FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Nathan Chappell

Minkowski-Weyl Theorem

Department of Applied Mathematics

Supervisor of the bachelor thesis:	Hans Raj Tiwary
Study programme:	Computer Science
Study branch:	IOIA

Prague 2019

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In .... date .... signature of the author

Dedication.

Title: Minkowski-Weyl Theorem

Author: Nathan Chappell

Department: Department of Applied Mathematics

Supervisor: Hans Raj Tiwary, Department of Applied Mathematics

Abstract: The Minkowski-Weyl Theorem is proven for polyhedra by first showing the proof for cones, then the reductions from polyhedra to cones. The proof follows Ziegler [**?**], and uses Fourier-Motzkin elimination. A C++ implementation is given for the enumeration algorithm suggested by the proof.

Keywords: Minkowski-Weyl Theorem polyhedra Fourier-Motzkin C++

Contents

Introduction

Polyhedra are fundamental mathematical objects. Two ways of describing polyhedra are:

1. A finite intersection of half-spaces
2. The *Minkowski-Sum* of the *convex-hull* of a finite set of rays and a finite set of points

The Minkowski-Weyl Theoremis a fundamental result in the theory of polyhedra. It states that both means of representation are equivalent. The proof given here is algorithmic in nature, using a technique known as *Fourier-Motzkin elimination*. The algorithm implied by the proof is then implemented in C++.

1. Minkowski-Weyl Theorem

1.1 Polyhedra

Definition 1 (Non-negative Linear Combination)**.** Let $U \in \mathbb{R}^{d\times p}$, $\mathbf{t} \in \mathbb{R}^p, \mathbf{t} \geq \mathbf{0}$, then $\sum_{1\leq j\leq p} t_j U^j = U\mathbf{t}$ is called a *non-negative linear combination of* U*.*

Definition 2 (V-Cone)**.** Let $U \in \mathbb{R}^{d\times p}$. The set of all non-negative linear combinations of U is denoted cone(U). Such a set is called a *V-Cone.*

Definition 3 (Convex Combination)**.** Let $V \in \mathbb{R}^{d\times n}$, $\boldsymbol{\lambda} \in \mathbb{R}^n, \boldsymbol{\lambda} \geq \mathbf{0}, \sum_{1\leq j\leq n} \lambda_j = 1$, then $\sum_{1\leq j\leq n} \lambda_j V^j$ is called a *convex combination of V. The set of all convex combinations of* V *is denoted* conv(V)*.*

Definition 4 (V-Polyhedron)**.** Let $V \in \mathbb{R}^{d\times n}$, $U \in \mathbb{R}^{d\times p}$. Then the set

$$\{\mathbf{x} + \mathbf{y} \mid \mathbf{x} \in \mathrm{cone}(U),\, \mathbf{y} \in \mathrm{conv}(V)\}$$

is called a *V-Polyhedron.*

Definition 5 (H-Polyhedron)**.** Let $A \in \mathbb{R}^{m\times d}$, $\mathbf{b} \in \mathbb{R}^m$. Then the set

$$\left\{\mathbf{x} \in \mathbb{R}^d \;\middle|\; A\mathbf{x} \leq \mathbf{b}\right\}$$

is called an *H-Polyhedron.*

Definition 6 (H-Cone)**.** Let $A \in \mathbb{R}^{m\times d}$. Then the set

$$\left\{\mathbf{x} \in \mathbb{R}^d \;\middle|\; A\mathbf{x} \leq \mathbf{0}\right\}$$

is called an *H-Cone.*

A simple but useful property of cones is that they are closed under addition and positive scaling.

Proposition 1. *Let* C *be either an H-Cone or a V-Cone, for each* i $\mathbf{x}^i \in C$*, and* $c_i \geq 0$*. Then:*

$$\sum_i c_i \mathbf{x}^i \in C$$

Proof. First we prove Proposition 1 for H-Cones, then for V-Cones. If, for each i, $A\mathbf{x}^i \leq \mathbf{0}$, then $A(c_i\mathbf{x}^i) = t_i A\mathbf{x}^i \leq \mathbf{0}$, and

$$A\left(\sum_i c_i\mathbf{x}^i\right) = \sum_i A(c_i\mathbf{x}^i) = \sum_i c_i A\mathbf{x}^i \leq \sum_i \mathbf{0} \leq \mathbf{0}$$

So, $\sum_i c_i\mathbf{x}^i \in C$ when C is an H-Cone. Next, suppose that $C = \mathrm{cone}(U)$, and for each i, $\exists \mathbf{t}_i \geq \mathbf{0} : \mathbf{x}^i = U\mathbf{t}_i$. Then $c_i\mathbf{t}_i \geq \mathbf{0}$, and $\sum_i c_i\mathbf{t}_i \geq \mathbf{0}$. Therefore

$$\sum_i c_i\mathbf{x}^i = \sum_i c_i U\mathbf{t}_i = \sum_i U(c_i\mathbf{t}_i) = U\left(\sum_i c_i\mathbf{t}_i\right)$$

So, $\sum_i c_i\mathbf{x}^i \in C$ when C is a V-Cone. □

This proposition will be used in the following way: if we wish to show that $\sum_i c_i\mathbf{x}^i$ in a member of some cone C, it suffices to show that, for each i, $c_i \geq 0$ and $\mathbf{x}^i \in C$.

1.2 Minkowski-Weyl Theorem

The following theorem is the basic result to be proved in this thesis, which states that V-Polyhedra and H-Polyhedra are two different representations of the same objects.

Theorem 1 (Minkowski-Weyl Theorem). *Every V-Polyhedron is an H-Polyhedron, and every H-Polyhedron is a V-Polyhedron.*

The proof proceeds by first showing that V-Cones are representable as H-Cones, and H-Cones are representable as V-Cones. Then it is shown that the case of polyhedra can be reduced to cones.

Theorem 2 (Minkowski-Weyl Theorem for Cones). *Every V-Cone is an H-Cone, and every H-Cone is a V-Cone.*

2. Proof of the Minkowski-Weyl Theorem

2.1 Every V-Cone is an H-Cone

Definition 7 (Coordinate Projection)**.** Let I be the identity matrix. Then the matrix I' formed by deleting some rows from I is called a **coordinate-projection**.

The proof rests on the following two propostions:

(V1) Every V-Cone is a coordinate-projection of an H-Cone.

(V2) Every coordinate-projection of an H-Cone is an H-Cone.

