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 [1], and uses Fourier-Motzkin elimination. A C++ implementation is given for the enumeration algorithm suggested by the proof, as well a means of testing the implementation against some special polyhedra. The Farkas Lemma is then proven and used to prove the validity of the testing methods.

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 $\operatorname{cone}(U)$. 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 $\operatorname{conv}(V)$.

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

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

is called a *V-Polyhedron.*

Note: Given two sets P and Q, the set $P + Q = \{p + q \mid p \in p,\, q \in Q\}$ is called the *Minkowski Sum* of P and Q. Therefore, we will write a V-Polyhedron as $\operatorname{cone}(U) + \operatorname{conv}(V)$ for some U and V.

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 $C = \operatorname{cone}(U)$, 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 $A' \in \mathbb{R}^{(p+2d)\times(d+p)}$:

$$A' = \begin{pmatrix} \mathbf{0} & -I \\ I & -U \\ -I & U \end{pmatrix} \tag{2.3}$$

then we can rewrite 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:

$$\operatorname{cone}(U) = \Pi\left(\left\{\mathbf{y} \in \mathbb{R}^{d+p} \mid A'\mathbf{y} \leq \mathbf{0}\right\}\right) \tag{2.4}$$

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 $(\forall i)\,\langle B_i, \mathbf{y}\rangle \leq 0$. Because the last p columns of B are $\mathbf{0}$, any row B_i of B can be written $(B_i', \mathbf{0})$, with $\mathbf{0} \in \mathbb{R}^p$. We can also rewrite $\mathbf{y} \in \mathbb{R}^{d+p}$ 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. □

In order to use the above proposition, we need a matrix with columns which are $\mathbf{0}$. The next proposition shows us how to obtain such a matrix from another, while maintaining certain properties.

Proposition 3. *Let $B \in \mathbb{R}^{m_1 \times (d+p)}$, $1 \leq k \leq (d+p)$, 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:

$$\begin{aligned} \langle C_l, \mathbf{e}_k\rangle &= 0 \\ \langle C_{ij}, \mathbf{e}_k\rangle &= \left\langle B_i^k B_j - B_j^k B_i, \mathbf{e}_k\right\rangle = B_i^k B_j^k - B_j^k B_i^k = 0 \end{aligned} \tag{2.5}$$

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 (2.5), 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 (2.5), 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

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

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

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

In particular,

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

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,

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

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. $\square$

Remark: Proposition 3 highlights the properties of the matrix B'. Upon close inspection, we can create a Matrix Y such that $B' = YB$, and every element of Y is non-negative. Create the following set of row vectors Y

$$\begin{array}{ll} \mathbf{e}_l & \mid l \in Z \\ B_i^k \mathbf{e}_j - B_j^k \mathbf{e}_i & \mid i \in P,\ j \in N \end{array}$$

Since the basis vectors simply select rows during matrix multiplication, it is clear that

$$B' = YB$$

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 that

$$\Pi\left\{\mathbf{y} \in \mathbb{R}^{d+p} \mid A'\mathbf{y} \leq \mathbf{0}\right\} = \left\{\mathbf{x} \in \mathbb{R}^{d} \mid A\mathbf{x} \leq \mathbf{0}\right\} \tag{2.6}$$

This follows from the following:

$$A'\mathbf{y} \leq \mathbf{0} \Rightarrow A(\Pi(\mathbf{y})) \leq \mathbf{0} \tag{2.7}$$

$$A\mathbf{x} \leq \mathbf{0} \Rightarrow (\exists t_1)\dots(\exists t_p)A'(\mathbf{x} + t_1\mathbf{e}_{d+1} + \cdots + t_p\mathbf{e}_{d+p}) \leq \mathbf{0} \tag{2.8}$$

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:

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

Because $A' = B_0$, and A is B_p with the last p columns deleted, (2.7) and (2.8) hold, therefore (2.6) 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:

$$\left\{\mathbf{x} \in \mathbb{R}^d \;\middle|\; (\exists \mathbf{t} \geq 0) \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} = U'\mathbf{t}\right\} \tag{2.9}$$

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:

$$\Pi\Big(\operatorname{cone}(U') \cap \{x_{d+1} = 0\} \cap \cdots \cap \{x_{d+m} = 0\}\Big) \tag{2.10}$$

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':

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

Note that U' can be written:

$$U' = \begin{pmatrix} \mathbf{0} & I & -I \\ I & A & -A \end{pmatrix} \tag{2.11}$$

We then claim:

$$\left\{\mathbf{x} \in \mathbb{R}^d \mid A\mathbf{x} \leq \mathbf{0}\right\} = \left\{\mathbf{x} \in \mathbb{R}^d \;\middle|\; (\exists \mathbf{t} \geq 0) \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} = U'\mathbf{t}\right\} \tag{2.12}$$

First, considering (2.9) and (2.10), 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

$$\begin{pmatrix} \mathbf{x} \\ A\mathbf{x} \end{pmatrix} = \sum_{1 \leq j \leq d} x_j^+ \begin{pmatrix} \mathbf{e}_j \\ A^j \end{pmatrix} + \sum_{1 \leq j \leq d} x_j^- \begin{pmatrix} -\mathbf{e}_j \\ -A^j \end{pmatrix} \tag{2.13}$$

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

$$A\mathbf{x} \leq \mathbf{0} \Leftrightarrow (\exists \mathbf{w} \geq \mathbf{0}) \mid \begin{pmatrix} \mathbf{x} \\ A\mathbf{x} \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \mathbf{w} \end{pmatrix} = \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} \tag{2.14}$$

(2.13) and (2.14) 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 (2.14) we have $A\mathbf{x} \leq \mathbf{0}$. So (2.12) 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 columns of Y.*
2. *n_2 is finite.*
3. *The k-th row of Y' is $\mathbf{0}$.*
4. *$(\exists \mathbf{t} \geq \mathbf{0})\mathbf{x} = Y\mathbf{t} \Leftrightarrow (\exists \mathbf{t}' \geq \mathbf{0})\mathbf{x} = Y'\mathbf{t}'$*

Proof. We partition the columns of Y:

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

We then define Y':

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

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

$$\begin{aligned} &\left\langle Y'^l, \mathbf{e}^k \right\rangle = 0 \\ &\left\langle Y'^{ij}, \mathbf{e}^k \right\rangle = \left\langle Y_k^i Y^j - Y_k^j Y^i, \mathbf{e}^k \right\rangle = Y_k^i Y_k^j - Y_k^j Y_k^i = 0 \end{aligned} \tag{2.15}$$

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

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

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

Then, by proposition 1 (cone closure), 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

$$\sum_{i \in P} t_i Y^i + \sum_{j \in N} t_j Y^j = \sum_{\substack{i\in P \\ j \in N}} t_{ij} (Y_k^i Y^j - Y_k^j Y^i) \tag{2.16}$$

Proposition 5. *Suppose that*

$$\sum_{i \in P} t_i Y_{d+1}^i + \sum_{j \in N} t_j Y_{d+1}^j = 0 \qquad Y_k^j < 0 < Y_k^i$$

Then the following holds

$$\begin{aligned} (t_i, t_j \geq 0) &\Rightarrow (\exists t_{ij} \geq 0) \quad \text{such that (2.16) holds} \\ (t_{ij} \geq 0) &\Rightarrow (\exists t_i, t_j \geq 0) \text{ such that (2.16) holds} \end{aligned}$$

Proof. First note that if all $t_i = 0, t_j = 0$, then choosing $t_{ij} = 0$ satisfies (2.16), likewise if all $t_{ij} = 0$, then $t_i = 0, t_j = 0$ satisfies (2.16). So suppose that some $t_i \neq 0, t_j \neq 0, t_{ij} \neq 0$.

The right hand side of (2.16) can be written

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

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

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

$\square$

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. $\square$

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 $\text{cone}(Y')$. $\square$

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,

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

$\square$

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

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

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:

$$A\mathbf{x} \leq \mathbf{b} \Leftrightarrow -\mathbf{b} + A\mathbf{x} \leq \mathbf{0} \Leftrightarrow \left[-\mathbf{b}|A\right] \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} \leq \mathbf{0} \tag{2.17}$$

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 (2.17):

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

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. □

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 $\left\{\mathbf{y} \in \mathbb{R}^{d+1} \mid y_0 = 1\right\}$. Given two sets $V \in \mathbb{R}^{d \times n}$ and $U \in \mathbb{R}^{d \times p}$, the V-Polyhedron is given by:

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

It isn't hard to see that

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

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 $\left\{\mathbf{y} \in \mathbb{R}^{d+1} \mid y_0 = 1\right\}$ and get a V-Polytope out of it. So let

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

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:

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

Then I claim that

$$C_V = \{\mathbf{x} + \mathbf{y} \mid \mathbf{x} \in \text{cone}(U'),\, \mathbf{y} \in \text{conv}(V)\} \tag{2.18}$$

Say $\mathbf{x} \in \text{cone}(U')$, $\mathbf{x}$ 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 $\mathbf{x} \in \text{cone}(U)$. 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 $\mathbf{y} \in \text{cone}(U)$, and then $\mathbf{x} + \mathbf{y} \in \text{cone}(U)$. 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 (cone closure), $\mathbf{x} + \mathbf{y} \in C_V$.

