underline{x}^\alpha = \underline{x}^\beta$. (In particular: $>$ is total.)

- $>$ is transitive: If $\underline{x}^\alpha > \underline{x}^\beta$ and $\underline{x}^\beta > \underline{x}^\gamma$, then $\underline{x}^\alpha > \underline{x}^\gamma$.
- $>$ respects multiplication:
  If $\underline{x}^\alpha > \underline{x}^\beta$, then for any monomial $\underline{x}^\gamma$: $\underline{x}^\alpha * \underline{x}^\gamma = \underline{x}^{\alpha+\gamma} > \underline{x}^{\beta+\gamma} = \underline{x}^\beta * \underline{x}^\gamma$.

We call $>$ *global* if $x_i > 1$ for any $i \in \{1, ..., n\}$.

Given a monomial ordering $>$ and a polynomial f (respectively an ideal I), we can define the notions of leading monomial $\mathrm{LM}_>(f)$ and leading term $\mathrm{LT}_>(f)$ (respectively leading ideal $L_>(I)$) as usual.

5.14 Definition
Let $>$ be any monomial ordering on $K[\underline{x}]$. Then $>_h$ defined by

$$\underline{x}^\alpha >_h \underline{x}^\beta \Leftrightarrow \deg(\underline{x}^\alpha) > \deg(\underline{x}^\beta) \\ \text{or (degrees equal and } \underline{x}^\alpha > x^\beta).$$

is a global monomial ordering on $K[\underline{x}]$, even if $>$ is not global.

5.15 Definition
Let $>$ be a global monomial ordering on $K[\underline{x}]$ and $I \subseteq K[\underline{x}]$ an ideal, then we may look at the set of weight vectors $w \in \mathbb{R}^n$ such that the initial ideal and the leading ideal coincide. We define $C_>(I)$ to be its closure in the euclidean topology, i.e.

$$C_>(I) = \overline{\{w' \in \mathbb{R}^n \mid \mathrm{in}_{w'}(I) = L_>(I)\}}.$$

Analogously, given a weight vector $w \in \mathbb{R}^n$, we define

$$C_w(I) = \overline{\{w' \in \mathbb{R}^n \mid \mathrm{in}_{w'}(I) = \mathrm{in}_w(I)\}}.$$

5.16 Remark
We summarize some properties of the sets defined above. See [Jen07] for proofs of these statements.

a) The definition only requires the ideals to be equal, it does not enforce $\mathrm{in}_{w'}(f) = L_>(f)$ respectively $\mathrm{in}_{w'}(f) = \mathrm{in}_w(f)$ for all $f \in I$.

b) If $>$ is a global monomial ordering, there is a weight vector w with $C_>(I) = C_w(I)$.

c) There are only finitely many different $C_>(I)$.

d) For every point $w' \in \mathbb{R}^n_{\geq 0}$ with non-negative components, there is at least one global monomial ordering $>$ with $w' \in C_>(I)$.

5.17 Definition
Let $>$ be a global monomial ordering on $K[\underline{x}]$ and $I \subseteq K[\underline{x}]$ an ideal. A *reduced Groebner basis* for I is a finite set $G \subseteq I$ such that the following properties hold:

a) $L_>(G) = L_>(I)$.

b) There are no two elements $f, g \in G$, $f \neq g$ such that $\mathrm{LM}_>(f)$ divides $\mathrm{LM}_>(g)$.

c) For all $g \in G$: $\mathrm{LC}_>(g) = 1$.

d) For all $g \in G$: No term of $g - \mathrm{LT}_>(g)$ is in $L_>(G)$.

5.18 Theorem
Let $I \subseteq \mathbb{Q}[\underline{x}]$ be an ideal and $>$ a global monomial ordering on $\mathbb{Q}[\underline{x}]$.

a) There is a unique reduced Groebner basis for I with respect to $>$, denoted by $G_>$.

b) Let $f \in \mathbb{Q}[\underline{x}]$. A standard representation for f with respect to a finite set G is a sum

$$f = \sum_{g \in G} c_g g$$

with $c_g \in \mathbb{Q}[\underline{x}]$ and $\mathrm{LM}(f) \geq \mathrm{LM}(c_g g)$. There is a standard representation for every $f \in I$ if and only if G is a Groebner basis for I, i.e. G is a finite subset of I which fulfills condition a) of definition 5.17.

c) If I is w-homogeneous for some weight vector $w \in \mathbb{R}^n$, the reduced Groebner basis $G_>$ consists only of w-homogeneous elements.

Proof:
See [Bö]. □

5.19 Lemma
Suppose $I \subseteq K[\underline{x}]$ is a homogeneous ideal, $>$ is a global monomial ordering and $>_h$ is the ordering introduced in definition 5.14. Then the reduced Groebner bases with respect to those orderings coincide: $G_> = G_{>_h}$.

Proof:
By theorem 5.18, the reduced Groebner basis $G_>$ consists only of homogeneous elements. In particular, $LM_>(g) = LM_{>_h}(g)$ for any $g \in G_>$. Therefore, if we use the Buchberger algorithm to compute Groebner bases with respect to $>_h$ starting from the set of generators $G_>$, it will terminate without adding any polynomial. See [Jen07, p. 19] for more details. □

5.20 Proposition

Let $>$ be a monomial ordering on $K[\underline{x}]$, $I \subseteq K[\underline{x}]$ an ideal and $w \in \mathbb{R}^n$. Then

$$w \in C_>(I) \Leftrightarrow \text{ for all } g \in G_> : \mathrm{LM}_>(\mathrm{in}_w(g)) = \mathrm{LM}_>(g).$$

Proof:

See [Jen07, p. 32]. □

5.21 Remark

The sets $C_>(I)$ are fundamental for the computation of Trop (I) in general and proposition 5.20 allows us to use Groebner bases to decide membership. Since Groebner bases are finite sets, the proposition provides a way to check $w \in C_>(J)$ via finitely many conditions. The conditions can be expressed as linear inequalities and those can be solved using tools from optimization. Anders Jensen's PhD thesis [Jen07] explains in detail how this works and implementations of his work are available in his own software Gfan [Gfan] as well as in the computer algebra system SINGULAR [Sing].

6. Tropicalizations using Puiseux series

In the introduction, we have stated the tropicalization implies a restriction to the torus, but so far, we have not seen this explicitly. We will now show a purely algebraic approach to tropicalizations, which explains this in more detail.

6.1 Definition
The *(affine) torus* T_K^n over a field K is the n-dimensional affine space without the coordinate hyperplanes, i.e.

$$T_K^n = \{p \in \mathbb{A}_K^n \mid p_i \neq 0 \text{ for all } i \in \{1, ..., n\}\}.$$

We can restrict any algebraic set to T_K^n to get the vanishing set in the torus:

$$\begin{aligned} V_{T_K^n}(f) &= \{\, p \in T_K^n \mid f(p) = 0\}, \\ V_{T_K^n}(I) &= \bigcap_{f \in I} V_{T_K^n}(f), \end{aligned}$$

where $f \in \mathbb{Q}[\underline{x}]$ and $I \subset \mathbb{Q}[\underline{x}]$ is an ideal.

Note that this notion of vanishing set is also well-defined if we allow ideals in the ring of Laurent polynomials. This could be used to simplify some parts of the theory as mentioned in remark 5.10.

6.2 Remark
Let $f \in K[\underline{x}]$ be some polynomial. Instead of taking the vanishing set over K - which will lead to a subset of $\mathbb{A}_K^n$, we may as well take the vanishing set over any extension field $L \supseteq K$ to get an algebraic set in $\mathbb{A}_L^n$.

$$V_{\mathbb{A}_L^n}(f) = \{w \in \mathbb{A}_L^n \mid f(w) = 0\}.$$

The same holds true for the definition of vanishing sets for ideals and we may as well replace the affine space $\mathbb{A}_L^n$ by the torus T_L^n.

6.3 Definition
Let K be a field. Then we denote by $K\{\{t\}\}$ the field of *Puiseux series* over K. Its elements are formal power series of the form

$$c_1 t^{a_1} + c_2 t^{a_2} + ...$$

where all $c_i \in K$ and $a_1 < a_2 < ...$ is a strictly monotonously increasing sequence of rational numbers such that all a_i share a common denominator.

6.4 Remark
$K\{\{t\}\}$ is an algebraically closed field if K is algebraically closed and $\text{char}(K) = 0$. In particular, $\mathbb{C}\{\{t\}\}$ is algebraically closed. See [MS15, p. 49f] for a proof.

6.5 Definition
We define a valuation on $K\{\{t\}\}$ by sending a Puiseux series to its lowest exponent:

$$\begin{array}{rccl} \text{val} & : & K\{\{t\}\} & \to & \mathbb{Q} \cup \{\infty\} \\ & & c_1 t^{a_1} + c_2 t^{a_2} + \ldots & \mapsto & a_1. \end{array}$$

To get a complete definition, we set

$$\text{val}(0) = \infty$$

by convention. Note that no Puiseux series but 0 gets mapped to ∞.

6.6 Remark
The map val defines a (multiplicative) non-archimedian valuation on $K\{\{t\}\}$. It is surjective, but not injective. For each $f \in K\{\{t\}\}$, the following "splitting" holds:

$$\text{val}(t^{\text{val}(f)}) = \text{val}(f).$$

We can use these definitions to give a purely algebraic definition of tropicalizations.

6.7 Definition
Let $I \subseteq \mathbb{Q}[\underline{x}]$ be an ideal. Since $\mathbb{C}\{\{t\}\}$ is an extension field of $\mathbb{Q}$, we can take the vanishing set in $T^n_{\mathbb{C}\{\{t\}\}}$ as explained in remark 6.2. Afterwards, we can apply the valuation map component-wise to get the set

$$\text{val}(V_{T^n_{\mathbb{C}\{\{t\}\}}}(I)) \subseteq \mathbb{Q}^n.$$

We define the tropicalization of I as the closure in the euclidean topology on $\mathbb{R}^n$ of the set above reflected across the origin:

$$\text{Trop}\,(I) = \overline{-\text{val}(V_{T^n_{\mathbb{C}\{\{t\}\}}}(I))} \subseteq \mathbb{R}^n.$$

6.8 Remark
Note that we need to restrict ourselves to the torus $T^n_{\mathbb{C}\{\{t\}\}}$ if we want $\text{val}(V_{T^n_{\mathbb{C}\{\{t\}\}}}(I))$ to be a subset of $\mathbb{Q}^n$, since $val(0) = -\infty$. We will see later that this restriction to the torus also happens implicitly in the other two definitions of tropicalizations.

6.9 Example
Let $f = \sum_{n\in\mathbb{N}} t^n \in \mathbb{C}\{\{t\}\}$. We have $(f, -f) \in V_{T^2_{\mathbb{C}\{\{t\}\}}}(x + y)$. After applying the valuation map component-wise, we get that $-(0,0) = (0,0) = (\text{val}(f), \text{val}(-f)) \in \text{Trop}\left(\langle\, x + y \,\rangle\right)$, which coincides with the results from example 5.12

7. Structural results about tropical varieties

In this section, we will cite the two most important results about tropical varieties. The first one called *fundamental theorem* states that the three ways to define tropicalizations presented in this part are equivalent.
As mentioned before, tropical varieties can be studied using methods from combinatorics and polyhedral theory because of their polyhedral structure. Before we can state the *structure theorem* which makes this precise, we will need to introduce some basic notions from polyhedral theory. We will also be in dire need of those when we define the boundary via projection in the last part of this thesis.

7.1 Theorem (Fundamental theorem)
Let $I \subseteq \mathbb{Q}[\underline{x}]$ be some ideal. The following three sets are equal:

a) The set $\mathrm{Trop}(I)$ as in definition 4.6 using geometry over the tropical semiring.

b) The set $\mathrm{Trop}(I)$ as in definition 5.5 using initial ideals.

c) The set $\mathrm{Trop}(I)$ as in definition 6.7 using a valuation on the field of Puiseux series.

Proof:
We have shown a part of the statement in proposition 5.11. See [MS15, p. 99ff] for a full proof in a more general setting. □

7.2 Definition
We call any $\mathrm{Trop}(I)$ for $I \subseteq \mathbb{Q}[\underline{x}]$ homogeneous and prime a *tropical variety*.

7.3 Remark
Parts of the literature define tropicalizations not in terms of ideals, but in terms of their vanishing sets. This is justified since $\mathrm{Trop}(I) = \mathrm{Trop}\left(\sqrt{I}\right)$, so two ideals defining the same vanishing set have the same tropicalization, see [JMM08, p. 8]. We will see that the converse does not hold, i.e. two different ideals may have the same tropicalization.

As mentioned before, taking the tropicalization implies a restriction to the torus. To make this precise, we will need an algebraic concept which corresponds to this restriction.

7.4 Definition
Let $I \subseteq K[\underline{x}]$ be an ideal and $g \in K[\underline{x}]$ a polynomial. We define the *ideal quotient* of I with respect to g to be the ideal

$$I : g = \{f \in K[\underline{x}] \mid fg \in I\}$$

and the *saturation* of I with respect to g to be the ideal

$$I : g^\infty = \{f \in K[\underline{x}] \mid \exists n \in \mathbb{N} : fg^n \in I\}.$$

We call I to be *saturated* with respect to g if $I = (I : g)$.

7.5 Remark

Note that $I : g$ and $I : g^\infty$ are ideals of $K[\underline{x}]$ with $I \subseteq (I : g) \subseteq (I : g^\infty)$. It makes sense to call I saturated with respect to g if $I = (I : g)$, since in this case, $I = (I : g^\infty)$ follows, see [Mus13, p. 26f] for proofs of these statements. The saturation occurs in a chain of ideal quotients, which gets stationary because the polynomial ring is noetherian, and there are algorithms to compute ideal quotients, see [Bö, p. 77ff].

7.6 Proposition

Let $I \subset \mathbb{Q}[\underline{x}]$ be an ideal and let $I : \underline{x}^\infty$ be its saturation with respect to the product of all variables $\underline{x} = x_1 * ... * x_n$. Their tropicalizations coincide:

$$\mathrm{Trop}\,(I) = \mathrm{Trop}\,(I : \underline{x}^\infty)\,.$$

Proof:

"$\subseteq$"

Assume $w \in \mathrm{Trop}\,(I)$, so by definition 5.5, the initial form $\mathrm{in}_w\,(f)$ of any polynomial $f \in I$ is not a monomial. For any element g of $I : \underline{x}^\infty$, there is an $n \in \mathbb{N}$ such that $\underline{x}^n g \in I$. Note that the multiplication with $\underline{x}^n$ increases the w-degree of any term of f by the constant $n * (w_1 + ... + w_n)$, so the maximal terms of g are the maximal terms of f multiplied with $\underline{x}^n$ and we get

$$\mathrm{in}_w\left(x^n g\right) = \underline{x}^n\,\mathrm{in}_w\,(g)\,.$$

Since we know that $\mathrm{in}_w\,(x^n g)$ is not a monomial, $\mathrm{in}_w\,(g)$ cannot be a monomial and we conclude $w \in \mathrm{Trop}\,(I : \underline{x}^\infty)$.

"$\supseteq$"

We have $I \subseteq (I : \underline{x}^\infty)$, so we can conclude $\mathrm{Trop}\,(I) \supseteq \mathrm{Trop}\,(I : \underline{x}^\infty)$, see [JMM08, p. 8].

□

In the later parts of this thesis, this proposition justifies the restriction to saturated ideals, since the saturation can be computed and it has the same tropicalization.

Before moving on with the theory, we will study some examples.

7.7 Theorem

The Pluecker ideal $I_{r,m}$ for any $r, m \in \mathbb{N}, m \geq r$ is saturated with respect to the product of the variables $\underline{p} = \prod_{I \in \Lambda(r,m)} p_I$.

Proof:
We have to show $(I_{r,m} : \underline{p}) \subseteq I$ since the reverse inclusion does always hold. Suppose $f \in (I : \underline{p})$, that is $\underline{p}f \in I_{r,m}$. Since $I_{r,m}$ is prime, we conclude that either $f \in I_{r,m}$ or $\underline{p} \in I_{r,m}$. Let p_I be any variable of the Pluecker coordinate ring and let $S_{r,m}$ be the Groebner basis for $I_{r,m}$ from theorem 3.26. Since $S_{r,m}$ is a Groebner basis with homogeneous elements of degree two and p_I has degree one, we have $p_I \notin I_{r,m}$. By the primeness of $I_{r,m}$, $\underline{p} = \prod_{I \in \Lambda(r,m)} p_I \notin I_{r,m}$ follows and we are done, since we have shown $f \in I$. □

7.8 Example

a) Let $I = \langle\, x + y \,\rangle \subseteq \mathbb{Q}[x, y]$. There are various ways to show that I is saturated with respect to $x * y$. We may compute the ideal quotient $I : (x * y)$ using algorithms from computer algebra to verify $I = (I : (x * y))$.

b) The ideal $I = \langle\, x^2 + y^2 - w^2, z \,\rangle \subseteq \mathbb{Q}[w, x, y, z]$ is not saturated with respect to the product of the variables. In fact, we have

$$(I : (w * x * y * z)) = (I : (w * x * y * z)^\infty) = \langle\, 1 \,\rangle$$

since for any polynomial $f \in \mathbb{Q}[w, x, y, z]$, we have $w * x * y * z * f \in \langle\, w \,\rangle \subseteq I$.

7.9 Remark
The saturation $I : g^\infty$ corresponds to removing the vanishing set of g from the vanishing set of I. To be more specific,

$$V(I : g^\infty) = \overline{V(I) \setminus V(g)}$$

where the line denotes the closure in the Zariski topology (since the right hand side without the closure may not be an algebraic set).
In the special case of $g = \underline{x} = x_1 * \ldots * x_n$, we have

$$V(I : \underline{x}^\infty) = \overline{V(I) \setminus V(\underline{x})} = \overline{V(I) \setminus \{p \in \mathbb{A}_K^n \mid p_i = 0 \text{ for some } i\}} = \overline{V(I) \cap T_K^n}.$$

Therefore, proposition 7.6 implies that we actually loose all irreducible parts of the vanishing set completely contained in the complement of the torus by tropicalizing it.

This motivates the goal of this thesis, which is to study how to recover the tropicalization of the intersection of $V(I)$ with some coordinate hyperplanes. Especially for the second way to do this, we will need more information about the structure of tropical varieties. To precisely capture this structure, we introduce polyhedral theory.

7.10 Definition
A *polyhedron* in $\mathbb{R}^n$ is a finite intersection of half-spaces.
If P is a polyhedron, there exist some $m \in \mathbb{N}$, a matrix $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ with

$$P = P(A, b) = \{x \in \mathbb{R}^n \mid Ax \geq b\}.$$

We call P *rational* if there are $A \in \mathbb{Q}^{m\times n}$, $b \in \mathbb{Q}^m$ with $P = P(A, b)$.

7.11 Remark
If P_1 and P_2 are polyhedra in $\mathbb{R}^n$, their intersection $P_1 \cap P_2$ is also a polyhedron.
Suppose $P_1 = P(A_1, b_1)$ and $P_2 = P(A_2, b_2)$. The points in $P_1 \cap P_2$ are exactly the points fulfilling the inequalities defining P_1 and P_2, so

$$P_1 \cap P_2 = P\left(\begin{bmatrix} A_1 \\ A_2 \end{bmatrix}, \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}\right).$$

7.12 Definition
A *conical combination* of the elements of some set $C \subseteq \mathbb{R}^n$ is a finite sum

$$y = \sum_{x\in C} \lambda_x x \in \mathbb{R}^n$$

where $\lambda_x \geq 0$ for all $x \in C$ and $\lambda_x = 0$ for all but finitely many.

A subset $C \subseteq \mathbb{R}^n$ is called a *(convex) cone* if C is closed under conical combinations, i.e. any conical combination of elements of C lies in C.

If a subset $C \subseteq \mathbb{R}^n$ is a polyhedron of type $P(A, 0)$, we call it *polyhedral cone*.

Figure 1 shows a schematic representation of a non-polyhedral and a polyhedral cone in $\mathbb{R}^3$. Note that the intersection of the polyhedral cone with any plane parallel to the x-y-plane is always a polyhedron. The intersection of such a plane with the non-polyhedral cone may be a circle.

7.13 Lemma
As the name suggests, any polyhedral cone $C = P(A, 0)$ is a cone.

Proof:
Let $\sum_{x\in C} \lambda_x x$ be some conical combination. Since matrix multiplication is linear and $\lambda_x \geq 0$, we get

$$A\sum_{x\in C} \lambda_x x = \sum_{x\in C} \lambda_x Ax \geq \sum_{x\in C} 0 = 0,$$

so $\sum_{x\in C} \lambda_x x \in P(A, 0) = C$. □

7.14 Remark
If C_1 and C_2 are cones, then any conical combination of the elements of $C_1 \cap C_2$ an be seen as conical combination of elements of $C_1 \supseteq (C_1 \cap C_2)$. By definition, it lies in C_1 and analogously also in C_2. Hence, it lies in $C_1 \cap C_2$. This shows that the intersections of cones are cones.
By taking remark 7.11 into account, the intersections of polyhedral cones are polyhedral cones since $b_1 = 0$ and $b_2 = 0$ in this case.

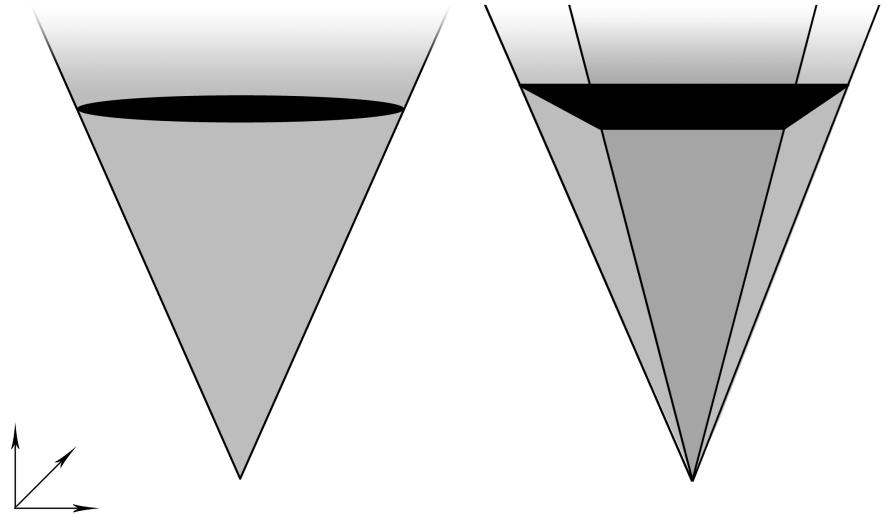

Figure 1: Schematic representation of a non-polyhedral and a polyhedral cone in $\mathbb{R}^3$, where the black area indicates the intersection of the cone with a plane parallel to the x-z-plane.

7.15 Remark

The computer algebra system SINGULAR allows the definition of cones via two methods. One of them is `coneViaInequalities` which takes a matrix $IE \in \mathbb{Z}^{m\times n}$ (inequalities) and a matrix $E \in \mathbb{Z}^{k\times n}$ (equalities) and returns an object representing

$$C = \{x \mid IEx \geq 0, Ex = 0\}.$$

We may rewrite the equalities as inequalities

$$Ex = 0 \Leftrightarrow Ex \leq 0 \text{ and } Ex \geq 0 \Leftrightarrow -Ex \geq 0 \text{ and } Ex \geq 0,$$

so C is really a polyhedral cone:

$$C = P\left(\begin{bmatrix} IE \\ E \\ -E \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}\right).$$

Vice versa, the procedures `inequalities` and `equations` are provided to retrieve the matrices IE and E as above from a cone.

Note that even if we are given rational matrices and vectors, we may multiply all entries with their least common multiple to get integer data.

7.16 Example
Let $A = \begin{bmatrix} 1 & -1 \end{bmatrix} \in \mathbb{Z}^{1\times 2}$. We can use it to define the two cones $C_1 = P(A, 0)$ and $C_2 = P(-A, 0)$ in $\mathbb{R}^2$. Their intersection is the cone

$$C_0 = C_1 \cap C_2 = P\left(\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, 0\right).$$

Figure 2 depicts these cones.