Proof. Given (V1) and (V2), the proof follows simply. Given a V-Cone, we use (V1), to get a description involving coordinate-projection of an H-Cone. Then we can apply (V2) in order to get an H-Cone. □

Proof of (V1). We prove that every V-Cone is a coordinate-projection of an H-Cone, by giving an explicit formula. Let $U \in \mathbb{R}^{d\times p}$, and observe that

$$\operatorname{cone}(U) = \{U\mathbf{t} \mid \mathbf{t} \in \mathbb{R}^p,\, \mathbf{t} \geq \mathbf{0}\} = \left\{\mathbf{x} \in \mathbb{R}^d \mid (\exists \mathbf{t} \in \mathbb{R}^p)\, \mathbf{x} = U\mathbf{t},\, \mathbf{t} \geq \mathbf{0}\right\}$$

We will collect $\mathbf{t}$ and $\mathbf{x}$ on the left side of the inequality, treating $\mathbf{t}$ as a variable and expressing its contraints as linear inequalities, then project away the coordinates corresponding to $\mathbf{t}$. The following expression takes one step:

$$\mathbf{t} \geq \mathbf{0} \Leftrightarrow -I\mathbf{t} \leq \mathbf{0} \tag{2.1}$$

And using the equality: $a = 0 \Leftrightarrow a \leq 0 \wedge -a \leq 0$, and block matrix notation, we take the second step.

$$\mathbf{x} = U\mathbf{t} \Leftrightarrow \mathbf{x} - U\mathbf{t} = \mathbf{0} \Leftrightarrow \begin{pmatrix} I & -U \\ -I & U \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{t} \end{pmatrix} \leq \mathbf{0} \tag{2.2}$$

Comparing (2.1) and (2.2), we define a new matrix $A' \in \mathbb{R}^{(p+2d)\times(d+p)}$:

$$A' = \begin{pmatrix} \mathbf{0} & -I \\ I & -U \\ -I & U \end{pmatrix}$$

then we can rewrite $\operatorname{cone}(U)$:

$$\operatorname{cone}(U) = \left\{\mathbf{x} \in \mathbb{R}^d \;\middle|\; A' \begin{pmatrix} \mathbf{x} \\ \mathbf{t} \end{pmatrix} \leq \mathbf{0}\right\}$$

Let $\Pi \in \{0,1\}^{d\times(d+p)}$ be the identity matrix in $\mathbb{R}^{(d+p)\times(d+p)}$, but with the last p-rows deleted. Then Π is a coordinate projection, and the above expression can be written:

$$\operatorname{cone}(U) = \Pi\left(\left\{\mathbf{y} \in \mathbb{R}^{d+p} \mid A'\mathbf{y} \leq \mathbf{0}\right\}\right) \tag{2.3}$$

This is a coordinate projection of an H-Cone, and (V1) is shown. □

To prove (V2), we use two separate propositions.

Proposition 2. *Let $B \in \mathbb{R}^{m' \times (d+p)}$, B' be B with the last p columns deleted, and Π the identity matrix with the last p rows deleted (i.e. $B' = \Pi B$). Furthermore, suppose that the last p columns of B are $\mathbf{0}$. Then*

$$\Pi\left(\left\{\mathbf{y} \in \mathbb{R}^{d+p} \mid B\mathbf{y} \leq \mathbf{0}\right\}\right) = \left\{\mathbf{x} \in \mathbb{R}^{d} \mid B'\mathbf{x} \leq \mathbf{0}\right\}$$

Proof. Recall that $B\mathbf{y} \leq \mathbf{0}$ means that $(\forall i)\, \langle B_i, \mathbf{y}\rangle \leq 0$. By the way we've defined B, any row B_i of B can be written $(B_i', \mathbf{0})$, with $\mathbf{0} \in \mathbb{R}^p$. Rewriting $\mathbf{y} \in \mathbb{R}^{d+p}$ as $(\mathbf{x}, \mathbf{w})$ with $\mathbf{x} \in \mathbb{R}^d, \mathbf{w} \in \mathbb{R}^p$, so that $\mathbf{x} = \Pi(\mathbf{y})$. Then

$$\langle B, \mathbf{y}\rangle = \langle (B_i', \mathbf{0}), (\mathbf{x}, \mathbf{w})\rangle = \langle B_i', \mathbf{x}\rangle = \langle B_i', \Pi(\mathbf{y})\rangle$$

It follows that

$$\langle B_i, \mathbf{y}\rangle \leq 0 \Leftrightarrow \langle B_i', \Pi(\mathbf{y})\rangle \leq 0$$

Since B_i is an arbitrary row of B, the proposition is shown. $\square$

In order to use the above proposition, we need a matrix with $\mathbf{0}$ columns. The next proposition shows us how to do so, one column at a time.

Proposition 3. *Let $B \in \mathbb{R}^{m_1 \times (d+p)}$, $1 \leq k \leq p$, and $\mathbf{x} = \sum_{i \neq k} x_i \mathbf{e}_i$. Then there exists a matrix $B' \in \mathbb{R}^{m_2 \times (d+p)}$ with the following properties:*

1. *Every row of B' is a postive linear combination of rows of B.*
2. *m_2 is finite.*
3. *The k-th column of B' is $\mathbf{0}$.*
4. *$(\exists t) B(\mathbf{x} + t\mathbf{e}_k) \leq \mathbf{0} \Leftrightarrow B'\mathbf{x} \leq \mathbf{0}$*

Proof. Partition the rows of B as follows:

$$\begin{aligned} P &= i \mid B_i^k > 0 \\ N &= j \mid B_j^k < 0 \\ Z &= l \mid B_l^k = 0 \end{aligned}$$

Then let B' be a matrix with rows of the following forms:

$$\begin{aligned} C_l &= B_l && \mid l \in Z \\ C_{ij} &= B_i^k B_j - B_j^k B_i && \mid i \in P, j \in N \end{aligned}$$

1 and *2* are clear. *3* can be seen from:

$$\langle C_l, \mathbf{e}_k\rangle = 0$$

$$\langle C_{ij}, \mathbf{e}_k\rangle = \left\langle B_i^k B_j - B_j^k B_i, \mathbf{e}_k\right\rangle = B_i^k B_j^k - B_j^k B_i^k = 0 \tag{2.4}$$

The right direction of *4* is shown in the following calculations. Because $B_l^k = 0$:

$$\langle B_l, \mathbf{x} + t\mathbf{e}_k\rangle = \langle B_l, \mathbf{x}\rangle + tB_l^k = \langle B_l, \mathbf{x}\rangle = \langle C_l, \mathbf{x}\rangle$$

So:

$$\langle B_l, \mathbf{x} + t\mathbf{e}_k \rangle \leq 0 \Rightarrow \langle C_l, \mathbf{x} \rangle \leq 0$$

For rows indexed by P, N, we observe (2.13), and have:

$$\left\langle B_i^k B_j - B_j^k B_i, \mathbf{x} + t\mathbf{e}_k \right\rangle = \left\langle B_i^k B_j - B_j^k B_i, \mathbf{x} \right\rangle$$

Now, we use property *1*:

$$\langle B_i, \mathbf{x} + t\mathbf{e}_k \rangle \leq 0,\ \langle B_j, \mathbf{x} + t\mathbf{e}_k \rangle \leq 0 \Rightarrow \left\langle B_i^k B_j - B_j^k B_i, \mathbf{x} + t\mathbf{e}_k \right\rangle \leq 0$$

Therefore

$$\left\langle B_i^k B_j - B_j^k B_i, \mathbf{x} \right\rangle \leq 0$$

Now suppose that $B'\mathbf{x} \leq \mathbf{0}$. The task is to find a t so that $B\mathbf{x} \leq \mathbf{0}$. Looking at (2.13), any choice of t we make will be okay for rows indexed by Z. So the task is to find a t so that the inequality holds for rows indexed by P and N. Observe

$$\begin{aligned}
\forall i \in P, \forall j \in N \quad & \left\langle B_i^k B_j - B_j^k B_i, \mathbf{x} \right\rangle \leq 0 \Leftrightarrow \\
\forall i \in P, \forall j \in N \quad & \left\langle B_i^k B_j, \mathbf{x} \right\rangle \ \leq\ \left\langle B_j^k B_i, \mathbf{x} \right\rangle \ \Leftrightarrow \\
\forall i \in P, \forall j \in N \quad & \left\langle B_j / B_j^k, \mathbf{x} \right\rangle \geq\ \left\langle B_i / B_i^k, \mathbf{x} \right\rangle \Leftrightarrow
\end{aligned}$$

$$\min_{j \in N} \left\langle B_j / B_j^k, \mathbf{x} \right\rangle \ \geq\ \max_{i \in P} \left\langle B_i / B_i^k, \mathbf{x} \right\rangle$$