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:

$$\begin{aligned}
\boldsymbol{\sigma}_Z &= \sum_{l \in Z} t_l U^l, & \sigma_l &= \sum_{l \in Z} t_l U_0^l = 0 \\
\boldsymbol{\sigma}_P &= \sum_{i \in P} t_i U^i, & \sigma_i &= \sum_{i \in P} t_i U_0^i \\
\boldsymbol{\sigma}_N &= \sum_{j \in N} t_j U^j, & \sigma_j &= \sum_{j \in N} t_j U_0^j
\end{aligned}$$

Then it holds that

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

$$\boldsymbol{\sigma}_P = \boldsymbol{\sigma}_P/\sigma_i + (1 - 1/\sigma_i)\boldsymbol{\sigma}_P = \boldsymbol{\sigma}_P/\sigma_i - (\sigma_j/\sigma_i)\boldsymbol{\sigma}_P$$

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

$$\mathbf{z} = \boldsymbol{\sigma}_Z + \boldsymbol{\sigma}_P + \boldsymbol{\sigma}_N = \boldsymbol{\sigma}_Z + \frac{\boldsymbol{\sigma}_P}{\sigma_i} - \frac{\sigma_j}{\sigma_i}\boldsymbol{\sigma}_P + \frac{\sigma_i}{\sigma_i}\boldsymbol{\sigma}_N = \boldsymbol{\sigma}_Z + \frac{\boldsymbol{\sigma}_P}{\sigma_i} + \frac{\sigma_i\boldsymbol{\sigma}_N - \sigma_j\boldsymbol{\sigma}_P}{\sigma_i}$$

Using proposition 1, we need only show that

1. $\boldsymbol{\sigma}_Z \in \operatorname{cone}(U')$
2. $(\sigma_i\boldsymbol{\sigma}_N - \sigma_j\boldsymbol{\sigma}_P) \in \operatorname{cone}(U')$
3. $\boldsymbol{\sigma}_P/\sigma_i \in \operatorname{conv}(V)$

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

$$\sigma_i\boldsymbol{\sigma}_N - \sigma_j\boldsymbol{\sigma}_P = \sum_{i\in P} t_i \sum_{j \in N} t_j U_0^i U^j - \sum_{j\in N} t_j \sum_{i\in P} t_i U_0^j U^i = \sum_{\substack{i\in P\\ j\in N}} t_i t_j (U_0^i U^j - U_0^j U^i)$$

So (2) holds. Finally,

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

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

2.4 Picture of the Proof

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

$$\begin{tikzcd}
\boxed{A\mathbf{x} \le \mathbf{b}} \arrow[dd, leftrightarrow] & & \boxed{\operatorname{cone}(U) + \operatorname{conv}(V)} \arrow[dd, leftrightarrow] \\
& \boxed{\operatorname{cone}\begin{pmatrix} 0 & I & -I \\ I & A & -A \end{pmatrix}} \arrow[dr, "\bigcap"] & \\
\boxed{[-\mathbf{b}|A]\left(\begin{smallmatrix} x_0 \\ \mathbf{x} \end{smallmatrix}\right) \le \mathbf{0}} \arrow[ur, "\text{lift}"] & & \boxed{\operatorname{cone}\{\left(\begin{smallmatrix} \mathbf{0} \\ U' \end{smallmatrix}\right) \cup \left(\begin{smallmatrix} \mathbf{1} \\ V' \end{smallmatrix}\right)\}} \arrow[dl, "\text{lift}"] \\
& \boxed{\begin{pmatrix} \mathbf{0} & -I \\ I & -U \\ -I & U \end{pmatrix} \mathbf{x} \le \mathbf{0}} \arrow[ul, "\Pi"] &
\end{tikzcd}$$

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 transformations have been implemented in C++. Program `main.cpp` takes one argument specifying the type of input object. It reads the description of the object from standard input, and writes the result of the implied transformation to standard output. If no arguments are supplied, then a `usage` message is given. The `usage` message, which also contains the input format for the objects, is:

```
usage: ./main input_type

The input object is read on stdin, and the result of the
transform to sent to stdout.  input_type determines the type
of input and output:

    -vc  # transforms a vcone into an hcone
    -vp  # transforms a vpolyhedron into an hpolyhedron
    -hc  # transforms an hcone into a vcone
    -hp  # transforms an hpolyhedron into a vpolyhedron

input format is as follows:
  hcone := dimension   ws (vector ws)*
  vcone := dimension   ws (vector ws)*
  hpoly := dimension+1 ws (vector ws constraint ws)*
  vpoly := dimension   ws ('U' | 'V') ws vpoly_vecs*

  ws         := whitespace, as would be read by "cin >> ws;"
  dimension  := a positive integer.  For hpoly, add one to
                the dimension of the space (this extra
                dimension is for the constraint)
  vector     := (dimension) doubles separated by whitespace
  constraint := a double (the value b_i in <A_i,x> <= b_i)
  'V' | 'U'  := the literal character 'U' or 'V'
  vpoly_vecs := (['U'] ws vector) | (['V'] ws vector)

VPOLY ONLY:

  vpoly contains two matrices:
    U - contains the rays of the vpolyhedron
    V - contains the points of the vpolyhedron

  On input, enter 'U' or 'V' to indicate which matrix should
  receive the vectors that follow.  You can switch back and
  forth as you like, but either 'U' or 'V' must be entered
  before starting to input vectors.

EXAMPLES:
```

```
$ ./main -vc <<< "2 1 0"

OUTPUT:

2
-0 -1
0 1
0 0
-1 0

$ ./main -hc <<< "2 1 0 0 1"

OUTPUT:

2
-1 0
0 0
0 0
0 -1

$ ./main -vp <<< "2 U 1 0 V 0 0 1 1"

OUTPUT:

3
0 0 -0
0 0 -0
0 1 1
0 0 1
-1 1 -0
-1 0 -0
0 0 -0
0 -1 -0
$ ./main -hp <<< "3 0 -1 0 0 1 1 -1 1 0"

OUTPUT:

2
U
1 0
0 0
0 0
0 0
V
0 0
1 1
```

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
`linear_algebra.h`	
`fourier_motzkin.h`	`linear_algebra.h`
`polyhedra.h`	`fourier_motzkin.h`
`main.cpp`	`polyhedra.h`
`test_functions.h`	`linear_alebra.h`
`test.cpp`	`test_functions.h`, `polyhedra.h`

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

- `linear_algebra.h`
  Types `Vector` and `Matrix`, and some basic functionality for them
- `fourier_motzkin.h`
  Fourier Motzkin elimination, Minkowski-Weyl Theorem for cones
- `polyhedra.{cpp,h}`
  Transforms between polytopes and polyhedra, Minkowski-Weyl Theorem
- `test_functions.h`
  Types and functions for testing the algorithms
- `test.cpp`
  Test cases for the algorithms and the functions from `test_functions.h`

3.1 Code

The relevant code will be displayed with commentary below. Some of the code relating to C++ specific technicalities and I/O is ommitted.

3.2 `linear_algebra.h`

The types `Vector` and `Vectors` are used in the representation of polyhedra. The `std::valarray` template is used because it has built-in vector-space operations (sum and scaling). `std::vector`, is used as a container of `Vectors`, however other containers could be used.

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

The `class` `Matrix` implements a subset of what a *C++ Container* should. It is the primary type for representing polyhedra, and directly represents Cones, as well as H-Polyhedra. The interface is designed to enforce the following invariant:

$(\forall$`v` $\in$ `vectors`$)$ `v.size() == d`

The *factory* function `read_Matrix` is provided to read a `Matrix` from an `istream`. It is necessary because the value of `d` can't be known before reading some of the stream.

```
class Matrix {
// invariant: d >= 0
// invariant: (forall valid i) vectors[i].size() == d
public:
  const size_t d; // size of all Vectors
private:
  Vectors vectors;
public:
  // needed for back_insert_iterator
  using value_type = Vector;

  Matrix(size_t d);
  Matrix(std::initializer_list<Vector>&&);
  bool check() const; // checks each Vector has size d

  //defaults don't work because of const member
  Matrix(const Matrix&);
  Matrix(Matrix&&);
  Matrix &operator=(const Matrix&);
  Matrix &operator=(Matrix&&);
  Matrix &operator=(std::initializer_list<Vector>&&);

  static Matrix read_Matrix(std::istream&);

  Vectors::iterator       begin();
  Vectors::iterator       end();
  Vectors::const_iterator begin() const;
  Vectors::const_iterator end()   const;

  bool    empty() const;
  size_t  size()  const;
  Vector& back();

  Vector& add_Vector();
  void push_back(const Vector &v);
  void push_back(Vector &&v);
};
```

The `struct` `VPoly` gather two `Matrix`s needed to represent a V-Polyhedron. The `Matrix U` corresponds to the rays that generate the cone, and the `Matrix V` corresponds to the points, i.e.

$$\texttt{vpoly} = \mathrm{cone}(\texttt{vpoly.U}) + \mathrm{conv}(\texttt{vpoly.V})$$

```
struct VPoly {
  const size_t d;
  Matrix U; // rays
  Matrix V; // points

  VPoly(size_t d) : d{d}, U{d}, V{d} {}
  VPoly(std::initializer_list<Vector>&&,
        std::initializer_list<Vector>&&);
```

```
  bool check() const;

  static VPoly read_VPoly(std::istream&);
};
```

The class input_error is thrown to indicate an invalid input to the program, and provide some clue as to why it failed. Here are two command line examples:

```
$ ./main -vc <<< "0"
terminate called after throwing an instance of 'input_error'
  what():  bad d: 0
Aborted (core dumped)
$ ./main -vc <<< "2 1"
error reading matrix, vector 1
terminate called after throwing an instance of 'input_error'
  what():  failed to read vector: istream failed
Aborted (core dumped)
```

```
class input_error : public std::runtime_error {
public:
  input_error(const char*s);
  input_error(const std::string &s);
};
```

operator>> and operator<< implement the input format described in usage.txt.

```
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&);
```

usage() outputs the usage message shown above.

```
int usage();
```

3.3 linear_algebra.cpp

e_k creates the canonical basis Vector $\mathbf{e}_k \in \mathbb{R}^d$.

```
Vector e_k(size_t d, size_t k) {
  Vector result(d);
  result[k] = 1;
  return result;
}
```

concatentate takes the Vectors l $\in \mathbb{R}^{\texttt{l.size()}}$ and r $\in \mathbb{R}^{\texttt{r.size()}}$ and returns the Vector (l,r) $\in \mathbb{R}^{\texttt{l.size() + r.size()}}$

```
Vector concatenate(const Vector &l, const Vector &r) {
  Vector result(l.size() + r.size());
  copy(begin(l), end(l), begin(result));
  copy(begin(r), end(r), next(begin(result), l.size()));
  return result;
}
```

