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 important 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 Theorem is 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 correctness of the algorithm also proves a result known as The Farkas Lemma.

This thesis is broken up into four chapters. Chapter 1 states the definitions necessary for Minkowski-Weyl Theorem, and states the theorem. Chapter 2 proves the theorem, by first considering the case that the polyhedron is a cone, then shows how to reduce the case of general polyhedra to that of cones. Chapter 3 shows a C++ implementation of the transformations described in Chapter 2. Chapter 4 presents a method of testing the program for special cases of polyhedra, known as either pointed or full-dimensional polyhedra. The Farkas Lemma is proven and extensively used to show the validity of the testing methods.

1. Minkowski-Weyl Theorem

We begin somewhat tersely, stating some basic definitions in order to state the theorem. The only noteworthy part of this section is Proposition 1.1.1, which will be used a number of times throughout the paper.

1.1 Polyhedra

Definition 1.1.1 (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 1.1.2 (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 1.1.3 (Convex Combination)**.** Let $V \in \mathbb{R}^{d \times n}$, and let $\boldsymbol{\lambda} \in \mathbb{R}^n$ satisfy $\sum_{1 \leq j \leq n} \lambda_j = 1$, $\boldsymbol{\lambda} \geq \mathbf{0}$, 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 1.1.4 (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 1.1.5 (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 1.1.6 (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 1.1.1 (Closure Property of Cones)**.** *Let C be either an H-Cone or a V-Cone, and for each i let $\mathbf{x}^i \in C$, and $c_i \geq 0$. Then:*

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

Proof. First we prove Proposition 1.1.1 for H-Cones, then for V-Cones. If, for each i, $A\mathbf{x}^i \leq \mathbf{0}$, then $A(c_i\mathbf{x}^i) = c_iA\mathbf{x}^i \leq \mathbf{0}$, and

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

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

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

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

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 $\mathbf{x}^i \in C$.

Remark 1. Proposition 1.1.1 is a fundamental property of cones. In fact, it could be used as an abstract definition of a cone, removed from geometric interpretation, then cones in euclidean space could be examined as important special classes.

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.*

2. Proof of the Minkowski-Weyl Theorem

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.*

2.1 Every V-Cone is an H-Cone

To be clear, what we shall now show is that, given a set of the form $\mathrm{cone}(U)$, there is an A such that $\mathrm{cone}(U) = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$. The first step in this construction is to rewrite $\mathrm{cone}(U)$ as $\Pi\left(\{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\}\right)$, where Π is a coordinate projection. We then show how to calculate these projections, and that the result is a set of the form $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$.

Definition 2.1.1 (Coordinate Projection)**.** Let I be the identity matrix. Then the matrix I' formed by deleting some rows from I is called a **coordinate-projection**.

Lemma 2.1.1 (Lifting a V-Cone)**.** *Every V-Cone is a coordinate-projection of an H-Cone.*

Lemma 2.1.2 (Projecting an H-Cone)**.** *Every coordinate-projection of an H-Cone is an H-Cone.*

First, we quickly use the two lemmas to conclude Theorem 2. The rest of the section will be the proof of the two lemmas.

Proof of Theorem 2. Given Lemma 2.1.1 and Lemma 2.1.2, the proof follows simply. Given a V-Cone, we use Lemma 2.1.1 to get a description involving coordinate-projection of an H-Cone. Then we can apply Lemma 2.1.2 in order to get an H-Cone. □

Proof of Lemma 2.1.1. 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

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

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:

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

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 matrix tranform:

Transform 1 (V-Cone Lift)**.**

$$\mathrm{T_V}(U) = \begin{pmatrix} \mathbf{0} & -I \\ I & -U \\ -I & U \end{pmatrix}$$

So we can rewrite cone(U):

$$\mathrm{cone}(U) = \left\{ \mathbf{x} \in \mathbb{R}^d \;\middle|\; \mathrm{T_V}(U) \begin{pmatrix} \mathbf{x} \\ \mathbf{t} \end{pmatrix} \leq \mathbf{0} \right\}$$

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:

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

This is a coordinate projection of an H-Cone, and Lemma 2.1.1 is shown. □

To prove Lemma 2.1.2, we use two separate propositions.

Proposition 2.1.3 (Projecting Null Columns)**.** *Let $B \in \mathbb{R}^{m' \times (d+p)}$, with the last p columns all $\mathbf{0}$. Let B' be B with the last p columns deleted, and Π the identity matrix with the last p rows deleted (i.e. $B' = \Pi B$). 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 useful properties.

Proposition 2.1.4 (Fourier Motzkin Elimination for H-Cones)**.** *Let $B \in \mathbb{R}^{m_1 \times (d+p)}$, and $\mathbf{x} = \sum_{i \neq k} x_i \mathbf{e}_i$ for some $1 \leq k \leq (d+p)$. 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* is satisfied for rows indexed by Z by definition That it holds for the other rows, observe:

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

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

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

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, because $B_i^k B_j^k - B_j^k B_i^k = 0$ we have:

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

Because B_i^k and $-B_j^k$ are non-negative:

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

Therefore $\left\langle B_i^k B_j - B_j^k B_i, \mathbf{x} \right\rangle \leq 0$, and the right implication is shown.

Now suppose that $B'\mathbf{x} \leq \mathbf{0}$. The task is to find a t so that $B\mathbf{x} \leq \mathbf{0}$. Because rows indexed by Z have $B_l^k = 0$, $B'(\mathbf{x} + t\mathbf{e}_k) \leq \mathbf{0} \Rightarrow B\mathbf{x} \leq \mathbf{0}$. So the task is to find a t so that the inequality holds for rows indexed by P and N. Observe

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

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,

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

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,

$$(\forall i \in P) \quad B_i/B_i^k \mathbf{x} \leq t \Rightarrow B_i\mathbf{x} - B_i^k t \leq 0$$

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. □

Remark 2 (Fourier Motzkin Matrix)**.** Proposition 2.1.4 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

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

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

$$B' = YB$$

Proof of Lemma 2.1.2. 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 rows deleted). For each $1 \leq k \leq p$ we can use Proposition 2.1.4 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 $d+k$, so the $(d+k)$-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 2.1.3, 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

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

This follows from the following:

$$A'\mathbf{y} \leq \mathbf{0} \Rightarrow A(\Pi(\mathbf{y})) \leq \mathbf{0} \tag{2.5}$$
$$A\mathbf{x} \leq \mathbf{0} \Rightarrow (\exists t_1)\ldots(\exists t_p)A'(\mathbf{x} + t_1\mathbf{e}_{d+1} + \cdots + t_p\mathbf{e}_{d+p}) \leq \mathbf{0} \tag{2.6}$$

The key observation of this verification utilizes property *4* of Proposition 2.1.4:

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

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:

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

Because $A' = B_0$, and A is B_p with the last p columns deleted, (2.5) and (2.6) hold, therefore (2.4) holds, and the proof of Lemma 2.1.2 is complete, and we've shown that a coordinate projection of an H-Cone is again an H-Cone. $\square$

With Lemma 2.1.1 and Lemma 2.1.2 proven, we are now certain that every V-Cone is also an H-Cone.

2.2 Every H-Cone is a V-Cone

Now we suppose that we are given a set of the form $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$, and we must show that there is some U such that $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\} = \text{cone}(U)$. In a manner similar to the previous section, we will first write the set $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$ as an intersection of a set $\text{cone}(U')$ with a number of hyperplanes, then give a process to get rid of those intersections.