Note that the third inequality changes directions because $B_j^k < 0$. Now we choose t to lie in this last interval, and show that we can use it to satisfy all of the constraints given by B. So, we have a t such that

$$\min_{j \in N} \left\langle B_j / B_j^k, \mathbf{x} \right\rangle \geq t \geq \max_{i \in P} \left\langle B_i / B_i^k, \mathbf{x} \right\rangle$$

In particular,

$$\begin{aligned}
(\forall j \in N) \quad & \left\langle B_j / B_j^k, \mathbf{x} \right\rangle \quad \geq t \Rightarrow \\
(\forall j \in N) \quad & \langle B_j, \mathbf{x} \rangle - B_j^k t \leq 0
\end{aligned}$$

Again, the inequality changes directions because $B_j^k < 0$. Now consider a row B_j from B:

$$\langle B_j, \mathbf{x} - t\mathbf{e}_k \rangle = B_j\mathbf{x} - B_j^k t \leq 0$$

Similarly,

$$\begin{aligned}
(\forall i \in P) \quad & t \geq B_i / B_i^k \mathbf{x} \quad \Rightarrow \\
(\forall i \in P) \quad & 0 \geq B_i\mathbf{x} - B_i^k t
\end{aligned}$$

Now consider a row B_i from B:

$$\langle B_i, \mathbf{x} - t\mathbf{e}_k \rangle = B_i\mathbf{x} - B_i^k t \leq 0$$

So, we've demonstrated that $\mathbf{x} - t\mathbf{e}_k$ satisfies all the constraints from B, and the left implication is shown. So *4* holds. $\square$

Now to prove:

(V2) Every coordinate-projection of an H-Cone is an H-Cone.

proof of (V2). Here we prove the case that the coordinate projection is onto the first d of $d+p$ coordinates. Let $\left\{\mathbf{y} \in \mathbb{R}^{d+p} : A'\mathbf{y} \leq \mathbf{0}\right\}$ be the H-Cone we need to project, and Π the coordinate-projection we need to apply (the identity matrix with the last p columns deleted). For each $1 \leq k \leq p$ we can use proposition 3 in an incremental manner, starting with A'.

let $B_0 := A'$
for $1 \leq k \leq p$
  let $B_k :=$ result of proposition 2 applied to B_{k-1}, $\mathbf{e}_{d+k}$
endfor
return B_p

Consider the resulting B. Property *2* holds throughout, so B is finite. After each iteration, property *3* holds for k, so the k-th column is $\mathbf{0}$. Since each iteration only results from non-negative combinations of the result of the previous iteration (property *1*), once a column is $\mathbf{0}$ it remains so. Therefore, at the end of the process, the last p columns of B are all $\mathbf{0}$. Then, by proposition 2, we can apply Π to B by simply deleting the last p columns of B. Denote this resulting matrix A. We still need to check:

$$A'\mathbf{y} \leq \mathbf{0} \Leftrightarrow A(\Pi(\mathbf{y})) \leq \mathbf{0} \tag{2.5}$$

$$(\exists t_1)\dots(\exists t_p)A'(\mathbf{x} + t_1\mathbf{e}_{d+1} + \cdots + t_p\mathbf{e}_{d+p}) \leq \mathbf{0} \Leftrightarrow A\mathbf{x} \leq \mathbf{0} \tag{2.6}$$

Then, using (2.5) and (2.6), it is easy to see that:

$$\Pi\left\{\mathbf{y} \in \mathbb{R}^{d+p} \mid A'\mathbf{y} \leq \mathbf{0}\right\} = \left\{\mathbf{x} \in \mathbb{R}^{d} \mid A\mathbf{x} \leq \mathbf{0}\right\} \tag{2.7}$$

The key observation of this verification utilizes property *4* of proposition 3:

$$(\exists t)B(\mathbf{x} + t\mathbf{e}_k) \leq \mathbf{0} \Leftrightarrow B'\mathbf{x} \leq \mathbf{0}$$

In what follows, let $\mathbf{x} = \sum_{1 \leq j \leq d} x_j\mathbf{e}_j$. The above property is applied sequentially to the sets B_k as follows:

$$\begin{array}{rlr}
(\exists t_p)(\exists t_{p-1})\dots(\exists t_1) & B_0(\mathbf{x} + t_1\mathbf{e}_p + t_2\mathbf{e}_{p-1} + \cdots + t_p\mathbf{e}_d) \leq \mathbf{0} & \Leftrightarrow \\
(\exists t_p)\dots(\exists t_2) & B_1(\mathbf{x} + t_2\mathbf{e}_{d+2} + \cdots + t_p\mathbf{e}_{d+p}) \leq \mathbf{0} & \Leftrightarrow \\
\vdots & \vdots & \vdots \\
(\exists t_p) & B_{p-1}(\mathbf{x} + t_p\mathbf{e}_{d+p}) \leq \mathbf{0} & \Leftrightarrow \\
 & B_p\mathbf{x} \leq \mathbf{0} & \Leftrightarrow
\end{array}$$

Because $A' = B_0$, and A is B_p with the last p columns deleted, (2.5) and (2.6) hold, therefore (2.7) holds, and the proof of (V2) is complete, and we've shown that a coordinate projection of an H-Cone is again an H-Cone. □

With (V1) and (V2) proven, we are now certain that any V-Cone is also an H-Cone.

2.2 Every H-Cone is a V-Cone

Definition 8 (Coordinate Hyperplane)**.** A set of the form

$$\left\{\mathbf{x} \in \mathbb{R}^{d+m} \mid \langle \mathbf{x}, \mathbf{e}_k \rangle = 0\right\} = \left\{\mathbf{x} \in \mathbb{R}^{d+m} \mid x_k = 0\right\}$$

is called a *coordinate-hyperplane.*

We will use coordinate-hyperplanes in the following way. We consider a V-Cone intersected with some coordinate hyperplanes, and write it in the following way:

$$\left\{\mathbf{x} \in \mathbb{R}^d \;\middle|\; (\exists \mathbf{t} \geq 0) \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} = U'\mathbf{t}\right\} \tag{2.8}$$

If we suppose that $U' \subset \mathbb{R}^{d+m}$, and Π is the identity matrix with the last m rows deleted, then this is just a convenient way of writing:

$$\Pi\Big(\operatorname{cone}(U') \cap \{x_{d+1} = 0\} \cap \cdots \cap \{x_{d+m} = 0\}\Big) \tag{2.9}$$

The proof rests on the following three propostions:

H1 Every H-Cone is a coordinate-projection of a V-Cone intersected with some coordinate hyperplanes.

H2 Every V-Cone intersected with a coordinate-hyperplane is a V-Cone

H3 Every coordinate-projection of a V-Cone is an V-Cone.