`get_column` returns the `k`-th column of the `Matrix M`. Note that while a `Matrix` may logically represent either a collection of row or column `Vector`s, `get_column` is only used in the function `transpose`, where this distinction is unimportant.

```
Vector get_column(const Matrix &M, size_t k) {
  if (!(0 <= k && k < M.d)) {
    throw std::out_of_range("k < 0 || M.d <= k");
  }
  Vector result(M.size());
  size_t result_row{0};
  for (auto &&row : M) {
    result[result_row++] = row[k];
  }
  return result;
}
```

`transpose` returns the transpose of `Matrix M`.

```
Matrix transpose(const Matrix &M) {
  if (M.empty()) {
    return M;
  }
  Matrix result{M.size()};
  // for every column of M,
  for (size_t k = 0; k < M.d; ++k) {
    result.push_back(get_column(M,k));
  }
  return result;
}
```

A `slice` object can be used to conveniently obtain a subset of a `valarray`. `slice_matrix` returns the `Matrix` obtained by applying the `slice s` to each `Vector` of the `Matrix`.

```
Matrix slice_matrix(const Matrix &M, const std::slice &s) {
  Matrix result{s.size()};
  transform(M.begin(), M.end(), back_inserter(result),
    [s](const Vector &v) { return v[s]; });
  return result;
}
```

3.4 fourier_motzkin.h

`Lift` is a function pointer `typedef` that is used in a generic cone-transformation function.

```
typedef Matrix(*Lift)(const Matrix&);
```

3.5 fourier_motzkin.cpp

A `slice` object is determined by three fields: `start`, `size`, and `stride`, and implicitly represents all indices of the form:

$$\sum_{0 \leq k < \texttt{size}} \texttt{start} + k \cdot \texttt{stride}$$

Therefore:

$$i \in \texttt{slice} \Leftrightarrow i - \texttt{start} \equiv 0 \mod (\texttt{stride}), \quad \texttt{start} \leq i \leq \texttt{start} + \texttt{stride} \cdot \texttt{size}$$

```
bool index_in_slice(size_t index, const slice &s) {
  return ((index - s.start()) % s.stride() == 0) &&
          s.start() <= index &&
          index <= s.start() + s.stride()*(s.size()-1);
}
```

`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.

```
Matrix fourier_motzkin(Matrix M, size_t k) {
  Matrix result{M.d};
  // 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 n_it = p_end; n_it != M.end(); ++n_it) {
      result.push_back(
        (*p_it)[k]*(*n_it) - (*n_it)[k]*(*p_it));
    }
  }
  return result;
}
```

The lines:

```
  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; });
```

Partition `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

The line:

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

Moves Z into the result. The lines:

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

$$\left\{B_i^k B_j - B_j^k B_i \mid i \in P,\, j \in N\right\}, \quad \left\{Y_k^i Y^j - Y_k^j Y^i \mid i \in P,\, j \in N\right\}$$

`sliced_fourier_motzkin` applies `fourier_motzkin` to `Matrix M` for each $k \notin$ `s`, then slices the resulting `Matrix` using `slice_matrix` and `s`. This is the realization of the algorithms indicated by the proofs of either direction of the Minkowski-Weyl Theorem for cones.

```
Matrix sliced_fourier_motzkin(Matrix M, const slice &s) {
  for (size_t k = 0; k < M.d; ++k) {
    if (!index_in_slice(k,s)) {
      M = fourier_motzkin(M, k);
    }
  }
  return slice_matrix(M, s);
}
```

When transforming an H-Cone to a V-Cone, it first must be written as a V-Cone of a new matrix, then it is intersected with coordinate-hyperplanes and projected. Similarly, when a V-Cone is transformed into an H-Cone, it must be written as and H-Cone of a new matrix then projected with coordinate-projections. The transformations are described in (2.3) and (2.11), and summarized here:

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

Note that the tranformation of U can be written:

$$U \to \begin{pmatrix} \mathbf{0} & I \\ I & A \\ -I & -A \end{pmatrix}^T$$

Remembering that a `Matrix` is either a collection of row *or* column `Vector`s, it is not surprising that these two transformations can be written as one function of a `Matrix` and some coefficients. In `generalized_lift`, the coefficients are given as an `array<double, 5> C`, so the overall transformation can be illustrated as:

$$\texttt{Matrix M} \to \begin{pmatrix} \mathbf{0} & \texttt{C[0]}I \\ \texttt{C[1]}I & \texttt{C[2]M} \\ \texttt{C[3]}I & \texttt{C[4]M} \end{pmatrix}$$

where `Matrix M` is a collection of row `Vector`s, or

$$\texttt{Matrix M} \rightarrow \begin{pmatrix} & \mathbf{0} & \texttt{C[1]}I & \texttt{C[3]}I \\ \texttt{C[0]}I & & \texttt{C[2]M} & \texttt{C[4]M} \end{pmatrix}$$

where `Matrix M` is a collection of column `Vector`s.

```
Matrix generalized_lift(const Matrix &cone,
                        const array<double,5> &C) {
  const size_t d = cone.d;
  const size_t n = cone.size();
  Matrix result{d+n};
  Matrix cone_t = transpose(cone);
  // |0  C[0]*I|  |0      |
  //              |C[0]*I|
  for (size_t i = 0; i < n; ++i) {
    result.add_Vector()[d+i] = C[0];
  }
  size_t k = 0;
  // |C[1]*I C[2]*U|  |C[1]*I|
  //                  |C[2]*A|
  for (auto &&row_t : cone_t) {
    result.push_back(
      concatenate(C[1]*e_k(d,k++), C[2]*row_t));
  }
  k = 0;
  // |C[3]*I C[4]*U|  |C[3]*I|
  //                  |C[4]*A|
  for (auto &&row_t : cone_t) {
    result.push_back(
      concatenate(C[3]*e_k(d,k++), C[4]*row_t));
  }
  return result;
}
```

`lift_vcone` and `lift_hcone` implement the appropriate transformation using `generalized_lift` and providing the appropriate coefficients in `array<double, 5> C`.

```
Matrix lift_vcone(const Matrix &vcone) {
  return generalized_lift(vcone, {-1,1,-1,-1,1});
}
```

```
Matrix lift_hcone(const Matrix &hcone) {
  return generalized_lift(hcone, {1,1,1,-1,-1});
}
```

`cone_transform` consolidates the logic of the V-Cone → H-Cone and H-Cone → V-Cone transformations by accepting a `Matrix cone` and a `Lift`.

```
Matrix cone_transform(const Matrix &cone, Lift lift) {
  if (cone.empty()) {
    throw logic_error{"empty cone for transform"};
  }
  return sliced_fourier_motzkin(
```

```
    lift(cone), slice(0, cone.d, 1));
}
```

`vcone_to_hcone` and `hcone_to_vcone` specialize `cone_transform` by providing the appropriate `Lift`.

```
Matrix vcone_to_hcone(Matrix vcone) {
  return cone_transform(vcone,lift_vcone);
}
```

```
Matrix hcone_to_vcone(Matrix hcone) {
  return cone_transform(hcone,lift_hcone);
}
```

3.6 polyhedra.cpp

`hpoly_to_hcone` and `hcone_to_hpoly` implement the `Matrix` transforms:

$$\texttt{hpoly\_to\_hcone} : (A|b) \to (-b|A), \quad \texttt{hcone\_to\_hpoly} : (-b|A) \to (A|b)$$

These very simple transforms are done with the `cshift` function, which “circularly shifts” the elements of a `Vector` (provided as part of the interface to `valarray`).

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

```
Matrix hcone_to_hpoly(Matrix hcone) {
  transform(hcone.begin(), hcone.end(), hcone.begin(),
      [](Vector v) {
        v[0] *= -1;
        return v.cshift(1);
      });
  return hcone;
}
```

`vpoly_to_vcone` implements the `VPoly` transform:

$$\texttt{vpoly} \to \begin{pmatrix} \mathbf{0} & \mathbf{1} \\ \texttt{vpoly.U} & \texttt{vpoly.V} \end{pmatrix}$$

```
Matrix vpoly_to_vcone(VPoly vpoly) {
  //requires increase in dimension
  Matrix result{vpoly.d+1};
  for (auto &&u : vpoly.U) {
    result.push_back(concatenate({0},u));
  }
```

```
  for (auto &&v : vpoly.V) {
    result.push_back(concatenate({1},v));
  }
  return result;
}
```

`normalized_P` calculates the V in (2.18). Let Π be the identity matrix with the 0-th row deleted, and $P = \{\mathbf{u} \in U : u_0 > 0\}$. then this is the result of:

$$\Pi(\text{cone}(P) \cap \{x_0 = 1\})$$

```
Matrix normalized_P(const Matrix &U) {
  if (U.d <= 1) {
    throw std::logic_error{"can't normalize U!"};
  }
  Matrix result{U.d-1};
  std::slice s{1,result.d,1};
  for (auto &&v : U) {
    // select the vectors with positive 0-th coordinate
    if (v[0] <= 0) { continue; }
    // normalize the selected vectors,
    result.push_back(v[0] == 1 ? v[s] : (v / v[0])[s]);
  }
  return result;
}
```

`vcone_to_vpoly` implements the full tranformation in (2.18).

```
VPoly vcone_to_vpoly(Matrix vcone) {
  VPoly result{vcone.d-1};
  result.U = sliced_fourier_motzkin(
                 vcone, slice(1,vcone.d-1,1));
  result.V = normalized_P(vcone);
  return result;
}
```

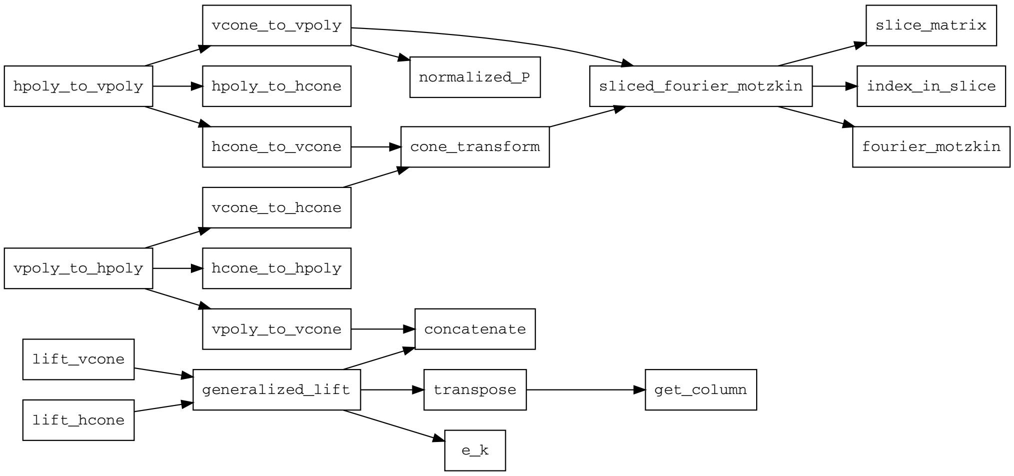
`hpoly_to_vpoly` and `vpoly_to_hpoly` implement the complete transformations promised by the file.

```
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))));
}
```

3.7 Picture of the Program

In the following diagram, the nodes represent functions, and the edges can be read as "calls." Such a diagram is known as a "callgraph," and is only intended to give an overview of the program.