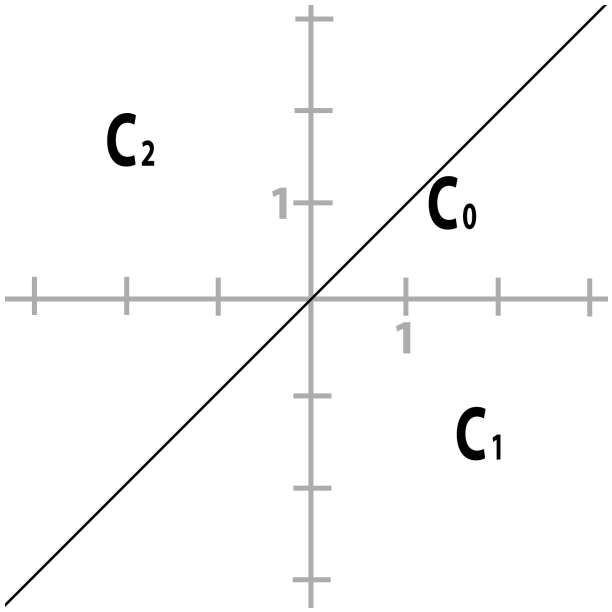

Figure 2: The three cones from example 7.16.

We want to study sets of polyhedra such that their elements are compatible in a certain way. To define whether two polyhedra are compatible, we need to formalize their borders.

7.17 Definition
Let $P \subseteq \mathbb{R}^n$ be some polyhedron. Some $w \in \mathbb{R}^n$ and $t \in \mathbb{R}$ define a *valid inequality* for P if all points of P satisfy $w^T x \geq t$:

$$P = \left\{x \in P \mid w^T x \geq t\right\}.$$

A subset $F \subseteq P$ is called *face* of P if it is the equality set of a valid inequality given by $w \in \mathbb{R}^n, t \in \mathbb{R}$:

$$F = \left\{x \in P \mid w^T x = t\right\} \subseteq P = \left\{x \in P \mid w^T x \geq t\right\}.$$

7.18 Remark
The inequalities $0^T x \geq 0$ and $0^T x \geq -1$ are valid for all polyhedra P. The first one is always fulfilled with equality and defines the face P. The second one is never fulfilled with equality and defines the face $\emptyset$.

We call P and $\emptyset$ the *trivial faces* of P. We are mostly interested in the non-trivial faces, called *proper faces*.

7.19 Lemma
A face of a polyhedron is a polyhedron.

Proof:
Let $P(A,b) \subseteq \mathbb{R}^n$ be some polyhedron and $w^T x \geq t$ be some valid inequality.
The face defined by the inequality is the polyhedron

$$\left\{x \in P(A,b) \mid w^T x = t\right\} = \left\{x \in P(A,b) \mid w^T x \geq t, -w^T x \geq -t\right\} = P\left(\begin{bmatrix} A \\ w \\ -w \end{bmatrix}, \begin{bmatrix} b \\ t \\ -t \end{bmatrix}\right).$$

□

To study the faces of cones, we will first need an alternative way to define them.

7.20 Remark
Let $P(A,b) \subseteq \mathbb{R}^n$ be a polyhedron for some $A \in \mathbb{R}^{r \times n}, b \in \mathbb{R}^r$. We may choose a set of row-indices $I \subseteq \{1,...,r\}$ to define the polyhedron P' we get by enforcing equality in the inequalities given by rows with index in I. By splitting the equalities into two inequalities, we can write it as

$$P' = P\left(\begin{bmatrix} A_{\{1,...,m\}\setminus I} \\ A_I \\ -A_I \end{bmatrix}, \begin{bmatrix} b_{\{1,...,m\}\setminus I} \\ b_I \\ -b_I \end{bmatrix}\right)$$

where A_I is the submatrix we get by only keeping the rows with indices in I. Any polyhedron of this form is a face of P, since it is given by the valid inequality $w^T x \geq t$ with

$$w^T = \sum_{i \in I} A_i, \qquad t = \sum_{i \in I} b_i$$

where A_i denotes the i^{th} row of A.
The inequality is valid since it is the sum of the valid inequalities $A_i x \geq b_i$ occurring in the definition of the original polyhedron P. The points in P' certainly fulfill it with equality since they fulfill each inequality $A_i x \geq b_i$ with equality by the definition of P'. Any point $x' \in P \setminus P'$ violates at least one equality, i.e. there is some $j \in I$ with $A_j x' > b_j$. Since we have $A_i x \geq b_i$ for all points in P and for all i, we get

$$w^T x' = \left(\sum_{i \in I} A_i\right) x' = \left(\sum_{i \in I, i \neq j} A_i\right) x' + A_j x' \overset{x' \in P}{\geq} \sum_{i \in I} b_i + A_j x' \overset{x' \notin P'}{>} \sum_{i \in I} b_i + b_j = \sum_{i \in I} b_i = t,$$

so x' is not in the equality set of $w^T x \geq t$.

7.21 Theorem

Any proper face of a polyhedron $P(A, b)$ can be obtained by replacing some of the inequalities $Ax \geq b$ by equalities as in remark 7.20 and any set obtained in this way is a face of $P(A, b)$.

In particular, any polyhedron has only finitely many faces and the faces of a rational polyhedron are rational polyhedra.

Proof:

We have already proven a part of the theorem in remark 7.20.

For a full proof, see [Kru, p. 19f]. □

7.22 Corollary

The intersection of two faces $F_1, F_2 \subseteq P(A, b)$ is a face of $P(A, b)$.

Proof:

By theorem 7.21, F_1 is given by some equality set $I_1 \subseteq \{1, ..., r\}$ and analogously F_2 by I_2. Their intersection is defined by the equality set $I_1 \cup I_2$. □

7.23 Corollary

Let $F_1 \subseteq P$ be a face. The faces of F_1 are exactly the faces of P contained in F_1.

Proof:

Let F_2 be a face of F_1. By theorem 7.21, F_1 is given by some equality set $I_1 \subseteq \{1, ..., r\}$ and F_2 is given by some equality set $I_2 \subseteq \{1, ..., r\} \setminus I_1$. The union $I_1 \cup I_2$ defines the face $F_1 \cap F_2 = F_2$ of P.

Assume F_2 is some face of P contained in F_1, then F_2 is given by some valid inequality $w^T x \geq t$ for P, which is also valid for $F_1 \subseteq P$ and thus defines a face of F_1. Since $F_2 \subseteq F_1$, this face is F_2 itself. □

7.24 Corollary

Any face of a polyhedral cone is also a polyhedral cone and it can be defined as the equality set of a valid inequality of the form $w^T x \geq 0$ for some $w^T \in \mathbb{R}^n$.

Proof:

By theorem 7.21, any face F of a polyhedron $P(A, b)$ can be defined by choosing some set I of row indices and then replacing the inequalities in the rows with index in I by equalities. In the case of a polyhedral cone, $b = 0 \in \mathbb{R}^m$, so we get $b_I = -b_I = 0 \in R^{|I|}$ in the definition of the face as in remark 7.20. If we construct a valid inequality $w^T x \geq t$ defining this face as the remark, we get

$$t = \sum_{i \in I} b_i = \sum_{i \in I} 0 = 0,$$

so we have found a defining inequality of the desired form. □

7.25 Definition
A *polyhedral complex* $\mathcal{P}$ in $\mathbb{R}^n$ is a finite set of polyhedra in $\mathbb{R}^n$ such that the following two properties hold:

- For any polyhedron in $\mathcal{P}$, all its proper faces are also in $\mathcal{P}$.
- The intersection $P \cap Q$ of two polyhedra P, Q in $\mathcal{P}$ is a face of both.

We call the union of all polyhedra in $\mathcal{P}$ its *support*:

$$\text{Supp}\,(\mathcal{P}) = \bigcup_{P \in \mathcal{P}} P \subseteq \mathbb{R}^n.$$

If all polyhedra contained in a polyhedral complex are polyhedral cones, we call it a *(polyhedral) fan*.

7.26 Definition
Let $P \subseteq \mathbb{R}^n$ be a polyhedron (respectively $\mathcal{P}$ a set of polyhedra). Its *lineality space* is the largest affine subspace of $\mathbb{R}^n$ contained in P (respectively contained in all $P \in \mathcal{P}$), and its dimension is called *lineality dimension*.
The dimension of P (respectively $\mathcal{P}$) is the dimension of the smallest affine subspace of $\mathbb{R}^n$ such that P (respectively all $P \in \mathcal{P}$) are contained in it. This subspace is called the *affine hull* of P (respectively $\mathcal{P}$).

Since the lineality space is contained in the affine hull, we get

$$\text{lineality dimension} \leq \text{ dimension } \leq \text{ ambient dimension} = \dim_{\mathbb{R}}(\mathbb{R}^n) = n.$$

7.27 Remark
If $C \subseteq \mathbb{R}^n$ is a polyhedral cone, then lineality space as well as affine hull are actually linear subspaces of $\mathbb{R}^n$.

7.28 Definition
Let $\mathcal{P}$ be a polyhedral complex and $P \in \mathcal{P}$. P is called *maximal* if it is not contained in any other polyhedron in $\mathcal{P}$. A polyhedral complex is called *pure* if all maximal polyhedra have the same dimension.

By corollary 7.24, it is sufficient to show that the maximal polyhedra of a polyhedral complex are cones, to conclude that the polyhedral complex is a fan.

7.29 Example
We continue example 7.16. The cone C_1 has three faces: itself, C_0 and the empty-set. In particular, C_0 is the only proper face of C_1 and it is obtained by replacing the inequality $1 * x_1 - 1 * x_2 \geq 0$ by an equality. Analogously, C_0 is also the only proper face of C_2.
Therefore, the set $\mathcal{F} = \{C_0, C_1, C_2\}$ is a fan, a polyhedral complex consisting of cones.

We have now gathered all prerequisites to describe the structure of tropicalizations in more detail.

7.30 Lemma
Let $I \subseteq \mathbb{Q}[\underline{x}]$ be a homogeneous ideal and $>$ a global monomial ordering. Then $C_>(I)$ is a polyhedral cone.

Proof:
In [Jen07, p. 28], it is shown that $C_>(I)$ is a polyhedral cone if it contains a strictly positive vector. We will now show $h = (1, ..., 1) \in C_>(I)$. Proposition 5.20 states that it is sufficient to show

$$\text{for all } g \in G_> : \mathrm{LM}_>(\mathrm{in}_h(g)) = \mathrm{LM}_>(g),$$

and we have stated in theorem 5.18 that $G_>$ consists only of homogeneous elements. For those elements, we have $\mathrm{in}_h(g) = g$, so the condition is trivially fulfilled. □

7.31 Definition
Let $I \subseteq \mathbb{Q}[\underline{x}]$ be a homogeneous ideal. The *Groebner fan* $GF(I)$ of I is the set containing all non-empty faces of the cones $C_>(I)$ where $>$ is a global monomial ordering, i.e.

$$GF(I) = \{F \mid F \text{ face of some } C_>(I), > \text{ global monomial ordering}, F \neq \emptyset\}.$$

Note that lemma 7.30 does not hold for non-homogeneous ideals and therefore, the definition of the Groebner fan for non-homogeneous ideals has to impose a restriction on the sets $C_>(I)$.

7.32 Theorem
Let $I \subseteq \mathbb{Q}[\underline{x}]$ be a homogeneous ideal. As the name suggests, its Groebner fan $GF(I)$ is a fan.

Proof:
See [Jen07, p. 35]. □

7.33 Example
We continue the examples 7.16 and 6.9 by bringing them together. Let $I = \langle x + y \rangle \subseteq \mathbb{Q}[x, y]$. There are only two different Groebner cones for I. Let $>_x$ be the lexicographic ordering with respect to the variable order $x > y > 1$ respectively $>_y$ the ordering for $y > x > 1$. We have $C_1 = C_{>_x}(I)$ and $C_2 = C_{>_y}(I)$. In fact, the fan $\mathcal{F} = \{C_0, C_1, C_2\}$ depicted in figure 2 is the Groebner fan of I.

7.34 Theorem (Structure theorem part 1)
Let $I \subseteq \mathbb{Q}[\underline{x}]$ be some homogeneous ideal. Then $\mathrm{Trop}(I) = \mathrm{Supp}(\mathcal{F})$ where $\mathcal{F}$ is the following polyhedral fan:

$$\mathcal{F} = \{C_w(I) \mid w \in \mathbb{R}^n \text{ such that } \mathrm{in}_w(I) \text{ is monomial-free}\}.$$

In particular, $\mathcal{F}$ is a subfan of $GF(I)$ consisting of the cones of $GF(I)$ such that $\text{in}_w(I)$ is monomial-free for the weight vectors in their relative interior.

Proof:
See [Jen07, p. 67]. □

7.35 Remark
In rest of this thesis, we will identify $\text{Trop}(I)$ with the fan above which has $\text{Trop}(I)$ as its support, if $I \subseteq \mathbb{Q}[\underline{x}]$ is homogeneous.
Note that for non-homogeneous ideals, the sets $C_w(I)$ are not convex and in particular no polyhedral cones. Examples for this case and ways to extend the theory to non-homogeneous ideals are studied in [Jen07]
The algorithms to compute $\text{Trop}(I)$ briefly mentioned in remark 5.21 do naturally return $\text{Trop}(I)$ as a fan and SINGULAR provides an implementation via the procedure `tropicalVariety`.

7.36 Example
We continue example 7.33. We already noted that $\text{Trop}(I) = \{(w_1, w_1) \in \mathbb{R}^2\}$. In fact, $\text{Trop}(I)$ is the support of the fan consisting of the single cone C_0, which is a subfan of the Groebner fan computed in the previous example. Note that C_0 has itself as linearity space and as affine hull, so dimension and lineality dimension are 1.

7.37 Theorem (Structure theorem part 2, Theorem of Bieri-Groves)
Let $I \subseteq \mathbb{Q}[\underline{x}]$ be some homogeneous monomial-free prime ideal of dimension d. Then $\text{Trop}(I)$ is a pure polyhedral fan of dimension d.

Proof:
See [MS15, p. 108ff] for the proofs of several theorems in more general settings which imply the statement above. □

7.38 Example
Let $f^h \in \mathbb{Q}[w, x, y]$ be the homogenization of the polynomial $f = x^2 + y^2 - 1 \in \mathbb{Q}[x, y]$ defining the circle with radius 1, i.e.

$$f^h = x^2 + y^2 - w^2 \in \mathbb{Q}[w, x, y].$$

The principal ideal $I = \langle f^h \rangle$ has the tropicalization

$$\text{Trop}(I) = \{C_0, C_1, C_2, C_3\}$$

where

$$C_1 = P\left(\begin{bmatrix} -1 & 0 & 1 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix}, 0\right),$$

$$C_2 = P\left(\begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 0 & 1 \end{bmatrix}, 0\right),$$

$$C_3 = P\left(\begin{bmatrix} 0 & 1 & -1 \\ 1 & -1 & 0 \\ -1 & 1 & 0 \end{bmatrix}, 0\right)$$

and

$$C_0 = C_1 \cap C_2 \cap C_3 = C_1 \cap C_2 = C_1 \cap C_3 = C_2 \cap C_3 = P\left(\begin{bmatrix} 1 & 0 & -1 \\ -1 & 0 & 1 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix}, 0\right).$$

Its lineality space is the one-dimensional space with $(1,1,1)^T$ as basis vector. The weight vectors in the relative interior of each cone define the same initial ideal, as indicated in the following table.

cone	example point w	$\text{in}_w\left(f^h\right)$	$\text{in}_w(I)$
C_0	$(\ 0\ ,\ 0\ ,\ 0\ )$	f^h	I
C_1	$(\ -2\ ,\ 1\ ,\ 1\ )$	x^2+y^2	$\langle\ x^2+y^2\ \rangle$
C_2	$(\ 1\ ,\ -2\ ,\ 1\ )$	w^2+y^2	$\langle\ w^2+y^2\ \rangle$
C_3	$(\ 1\ ,\ 1\ ,\ -2\ )$	w^2+x^2	$\langle\ w^2+x^2\ \rangle$

7.39 Example

Let $r = 2$, $m = 4$. We have computed in example 3.18 that

$$I_{2,4} = \langle\ f\ \rangle = \langle\ p_{\{1,4\}} * p_{\{2,3\}} - p_{\{1,3\}} * p_{\{2,4\}} + p_{\{1,2\}} * p_{\{3,4\}}\ \rangle$$

Similar to the example above, we have $\text{Trop}\,(I_{2,4}) = \{C_0, C_1, C_2, C_3\}$ where C_1, C_2 and C_3 are 5-dimensional cones in $\mathbb{R}^6$ and in the relative interior of each of those, the initial form of f consists of two of the quadric terms of f. Their intersection C_0 is a 4-dimensional cone and the initial form of f with respect to any weight vector in C_0 is f itself. $\text{Trop}\,(I_{2,4})$ has a 4-dimensional lineality spaces given as the row space of the matrix

$$E = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & -1 \\ 0 & 0 & 0 & -1 & -1 & -1 \end{bmatrix}.$$

Part IV.
Computing the boundary

As we have seen in proposition 7.6, tropicalizing a variety always implies a restriction to the torus where no component is zero. The goal of the next two sections will be to study how we can find a fan representing the part of the tropical variety living in the boundary.

8. Computing the boundary via elimination

Firstly, we will study a method purely relying on tropicalizing the vanishing set of a a modified ideal, which represents the projection onto the part normally excluded by the restriction to the torus.

8.1 Remark
Given a tropicalization Trop(I) of some homogeneous ideal $I \subseteq \mathbb{Q}[\underline{x}]$, we may as well view $V(I)$ as its vanishing set in the projective space $\mathbb{P}^{n-1}_{\mathbb{C}}$.
In projective space, the points which lie in the hyperplane H_i at infinity in the i^{th}-direction are points $\langle v \rangle$ with $v_i = 0$, i.e.

$$H_i = \{v \in \mathbb{P}^{n-1}_K \mid v_i = 0\}$$

Therefore, if we want to study the boundary of a tropicalization Trop(I) in the i^{th} direction, we may intersect $V(I)$ with H_i and tropicalize the ideal corresponding to this algebraic set.

Since $H_i = V(\langle x_i \rangle)$ and the intersection of vanishing sets corresponds to the ideal generated by the union of ideals, we have to look at the ideal $\langle\ I, x_i\ \rangle$ we get by adding x_i as a generator tor I. Note that if I was homogeneous, $\langle\ I, x_i\ \rangle$ is still homogeneous.

Unfortunately, this approach yields a problem: The set $X \cap H_i$ is completely contained in the complement of the torus. On the algebraic side, any initial ideal $\text{in}_w\left(\langle\ I, x_i\ \rangle\right)$ is never monomial-free since the monomial $\text{in}_w(x_i) = x_i$ will surely be contained in it.

To solve this, we can project the i^{th} component away (since it is zero anyway) and tropicalize the resulting subset of $\mathbb{P}^{n-2}_K$.

8.2 Definition
Let π_i for some $i \in \{1, ..., n\}$ be the projection which removes the i^{th} component, i.e.

$$\begin{array}{rccc} \pi_i: & \mathbb{P}^{n-1}_K \backslash \{\langle e_i \rangle\} & \to & \mathbb{P}^{n-2}_K \\ & (p_1, ..., p_n) & \mapsto & (p_1, ..., p_{i-1}, p_{i+1}, ..., p_n). \end{array}$$

Note that we need to exclude the point $\langle e_i \rangle$ generated by the i^{th} unit vector, since its image $(0, 0, ..., 0)$ is not generating an one-dimensional subspace and thus not contained in $\mathbb{P}_K^{n-2}$. For any other point p, the map is well-defined: If $q = \lambda p \in K^n$ for some $\lambda \in K^*$, we have

$$\pi_i(q) = (\lambda p_1, ..., \lambda p_{i-1}, \lambda p_{i+1}, ..., \lambda p_n) = \lambda(p_1, ..., p_{i-1}, p_{i+1}, ..., p_n) = \lambda \pi_i(p)$$

and those two vectors define the same point in $\mathbb{P}_K^{n-2}$.

To see a concept on the algebraic side which is equivalent to projection, we need to study elimination theory briefly.

8.3 Definition
Let $I \subseteq K[\underline{x}]$ be an ideal and $i \in \{1, ..., n\}$. Let

$$K[x_j \mid j \neq i\,] = K[x_1, ..., x_{i-1}, x_{i+1}, ..., x_n]$$

be the subring of $K[\underline{x}]$ not containing the i^{th} variable. We call the ideal

$$I_i^e = I \cap K[x_j \mid j \neq i\,] \subseteq K[x_j \mid j \neq i\,]$$

the *elimination ideal* of I (with respect to i).

8.4 Lemma
For an ideal $I \subseteq K[\underline{x}]$, its elimination ideal I_i^e is an ideal of $K[x_j \mid j \neq i\,]$.

Proof:
Since $0 \in I_i^e$, the elimination ideal is non-empty.
Any expression $af + bg$ with $f, g \in I_i \subseteq I$ and $a, b \in K[x_j \mid j \neq i\,] \subseteq K[\underline{x}]$ can be seen as expression in $K[\underline{x}]$. This shows $af + bg \in I$ since I was an ideal. Since all terms of this expression are in $K[x_j \mid j \neq i\,]$, we have $af + bg \in K[x_j \mid j \neq i\,]$ and thus $af + bg \in K[x_j \mid j \neq i\,] \cap I = I_i^e$. □

8.5 Example
In computer algebra systems like SINGULAR, ideals are given as a finite site of generators. Let

$$f = p_{\{1,4\}}p_{\{2,3\}} - p_{\{1,3\}}p_{\{2,4\}} + p_{\{1,2\}}p_{\{3,4\}} \in K[\underline{p}]$$

be the generator of the principal ideal $I_{r,m}$ for $r = 2, m = 4$ as computed in example 3.18. Let $J = \langle\ I_{2,4},\ p_{\{1,2\}}\ \rangle$ be the ideal we get by adding the generator $p_{\{1,2\}}$. If we intersect the set of generators with the subring not containing the variable $p_{\{1,2\}}$, we would end up with the empty set. But one can show that $p_{\{1,4\}}p_{\{2,3\}} - p_{\{1,3\}}p_{\{2,4\}} \in J \cap \mathbb{Q}[p_I \mid I \neq \{1, 2\}]$, so the ideal intersected with the subring is not the zero ideal $0 = \langle \emptyset \rangle$.

As the example above shows, in general, it is not sufficient to intersect a set of generators, i.e.

$$\langle G \rangle \cap K[x_j \mid j \neq i\,] \neq \langle G \cap K[x_j \mid j \neq i\,]\,\rangle$$

for some $G \subseteq K[\underline{x}]$, even when G is a Groebner basis for $\langle G \rangle$.

We will now define monomial orderings such that the procedure indicated above works for their corresponding Groebner bases.

8.6 Definition
A monomial ordering is called *elimination order* for x_i if we can decide subring membership by looking at the leading monomial, i.e.

$$\mathrm{LM}(f) \in K[x_j \mid j \neq i\,] \Rightarrow f \in K[x_j \mid j \neq i\,]$$

for all $f \in K[\underline{x}]$.

8.7 Definition
Let $x_{i_1} > ... > x_{i_n} > 1$ be some ordering of the variables with $\{i_1, ..., i_n\} = \{1, ..., n\}$. The *lexicographic ordering* $>_{lp}$ with respect to this ordering is the global monomial ordering defined by

$$\underline{x}^{\alpha} >_{lp} \underline{x}^{\beta} :\Leftrightarrow \alpha_{i_k} = \beta_{i_k} \text{ for } k = 1, ..., r-1 \text{ and}$$
$$\alpha_{i_r} > \beta_{i_r}.$$

8.8 Lemma
The lexicographic ordering with respect to the variable ordering

$$x_i > x_1 > x_2 > ... > x_{i-1} > x_{i+1} > ... > x_n > 1$$

is an elimination ordering for x_i.

Proof:
By the definition of $>_{lp}$, a monomial $\underline{x}^{\alpha}$ with $\alpha_i > 0$ is always bigger than any monomial $\underline{x}^{\beta}$ with $\beta_i = 0$. Therefore, x_i is a factor in the leading monomial if and only if it occurs in the polynomial at all. □

8.9 Theorem
Let G be a Groebner basis for $\langle G \rangle$ with respect to an elimination ordering for x_i. Then

$$\langle G \rangle \cap K[x_j \mid j \neq i\,] = \langle G \cap K[x_j \mid j \neq i\,]\,\rangle.$$

Proof:
This is a special case of [Bö, p. 61f]. In fact $G \cap K[x_j \mid j \neq i\,]$ is even a Groebner basis with respect to the induced ordering on the subring. □

8.10 Example
We continue example 8.10. We have seen that the generators of

$$J = \langle\, p_{\{1,2\}},\ p_{\{1,4\}}p_{\{2,3\}} - p_{\{1,3\}}p_{\{2,4\}} \,\rangle$$

are not well-behaved with respect to elimination. In fact, this set is not a Groebner basis with respect to $>_{lp}$ (for the variable ordering which orders the variables p_I by the sets I as in remark 3.4.) One can compute that the set of generators

$$\left\{p_{\{1,2\}},\ p_{\{1,4\}}p_{\{2,3\}} - p_{\{1,3\}}p_{\{2,4\}}\right\}$$

is a $>_{lp}$-Groebner basis for J, and thus

$$J \cap \mathbb{Q}[\, p_I \mid I \neq \{1,2\}] = \langle\, p_{\{1,4\}}p_{\{2,3\}} - p_{\{1,3\}}p_{\{2,4\}} \,\rangle.$$

8.11 Corollary
If $I \subseteq K[\underline{x}]$ is a homogeneous ideal, I_i^e is a homogeneous ideal of $K[x_j \mid j \neq i\,]$.