Proof. Given *H1*, *H2*, and *H3*, the proof follows simply. Given an H-Cone, we use *H1* to get a description involving the coordinate-projection of a V-Cone intersected with some coordinate-hyperplanes. We apply *H2* as many times as necessary to elimintate the intersections, then we can apply *H3* in order to get a V-Cone. □

Proof of H1. Let $A \in \mathbb{R}^{m \times d}$, we now show that the H-Cone

$$\left\{\mathbf{x} \in \mathbb{R}^d \mid A\mathbf{x} \leq \mathbf{0}\right\}$$

can be written as the projection of a V-Cone intersected with some hyperplanes. Define U':

$$U' = \left\{\begin{pmatrix} \mathbf{e}_j \\ A^j \end{pmatrix}, \begin{pmatrix} -\mathbf{e}_j \\ -A^j \end{pmatrix}, \begin{pmatrix} \mathbf{0} \\ \mathbf{e}_i \end{pmatrix}, 1 \leq j \leq d,\ 1 \leq i \leq m\right\}$$

Then we claim:

$$\left\{\mathbf{x} \in \mathbb{R}^d \mid A\mathbf{x} \leq \mathbf{0}\right\} = \left\{\mathbf{x} \in \mathbb{R}^d \;\middle|\; (\exists \mathbf{t} \geq 0) \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} = U'\mathbf{t}\right\} \tag{2.10}$$

First, considering (2.8) and (2.9), observe that this is a coordinate-projection of a V-Cone intersected with some coordinate-hyperplanes. Next, we note that

$$\begin{pmatrix} \mathbf{x} \\ A\mathbf{x} \end{pmatrix} = \sum_{1 \leq j \leq d} x_j \begin{pmatrix} \mathbf{e}_j \\ A^j \end{pmatrix}$$

We can write this as a sum with all positive coefficients if we split up the x_j as follows:

$$x_j^+ = \begin{cases} x_j & x_j \geq 0 \\ 0 & x_j < 0 \end{cases} \qquad x_j^- = \begin{cases} 0 & x_j \geq 0 \\ -x_j & x_j < 0 \end{cases}$$

Then we have

$$\begin{pmatrix} \mathbf{x} \\ A\mathbf{x} \end{pmatrix} = \sum_{1 \leq j \leq d} x_j^+ \begin{pmatrix} \mathbf{e}_j \\ A^j \end{pmatrix} + \sum_{1 \leq j \leq d} x_j^- \begin{pmatrix} -\mathbf{e}_j \\ -A^j \end{pmatrix} \tag{2.11}$$

where $x_j^+, x_j^- \geq 0$. Also observe that

$$A\mathbf{x} \leq \mathbf{0} \Leftrightarrow (\exists \mathbf{w} \geq \mathbf{0}) \mid A\mathbf{x} + \mathbf{w} = \mathbf{0}$$

This can also be written

$$A\mathbf{x} \leq \mathbf{0} \Leftrightarrow (\exists \mathbf{w} \geq \mathbf{0}) \mid \begin{pmatrix} \mathbf{x} \\ A\mathbf{x} \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \mathbf{w} \end{pmatrix} = \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} \tag{2.12}$$

(2.11) and (2.12) together show

$$A\mathbf{x} \leq \mathbf{0} \Rightarrow (\exists \mathbf{t} \geq 0) \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} = U'\mathbf{t}$$

Conversely, suppose

$$(\exists \mathbf{t} \geq 0) \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} = U'\mathbf{t}$$

We would like to show that $A\mathbf{x} \leq \mathbf{0}$. Let x_j^+, x_j^-, w_i take the values of $\mathbf{t}$ that are coefficients of $\begin{pmatrix} \mathbf{e}_j \\ A^j \end{pmatrix}$, $\begin{pmatrix} -\mathbf{e}_j \\ -A^j \end{pmatrix}$, and $\begin{pmatrix} \mathbf{0} \\ \mathbf{e}_i \end{pmatrix}$ respectively, and denote $x_j = x_j^+ - x_j^-$. Then we have

$$\begin{aligned} \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} &= \sum_{1 \leq j \leq d} x_j^+ \begin{pmatrix} \mathbf{e}_j \\ A^j \end{pmatrix} + \sum_{1 \leq j \leq d} x_j^- \begin{pmatrix} -\mathbf{e}_j \\ -A^j \end{pmatrix} + \sum_{1 \leq i \leq n} w_i \begin{pmatrix} \mathbf{0} \\ \mathbf{e}_i \end{pmatrix} \\ &= \sum_{1 \leq j \leq d} x_j \begin{pmatrix} \mathbf{e}_j \\ A^j \end{pmatrix} + \sum_{1 \leq i \leq n} w_i \begin{pmatrix} \mathbf{0} \\ \mathbf{e}_i \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{x} \\ A\mathbf{x} \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \mathbf{w} \end{pmatrix} \end{aligned}$$

where $\mathbf{w} \geq \mathbf{0}$. By (2.12) we have $A\mathbf{x} \leq \mathbf{0}$. So (2.10) holds. □

The proof of *H2* relies upon the following proposition.

Proposition 4. *Let $Y \in \mathbb{R}^{(d+m)\times n_1}$, $1 \leq k \leq m$, and $\mathbf{x}$ satisfy $x_k = 0$. Then there exists a matrix $Y' \in \mathbb{R}^{(d+m)\times n_2}$ with the following properties:*

1. *Every column of Y' is a postive linear combination of rows of B.*
2. *n_2 is finite.*
3. *The k-th row of Y' is $\mathbf{0}$.*

4. $(\exists \mathbf{t} \geq \mathbf{0})\mathbf{x} = Y\mathbf{t} \Leftrightarrow (\exists \mathbf{t}' \geq \mathbf{0})\mathbf{x} = Y'\mathbf{t}'$

Recall that Y^i is the i-th column of Y, and Y^i_k is the element of Y in the i-th column and k-th row.

Proof. We partition the columns of Y:

$$\begin{aligned} P &= i \ \mid Y^i_k > 0 \\ N &= j \ \mid Y^j_k < 0 \\ Z &= l \ \mid Y^l_k = 0 \end{aligned}$$

We then define Y':

$$Y' = \left\{Y^l \mid l \in Z\right\} \cup \left\{Y^i_k Y^j - Y^j_k Y^i \mid i \in P,\, j \in N\right\}$$

1 and *2* are clear. *3* can be seen from:

$$\left\langle Y'^l, \mathbf{e}^k \right\rangle = 0$$

$$\left\langle Y'^{ij}, \mathbf{e}^k \right\rangle = \left\langle Y^i_k Y^j - Y^j_k Y^i, \mathbf{e}^k \right\rangle = Y^i_k Y^j_k - Y^j_k Y^i_k = 0 \tag{2.13}$$

Before moving on to the proof, we first note how to write our vectors.

$$Y\mathbf{t} = \sum_{l\in Z} t_k Y^k + \sum_{i\in P} t_i Y^i + \sum_{j\in N} t_j Y^j$$

$$Y'\mathbf{t} = \sum_{l\in Z} t_k Y^k + \sum_{\substack{i\in P\\ j\in N}} t_{ij}(Y^i_k Y^j - Y^j_k Y^i)$$