4. Testing

In the next sections, the methods used for testing the program described above will be discussed.

Notation: Let $AU \leq \mathbf{b}$ be shorthand for $(\forall \mathbf{u} \in U) A\mathbf{u} \leq \mathbf{b}$.

4.1 Testing H-Cone $\rightarrow$ V-Cone

Suppose we have an H-Cone $C = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$, and would like to test if a V-Cone $\text{cone}(V')$ represents the same set. It's easy to check if

$$AV' \leq \mathbf{0} \Rightarrow \text{cone}(V') \subseteq C$$

It's not clear what to do to check if $C \subseteq \text{cone}(V')$. Suppose we had a set V, and we knew that $C = \text{cone}(V)$, and that $C = \text{cone}(V') \Rightarrow V \subseteq V'$. Then we'd have the following situation:

$$\begin{aligned} AV' \leq \mathbf{0} \quad &\Rightarrow \text{cone}(V') \subseteq C \\ V \subseteq V' \quad &\Rightarrow C \subseteq \text{cone}(V') \\ \text{cone}(V') = C &\Rightarrow V \subseteq V' \\ \text{cone}(V') = C &\Rightarrow AV' \leq \mathbf{0} \end{aligned}$$

The problem is now to come up with such a set V. We will need to relax the requirements on V a little bit, but not in a way that reduces its utility. The set is desribed in the next proposition, but first we introduce the notion of equivalence vectors:

Definition 10 (vector equivalence)**.** Let $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$, and suppose that $\mathbf{u}/\left|\left|\mathbf{u}\right|\right| = \mathbf{v}/\left|\left|\mathbf{v}\right|\right|$. Then say that $\mathbf{u}, \mathbf{v}$ are equivalent, and write:

$$\mathbf{u} \simeq \mathbf{v}$$

Definition 11 (Extreme)**.** Let $V \in \mathbb{R}^{d \times n}$, if no member of V is a positive linear combination of other elements of V then V is called *extreme.*

Proposition 7. *Let $V \in \mathbb{R}^{d\times n}$ be extreme, and $C = \text{cone}(V)$.. Then*

$$C = \text{cone}(V') \Rightarrow (\forall \mathbf{v} \in V)(\exists \mathbf{v}' \in V') : \mathbf{v} \simeq \mathbf{v}'$$

Proof. Let $\mathbf{v} \in V$, so that $\mathbf{v} = V\mathbf{e}_k$. Since $C = \text{cone}(V')$, there exists a matrix A with all non-negative entries such that $V' = VA$. There is also a non-negative b such that $\mathbf{v} = V'b$. Then $\mathbf{v} = (VA)b = V(A\mathbf{b})$. Since A and b contain only non-negative entries, so does $A\mathbf{b}$. Since $\mathbf{v}$ is not a non-negative combination of other vectors from V, $A\mathbf{b}$ must be the basis vector $\mathbf{e}_k$. Then if $i \neq k$, $\mathbf{e}_i^T(A\mathbf{b}) = 0$, or $(\mathbf{e}_i^T A)b = \sum_j A_i^j b_j = 0$. Since $A_i^j, b_j \geq 0$, we have:

$$\begin{aligned} (\forall i \neq k) \quad A_i^j > 0 \ &\Rightarrow \ b_j = 0 \\ (\forall i \neq k) \quad b_j > 0 \ &\Rightarrow A_i^j = 0 \end{aligned}$$

Furthermore, we have $\langle A_k, b\rangle = 1$, so for some l, $A_i^l, b_l > 0$. Then,

$$(\forall i \neq k) \quad A_i^l = 0$$

Now let $\mathbf{b}' = \mathbf{e}_l / A_k^l$. Then it immediately follows that $\langle A_k, \mathbf{b}'\rangle = 1$. Also,

$$(\forall i \neq k) \quad A_i^l = 0 \quad \Rightarrow \quad \langle A_i, b'\rangle = A_i^l / A_k^l = 0$$

We conclude that $A\mathbf{b}' = \mathbf{e}_k = A\mathbf{b}$, and that $\mathbf{v} = V(A\mathbf{b}) = V(A\mathbf{b}') = (VA)\mathbf{b}' = U(\mathbf{e}_l / A_k^l)$. If $\mathbf{v}' = U\mathbf{e}_l$, that is $\mathbf{v}'$ is the l-th vector of U, then $\mathbf{v} = \mathbf{v}'/A_k^l$, or

$$\mathbf{v}/\left|\left|\mathbf{v}\right|\right| = (\mathbf{v}'/A_k^l)/\left|\left|\mathbf{v}'/A_k^l\right|\right| = \mathbf{v}'/\left|\left|\mathbf{v}'\right|\right|$$

So $\mathbf{v} \simeq \mathbf{v}'$. $\square$

Let's denote $(\forall \mathbf{v} \in V)(\exists \mathbf{v}' \in V') : \mathbf{v} \simeq \mathbf{v}'$ as $V \sqsubseteq V'$. Then, considering the discussion before the proposition, we have the following result.

Proposition 8. *Say $V \in \mathbb{R}^{d\times n}$ is extreme, and let $C = \mathrm{cone}(V) = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$. Then*

$$C = \mathrm{cone}(V') \;\Leftrightarrow\; AV' \leq \mathbf{0},\, V \sqsubseteq V'$$

We now have a method for testing the program. First, we hand-craft an H-Cone $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$ based on some extreme set V, then run our program to get a set V', with the alleged property that $\mathrm{cone}(V') = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$. If we confirm that $\forall \mathbf{v}' \in V', A\mathbf{v}' \leq 0$ and $V \sqsubseteq V'$, then our program has succeeded.

4.1.1 Farkas Lemma

The procedure for the other direction is *almost* identical, but there is a slight catch. Call a set of row vectors **extreme** if no row is a non-negative combination of the other. We would be able to use proposition 7 to say something similar about an extreme set of row vectors A, if we could say that:

Proposition 9.

$$\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\} = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\} \Leftrightarrow \mathrm{cone}(A^T) = \mathrm{cone}(A'^T)$$

To prove this proposition, we use the Farkas Lemma:

Proposition 10 (The Farkas Lemma)**.** *Let $U \in \mathbb{R}^{d\times n}$. Precisely one of the following is true:*

$$(\exists \mathbf{t} \geq \mathbf{0}) : \mathbf{x} = U\mathbf{t}$$

$$(\exists \mathbf{y}) : U^T\mathbf{y} \leq 0,\ \langle \mathbf{x}, \mathbf{y}\rangle > 0$$

Proof. That both can't be true can be seen by:

$$\mathbf{x} = U\mathbf{t} \quad \Rightarrow \quad \mathbf{y}^T\mathbf{x} = \mathbf{y}^T U\mathbf{t} \quad \Rightarrow \quad 0 \neq 0$$

To see that at least one is true we must reconsider the process of converting a V-Cone to an H-Cone. First, from cone(U) we create the following matrix:

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

By the way A is constructed,

$$(\exists \mathbf{t}) : A \begin{pmatrix} \mathbf{x} \\ \mathbf{t} \end{pmatrix} \leq \mathbf{0} \Leftrightarrow (\exists \mathbf{t} \geq \mathbf{0})\ \mathbf{x} = U\mathbf{t} \tag{4.1}$$

In the proof of the transformation, we use proposition 3 to transform that matrix A. The remark 2.1 after the proof of the proposition promises a sequence of matrices $Y_{d+1}, \ldots, Y_{d+n}$ with certain properties. Let $Y = (Y_{d+n})(Y_{d+(n-1)}) \ldots (Y_{d+1})$, then it can be said of Y:

1. Every element of Y is non-negative.
2. Y is finite.
3. The last n columns of YA are all $\mathbf{0}$.
4. $(\exists t_{d+1}, \ldots, t_{d+n}) A(\mathbf{x} + \sum_{i=d+1}^{d+n} t_i \mathbf{e}_i) \leq \mathbf{0} \Leftrightarrow (YA)\mathbf{x} \leq \mathbf{0}$

Note that here $\mathbf{x} \in \mathbb{R}^{d+n}$. A has three blocks of rows, which can be labeled with Z, P, N in a fairly obvious way. Then, Y can be broken up into three blocks of columns, so that

$$Y = (Y_Z\ Y_P\ Y_N)$$

Where each of $Y_Z, Y_P, Y_N \geq \mathbf{0}$. Consolidating what is known about A and Y,

$$YA = (Y_Z\ Y_P\ Y_N) \begin{pmatrix} \mathbf{0} & -I \\ I & -U \\ -I & U \end{pmatrix} = (Y'\ \mathbf{0})$$

Here, we have let $Y' = Y_P - Y_N$. Then it follows that

$$\mathbf{0} = -Y_Z - Y_P(U) + Y_N(U) = -Y_Z - Y'(U) \ \Rightarrow\ Y_Z = -Y'U \ \Rightarrow\ Y'U \leq \mathbf{0}$$