Definition 2.2.1 (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.*

This is how coordinate hyperplanes will be used. We consider a V-Cone intersected with some coordinate hyperplanes, and write it in the following way:

$$\left\{ \mathbf{x} \in \mathbb{R}^d \;\middle|\; (\exists \mathbf{t} \geq 0) \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} = U'\mathbf{t} \right\} \tag{2.7}$$

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:

$$\Pi\Big(\operatorname{cone}(U') \cap \{x_{d+1} = 0\} \cap \cdots \cap \{x_{d+m} = 0\} \Big) \tag{2.8}$$

The proof that every H-Cone is a V-Cone rests on the following three propostions:

Lemma 2.2.1 (Lifting an H-Cone)**.** *Every H-Cone is a coordinate-projection of a V-Cone intersected with some coordinate hyperplanes.*

Lemma 2.2.2 (Intersecting a V-Cone)**.** *Every V-Cone intersected with a coordinate-hyperplane is a V-Cone.*

Lemma 2.2.3 (Projecting a V-Cone)**.** *Every coordinate-projection of a V-Cone is an V-Cone.*

Proof that Every H-Cone is a V-Cone. Given Lemma 2.2.1, Lemma 2.2.2, and Lemma 2.2.3, the proof follows simply. Given an H-Cone, we use Lemma 2.2.1 to get a description involving the coordinate-projection of a V-Cone intersected with some coordinate-hyperplanes. We apply Lemma 2.2.2 as many times as necessary to elimintate the intersections, then we can apply Lemma 2.2.3 in order to get a V-Cone. □

Proof of Lemma 2.2.1. 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. We use the following transform.

Transform 2 (H-Cone Lift)**.**

$$\mathrm{T_H}(A) = \begin{pmatrix} \mathbf{0} & I & -I \\ I & A & -A \end{pmatrix}$$

In other words,

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

We then claim:

$$\left\{ \mathbf{x} \in \mathbb{R}^d \mid A\mathbf{x} \leq \mathbf{0} \right\} = \left\{ \mathbf{x} \in \mathbb{R}^d \;\middle|\; (\exists \mathbf{t} \geq 0) \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} = \mathrm{T_H}(A)\mathbf{t} \right\} \tag{2.9}$$

First, considering (2.7) and (2.8), observe that this is a coordinate-projection of a V-Cone intersected with some coordinate-hyperplanes. Next, we note that

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

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

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

Then we have

$$\begin{pmatrix} \mathbf{x} \\ A\mathbf{x} \end{pmatrix} = \sum_{1 \le j \le d} x_j^+ \begin{pmatrix} \mathbf{e}_j \\ A^j \end{pmatrix} + \sum_{1 \le j \le d} x_j^- \begin{pmatrix} -\mathbf{e}_j \\ -A^j \end{pmatrix} \tag{2.10}$$

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

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

This can be written

$$A\mathbf{x} \le \mathbf{0} \Leftrightarrow (\exists \mathbf{w} \ge \mathbf{0}) \left| \begin{pmatrix} \mathbf{x} \\ A\mathbf{x} \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \mathbf{w} \end{pmatrix} = \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} \right. \tag{2.11}$$

(2.10) and (2.11) together show

$$A\mathbf{x} \le \mathbf{0} \Rightarrow (\exists \mathbf{t} \ge 0) \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} = \mathrm{T_H}(A)\mathbf{t}$$

Conversely, suppose

$$(\exists \mathbf{t} \ge 0) \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} = \mathrm{T_H}(A)\mathbf{t}$$

We would like to show that $A\mathbf{x} \le \mathbf{0}$. 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

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

where $\mathbf{w} \ge \mathbf{0}$. By (2.11) we have $A\mathbf{x} \le \mathbf{0}$. So (2.9) holds. □

The proof of Lemma 2.2.2 relies upon the following proposition.

Proposition 2.2.4 (Fourier Motzkin Elimination for V-Cones)**.** *Let* $Y \in \mathbb{R}^{(d+m)\times n_1}$*,* $1 \leq k \leq d+m$*, 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:

$$\begin{aligned} &\left\langle Y'^l, \mathbf{e}^k \right\rangle = 0 \\ &\left\langle Y'^{ij}, \mathbf{e}^k \right\rangle = \left\langle Y_k^i Y^j - Y_k^j Y^i, \mathbf{e}^k \right\rangle = Y_k^i Y_k^j - Y_k^j Y_k^i = 0 \end{aligned} \tag{2.12}$$

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

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

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

Then, by Closure Property of Cones, 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

$$\sum_{i\in P} t_i Y^i + \sum_{j\in N} t_j Y^j = \sum_{\substack{i\in P \\ j\in N}} t_{ij}(Y_k^i Y^j - Y_k^j Y^i) \tag{2.13}$$

Proposition 2.2.5 (Sum Mixture)**.** *Suppose that*

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

Then (2.13) *holds.*

Proof of proposition 2.2.5. First note that if all $t_i = 0, t_j = 0$, then choosing $t_{ij} = 0$ satisfies (2.13), likewise if all $t_{ij} = 0$, then $t_i = 0, t_j = 0$ satisfies (2.13). So suppose that some $t_i \neq 0, t_j \neq 0, t_{ij} \neq 0$.

The right hand side of (2.13) 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:

$$0 = \sum_{i\in P} t_iY_k^i + \sum_{j\in N} t_jY_k^j \Rightarrow \sum_{i\in P} t_iY_k^i = -\sum_{j\in N} t_jY_k^j$$

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

$$\sum_{i\in P} t_iY^i = \frac{-\sum_{j\in N} t_jY_k^j}{\sigma}\sum_{i\in P} t_iY^i = \sum_{\substack{i\in P\\ j\in N}} -\frac{t_it_j}{\sigma}Y_k^jY^i$$

$$\sum_{j\in N} t_jY^j = \frac{\sum_{i\in P} t_iY_k^i}{\sigma}\sum_{j\in N} t_jY^j = \sum_{\substack{i\in P\\ j\in N}} \frac{t_it_j}{\sigma}Y_k^iY^j$$

Combining these results, we have

$$\sum_{i\in P} t_iY^i + \sum_{j\in N} t_jY^j = \sum_{\substack{i\in P\\ j\in N}} \frac{t_it_j}{\sigma}(Y_k^iY^j - Y_k^jY^i)$$

□

Finally, we can conclude that, given $\mathbf{t} \geq \mathbf{0}$, if $Y\mathbf{t}$ has a 0 in the k-th 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 Lemma 2.2.2. In Proposition 2.2.4, the assumption that $x_k = 0$ in property *4* creates the set $\operatorname{cone}(Y) \cap \{\mathbf{x} \mid x_k = 0\}$. This set, by property *4*, is $\operatorname{cone}(Y')$. □

Proof of Lemma 2.2.3. 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,

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

□

The final step, as in the previous section, is to show that the iterative process of intersecting a V-Cone with coordinate hyperplanes iteratively yields V-Cones. This is merely applying Proposition 2.2.4 multiple times, so the details are omitted. Having shown that H-Cones are V-Cones, the proof of the Minkowski-Weyl Theorem for cones is complete.

2.3 Reducing Polyhedra to Cones

Definition 2.3.1 (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

Here we generalize the results from the previous sections. There is an asymmetry in the reductions, in that the reduction from H-Polyhedra to H-Cones is almost too simple. On the other hand, the reduction from V-Polyhedra to V-Cones is almost as difficult as proving that projections of V-Cones results in V-Cones.

Proposition 2.3.1. *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 begin by re-writing the expression:

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

Note that

$$\left\{\mathbf{x} \;\middle|\; \Big[-\mathbf{b}|A\Big]\begin{pmatrix}1\\ \mathbf{x}\end{pmatrix} \leq \mathbf{0}\right\} = \left\{\begin{pmatrix}x_0\\ \mathbf{x}\end{pmatrix} \;\middle|\; \Big[-\mathbf{b}|A\Big]\begin{pmatrix}x_0\\ \mathbf{x}\end{pmatrix} \leq \mathbf{0}\right\} \cap \{\mathbf{x} \mid x_0 = 1\}$$

It follows that

$$\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\} = \left\{\begin{pmatrix}x_0\\ \mathbf{x}\end{pmatrix} \;\middle|\; \Big[-\mathbf{b}|A\Big]\begin{pmatrix}x_0\\ \mathbf{x}\end{pmatrix} \leq \mathbf{0}\right\} \cap \{\mathbf{x} \mid x_0 = 1\}$$

$\square$

2.3.2 V-Polyhedra $\leftrightarrow$ V-Cone

Proposition 2.3.2 (V-Polyhedron $\to$ V-Cone)**.** *Let* Π *be the identity matrix with the first row deleted. Then*

$$\text{cone}(U) + \text{conv}(V) = \Pi\left(\text{cone}\begin{pmatrix}\mathbf{0} & \mathbf{1}\\ U & V\end{pmatrix} \cap \{\mathbf{x} \mid x_0 = 1\}\right)$$

Proof. 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. $\square$

This shows that a V-Polyhedron may be written as an intersection of a V-Cone and the hyperplane $\{\mathbf{x} \mid x_0 = 1\}$. It is more difficult to show that, given a V-Cone, that you can intersect it with the hyperplane $\{\mathbf{x} \mid x_0 = 1\}$ and get a V-Polyhedron out of it.