Proof:
Let G be the reduced Groebner basis for I with respect to the elimination orderings from lemma 8.8. By theorem 5.18, it consists of homogeneous elements. Its intersection with the subring is a set of homogeneous polynomials generating I_i^e by theorem 8.9. □

8.12 Theorem
Let $I \subseteq \mathbb{Q}[\underline{x}]$ be an ideal and I_i^e its elimination ideal as above. The vanishing set of I_i^e in $\mathbb{P}^{n-2}_{\mathbb{C}}$ is the closure of the projection of $V(I)$ in the Zariski-topolgy:

$$\overline{\pi_i(V(I))} = V(I_i^e) \subseteq \mathbb{P}^{n-2}_{\mathbb{C}}.$$

Proof:
See [Bö, p. 64f] for a proof of the affine case in a slightly more general setting. By swapping the i^{th} variable to the first position and by using the projective Nullstellensatz together with corollary 8.11, the proof for the projective case follows. □

By this theorem, we have found an ideal that corresponds to the projection of the vanishing set and can use it to define the boundary via elimination.

8.13 Definition
Let $I \subseteq \mathbb{Q}[\underline{x}]$ be a homogeneous ideal and let $i \in \{1, ..., n\}$. The *boundary via elimination* in the i^{th} direction is the tropicalization

$$\text{Bound}^{\text{e}}{}_i\,(I) = \text{Trop}\left(\langle\, I, x_i \,\rangle_i^e\right).$$

An algorithm to compute it follows immediately by theorem 8.9.

8.14 Algorithm

Input:

$I \subseteq \mathbb{Q}[\underline{x}]$, an ideal given via homogeneous generators $f_1, ..., f_r \in \mathbb{Q}[\underline{x}]$,
$i \in \{1, ..., n\}$

Output:

Bound$^e{}_i(I)$

1 $H := \{f_1, ..., f_r\}$
2 $H := H \cup \{x_i\}$
3 $G :=$ Groebner basis for $\langle H \rangle$ with respect to $>_{lp}$ for the variable ordering
$$x_i > x_1 > ... > x_{i-1} > x_{i+1} > ... > x_n > 1$$
4 $G_i := G \cap K[x_j \mid j \neq i\,]$
5 $E := \langle G_i \rangle$
6 **return** Trop(E)

Proof of termination:

Starting from a finite set of generators, a Groebner basis with respect to a global ordering like $>_{lp}$ can be computed in finitely many steps, for example with the Buchberger algorithm. As stated in remark 7.35, there are algorithms to compute tropicalizations in finite time. □

Proof of soundness:

We have shown in lemma 8.8 that $>_{lp}$ with respect to the variable ordering in the algorithm is an elimination ordering for x_i and we have shown in theorem 8.9 that it is sufficient to intersect a Groebner basis with respect to such an elimination ordering with the subring to get a set of generators for the elimination ideal. □

8.15 Remark

Computing with elimination orders is very expensive in terms of time consumption, since they do not respect the total degree of polynomials. Even though this is less of a problem for homogeneous polynomials, it may still be faster to start by computing a Groebner basis with respect to $>_{dp}$ and then use the information (like the Hilbert polynomial) obtainable by this Groebner basis to speed up the elimination process.

8.16 Example

Let $I = \langle\, x^2 + y^2 - w^2, z \,\rangle \subseteq \mathbb{Q}[w, x, y, z]$ be the non-saturated ideal from example 7.8. Note that it represents a circle in $\mathbb{P}^3_{\mathbb{C}}$ embedded in the w-x-y-plane where $z = 0$. In particular, it is contained in the complement of the torus and we have computed in example 7.8 that $\mathrm{Trop}(I) = \mathrm{Trop}(I : \underline{x}^\infty) = \mathrm{Trop}\left(\langle 1 \rangle\right) = \emptyset$.
The generators are already a Groebner basis with respect to $>_{lp}$, so we get that

$$I \cap \mathbb{Q}[w, x, y] = \langle\, x^2 + y^2 - w^2 \,\rangle.$$

We conclude that $\mathrm{Bound}^{\mathrm{e}}{}_4(I) = \mathrm{Trop}\left(\langle\, x^2 + y^2 - w^2 \,\rangle\right)$, which we have computed in example 7.38. We were able to recover the tropicalization of the circle by eliminating the z-component. Note that $\mathrm{Bound}^{\mathrm{e}}{}_4\left(\langle 1 \rangle\right) = \emptyset$, so the boundary depends on the ideal and two ideals with the same tropicalization may have different boundaries.

8.17 Example

In example 8.10, we have computed that

$$p_{\{1,4\}}p_{\{2,3\}} - p_{\{1,3\}}p_{\{2,4\}}$$

is a generator for $I_{2,4} \cap \mathbb{Q}[\, p_I \mid I \neq \{1, 2\}]$. We can use this to compute $\mathrm{Bound}^{\mathrm{e}}{}_1(I_{2,4}) = \{C\}$ where

$$C = P\left(\begin{bmatrix} -1 & 1 & 1 & -1 & 0 \\ 1 & -1 & -1 & 1 & 0 \end{bmatrix}, 0\right) \subseteq \mathbb{R}^4.$$

9. Computing the boundary via projection

Instead of altering the ideal, we can also compute the boundary starting from its tropicalization. As stated in theorem 7.34 respectively remark 7.35, tropical varieties of homogeneous ideals are (polyhedral) fans, so we can use the fan structure to define the boundary. Since we used a negative sign in the definition of tropicalizations, we are actually interested in cones that remain if we restrict the i^{th} component to "$-\infty$", i.e. the cones that contain $-\lambda e_i$ for any $\lambda \in \mathbb{R}_{\geq 0}$.

9.1 Definition
Let p_i for some $i \in \{1, ..., n\}$ be the projection which removes the i^{th} component, i.e.

$$\begin{array}{rccc} p_i: & \mathbb{R}^n & \to & \mathbb{R}^{n-1} \\ & (a_1, ..., a_n) & \mapsto & (a_0, ..., a_{i-1}, a_{i+1}, ..., a_n) \end{array}$$

Let $I \subseteq \mathbb{Q}[\underline{x}]$ be a homogeneus ideal and let $i \in \{1, ..., n\}$. The *boundary via projection* in the i^{th}-direction is the set

$$\text{Bound}^{p}{}_i(I) = \{\, p_i(C) \mid -\lambda e_i \in C \text{ for all } \lambda \in \mathbb{R}_{\geq 0}, C \in \text{Trop}(I)\}.$$

Here, e_i denotes the i^{th} unit vector of $\mathbb{R}^n$.

Note that in contrast to the boundary via elimination, this definition is only dependent on Trop(I), but it leads to many questions concerning the structure of Bound$^{p}{}_i(I)$. It is unclear whether $p_i(C)$ is a cone and one can show that the projection of a fan is not a fan in general, so we have to show that the restriction $-e_1 \in C$ is sufficient to guarantee that Bound$^{p}{}_i(I)$ is a fan. Furthermore, we need algorithms to compute the projection and to check the condition $-\lambda e_i \in C$ for all $\lambda \in \mathbb{R}_{\geq 0}$. We will address the issues concerning the structure soon. Let us see first that the condition is easy to check.

9.2 Lemma
Let C be a polyhedral cone. Then

$$-e_i \in C \Leftrightarrow -\lambda e_i \in C \text{ for all } \lambda \in \mathbb{R}_{\geq 0}.$$

Proof:

"$\Rightarrow$" By lemma 7.13, $-\lambda e_i$ is contained in C since it is a conical combination of -e_i.

"$\Leftarrow$" Choose $\lambda = 1$.

□

We will need a new way to describe cones to see that $p_i(C)$ is always a polyhedral cone easily.

9.3 Definition
Let $S \subseteq \mathbb{R}^n$ be a finite set of vectors. The *cone generated by S* is the cone defined via

$$\text{cone}(S) = \left\{ \sum_{x \in S} \lambda_x x \mid \lambda_x \in \mathbb{R}_{\geq 0} \text{ for all } x \right\}.$$

9.4 Theorem
Any polyhedral cone is of the form cone(S) for some finite set $S \subseteq \mathbb{R}^n$, and all sets of type cone(S) are polyhedral cones.
If the cone is rational, S can be chosen as subset of $\mathbb{Q}^n$ and vice versa, if $S \subseteq \mathbb{Q}^n$, cone(S) is a rational cone.

Proof:
A variant of this statement is called the *Theorem of Weyl-Minkowski-Farkas*, see [Kru, p. 23f] for a proof. □

9.5 Remark
The computer algebra system SINGULAR allows the definition of cones via two methods, one of them mentioned in remark 7.15. The other one is `coneViaPoints` which takes a matrix $HL \in \mathbb{Z}^{m \times n}$ (half-lines) and a matrix $L \in \mathbb{Z}^{k \times n}$ (lines) and returns an object representing

$$C = \left\{ \sum_{x \in \text{rows}(HL)} \lambda_x x + \sum_{y \in \text{rows}(L)} \mu_y y \mid \lambda_x \in \mathbb{R}_{\geq 0}\ \forall x,\ \mu_y \in \mathbb{R}\ \forall y \right\}$$

where rows(A) denotes the set of vectors given by the rows of a matrix A.
We may write

$$C = \text{cone}\left(\text{rows}(HL) \cup \text{rows}(L) \cup -\text{rows}(L)\right)$$

so C is indeed a polyhedral cone.

Vice versa, given any cone, the procedures `rays` and `generatorsOfLinealitySpace` yield matrices containing the half-lines respectively lines which generate the cone as its rows. Since the lineality space is a linear subspace (see remark 7.27), the rows of `generatorsOfLinealitySpace` form a vector space basis for the lineality space.
Note that we can replace any element of S by a positive scalar multiple and still get the same cone(S), so if we start with a set of rational vectors, we may multiply each vector with the least common multiple of its entries to get a set of integer vectors representing the same cone.
We indicated in remark 5.21 that the Groebner bases of an ideal can be used to define the cones occurring in the tropicalization. Since the degree-conditions give inequalities with integer coefficients, all cones in the tropicalization are rational cones.

9.6 Example
Let C_1 be the cone from example 7.39. It can be defined as

$$C_1 = P\left(\begin{bmatrix} -1 & 0 & 1 & 1 & 0 & -1 \\ 0 & -1 & 1 & 1 & -1 & 0 \\ 0 & 1 & -1 & -1 & 1 & 0 \end{bmatrix}, 0\right) \subseteq \mathbb{R}^6$$

Note that $-e_1$ fulfills all the defining inequalities, so by lemma 9.2, we have $-\lambda e_1 \in C_1$ for any $\lambda \in \mathbb{R}_{\geq 0}$. We can represent C_1 as cone(S) with

$$S = \left\{(-2,1,1,1,1,-2)^T\right\} \cup L \cup -L$$

where L is a set of generators of the lineality space of C_1:

$$L = \left\{(-1,-1,-1,0,0,0)^T, (-1,-1,0,-1,0,0)^T, (0,1,0,0,-1,0)^T, (1,0,0,0,0,-1)^T\right\}.$$

9.7 Lemma
Let p_i be a projection as above and let C be some polyhedral cone. $p_i(C)$ is also a polyhedral cone.

Proof:
By using proposition 9.4, we can write C = cone(S) for some finite set of vectors S.
We have

$$\begin{aligned} p_i(C) &= \{p_i(y) \mid y \in C\} = \{p_i(y) \mid y \in \text{cone}(S)\} \\ &= \left\{p_i\left(\textstyle\sum_{x\in S} \lambda_x x\right) \mid \lambda_x \in \mathbb{R}_{\geq 0}\ \forall x\right\} \\ &= \left\{\textstyle\sum_{x\in S} \lambda_x p_i(x) \mid \lambda_x \in \mathbb{R}_{\geq 0}\ \forall x\right\} \\ &= \text{cone}\left(\{p_i(x) \mid x \in S\}\right) = \text{cone}\left(p_i(S)\right) \end{aligned}$$

since p_i is a homomorphism of vector spaces. Using theorem 9.4 again, cone $(p_i(S))$ is a polyhedral cone. □

This shows that Bound$^{p}{}_i(I)$ is a set of cones. To see that it is in fact a fan, we will need to understand the faces of the projection of a cone.

9.8 Example
We continue example 9.6. Using lemma 9.7 we get

$$p_1(C_1) = \text{cone}(p_1(S)) = \text{cone}\left(\left\{(1,1,1,1,-2)^T\right\} \cup p_1(L) \cup -p_1(L)\right).$$

This representation is not minimal, since $(1,1,1,1,-2)$ is contained in the span of the vectors in $p_1(L)$. In fact, $p_1(C_1)$ is a linear subspace of $\mathbb{R}^5$ with $p_1(L)$ as basis:

$$p_1(C_1) = \text{cone}\left(p_1(L) \cup -p_1(L)\right).$$

There are ways to transform this representation to a representation using inequalities.

$$p_1(C_1) = P\left(\begin{bmatrix} 1 & -1 & -1 & 1 & 0 \\ -1 & 1 & 1 & -1 & 0 \end{bmatrix}, 0\right) \subseteq \mathbb{R}^5$$

9.9 Proposition
Let $C \subseteq \mathbb{R}^n$ be some cone and $p_i(C)$ its projection as above.
Any face $\overline{F} \subseteq p_i(C)$ is of the form $\overline{F} = p_i(F)$ for some face F of C.

Proof:
Any face $\overline{F}$ of $p_i(C)$ can be defined as the equality set of some valid inequality $\overline{w}^T\overline{x} \geq 0$ with $\overline{w} \in \mathbb{R}^{n-1}$ by corollary 7.24. Note that

$$p_i(C) = \{p_i(x) \mid x \in C\} = \left\{\overline{x} \in \mathbb{R}^{n-1} \mid \exists x \in \mathbb{R}^n : \overline{x} = p_i(x), x \in C\right\}.$$

Define $w \in \mathbb{R}^n$ by $p_i(w) = \overline{w}$ and $w_i = 0$.
By the construction of the projection $p_i(C)$, $w^T x \geq 0$ is a valid inequality for C since if $x \in C$ violates $w^T x \geq 0$, then its projection $\overline{x} = p_i(x)$ would violate $\overline{w}^T\overline{x} \geq 0$.
The equality set of $w^T x \geq 0$ defines a face F of C with $p_i(F) = \overline{F}$ since the points fulfilling $\overline{w}^T\overline{x} = t$ are exactly the projections of the points fulfilling $w^T x = t$. □

9.10 Remark
There are some faces lost by projection, i.e. even if F is a face of C, $p_i(F)$ may not be a face of $p_i(C)$. In fact, the faces of $p_i(C)$ correspond to the faces of C given by valid inequalities $w^T x \geq t$ with $w_i = 0$. For a proof and more information on the projection of faces, see [BO98, p. 7].

9.11 Corollary
Let $C \subseteq \mathbb{R}^n$ be a cone with $-e_i \in C$. Let $\overline{F}$ be a face of $p_i(C)$, then we have $\overline{F} = p_i(F)$ for some face F of C with $-e_i \in F$.

Proof:
We have shown the existence of F in proposition 8.14, it remains to show $-e_i \in F$. F can be defined as the equality set of a a valid inequality $w^T x \geq 0$ with $w_i = 0$. We have

$$w^T(-e_i) = w_1 * 0 + ... + w_i * (-1) + ... + w_n * 0 = -w_i = 0,$$

so $-e_i$ is in the equality set and since $-e_i \in P$, we conclude $-e_i \in F$. □

This shows that $\text{Bound}^{\text{p}}{}_i(I)$ fulfills the first condition of being a fan: any face of a cone contained in $\text{Bound}^{\text{p}}{}_i(I)$ is also contained in it.

9.12 Proposition
Let $C, D \subseteq \mathbb{R}^n$ be cones with $-e_i \in C, D$. In this case, intersection and projection commute:

$$p_i(C) \cap p_i(D) = p_i(C \cap D).$$

Proof:
We may write $C = P(A, 0)$ and $D = P(B, 0)$ and get

$$C \cap D = \{x \in \mathbb{R}^n \mid Ax \geq 0, Bx \geq 0\}.$$

Unfolding the definitions of projection and intersection, we get

$$p_i(C \cap D) = \{\overline{x} \in \mathbb{R}^{n-1} \mid \exists z \in \mathbb{R}^n : p_i(z) = \overline{x}, Az \geq 0, Bz \geq 0\}$$

and

$$p_i(C) \cap p_i(D) = \{\overline{x} \in \mathbb{R}^{n-1} \mid \exists x, y \in \mathbb{R}^n : p_i(x) = p_i(y) = \overline{x}, Ax \geq 0, By \geq 0\}.$$

By choosing $x = y = z$, the direction $p_i(C \cap D) \subseteq p_i(C) \cap p_i(D)$ is obvious.

For the reverse inclusion, choose some point $\overline{x} \in p_i(C) \cap p_i(D)$. By definition, there are points $x, y \in \mathbb{R}^n$ with $p_i(x) = p_i(y) = \overline{x}$ fulfilling the inequalities. We know that $x_j = y_j$ for all $j \neq i$. If $x_i = y_i$, we are done by choosing $x = y = z$ in the definition of $p_i(C \cap D)$. Assume without loss of generality $x_i < y_i$ (otherwise interchange the roles of C and D), so there is some $\lambda \in \mathbb{R}_{\geq 0}$ with $x = y - \lambda e_i$. Since $y \in D$, $-e_i \in D$ and cones are closed under conical combinations of their elements, we have $x = y - \lambda e_i \in D$ and thus x fulfills both sets of inequalities. As desired, $\overline{x} = p_i(x) \in p_i(C \cap D)$ follows. □

9.13 Proposition
Let $C, D \in \mathcal{F}$ be cones in a fan with $-e_i \in C, D$. The intersection $p_i(C) \cap p_i(D)$ is a face of both $p_i(C)$ and $p_i(D)$.

Proof:
Note that $C \cap D$ is a cone with $-e_i \in C \cap D$ and $C \cap D$ is a face of C since $\mathcal{F}$ is a fan. Suppose $w^T x \geq 0$ is a valid inequality for C such that its equality set is the face $C \cap D$. If $w_i \neq 0$, we have

$$w^T(-e_i) = w_i * (-1) = -w_i \neq 0,$$

which contradicts $e_l \in C \cap D$. Therefore, $C \cap D$ is defined via a valid inequality with $w_i = 0$ and its projection is a face of C by remark 9.10. By proposition 9.12, we conclude that $p_i(C \cap D) = p_i(C) \cap p_i(D)$ is a face of C.
By interchanging the roles of C and D in the proof, we can show that $p_i(C) \cap p_i(D)$ is a face of $p_i(D)$ analogously. □

We have gathered all ingredients to show that $\text{Bound}^p_i(I)$ preserves the structure of $\text{Trop}(I)$ in the sense that it is still a polyhedral fan.

9.14 Theorem
Let $I \subseteq \mathbb{Q}[x]$ be a homogeneous ideal. Then $\text{Bound}^p_i(I)$ is a polyhedral fan.

Proof:
We know by theorem 7.34 that $\text{Trop}(I)$ is a fan and we have seen in lemma 9.7 that the projection of a cone is a cone. We already concluded that $\text{Bound}^p_i(I)$ is a set of cones. Corollary 9.11 shows that every non-zero face of a polyhedron in $\text{Bound}^p_i(I)$ is also contained in $\text{Bound}^p_i(I)$ and we may apply proposition 9.13 for $\mathcal{P} = \text{Trop}(I)$ to see that the intersection of two cones in $\text{Bound}^p_i(I)$ is a face of both. □

An algorithm to compute $\text{Bound}^p_i(I)$ follows directly from the definition and lemma 9.2.

9.15 Algorithm
Input:
$I \subseteq \mathbb{Q}[x]$, a homogeneous ideal,
$i \in \{1, ..., n\}$
Output:
$\text{Bound}^p_i(I)$

```
1 F := ∅
2 for C ∈ Trop(I) do
3 |   if −e_i ∈ C then
4 |   |   F := F ∪ {p_i(C)}
5 |   end
6 end
7 return F
```

Proof of termination:
$\text{Trop}(I)$ is a polyhedral complex and thus contains only finitely many cones and it can be computed in finite time. □

Proof of soundness:
We have shown in lemma 9.2 that it is sufficient to check $-e_i \in C$. Together with definition 9.1, it is immediately clear that the fan returned by the algorithm is $\text{Bound}^p_i(I)$. □

9.16 Remark
Singular provides the procedure `insertCone` to insert a cone into a fan, and we can use this command to implement the algorithm above. By default, the procedure checks whether the cone is compatible with the fan, i.e. if the intersection of the cone with any cone in the fan is a face of both, and returns an error if this is not the case. By proposition 9.13, we know that during our algorithm to compute $\text{Bound}^p_i(I)$, no incompatibilities will

occur and we can use an additional parameter to disable the checks to minimize the running time. Furthermore, the procedure will not only add the cone to the fan, but it will also add all of its faces. If we order the cones of Trop (I) by their dimension in ascending order, we know that if we reach some cone $C \in \text{Trop}(I)$ during the algorithm, we already have added all proper faces of $p_1(C)$ to $\mathcal{F}$, since they are the projections of faces of C containing $-e_1$ by corollary 9.11.

Theorem 7.37 stated that Trop (I) is a pure fan if we start with a homogeneous prime ideal. The final goal of this section is to study whether $\text{Bound}^{\text{p}}{}_i(I)$ retains this property.

9.17 Lemma
If $C \subseteq \mathbb{R}^n$ is a cone of dimension k with $-e_i \in C$, its projection has dimension $k-1$.

Proof:
By the definition of the projection,

$$\dim(p_i(C)) \in \{k, k-1\}$$

is clear. The question is whether the case $\dim(p_i(C)) = k$ can occur. Let U be the smallest affine space containing C, and note that U is actually a linear space and its projection $p_i(U)$ is a linear space containing $p_i(C)$. Since $-e_i \in U$ and vector spaces are closed under scalar multiplication, $\lambda e_i \in U$ for all $\lambda \in \mathbb{R}$. We loose this whole line by projecting, so we have $\dim_{\mathbb{R}}(p_i(U)) < \dim_{\mathbb{R}}(U)$ and thus $\dim(p_i(C)) \leq k-1$. □

9.18 Theorem
Let $I \subseteq \mathbb{Q}[\underline{x}]$ be a homogeneous prime ideal, then $\text{Bound}^{\text{p}}{}_i(I)$ is a pure fan.

Proof:
By theorem 9.14, only the pureness has to be shown.
Let $\overline{C}$ be a maximal cone of $\text{Bound}^{\text{p}}{}_i(I)$, i.e. $\overline{C} = p_i(C)$ for some cone $C \in \text{Trop}(I)$.
We can assume that C is maximal without loss of generality. If C is not maximal, then there is some $D \in \text{Trop}(I)$ with $C \subseteq D$, $-e_i \in D$ and $\overline{C} = p_i(C) \subseteq p_i(D)$. By the choice of $\overline{C}$ as maximal cone, equality has to hold for the projections, so we may replace C by D.
The structure theorem 7.37 states that Trop (I) is a pure fan if I is monomial-free, i.e. its maximal cones have the same dimension k for some $k \in \mathbb{N}$. By lemma 9.17, we conclude $\dim(\overline{C}) = \dim(C) - 1 = k - 1$ and we are done since $\overline{C}$ was an arbitrary cone and k is independent of $\overline{C}$. If I is not monomial-free, we have $\text{Trop}(I) = \text{Bound}^{\text{p}}{}_i(I) = \emptyset$, which is also a pure fan. □

9.19 Example
We continue example 9.8. Note that C_1 is the only cone of Trop $(I_{2,4})$ containing $-e_1$, so by definition,

$$\text{Bound}^{\text{p}}{}_1(I_{2,4}) = \{p_1(C_1)\}.$$

9.20 Example

Let $I = \langle\, x^2 + y^2 - w^2, z \,\rangle \subseteq \mathbb{Q}[w, x, y, z]$ be the non-saturated ideal from example 7.8. We have seen that $\mathrm{Trop}\,(I) = \mathrm{Trop}\,(I : \underline{x}^{\infty}) = \mathrm{Trop}\left(\langle 1 \rangle\right) = \emptyset$.