Then, by proposition 1, to show that the proposition is true, we need only show that, given any $t_i, t_j \geq 0$ ($t_{ij} \geq 0$), there exists $t_{ij} \geq 0$ ($t_i, t_j \geq 0$) such that

$$\sum_{i\in P} t_i Y^i + \sum_{j\in N} t_j Y^j = \sum_{\substack{i\in P\\ j\in N}} t_{ij}(Y^i_k Y^j - Y^j_k Y^i) \tag{2.14}$$

Proposition 5. *Suppose that*

$$\sum_{i\in P} t_i Y^i_{d+1} + \sum_{j\in N} t_j Y^j_{d+1} = 0 \qquad Y^j_k < 0 < Y^i_k$$

Then the following holds

$$\begin{aligned} (t_i, t_j \geq 0) \Rightarrow (\exists t_{ij} \geq 0) \quad &\text{such that (2.14) holds} \\ (t_{ij} \geq 0) \Rightarrow (\exists t_i, t_j \geq 0)\ &\text{such that (2.14) holds} \end{aligned}$$

Proof. First note that if all $t_i = 0, t_j = 0$, then choosing $t_{ij} = 0$ satisfies (2.14), likewise if all $t_{ij} = 0$, then $t_i = 0, t_j = 0$ satisfies (2.14). So suppose that some $t_i \neq 0, t_j \neq 0, t_{ij} \neq 0$.

The right hand side of (2.14) can be written

$$\sum_{j\in N}\left(\sum_{i\in P} t_{ij} Y^i_k\right) Y^j + \sum_{i\in P}\left(-\sum_{j\in N} t_{ij} Y^j_k\right) Y^i$$

This means, given $t_{ij} \geq 0$, we can choose $t_j = \sum_{i \in P} t_{ij} Y_k^i$, and $t_i = -\sum_{j \in N} t_{ij} Y_k^j$, both of which are greater than 0.

Now suppose we have been given $t_i \geq 0, t_j \geq 0$. First observe:

$$0 = \sum_{i \in P} t_i Y_k^i + \sum_{j \in N} t_j Y_k^j \Rightarrow \sum_{i \in P} t_i Y_k^i = -\sum_{j \in N} t_j Y_k^j$$

Denote the value in this equality as σ, and note that $\sigma > 0$. Then

$$\begin{aligned} \sum_{i \in P} t_i Y^i &= \frac{-\sum_{j \in N} t_j Y_k^j}{\sigma} \sum_{i \in P} t_i Y^i = \sum_{\substack{i \in P \\ j \in N}} -\frac{t_i t_j}{\sigma} Y_k^j Y^i \\ \sum_{j \in N} t_j Y^j &= \frac{\sum_{i \in P} t_i Y_k^i}{\sigma} \sum_{j \in N} t_j Y^j = \sum_{\substack{i \in P \\ j \in N}} \frac{t_i t_j}{\sigma} Y_k^i Y^j \end{aligned}$$

Combining these results, we have

$$\sum_{i \in P} t_i Y^i + \sum_{j \in N} t_j Y^j = \sum_{\substack{i \in P \\ j \in N}} \frac{t_i t_j}{\sigma} (Y_k^i Y^j - Y_k^j Y^i)$$

□

Finally, we can conclude that, given $\mathbf{t} \geq \mathbf{0}$, if $Y\mathbf{t}$ has a 0 in the final coordinate, then we can write it as $Y'\mathbf{t}'$ where $\mathbf{t}' \geq \mathbf{0}$, and any non-negative linear combination of vectors from Y' can be written as a non-negative linear combination of vetors from Y, and will necessarily have the k-th coordinate be 0 by property *3*. So property *4* holds. □

Proof of H2. In proposition 4, the assumption that $x_k = 0$ in property *4* creates the set $\text{cone}(Y) \cap \{\mathbf{x} \mid x_k = 0\}$. This set, by property *4*, is $\text{cone}(Y')$. □

Proof of H3. We shall prove that the coordinate-projection of a V-Cone is again a V-Cone. Let Π be the relevant projection, then we have:

$$\Pi \{U\mathbf{t} \mid \mathbf{t} \geq \mathbf{0}\} = \{\Pi(U\mathbf{t}) \mid \mathbf{t} \geq \mathbf{0}\} = \{\Pi(U)\mathbf{t} \mid \mathbf{t} \geq \mathbf{0}\}$$

The last equality follows from associativity of matrix multiplication. Therefore,

$$\Pi\Big(\text{cone}(U)\Big) = \text{cone}\Big(\Pi(U)\Big)$$

□

2.3 Reducing Polyhedra to Cones

Definition 9 (Hyperplane)**.** Let $\mathbf{y} \in \mathbb{R}^d$, $c \in \mathbb{R}$. Then a set of the form

$$\Big\{\mathbf{x} \in \mathbb{R}^d \mid \langle \mathbf{y}, \mathbf{x} \rangle = c\Big\}$$

is called a *hyperplane*.

2.3.1 H-Polyhedra $\leftrightarrow$ H-Cones

We show that an H-Polyhedron can be represented as the projection of an H-Cone intersected with a hyperplane. We begin by re-writing the expression:

$$A\mathbf{x} \leq \mathbf{b} \Leftrightarrow -\mathbf{b} + A\mathbf{x} \leq \mathbf{0} \Leftrightarrow \Big[-\mathbf{b}|A\Big]\begin{pmatrix}1\\ \mathbf{x}\end{pmatrix} \leq \mathbf{0} \tag{2.15}$$

Proposition 6. *Every H-Polyhedron can be written as an H-Cone intersected with the set $\{\mathbf{x} \mid x_0 = 1\}$, and any H-Cone intersected with the set $\{\mathbf{x} \mid x_0 = 1\}$ is an H-Polyhedron.*

Proof. We extend (2.15):

$$\mathbf{x} \in \Big\{\mathbf{x} \in \mathbb{R}^d \mid A\mathbf{x} \leq \mathbf{b}\Big\} \Leftrightarrow \begin{pmatrix}1\\ \mathbf{x}\end{pmatrix} \in \Big\{\mathbf{y} \in \mathbb{R}^{d+1} \mid \Big[-\mathbf{b}|A\Big]\mathbf{y} \leq \mathbf{0}\Big\}$$

We conclude, given an H-Polyhedron, we can add an extra coordinate and prepend the vector $\mathbf{b}$ to the left of A, and later we can just move this column back to the right side of the inequality and drop the extra coordinate. $\square$

2.3.2 V-Polyhedra $\leftrightarrow$ V-Cone

We show that a V-Polyhedra can be reprented as a projection of a V-Cone intersected with the hyperplane $\Big\{\mathbf{y} \in \mathbb{R}^{d+1} \mid y_0 = 1\Big\}$. Given two sets $V \in \mathbb{R}^{d\times n}$ and $U \in \mathbb{R}^{d\times p}$, the V-Polyhedron is given by:

$$P_V = \{\mathbf{x} + \mathbf{y} \mid \mathbf{x} \in \text{cone}(U),\, \mathbf{y} \in \text{conv}(V)\}$$

It isn't hard to see that

$$\mathbf{x} \in P_V \Leftrightarrow \begin{pmatrix}1\\ \mathbf{x}\end{pmatrix} \in \text{cone}\begin{pmatrix}\mathbf{0} & \mathbf{1}\\ U & V\end{pmatrix}$$

For the value 1 to appear in the first coordinate, a convex combination of the vectors from $(\mathbf{1}, V)$ must be taken. After that, any non-negative combination of $(\mathbf{0}, U)$ added to this vector won't affect the 1 in the first coordinate.