Proposition 2.3.3 (V-Cone → V-Polyhedron)**.** *Given some finite U, there are finite sets W and V such that*

$$\operatorname{cone}(U) \cap \{\mathbf{x} \mid x_0 = 1\} = \operatorname{cone}(W) + \operatorname{conv}(V)$$

Proof. 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:

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

Then I claim that

$$\operatorname{cone}(U) \cap \{\mathbf{x} \mid x_0 = 1\} = \operatorname{cone}(W) + \operatorname{conv}(V)$$

Say $\mathbf{x} \in \operatorname{cone}(W)$, and $\mathbf{y} \in \operatorname{conv}(V)$. Then $\mathbf{x}$ 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, $\mathbf{y}$ can be written:

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

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

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

So $y_0 = 1$ and $x_0 + y_0 = 1$. It follows that $\mathbf{x} + \mathbf{y} \in \operatorname{cone}(U) \cap \{\mathbf{x} \mid x_0 = 1\}$.

Next, suppose that $\mathbf{z} \in \operatorname{cone}(U) \cap \{\mathbf{x} \mid x_0 = 1\}$, 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 'Closure Property of Cones' on page 3, we need only show that

1. $\boldsymbol{\sigma}_Z \in \operatorname{cone}(W)$
2. $(\sigma_i\boldsymbol{\sigma}_N - \sigma_j\boldsymbol{\sigma}_P) \in \operatorname{cone}(W)$
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:

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

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 $\boldsymbol{\sigma}_P/\sigma_i \in \operatorname{conv}(V)$. $\square$

2.4 Picture of the Proof

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

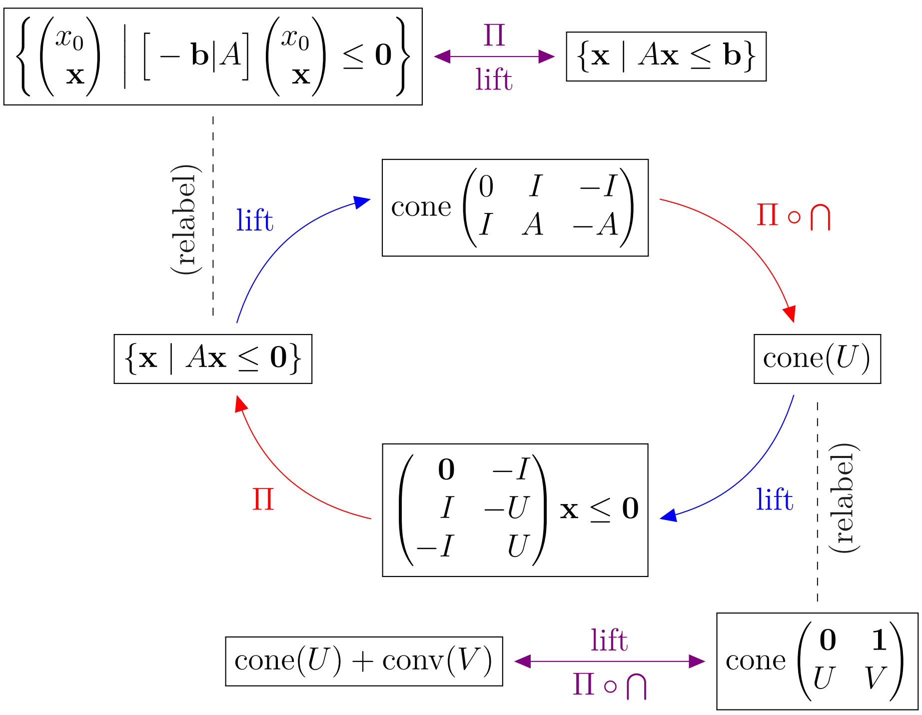


Figure 2.1: Diagram of the proof $P_H \leftrightarrow P_V$

Figure 2.1 shows the flow from an H-Polyhedron to a V-Polyhedron and back. There are violet arrows for transformations back and forth from polyhedra to cones, blue arrows to show the transformation between cones and intermediate representations, and red arrows to show where Fourier Motzkin elimination is applied to reduce these intermediate represenations to standard 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 and projected ($\Pi \circ \bigcap$).

3. C++ Implementation

The above transformations have been implemented in C++. Program `main` takes one argument specifying the type of input object (V-Cone, V-Polyhedra, H-Cone, or H-Polyhedra). It reads the description of the object from standard input, and writes the result of the implied transformation to standard output (details below). 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:

  arg:   input:         output:
  -vc    V-Cone         H-Cone
  -vp    V-Polyhedra    H-Polyhedra
  -hc    V-Cone         H-Cone
  -hp    V-Polyhedra    H-Polyhedra

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`	`<C++ standard library>`
`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 subsequent sections.

- `linear_algebra.h`
  Defines the types `Vector` and `Matrix`, which are the primary means of representing polyhedra, and some basic functionality for them

- `fourier_motzkin.h`
  Implementation of Fourier Motzkin elimination, and the Minkowski-Weyl Theorem for cones

- `polyhedra.{cpp,h}`
  Transforms between polytopes and polyhedra, completing the Minkowski-Weyl Theorem

- `test_functions.h`
  Types and functions for testing the algorithms (see Chapter 4).

- `test.cpp`
  Test cases for the algorithms and 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 class is designed to enforce the following invariant:

$$(\forall \texttt{v} \in \texttt{vectors})\, \texttt{v.size() == d}$$

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 `Matrixs` 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.

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

```
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 `Vector`s $\texttt{l} \in \mathbb{R}^{\texttt{l.size()}}$ and $\texttt{r} \in \mathbb{R}^{\texttt{r.size()}}$ and `return`s the `Vector` $(\texttt{l},\texttt{r}) \in \mathbb{R}^{\texttt{l.size() + r.size()}}$

```
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` `return`s 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` `return`s 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` `return`s 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.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:

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

```cpp
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 reducing the dimensionality), or the intersection of a V-Cone with a coordinate-hyperplane.

```cpp
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:

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

convolute the vectors in the way described in 'Fourier Motzkin Elimination for H-Cones' on page 7 and 'Fourier Motzkin Elimination for V-Cones' on page 12 (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. (Here, the slice operation is used to reduce the dimensionality as appropriate).

```
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 V-Cone Lift and H-Cone Lift, and summarized here:

$$\mathrm{T_H}(A) = \begin{pmatrix} \mathbf{0} & I & -I \\ I & A & -A \end{pmatrix} \quad \mathrm{T_V}(U) = \begin{pmatrix} \mathbf{0} & -I \\ I & -U \\ -I & U \end{pmatrix}$$

Note that the tranformation of U can be written:

$$\mathrm{T_V}(U) = \begin{pmatrix} \mathbf{0} & I & -I \\ -I & -U & U \end{pmatrix}^T$$

Remembering that a `Matrix` is either a collection of row *or* column `Vectors`, it is not surprising that these two transformations can be written as one function whose parameters are 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 `Vectors`, or

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

where `Matrix M` is a collection of column `Vectors`.