Then it holds that, for any row $\mathbf{y}' \in Y'$:

$$\mathbf{y}'U \leq \mathbf{0} \tag{4.2}$$

It is also true that

$$(YA) \begin{pmatrix} \mathbf{x} \\ \mathbf{t} \end{pmatrix} = (Y'\ \mathbf{0}) \begin{pmatrix} \mathbf{x} \\ \mathbf{t} \end{pmatrix} = Y'\mathbf{x}$$

We also have

$$(\exists \mathbf{t}) : A \begin{pmatrix} \mathbf{x} \\ \mathbf{t} \end{pmatrix} \leq \mathbf{0} \Leftrightarrow (YA) \begin{pmatrix} \mathbf{x} \\ \mathbf{t} \end{pmatrix} \leq \mathbf{0} \Leftrightarrow Y'\mathbf{x} \leq \mathbf{0} \tag{4.3}$$

Note that here $\mathbf{x} \in \mathbb{R}^d$. So, if given some $\mathbf{x}$, the left side of (4.3) is not satisfied, then neither is the right, and there must be some row $\mathbf{y}' \in Y'$ such that the following holds:

$$\langle \mathbf{y}', \mathbf{x} \rangle > 0 \tag{4.4}$$

Then we conclude that, if the right side of (4.1) fails, then there is a vector $\mathbf{y}' \in Y'$ satisfying (4.2) and (4.4). $\square$

4.2 Testing V-Cone → H-Cone

Now we can prove proposition 9:

$$\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\} = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\} \Leftrightarrow \text{cone}(A^T) = \text{cone}(A'^T)$$

Proof. First suppose that $\text{cone}(A^T) = \text{cone}(A'^T)$. Then there exists a non-negative matrix B such that $A'^T = A^T B$. Then $A\mathbf{x} \leq \mathbf{0} \Rightarrow B^T A\mathbf{x} \leq \mathbf{0} \Rightarrow A'\mathbf{x} \leq \mathbf{0}$. Precisely the same reasoning shows that $A'\mathbf{x} \leq \mathbf{0} \Rightarrow A\mathbf{x} \leq \mathbf{0}$, and we conclude that $\text{cone}(A^T) = \text{cone}(A'^T) \Rightarrow \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\} = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\}$.

Next suppose that $\text{cone}(A^T) \neq \text{cone}(A'^T)$, that is, let $\mathbf{z} \in \text{cone}(A), \mathbf{z} \notin \text{cone}(A')$. We must show that $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\} \neq \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\}$. By the Farkas Lemma, we have a $\mathbf{y}$ such that $\langle \mathbf{y}, \mathbf{z} \rangle > 0,\ A'\mathbf{y} \leq \mathbf{0}$. Clearly this means that $\mathbf{y} \in \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\}$. Since $\mathbf{z} \in \text{cone}(A)$, there is some $(\mathbf{t} \geq \mathbf{0}) : \mathbf{z}^T = \mathbf{t}^T A$. Then if $A\mathbf{y} \leq \mathbf{0}$, we would have $\langle \mathbf{y}, \mathbf{z} \rangle = \mathbf{t}^T A\mathbf{y} \leq 0 < \langle \mathbf{y}, \mathbf{z} \rangle$, a contradiction. So we conclude that $\mathbf{y} \notin \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$. □

The Test *Finally*, we can make a test for the transformation from V-Cone to H-Cone. Let A be an *extreme* set of row vectors. Then clearly A^T is an extreme set of column vectors, and we can use propositions 7 and 9 to say that

$$\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\} = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\} \Leftrightarrow \text{cone}(A^T) = \text{cone}(A'^T) \Rightarrow A^T \sqsubseteq A'^T$$

Given the last implication, we can also write $A \sqsubseteq A'$, where we are now considering A and A' as sets of row vectors as opposed to column vectors.

Now, as before, we suppose that we have a set V and A, where A is extreme, and we know that $C := \text{cone}(V) = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$. We run our program on V and get a new set A', and would like to know if $C = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\}$. We have the following situation:

$$\begin{aligned}
&C = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\} \Rightarrow A \sqsubseteq A' \\
&C = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\} \Rightarrow A'V \leq \mathbf{0} \\
&A \sqsubseteq A' \qquad\qquad\quad\;\; \Rightarrow \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\} \subseteq C \\
&A'V \leq \mathbf{0} \qquad\qquad\;\;\; \Rightarrow C \subseteq \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\}
\end{aligned}$$

So we have the following.

Proposition 11. *Let $A \in \mathbb{R}^{m \times d}$ be extreme, and $C = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\} = \text{cone}(V)$. Then*

$$C = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\} \;\Leftrightarrow\; A'V \leq \mathbf{0},\ A \sqsubseteq A'$$

4.3 Testing H-Polyhedron → V-Polyhedron

Say we have an H-Polyhedron $P = \{\mathbf{x} \mid A\mathbf{x} \leq b\}$, and wish to check that our program correctly calculates a V' and U' such that $P = \text{cone}(U') + \text{conv}(V')$. Again, we shall use the notion of extremity and show that under certain circumstances we can use extreme sets to demonstrate the validity of our algorithm. In this case, the definition of extreme is a little more complicated, but it asserts that a set U is extreme as before, and that no element of another set V can be expressed as a non-trival sum of a convex combination of V and a non-negative linear combination of members of U.

Definition 12 (Extreme Pair)**.** A pair of sets $U \in \mathbb{R}^{d\times n}, V \in \mathbb{R}^{d\times p}$ is called an *extreme pair* if for any $\mathbf{u} = U\mathbf{e}_k, \mathbf{v} = V\mathbf{e}_l$ the following is true:

$$\mathbf{t} \geq \mathbf{0},\ \mathbf{u} = U\mathbf{t} \Rightarrow \mathbf{t} = \mathbf{e}_k$$
$$\mathbf{t} \geq \mathbf{0}, \boldsymbol{\lambda} \geq \mathbf{0}, \langle \boldsymbol{\lambda}, \mathbf{1}\rangle = 1, \mathbf{v} = U\mathbf{t} + V\boldsymbol{\lambda} \Rightarrow \mathbf{t} = \mathbf{0}, \boldsymbol{\lambda} = \mathbf{e}_l$$

Let us now consider the set U in the expression $\{\mathbf{x} \mid A\mathbf{x} \leq b\} = \text{cone}(U) + \text{conv}(V)$.

Proposition 12. *Suppose that* $P := \{\mathbf{x} \mid A\mathbf{x} \leq b\} = \text{cone}(U) + \text{conv}(V)$*. Then*

$$\text{cone}(U) = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$$

Proof. We show that the following three statements are equivalent:

1. $A\mathbf{r} \leq \mathbf{0}$
2. $(\forall \mathbf{x} \in P)(\forall \alpha > 0)\ \mathbf{x} + \alpha\mathbf{r} \in P$
3. $\mathbf{r} \in \text{cone}(U)$

$(1 \Rightarrow 2)$. $\mathbf{x} \in P$ means that $A\mathbf{x} \leq \mathbf{b}$, and $A\mathbf{r} \leq \mathbf{0}$ means that $A(\mathbf{x}+\alpha\mathbf{r}) \leq A\mathbf{x} \leq \mathbf{b}$.
$(\neg 1 \Rightarrow \neg 2)$. Suppose $\langle A_i, \mathbf{r}\rangle > 0$, then let $\alpha > (b_i - \langle A_i, \mathbf{x}\rangle)/\langle A_i, \mathbf{r}\rangle$. We have:

$$\langle A_i, \mathbf{x} + \alpha\mathbf{r}\rangle > \langle A_i, \mathbf{x}\rangle + \frac{b_i \langle A_i, \mathbf{r}\rangle - \langle A_i, \mathbf{x}\rangle \langle A_i, \mathbf{r}\rangle}{\langle A_i, \mathbf{r}\rangle} = b_i$$

$(3 \Rightarrow 2)$. This is essentially the definition of $\text{cone}(U) + \text{conv}(V)$.
$(2 \Rightarrow 3)$. Now for the real work. Suppose that (2) holds, but $\mathbf{r} \notin \text{cone}(U)$. Then by the Farkas Lemma, we have a $\mathbf{y}$ that satisfies $(\forall \mathbf{r} \in U)\ \langle \mathbf{r}, \mathbf{y}\rangle \leq 0,\ \langle \mathbf{y}, \mathbf{r}\rangle > 0$. From (2) we construct a sequence: $(\mathbf{x}_n) = \mathbf{v} + n \cdot \mathbf{r}$. Then it is clear that the sequence $\langle \mathbf{y}, \mathbf{x}_n\rangle \to \infty$. It is also clear that $(\forall n)\, \mathbf{x}_n \in P$. We now need the following:

- A linear, real-valued function on the set $\text{conv}(V)$ achieves its maximal value at some $\bar{\mathbf{v}} \in V$.