It is more difficult to show that, given a V-Cone, that you can intersect it with the hyperplane $\Big\{\mathbf{y} \in \mathbb{R}^{d+1} \mid y_0 = 1\Big\}$ and get a V-Polytope out of it. So let

$$C_V = \text{cone}(U) \cap \Big\{\mathbf{y} \in \mathbb{R}^{d+1} \mid y_0 = 1\Big\}$$

We partition U into the sets:

$$\begin{aligned} P &= i \mid U_0^i > 0\\ N &= j \mid U_0^j < 0\\ Z &= l \mid U_0^l = 0 \end{aligned}$$

And define two new sets:

$$\begin{aligned} U' &= \Big\{U^l \mid l \in Z\Big\} \cup \Big\{U_0^i U^j - U_0^j U^i \mid i \in P,\, j \in N\Big\}\\ V &= \Big\{U^i / U_0^i \mid i \in P\Big\} \end{aligned}$$

Then I claim that

$$C_V = \{\mathbf{x} + \mathbf{y} \mid \mathbf{x} \in \operatorname{cone}(U'),\, \mathbf{y} \in \operatorname{conv}(V)\} \tag{2.16}$$

Say $\mathbf{x} \in \operatorname{cone}(U')$, $\mathbf{x}$ can be written

$$\begin{aligned}\mathbf{x} &= \sum_{l\in Z} t_l U^l + \sum_{\substack{i\in P\\ j\in N}} t_{ij}(U_0^i U^j - U_0^j U^i)\\ &= \sum_{l\in Z} t_l U^l + \sum_{j\in N}\left(\sum_{i\in P} t_{ij}U_0^i\right)U^j + \sum_{i\in P}\left(\sum_{j\in N} -t_{ij}U_0^j\right)U^i\end{aligned}$$

So $\mathbf{x} \in \operatorname{cone}(U)$. Furthermore,

$$\langle \mathbf{e}_0, \mathbf{x}\rangle = \sum_{l\in Z} t_l U_0^l + \sum_{\substack{i\in P\\ j\in N}} t_{ij}(U_0^i U_0^j - U_0^j U_0^i) = 0$$

So $x_0 = 0$. Similarly, for $\mathbf{y}$,

$$\mathbf{y} = \sum_{i\in P} \lambda_i U^i / U_0^i, \quad \sum_{i\in P} \lambda_i = 1$$

So $\mathbf{y} \in \operatorname{cone}(U)$, and then $\mathbf{x} + \mathbf{y} \in \operatorname{cone}(U)$. Furthermore,

$$\langle \mathbf{e}_0, \mathbf{y}\rangle = \sum_{i\in P} \lambda_i U_0^i / U_0^i = 1$$

So $y_0 = 1$ and $x_0 + y_0 = 1$. Then, by proposition 1, $\mathbf{x} + \mathbf{y} \in C_V$.

Next, suppose that $\mathbf{z} \in C_V$, then $\mathbf{z}$ can be written

$$\mathbf{z} = \sum_{l\in Z} t_l U^l + \sum_{i\in P} t_i U^i + \sum_{j\in N} t_j U^j$$

It will be convenient to use shorter notation for these sums. Define the following:

$$\begin{aligned}\sigma_Z &= \sum_{l\in Z} t_l U^l, & \sigma_l &= \sum_{l\in Z} t_l U_0^l \ = 0\\ \sigma_P &= \sum_{i\in P} t_i U^i, & \sigma_i &= \sum_{i\in P} t_i U_0^i\\ \sigma_N &= \sum_{j\in N} t_j U^j, & \sigma_j &= \sum_{j\in N} t_j U_0^j\end{aligned}$$

Then it holds that

$$\langle \mathbf{e}_0, \mathbf{z}\rangle = \sigma_l + \sigma_i + \sigma_j = \sigma_i + \sigma_j = 1 \quad \Rightarrow \quad -\sigma_j/\sigma_i = 1 - 1/\sigma_i$$

$$\sigma_P = \sigma_P/\sigma_i + (1 - 1/\sigma_i)\sigma_P = \sigma_P/\sigma_i - (\sigma_j/\sigma_i)\sigma_P$$

Using the new notation, we can rewrite $\mathbf{z}$:

$$\mathbf{z} = \sigma_Z + \sigma_P + \sigma_N = \sigma_Z + \frac{\sigma_P}{\sigma_i} - \frac{\sigma_j}{\sigma_i}\sigma_P + \frac{\sigma_i}{\sigma_i}\sigma_N = \sigma_Z + \frac{\sigma_P}{\sigma_i} + \frac{\sigma_i\sigma_N - \sigma_j\sigma_P}{\sigma_i}$$

Using proposition 1, we need only show that

1. $\sigma_Z \in \text{cone}(U')$
2. $(\sigma_i \sigma_N - \sigma_j \sigma_P)/\sigma_i \in \text{cone}(U')$
3. $\sigma_P/\sigma_i \in \text{conv}(V)$

Since each $U^l : l \in Z$ is in C_V, (1) holds. We also have:

$$\sigma_i \sigma_N - \sigma_j \sigma_P = \sum_{i \in P} t_i \sum_{j \in N} t_j U_0^i U^j - \sum_{j \in N} t_j \sum_{i \in P} t_i U_0^j U^i = \sum_{\substack{i \in P \\ j \in N}} t_i t_j (U_0^i U^j - U_0^j U^i)$$

So (2) holds. Finally,

$$\sigma_P/\sigma_i = \sum_{i \in P} t_i U^i/\sigma_i = \sum_{i \in P} (t_i U_0^i/\sigma_i)(U^i/U_0^i)$$

Since $\sum_{i \in P}(t_i U_0^i/\sigma_i) = \sigma_i/\sigma_i = 1$, it follows that $\sigma_P/\sigma_i \in \text{conv}(V)$.

2.4 Picture of the Proof

Here we show a diagram that represent the proof of the Minkowski-Weyl Theorem.

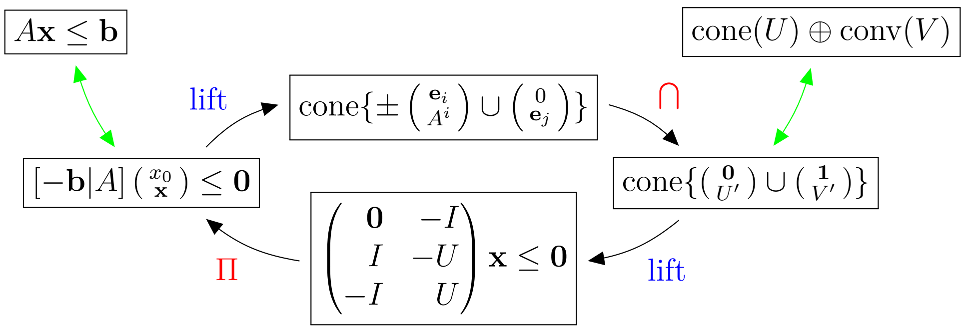

Figure 2.1: Diagram of the proof $P_H \leftrightarrow P_V$

Figure 2.1 shows the flow from an H-Polyhedron to a V-Polyhedron and back. The colored arrows are the transformations back and forth from polyhedra to cones. The black arrows show the transformation between cones. V-Cones are lifted to H-Cones which need to be projected (Π), and H-Cones are lifted to V-Cones which need to be intersected ($\bigcap$) with some coordinate-hyperplanes then projected.