```
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 `LiftSelector`. The LiftSelector type is an enumerable class, used to avoid the need for function pointers.

```
Matrix cone_transform(const Matrix &cone,
                      LiftSelector lift) {
  if (cone.empty()) {
    throw logic_error{"empty cone for transform"};
  }
  switch (lift) {
    case LiftSelector::lift_vcone: {
      return sliced_fourier_motzkin(
        lift_vcone(cone), slice(0, cone.d, 1));
    } break;
    case LiftSelector::lift_hcone: {
      return sliced_fourier_motzkin(
        lift_hcone(cone), slice(0, cone.d, 1));
    } break;
    default: {
      throw std::logic_error{"invalid LiftSelector"};
    }
  }
}
```

`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,LiftSelector::lift_vcone);
}
```

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

3.5 `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` takes the members of `U` that have $x_0 > 0$, scaled by $1/x_0$. 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:

$$\Pi\left(\{\mathbf{x}/x_0 : \mathbf{x} \in P\} \cap \{x_0 = 1\}\right)$$

```
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 V-Cone → V-Polyhedron.

```
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.

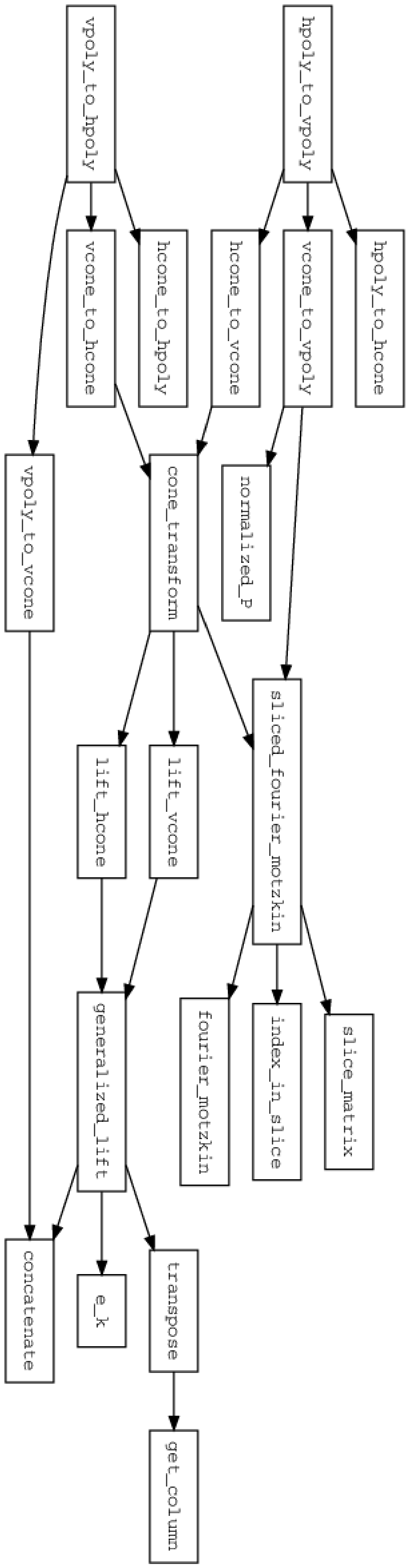
```
VPoly hpoly_to_vpoly(Matrix hpoly) {
  return vcone_to_vpoly(
           hcone_to_vcone(
             hpoly_to_hcone(move(hpoly))));
}
```

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

3.6 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.

For such a small callgraph, observation is enough to get some insight into the program. In particular, the nodes with the highest degrees (5) are: fourier_motzkin, cone_transform, and generalized_lift. Each have two incoming edges, reflecting the "H" vs "V" aspects of the program. It makes sense that these would be the functions getting higher degree in the program, as these are (roughly speaking) the most important parts of the proof of the theorem.


vpoly_to_hpoly
hpoly_to_vpoly
vcone_to_hcone
hcone_to_hpoly
hcone_to_vcone
vcone_to_vpoly
hpoly_to_hcone
vpoly_to_vcone
cone_transform
normalized_p
sliced_fourier_motzkin
lift_hcone
lift_vcone
generalized_lift
fourier_motzkin
index_in_slice
slice_matrix
concatenate
e_k
transpose
get_column

4. Testing

In the next sections, the methods used for testing the program described above will be discussed. It will be convenient to assume that sets representing row vectors and cone-generators do not contain $\mathbf{0}$. This results in no loss of generality, only the annoyance of constantly assuming some triviality does not occur.

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

$$AV' \leq \mathbf{0} \Rightarrow C_{V'} \subseteq C_A$$

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

$$\begin{aligned}
&AV' \leq \mathbf{0} &&\Rightarrow C_{V'} \subseteq C_A \\
&V \subseteq V' &&\Rightarrow C_A \subseteq C_{V'} \\
&C_{V'} = C_A &&\Rightarrow V \subseteq V' \\
&C_{V'} = C_A &&\Rightarrow AV' \leq \mathbf{0}
\end{aligned}$$

That is, we'd have necessary and sufficient conditions to test cone-equality (not to mention an obvious way to implement tests for these conditions). However, as of right now, this test is just wishful thinking.

The problem is to come up with such a set V, and to determine when such a set may or may not exist for a given cone. We will need to relax the requirements on V a little bit, but not in a way that reduces its utility. First, we consider a *minimal* set generating a cone.

Definition 4.1.1 (Minimal Set)**.** A set V is called *minimal* for $\text{cone}(V)$ if

$$(\forall \mathbf{v} \in V)\ \text{cone}(V \setminus \{\mathbf{v}\}) \subset \text{cone}(V)$$

Proposition 4.1.1. *If a set V is not minimal for* $\text{cone}(V)$ *then*

$$\exists \mathbf{v} \in V, \mathbf{t} \geq \mathbf{0}, \mathbf{v} = V\mathbf{e}_i, \mathbf{t} \neq \mathbf{e}_i : \quad \mathbf{v} = V\mathbf{t}$$

That is, there is a member of V which is a non-trivial non-negative linear combination of elements of V.

Proof. Say $\text{cone}(V \setminus \{\mathbf{v}\}) = \text{cone}(V)$ where $\mathbf{v} = V\mathbf{e}_i$. Then $\exists \mathbf{t} \geq \mathbf{0}$ such that $\mathbf{v} = (V \setminus \{\mathbf{v}\})\mathbf{t}$. Let $\mathbf{t}'$ be $\mathbf{t}$ with a 0 in the position corresponding to $\mathbf{v}$ in V. Then $\mathbf{v} = V\mathbf{t}$. □

Is the converse true? That is, is it true that, if V is minimal, then

$$\mathbf{t} \geq \mathbf{0},\ \mathbf{v} = V\mathbf{e}_i,\ [\mathbf{v} = V\mathbf{t} \Rightarrow \mathbf{t} = \mathbf{e}_i] \tag{4.1}$$

Not quite. There is one catch, if there is some

$$\mathbf{t} \geq \mathbf{0},\ \mathbf{t} \neq \mathbf{0},\ V\mathbf{t} = \mathbf{0} \tag{4.2}$$

then (4.1) fails. So, for what cones does (4.2) fail? It turns out that there is a useful class of cones called *pointed* having this property.

Definition 4.1.2 (Vertex)**.** Let P be a polyhedron. A point $\mathbf{v} \in P$ is called a *vertex* if, for any $\mathbf{u} \neq \mathbf{0}$, at least one of the following is true:

$$\mathbf{v} + \mathbf{u} \notin P$$
$$\mathbf{v} - \mathbf{u} \notin P$$

Definition 4.1.3 (Pointed Cones)**.** A cone is called *pointed* if it has a vertex.

Proposition 4.1.2. *The following statements are equivalent.*

1. $\mathrm{cone}(V)$ *is pointed.*
2. $\mathbf{t} \geq \mathbf{0},\ \mathbf{t} \neq \mathbf{0},\ [V\mathbf{t} = \mathbf{0} \Rightarrow \mathbf{t} = \mathbf{0}]$

Proof. First, observe that, due to Closure Property of Cones, if a cone has a vertex, then it is the origin. To see this, take any other point in the cone, and scale it by $1 \pm \epsilon$ for some appropriately small value ϵ.

Suppose that the origin is a vertex, but that (2) fails. Since $\mathbf{0} \notin V$, $\mathbf{t}$ has at least two non-zero elements, let one be t_i. Then $\mathbf{0} = V(t_i\mathbf{e}_i) + V(\mathbf{t} - t_i\mathbf{e}_i)$. Let $\mathbf{u} = Vt_i\mathbf{e}_i$. Clearly $\mathbf{u} \neq \mathbf{0}$, but also $-\mathbf{u} = V(\mathbf{t} - t_i\mathbf{e}_i) \in C$, so that $\mathbf{u}, -\mathbf{u} \in C$. Then the origin is not a vertex, a contradiction.

Next, suppose that $\mathbf{0}$ is not a vertex, then $\exists \mathbf{t}_1, \mathbf{t}_2 \geq \mathbf{0}$, $\mathbf{t}_{1,2} \neq \mathbf{0}$, $\mathbf{u} = V\mathbf{t}_1$, $-\mathbf{u} = V\mathbf{t}_2$. Then $\mathbf{t}_1 + \mathbf{t}_2 \geq \mathbf{0}$, $\mathbf{t}_1 + \mathbf{t}_2 \neq \mathbf{0}$, and $V(\mathbf{t}_1 + \mathbf{t}_2) = \mathbf{0}$. ☐

Now we can consider the converse of Proposition 4.1.1.

Proposition 4.1.3 (Minimal V-Cone Generators)**.** *Suppose that* $\mathrm{cone}(V)$ *is pointed. Then the following two statements are equivalent:*

1. *V is minimal*
2. $\mathbf{t} \geq \mathbf{0},\ \mathbf{v} = V\mathbf{e}_i,\ [\mathbf{v} = V\mathbf{t} \Rightarrow \mathbf{t} = \mathbf{e}_i]$