To see this is true, suppose that the linear function is given by $\langle \mathbf{y}, \cdot\rangle$, and that $\bar{\mathbf{v}}$ is an element of V such that $(\forall \mathbf{v} \in V)\ \langle \mathbf{y}, \bar{\mathbf{v}}\rangle \geq \langle \mathbf{y}, \mathbf{v}\rangle$. Then, for any $\mathbf{r} \in \text{conv}(V)$, $\mathbf{r} = \sum_{\mathbf{v}\in V} \lambda_v \mathbf{v}$ where $\sum \lambda_v = 1 \Rightarrow \lambda_v \leq 1$, and it follows

$$\langle \mathbf{y}, \mathbf{r}\rangle = \left\langle \mathbf{y}, \sum_{\mathbf{v}\in V} \lambda_v v \right\rangle = \sum_{v\in V} \lambda_v \langle \mathbf{y}, \mathbf{v}\rangle \leq \sum_{v\in V} \lambda_v \langle \mathbf{y}, \bar{\mathbf{v}}\rangle = \langle \mathbf{y}, \bar{\mathbf{v}}\rangle$$

Now consider the maximum value of the function $\langle \mathbf{y}, \cdot\rangle$ on P. Since any element of P can be written $\mathbf{r} + \mathbf{v} \mid \mathbf{r} \in \text{cone}(U),\ \mathbf{v} \in \text{conv}(V)$, and $(\forall \mathbf{r} \in U)\, \langle \mathbf{y}, \mathbf{r}\rangle \leq 0$, we can find the maximum value on $\text{conv}(V)$. However, $\langle \mathbf{y}, \cdot\rangle$ achievs its maximal value on $\text{conv}(V)$ at some $\bar{\mathbf{v}} \in V$, which is a contradiction with the fact that $\langle \mathbf{y}, \mathbf{x}_n\rangle \to \infty$, so we conclude that $\mathbf{r} \in \text{cone}(U)$. □

Remark. Note that (2) in the proof above is independent of A and U. This means that the cone of a polyhedron is independent of its representation, i.e. if $\text{cone}(U) + \text{conv}(V) = \text{cone}(U') + \text{conv}(V')$, then $\text{cone}(U) = \text{cone}(U')$, while it is not necessarily true that $\text{conv}(V) = \text{conv}(V')$. Similarly, if $\{\mathbf{x} \mid A\mathbf{x} \le \mathbf{b}\} = \{\mathbf{x} \mid A'\mathbf{x} \le \mathbf{b}'\}$, the it holds that $\{\mathbf{x} \mid A\mathbf{x} \le \mathbf{0}\} = \{\mathbf{x} \mid A'\mathbf{x} \le \mathbf{0}\}$.

Proposition 13. *A convex combination of convex combinations is another convex combination*

Proof. Let Λ represent a collection of convex combinations, that is, $\mathbf{1}^T\Lambda = \mathbf{1}^T$, and let $\boldsymbol{\lambda} \ge \mathbf{0}$, $\mathbf{1}^T\boldsymbol{\lambda} = 1$ be a convex combinator. Then $\Lambda\boldsymbol{\lambda} = \boldsymbol{\lambda}'$ where $\boldsymbol{\lambda}' \ge \mathbf{0}$, $\mathbf{1}^T\boldsymbol{\lambda}' = 1$. That $\boldsymbol{\lambda}' \ge \mathbf{0}$ is clear, then just note that $\mathbf{1}^T\boldsymbol{\lambda}' = \mathbf{1}^T\Lambda\boldsymbol{\lambda} = \mathbf{1}^T\boldsymbol{\lambda} = 1$. □

Proposition 14. *The following two statements hold*

1. $A \subseteq B, C \subseteq D \Rightarrow A + C \subseteq B + D$
2. $P + \text{cone}(U) + \text{cone}(U) = P + \text{cone}(U)$

Proof. (1) $a \in A \Rightarrow a \in B$, $c \in C \Rightarrow c \in D$. Taken together, $a + c \in B + D$.
(2) $\mathbf{t}, \mathbf{t}' \ge \mathbf{0} \Rightarrow p + U\mathbf{t} + U\mathbf{t}' = p + U(\mathbf{t} + \mathbf{t}') = p + U\mathbf{t}'', \mathbf{t}'' \ge \mathbf{0}$. □

Proposition 15. *Suppose that there is an extreme pair U, V such that* $\text{cone}(U) + \text{conv}(V) = \{\mathbf{x} \mid A\mathbf{x} \le b\}$*. Denote this Polyhedron P. Then the following are equivalent:*

1. $P = \text{cone}(U') + \text{conv}(V')$
2. $U \sqsubseteq U'$, $V \subseteq V'$, $AU' \le \mathbf{0}$, $AV' \le \mathbf{b}$

Proof. ($2 \Rightarrow 1$). There's not too much to say about this direction, it's mostly just collecting some straightforward observations and results.

(a) $U \sqsubseteq U' \Rightarrow \text{cone}(U) \subseteq \text{cone}(U')$

(b) $V \subseteq V' \Rightarrow \text{conv}(V) \subseteq \text{conv}(V')$

(c) (a) + (b) $\Rightarrow P \subseteq \text{cone}(U') + \text{conv}(V')$

(d) $AU' \le \mathbf{0} \Rightarrow \text{cone}(U') \subseteq \text{cone}(U)$

(e) $AV' \le \mathbf{b} \Rightarrow \text{conv}(V') \subseteq P$

(f) (d) + (e) $\Rightarrow \text{cone}(U') + \text{conv}(V') \subseteq P + \text{cone}(U) = P$

- (c) + (f) $\Rightarrow$ ($2 \Rightarrow 1$)

(a) and (b) are clear, (c) uses proposition 14, (d) requires proposition 12, (e) is clear, and (f) uses proposition 14.
($1 \Rightarrow 2$). This direction is a little more interesting. First we observe:

$$\text{cone}(U) = \{\mathbf{x} \mid A\mathbf{x} \le \mathbf{0}\} = \text{cone}(U') \Rightarrow U \sqsubseteq U'$$

The equalities follow proposition 12, and the implication from proposition 7. Note that the extremeness of U and the Farkas lemma are both used here. Since we know that $\text{cone}(U) = \text{cone}(U')$, we also know that $\text{cone}(U') + \text{conv}(V') = \text{cone}(U) + \text{conv}(V')$. Next, we consider V and exploit its extremeness. Since $P = \text{cone}(U) + \text{conv}(V')$, each $\mathbf{v}' \in V'$ can be written $U\mathbf{t} + V\boldsymbol{\lambda}$, where $\mathbf{t} \geq \mathbf{0}$ and $\boldsymbol{\lambda}$ is a convex combinator. We combine these into matrices T and Λ, so $V' = UT + V\Lambda$. But it is also true that every $\mathbf{v} \in V$ can be written as

$$\mathbf{v} = U\mathbf{t} + V'\boldsymbol{\lambda} = U\mathbf{t} + (UT + V\Lambda)\boldsymbol{\lambda} = U\mathbf{t}' + V\boldsymbol{\lambda}'$$

Where $\mathbf{t}' \geq \mathbf{0}$, and $\boldsymbol{\lambda}'$ is a convex combinator. Because U, V is an extreme pair, we have that $\mathbf{t}' = \mathbf{0}$, and $\boldsymbol{\lambda} = \mathbf{e}_k$ for some k. Because U is extreme, it does not contain $\mathbf{0}$, and so $\mathbf{t} = \mathbf{0}$. This puts us at $\mathbf{v} = V'\boldsymbol{\lambda} = V\Lambda\boldsymbol{\lambda} = V\boldsymbol{\lambda}'$, and $\boldsymbol{\lambda}' = \mathbf{e}_k$. In order that $\boldsymbol{\lambda}' = \mathbf{e}_k$, for every column of Λ corresponding to a positive entry in $\boldsymbol{\lambda}$, only one row may contain a positive entry, and that entry must be 1. Then instead of $\boldsymbol{\lambda}$, use instead $\mathbf{e}_l$ where $\Lambda_k^l = 1$. Then $\Lambda\boldsymbol{\lambda} = \Lambda\mathbf{e}_l$, so $V\Lambda\boldsymbol{\lambda} = V\Lambda\mathbf{e}_l = V'\mathbf{e}_l = \mathbf{v}'$ where $\mathbf{v}' \in V'$. Then $\mathbf{v} \in V'$.

That $AV' \leq \mathbf{b}$ is obvious, and that $AU' \leq \mathbf{0}$ is mentioned in the remarks after proposition 12. □

The Test. To test our program, we *simply* create Polyhedron $P = \{\mathbf{x} \mid A\mathbf{x} \leq b\}$ $= \text{cone}(U) + \text{conv}(V)$ where (U, V) is an extreme pair, run our program on (A, b) to get some new (V', U'). Then, we check that $A\mathbf{v}' \leq \mathbf{b}$, $A\mathbf{u}' \leq \mathbf{0}$, $U \sqsubseteq U'$, and $V \subseteq V'$.

4.4 Testing V-Polyhedron → H-Polyhedron

Now we suppose we have a V-Polyhedron $P = \text{cone}(U) + \text{conv}(V)$, and would like to test the program which returns a matrix-vector pair $A', \mathbf{b}'$ where supposedly $P = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{b}'\}$. Again, we will start off with a pair $A, \mathbf{b}$ where we know that $P = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$, where $A, \mathbf{b}$ satisfy some nice properties, and use those properties to test if $P = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{b}'\}$. In order to demonstrate these properties, the Farkas Lemma will be used, but in different forms. We want to use proposition 9, but first we have to check:

Proposition 16. *The following statements are equivalent:*

1. $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\} = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{b}'\}$
2. $\left\{\begin{pmatrix} x_0 \\ \mathbf{x} \end{pmatrix} \middle| \begin{pmatrix} -1 & \mathbf{0} \\ -\mathbf{b} & A \end{pmatrix} \begin{pmatrix} x_0 \\ \mathbf{x} \end{pmatrix} \leq \mathbf{0}\right\} = \left\{\begin{pmatrix} x_0 \\ \mathbf{x} \end{pmatrix} \middle| \begin{pmatrix} -1 & \mathbf{0} \\ -\mathbf{b}' & A' \end{pmatrix} \begin{pmatrix} x_0 \\ \mathbf{x} \end{pmatrix} \leq \mathbf{0}\right\}$

Proof. $(2 \Rightarrow 1)$. Just set $x_0 = 1$, and move $\mathbf{b}, \mathbf{b}'$ to the right side of the inequalities.