3. C++ Implementation

The above transformation have been implemented in C++. There are four programs:

- `hcone_to_vcone.cpp`
- `hpoly_to_vpoly.cpp`
- `vcone_to_hcone.cpp`
- `vpoly_to_hpoly.cpp`

They all run the indicated transformations. They read the description of the object from standard input, and write the result to standard output. They take no arguments, however if any arguments are passed then they display the following usage information, which also includes the input format:

```
usage:
  (vcone_to_hcone|vpoly_to_hpoly|hcone_to_vcone|hpoly_to_vpoly)
input is read on stdin,transformed object written on stdout
input format is as follows:

hcone, vcone:= dimension   (vector)*
       hpoly:= dimension+1 (vector constraint)*
       vpoly:= dimension   (vector)* 'U' (vector)*

dimension   is a positive integer
vector      is (dimension) doubles separated by whitespace
constraint  is a double (the value b_i in <A_i,x> <= b_i)
hvector     is (dimension) doubles separated by whitespace
'U'         is the literal character 'U'
```

The files pertaining to the implementation will be discussed in the following sections, but here is a table showing the include dependencies followed by a short summary of the files.

file	includes
`common.cpp`	`common.h`
`hcone.cpp`	`hcone.h`
`polyhedra.cpp`	`hcone.h`
	`polyhedra.h`
	`vcone.h`
`vcone.cpp`	`vcone.h`
`hcone.h`	`common.h`
`polyhedra.h`	`common.h`
`vcone.h`	`common.h`

Here is a very brief summary of the files mentioned in the above table, more details are given in sequent sections.

- `common.{cpp,h}`
  Types, IO, and Fourier Motzkin elimination.
- `hcone.{cpp,h}`
  Functions to transform H-Cone → V-Cone.
- `polyhedra.{cpp,h}`
  Transforms between polytopes and polyhedra.
- `vcone.{cpp,h}`
  Functions to transform V-Cone → H-Cone.

3.1 include/common.h, src/common.cpp

```
using Vector = std::valarray<double>;
using Matrix = std::vector<Vector>;
```

The types `Vector` and `Matrix` are used for representing the polyhedra. The `std::valarray` template is used because it has built-in vector-space operations (sum and scaling). `std::vector`, is used, however other sequence containers could be used.

```
struct VPoly {
  Matrix U;
  Matrix V;
};
```

Because a V-Polyhedron needs two matrices two represent it, a the simple struct `VPoly` is defined.

```
extern size_t d;
```

`d` is a global variable denoting "dimension" used by some operations (i.e. reading vectors and projections). `transpose` is used in Fourier-Motzkin elimination when creating the alternate representations. `check_empty_matrix` returns true if there are either no `Vector`s, or the first `Vector` is empty.

```
std::istream& operator>>(std::istream&, Vector&);
std::istream& operator>>(std::istream&, Matrix&);
std::istream& operator>>(std::istream&, VPoly&);
```

```
std::ostream& operator<<(std::ostream& o, const Vector&);
std::ostream& operator<<(std::ostream& o, const Matrix&);
std::ostream& operator<<(std::ostream& o, const VPoly&);
```

The stream input and output `operator>>` and `operator<<` are defined to handle input and output as described in *usage*.

```
class input_error : public std::runtime_error {
```

The input operators may throw an exception of type `input_error` if the input dimension is not positive, or if there is an invalid number of values following the dimension.

```
int usage();
```

Outputs the usage message above.

```
bool check_empty_matrix(const Matrix &M) {
  return (M.empty() || !M.front().size());
}
```

`check_empty_matrix` checks for the corner case of `Matrix` operations. It return `true` if either the `Matrix` has no rows (columns) or the first row (column) is empty.

```
Matrix transpose(const Matrix &M);
```

`transpose` transposes the matrix.

```
Matrix project_matrix(const Matrix &M);
```

`project_matrix` is used to take only only the first `d` entries of each vector in the `Matrix`.

```
Matrix fourier_motzkin(Matrix M, size_t k) {
  Matrix result;
  // Partition into Z,P,N
  const auto z_end = partition(M.begin(),M.end(),
      [k](const Vector &v) { return v[k] == 0; });
  const auto p_end = partition(z_end, M.end(),
      [k](const Vector &v) { return v[k] > 0; });
  // Move Z to result
  move(M.begin(), z_end, back_inserter(result));
  // convolute vectors from P,N
  for (auto p_it = z_end; p_it != p_end; ++p_it) {
    for (auto z_it = p_end; z_it != M.end(); ++z_it) {
      result.push_back(
        (*p_it)[k]*(*z_it) - (*z_it)[k]*(*p_it));
    }
  }
  return result;
}
```

`fourier_motzkin` takes a `Matrix M` and a coordinate `k` and creates the set which either corresponds to a projection of an H-Cone (without actually doing the projection), or the intersection of a V-Cone with a coordinate-hyperplane.

```
  const auto z_end = partition(M.begin(),M.end(),
      [k](const Vector &v) { return v[k] == 0; });
  const auto p_end = partition(z_end, M.end(),
      [k](const Vector &v) { return v[k] > 0; });
```

Partitions `M` into logical sets Z, P, N that satisfy the following:

set	range	property
Z	[`M.begin()`, `z_end`)	`it` $\in Z \Leftrightarrow$ `(*it)[k]` $= 0$
P	[`z_end`, `p_end`)	`it` $\in P \Leftrightarrow$ `(*it)[k]` > 0
N	[`p_end`, `M.end()`)	`it` $\in N \Leftrightarrow$ `(*it)[k]` < 0

```
  move(M.begin(), z_end, back_inserter(result));
```

Moves Z into the result.

```
  for (auto p_it = z_end; p_it != p_end; ++p_it) {
    for (auto z_it = p_end; z_it != M.end(); ++z_it) {
      result.push_back(
        (*p_it)[k]*(*z_it) - (*z_it)[k]*(*p_it));
    }
  }
```

Convolutes the vectors in the way described in Propositions 3 and 4 (concerning projecting an H-Cone and intersecting a V-Cone with a coordinate-hyperplane), and push them into the result `Matrix`. In particular, it creates the sets which correspond to

$$B_i^k B_j - B_j^k B_i \mid i \in P,\ j \in N$$

3.2 include/hcone.h, src/hcone.cpp

```
//hcone.h

#include "common.h"

// V-Cone -> H-Cone operations
namespace HCone {

// represent hcone as projection of vcone
//             d  d m
//    d       d|I -I 0|
// m|A| -> m|A -A I|
//
Matrix lift_hcone(Matrix hcone);

// intersect vcone with {x_k = 0 | d+1 <= k <= d+m}
Matrix intersect_vcone(Matrix vcone);

} //namespace

Matrix hcone_to_vcone (Matrix hcone);
```

`hcone.h` and `hcone.cpp` implement the transformation from H-Cone to V-Cone.

```
Matrix lift_hcone(Matrix hcone);
```