Proof. $(\neg 1 \Rightarrow \neg 2)$ is Proposition 4.1.1. So suppose that $\mathbf{t} \geq \mathbf{0}$, $\mathbf{v} = V\mathbf{e}_i$, and $\mathbf{v} = V\mathbf{t}$. If $0 \leq t_i < 1$, then $\mathbf{v} = V(\mathbf{t} - t_i\mathbf{e}_i)/(1 - t_i)$, and $\mathbf{v} \in \mathrm{cone}(V \setminus \{\mathbf{v}\})$, which would mean that V is not minimal. Suppose that $t_i \geq 1$. Then $\mathbf{t} - \mathbf{e}_i \geq \mathbf{0}$, and $\mathbf{0} = V(\mathbf{t} - \mathbf{e}_i)$. Because V is pointed, by Proposition 4.1.2 $\mathbf{0} = \mathbf{t} - \mathbf{e}_i$, so $\mathbf{t} = \mathbf{e}_i$. ☐

Proposition 4.1.3 gives us a way to characterize the minimal sets generating V-Cones. Clearly, there is not a unique minimal set generating any V-Cone, since any positive scaling of any of the vectors generating the cone results in the same cone. However, as one is wont to do upon encountering such trifles, we can relax the requirement of unicity to equivalence, in the following way.

Definition 4.1.4 (vector equivalence)**.** Let $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$, non-zero, and suppose that $\mathbf{u}/\,||\mathbf{u}|| = \mathbf{v}/\,||\mathbf{v}||$. Then say that $\mathbf{u}, \mathbf{v}$ are *equivalent*, and write $\mathbf{u} \simeq \mathbf{v}$. If for every $\mathbf{u} \in U$ there is a $\mathbf{v} \in V$ such that $\mathbf{u} \simeq \mathbf{v}$, write $U \sqsubseteq V$. Write $U \simeq V$ if $U \sqsubseteq V$ and $V \sqsubseteq U$.

Proposition 4.1.4. *Then the following two statements are equivalent:*

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

2. $(\exists t > 0)\ \mathbf{v} = t\mathbf{u}$

Proof. $(1 \Rightarrow 2)$. Let $t = ||\mathbf{v}||\,/\,||\mathbf{u}||$. Then $t > 0$, and $\mathbf{v} = t\mathbf{u}$.
$(2 \Rightarrow 1)$. $\mathbf{v}/\,||\mathbf{v}|| = t\mathbf{u}/\,||t\mathbf{u}|| = \mathbf{u}/\,||\mathbf{u}||$ □

We now show that the minimal sets generating pointed V-Cones are essentially unique.

Proposition 4.1.5 (Minimal Generators of a Pointed Cone)**.** *Suppose that V is minimal, and* $\text{cone}(V) = \text{cone}(V')$ *is pointed. Then $V \sqsubseteq V'$. It follows that if V' is also minimal, then $V \simeq V'$.*

We'll use this short lemma in the proof of the above proposition.

Lemma 4.1.6. *Suppose A is a non-negative matrix, $\mathbf{b} \geq \mathbf{0}$, and $A\mathbf{b} = \mathbf{e}_i$. Then there exists an l, $t > 0$ such that $A(t\mathbf{e}_l) = \mathbf{e}_i$*

Proof. Since A and $\mathbf{b}$ are non-negative, the following holds:

$$(\forall j, k \neq i)\ b_j > 0 \Rightarrow A^j_k = 0 \tag{4.3}$$

Since $A\mathbf{b} = \mathbf{e}_i$, there is some $b_l > 0$, and $A^l_k > 0$. (4.3) shows that the entire column is zero except for the entry in row i, so $A(\mathbf{e}_l/A^l_k) = \mathbf{e}_i$. □

Proof of Proposition 4.1.5. Let $\mathbf{v} \in V$, $\mathbf{v} = V\mathbf{e}_i$. If we can show that there is some $\mathbf{v}' \in V'$ such that $\mathbf{v} \simeq \mathbf{v}'$, then we're done. Since $\text{cone}(V) = \text{cone}(V')$, there is a non-negative matrix A such that $V' = VA$. Furthermore, there is a non-negative vector $\mathbf{b}$ such that $\mathbf{v} = V'\mathbf{b}$. Then $\mathbf{v} = V'\mathbf{b} = (VA)\mathbf{b} = V(A\mathbf{b})$. By Proposition 4.1.3, $A\mathbf{b} = \mathbf{e}_i$. By Lemma 4.1.6, there is a $t > 0, l$ such that $A\mathbf{b} = A(t\mathbf{e}_l)$. Then $\mathbf{v} = VA(t\mathbf{e}_l) = tV'\mathbf{e}_l = t\mathbf{v}'$ where $\mathbf{v}' \in V'$. By Proposition 4.1.4, $\mathbf{v} \simeq \mathbf{v}'$. □

So now we know that pointed cones have essentially unique generating sets. We now turn to the question of using this knowledge to create a test for the program. We suppose that we have a minimal generating set V for some pointed V-Cone C, and have created a matrix A so that $C = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\} = \text{cone}(V)$. We run the program and get a set V', and let $C' = \text{cone}(V')$. We must check that

$C' = C$. The situation is summarized here below, and formalized in the following Equivalence Criteria.

$$
\begin{aligned}
AV' \leq \mathbf{0} &\Rightarrow C' \subseteq C \\
V \sqsubseteq V' \; &\Rightarrow C \subseteq C' \\
C' = C \; &\Rightarrow V \sqsubseteq V' \\
C' = C \; &\Rightarrow AV' \leq \mathbf{0}
\end{aligned}
$$

Equivalence Criteria 1 (H-Cone $\rightarrow$ V-Cone)**.** *Say V is a minimal generating set for the pointed V-Cone C, and suppose $C = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\} = \operatorname{cone}(V)$. Then*

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

Test 1 (H-Cone $\rightarrow$ V-Cone)**.** 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 minimal set V for some pointed V-Cone. We then run our program to get a set V', with the alleged property that $\operatorname{cone}(V') = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$. If we confirm Equivalence Criteria 1, then our program has succeeded.

Remark 3. Can we test the program for non-pointed cones? Yes, but it is slightly more complicated. Instead of prior knowledge of a minimal generating set for the cone, we also need to know what the largest linear subspace L contained in the cone. If we project away this linear subspace, then we will have a pointed cone. Given another set V', we may project away this subspace from V' using a projection matrix, and use Test 1. Then we need to see if $\operatorname{cone}(V')$ spans L. This can be done with a modified fourier-motzkin elimination, but unfortunately we are trying to test the implementation of fourier-motzkin elimination.

It may still be worthwhile to do such tests, but it should be noted that a test isn't designed to prove a program correct, only prove it incorrect. If we analyze the program well and test the fourier-motzkin elimination extensively, then the added complexity of the more general testing may not be worth it. As of now this is left as a possible future extension of the program.

Remark 4. While not important for testing the program, one may ask if pointed V-Cones are the only cones with essentially unique generating sets. The answer is no, for any line has an essentially unique generating set, but is not pointed. However, this is the only exception. It isn't hard to see that, given a non-pointed cone, if it occupies more than one-dimension, then it must at least occupy a half-plane, and a halfplane has uncountably many non-equivalent generators. So, technically, the Test 1 would work for one-dimensional non-pointed cones (lines).

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

In this section we create a method in the vein of Test 1, but for testing the program transforming V-Cones to H-Cones. This section is almost identical to the previous, with the exception of requiring the Farkas Lemma.

Definition 4.2.1 (Minimal Set of Constraints)**.** A set A is called *minimal* for $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$ if