It is obvious that our algorithm to compute the boundary via projection can not recover the tropicalization of the circle and we have $\mathrm{Bound}^{\mathrm{p}}{}_4\,(I) = \emptyset$.

10. The correspondence between elimination and projection

The next question is how the two methods to compute the boundary are related. We have seen in section 8, that in classical algebraic geometry, there is some correspondence between elimination on the algebraic and projection on the geometric side, and we have used this correspondence to justify our definition of $\text{Bound}^{e}{}_{i}(\text{Trop}(I))$. We will show that there is a similar correspondence in tropical geometry.

When comparing the examples 8.17 and 9.19 respectively 8.16 and 9.20, one can see that the two methods did not always lead to the same result, but at least the support of $\text{Bound}^{p}{}_{i}(I)$ was always contained in the support of $\text{Bound}^{e}{}_{i}(I)$. The next theorem states that this inclusion holds in general.

10.1 Theorem
Let $I \subseteq \mathbb{Q}[\underline{x}]$ be a homogeneous ideal. Then the following inclusion holds:

$$\text{Supp}\left(\text{Bound}^{p}{}_{i}(I)\right) \subseteq \text{Supp}\left(\text{Bound}^{e}{}_{i}(I)\right).$$

Proof:
Assume $\overline{w}$ is in the support of $\text{Bound}^{p}{}_{i}(I)$. By definition, $\overline{w}$ is in some $p_i(C)$ for $C \in \text{Trop}(I)$ with $-e_i \in C$. This means that there is some $w \in C$ with $p_i(w) = \overline{w}$. Since $-e_i \in C$ and C is a cone, we know that the conical combination $w - \lambda e_i$ is also contained in C for any $\lambda \in \mathbb{R}_{\geq 0}$.
Our goal is to show $\overline{w} \in \text{Supp}(\text{Bound}^{e}{}_{i}(I))$ and by proposition 5.9, it is sufficient to show that $\text{in}_{\overline{w}}(f)$ is not a monomial for any $0 \neq f \in E$, where $E = \langle I, x_i \rangle \cap \mathbb{Q}[x_j \mid j \neq i]$.
If we show that for any $f \in E$, there is some $f' \in I$ and $\lambda \in \mathbb{R}_{\geq 0}$ such that

$$\text{in}_{\overline{w}}(f) = \text{in}_{w-\lambda e_i}\left(f'\right),$$

we are done, since we know that the right hand side is never a monomial by the assumption $w - \lambda e_i \in C \subseteq \text{Supp}(\text{Trop}(I))$.

Any polynomial $0 \neq f \in E$ can be written as $f = g + d * x_i$ for some $g \in I, d \in \mathbb{Q}[\underline{x}]$.

1. Case $d = 0$: Since $f \in \mathbb{Q}[\underline{x}]$ does not contain a term in which the i^{th} variable occurs, we have
$$\text{in}_{\overline{w}}(f) = \text{in}_{\overline{w}}(g + d * x_i) = \text{in}_{\overline{w}}(g) = \text{in}_{w}(g)$$
and we are done by choosing $f' = g \in I$ and $\lambda = 0$.

2. Case $d \neq 0$: Since $f \in \mathbb{Q}[x_j \mid j \neq i\,]$, but $d * x_i \in \langle x_i \rangle$, we may write

$$g = f - d * x_i.$$

Given any polynomial $\sum_{\alpha \in \mathbb{N}^n} c_\alpha \underline{x}^\alpha$, we may choose λ large enough such that $(w - \lambda e_i)^T \alpha$ is not maximal for any term $c_\alpha \underline{x}^\alpha$ with $\alpha_i > 0$ unless all terms with $c_\alpha \neq 0$ have x_i as a factor.

Since we assumed $f \neq 0$, we know that terms without x_i as factor do exist in g. We may choose λ large enough such that

$$\mathrm{in}_{\overline{w}}(f) = \mathrm{in}_{w-\lambda e_i}(f) = \mathrm{in}_{w-\lambda e_i}(g),$$

so the terms coming from the $(d * x_i)$-part do not occur in the initial form of g. We are done by using this λ and choosing $f' = g \in I$.

□

We have seen that the reverse inclusion does not hold, but as shown in example 7.8, the ideal used in example 9.20 is not saturated. If we replace it by its saturation with respect to the product of the variables $\underline{x}$, we get $\mathrm{Bound}^{\mathrm{e}}{}_i\left(\langle 1 \rangle\right) = \mathrm{Bound}^{\mathrm{p}}{}_i\left(\langle 1 \rangle\right) = \emptyset$ for any $i \in \{1, ..., 4\}$.

Therefore, our final goal is to show that the two ways yield the same result if we start with a homogeneous ideal which is saturated with respect to $\underline{x}$. We present a proof developed by Thomas Markwig.

To simplify the notation in the following proofs, we will assume that $i = 1$, i.e. we compute the boundary in the first direction. The following lemma justifies that this is valid and the results also hold for any other i with analogous proofs by swapping the variables.

10.2 Lemma
If an ideal $I \subseteq \mathbb{Q}[\underline{x}]$ is saturated with respect to the product of the variables $\underline{x}$, then it is saturated with respect to any variable x_i (for some $i \in \{1, ..., n\}$).

Proof:
We know $I \subseteq (I : x_i)$ by definition and $I = (I : \underline{x})$ by the assumption. We are done if we show $(I : x_i) \subseteq (I : \underline{x})$, since then $I \subseteq (I : x_i) \subseteq (I : \underline{x}) = I$ follows and equality has to hold in every step. Assume $f \in (I : x_i)$, i.e. $xf \in I$. The ideal I is closed under the multiplication with ring elements, so

$$\left(\prod_{\substack{j \in \{1,...,n\}, \\ j \neq i}} x_j \right) (x_i f) = \underline{x} f \in I,$$

and we are done since this is the defining condition for $f \in (I : \underline{x})$. □

10.3 Notation

In the following, $I \subseteq \mathbb{Q}[\underline{x}]$ will always be a homogeneous ideal which is saturated with respect to $\underline{x}$ (and therefore also saturated with respect to x_1).
We write $\underline{xs} = x_2 * ... * x_n$ and $\mathbb{Q}[\underline{xs}]$ for the subring

$$\mathbb{Q}[\underline{xs}] = \mathbb{Q}[x_2, ..., x_n] = \mathbb{Q}[x_j \mid j \neq 1\ ].$$

The ideal

$$E = \langle\ I, x_1\ \rangle_1^e = \langle\ I, x_1\ \rangle \cap \mathbb{Q}[\underline{xs}] \subseteq \mathbb{Q}[\underline{xs}]$$

will be the elimination ideal used to define $\text{Bound}^e{}_1\,(\text{Trop}\,(I))$.
Given a polynomial

$$g = \sum_{\alpha \in \mathbb{N}^n} c_\alpha x^\alpha \in \mathbb{Q}[\underline{x}]$$

we define $g(0, \underline{xs})$ as the polynomial in $\mathbb{Q}[\underline{xs}]$ we get by letting all terms containing x_1 vanish, i.e.

$$g(0, \underline{xs}) = \sum_{\alpha \in \mathbb{N}^n, \alpha_1 = 0} c_\alpha x^\alpha \in \mathbb{Q}[\underline{xs}].$$

10.4 Remark

Any monomial ordering $>$ on $\mathbb{Q}[\underline{x}]$ induces a monomial ordering on $\mathbb{Q}[\underline{xs}]$ which we will also call $>$. If $>$ is global, the induced ordering is also global.

10.5 Definition

Given a polyhedral cone $C \subseteq \mathbb{R}^n$, we define its interior $\text{Int}\,(C)$ as the interior of C as set in the euclidean topology on $\mathbb{R}^n$. The relative interior $\text{relInt}\,(C)$ is the interior of C in the induced topology on its affine hull, the smallest linear subspace of $\mathbb{R}^n$ which contains C.

10.6 Remark

If C has dimension n in the definition above, interior and relative interior coincide.

Let $C = P(A, 0)$ be given via irredundant inequalities, i.e. we can not remove rows of A without changing the polyhedron. Then the interior is the set of points fulfilling $Ax > 0$. Let A' be the submatrix of A we get by deleting all inequalities which are fulfilled with equality for all points of C. The relative interior is the set of points in C fulfilling $A'x > 0$.

10.7 Proposition

Let $>$ be a global monomial ordering such that $-e_1 \in C_>(I)$ and $G = G_>$ the corresponding reduced Groebner basis for I. Suppose $g \in G$ is a polynomial in the Groebner basis.

a) x_1 does not divide $\text{LM}_>(g)$.

b) For all $w \in C_>(I), \lambda \in \mathbb{R}_{>0}$: x_1 does not divide any term of $\text{in}_{w-\lambda e_1}\,(g)$

c) For all $w \in \mathrm{Int}(C_>(I))$: x_1 does not divide any term of $\mathrm{in}_w(g)$

Proof:

a) Suppose $\mathrm{LM}_>(g) = \underline{x}^\alpha$ with $\alpha_1 > 0$ and let $w \in C_>(I)$.

By proposition 5.20, we know that $\mathrm{LM}_>(\mathrm{in}_{w-\lambda e_1}(g)) = \mathrm{LM}_>(g)$ for any $\lambda \in \mathbb{R}_{\geq 0}$ since $w - \lambda e_1$ is a conical combination of vectors in the cone and cones are closed under those.

This implies that the $(w - \lambda e_1)$-degree of g and its leading monomial coincide and no term in g has a larger $(w - \lambda e_1)$-degree than $(w - \lambda e_1)^T \alpha$ for any $\lambda \in \mathbb{R}_{\geq 0}$.

If a monomial of the form $\underline{x}^\beta$ with $\beta_1 = 0$ occurs in g, then we can use this observation to conclude

$$\deg_w(\underline{x}^\beta) = \deg_{w-\lambda e_1}(\underline{x}^\beta) \leq \deg_{w-\lambda e_1}(\underline{x}^\alpha) = \deg_w(\underline{x}^\alpha) - \lambda\alpha_0$$

for any $\lambda \in \mathbb{R}_{\geq 0}$. This yields a contradiction, since we can choose λ large enough such that the inequality does not hold.

Therefore, all terms of g are divisible by x_1 and we can write $g = x_1 * f$ for some $f \in \mathbb{Q}[\underline{x}]$. Since I is saturated with respect to x_1, we conclude $f \in I$. The leading monomial of f is $\frac{\underline{x}^\alpha}{x_1}$ and since G is a Groebner basis, we can find a $g' \in G$ such that $LM_>(g')$ divides it. This implies that $\mathrm{LM}_>(g')$ divides $\mathrm{LM}_>(g)$, which is a contradiction to the definition of a reduced Groebner basis.

b) Suppose there is a monomial $\underline{x}^\beta$ in $\mathrm{in}_{w-\lambda e_1}(g)$ divisible by x_1 and let $\underline{x}^\alpha = \mathrm{LT}_>(g)$.

With the same argument as in the proof of part a), we get

$$\deg_w(\underline{x}^\beta) \leq \deg_w(\underline{x}^\alpha) = \deg_{w-\lambda e_1}(\underline{x}^\alpha).$$

Furthermore, the definition of the weighted degree and the assumption $\beta_1 > 0$ yield

$$\deg_{w-\lambda e_1}(\underline{x}^\beta) = \deg_w(\underline{x}^\beta) - \lambda e_1 < \deg_w(\underline{x}^\beta).$$

The monomials $\underline{x}^\alpha$ and $\underline{x}^\beta$ occur both in $\mathrm{in}_{w-\lambda e_1}(g)$, so their $(w - \lambda e_1)$-degrees coincide and we derive the contradiction

$$\deg_w(\underline{x}^\beta) \leq \deg_{w-\lambda e_1}(\underline{x}^\alpha) = \deg_{w-\lambda e_1}(\underline{x}^\beta) < \deg_w(\underline{x}^\beta).$$

c) If $w \in \mathrm{Int}(C_>(I))$, there is a small ball $B_\epsilon(w) \subseteq C_>(I)$ for some $\epsilon \in \mathbb{R}_{>0}$. In particular, we have $w + \epsilon e_1 \in C_>(w)$. We may apply part b) for the vector $w + \epsilon e_1$ and $\lambda = \epsilon$ to conclude the desired statement.

□

Note that part a) implies that $g(0, \underline{xs}) \neq 0$ for any $g \in G_>$.

10.8 Proposition

Let $>$ be a global monomial ordering with $-e_1 \in C_>(I)$ and let $G = G_>$ be the corresponding reduced Groebner basis for I. Then G' is the reduced Groebner basis for $\langle\, I, x_1 \,\rangle$, where

$$G' = \{g(0, \underline{xs}) \mid g \in G\} \cup \{x_1\}\,.$$

Proof:

Note that $g - g(0, \underline{xs}) \in \langle x_1 \rangle$, so all terms occurring in the expression

$$g(0, \underline{xs}) = g - x_1 \frac{g - g(0, \underline{xs})}{x_1}$$

are in $\mathbb{Q}[\underline{x}]$ and we conclude $g(0, \underline{xs}) \in \langle\, I, x_1 \,\rangle$. Since $x_1 \in \langle\, I, x_1 \,\rangle$, we have shown that G' is a subset of $\langle\, I, x_1 \,\rangle$.

Out next goal is to show that the leading ideal of the G' is the leading ideal of $\langle\, I, x_1 \,\rangle$, where only the inclusion $L_>(G') \supseteq L_>(\langle\, I, x_1 \,\rangle)$ is non-trivial. Let $h \in \langle\, I, x_1 \,\rangle$ be an arbitrary element. We may write it as $h = f + dx_1$ for some $f \in I, d \in \mathbb{Q}[\underline{x}]$.

If x_1 divides $\mathrm{LM}_>(h)$, we know that $\mathrm{LM}_>(h) \in L_>(G')$ since $x_1 \in G'$, so we can assume $x_1 \nmid \mathrm{LM}_>(h)$ in the following.

G was a Groebner basis for I, so by theorem 5.18, there is a representation

$$f = \sum_{g \in G} c_g g$$

with $c_g \in \mathbb{Q}[\underline{x}]$ and for the finitely many c_g with $c_g \neq 0$, $LM_>(f) \geq \mathrm{LM}_>(c_g g)$ holds. We can decompose this representation into

$$f = \sum_{\substack{g \in G, \\ x_1 \nmid c_g}} c_g g + \sum_{\substack{g \in G, \\ x_1 \mid c_g}} c_g g = \sum_{\substack{g \in G, \\ x_1 \nmid c_g}} c_g g + x_1 \sum_{\substack{g \in G, \\ x_1 \mid c_g}} \frac{c_g}{x_1} g$$

and plug this in to get

$$f = \sum_{\substack{g \in G, \\ x_1 \nmid c_g}} c_g g + x_1 \sum_{\substack{g \in G, \\ x_1 \mid c_g}} \frac{c_g}{x_1} + d * x_1 = \sum_{\substack{g \in G, \\ x_1 \nmid c_g}} c_g g + x_1 \left(\sum_{\substack{g \in G, \\ x_1 \mid c_g}} \frac{c_g}{x_1} + d \right).$$

By replacing d with $\sum_{g \in G,\ x_1 \mid c_g} \frac{c_g}{x_1} + d$ and f by $\sum_{g \in G,\ x_1 \nmid c_g} c_g g$, we can assume that we have a representation

$$h = f + x_1 d = \sum_{g \in G} c_g g + x_1 d$$

such that all non-zero c_g are not divisible by x_1. By proposition 10.7, we know that $\mathrm{LM}_>(g)$ is not divisible by x_1 and we can conclude that x_1 does not divide $\mathrm{LM}_>(f) = \mathrm{LM}_> \left(\sum_{g \in G} c_g g \right)$.

Since x_1 does neither divide $\mathrm{LM}_>(h)$ nor $\mathrm{LM}_>(f)$, but it divides $\mathrm{LM}_>(x_1 * d)$, we conclude

$$\mathrm{LM}_>(h) = \mathrm{LM}_>(f) = \mathrm{LM}_> \left(\sum_{g \in G} c_g g \right) \in \langle\, \mathrm{LM}_>(g) \mid g \in G \,\rangle.$$

By proposition 10.7, we know that x_1 does not divide $\mathrm{LM}_>(g)$ and in particular, their leading monomials coincide: $\mathrm{LM}_>(g) = \mathrm{LM}_>(g(0, \underline{xs}))$. We can use this to obtain

$$\mathrm{LM}_>(h) \in \langle\, \mathrm{LM}_>(g) \mid g \in G \,\rangle = \langle\, \mathrm{LM}_>(g(0, \underline{xs})) \mid g \in G \,\rangle \subseteq L_>(G').$$

To see that G' is a reduced Groebner basis, observe that all terms occurring in $g(0, \underline{xs})$ also occur in g and furthermore, no term of $g(0, \underline{xs})$ is divisible by x_1, so the conditions c) and d) in definition 5.17 are fulfilled. We already stated $\mathrm{LT}_>(g(0, \underline{xs})) = \mathrm{LT}_>(g) = \mathrm{LM}_>(g)$, so the $g(0, \underline{xs})$ are normalized and we are done. □

10.9 Corollary
Let $>$ be a global monomial ordering with $-e_1 \in C_>(I)$ and let $G = G_>$ be the corresponding reduced Groebner basis for I. Then G'' is the reduced Groebner basis for E with respect to the induced ordering, where

$$G'' = \{ g(0, \underline{xs}) \mid g \in G \}.$$

Proof:
We have seen in the proof of proposition 10.8 that $g(0, \underline{xs}) \in \langle\, I, x_1 \,\rangle$, and by definition we have $g(0, \underline{xs}) \in \mathbb{Q}[\underline{xs}]$. We conclude $g(0, \underline{xs}) \in E$ and $G'' \subseteq E$.
Any polynomial $h \in E$ is also a polynomial in $\langle\, I, x_1 \,\rangle$ and has a standard representation

$$h = \sum_{g \in G} c_g g(0, \underline{xs}) + c_{x_1} x_1$$

for some $c_g, c_{x_1} \in \mathbb{Q}[\underline{x}]$ since G' is a Groebner basis for $\langle\, I, x_1 \,\rangle$ proposition 10.8. Since both the left hand side h and the Groebner basis elements of type $g(0, \underline{xs})$ are contained in the subring $\mathbb{Q}[\underline{xs}]$, we can assume $c_{x_1} = 0$ and $c_g \in \mathbb{Q}[\underline{xs}]$. By this, we get a standard representation $h = \sum_{g \in G} c_g g(0, \underline{xs})$ for h in $\mathbb{Q}[\underline{xs}]$ with respect to G''. By theorem 5.18, this is an equivalent condition for G'' being a Groebner basis, i.e. $L_>(G'') = L_>(E)$. By definition, $G'' \subseteq G'$, and G' was minimal, reduced and normalized, so G'' retains these properties. We have shown that G'' is a reduced Groebner basis.

□

10.10 Proposition

Let $>$ be a monomial ordering with $-e_1 \in C_>(I)$. Then the projection of the Groebner cone $C_>(I)$ is the Groebner cone $C_>(E)$ for the induced ordering:

$$C_>(E) = p_1(C_>(I)) = \{\overline{w} \in \mathbb{R}^{n-1} \mid \exists w_1 \in \mathbb{R} : (w_1, \overline{w}) \in C_>(I)\}.$$

Proof:

Note that the second equality holds by the definition of p_1, so only the first equality remains to show.

Let $G = G_>$ be the reduced Groebner basis for I, we have shown in corollary 10.9 that $G'' = \{g(0, \underline{xs}) \mid g \in G\}$ is the reduced Groebner basis for E.

"$C_>(E) \supseteq p_1(C_>(I))$"

Assume $w = (w_1, \overline{w}) \in C_>(I)$ and let $g \in G$ be an arbitrary element of the Groebner basis. By the propositions 10.7 and 5.20, we have

$$\mathrm{LM}_>(g(0, \underline{xs})) = \mathrm{LM}_>(g) = \mathrm{LM}_>(\mathrm{in}_w(g)).$$

This means $\mathrm{LM}_>(g)$ occurs in $\mathrm{in}_w(g)$, but since $\mathrm{LM}_>(g)$ is also the leading monomial $g(0, \underline{xs})$ it also occurs in $\mathrm{in}_w(g(0, \underline{xs}))$. Since $g(0, \underline{xs}) \in \mathbb{Q}[\underline{xs}]$, we can write

$$\mathrm{in}_w(g(0, \underline{xs})) = \mathrm{in}_{\overline{w}}(g(0, \underline{xs}))$$

and we conclude

$$\mathrm{LM}_>(g(0, \underline{xs})) = \mathrm{LM}_>(\mathrm{in}_{\overline{w}}(g(0, \underline{xs}))).$$

The polynomial g was an arbitrary element of G and as stated above, $g(0, \underline{xs}) \in G''$ is an element of the reduced Groebner basis for E and all elements of G'' are of this form. We can apply proposition 5.20 to get $\overline{w} \in C_>(E)$.

"$C_>(E) \subseteq p_1(C_>(I))$"

Assume $\overline{w} \in C_>(E)$. Then by proposition 5.20

$$\mathrm{LM}(g(0, \underline{xs})) = LM_>(\mathrm{in}_{\overline{w}}(g(0, \underline{xs})))$$

for any $g(0, \underline{xs}) \in G''$ since G'' is a reduced Groebner basis for E. We can consider $g(0, \underline{xs})$ as a polynomial in $\mathbb{Q}[\underline{x}]$ and write

$$\mathrm{LM}_>(\mathrm{in}_{\overline{w}}(g(0, \underline{xs}))) = \mathrm{LM}_>(\mathrm{in}_w(g(0, \underline{xs})))$$

for $w = (w_1, \overline{w})$ with $w_1 \in \mathbb{R}$ arbitrary. Similar to our argument in the proof of theorem 10.1, we can choose λ_g large enough such that for $w_g = (-\lambda_g, \overline{w})$, we have

$$\mathrm{in}_{w_g}(g(0, \underline{xs})) = \mathrm{in}_{w_g}(g)$$

and thus

$$\mathrm{LM}_{>}(\mathrm{in}_{w_g}(g(0, \underline{xs}))) = \mathrm{LM}_{>}(\mathrm{in}_{w_g}(g)).$$

This property continues to hold if we increase λ even further than necessary, since the left hand side does not contain any term divisible by x_1. Standard bases are finite sets, so we can choose $\lambda = \max\{\lambda_g \mid g \in G\}$ and $w = (-\lambda, \overline{w})$ to get a vector with

$$\mathrm{LM}_{>}(\mathrm{in}_{w_g}(g(0, \underline{xs}))) = \mathrm{LM}_{>}(\mathrm{in}_{w_g}(g))$$

for all $g \in G$. Since $\mathrm{LM}_{>}(\mathrm{in}_{w_g}(g(0, \underline{xs}))) = \mathrm{LM}_{>}(\mathrm{in}_{\overline{w}}(g(0, \underline{xs}))) = \mathrm{LM}_{>}(g)$ by proposition 10.7, we conclude

$$\mathrm{LM}_{>}(g) = \mathrm{LM}_{>}(\mathrm{in}_{w_g}(g))$$

for all $g \in G$. This shows $w \in C_{>}(I)$ by proposition 5.20.

□

10.11 Lemma

Let $J \subseteq K[\underline{x}]$ be a homogeneous ideal, $>$ a global monomial ordering and $w \in C_{>}(J)$ a weight vector. Then the set $\{\mathrm{in}_w(g) \mid g \in G_{>}\}$ is a reduced Groebner basis for $\mathrm{in}_w(J)$ with respect to $>$.

Proof:

See [Jen07, p. 33]. □

10.12 Lemma

The relative interior of a cone in the Groebner fan is an equivalence class with respect to the equality of initial ideals. More precisely, let C be a cone in the Groebner fan of I and let $v, w \in \mathrm{relInt}(C)$. Then $\mathrm{in}_v(I) = \mathrm{in}_w(I)$.

Proof:

See [Jen07, p. 34]. □

10.13 Proposition

Let $C \in GF(I)$ be a cone in the Groebner fan of I with $-e_1 \in C$. For any weight vector $w = (w_1, \overline{w}) \in \mathrm{relInt}(C)$:

$$\mathrm{in}_w(I) \text{ is monomial-free} \Leftrightarrow \mathrm{in}_{\overline{w}}(E) \text{ is monomial-free},$$

where E is the elimination ideal defined in 10.3.