$(\neg 2 \Rightarrow \neg 1)$. Suppose that:

$$\begin{pmatrix} -1 & \mathbf{0} \\ -\mathbf{b} & A \end{pmatrix} \begin{pmatrix} x_0 \\ \mathbf{x} \end{pmatrix} \leq \mathbf{0}, \quad \begin{pmatrix} -1 & \mathbf{0} \\ -\mathbf{b}' & A' \end{pmatrix} \begin{pmatrix} x_0 \\ \mathbf{x} \end{pmatrix} \not\leq \mathbf{0}$$

Observe that, by the way these sets are constructed, $x_0 \geq 0$. If $x_0 = 0$, then we have $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\} \neq \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\}$, which, by the remark following proposition 12 means that $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\} \neq \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{b}'\}$. If $x_0 > 0$, then we have:

$$A\mathbf{x} \leq x_0\mathbf{b},\ A'\mathbf{x} \not\leq x_0\mathbf{b}' \Rightarrow A(\mathbf{x}/x_0) \leq \mathbf{b},\ A'(\mathbf{x}/x_0) \not\leq \mathbf{b}'$$

So $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\} \neq \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{b}'\}$. □

Now, combining the results of propositions 9 and 16, we have the following result:

Proposition 17. *The following two statement are equivalent:*

1. $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\} = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{b}'\}$
2. $\operatorname{cone}\begin{pmatrix} -\mathbf{b}^T & -1 \\ A^T & \mathbf{0} \end{pmatrix} = \operatorname{cone}\begin{pmatrix} -\mathbf{b}'^T & -1 \\ A'^T & \mathbf{0} \end{pmatrix}$

Proof. This immediately follows from propositions 16 and 9. □

4.4.1 Extreme H-Polyhedra Pairs

Definition 13 (Extreme Pair)**.** A pair $A, \mathbf{b}$ is called **extreme** if $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$ is non-empty, and

$$\mathbf{t} \geq \mathbf{0},\ \mathbf{t}^T(A, b) = (A_i, b_i) \Rightarrow [\mathbf{t} \neq \mathbf{0} \Rightarrow \mathbf{t} = \mathbf{e}_i]$$

Of course, we want to say that if $P = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$ where $(A, \mathbf{b})$ is an extreme pair, and $P = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{b}'\}$, then, by proposition 17, $(A, \mathbf{b}) \sqsubseteq (A', \mathbf{b}')$. The catch is that the cones in proposition 17 have a strange form. What we want to be true turns out to be so, but before we can prove this fact, we need a property of extreme pairs, which requires a new form of the Farkas Lemma.

4.4.2 Farkas Lemma: Round 2

Let us restate the conclusion of the Farkas Lemma:

$$\exists \mathbf{t} \geq \mathbf{0} \mid U\mathbf{t} = \mathbf{x} \ \Leftrightarrow\ \neg\exists \mathbf{y} \mid \mathbf{y}^T U \leq \mathbf{0},\, \mathbf{y}^T\mathbf{x} > 0$$

If we let $U = (A, -A, I)$, and we get a new form:

$$\exists \mathbf{t} \geq \mathbf{0} \mid (A, -A, I)\mathbf{t} = \mathbf{x} \ \Leftrightarrow\ \neg\exists \mathbf{y} \mid \mathbf{y}^T(A, -A, I) \leq \mathbf{0},\, \mathbf{y}^T\mathbf{x} > 0$$

Then breaking apart $\mathbf{t} = (\mathbf{t}_P, \mathbf{t}_N, \mathbf{t}_I)$, we have

$$(A, -A, I)\begin{pmatrix} \mathbf{t}_P \\ \mathbf{t}_N \\ \mathbf{t}_I) \end{pmatrix} = \mathbf{x} \Rightarrow A(\mathbf{t}_P - \mathbf{t}_N) = \mathbf{x} - \mathbf{t}_I$$

If we let $\mathbf{t}_P - \mathbf{t}_N = \mathbf{z}$, then $\mathbf{z}$ is no longer constrained by $\mathbf{t} \geq \mathbf{0}$, and we have that $A\mathbf{z} \leq \mathbf{x}$. Since $\mathbf{y}^T A \leq \mathbf{0}$ and $-\mathbf{y}^T A \leq \mathbf{0}$, it must be that $\mathbf{y}^T A = \mathbf{0}$. Combining these results, relabeling $\mathbf{x}$ as $\mathbf{b}$ and $\mathbf{z}$ as $\mathbf{x}$, and $\mathbf{y}$ as $-\mathbf{y}$, we see a new form of Farkas Lemma:

$$\exists \mathbf{x} \mid A\mathbf{x} \le \mathbf{b} \;\Leftrightarrow\; \neg\exists \mathbf{y} \ge \mathbf{0} \mid \mathbf{y}^T A = \mathbf{0},\, \mathbf{y}^T\mathbf{b} < 0 \tag{4.5}$$

This form tells us that an H-Polyhedron is non-empty, or we can create an inequality that is impossible to solve from it's matrix. The next form shows a similar result, but this time it is for a specific $\mathbf{x}$, not implying the emptyness of the Polyhedra, just the constraints' failure to be satisfied at a specific point.

First observe that (using (4.5))

$$A\mathbf{x} \not\le \mathbf{b} \Leftrightarrow \neg\exists \mathbf{z} \mid \begin{pmatrix} A \\ I \\ -I \end{pmatrix} \mathbf{z} \le \begin{pmatrix} \mathbf{b} \\ \mathbf{x} \\ -\mathbf{x} \end{pmatrix} \Leftrightarrow \exists \mathbf{y} \ge \mathbf{0} \mid \mathbf{y}^T \begin{pmatrix} A \\ I \\ -I \end{pmatrix} = \mathbf{0},\, \mathbf{y}^T \begin{pmatrix} \mathbf{b} \\ \mathbf{x} \\ -\mathbf{x} \end{pmatrix} < 0$$

Splitting up $\mathbf{y}$ into components like we did before with $\mathbf{t}$, we can rewrite this as

$$A\mathbf{x} \not\le \mathbf{b} \Leftrightarrow \exists \mathbf{y} \mid \mathbf{y}^T A = \mathbf{w},\, \mathbf{y}^T\mathbf{b} < \mathbf{w}^T\mathbf{b} \tag{4.6}$$

We can now prove some useful properties of extreme pairs.

Proposition 18. *Let $A, \mathbf{b}$ be an extreme pair. Then the following holds:*

$$\mathbf{t} \ge \mathbf{0},\, \mathbf{t}^T A = \mathbf{0} \Rightarrow [\mathbf{t} \ne \mathbf{0} \Rightarrow \langle \mathbf{t}, \mathbf{b}\rangle > 0] \tag{4.7}$$

$$\mathbf{t} \ge \mathbf{0},\, \mathbf{t}^T A = A_i \Rightarrow [\mathbf{t} \ne \mathbf{e}_i \Rightarrow \langle \mathbf{t}, \mathbf{b}\rangle > b_i] \tag{4.8}$$

Proof. Suppose that $\mathbf{t}^T A = \mathbf{0}$, $\mathbf{t} \ne \mathbf{0}$. Next suppose that $\mathbf{t}^T\mathbf{b} = 0$. Then $(\mathbf{t} + \mathbf{e}_i)(A, b) = (A_i, b_i)$, but $\mathbf{t} + \mathbf{e}_i \ne \mathbf{e}_i$, a contradiction. Next suppose that $\mathbf{t}^T\mathbf{b} < 0$. Then we have that $\exists \mathbf{t} \ge \mathbf{0}$, $\mathbf{t}^T A = \mathbf{0}$, $\mathbf{t}^T\mathbf{b} < 0$, which by (4.5) means that $\{\mathbf{x} \mid A\mathbf{x} \le b\}$ is empty, a contradiction. So the first property is proven.

Now suppose that $\mathbf{t}^T A = A_i, \mathbf{t} \ne \mathbf{e}_i$. Then we know that $\mathbf{t}^T\mathbf{b} \ne b_i$, but if $\mathbf{t}^T\mathbf{b} < 0$, we'd have that $(\mathbf{t} - \mathbf{e}_i)^T A = \mathbf{0}, (\mathbf{t} - \mathbf{e}_i)^T\mathbf{b} < 0$, contradicting the first property. $\square$

We are now prepared to prove the following proposition.

Proposition 19. *Suppose that $\{\mathbf{x} \mid A\mathbf{x} \le \mathbf{b}\} = \{\mathbf{x} \mid A'\mathbf{x} \le \mathbf{b}'\}$, where $(A, \mathbf{b})$ is an extreme pair. Then $(A, \mathbf{b}) \sqsubseteq (A', \mathbf{b}')$.*

Proof. It suffices to show that $\begin{pmatrix} -\mathbf{b}^T & -1 \\ A^T & \mathbf{0} \end{pmatrix}$ is extreme, for then:

$$\begin{pmatrix} -\mathbf{b}^T & -1 \\ A^T & \mathbf{0} \end{pmatrix} \sqsubseteq \begin{pmatrix} -\mathbf{b}'^T & -1 \\ A'^T & \mathbf{0} \end{pmatrix} \Rightarrow \begin{pmatrix} -\mathbf{b}^T \\ A^T \end{pmatrix} \sqsubseteq \begin{pmatrix} -\mathbf{b}'^T \\ A'^T \end{pmatrix} \Rightarrow (A, \mathbf{b}) \sqsubseteq (A', \mathbf{b}')$$

So, suppose that $(\mathbf{t}, t) \ge \mathbf{0}$, $\begin{pmatrix} -\mathbf{b}^T & -1 \\ A^T & \mathbf{0} \end{pmatrix} (\mathbf{t}, t) = \begin{pmatrix} -b_i \\ A_i^T \end{pmatrix}$. We must show that $\mathbf{t} = \mathbf{e}_i$, $t = 0$. Suppose that it's not. Then we have $\mathbf{t}^T A = A_i$, and by (4.8) $\mathbf{t}^T - \mathbf{b} < b_i$. Since $-t \le 0$, $\mathbf{t}^T - \mathbf{b} - t < b_i$, a contradiction. So we have shown that $\mathbf{t} = \mathbf{e}_i$, in which case $t = 0$, and the proposition follows. $\square$

The Test Suppose that we have a (U, V) and $(A, \mathbf{b})$ such that $P := \text{cone}(U) + \text{conv}(V) = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$, and $(A, \mathbf{b})$ is an extreme pair. We run our program and get a new pair $(A', \mathbf{b}')$. Denote $P' := \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{b}'\}$. We would like to verify that $P = P'$. We have the following:

1. $A'U \leq \mathbf{0}$, $A'V \leq \mathbf{b}' \Rightarrow P \subseteq P'$
2. $(A, \mathbf{b}) \sqsubseteq (A', \mathbf{b}') \Rightarrow P' \subseteq P$
3. $P = P' \Rightarrow A'U \leq \mathbf{0}$, $A'V \leq \mathbf{b}'$
4. $P = P' \Rightarrow (A, \mathbf{b}) \sqsubseteq (A', \mathbf{b}')$

So we conclude that

$$P = P' \Leftrightarrow A'U \leq \mathbf{0},\ A'V \leq \mathbf{b}',\ A \sqsubseteq A'$$

To test our program, we just verify the right side of the equivalence, which is fairly straightforward.