$$(\forall A_i \in A)\ \{\mathbf{x} \mid A \setminus \{A_i\}\,\mathbf{x} \leq \mathbf{0}\} \supset \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$$

Proposition 4.2.1. *If a set A is not minimal for $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$ then*

$$\exists A_i \in A,\, \mathbf{t} \geq \mathbf{0},\, A_i = \mathbf{e}_i^T A,\, \mathbf{t} \neq \mathbf{e}_i : \quad A_i = \mathbf{t}^T V$$

That is, there is a member of A which is a non-trivial non-negative linear combination of elements of V.

In order to prove Proposition 4.2.1, we require the Farkas Lemma.

4.2.1 The Farkas Lemma

Proposition 4.2.2 (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 is simple. Suppose they both were, then:

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

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,

$$(\exists \mathbf{t}) : A \begin{pmatrix} \mathbf{x} \\ \mathbf{t} \end{pmatrix} \leq \mathbf{0} \Leftrightarrow (\exists \mathbf{t} \geq \mathbf{0})\ \mathbf{x} = U\mathbf{t} \tag{4.4}$$

In the proof of the transformation, we use 'Fourier Motzkin Elimination for H-Cones' on page 7 to transform that matrix A. The 'Fourier Motzkin Matrix' on page 8 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, in particular that the last columns are $\mathbf{0}$,

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

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 $\mathbf{y}' \in Y'$:

$$\mathbf{y}'U \leq \mathbf{0} \tag{4.5}$$

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

$$(\exists \mathbf{t}) : A \begin{pmatrix} \mathbf{x} \\ \mathbf{t} \end{pmatrix} \leq \mathbf{0} \Leftrightarrow (YA) \begin{pmatrix} \mathbf{x} \\ \mathbf{t} \end{pmatrix} \leq \mathbf{0} \Leftrightarrow Y'\mathbf{x} \leq \mathbf{0} \tag{4.6}$$

Note that here $\mathbf{x} \in \mathbb{R}^d$. So, if given some $\mathbf{x}$, the left side of (4.6) is not satisfied, then neither is the right, and there must be some row $\mathbf{y}' \in Y'$ such that the following holds:

$$\langle \mathbf{y}', \mathbf{x} \rangle > 0 \tag{4.7}$$

Then we conclude that, if the right side of (4.4) fails, then there is a vector $\mathbf{y}' \in Y'$ satisfying (4.5) and (4.7). ☐

Remark 5. The Farkas Lemma above can be equivalently stated:

$$(\exists \mathbf{t} \geq \mathbf{0}) : \mathbf{t}^T A = \mathbf{y} \quad \Leftrightarrow \quad (\forall \mathbf{x}) : A\mathbf{x} \leq 0 \Rightarrow \langle \mathbf{y}, \mathbf{x} \rangle \leq 0$$

This way of writing it makes it clear that, if $\mathbf{y}^T\mathbf{x} \leq 0$ holds for every $\mathbf{x}$ in some H-Cone $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$, then $\mathbf{y}$ is a non-negative linear combination of the rows of A.

Proof of Proposition 4.2.1. Say $\{\mathbf{x} \mid (A \setminus \{A_i\})\mathbf{x} \leq \mathbf{0}\} = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$. Then, by Remark 5, $A_i^T\mathbf{x} \leq 0$ holds for $\{\mathbf{x} \mid (A \setminus \{A_i\})\mathbf{x} \leq \mathbf{0}\} = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$, so A_i is a non-negative linear combination of some rows of $A \setminus \{A_i\}$. ☐

As before, the converse will fail if we can combine rows of A in a non-trivial way to get $\mathbf{0}$. For which cones does this occur? Well, it would be necessary that the following holds for some $\mathbf{y}$:

$$\mathbf{y}^T\mathbf{x} \leq 0, \quad -\mathbf{y}^T\mathbf{x} \leq 0$$

But this means that $\mathbf{y}^T\mathbf{x} = 0$ holds for every member of the cone. We can prevent this from occuring by forcing the cone to contain a basis.

Definition 4.2.2 (Full-Dimensional Cones)**.** A cone is called *full-dimensional* if it contains a basis of its ambient space.

The most important property (well known from linear algebra) of a basis B that we shall use is:

$$\mathbf{y}^T B = \mathbf{0} \Rightarrow \mathbf{y} = \mathbf{0} \tag{4.8}$$

Proposition 4.2.3. *The following statements are equivalent.*

1. $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$ *is full-dimensional.*
2. $\mathbf{t} \geq \mathbf{0}$, $\mathbf{t} \neq \mathbf{0}$, $[\mathbf{t}^T A = \mathbf{0} \Rightarrow \mathbf{t} = \mathbf{0}]$

Proof. ($\neg 1 \Rightarrow \neg 2$). If $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$ is not full-dimensional, then there is some $\mathbf{y}$ so that for every $\mathbf{x}$ the cone $\mathbf{y}^T\mathbf{x} = 0$. Then, by Remark 5, we'd have some non-negative $\mathbf{t}_1, \mathbf{t}_2$ such that $\mathbf{t}_1^T A = \mathbf{y}$ and $\mathbf{t}_2^T A = -\mathbf{y}$, in which case $\mathbf{t}_1 + \mathbf{t}_2$ is a counter example to (2).

($\neg 2 \Rightarrow \neg 1$). Suppose $\mathbf{t} \geq \mathbf{0}$, $\mathbf{t}^T A = \mathbf{0}$, and $\mathbf{t} \neq \mathbf{0}$. Since $\mathbf{0} \notin A$, at least two elements of $\mathbf{y}$ are non-zero, say one is y_i. Then $\mathbf{0} = y_i A_i + (\mathbf{y} - y_i \mathbf{e}_i)^T A$, which then means both $A_i\mathbf{x} \leq 0$ and $-A_i\mathbf{x} \leq 0$ holds for $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$, in which case it is not full dimensional. ☐

Proposition 4.2.4. *Suppose that* $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$ *is full-dimensional. Then the following two statements are equivalent:*

1. *A is minimal*
2. $\mathbf{t} \geq \mathbf{0}$, $[A_i = \mathbf{t}^T A \Rightarrow \mathbf{t} = \mathbf{e}_i]$

Proof. ($\neg 1 \Rightarrow \neg 2$) is Proposition 4.2.1. So suppose that $\mathbf{t} \geq \mathbf{0}$, and $A_i = \mathbf{t}^T A$. If $0 \leq t_i < 1$, then $A_i = (\mathbf{t} - t_i\mathbf{e}_i)^T A/(1 - t_i)$, and $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\} = \{\mathbf{x} \mid (A \setminus \{A_I\})\mathbf{x} \leq \mathbf{0}\}$, which would mean that A is not minimal. Suppose that $t_i \geq 1$. Then $\mathbf{t} - \mathbf{e}_i \geq \mathbf{0}$, and $\mathbf{0} = (\mathbf{t} - \mathbf{e}_i)^T A$. Because A is full-dimensional, by Proposition 4.2.3, $\mathbf{0} = \mathbf{t} - \mathbf{e}_i$, so $\mathbf{t} = \mathbf{e}_i$. ☐

Proposition 4.2.5. *The following two statements are equivalent:*

1. $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$ *is full dimensional and A is minimal*
2. $\operatorname{cone}(A^T)$ *is pointed and A is minimal*

Proof. This follows from the nearly identical form of (2) in Proposition 4.2.4 and Proposition 4.1.3. ☐

In order to create an equivalence criterion like H-Cone → V-Cone, we use the following result.

Theorem 3 (Dual Cone)**.**

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

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 $\mathbf{z} \in \text{cone}(A^T), \mathbf{z} \notin \text{cone}(A'^T)$. 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 $\mathbf{y} \notin \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$. ☐

Proposition 4.2.6. *Suppose that A is minimal, and $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\} = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\}$ is full-dimensional. Then $A \sqsubseteq A'$. It follows that if A' is also minimal, then $A \simeq A'$.*