Proof:

First note that it is sufficient to show the statement for "small" w_1: If $w \in \mathrm{relInt}(C)$, then $w - \lambda e_1 \in \mathrm{relInt}(C)$ for any $\lambda \in \mathbb{R}_{\geq 0}$ and by lemma 10.12, we have $\mathrm{in}_w(I) = \mathrm{in}_{w-\lambda e_1}(I)$. Furthermore if C is a cone in the Groebner fan and $-e_1 \in C$, there is a global monomial ordering with $C \subseteq C_>(I)$ and $-e_1 \in C_>(I)$. Let G be the reduced Groebner basis with respect to this ordering.

"$\Rightarrow$"

Suppose the monomial $\underline{x}^\alpha$ is contained in $\mathrm{in}_{\overline{w}}(E)$. Since G'' as in corollary 10.9 is a Groebner basis for E, we may use lemma 10.11 to derive a standard representation

$$\underline{x}^\alpha = \sum_{g \in G} c_g \, \mathrm{in}_{\overline{w}}(g(0, \underline{xs}))$$

for some $c_g \in \mathbb{Q}[\underline{xs}]$. As in the proof of proposition 10.10, we can choose λ large enough such that for $w' = (-\lambda, \overline{w}) = w - (\lambda + w_1)e_1$ we have

$$\mathrm{in}_{\overline{w}}(g(0, \underline{xs})) = \mathrm{in}_{w'}(g)$$

for all $g \in G$. This yields the contradiction

$$\underline{x}^\alpha = \sum_{g \in G} c_g \, \mathrm{in}_{\overline{w}}(g(0, \underline{xs})) = \sum_{g \in G} c_g \, \mathrm{in}_{w'}(g) \in \mathrm{in}_{w'}(I)\,.$$

"$\Leftarrow$"

Suppose the monomial $\underline{x}^\alpha$ is contained in $\mathrm{in}_w(I)$. Again, lemma 10.11 yields a standard representation

$$\underline{x}^\alpha = \sum_{g \in G} c_g \, \mathrm{in}_w(g)$$

for some $c_g \in \mathbb{Q}[\underline{x}]$. Since the left-hand side is w-homogeneous and the $\mathrm{in}_w(g)$ are w-homogeneous by definition, we can assume that the c_g are w-homogeneous such that

$$\deg_w(c_g) = \deg_w(\underline{x}^\alpha) - \deg_w(\mathrm{in}_w(g))\,.$$

We distinguish two cases.

1. Case $\alpha_1 = 0$:
   Since the left hand side is not divisible by x_1, all terms of the right hand side in which x_1 occurs have to cancel out and we may write

   $$\underline{x}^\alpha = \sum_{g \in G} c_g(0, \underline{xs}) \, \mathrm{in}_w(g(0, \underline{xs})) = \sum_{g \in G} c_g(0, \underline{xs}) \, \mathrm{in}_{\overline{w}}(g(0, \underline{xs}))$$

   where the last equality holds since w_1 does not occur as factor in any term of $g(0, \underline{xs})$. We have shown that the set of all $g(0, \underline{xs})$ is a Groebner basis for

E. In particular, the right hand side is a expression in $\mathrm{in}_{\overline{w}}(E)$ and we have derived a contradiction to the assumption that $\mathrm{in}_{\overline{w}}(E)$ is monomial-free.

2. Case $\alpha_1 \neq 0$:
We have seen in proposition 10.7 that x_1 does not divide any term of $\mathrm{in}_w(g)$ if we assume w_1 small enough. We can view the c_g as polynomials in $(\mathbb{Q}[\underline{xs}])[x_1]$, i.e.

$$c_g = \sum_{i=0}^{\deg_{e_1}(c_g)} c_{g,i} x_1^i$$

where $c_{g,i} \in \mathbb{Q}[\underline{xs}]$. As in the first case, all terms with e_1-degree not equal to α_1 have to cancel out, so we may write

$$\underline{x}^{\alpha} = \sum_{g \in G} c_{g,\alpha_1} x^{\alpha_1} \mathrm{in}_w(g)\,.$$

By just removing the factor x^{α_1} in each term, we get a new monomial in $\mathrm{in}_w(I)$

$$\underline{x}^{\beta} = \sum_{g \in G} c_{g,\alpha_1} \mathrm{in}_w(g)$$

with $\beta_0 = 0$. We have already shown in the first case that this yields a contradiction to the assumption.

□

10.14 Theorem
The Groebner fan of the elimination ideal consists exactly of the projections of the cones in the Groebner fan of I containing $-e_1$:

$$GF(E) = \{p_1(C) \mid C \in GF(I), -e_1 \in C\}.$$

Proof:
Suppose C is in the Groebner fan of E. By the definition of the Groebner fan, there is a global monomial ordering $>$ on $\mathbb{Q}[\underline{xs}]$ such that C is a face of $C_>(E)$. We define the monomial ordering $>'$ on $\mathbb{Q}[\underline{x}]$ by

$$\underline{x}^{\alpha} >' \underline{x}^{\beta} \Leftrightarrow -\alpha_1 > -\beta_1 \quad \text{or } \left(\alpha_1 = \beta_1 \text{ and } \tfrac{\underline{x}^{\alpha}}{x_1^{\alpha_1}} > \tfrac{\underline{x}^{\beta}}{x_1^{\alpha_1}}\right).$$

This monomial ordering is not global since $x_1 < 1$, but we may construct the global monomial ordering $>'_h$ as in definition 5.14. The set $C_{>'_h}(I)$ is contained in the Groebner fan of I, let us check whether $e_1 \in C_{>'_h}(I)$.
By theorem 5.18, the reduced Groebner basis $G_{>'_h}$ is homogeneous, so taking the leading monomial with respect to $>'_h$ and with respect to $>'$ yields the same result. Furthermore, by the definition of $>'$, the leading monomial is always among the terms with

minimal e_1-degree. Therefore, we have $\mathrm{LM}_{>'_h}(\mathrm{in}_{-e_1}(g)) = \mathrm{LM}_{>'_h}(g)$ for any $g \in G_{>'_h}$ and we can conclude by proposition 5.20 that $-e_1 \in C_{>'_h}(I)$. We have already shown in proposition 10.10 that in this case, the projection of the cone is the Groebner cone of E with respect to the induced order. Since x_1 does not occur in polynomials in $\mathbb{Q}[\underline{xs}]$, we see that the ordering induced by $>'_h$ on $\mathbb{Q}[\underline{xs}]$ is $>_h$.
Using proposition 5.20 again, it is easy to show $C_>(E) = C_{>_h}(E)$ since again the Groebner bases $G_>$ and $G_{>_h}$ coincide by lemma 5.19 and they contain only homogeneous polynomials, so taking the leading monomial with respect to $>$ and $>_h$ yields the same result.
We summarize what we have shown: any maximal cone $C_>(I)$ containing $-e_1$ gets projected to a cone of $GF(E)$ by proposition 10.10 and we have just seen that any cone in $GF(E)$ is contained in a maximal cone $C_>(E)$ which can be obtained by projecting a cone $C_>(I)$. To finish the proof, we can use the theory developed in last section to see that even the non-maximal cones of $GF(E)$ are projections of cones in $GF(I)$ containing $-e_1$ by corollary 9.11. □

10.15 Corollary
The tropicalization of the elimination ideal consists exactly of the projections of the cones in the tropicalization of I containing $-e_1$:

$$\mathrm{Trop}(E) = \{p_1(C) \mid C \in \mathrm{Trop}(I), -e_1 \in C\}.$$

In particular,

$$\mathrm{Supp}(\mathrm{Trop}(E)) = \bigcup_{\substack{C \in \mathrm{Trop}(I), \\ -e_1 \in C}} p_1(C).$$

Proof:
We have seen in theorem 7.34 that $\mathrm{Trop}(E)$ consists of the cones of $GF(E)$ such that for all weight vectors w in their relative interior, $\mathrm{in}_w(E)$ is monomial-free, so we can apply theorem 10.14 together with proposition 10.10. □

To conclude this thesis, we drop the special notation introduced in this chapter and state the theorem for the general case.

10.16 Theorem
Let $I \subseteq \mathbb{Q}[\underline{x}]$ be a homogeneous ideal which is saturated with respect to $\underline{x} = x_1 * ... * x_n$ and let $i \in \{1, ..., n\}$. The two ways to compute the boundary lead to the same result:

$$\mathrm{Bound}^{\mathrm{p}}{}_i(I) = \mathrm{Bound}^{\mathrm{e}}{}_i(I).$$

Proof:
This follows directly from theorem 10.15, if we swap the first and the i^{th} variable. □

10.17 Corollary

Let $r, m \in \mathbb{N}$ with $m \geq r$ and let $i \in \left\{1, ..., \binom{m}{r}\right\}$. The two ways to compute the boundary applied to the Pluecker ideal $I_{r,m}$ lead to the same result:

$$\text{Bound}^{p}{}_{i}\left(I_{r,m}\right) = \text{Bound}^{e}{}_{i}\left(I_{r,m}\right).$$

Proof:

By theorem 7.7, the Pluecker ideal is saturated with respect to the product of variables, so this follows from theorem 10.16. □

Part V.
Appendix: SINGULAR code

The appendix contains code for the computer algebra system SINGULAR [Sing] providing implementations for the algorithms mentioned in the theoretical part of the thesis.

Note that at the moment, SINGULAR does not support polyhedral objects by default. The compilation of the binary library `gfanlib.so` [gfanlib] has to be explicitly enabled when compiling SINGULAR . This has to be done, or any of the following code (except `grassmanian.lib`) will not work.

A. grassmanian.lib

This section contains the code implementing the algorithms studied in the second part of this thesis to compute the Pluecker ideal defining the Grassmanian.

A.1 Library overview

Library: `grassmanian.lib`
Purpose: Compute the Pluecker ideal defining the Grassmanian
Category: Algebraic geometry
Author: Sebastian Muskalla (muskalla@mathematik.uni-kl.de)
References: [Stu93], this thesis
Procedures:

`subsets(m, r)`	Subsets of $\{1, ..., m\}$ of size r
`sublists(L, r)`	Sublists of L of size r
`plueckerCoordRing(r,m)`	Pluecker coordinate ring $K[\underline{p}]$
`plueckerRelation(I, J)`	Pluecker relation $P_{I,J}$
`grassmanianGenerators(r, m)`	Generators for the Pluecker ideal $I_{r,m}$
`vanDerWaerdenSyzygy(a, b, c)`	Van Der Waerden syzygy $[[a\dot{b}c]]$
`grassmanianGB(r, m)`	$>_{dp}$-Groeber basis for the Pluecker ideal

A.2 Exported procedures

The following procedures are available after the library has been loaded,

- **subsets**

  Usage: `subsets(r, m, [upto]);`
  r, m natural numbers, $m \geq r$, upto 0 (default) or 1

  Purpose: Compute all subsets of $\{1, ..., m\}$ of size = r (or *leqr* iff upto was 1)

  Return: Subsets as list of lists

  Note: The subsets are ordered with respect to the order from remark 3.4

  Example:

```
> example subsets;
// proc subsets from lib grassmanian.lib
EXAMPLE:
    list eq2 = subsets (2, 3);
    print (eq2);
[1]:
   [1]:
      1
   [2]:
      2
[2]:
   [1]:
      1
   [2]:
      3
[3]:
   [1]:
      2
   [2]:
      3
    list leq2 = subsets (2, 3, 1);
    print (leq2);
[1]:
   empty list
[2]:
   [1]:
      1
[3]:
   [1]:
      2
[4]:
   [1]:
      3
[5]:
   [1]:
      1
   [2]:
      2
[6]:
   [1]:
      1
   [2]:
      3
[7]:
   [1]:
      2
   [2]:
      3
```

- **sublists**

  Usage: `sublists(r, L);`
  r natural number, L list such that $|L| \geq r$

  Purpose: Compute all sublists of L of size r

  Return: List of sublists

  Example:

```
> example sublists;
// proc sublists from lib grassmanian.lib
EXAMPLE:
    list testparam = 1,3,7;
    list ls = sublists (2, testparam);
    print (ls);
[1]:
   [1]:
      1
   [2]:
      3
[2]:
   [1]:
      1
   [2]:
      7
[3]:
   [1]:
      3
   [2]:
      7
```

- **plueckerCoordRing**

  Usage: `plueckerCoordRing(r, m);`
  r, m natural numbers with $m \geq r$

  Purpose: Computes the Pluecker coordinate ring $K[\underline{p}]$ (with $>_{dp}$ / the tableaux order as monomial ordering)

  Return: The Pluecker coordinate ring for the given r, m

  Example:

```
> example plueckerCoordRing ;
// proc plueckerCoordRing from lib grassmanian.lib
EXAMPLE:
    def Kp = plueckerCoordRing(2,3);
    setring Kp;
    print (Kp);
polynomial ring, over a field, global ordering
//   characteristic : 0
//   number of vars : 3
//        block   1 : ordering dp
//                  : names    p_1_2 p_1_3 p_2_3
//        block   2 : ordering C
```

- **plueckerRelation**

  Usage: `plueckerRelation(I, J);`
  I, J lists consisting of natural numbers sorted in ascending order of size $|I| = r - 1, |J| = r + 1$

  Assume: The Pluecker coordinate ring $K[\underline{p}]$ for appropriate r and m is active

  Purpose: Compute the quadric Pluecker relation $P_{I,J}$ given by the sets I and J

  Return: The Pluecker relation as a polynomial in the Plucker coordinate ring

  Example:

```
> example plueckerRelation ;
// proc plueckerRelation from lib grassmanian.lib
EXAMPLE:
    def Kp = plueckerCoordRing(2,3);
    setring Kp;
    list I = 1;
    list J = 1, 2, 3;
    print(plueckerRelation(I, J));
// j = J[1] = 1
//     j already contained in I
// j = J[2] = 2
//     p_1_2*p_1_3
// j = J[3] = 3
//     -p_1_2*p_1_3
0
```

- **grassmanianGenerators**

  Usage: `grassmanianGenerators(r, m);`
  r, m natural numbers with $m \geq r$

  Purpose: Compute a set of generators for the Pluecker ideal using the Pluecker relations

  Return: The Pluecker coordinate ring with the Pluecker ideal called `Irm` in it

  Note: The generators of the ideal may not form a Groebner basis.
  The procedure returns a ring with the ideal inside, see the enclosed example.

  Example:

```
> example grassmanianGenerators ;
// proc grassmanianGenerators from lib grassmanian.lib
EXAMPLE:
    def Kp = grassmanianGenerators(2,3);
// Pluecker relation for
//     I = 1
//     J = 1,2,3
// P_I,J = 0
// Pluecker relation for
//     I = 2
//     J = 1,2,3
// P_I,J = 0
// Pluecker relation for
//     I = 3
//     J = 1,2,3
// P_I,J = 0
```

```
// This method returns a ring containing the ideal "Irm"
// USAGE: def Kp = grassmanianGenerators(_,_); setring Kp; Irm;
    setring Kp;
    // Irm = 0 because G(r,m) = |P^{2}
    print (Irm);
0
```

- **vanDerWaerdenSyzygy**

  Usage: `vanDerWaerdenSyzygy(alpha, beta, gamma);`
  α, β, γ subsets of $\{1, ..., m\}$ of size $s-1, r+1, r-s$

  Assume: The Pluecker coordinate ring $K[\underline{p}]$ for appropriate r and m is active.

  Purpose: Compute the quadric van der Waerden syzygy $[[\alpha\dot{\beta}\gamma]]$.

  Return: The van der Waerden Syzygy as a polynomial in the Pluecker coordinate ring

  Example:

```
> example vanDerWaerdenSyzygy ;
// proc vanDerWaerdenSyzygy from lib grassmanian.lib
EXAMPLE:
    def Kp = plueckerCoordRing(2,4);
    setring Kp;
    list alpha = list(); // empty list
    list beta = 1,2,3;
    list gamma = 4;
    print(vanDerWaerdenSyzygy(alpha, beta, gamma));
// beta_tau     = 1
// beta_tau_bar = 2,3
//    p_1_4*p_2_3
// beta_tau     = 2
// beta_tau_bar = 1,3
//    -p_1_3*p_2_4
// beta_tau     = 3
// beta_tau_bar = 1,2
//    p_1_2*p_3_4
p_1_4*p_2_3-p_1_3*p_2_4+p_1_2*p_3_4
```

- **grassmanianGB**

  Usage: `grassmanianGB(r, m);`
  r, m natural numbers with $m \geq r$

  Purpose: Compute a $>_{dp}$-Groebner basis for the Pluecker ideal using the van Der Waerden syzygies

  Return: The Pluecker coordinate ring with the Pluecker ideal given via a Groebner basis called `Irm` in it

  Note: The generators of the ideal will form a Groebner basis with respect to $>_{lp}$ / the tableaux order.
  The procedure returns a ring with the ideal inside, see the enclosed example.

  Example:

```
> example grassmanianGB;
// proc grassmanianGB from lib grassmanian.lib
EXAMPLE:
    def Kp = grassmanianGB(2, 4);
// Splitted
//     R = 1,4
//     S = 2,3
// into
//     a = 1
//     b = 2,3,4
//     c =
// [[a (dot b) c] =
//     p_1_4*p_2_3-p_1_3*p_2_4+p_1_2*p_3_4
// This method returns a ring containing the ideal "Irm"
// USAGE: def Kp = grassmanianGB(_,_); setring Kp; Irm;
    setring Kp;
    print (Irm);
p_1_4*p_2_3-p_1_3*p_2_4+p_1_2*p_3_4
```