4.5 test_functions.h

The following types are defined for running tests of the different algorithms. They are expected to be given a descriptive name, the object on which the test will be run, and a `key` with which the result of the test will be compared. The `key` object is one of the extreme objects described above.

```
struct hcone_test_case {
  std::string name;
  Matrix hcone; // vectors for H or V cone
  Matrix key;   // minimal generating set

  bool run_test() const;
};
```

```
struct vcone_test_case {
  std::string name;
  Matrix vcone; // vectors for H or V cone
  Matrix key;   // minimal generating set

  bool run_test() const;
};
```

```
struct hpoly_test_case {
  std::string name;
  Matrix hpoly; // vectors for H-Polyhedron
  VPoly key;    // minimal generating set

  bool run_test() const;
};
```

```
struct vpoly_test_case {
  std::string name;
  VPoly  vpoly; // vectors for V-Polyhedron
  Matrix key;   // minimal generating set

  bool run_test() const;
};
```

4.6 test_functions.cpp

The dot-product and norm (in terms of dot product).

```
double operator*(const Vector &l, const Vector &r) {
  if (l.size() > r.size()) {
    throw runtime_error{"inner product: l > r"};
  }
  return inner_product(begin(l), end(l), begin(r), 0.);
}
```

```
double norm(const Vector &v) {
  return sqrt(v*v);
}
```

`approximately_zero` is used during tests to avoid issues involving floating point rounding errors. For example, `1/6.0 * 2.5 - 5/12.0 == 0` will give `false`, while `approximately_zero(1/6.0 * 2.5 - 5/12.0)` will return `true`. Test cases are used where intermediate calculations don't depend on such high accuracy, and these discrepencies can be ignored.

`approximately_zero(c) == true` is be denoted `c` ≈ 0.

```
bool approximately_zero(double d) {
  const double error = .000001;
  bool result = abs(d) < error;
  if (d != 0 && result) {
    ostringstream oss;
    oss << scientific << d;
    log("approximately_zero " + oss.str(), 1);
  }
  return result;
}
```

Tests $c < 0 \vee c \approx 0$.

```
bool approximately_lt_zero(double d) {
  return d < 0 || approximately_zero(d);
}
```

Tests $||\mathbf{v}|| \approx 0$. This is be denoted $\mathbf{v} \approx \mathbf{0}$.

```
bool approximately_zero(const Vector &v) {
  return approximately_zero(norm(v));
}
```

Tests $\mathbf{u}/\,||\mathbf{u}|| - \mathbf{v}/\,||\mathbf{v}|| \approx \mathbf{0}$. This is be denoted $\mathbf{u} \simeq \mathbf{v}$.

```
bool is_equivalent(const Vector &l, const Vector &r) {
  if (l.size() != r.size()) return false;
  if (norm(l) == 0 || norm(r) == 0) {
    return norm(l) == 0 && norm(r) == 0;
  }
  return approximately_zero(l / norm(l) - r / norm(r));
}
```

Tests $\mathbf{u} - \mathbf{v} \approx \mathbf{0}$. This is be denoted $\mathbf{u} \approx \mathbf{v}$.

```
bool is_equal(const Vector &l, const Vector &r) {
  if (l.size() != r.size()) return false;
  return approximately_zero(l - r);
}
```

Tests $(\exists \mathbf{u} \in U) \mid \mathbf{v} \simeq \mathbf{u}$.

```
bool has_equivalent_member(const Matrix &M,
                           const Vector &v) {
  if (!any_of(M.begin(), M.end(),
    [&](const Vector &u) {
      return is_equivalent(u,v); })) {
    ostringstream oss;
    oss << dashes
        << " no equivalent member found for:\n"
        << v << endl;
    log(oss.str(),1);
    return false;
  }
  return true;
}
```

Tests $(\exists \mathbf{u} \in U) \mid \mathbf{v} \approx \mathbf{u}$.

```
bool has_equal_member(const Matrix &M,
                      const Vector &v) {
  if (!any_of(M.begin(), M.end(),
    [&](const Vector &u) { return
      is_equal(u,v); })) {
    ostringstream oss;
    oss << dashes
        << " no equal member found for:\n"
        << v << endl;
    log(oss.str(),1);
    return false;
  }
  return true;
}
```

Tests $(\forall v \in V)(\exists \mathbf{u} \in U) \mid \mathbf{v} \simeq \mathbf{u}$. This is be denoted $V \sqsubseteq U$.

```
bool subset_mod_eq(const Matrix &generators,
                   const Matrix &vcone) {
  return all_of(generators.begin(), generators.end(),
    [&](const Vector &g) {
      return has_equivalent_member(vcone, g); });
```

```
}
```

Tests $(\forall v \in V)(\exists \mathbf{u} \in U) \mid \mathbf{v} \approx \mathbf{u}$. This is be denoted $V \subseteq U$.

```
bool subset(const Matrix &generators,
            const Matrix &vcone) {
  return all_of(generators.begin(), generators.end(),
    [&](const Vector &g) {
      return has_equal_member(vcone, g); });
}
```

Given a `Vector constraint` and `Vector ray`, tests if `approximately_lt_zero(ray * constraint)`. Note that if the constraint is of the form $\langle A_i, \mathbf{v} \rangle \leq b$ for some value b, then this tests $\langle A_i, \texttt{ray} \rangle \leq \mathbf{0}$.

```
bool ray_satisfied(const Vector &constraint,
                   const Vector &ray) {
  if (constraint.size() != ray.size() &&
      constraint.size()-1 != ray.size()) {
    throw runtime_error{"bad ray vs constraint"};
  }
  double ip = ray * constraint;
  if (!(approximately_lt_zero(ip))) {
    ostringstream oss;
    oss << dashes << " ray not satisfied!\n"
        << "ray: " << ray
        << "\nconstraint: " << constraint
        << "\n ray * constraint = " << ip << endl;
    log(oss.str(), 1);
    return false;
  }
  return true;
}
```

Test $A\mathbf{v} \leq \mathbf{0}$

```
bool ray_satisfied(const Matrix &constraints,
                   const Vector &ray) {
  return all_of(constraints.begin(), constraints.end(),
    [&](const Vector &cv) {
      return ray_satisfied(cv, ray); });
}
```

Test $AV \leq \mathbf{0}$

```
bool rays_satisfied(const Matrix &constraints,
                    const Matrix &rays) {
  return all_of(rays.begin(), rays.end(),
    [&](const Vector &ray) {
      return ray_satisfied(constraints, ray); });
}
```

Test $\langle A_i, \mathbf{v} \rangle \leq b_i$

```
bool vec_satisfied(const Vector &constraint,
                   const Vector &vec) {
  size_t cback_i = constraint.size()-1;
```

```
  if (cback_i != vec.size()) {
    throw runtime_error{"bad vec vs constraint"};
  }
  double ip = vec * constraint;
  double c_val = constraint[cback_i];
  if (!(approximately_lt_zero(ip - c_val))) {
    ostringstream oss;
    oss << dashes << " vec not satisfied!\n"
        << "vec: " << vec
        << "\nconstraint: " << constraint
        << "\n vec * constraint = " << ip << endl;
    log(oss.str(), 1);
    return false;
  }
  return true;
}
```

Test $A\mathbf{v} \leq \mathbf{b}$

```
bool vec_satisfied(const Matrix &constraints,
                   const Vector &vec) {
  return all_of(constraints.begin(), constraints.end(),
    [&](const Vector &cv) {
      return vec_satisfied(cv, vec); });
}
```

Test $AV \leq \mathbf{b}$

```
bool vecs_satisfied(const Matrix &constraints,
                    const Matrix &vecs) {
  return all_of(vecs.begin(), vecs.end(),
    [&](const Vector &vec) {
      return vec_satisfied(constraints, vec); });
}
```

Given an H-Cone $C = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\} = \mathrm{cone}(U)$ where U is extreme, and a `Matrix` U', determines if $C = \mathrm{cone}(U')$.

Similarly, given a V-Cone $C = \mathrm{cone}(U) = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$ where A is extreme, and a `Matrix` A', determines if $C = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\}$.

```
bool equivalent_cone_rep(const Matrix &cone,
                         const Matrix &key,
                         const Matrix &alt_rep) {
  return rays_satisfied (cone, alt_rep) &&
         subset_mod_eq  (key, alt_rep);
}
```

Given an H-Polytope $P = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\} = \mathrm{cone}(U) + \mathrm{conv}(V)$ where U and V are extreme, and a pair (U', V'), determines if $P = \mathrm{cone}(U') + \mathrm{conv}(V')$.

```
bool equivalent_hpoly_rep(const Matrix &hpoly,
                          const VPoly  &key,
                          const VPoly  &vpoly) {
  return rays_satisfied (hpoly, vpoly.U) &&
         vecs_satisfied (hpoly, vpoly.V) &&
         subset_mod_eq  (key.U, vpoly.U) &&
```

```
            subset           (key.V, vpoly.V);
}
```

Given a V-Polytope $P = \text{cone}(U) + \text{conv}(V) = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$ where A is extreme, and a `Matrix` $(A', \mathbf{b}')$, determines if $P = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{b}'\}$.

```
bool equivalent_vpoly_rep(const VPoly  &vpoly,
                          const Matrix &key,
                          const Matrix &hpoly) {
  return rays_satisfied (hpoly, vpoly.U) &&
         vecs_satisfied (hpoly, vpoly.V) &&
         subset_mod_eq  (key, hpoly);
}
```

Bibliography

[1] Ziegler, Gunter. *Lectures on Polytopes*. Springer-Verlag, New York, 1995. ISBN 0-387-94329-3.