Proof. By Proposition 4.2.5 and Theorem 3, Proposition 4.2.6 is true if it is true for cones, which is shown in Minimal Generators of a Pointed Cone. ☐

Say we know that $C = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\} = \text{cone}(V)$ is full-dimensional, with A minimal. We have another set A' and let $C' = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\}$. Then we can test if $C' = C$. The following summarizes the situation:

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

Equivalence Criteria 2 (V-Cone → H-Cone)**.** *Say H is a minimal generating set of constraints for the full-dimensional H-Cone C, and suppose $C = \text{cone}(V) = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$. Then*

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

Test 2 (H-Cone → V-Cone)**.** We now have a method for testing the program. First, we hand-craft a V-Cone $\text{cone}(V)$ based on minimal set A for some pointed H-Cone. We then run our program to get a set A', with the alleged property that $\text{cone}(V) = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\}$. If we confirm Equivalence Criteria 2, then our program has succeeded.

Remark 6. Can we test the program for non-full-dimensional cones? Once again, this is more complicated and requires a technique to remove the "degeneracy," and once again test for equivalence otherwise.

As of now this is left as a possible future extension of the program.

Remark 7. While not important for testing the program, one may ask if full-dimensional H-Cones are the only cones with essentially unique generating setsof constraints . The answer is no, for any set of the form $\mathbf{y}^T\mathbf{x} = c$ has an essentially unique generating of constraints. However, this is the only exception. It isn't hard to see that, given indpendent constraints of the form $A\mathbf{x} = \mathbf{0}$, if A has more than two rows, then, for any non-singular B, $BA\mathbf{x} = \mathbf{0}$ is an equivalent constraint. So, technically, the Test 1 would work for hyperplanes.

Generalizing to Polyhedra In the following sections we generalize Test 1 and Test 2 to polyhedra.

4.3 Testing H-Polyhedron → V-Polyhedron

Say we have an H-Polyhedron $P_{A,\mathbf{b}} = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$, and wish to check that our program correctly calculates a V' and U' such that $P_{A,\mathbf{b}} = \mathrm{cone}(U') + \mathrm{conv}(V')$. Again, we shall use the notion of minimality and show that under certain circumstances we can use minimal sets to demonstrate the validity of our algorithm. First, we consider the case of a V-Polyhedron with no cone.

4.3.1 Polytopes

First we consider the special case of a V-Polyhedron given by $P = \mathrm{conv}(V)$. Such a set is known as a *polytope*.

Definition 4.3.1 (Minimal Set for Polytopes)**.** A set V is called *minimal* for the polytope $\mathrm{conv}(V)$ if:

$$(\forall \mathbf{v} \in V)\ \mathrm{conv}(V \setminus \{\mathbf{v}\}) \subset \mathrm{conv}(V)$$

Proposition 4.3.1. *V is minimal for* $\mathrm{conv}(V)$ *if and only if V is the set of vertices of* $\mathrm{conv}(V)$*.*

We will need:

Proposition 4.3.2. *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 $\boldsymbol{\lambda} \geq \mathbf{0}$, $\mathbf{1}^T\boldsymbol{\lambda} = 1$ be a convex combinator. Then $\Lambda\boldsymbol{\lambda} = \boldsymbol{\lambda}'$ where $\boldsymbol{\lambda}' \geq \mathbf{0}$, $\mathbf{1}^T\boldsymbol{\lambda}' = 1$. That $\boldsymbol{\lambda}' \geq \mathbf{0}$ is clear, then just note that $\mathbf{1}^T\boldsymbol{\lambda}' = \mathbf{1}^T\Lambda\boldsymbol{\lambda} = \mathbf{1}^T\boldsymbol{\lambda} = 1$. □

Proof of Proposition 4.3.1. First, suppose that V is not minimal. Then there is a $v \in V$ that satisfies $\mathrm{conv}(V \setminus \{\mathbf{v}\}) = \mathrm{conv}(V)$. Denote $V' = V \setminus \{\mathbf{v}\}$. Then $\mathbf{v} = V'\boldsymbol{\lambda}'$, where there is some λ_i with $0 < \lambda_i < 1$. Let $\mathbf{u} = V'(\mathbf{e}_i - \boldsymbol{\lambda}')$. Then

$$\mathbf{v} + \lambda_i\mathbf{u} = V'\boldsymbol{\lambda}' + \lambda_i V'(\mathbf{e}_i - \boldsymbol{\lambda}') = (1 - \lambda_i)V'\boldsymbol{\lambda}' + \lambda_i V'\mathbf{e}_i$$

By Proposition 4.3.2, the right hand side of this equation is a convex combination of members of V', so $\mathbf{v} - \lambda_i \mathbf{u} \in \text{conv}(V')$. Similarly,

$$\mathbf{v} - \lambda_i \mathbf{u} = V'\boldsymbol{\lambda}' - \lambda_i V'(\mathbf{e}_i - \boldsymbol{\lambda}') = V'(\boldsymbol{\lambda}' - \lambda_i \mathbf{e}_i) + V'(\lambda_i \boldsymbol{\lambda}')$$

Consider $(\boldsymbol{\lambda}' - \lambda_i \mathbf{e}_i)$. Note that this expression is non-negative, and sums to $1 - \lambda_i$. Next note that $\lambda_i \boldsymbol{\lambda}'$ is non-negative, and sums to λ_i. This means that the right hand side of the equation is a convex combination of V', so $\mathbf{v} + \lambda_i \mathbf{u} \in \text{conv}(V')$, and $\mathbf{v}$ is not a vertex.
Next, suppose that $\mathbf{v} \in V$ is not a vertex, and let $V' = V \setminus \{\mathbf{v}\}$. Then there is some non-zero $\mathbf{u}$ such that $\mathbf{v} + \mathbf{u} \in \text{conv}(V)$, $\mathbf{v} - \mathbf{u} \in \text{conv}(V)$. First, let $\alpha, \beta > 0$, and consider $\mathbf{v} + \alpha \mathbf{u}$ and $\mathbf{v} - \beta \mathbf{u}$. Observe that

$$\frac{\beta(\mathbf{v} + \alpha \mathbf{u})}{\alpha + \beta} + \frac{\alpha(\mathbf{v} - \beta \mathbf{u})}{\alpha + \beta} = \frac{\alpha \mathbf{v} + \beta \mathbf{v}}{\alpha + \beta} = \mathbf{v}$$

This shows that we can positively scale α and β, and still get $\mathbf{v}$ as a convex combination of the result. So we search for positive α and β that give a point of $\text{conv}(V')$, which by Proposition 4.3.2 shows that $\mathbf{v} \in \text{conv}(V')$ so V is not minimal. First observe that $\mathbf{v} + \mathbf{u} = V\boldsymbol{\lambda}$ for some $\boldsymbol{\lambda}$. Then let $\boldsymbol{\lambda}' = \boldsymbol{\lambda} - \lambda_i \mathbf{e}_i$, so $\mathbf{u} + \mathbf{v} = V'\boldsymbol{\lambda}' + \lambda_i \mathbf{v}$, and $\mathbf{u} = V'(\boldsymbol{\lambda}') + (\lambda_i - 1)\mathbf{v}$ Then

$$\mathbf{v} + \alpha \mathbf{u} = \mathbf{v} + \alpha(\lambda_i - 1)\mathbf{v} + \alpha V'\boldsymbol{\lambda}'$$

So we let $\alpha = 1/(1 - \lambda_i)$, and the term in $\mathbf{v}$ disappears, while $\boldsymbol{\lambda}'/(1 - \lambda_i)$ is a convex combination. Similarly, we have $\mathbf{v} - \mathbf{u} = V\boldsymbol{\mu}$, and $\mathbf{u} = \mathbf{v}(1 - \mu_i) - V'\boldsymbol{\mu}'$. Then

$$\mathbf{v} - \beta \mathbf{u} = \mathbf{v}(1 - \beta(1 - \mu_i)) + \beta V'\boldsymbol{\mu}'$$

So let $\beta = 1/(1 - \mu_i)$, so the right hand side is a convex combination of members of V'. $\square$

4.3.2 Characterstic Cone