A.3 Source code

Listing 1: grassmanian.lib

```
////////////////////////////////////////////////////////////////////
version="version grassmanian.lib 4.0.0.0 Apr_2015 "; // $Id: $
category="Algebraic Geometry";
info="
LIBRARY:   grassmanian.lib  Compute the Pluecker ideal defining the Grassmanian
AUTHOR:    Sebastian Muskalla, email: muskalla@mathematik.uni-kl.de

KEYWORDS: grassmanian; pluecker; subsets; van der waerden;

REFERENCES:
[1] Bernd Sturmfels: Algorithms in invariant theory, Springer (1991)@*
[2] Sebastian Muskalla: Computing the boundaries of tropical varieties,@*
    Master thesis, TU Kaiserslautern (2015)

PROCEDURES:
subsets(m, r);                   subsets of {1, ..., m} of size r
sublists(L, r);                  sublists of L of size r

plueckerCoordRing(r,m);          Pluecker coordinate ring

plueckerRelation(I, J);          Pluecker relation P_I,J
grassmanianGenerators(r, m);     generators for the Pluecker ideal

vanDerWaerdenSyzygy(a, b, c);    van Der Waerden syzygy [[ a (dot b) c]]
grassmanianGB(r, m);             dp-Groeber basis for the Pluecker ideal
";
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
// EXPORTED PROCEDURES
////////////////////////////////////////////////////////////////////
/**
 * @param r a natural number
 * @param m a natural number >= r
 * @param upto set to 1 to compute subsets of size <= r instead of subsets of size ==r
 * @returns a list of all subsets of {1, ..., m} of size == r
 *          (respectively <= r) (as list of lists)
 */
proc subsets (int r, int m, list #)
"USAGE:   subsets(r, m, [upto]);
         @* r, m natural numbers, m @math{\geq} r, upto 0 (default) or 1
PURPOSE: compute all subsets of {1, ..., m} of size @math{=} r
         (or @math{\leq} r iff upto was 1)
RETURN:  subsets as list of lists
NOTE:    the subsets are ordered with respect to the following order:
         @* @math{A < B} iff @math{|A| < |B|} or @math{|A| = |B|}
         and the smallest element not contained in both is contained in @math{A}
EXAMPLE: example subsets; shows an example"
{
    int upto = 0;
    list result;

    // find out whether upto was given and set to 1
    if (size(#) > 0)
    {
        if (typeof(#[1]) == "int")
        {
            if (#[1] == 1)
            {
```

```
59                  upto = 1;
60              }
61          }
62      }
63
64      // if yes, compute also subsets of size < r
65      if (upto)
66      {
67          for (int i = 0; i <= r-1; i++)
68          {
69              result = result + subsetsStartingFrom(i, m, 1);
70          }
71      }
72
73      // call helper procedure to compute subsets of size ==r
74      result = result + subsetsStartingFrom(r, m, 1);
75      return (result);
76  }
77  example
78  { "EXAMPLE:"; echo = 2;
79      list eq2 = subsets (2, 3);
80      print (eq2);
81      list leq2 = subsets (2, 3, 1);
82      print (leq2);
83  }
84  ////////////////////////////////////////////////////////////////////
85  /**
86   * @param r a natural number
87   * @param L a list
88   * @returns a list of all subsets of the elements of L of size r (as list of lists)
89   */
90  proc sublists (int r, list L)
91  "USAGE:   sublists(r, L);
92           @* r natural number, L list such that @math{|L| \geq r}
93  PURPOSE: compute all sublists of L of size r
94  RETURN:  list of sublists
95  EXAMPLE: example sublists; shows an example"
96  {
97      list result;
98
99      if (r == 0)
100     {
101         result[1] = list();
102         return (result);
103     }
104
105     int m = size(L);
106     if (m <= 0)
107     {
108         return (result);
109     }
110
111     // list without first element
112     list partial;
113     if (m > 1) // workaround because L[2..1] is illegal
114     {
115
116         partial = L[2 .. m];
117     }
118
119     list without = sublists(r  , partial);
120     list with    = sublists(r-1, partial);
121
122     for (int i = 1; i <= size(with); i++)
```

```
123     {
124         with[i] = list(L[1]) + with[i];
125     }
126
127     result = with + without;
128     return (result);
129 }
130 example
131 { "EXAMPLE:"; echo = 2;
132     list testparam = 1,3,7;
133     list ls = sublists (2, testparam);
134     print (ls);
135 }
136 ////////////////////////////////////////////////////////////////////
137 /**
138  * @param r a natural number
139  * @param m a natural number >= m
140  * @returns The Pluecker coordinate ring for r,m
141  */
142 proc plueckerCoordRing (int r, int m)
143 "USAGE:   plueckerCoordRing(r, m);
144          @* r, m natural numbers with @math{m \geq r}
145 PURPOSE: Computes the Pluecker coordinate ring @math{K[p]}
146          (with dp / the tableaux order as monomial ordering)
147 NOTE:    The result of this procedure should be stored in a variable of type \"def\",
148          see the enclosed example.
149 RETURN:  The Pluecker coordinate ring for the given r, m
150 EXAMPLE: example plueckerCoordRing; shows an example"
151 {
152     string ringstring = "ring Kpluecker_INTERNAL = 0,(";
153     string vars_string = "";
154
155     list rm_subsets = subsets(r, m);
156
157     // add one variable for each subset
158     for (int i = 1; i <= size(rm_subsets); i++)
159     {
160         vars_string = vars_string + subsetToVarstring(rm_subsets[i]);
161         if (i != size(rm_subsets))
162         {
163             vars_string = vars_string + ",";
164         }
165     }
166     ringstring = ringstring + vars_string + "),dp";
167     execute(ringstring);
168     return (Kpluecker_INTERNAL);
169 }
170 example
171 { "EXAMPLE:"; echo = 2;
172     def Kp = plueckerCoordRing(2,3);
173     setring Kp;
174     print (Kp);
175 }
176 ////////////////////////////////////////////////////////////////////
177 /**
178  * @param I a list of natural numbers, sorted in ascending order
179  * @param J a list of natural numbers, sorted in ascending order
180  * @returns the Pluecker relation given by I and J
181  */
182 proc plueckerRelation (list I, list J)
183 "USAGE:   plueckerRelation(I, J);
184          @* I,J lists consisting of natural numbers sorted in ascending order
185          of size @math{|I| = r-1} resp. @math{|J| = r+1}
186 ASSUME:  The Pluecker coordinate ring @math{K[p]} for appropriate r and m is active
```

```
187 PURPOSE: Compute the quadric Pluecker relation @math{P_I_,_J} given by the sets I and J
188 RETURN:  the Pluecker relation as a polynomial in the Plucker coordinate ring
189 EXAMPLE: example plueckerRelation; shows an example"
190 {
191     poly PIJ = 0;
192
193     int pluecker_sign;
194     int pluecker_sign_exponent;
195
196     int j;
197
198     /** index of j in J */
199     int remove_index;
200     /** index of j in I \cup {j} */
201     int insert_index;
202     list Icupj;
203
204     /** output will be printed if p > 0 */
205     int p = printlevel - voice +3;
206
207     for (remove_index = 1; remove_index <= size(J); remove_index++)
208     {
209         j = J[remove_index];
210         (Icupj, insert_index) = insertSorted(I, j);
211
212         dbprint(p , "// j = J[" + string(remove_index) + "] = " + string(j));
213
214         // check whether element was already in there
215         if (insert_index != 0)
216         {
217             pluecker_sign_exponent = size(J) - 1 - remove_index + insert_index;
218             pluecker_sign = 1;
219             if (pluecker_sign_exponent % 2 != 0)
220             {
221                 pluecker_sign = -1;
222             }
223
224             dbprint(p, "//     " + string(pluecker_sign * subsetToVar(Icupj)
225                         * subsetToVar (delete(J, remove_index))));
226
227             PIJ = PIJ
228                 + pluecker_sign
229                     * subsetToVar(Icupj)
230                     * subsetToVar (delete(J, remove_index));
231         }
232         else
233         {
234             dbprint(p, "//     j already contained in I");
235         }
236     }
237     return (PIJ);
238 }
239 example
240 { "EXAMPLE:"; printlevel = 1; echo = 2;
241     def Kp = plueckerCoordRing(2,3);
242     setring Kp;
243     list I = 1;
244     list J = 1, 2, 3;
245     print(plueckerRelation(I, J));
246 }
247 ////////////////////////////////////////////////////////////////
248 /**
249  * Computes the Pluecker coordinate ring with the the Pluecker ideal in it
250  *
```

```
251  * Usage:
252  *     def Kp = grassmanianGenerators(2, 4);
253  *     setring Kp;
254  *     Irm;
255  *
256  * @param r a natural number
257  * @param m a natural number >= m
258  * @returns The Pluecker coordinate ring with the Pluecker ideal called "Irm" in it
259  */
260 proc grassmanianGenerators (int r, int m)
261 "USAGE:   grassmanianGenerators(r, m);
262          @* r, m natural numbers with m @math{\geq} r
263 PURPOSE: Compute a set of generators for the Pluecker ideal using the Pluecker relations
264 RETURN:  The Pluecker coordinate ring with the Pluecker ideal called \"Irm\" in it
265 NOTE:    The generators of the ideal may not form a Groebner basis.
266          @* The procedure returns a ring with the ideal inside, see the enclosed example.
267 EXAMPLE: example grassmanianGenerators; shows an example"
268 {
269     def Kp = plueckerCoordRing(r, m);
270     setring Kp;
271
272     ideal Irm;
273
274     int index_I;
275     int index_J;
276     list list_I = subsets(r-1, m);
277     list list_J = subsets(r+1, m);
278
279     /** output will be printed if p > 0 */
280     int p = printlevel - voice +3;
281
282     // compute all Pluecker relations P_{I, J}
283     for (index_I = 1; index_I <= size(list_I); index_I++)
284     {
285         for (index_J = 1; index_J <= size(list_J); index_J++)
286         {
287             dbprint (p, "// Pluecker relation for");
288             dbprint (p, "//     I = " + string (list_I[index_I]));
289             dbprint (p, "//     J = " + string (list_J[index_J]));
290             dbprint (p, "// P_I,J = "
291                 + string(plueckerRelation (list_I[index_I], list_J[index_J])));
292
293             Irm = Irm + plueckerRelation (list_I[index_I], list_J[index_J]);
294         }
295     }
296
297     dbprint (p, "// This method returns a ring containing the ideal \"Irm\"");
298     dbprint (p, "// USAGE: def Kp = grassmanianGenerators(_,_); setring Kp; Irm;");
299
300     export (Irm);
301     return (Kp);
302 }
303 example
304 { "EXAMPLE:"; printlevel = 1; echo = 2;
305     def Kp = grassmanianGenerators(2,3);
306     setring Kp;
307     // Irm = 0 because G(r,m) = |P^{2}
308     print (Irm);
309 }
310 ////////////////////////////////////////////////////////////////
311 /**
312  * @param alpha a subset of {1, ..., m} of size s-1
313  * @param beta a subset of {1, ..., m} of size r+1
314  * @param gamma a subset of {1, ..., m} of size r-s
```

```
315  * @returns the van der Wearden Syzygy [[alpha (dot beta) gamma]]
316  */
317 proc vanDerWaerdenSyzygy (list alpha, list beta, list gamma)
318 "USAGE:   vanDerWaerdenSyzygy(alpha, beta, gamma);
319          @* alpha subset of {1, ..., m} of size s-1
320          @* beta subset of {1, ..., m} of size r+1
321          @* gamma subset of {1, ..., m} of size r-s
322 ASSUME:  The Pluecker coordinate ring for appropriate r and m is active
323 PURPOSE: Compute the quadric van der Waerden syzygy @math{[[\alpha \dot \beta \gamma]]}
324 RETURN:  the van der Waerden Syzygy as a polynomial in the Pluecker coordinate ring
325 EXAMPLE: example vanDerWaerdenSyzygy; shows an example"
326 {
327     poly vdWs = 0;
328
329     int s = size(alpha) + 1;
330     int r = size(beta)- 1;
331
332     if (size(gamma) != r - s)
333     {
334         ERROR ("vanDerWaerdenSyzygy: unexpected input size");
335     }
336
337     list beta_taus = sublists(s, beta);
338     list beta_tau;
339     list beta_tau_bar;
340
341     int sgn_fst_term;
342     int sgn_snd_term;
343     list fst_term;
344     list snd_term;
345
346     /** output will be printed if p > 0 */
347     int p = printlevel - voice +3;
348
349     for (int taui = 1; taui <= size(beta_taus); taui++)
350     {
351         beta_tau = beta_taus[taui];
352         beta_tau_bar = complementOfSet(beta_tau, beta);
353
354         (sgn_fst_term, fst_term) = concatSorted(alpha, beta_tau_bar);
355         (sgn_snd_term, snd_term) = concatSorted(beta_tau, gamma);
356
357         // term vanishes if double element occur
358         if (sgn_fst_term != 0 && sgn_snd_term != 0)
359         {
360
361             dbprint(p, "// beta_tau     = " + string(beta_tau));
362             dbprint(p, "// beta_tau_bar = " + string(beta_tau_bar));
363             dbprint(p, "//    " + string(sgn_fst_term
364                     * sgn_snd_term
365                     * signOfShuffle(beta_tau, beta_tau_bar)
366                     * subsetToVar(fst_term)
367                     * subsetToVar(snd_term)));
368
369             vdWs = vdWs + (sgn_fst_term * sgn_snd_term
370                         * signOfShuffle(beta_tau, beta_tau_bar)
371                         * subsetToVar(fst_term)
372                         * subsetToVar(snd_term));
373         }
374         else
375         {
376             dbprint(p, "// beta_tau     = " + string(beta_tau));
377             dbprint(p, "// beta_tau_bar = " + string(beta_tau_bar));
378             dbprint(p, "//     bracket contains duplicate entries");
```

```
379         }
380     }
381     return (vdWs);
382 }
383 example
384 { "EXAMPLE:"; printlevel = 1; echo = 2;
385     def Kp = plueckerCoordRing(2,4);
386     setring Kp;
387     list alpha = list(); // empty list
388     list beta = 1,2,3;
389     list gamma = 4;
390     print(vanDerWaerdenSyzygy(alpha, beta, gamma));
391 }
392 ////////////////////////////////////////////////////////////////////
393 /**
394  * Computes the Pluecker coordinate ring with the the Pluecker ideal given via a Grobener
395  * basis in it
396  *
397  * Usage:
398  *     def Kp = grassmanianGB(2, 4);
399  *     setring Kp;
400  *     Irm;
401  *
402  * @param r a natural number
403  * @param m a natural number >= m
404  * @returns The Pluecker coordinate ring with the Pluecker ideal called "Irm" in it
405  */
406 proc grassmanianGB (int r, int m)
407 "USAGE:   grassmanianGB(r, m); r, m natural numbers with m @math{\geq r} r
408 PURPOSE: Compute a dp-Groebner basis for the Pluecker ideal
409          using the van Der Waerden syzygies
410 RETURN:  The Pluecker coordinate ring with the Pluecker ideal
411          given via a Groebner basis called \"Irm\" in it
412 NOTE:    The generators of the ideal will form
413          a Groebner basis with respect to dp / the tableaux order.
414          @* The procedure returns a ring with the ideal inside,
415          see the enclosed example.
416 EXAMPLE: example grassmanianGB; shows an example"
417 {
418     def Kp = plueckerCoordRing(r, m);
419     setring Kp;
420
421     ideal Irm;
422
423     list L = subsets(r, m);
424
425     int i;
426     int j;
427
428     list R;
429     list S;
430
431     list beta;
432     list beta1;
433     list beta2;
434     list alpha;
435     list gamma;
436
437     int s;
438
439     /** output will be printed if p > 0 */
440     int p = printlevel - voice +3;
441
442     for (i = 1; i <= size(L); i++)
```

```
443     {
444         // only check sets S with S > R
445         for (j = i+1; j <= size(L); j++)
446         {
447             R = L[i]; // alpha and last part of beta
448             S = L[j]; // first part of beta and gamma
449
450             s = firstLargerPosition(R, S);
451
452             if (s != 0)
453             {
454                 // if we just write "beta = S[1 .. s]; + R[s .. (r+1)]",
455                 // Singular adds componentwise ???
456                 beta1 = S[1 .. s];
457                 beta2 = R[s .. r];
458                 beta = beta1 + beta2;
459
460
461                 if (s == 1)
462                 {
463                     // workaround because R[1 .. 0] is illegal
464                     alpha = list();
465                 }
466                 else
467                 {
468                     alpha = R[1 .. (s-1)];
469                 }
470
471                 if (s+1 > r)
472                 {
473                     // see above
474                     gamma = list();
475                 }
476                 else
477                 {
478                     gamma = S[(s+1) .. r];
479                 }
480
481                 dbprint (p, "// Splitted");
482                 dbprint (p, "//     R = " + string (R));
483                 dbprint (p, "//     S = " + string (S));
484                 dbprint (p, "// into");
485                 dbprint (p, "//     a = " + string (alpha));
486                 dbprint (p, "//     b = " + string (beta));
487                 dbprint (p, "//     c = " + string (gamma));
488                 dbprint (p, "// [[a (dot b) c] = ");
489                 dbprint (p, "//     "
490                     + string(vanDerWaerdenSyzygy(alpha, beta, gamma)));
491
492                 Irm = Irm + vanDerWaerdenSyzygy(alpha, beta, gamma);
493             }
494
495         }
496     }
497
498     dbprint (p, "// This method returns a ring containing the ideal \"Irm\"");
499     dbprint (p, "// USAGE: def Kp = grassmanianGB(_,_); setring Kp; Irm;");
500
501     attrib(Irm,"isSB",0);  // we know that Irm is a dp-Groebner basis
502
503     export (Irm);
504     return (Kp);
505 }
506 example
```

```
{ "EXAMPLE:"; printlevel = 1; echo = 2;
    def Kp = grassmanianGB(2, 4);
    setring Kp;
    print (Irm);
}
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
// STATIC HELPER PROCEDURES
////////////////////////////////////////////////////////////////////
/**
 * @param r a natural number
 * @param m a natural number
 * @param start a natural number
 * @returns a list of all subsets of {start, ..., m} of size == r (as list of lists)
 */
static proc subsetsStartingFrom (int r, int m, int start)
{
    list result;

    if (r == 0)
    {
        result[1] = list();
        return (result);
    }

    if (m - start < 0)
    {
        return (result);
    }

    list without = subsetsStartingFrom(r,   m, start+1);
    list with    = subsetsStartingFrom(r-1, m, start+1);

    for (int i = 1; i <= size(with); i++)
    {
        with[i] = list(start) + with[i];
    }

    result = with + without;

    return (result);
}
////////////////////////////////////////////////////////////////////
/**
 * Gives the variable name corresponding to a subset as string
 * @param subset a list of natural numbers
 * @returns a string of form p_v1_..._vr where the vi are the entries of subset
 */
static proc subsetToVarstring (list subset)
{
    string s = "p";
    for (int i = 1; i <= size(subset); i++)
    {
        s = s + "_" + string(subset[i]);
    }
    return (s);
}
////////////////////////////////////////////////////////////////////
/**
 * Warning: a ring with the correct variables has to be the active basering!
 * @param subset a list of natural numbers
 * @returns a polynomial p_v1_..._vr where the vi are the entries of subset
 */
static proc subsetToVar (list subset)
```

```
{
    execute ("poly f = " + subsetToVarstring(subset));
    return (f);
}
////////////////////////////////////////////////////////////////////
/**
 * Inserts an element into a sorted list.
 * @param L a list of natural numbers, sorted in ascending order
 * @param j the element which should be inserted
 * @returns (L', i)
 *           L' a new list containing the elements of L and j, sorted in ascending order
 *           i the index at which j was inserted (or 0 if i was already contained in L)
 */
static proc insertSorted (list L, int j)
{
    int i = 0;

    // variables for binary search
    int lb = 1;
    int ub = size(L);
    int mid = (lb + ub) div 2;
    int oldmid = -1;

    // do binary search
    while (1)
    {
        if (oldmid == mid ||  mid > size(L) || mid < 1)
        {
            return (insert(L, j, mid), mid+1);
        }

        if (L[mid] == j)
        {
            return (L, 0);
        }

        if (L[mid] < j)
        {
            lb = mid + 1;
            oldmid = mid;
            mid = (lb + ub) div 2;
        }
        if (L[mid] > j)
        {
            ub = mid - 1;
            oldmid = mid;
            mid = (lb + ub) div 2;
        }
    }
}
////////////////////////////////////////////////////////////////////
/**
 * @param alpha a list of natural numbers, sorted in ascending order
 * @param beta a list of natural numbers, sorted in ascending order
 * @returns (i, L)
 *           i = 0 iff the "intersection" of alpha and beta is nonempty,
 *               and the sign of the permution which sorts alpha++beta otherwise
 *           L = sort(alpha + beta)
 */
static proc concatSorted (list alpha, list beta)
{
    int a = 1;
    int b = 1;

```

```
635     int sgn_exponent = 0;
636     int sgn = 1;
637
638     list result;
639     for (int i = 1; i <= size(alpha) + size(beta); i++)
640     {
641         if (a > size(alpha))
642         {
643             result[i] = beta[b];
644             b++;
645         }
646         else { if (b > size (beta))
647         {
648             result[i] = alpha[a];
649             a++;
650         }
651         else { if (alpha[a] == beta[b])
652         {
653             return (0, result);
654         }
655         else { if (alpha[a] < beta[b])
656         {
657             result[i] = alpha[a];
658             a++;
659         }
660         else
661         {
662             result[i] = beta[b];
663             sgn_exponent = sgn_exponent + size(alpha) - a + 1;
664             b++;
665         }}}}
666     }
667
668     if (sgn_exponent % 2 != 0)
669     {
670         sgn = -1;
671     }
672
673     return (sgn, result);
674 }
675 ////////////////////////////////////////////////////////////////////
676 /**
677  * @param r a list of natural numbers
678  * @param s a list of natural numbers of the same size as r
679  * @returns the first index such that r[i] > s[i] or 0 if no such index exists
680  */
681 static proc firstLargerPosition (list r, list s)
682 {
683     for (int i = 1; i <= size(r); i++)
684     {
685         if (r[i] > s[i])
686         {
687             return (i);
688         }
689     }
690     return (0);
691 }
692 ////////////////////////////////////////////////////////////////////
693 /**
694  * @param tau a list
695  * @param beta a list which contains tau as sublist
696  * @returns the list of entries which are contained in beta but not in tau
697  */
698 static proc complementOfSet(list tau, list beta)
```

```
699 {
700     list tau_bar;
701
702     int taui = 1;
703     int taun = 1;
704
705     for (int i = 1; i <= size(beta); i++)
706     {
707
708         if (tau[taui] == beta[i])
709         {
710             if (taui < size(tau))
711             {
712                 taui++;
713             }
714         }
715         else
716         {
717             tau_bar[taun] = beta[i];
718             taun++;
719         }
720     }
721     return (tau_bar);
722 }
723 ////////////////////////////////////////////////////////////////////
724 /**
725  * @param tau, a subset of {1, ..., t}
726  * @param tau_complement the complement of tau in {1, ..., t}
727  * @returns sgn(tau, tau_complement)
728  */
729 static proc signOfShuffle (list tau, list tau_complement)
730 {
731     int sgn_exponent = 0;
732     int a = 1;
733     int b = 1;
734     while (a <= size(tau) && b <= size(tau_complement))
735     {
736         // this should never happen
737         if (tau[a] == tau_complement[b])
738         {
739             return (0);
740         }
741         if (tau[a] < tau_complement[b])
742         {
743             a++;
744         }
745         else
746         {
747             sgn_exponent = sgn_exponent + size(tau) - a + 1;
748             b++;
749         }
750     }
751     if (sgn_exponent % 2 == 0)
752     {
753         return (1);
754     }
755     else
756     {
757         return (-1);
758     }
759 }
760 ////////////////////////////////////////////////////////////////////
```

B. tropicalboundaries.lib

This section contains the code implementing the algorithms studied in the fourth part of this thesis to compute boundary of tropical varieties starting from a set of homogeneous generators for an defining ideal.

B.1 Library overview

Library: `tropicalboundaries.lib`
Purpose: Compute the boundary of tropical varieties
Category: Tropical geometry
Author: Sebastian Muskalla (muskalla@mathematik.uni-kl.de)
References: This thesis
Procedures:

`intersectAndProject(I, #)`	$\langle I, x_i \mid i \in \# \rangle \cap K[x_i, i \notin \#]$
`boundaryViaElimination(I, #)`	Boundary $\mathrm{Bound}^{e}_{\#}(I)$
`boundaryViaProjection(I, #)`	Boundary $\mathrm{Bound}^{p}_{\#}(I)$

B.2 Exported procedures

The following procedures are available after the library and `gfanlib.so` have been loaded.

- **intersectAndProject**

  Usage: `intersectAndProject(I, #);`
  I homogeneous ideal, # list of variable indices (natural numbers) or variables (polynomials)

  Assume: The current basering uses a global or local ordering listed in the points 1. or 2. in the SINGULAR documentation on monomial orderings or one of those with an extra weight vector

  Purpose: Compute the "elimination ideal" $\langle I, x_i \mid i \in \# \rangle \cap K[x_i \mid i \notin \#]$

  Return: A ring containing the "elimination ideal" called E

  Note: The result of this procedure should be stored in a variable of type def.
  On the geometric side, the ideal returned by this procedure corresponds to the algebraic set we get by the following procedure:
  1. Intersect $V(I)$ with all hyperplanes H_i where i in #.
  2. Project the resulting set to the space not containing the directions in #.
  3. Take the closure in the Zariski topology.

  Example:

```
> example intersectAndProject;
// proc intersectAndProject from lib tropicalboundaries.lib
EXAMPLE:
    ring R = 0,(x,y,z,q,r,s),dp;
    ideal I = xy + zq + rs;
    // second argument should either be a list of variable indices...
    def R1 = intersectAndProject(I, 1);
// This procedure returns a ring containg the ideal "E"
// Usage: def R = intersectAndProject(I); setring R; E;
    setring (R1);
    print (E);
zq+rs
    // ... or a list of variables as polynomials
    setring (R);
    def R2 = intersectAndProject(I, x);
// This procedure returns a ring containg the ideal "E"
// Usage: def R = intersectAndProject(I); setring R; E;
    setring (R1);
    print (E);
zq+rs
```

- **boundaryViaElimination**

  Usage: `boundaryViaElimination(I, #);`
  I homogeneous ideal, # list of variable indices (natural numbers) or variables (polynomials)

  Assume: The current basering uses a global or local ordering listed in the points 1. or 2. in the SINGULAR documentation on monomial orderings or one of those with an extra weight vector

  Purpose: Compute the boundary via elimination $\mathrm{Bound}^{e}_{\#}(I)$ in the directions given by #

  Return: The boundary as fan

  Example:

```
> example boundaryViaElimination ;
// proc boundaryViaElimination from lib tropicalboundaries.lib
EXAMPLE:
    ring R = 0,(x,y,z,q,r,s),dp;
    ideal I = xy + zq + rs;

    print (boundaryViaElimination(I, 1));
_application PolyhedralFan
_version 2.2
_type PolyhedralFan

AMBIENT_DIM
5

DIM
4

LINEALITY_DIM
4

RAYS

N_RAYS
0

LINEALITY_SPACE
-1 0 0 0 0      # 0
0 1 0 0 1       # 1
0 0 -1 0 -1     # 2
0 0 0 1 -1      # 3

ORTH_LINEALITY_SPACE
0 1 1 -1 -1     # 0

F_VECTOR
1

SIMPLICIAL
1

PURE
1

CONES
{}      # Dimension 4
```

```
46
47 MAXIMAL_CONES
48 {}      # Dimension 4
49
50 MULTIPLICITIES
51 1       # Dimension 4
52
53     // one can also use a list variables as the second argument:
54     // boundaryViaElimination(I, x);
55     // yields the same result.
```

- **boundaryViaProjection**

  Usage: `boundaryViaProjection(IorF, #);`
  *I*or*F* homogeneous ideal or fan, # list of variable indices (natural numbers) or variables (polynomials)

  Assume: The current basering uses a global or local ordering listed in the points 1. or 2. in the SINGULAR documentation on monomial orderings or one of those with an extra weight vector

  Purpose: Compute the boundary via projection $\text{Bound}^p_{\#}(I)$ in the directions given by #

  Return: The boundary as fan

  Note: If an ideal is passed as first argument, the procedure starts by computing its tropicalization. $\text{Bound}^p_i(I)$ only depends on $\text{Trop}(I)$.
  The result coincides with the result of `boundaryViaElimination` if the ideal is saturated with respect to the product of the ring variables.