Takes a `Matrix` representing an H-Cone and creates the new matrix

$$\left\{ \begin{pmatrix} \mathbf{e}_j \\ A^j \end{pmatrix}, \begin{pmatrix} -\mathbf{e}_j \\ -A^j \end{pmatrix}, \begin{pmatrix} \mathbf{0} \\ \mathbf{e}_i \end{pmatrix}, 1 \le j \le d,\ 1 \le i \le m \right\}$$

where A represents `hcone`. This operation is justified by *H1*.

```
Matrix intersect_vcone(Matrix vcone) {
  Matrix result = move(vcone);
  for (size_t i = d; i < d+m; ++i) {
    result = fourier_motzkin(move(result), i);
  }
  return project_matrix(result);
}
```

Here the tools from `common.h` are used to implement the algorithm described in proposition 4, where a V-Cone is sequentially intersected with coordinate-hyperplanes. The result of these intersections is the projected to the original space. These operations are justified by *H2* and *H3*.

```
Matrix hcone_to_vcone(Matrix hcone) {
  if (check_empty_matrix(hcone)) {
    throw logic_error{"empty␣hcone"};
  }
  m = hcone.size();
  return HCone::intersect_vcone(HCone::lift_hcone(hcone));
}
```

This function does a sanity check and then return the transformed `hcone`.

3.3 include/vcone.h, src/vcone.cpp

```
//vcone.cpp

#include "common.h"

// V-Cone -> H-Cone operations
namespace VCone {

// represent vcone as projection of hcone
Matrix lift_vcone(const Matrix &vcone);

// project away d+1 to p
Matrix project_hcone(Matrix &&hcone);

} //namespace

Matrix vcone_to_hcone(Matrix vcone);
```

`vcone.h` and `vcone.cpp` implement the transformation from V-Cone to H-Cone.

```
Matrix lift_vcone(const Matrix &vcone);
```

Takes a `Matrix` representing an V-Cone and creates the new matrix

$$\begin{pmatrix} \mathbf{0} & -I \\ I & -U \\ -I & U \end{pmatrix}$$

where U represents `vcone`. This operation is justified by (V1).

```
Matrix project_hcone(Matrix &&hcone) {
  Matrix result = move(hcone);
  for (size_t i = d; i < d+p; ++i) {
    result = fourier_motzkin(move(result), i);
  }
  return project_matrix(result);
}
```

Here the tools from `common.h` are used to implement the algorithm described in proposition 3, where an H-Cone is sequentially projected down to coordinate-axis. This result is then projected to the original space, i.e. the first d coordinates are taken from each `Vector` in the `Matrix`. These operations are justified by (V2) and proposition 2.

```
Matrix vcone_to_hcone(Matrix vcone) {
  if (check_empty_matrix(vcone)) {
    throw logic_error{"empty vcone"};
  }
  p = vcone.size();
  return VCone::project_hcone(VCone::lift_vcone(vcone));
}
```

This function does a sanity check and then return the transformed `hcone`.

3.4 include/polyhedra.h, src/polyhedra.cpp

```
//polyhedra.h

#include "common.h"

// HP -> HC
// A|b -> -b|A
Matrix hpoly_to_hcone(Matrix hpoly);

// HC -> HP
// -b|A -> A|b
Matrix hcone_to_hpoly(Matrix hpoly);

// VP -> VC
// U -> |0 1|
//      |U V|
Matrix vpoly_to_vcone(VPoly vpoly);

// VC -> VP
// U -> {Z \cup P*N, P'}
VPoly vcone_to_vpoly(Matrix vcone);

// transformations
VPoly  hpoly_to_vpoly(Matrix hpoly);
Matrix vpoly_to_hpoly(VPoly vpoly);
```

`vcone.h` and `vcone.cpp` implement the reductions from Polyhedra to Cones.

```
Matrix project_zero(Matrix M) {
  transform(M.begin(), M.end(), M.begin(),
    [](const Vector &v) {
      return v[slice(1,v.size()-1,1)];
    });
  return M;
}
```

Using the `std::slice` object, the first coordinate of each `Vector` is dropped from the `Matrix`.

```
Matrix normalize_P(Matrix M) {
  Matrix result;
  copy_if(M.begin(), M.end(), back_inserter(result),
    [](const Vector &v) { return v[0] > 0; }
  );
  for (auto &&v : result) {
    if (v[0] != 1) {
      v /= v[0];
    }
  }
  return result;
}
```

Creates the set V in (2.16). Note that the operation U^i/U_0^i is only done if U_0^i is not already 1

```
Matrix hpoly_to_hcone(Matrix hpoly) {
  transform(hpoly.begin(), hpoly.end(), hpoly.begin(),
      [](const Vector &v) {
        auto tmp = v.cshift(-1);
        tmp[0] *= -1;
        return tmp;
      });
  return hpoly;
}
```

Each `Vector` is supposed to be of the form (A_i, b), expressing the constraint $\langle A_i, \mathbf{x}\rangle \leq b$. b is moved to the first coordinate, and negated to transform

$$[A|b] \to [-b|A]$$

```
Matrix hcone_to_hpoly(Matrix hcone) {
  transform(hcone.begin(), hcone.end(), hcone.begin(),
      [](const Vector &v) {
        auto tmp = v.cshift(1);
        tmp[tmp.size()-1] *= -1;
        return tmp;
      });
  return hcone;
}
```

This is the inverse to `hcone_to_hpoly`:

$$[-b|A] \to [A|b]$$

```
Matrix vpoly_to_vcone(VPoly vpoly) {
  //requires increase in dimension
  ++d;
  Matrix result;
  for (auto &&v : vpoly.U) {
    result.emplace_back(v.size()+1);
```

```
    result.back()[0] = 0;
    copy(begin(v), end(v), next(begin(result.back())));
  }
  for (auto &&v : vpoly.V) {
    result.emplace_back(v.size()+1);
    result.back()[0] = 1;
    copy(begin(v), end(v), next(begin(result.back())));
  }
  return result;
}
```

`d` must be increased for operations which depend on it to function correctly. Then conducts the transform:

$$\mathrm{cone}(U) + \mathrm{conv}(V) \to \mathrm{cone}\begin{pmatrix} \mathbf{0} & \mathbf{1} \\ U & V \end{pmatrix}$$

```
VPoly vcone_to_vpoly(Matrix vcone) {
  VPoly result;
  --d;
  result.U = project_matrix(fourier_motzkin(vcone, 0));
  for (auto &&v : vcone) {
    // handle case v[0] == 1 separately to avoid
    // floating point shenanigans
    if (v[0] == 1) {
      result.V.push_back(v[slice(1,v.size()-1,1)]);
    } else if (v[0] > 1) {
      result.V.push_back(v[slice(1,v.size()-1,1)]);
      result.V.back() /= v[0];
    }
  }
  return result;
}
```

This implements the tranformation justified by (2.16), returning the result of

$$\texttt{vcone} \cap \{\mathbf{x} \mid \langle \mathbf{e}_0, \mathbf{x} \rangle = 1\}$$

```
VPoly hpoly_to_vpoly(Matrix hpoly) {
  return vcone_to_vpoly(
           hcone_to_vcone(
             hpoly_to_hcone(move(hpoly))));
}
```

```
Matrix vpoly_to_hpoly(VPoly vpoly) {
  return hcone_to_hpoly(
           vcone_to_hcone(
             vpoly_to_vcone(move(vpoly))));
}
```

Using the other functions in the file, these implement the described transformations.