  Example:

```
1 > example boundaryViaProjection ;
2 // proc boundaryViaProjection from lib tropicalboundaries.lib
3 EXAMPLE:
4     ring R = 0,(x,y,z,q,r,s),dp;
5     ideal I = xy + zq + rs;
6     fan F = tropicalVariety(I);
7     print (boundaryViaProjection(F, 1));
8 _application PolyhedralFan
9 _version 2.2
10 _type PolyhedralFan
11
12 AMBIENT_DIM
13 5
14
15 DIM
16 4
17
18 LINEALITY_DIM
19 4
20
21 RAYS
22
23 N_RAYS
24 0
25
26 LINEALITY_SPACE
27 -1 0 0 0 0      # 0
```

```
0 1 0 0 1       # 1
0 0 -1 0 -1     # 2
0 0 0 1 -1      # 3

ORTH_LINEALITY_SPACE
0 1 1 -1 -1     # 0

F_VECTOR
1

SIMPLICIAL
1

PURE
1

CONES
{}      # Dimension 4

MAXIMAL_CONES
{}      # Dimension 4

MULTIPLICITIES
1       # Dimension 4
```

```
    // one can also use a ideal as the first arguement
    // or/and  use a list of variables as polynomials as second argument:
    // boundaryViaProjection(I, x);
    // yields the same result.
```

B.3 Source code

Listing 2: tropicalboundaries.lib

```
1  ////////////////////////////////////////////////////////////////////
2  version="version tropicalboundaries.lib 4.0.0.0 Apr_2015 "; // $Id: $
3  category="Tropical Geometry";
4  info="
5  LIBRARY:   tropicalboundaries.lib  Compute the boundary of tropical varieties
6  AUTHOR:    Sebastian Muskalla, email: muskalla@mathematik.uni-kl.de
7
8  KEYWORDS: tropic; tropical; boundary; tropicalization;
9
10 REFERENCES:
11 [1] Sebastian Muskalla: Computing the boundaries of tropical varieties,@*
12 Master thesis, TU Kaiserslautern (2015)
13
14 PROCEDURES:
15 intersectAndProject(I, #);        <I, vars(#)> intersected with K[x, x not in #]
16 boundaryViaElimination(I, #);     boundary via elimination in the directions in #
17 boundaryViaProjection(IorF, #);   boundary via projection in the directions in #
18 ";
19 ////////////////////////////////////////////////////////////////////
20 // DEPENDENCIES
21 ////////////////////////////////////////////////////////////////////
22 LIB "gfanlib.so";
23 ////////////////////////////////////////////////////////////////////
24 ////////////////////////////////////////////////////////////////////
25 // EXPORTED PROCEDURES
26 ////////////////////////////////////////////////////////////////////
27 /**
28  * WARNING:
29  * Weigth vector projection works only for the following term orders:
30  *  1. Global orderings
31  *  2. Local orderings
32  *  4. Extra weight vector (where a is a term ordering from 1. or 2.)
33  *
34  * @param I homogeneous ideal
35  * @param # list of variable indices (natural numbers) or variables (polynomials)
36  * @returns the ideal <I, V> \cup R where V is the ideal generated by the variables
37  *      given by the list #
38  *      and R is the subring of the basering without the variables in #
39  */
40 proc intersectAndProject (ideal I, list #)
41 "USAGE:   intersectAndProject(I, #);
42          @* I homogeneous ideal
43          @* # list of variable indices (natural numbers) or variables (polynomials)
44 ASSUME:  the current basering uses a monomial ordering of one of the following types:
45          @* 1. Global orderings
46          @* 2. Local orderings
47          @* 4. Extra weight vector (where a is a term ordering from 1. or 2.)
48          @* (numbers correspond to the Singular documentation
49          for monomial orderings)
50 PURPOSE: compute the ideal <I, V> intersected with the subring not containing
51          @* the variables in #, where V is the ideal generated by the variables in #
52 RETURN:  a ring containing the \"elimination ideal\" called \"E\"
53 NOTE:    The result of this procedure should be stored in a variable of type def.
54          @* On the geometric side, the ideal returned by this procedure corresponds to
55          the algebraic set we get by the following procedure:
56          @* 1. intersect @math{V(I)} with all hyperplanes @math{H_i} where i in #
57          @* 2. project the resulting set to the space not containing the directions in #
58          @* 3. take the closure in the Zariski topology
```

```
EXAMPLE: example intersectAndProject; shows an example"
{
    /** output will be printed if p > 0 */
    int p = printlevel - voice +3;

    dbprint (p, "// This procedure returns a ring containg the ideal \"E\"");
    dbprint (p, "// Usage: def R = intersectAndProject(I); setring R; E;");

    // handle the empty list
    if (size (#) == 0)
    {
        ideal E = I;
        export (E);
        return (basering);
    }

    // check type of first list entry
    if (typeof(#[1]) == "int")
    {
        return (intersectAndProjectInds(I, #));
    }
    if (typeof(#[1]) == "poly")
    {
        return (intersectAndProjectVars(I, #));
    }

    // all other types should not occur
    ERROR ("intersectAndProject: 2nd argument should be a list of ints"
            + "or a list of variables");
}
example
{ "EXAMPLE:"; printlevel = 1; echo = 2;
    ring R = 0,(x,y,z,q,r,s),dp;
    ideal I = xy + zq + rs;
    // second argument should either be a list of variable indices...
    def R1 = intersectAndProject(I, 1);
    setring (R1);
    print (E);
    // ... or a list of variables as polynomials
    setring (R);
    def R2 = intersectAndProject(I, x);
    setring (R1);
    print (E);
}
////////////////////////////////////////////////////////////////////
/**
 * @param I homogeneous ideal
 * @param # list of variable indices (natural numbers) or variables (polynomials)
 * @return the fan representing the boundary via elimination in the directions
 *      corresponding to the variables in #
 *
 * WARNING:
 * Weigth vector projection works only for the following term orders:
 *  1. Global orderings
 *  2. Local orderings
 *  4. Extra weight vector (where a is a term ordering from 1. or 2.)
 */
proc boundaryViaElimination (ideal I, list #)
"USAGE:   boundaryViaElimination(I, #);
         @* I homogeneous ideal
         @* # list of variable indices (natural numbers) or variables (polynomials)
ASSUME:  the current basering uses a monomial ordering of one of the following types:
         @* 1. Global orderings
         @* 2. Local orderings
```

```
123          @* 4. Extra weight vector (for a term ordering from 1. or 2.)
124          @* (numbers correspond to the numbers in the Singular documentation
125          for monomial orderings)
126 PURPOSE: compute the boundary via elimination in the directions given by #
127 RETURN:  the boundary as fan
128 EXAMPLE: example boundaryViaElimination; shows an example"
129 {
130     def Ering = intersectAndProject(I, #);
131     setring Ering;
132     return (tropicalVariety(E));
133 }
134 example
135 { "EXAMPLE:"; printlevel = 1; echo = 2;
136     ring R = 0,(x,y,z,q,r,s),dp;
137     ideal I = xy + zq + rs;
138     print (boundaryViaElimination(I, 1));
139     // one can also use a list variables as the second argument:
140     // boundaryViaElimination(I, x);
141     // yields the same result.
142 }
143 ////////////////////////////////////////////////////////////////////
144 /**
145  * @param IorF ideal or fan
146  * @param # list of variable indices (natural numbers) or variables (polynomials)
147  * @return the fan representing the boundary via projection in the directions
148  *      corresponding to the variables in #
149  */
150 proc boundaryViaProjection (IorF, list #)
151 "USAGE:   boundaryViaProjection(IorF, #);
152          @* IorF homogeneous ideal or fan
153          @* # list of variable indices (natural numbers) or variables (polynomials)
154 PURPOSE: compute the boundary via projection in the directions given by #
155 RETURN:  the boundary as fan
156 NOTE:    If an ideal is passed as first argument, the procedure starts by computing
157          its tropicalization.
158          @* The result coincides with the result of \"boundaryViaElimination"\
159          if the ideal is saturated with respect to the product of the ring variables.
160 EXAMPLE: example boundaryViaProjection; shows an example"
161 {
162     fan F;
163
164     // check type of first argument
165     if (typeof(IorF) == "fan")
166     {
167         F = IorF;
168     }
169     else { if (typeof(IorF) == "ideal")
170     {
171         F = tropicalVariety(IorF);
172     }
173     else
174     {
175         // all other types should not occur
176         ERROR ("boundaryViaProjection: 1st argument should be an ideal or a fan");
177     }}
178
179     // handle the empty list
180     if (size(#) == 0)
181     {
182         return (F);
183     }
184
185     // check type of first list entry
186     if (typeof(#[1]) == "int")
```

```
187     {
188         return (boundaryViaProjectionInds(F, #));
189     }
190     if (typeof(#[1]) == "poly")
191     {
192         return (boundaryViaProjectionInds(F, varIndices(#)));
193     }
194     // all other types should not occur
195     ERROR ("boundaryViaProjection: 2nd argument should be a list of ints"
196     + "or a list of variables");
197 }
198 example
199 { "EXAMPLE:"; printlevel = 1; echo = 2;
200     ring R = 0,(x,y,z,q,r,s),dp;
201     ideal I = xy + zq + rs;
202     fan F = tropicalVariety(I);
203     print (boundaryViaProjection(F, 1));
204     // one can also use a ideal as the first arguement
205     // or/and  use a list of variables as polynomials as second argument:
206     // boundaryViaProjection(I, x);
207     // yields the same result.
208 }
209 ////////////////////////////////////////////////////////////////////
210 ////////////////////////////////////////////////////////////////////
211 // STATIC HELPER PROCEDURES
212 ////////////////////////////////////////////////////////////////////
213 /**
214  * @param lst some list (or ideal) of elements
215  *              such that they are comparable via == with elem
216  * @param elem some element
217  * @returns 1 iff there is some i with lst[i] == elem
218  */
219 static proc elem (lst, elem)
220 {
221     for (int i = 1; i <= size(lst); i++)
222     {
223         if (lst[i] == elem)
224         {
225             return (1);
226         }
227     }
228     return (0);
229 }
230 ////////////////////////////////////////////////////////////////////
231 /**
232  * @param lst some list of variables (as polynomials)
233  * @returns the list of corresponding indices
234  */
235 static proc varIndices (lst)
236 {
237     int i;
238     int j;
239     int found;
240     list indices;
241     for (i = 1; i <= size(lst); i++)
242     {
243         found = 0;
244         for (j = 1; j <= nvars(basering) && found == 0; j++)
245         {
246             if (lst[i] == var(j))
247             {
248                 indices[i] = j;
249                 found = 1;
250             }
```

```
251         }
252     }
253     return (indices);
254 }
255 ////////////////////////////////////////////////////////////////////
256 /**
257  * Eliminates a list of variables from an ideal
258  *
259  * @param I homogeneous ideal
260  * @param # a list of variables as polynomials that should be eliminated.
261  * @returns The elimination ideal
262  */
263 static proc addAndEliminate (ideal I, list #)
264 {
265     // nothing to do if list is empty
266     if (size(#) == 0)
267     {
268         return (std(I));
269     }
270
271     // add variables to ideal ("intersection")
272     // and compute the product of the variables
273     ideal I2 = I;
274     poly varprod = 1;
275
276     for (int j = 1; j <= size(#); j++)
277     {
278         I2 = I2, #[j];
279         varprod = varprod * #[j];
280     }
281
282     if (homog(I))
283     {
284         // compute a dp Groebner basis
285         // and use the hilbert function to speed up elimination
286         // if the ideal is homogeneous
287         def Rbase = basering;
288         execute ("ring Rdp = "
289         + string(ringlist(Rbase)[1]) + ",(" + string(varstr(basering)) + "),dp");
290         setring Rdp;
291
292         ideal Idp = imap (Rbase, I2);
293         Idp = std(Idp);
294         intvec hilbvec = hilb(Idp, 1);
295         setring Rbase;
296         return (std (eliminate(I2, varprod, hilbvec)));
297     }
298     {
299         // "slow" elimination if I is not homogeneous
300         return (std (eliminate(I2, varprod)));
301     }
302 }
303 ////////////////////////////////////////////////////////////////////
304 /**
305  * @param I ideal
306  * @param # list of variable indices (natural numbers)
307  *
308  * Computes the polynomials corresponding the the variables with indices in #, then calls
309  * @see intersectAndProjectBoth
310  */
311 static proc intersectAndProjectInds (ideal I, list #)
312 {
313     // find variables corresponding to the polynomials
314     list polys;
```

```
315
316     for (int i = 1; i <= size(#); i++)
317     {
318         polys = polys + list(var(#[i]));
319     }
320     return (intersectAndProjectBoth(I, #, polys));
321 }
322 ////////////////////////////////////////////////////////////////////
323 /**
324  * @param I ideal
325  * @param # list of variables (polynomials)
326  *
327  * Computes the variable indices of the variables in #, then calls
328  * @see intersectAndProjectBoth
329  */
330 static proc intersectAndProjectVars (ideal I, list #)
331 {
332     return (intersectAndProjectBoth(I, varIndices(#), #));
333 }
334 ////////////////////////////////////////////////////////////////////
335 /**
336  * Adds variables to an ideal (geometric intersection) and intersects it with the subring
337  * not containing these variables (geometric projection).
338  *
339  * WARNING:
340  * Weigth vector projection works only for the following term orders:
341  *  1. Global orderings
342  *  2. Local orderings
343  *  4. Extra weight vector (where a is a term ordering from 1. or 2.)
344  *
345  * @param I homogeneous ideal
346  * @param indices list of variable indices
347  * @param polys list of corresponding polynomials
348  *                  (i.e. for all i: var(indices[i]) == polys[i]
349  * @returns The ring without the eliminated variables
350  *              with the elimination ideal called "E" inside it
351  */
352 static proc intersectAndProjectBoth (ideal I, list indices, list polys)
353 {
354     def R = basering;
355     list ringparams = ringlist(R);
356
357     intvec var_weights = ringparams[3][1][2];
358     intvec new_var_weights;
359
360     list vars_as_strings = ringparams[2];
361     int n = size(vars_as_strings);
362     list new_vars_as_strings;
363
364     // compute the "elimination ideal"
365     ideal Ie = addAndEliminate(I, polys);
366
367     // create the variable list for the new ring
368     int remove_this;
369     int h = 1;
370
371     for (int l = 1; l <= n; l++)
372     {
373         if (elem(indices, l) == 0)
374         {
375             new_vars_as_strings = new_vars_as_strings + list(vars_as_strings[l]);
376             new_var_weights[h] = var_weights[l];
377             h++;
378         }
```

```
379     }
380     if (size(new_vars_as_strings) < 1)
381     {
382         ERROR ("intersectAndProjectBoth: New ring has no variables!");
383     }
384
385     // create ring with new variable list
386     ringparams[2] = new_vars_as_strings;
387     ringparams[3][1][2] = new_var_weights;
388     def R_elim = ring(ringparams);
389     setring R_elim;
390
391     // map "elimination ideal" to the new ring ("projection")
392     ideal E = imap(R,Ie);
393     E = std(E);
394
395     export (E);
396     return (R_elim);
397 }
398 ////////////////////////////////////////////////////////////////////
399 /**
400  * @param n dimension of the vector space
401  * @param # list of indices
402  * @returns size(#) x n matrix where the i-th column is 0 but -1 at position (i, #[i])
403  */
404 static proc negative_basis_vectors (int n, list #)
405 {
406     int m = size(#);
407     intmat vs[m][n];
408     for (int i = 1; i <= m; i++)
409     {
410         vs[i,#[i]] = -1;
411     }
412     return (vs);
413 }
414 ////////////////////////////////////////////////////////////////////
415 /**
416  * @param m a matrix
417  * @param # a list of column indices
418  * @returns a matrix containing the columns with column index not in #
419  */
420 static proc copyButColumns(intmat m, list #)
421 {
422     int rows = nrows(m);
423     int cols = ncols(m);
424     int count_remove = size(#);
425
426     intmat res [rows][cols-count_remove];
427     int i;
428     int j;
429     int skipped;
430     for (i = 1;  i <= rows; i++)
431     {
432         skipped = 0;
433         for (j = 1; j <= cols; j++)
434         {
435             if (elem(#, j))
436             {
437                 skipped++;
438             }
439             else
440             {
441                 res[i,j-skipped] = m[i,j];
442             }
```

```
443         }
444     }
445     return (res);
446 }
447 ////////////////////////////////////////////////////////////////////
448 /**
449  * @param c a cone
450  * @param vs a matrix where each row represents a vector
451  * @returns 1 iff there vs[i] is in the support of c for all i
452  */
453 static proc containsAllInSupport(cone c, intmat vs)
454 {
455     intvec v;
456     for (int i = 1; i <= nrows(vs); i++)
457     {
458         v = vs[i,1 .. ncols(vs)];
459         if (!containsInSupport(c,v))
460         {
461             return (0);
462         }
463     }
464     return (1);
465 }
466 ////////////////////////////////////////////////////////////////////
467 /**
468  * @param F fan
469  * @param # list of indices in {1, ..., ambientDimension(F)}
470  * @returns the fan { proj(C) | C in F, -e_i in C for all i in # }
471  */
472 static proc boundaryViaProjectionInds (fan F, list #)
473 {
474     int elim_count = size(#);
475     int ambient = ambientDimension (F);
476
477     // construct new fan
478     fan proj_F = emptyFan(ambient - elim_count);
479
480     /** zero matrix */
481     intmat empt[1][ambient - elim_count];
482
483     /** matrix containing -e_i for all i in # */
484     intmat minus_ei_s = negative_basis_vectors(ambient, #);
485
486     /** number of cones of a certain dimension */
487     int nr_cones;
488
489     /** loop counter for dimension */
490     int i; //
491
492     /** loop counter for cone index */
493     int j;
494
495     cone c;
496     cone proj_c;
497
498     intmat halflines;
499     intmat lines;
500
501     for (i = 1; i <= ambient; i++)
502     {
503         nr_cones = numberOfConesOfDimension (F, i, 0, 0);
504
505         for (j = 1; j <= nr_cones; j++)
506         {
```

```
507             c = getCone(F, i, j);
508
509             if (containsAllInSupport(c, minus_ei_s))
510             {
511                 // find halflines generating the projected cone
512                 halflines = rays(c);
513                 if (nrows(halflines) > 0)
514                 {
515                     halflines = copyButColumns(halflines, #);
516                 }
517                 else
518                 {
519                     halflines = empt;
520                 }
521
522                 // find lines generating the projected cone
523                 if (linealityDimension(c) > 0)
524                 {
525                     lines = generatorsOfLinealitySpace(c);
526                     lines = copyButColumns(lines, #);
527                     proj_c = coneViaPoints(halflines, lines);
528                 }
529                 else
530                 {
531                     proj_c = coneViaPoints(halflines);
532                 }
533
534                 if (dimension(proj_c) > 0)
535                 {
536                     // we do not need to check compatiblity
537                     insertCone(proj_F, proj_c, 0);
538                 }
539             }
540         }
541     }
542     return (proj_F);
543 }
544 ////////////////////////////////////////////////////////////////////////
```

C. test_tropicalboundaries.c

The following code contains procedures to study whether the results of the two algorithms presented in the fourth part are equal. It was written before the theoretic results presented in section 10 were developed, but it may continue to be useful to study the boundary for non-saturated ideals.

Listing 3: test_tropicalboundaries.c

```
1  LIB "gfanlib.so";
2  LIB "random.lib";
3  LIB "tropicalboundaries.lib";
4  LIB "grassmanian.lib"; // for the subsets method
5
6  /**
7   * Checks whether two fans are equal. Note that by definition,
8   * it is sufficient to check that the maximal cones coincide.
9   *
10  * @param F
11  * @param F2
12  * @returns 1 iff F == F2 (as set of cones)
13  */
14 proc fansEqual (fan F, fan F2)
15 {
16     if (ambientDimension(F) != ambientDimension(F2))
17     {
18         return (0);
19     }
20
21     if (nmaxcones(F) != nmaxcones(F2))
22     {
23         return (0);
24     }
25
26     int ambient = ambientDimension (F2);
27     int nrc;
28     int i;
29     int dimi;
30     cone c;
31
32     for (dimi = ambient; dimi >= 1; dimi--)
33     {
34         nrc = numberOfConesOfDimension (F2, dimi, 0, 1);
35
36         for (i = 1; i <= nrc; i++)
37         {
38             c = getCone(F2, dimi, i);
39
40             if (! containsInCollection(F,c) )
41             {
42                 return (0);
43             }
44         }
45     }
46     return (1);
47 }
48
49
50 /**
51  * Tests for all subsets of directions # whether boundaryViaElimination(I, #) and
52  * boundaryViaProjection(I, #) yield the same result
```

```
53  *
54  * @param I ideal
55  */
56 proc testAllBoundaries(ideal I)
57 {
58     fan F = tropicalVariety (I);
59     return (testAllBoundariesWithGivenFan(I, F));
60 }
61
62 /**
63  * Tests for the sets given by the entries of the second parameter whether
64  * boundaryViaElimination(I, _) and boundaryViaProjection(I, _) yield the same result
65  *
66  * @param I ideal
67  * @param indicies list of lists where each inner list represents a set of directions.
68  *      The inner lists do not have to be sorted, but they should not contain duplicate
69  *      entries.
70  */
71 proc testBoundaries(ideal I, list indices)
72 {
73     return (testBoundariesWithGivenFan(I, tropicalVariety (I), indices));
74 }
75
76 /**
77  * Just as testAllBoundaries, but the fan used for "boundaryViaProjection" is passed as
78  * parameter to avoid a time-consuming computation via "tropicalVariety(I)"
79  *
80  * @param I ideal
81  * @param F fan which should be equal to tropicalVariety(I)
82  */
83 proc testAllBoundariesWithGivenFan(ideal I, fan F)
84 {
85     return (testBoundariesWithGivenFan(I, F,
86                 subsets (nvars(basering), nvars(basering), 1)));
87 }
88
89 /**
90  * Just as testBoundaries, but the fan used for "boundaryViaProjection" is passed as
91  * parameter to avoid a time-consuming computation via "tropicalVariety(I)"
92  *
93  * @param I ideal
94  * @param F fan which should be equal to tropicalVariety(I)
95  * @param indicies list of lists where each inner list represents a set of directions.
96  *      The inner lists do not have to be sorted, but they should not contain duplicate
97  *      entries.
98  */
99 proc testBoundariesWithGivenFan (ideal I, fan F, list indices)
100 {
101     fan F_boundary;
102     fan F_vanish;
103     list varinds;
104
105     int nr = size(indices);
106     print("// Checking " + string(nr) + " combinations for equality:");
107
108     for (int i = 1; i <= nr; i++)
109     {
110         varinds = indices[i];
111
112         // if varinds contains all variables, nothing is left and the methods to compute
113         // the boundary will return errors.
114         if (size(varinds) == nvars(basering))
115         {
116             print ("// TRIVIALLY EQUAL");
```

```
117         }
118         else
119         {
120             print ("// Checking " + string(varinds));
121             F_vanish   = boundaryViaElimination (I, varinds);
122             F_boundary = boundaryViaProjection  (F, varinds);
123 
124             if (!fansEqual(F_vanish, F_boundary))
125             {
126                 print ("// NOT EQUAL:");
127                 print ("// Indices were:" + string(varinds));
128                 print ("// ");
129                 print ("// F_vanish:");
130                 print ("//  ");
131                 print (F_vanish);
132                 print ("// ");
133                 print ("// F_boundary:");
134                 print ("// ");
135                 print (F_boundary);
136                 print ("// ");
137                 print ("// STOP");
138                 return (0);
139             }
140             print ("// EQUAL");
141             print ("// " + string((i * 100) div nr ) + "% DONE");
142         }
143     }
144     print ("// EQUAL FOR ALL COMBINATIONS!");
145     return (1);
146 }
147 
148 /**
149  * Creates a random homogeneous ideal in the ring with the specified number of variables
150  * and calls testAllBoundaries on it.
151  * @param varcount number of variables
152  */
153 proc rndTest(int varcount)
154 {
155     int i;
156     string varstring = "";
157     string weightstring = "";
158     for (i = 1; i <= varcount; i++)
159     {
160         varstring = varstring + "x" + string(i);
161         weightstring = weightstring + string (1);
162         if (i != varcount)
163         {
164             varstring = varstring + ",";
165             weightstring = weightstring + ",";
166         }
167     }
168     string ringstring = "ring RND = 0,(" + varstring + "),wp(" + weightstring + ");";
169     execute(ringstring);
170     ideal I = sparseHomogIdeal(5, 3,5,  75,  100);
171 
172     print ("// Ideal:");
173     print (I);
174     fan F = tropicalVariety(I);
175     print ("// Tropicalization:");
176     print (F);
177     print ("// Saturated?");
178     print (isSaturated(I));
179     return (testAllBoundariesWithGivenFan(I, F));
180 }
```

```
181
182 /**
183  * @returns the product of the ring variables of the current basering as polynomial
184  */
185 proc varprod()
186 {
187     poly prod = 1;
188     for (int i = 1; i <= nvars(basering); i++)
189     {
190         prod = prod * var(i);
191     }
192     return (prod);
193 }
194
195 /**
196  * @param I ideal
197  * @returns 1 iff I is saturated with respect to the product of the ring variables
198  */
199 proc isSaturated(ideal I)
200 {
201     ideal IdealQuot = quotient(I, varprod());
202     return (reduce (std(IdealQuot), std(I)) == 0);
203 }
```

D. Examples

D.1 Example

The following code deals with the tropicalization of $\langle x + y \rangle \in \mathbb{Q}[x, y]$. It is mentioned during the thesis in the examples 4.7, 5.12, 6.9, 7.16, 7.29, 7.33 and 7.36.

Listing 4: xplusy.c

```
LIB "gfanlib.so";

ring r = 0,(x,y), dp; // Q[x,y]
ideal I = x+y;

print ("// Ideal:");
print (I);

fan F = tropicalVariety(I);

// the tropical Variety consists of one cone of dimension 1
print ("// Tropicalization:");
print (F);

// store first cone of dimension 1 in F
cone c = getCone(F, 1, 1);

print ("");

print ("// F = {c} where c is defined via...");
// print it out - representation using (in)equalities
print ("// Inequalities:");
print ( inequalities(c));
print ("// Equalities:");
print ( equations(c));

// print it out - representation cone (HL \cup L \cup -L)
print ("// ... respectively c = cone (Half-Lines cup Lines cup -Lines) where...");
print ("// Half-Lines:");
print ( rays(c));
print ("// Lines:");
print ( generatorsOfLinealitySpace(c));


print ("");
// since neither (-1, 0) nor (0, -1) is contained in the support of F
// it is not interesting to see the boundary in this case - it is the empty set
intvec minus_e1 = -1, 0;
intvec minus_e2 = 0, -1;
print ("// -e_1 in c?");
containsInSupport(c, minus_e1);
print ("// -e_2 in c?");
containsInSupport(c, minus_e2);
```

Output:

```
// Ideal:
x+y
// Tropicalization:
_application PolyhedralFan
_version 2.2
_type PolyhedralFan

AMBIENT_DIM
```

```
2

DIM
1

LINEALITY_DIM
1

RAYS

N_RAYS
0

LINEALITY_SPACE
-1 -1   # 0

ORTH_LINEALITY_SPACE
1 -1    # 0

F_VECTOR
1

SIMPLICIAL
1

PURE
1

CONES
{}      # Dimension 1

MAXIMAL_CONES
{}      # Dimension 1

MULTIPLICITIES
1       # Dimension 1


// F = {c} where c is defined via...
// Inequalities:

// Equalities:
1,-1
// ... respectively c = cone (Half-Lines cup Lines cup -Lines) where...
// Half-Lines:

// Lines:
-1,-1

// -e_1 in c?
0
// -e_2 in c?
0
```

D.2 Example

The following code computes the tropicalization of the homogenization of the circle

$$\langle\, x^2 + y^2 - w^2 \,\rangle \subseteq \mathbb{Q}[w, x, y]$$

respectively its embedding in $\mathbb{P}^4_{\mathbb{C}}$ given by

$$\langle\, x^2 + y^2 - w^2, z \,\rangle \subseteq \mathbb{Q}[w, x, y, z].$$

It occurs in the following examples: 7.8, 7.38, 8.16 and 9.20.

Listing 5: circle.c

```
1  LIB "gfanlib.so";
2  LIB "tropicalboundaries.lib";
3
4  ring R = 0,(w,x,y), dp; // Q[x,y]
5  ideal Circle = x2 + y2 - w2; // homogenization of x2 + y2 - 1
6
7  print ("// Ideal:");
8  print (Circle);
9
10 fan F = tropicalVariety(Circle);
11
12 // the tropical Variety consists only of four cones
13 print ("// Tropicalization:");
14 print (F);
15
16 print ("// F = {C0, C1, C2, C3}");
17
18 // store first cone of dimension 1 in F
19 cone c;
20
21 int i = 0;
22
23 c = getCone(F, 1, 1);
24 // print it out
25 print ("// C0:");
26 print ("//    Inequalities:");
27 print ( inequalities(c));
28 print ("//    Equalities:");
29 print ( equations(c));
30
31 for (i = 1; i <= 3; i++)
32 {
33     c = getCone(F, 2, i);
34     print ("// C" + string (i) + ":");
35     print ("//    Inequalities:");
36     print ( inequalities(c));
37     print ("//    Equalities:");
38     print ( equations(c));
39 }
40
41 // Let us embed the circle in the w-x-y-plane in a 4 dimensional space
42 ring Rz = 0,(w,x,y,z), dp; // Q[x,y]
43 ideal CircleEmbedded = x2 + y2 - w2, z;
44
45 print ("// Ideal:");
46 print (CircleEmbedded);
```

```
47
48 print ("// The ideal is not saturated.");
49 print ("// Its saturation is the whole ring:");
50 print (quotient(CircleEmbedded, w*x*y*z));
51
52 print("// Therefore, its tropicalization is the empty set");
53 print (tropicalVariety(CircleEmbedded));
54
55 print ("// The boundary via projection can't recover the original tropicaliation:");
56 print (boundaryViaProjection(CircleEmbedded, 4));
57
58 print ("// But the boundary via elimination can!");
59 print (boundaryViaElimination(CircleEmbedded, 4));
```

Output:

```
1 // Ideal:
2 -w2+x2+y2
3 // Tropicalization:
4 _application PolyhedralFan
5 _version 2.2
6 _type PolyhedralFan
7
8 AMBIENT_DIM
9 3
10
11 DIM
12 2
13
14 LINEALITY_DIM
15 1
16
17 RAYS
18 -2 1 1  # 0
19 1 -2 1  # 1
20 1 1 -2  # 2
21
22 N_RAYS
23 3
24
25 LINEALITY_SPACE
26 -1 -1 -1        # 0
27
28 ORTH_LINEALITY_SPACE
29 1 -1 0  # 0
30 1 0 -1  # 1
31
32 F_VECTOR
33 1 3
34
35 SIMPLICIAL
36 1
37
38 PURE
39 1
40
41 CONES
42 {}      # Dimension 1
43 {0}     # Dimension 2
44 {1}
45 {2}
46
47 MAXIMAL_CONES
48 {0}     # Dimension 2
```

```
{1}
{2}

MULTIPLICITIES
1       # Dimension 2
1
1

// F = {C0, C1, C2, C3}
// C0:
// Inequalities:

// Equalities:
1,-1, 0,
0, 1,-1
// C1:
// Inequalities:
-1,0,1
// Equalities:
0,1,-1
// C2:
// Inequalities:
0,-1,1
// Equalities:
1,0,-1
// C3:
// Inequalities:
0,1,-1
// Equalities:
1,-1,0
// Ideal:
-w2+x2+y2,
z
// The ideal is not saturated.
// Its saturation is the whole ring:
1
// Therefore, its tropicalization is the empty set
_application PolyhedralFan
_version 2.2
_type PolyhedralFan

AMBIENT_DIM
4

DIM
-1

LINEALITY_DIM
4

RAYS

N_RAYS
0

LINEALITY_SPACE
1 0 0 0 # 0
0 1 0 0 # 1
0 0 1 0 # 2
0 0 0 1 # 3

ORTH_LINEALITY_SPACE

F_VECTOR
```

```
113
114
115 SIMPLICIAL
116 1
117
118 PURE
119 1
120
121 CONES
122
123 MAXIMAL_CONES
124
125 MULTIPLICITIES
126
127 // The boundary via projection can't recover the original tropicaliation:
128 _application PolyhedralFan
129 _version 2.2
130 _type PolyhedralFan
131
132 AMBIENT_DIM
133 3
134
135 DIM
136 -1
137
138 LINEALITY_DIM
139 3
140
141 RAYS
142
143 N_RAYS
144 0
145
146 LINEALITY_SPACE
147 1 0 0   # 0
148 0 1 0   # 1
149 0 0 1   # 2
150
151 ORTH_LINEALITY_SPACE
152
153 F_VECTOR
154
155
156 SIMPLICIAL
157 1
158
159 PURE
160 1
161
162 CONES
163
164 MAXIMAL_CONES
165
166 MULTIPLICITIES
167
168 // But the boundary via elimination can!
169 _application PolyhedralFan
170 _version 2.2
171 _type PolyhedralFan
172
173 AMBIENT_DIM
174 3
175
176 DIM
```

```
2

LINEALITY_DIM
1

RAYS
-2 1 1  # 0
1 -2 1  # 1
1 1 -2  # 2

N_RAYS
3

LINEALITY_SPACE
-1 -1 -1        # 0

ORTH_LINEALITY_SPACE
1 -1 0  # 0
1 0 -1  # 1

F_VECTOR
1 3

SIMPLICIAL
1

PURE
1

CONES
{}      # Dimension 1
{0}     # Dimension 2
{1}
{2}

MAXIMAL_CONES
{0}     # Dimension 2
{1}
{2}

MULTIPLICITIES
1       # Dimension 2
1
1
```

D.3 Example

The Grassmanian $G(2,4)$ respectively its defining ideal was used as the main example in the thesis and it was studied in the examples 3.7, 3.11, 3.15, 3.18, 3.31, 7.39, 8.5, 8.10, 8.17, 9.6, 9.8 and 9.19.

Listing 6: g24.c

```
LIB "grassmanian.lib";
execute(read("test_tropicalboundaries.c"));

int r = 2;
int m = 4;
def Kp = plueckerCoordRing(r, m);
setring Kp;

print ("// The Pluecker ideal Irm for r = 2, m = 4 is a principal ideal.");
print ("// Its generator can be computed as the following Pluecker relation:");
list I = 2;
list J = 1, 3, 4;
print ("// P_{2},{1,3,4} = ");
print (plueckerRelation(I,J));

print ("// or as the following van der Waerden syzygy:");
list alpha = 1;
list beta = 2,3,4;
list gamma = list(); // empty
print ("// [[ 1 .2 .3 .4]] =");
print (vanDerWaerdenSyzygy(alpha, beta, gamma));

// suppress output of "grassmanianGB"
printlevel = -1;
Kp = grassmanianGB(2,4);
setring Kp;
printlevel = 1;

// use gfanlib
fan F2 = tropicalVariety(Irm);

print ("");

print ("// Irm:");
print (Irm);
print ("// Trop(Irm):");
print (F2);

// create fan returned by gfan by hand
fan F = emptyFan(6);
intvec r0 = -2,1,1,1,1,-2;
intvec r1 = 1,-2,1,1,-2,1;
intvec r2 = 1,1,-2,-2,1,1;
intmat L[4][6] = 1,0,0,0,0,-1,0,1,0,0,-1,0,0,0,1,0,1,1,0,0,0,1,1,1;
intmat C1[1][6] = -2,1,1,1,1,-2;
intmat C2[1][6] = 1,-2,1,1,-2,1;
intmat C3[1][6] = 1,1,-2,-2,1,1;
cone c1 = coneViaPoints(C1,L);
cone c2 = coneViaPoints(C2,L);
cone c3 = coneViaPoints(C3,L);
insertCone(F, c1);
insertCone(F, c2);
insertCone(F, c3);

print ("// One can also compute Trop(Irm) in Gfan and import it to get the same fan");
```

```
print (fansEqual(F, F2));

print ("// The Pluecker ideal Irm is always saturated wrt the product of the variables");
print ("// p = ");
print (varprod());
print ("// Irm = (Irm : p)");
print (isSaturated(Irm));

print ("// Therefore, the two ways to compute the boundary always yield the same result")
    ;
testAllBoundariesWithGivenFan(Irm, F);
```

Output:

```
// The Pluecker ideal Irm for r = 2, m = 4 is a principal ideal.
// Its generator can be computed as the following Pluecker relation:
// P_{2},{1,3,4} =
// j = J[1] = 1
//     p_1_2*p_3_4
// j = J[2] = 3
//     p_1_4*p_2_3
// j = J[3] = 4
//     -p_1_3*p_2_4
p_1_4*p_2_3-p_1_3*p_2_4+p_1_2*p_3_4
// or as the following van der Waerden syzgy:
// [[ 1 .2 .3 .4]] =
// beta_tau     = 2,3
// beta_tau_bar = 4
//    p_1_4*p_2_3
// beta_tau     = 2,4
// beta_tau_bar = 3
//    -p_1_3*p_2_4
// beta_tau     = 3,4
// beta_tau_bar = 2
//    p_1_2*p_3_4
p_1_4*p_2_3-p_1_3*p_2_4+p_1_2*p_3_4

// Irm:
p_1_4*p_2_3-p_1_3*p_2_4+p_1_2*p_3_4
// Trop(Irm):
_application PolyhedralFan
_version 2.2
_type PolyhedralFan

AMBIENT_DIM
6

DIM
5

LINEALITY_DIM
4

RAYS
-2 1 1 1 1 -2   # 0
1 -2 1 1 -2 1   # 1
1 1 -2 -2 1 1   # 2

N_RAYS
3

LINEALITY_SPACE
1 0 0 0 0 -1    # 0
0 1 0 0 -1 0    # 1
```

```
0 0 -1 0 -1 -1  # 2
0 0 0 -1 -1 -1  # 3

ORTH_LINEALITY_SPACE
0 -1 1 1 -1 0   # 0
-1 0 1 1 0 -1   # 1

F_VECTOR
1 3

SIMPLICIAL
1

PURE
1

CONES
{}      # Dimension 4
{0}     # Dimension 5
{1}
{2}

MAXIMAL_CONES
{0}     # Dimension 5
{1}
{2}

MULTIPLICITIES
1       # Dimension 5
1
1

// One can also compute Trop(Irm) in Gfan and import it to get the same fan
1
// The Pluecker ideal Irm is always saturated wrt the product of the variables
// p =
p_1_2*p_1_3*p_1_4*p_2_3*p_2_4*p_3_4
// Irm = (Irm : p)
1
// Therefore, the two ways to compute the boundary always yield the same result
// Checking 64 combinations for equality:
// Checking
// EQUAL
// 1% DONE
// Checking 1
// EQUAL
// 3% DONE
// Checking 2
// EQUAL
// 4% DONE
// Checking 3
// EQUAL
// 6% DONE
// Checking 4
// EQUAL
// 7% DONE
// Checking 5
// EQUAL
// 9% DONE
// Checking 6
// EQUAL
// 10% DONE
// Checking 1,2
// EQUAL
```

```
// 12% DONE
// Checking 1,3
// EQUAL
// 14% DONE
// Checking 1,4
// EQUAL
// 15% DONE
// Checking 1,5
// EQUAL
// 17% DONE
// Checking 1,6
// EQUAL
// 18% DONE
// Checking 2,3
// EQUAL
// 20% DONE
// Checking 2,4
// EQUAL
// 21% DONE
// Checking 2,5
// EQUAL
// 23% DONE
// Checking 2,6
// EQUAL
// 25% DONE
// Checking 3,4
// EQUAL
// 26% DONE
// Checking 3,5
// EQUAL
// 28% DONE
// Checking 3,6
// EQUAL
// 29% DONE
// Checking 4,5
// EQUAL
// 31% DONE
// Checking 4,6
// EQUAL
// 32% DONE
// Checking 5,6
// EQUAL
// 34% DONE
// Checking 1,2,3
// EQUAL
// 35% DONE
// Checking 1,2,4
// EQUAL
// 37% DONE
// Checking 1,2,5
// EQUAL
// 39% DONE
// Checking 1,2,6
// EQUAL
// 40% DONE
// Checking 1,3,4
// EQUAL
// 42% DONE
// Checking 1,3,5
// EQUAL
// 43% DONE
// Checking 1,3,6
// EQUAL
// 45% DONE
```

```
// Checking 1,4,5
// EQUAL
// 46% DONE
// Checking 1,4,6
// EQUAL
// 48% DONE
// Checking 1,5,6
// EQUAL
// 50% DONE
// Checking 2,3,4
// EQUAL
// 51% DONE
// Checking 2,3,5
// EQUAL
// 53% DONE
// Checking 2,3,6
// EQUAL
// 54% DONE
// Checking 2,4,5
// EQUAL
// 56% DONE
// Checking 2,4,6
// EQUAL
// 57% DONE
// Checking 2,5,6
// EQUAL
// 59% DONE
// Checking 3,4,5
// EQUAL
// 60% DONE
// Checking 3,4,6
// EQUAL
// 62% DONE
// Checking 3,5,6
// EQUAL
// 64% DONE
// Checking 4,5,6
// EQUAL
// 65% DONE
// Checking 1,2,3,4
// EQUAL
// 67% DONE
// Checking 1,2,3,5
// EQUAL
// 68% DONE
// Checking 1,2,3,6
// EQUAL
// 70% DONE
// Checking 1,2,4,5
// EQUAL
// 71% DONE
// Checking 1,2,4,6
// EQUAL
// 73% DONE
// Checking 1,2,5,6
// EQUAL
// 75% DONE
// Checking 1,3,4,5
// EQUAL
// 76% DONE
// Checking 1,3,4,6
// EQUAL
// 78% DONE
// Checking 1,3,5,6
```

```
// EQUAL
// 79% DONE
// Checking 1,4,5,6
// EQUAL
// 81% DONE
// Checking 2,3,4,5
// EQUAL
// 82% DONE
// Checking 2,3,4,6
// EQUAL
// 84% DONE
// Checking 2,3,5,6
// EQUAL
// 85% DONE
// Checking 2,4,5,6
// EQUAL
// 87% DONE
// Checking 3,4,5,6
// EQUAL
// 89% DONE
// Checking 1,2,3,4,5
// EQUAL
// 90% DONE
// Checking 1,2,3,4,6
// EQUAL
// 92% DONE
// Checking 1,2,3,5,6
// EQUAL
// 93% DONE
// Checking 1,2,4,5,6
// EQUAL
// 95% DONE
// Checking 1,3,4,5,6
// EQUAL
// 96% DONE
// Checking 2,3,4,5,6
// EQUAL
// 98% DONE
// TRIVIALLY EQUAL
// EQUAL FOR ALL COMBINATIONS!
1
```

D.4 Example

The tropicalization of the Pluecker ideal $I_{3,6}$ is a 10-dimensional fan with about 3000 cones in $\mathbb{R}^{20}$. Because of its size, it is only mentioned briefly in example 3.32 and we do not give the full output here.

Listing 7: g36.c

```
LIB "grassmanian.lib";
execute(read("test_tropicalboundaries.c"));

int r = 3;
int m = 6;
print ("// For r = 3, m = 6, the set of all Pluecker relations is not a Groebner basis");

def Kp = grassmanianGenerators(r, m);
setring Kp;
print ("// Irm: (set of generators, not a Groebner basis)");
print (Irm);

Kp = grassmanianGB(r, m);
setring Kp;
print ("// Irm: (dp-Groebner basis)");
print (Irm);

// takes about 10 minutes
// print (tropicalVariety(Irm));

print ("// It takes quite long to compute Trop(Irm)");
print ("// so we can compute in Gfan and import the result");

string g36gfan = read("g36out.gfan");
fan F = fanFromString(g36gfan);

// ~3000 lines
// print ("// Trop(Irm):");
// print (F)

print ("// Trop(Irm) is a pure fan of dimension 10 in a 20-dimensional ambient space");
print ("// Number of maximal cones:");
print (nmaxcones(F));

// takes about 20*10 minutes
print ("// As for r = 2, m = 4, the two ways to compute the boundaries are equivalent");
print ("// To verify this, one can change Bound^p_i (Irm) = Bound^e_i(Irm) for all i");
list single_vars = subsets(1, 20);
testBoundariesWithGivenFan(Irm, F, single_vars);
```

Input for Gfan to compute Trop $(I_{3,6})$**:**

Listing 8: g36in.gfan

```
Q[p_1_2_3,p_1_2_4,p_1_2_5,p_1_2_6,p_1_3_4,p_1_3_5,p_1_3_6,p_1_4_5,p_1_4_6,p_1_5_6,p_2_3_4
    ,p_2_3_5,p_2_3_6,p_2_4_5,p_2_4_6,p_2_5_6,p_3_4_5,p_3_4_6,p_3_5_6,p_4_5_6]
{
    p_2_5_6*p_3_4_6-p_2_4_6*p_3_5_6+p_2_3_6*p_4_5_6,
    p_1_5_6*p_3_4_6-p_1_4_6*p_3_5_6+p_1_3_6*p_4_5_6,
    p_2_5_6*p_3_4_5-p_2_4_5*p_3_5_6+p_2_3_5*p_4_5_6,
    p_2_4_6*p_3_4_5-p_2_4_5*p_3_4_6+p_2_3_4*p_4_5_6,
    p_2_3_6*p_3_4_5-p_2_3_5*p_3_4_6+p_2_3_4*p_3_5_6,
    p_1_5_6*p_3_4_5-p_1_4_5*p_3_5_6+p_1_3_5*p_4_5_6,
    p_1_4_6*p_3_4_5-p_1_4_5*p_3_4_6+p_1_3_4*p_4_5_6,
    p_1_3_6*p_3_4_5-p_1_3_5*p_3_4_6+p_1_3_4*p_3_5_6,
    p_1_2_6*p_3_4_5-p_1_2_5*p_3_4_6+p_1_2_4*p_3_5_6-p_1_2_3*p_4_5_6,
    p_1_5_6*p_2_4_6-p_1_4_6*p_2_5_6+p_1_2_6*p_4_5_6,
    p_2_3_6*p_2_4_5-p_2_3_5*p_2_4_6+p_2_3_4*p_2_5_6,
    p_1_5_6*p_2_4_5-p_1_4_5*p_2_5_6+p_1_2_5*p_4_5_6,
    p_1_4_6*p_2_4_5-p_1_4_5*p_2_4_6+p_1_2_4*p_4_5_6,
    p_1_3_6*p_2_4_5-p_1_3_5*p_2_4_6+p_1_3_4*p_2_5_6+p_1_2_3*p_4_5_6,
    p_1_2_6*p_2_4_5-p_1_2_5*p_2_4_6+p_1_2_4*p_2_5_6,
    p_1_5_6*p_2_3_6-p_1_3_6*p_2_5_6+p_1_2_6*p_3_5_6,
    p_1_4_6*p_2_3_6-p_1_3_6*p_2_4_6+p_1_2_6*p_3_4_6,
    p_1_4_5*p_2_3_6-p_1_3_5*p_2_4_6+p_1_3_4*p_2_5_6+p_1_2_5*p_3_4_6-p_1_2_4*p_3_5_6+
        p_1_2_3*p_4_5_6,
    p_1_5_6*p_2_3_5-p_1_3_5*p_2_5_6+p_1_2_5*p_3_5_6,
    p_1_4_6*p_2_3_5-p_1_4_5*p_2_3_6-p_1_3_4*p_2_5_6+p_1_2_4*p_3_5_6,
    p_1_4_5*p_2_3_5-p_1_3_5*p_2_4_5+p_1_2_5*p_3_4_5,
    p_1_3_6*p_2_3_5-p_1_3_5*p_2_3_6+p_1_2_3*p_3_5_6,
    p_1_2_6*p_2_3_5-p_1_2_5*p_2_3_6+p_1_2_3*p_2_5_6,
    p_1_5_6*p_2_3_4-p_1_3_4*p_2_5_6+p_1_2_4*p_3_5_6-p_1_2_3*p_4_5_6,
    p_1_4_6*p_2_3_4-p_1_3_4*p_2_4_6+p_1_2_4*p_3_4_6,
    p_1_4_5*p_2_3_4-p_1_3_4*p_2_4_5+p_1_2_4*p_3_4_5,
    p_1_3_6*p_2_3_4-p_1_3_4*p_2_3_6+p_1_2_3*p_3_4_6,
    p_1_3_5*p_2_3_4-p_1_3_4*p_2_3_5+p_1_2_3*p_3_4_5,
    p_1_2_6*p_2_3_4-p_1_2_4*p_2_3_6+p_1_2_3*p_2_4_6,
    p_1_2_5*p_2_3_4-p_1_2_4*p_2_3_5+p_1_2_3*p_2_4_5,
    p_1_3_6*p_1_4_5-p_1_3_5*p_1_4_6+p_1_3_4*p_1_5_6,
    p_1_2_6*p_1_4_5-p_1_2_5*p_1_4_6+p_1_2_4*p_1_5_6,
    p_1_2_6*p_1_3_5-p_1_2_5*p_1_3_6+p_1_2_3*p_1_5_6,
    p_1_2_6*p_1_3_4-p_1_2_4*p_1_3_6+p_1_2_3*p_1_4_6,
    p_1_2_5*p_1_3_4-p_1_2_4*p_1_3_5+p_1_2_3*p_1_4_5
}
```

Bibliography

[BO98] Egon Balas and Maarten Oosten. “On the dimension of projected polyhedra”. In: *Discrete Applied Mathematics* 87.1–3 (1998), pp. 1 –9.

[Bö] Janko Böhm. “Computer Algebra”. Lecture Notes (TU Kaiserslautern, winter term 2012/13), `http://www.mathematik.uni-kl.de/~boehm/lehre/1213_CA/ca.pdf`.

[Gfan] Anders N. Jensen. *Gfan, a software system for Gröbner fans and tropical varieties*. `http://home.imf.au.dk/jensen/software/gfan/gfan.html`.

[gfanlib] Yue Ren, Anders Nedergaard Jensen, and Frank Seelisch. `gfanlib.so`. *A* SINGULAR *4-0-2 library to access polymake on singular interpreter level.* `http://www.singular.uni-kl.de`. 2015.

[Jen07] A.N. Jensen. *Algorithmic Aspects of Gröbner Fans and Tropical Varieties: Ph.D. Dissertation*. Department of Mathematical Sciences, University of Aarhus, 2007.

[JMM08] Anders Nedergaard Jensen, Hannah Markwig, and Thomas Markwig. “An Algorithm for Lifting Points in a Tropical Variety”. In: *Collect. Math.* (2008).

[Kru] Sven O. Krumke. “Polyhedral Theory and Integer Programming - Chapter 3: Polyhedra and Integer Programs”. Lecture Notes (TU Kaiserslautern, winter term 2013/14), `http://www.mathematik.uni-kl.de/fileadmin/AGs/opt/Lehre/WS1314/IntegerProgramming_WS1314/ip-chapter3_3_.pdf`.

[MS05] Ezra Miller and Bernd Sturmfels. *Combinatorial commutative algebra*. New York, NY: Springer, 2005.

[MS15] Diane Maclagan and Bernd Sturmfels. *Introduction to Tropical Geometry*. Vol. 161. Graduate Studies in Mathematics. American Mathematical Society, Providence, RI, 2015, pp. vii+359.

[Mus13] Sebastian Muskalla. “Solving Integer Programs using the Algorithm of Hosten and Sturmfels”. Bachelor thesis. TU Kaiserslautern, 2013.

[Sing] Wolfram Decker et al. SINGULAR *4-0-2 — A computer algebra system for polynomial computations*. `http://www.singular.uni-kl.de`. 2015.

[Stu93] Bernd Sturmfels. *Algorithms in invariant theory*. Springer, 1993.

Declaration of academic honesty

I hereby declare that this master thesis is my own work and has not been submitted in any form for any other degree or diploma at any university or other institute.
Information derived from the work of others has been acknowledged in the text and a list of references is given in the bibliography.

Signature Place, Date