Now we consider the set $\text{cone}(U)$ in $\text{cone}(U) + \text{conv}(V)$. The next proposition shows that it is essentially unique for any given Polyhedron.

Proposition 4.3.3 (Characterstic Cone). *Suppose that* $P = \{\mathbf{x} \mid A\mathbf{x} \leq b\} = \text{cone}(U) + \text{conv}(V)$. *Then 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. $\mathbf{r} \in \text{cone}(U)$

Proof. $(1 \Rightarrow 2)$. $\mathbf{x} \in P$ 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:

Proposition 4.3.4. *A linear, real-valued function on the set* $\text{conv}(V)$ *achieves its maximal value at some* $\bar{\mathbf{v}} \in V$.

Proof. 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 8 (Characteristic Cone)**.** 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 $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\} = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{b}'\}$, the it holds that $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\} = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\}$.

4.3.3 Minimal V-Polyhedra Pairs

Definition 4.3.2. A pair (U, V) is said to be *minimal* for $\text{cone}(U) + \text{conv}(V)$ if

$$(\forall \mathbf{u} \in U)\ \text{cone}(U \setminus \{\mathbf{u}\}) + \text{conv}(V) \subset \text{cone}(U) + \text{conv}(V) \tag{4.9}$$
$$(\forall \mathbf{v} \in V)\ \text{cone}(U) + \text{conv}(V \setminus \{\mathbf{v}\}) \subset \text{cone}(U) + \text{conv}(V) \tag{4.10}$$

As before, the pair may not be essentially unique. This can happen if U is not pointed. So we will consider only pointed cones for $\text{cone}(U)$.

Proposition 4.3.5. *If* (U, V) *is minimal, then* U *is minimal for* $\text{cone}(U)$.

Proof. By Remark 8, $\text{cone}(U) + \text{conv}(V) = \text{cone}(U') + \text{conv}(V)$ if and only if $\text{cone}(U) = \text{cone}(U')$. So this means that the minimality of U is only dependent on U. □

Now we consider the vertices of $\text{cone}(U) + \text{conv}(V)$.

Proposition 4.3.6. *If* $\mathbf{v}$ *is a vertex of* $\text{cone}(U) + \text{conv}(V)$, *then* $[\mathbf{v} = U\mathbf{t} + V\boldsymbol{\lambda}] \Rightarrow \mathbf{t} = \mathbf{0}$.

Proof. If $\mathbf{v}$ can be written with some non-zero contribution from $\text{cone}(U)$, then you may decrease this contribution by some amount while staying in $\text{cone}(U) + \text{conv}(V)$, and you may increase the contribution by the same amount, so $\mathbf{v}$ is not a vertex. $\square$

It will be useful to refer to the property of a set V such that no member of V may be written with a non-zero contribution from $\text{cone}(U)$ (this is the property described by Proposition 4.3.6). We will call it U-free.

Proposition 4.3.7. *If $\mathbf{v}$ is a vertex of $\text{cone}(U) + \text{conv}(V)$, then $\mathbf{v}$ is a vertex of $\text{conv}(V)$.*

Proof. By Proposition 4.3.6, $\mathbf{v} \in \text{conv}(V)$. If $\mathbf{v}$ is not a vertex of $\text{conv}(V)$, then because $\text{conv}(V) \subseteq \text{cone}(U) + \text{conv}(V)$ it can't be a vertex of P. $\square$

Now we can show the following.

Proposition 4.3.8. *Suppose that (U, V) is a minimal pair for $\text{cone}(U)+\text{conv}(V)$. Then V is the set of vertices of $\text{cone}(U) + \text{conv}(V)$.*

Proof. By Proposition 4.3.6, if $\mathbf{v}$ is a vertex of P, then it must be a vertex of V. Clearly, V is minimal for V, and is precisely the vertices of $\text{conv}(V)$. The only question is if the vertices of V are the vertices of P. Suppose that $\mathbf{v}$ is a vertex of $\text{conv}(V)$. Then we must show that for any $\mathbf{u} \neq \mathbf{0}$, if $\mathbf{v} + \mathbf{u} \in P$ then $\mathbf{v} - \mathbf{u} \notin P$. Suppose that $\mathbf{u} \in \text{cone}(U)$. Then, since V is U-free, $\mathbf{v} - \mathbf{u} \notin P$, otherwise $\mathbf{v} = (\mathbf{v} - \mathbf{u}) + \mathbf{u}$ and V is not U-free. If $\mathbf{u} \notin \text{cone}(U)$, then if $\mathbf{v} + \mathbf{u} \in P$, $\mathbf{v} + \mathbf{u} \in \text{conv}(V)$. Then because $\mathbf{v}$ is a vertex of $\text{conv}(V)$, $\mathbf{v} - \mathbf{u} \notin \text{conv}(V)$. $\square$

Proposition 4.3.9. *Let $P = \text{cone}(U) + \text{cone}(V)$. Then the following are equivalent*

1. *(U, V) is minimal for P*
2. *U is minimal for $\text{cone}(U)$, V is the vertex set of P*
3. *U is minimal for $\text{cone}(U)$, V is the vertex set of $\text{conv}(V)$, and V is U-free*

Proof. ($1 \Rightarrow 2$). This combines the results of Proposition 4.3.5 and Proposition 4.3.8
($2 \Rightarrow 3$). That V is U-free follows from Proposition 4.3.6. By Proposition 4.3.7, the vertex set of P is a subset of the vertices of $\text{conv}(V)$. Let $\mathbf{v}$ be a vertex of $\text{conv}(V)$, we must show that it is a vertex of P. Because it is a vertex of $\text{conv}(V)$, if $\mathbf{v} + \mathbf{u} \in \text{conv}(V)$ then $\mathbf{v} - \mathbf{u} \notin \text{conv}(V)$. Say $\mathbf{v} + \mathbf{u} \in \text{conv}(V) + \text{cone}(U)$. Then $\mathbf{u}$ must have some non-zero contribution of $\text{cone}(U)$. If $\mathbf{v} - \mathbf{u} \in P$, then $\mathbf{v}$ could be written as $(\mathbf{v} + \mathbf{u})/2 + (\mathbf{v} - \mathbf{u})/2$, which has an overall positive contribution from $\text{cone}(U)$, meaning that V is not U-free.
($3 \Rightarrow 1$). Since V is the vertex set of $\text{conv}(V)$, if $\mathbf{v} \in V$ is also in $\text{cone}(U) + \text{conv}(V \setminus \{\mathbf{v}\})$, then $\mathbf{v}$ can be written with a non-negative contribution from $\text{cone}(U)$, so V is not U-free. Next let $\mathbf{u} \in U$, and $U' = U \setminus \{\mathbf{u}\}$. We must find a point in $\text{cone}(U) + \text{conv}(V)$ that is not in $\text{cone}(U') + \text{conv}(V)$. Because $\mathbf{u} \notin \text{cone}(U')$, there is an $\mathbf{x}$ that satisfies: $\mathbf{x}^T U' \leq \mathbf{0}$, and $\mathbf{x}^T \mathbf{u} > 0$. By Proposition 4.3.4, there is some maximum value c such that $(\forall \mathbf{x} \in \text{conv}(V))\mathbf{x}^T \mathbf{u} \leq c$.

This means that $\left\{\mathbf{x}^T\mathbf{y} : \mathbf{y} \in \text{cone}(U') + \text{conv}(V)\right\}$ is upper-bounded by c. But the set $\left\{\mathbf{x}^T\mathbf{y} : \mathbf{y} \in \text{cone}(U) + \text{conv}(V)\right\}$ is unbounded, since $\mathbf{x}^T\mathbf{u} > 0$. So we can conlude that $\text{cone}(U') + \text{conv}(V) \subset \text{cone}(U) + \text{conv}(V)$. We conclude that (U, V) are minimal. $\square$

Now we see that the minimal pairs for V-Polyhedra are essentially unique.

Proposition 4.3.10. *Let (U, V) be minimal for $P = \text{cone}(U) + \text{conv}(V) = \text{cone}(U') + \text{conv}(V')$. Then $U \sqsubseteq U'$, and $V \subseteq V'$.*

Proof. Since $\text{cone}(U) = \text{cone}(U')$, and U is minimal for $\text{cone}(U)$, by Equivalence Criteria 1 $U \sqsubseteq U'$. By Proposition 4.3.7 every vertex of P must be a vertex of V', and because V contains precisely the vertices of P, $V \subseteq V'$. $\square$

Say we know that $P = \text{cone}(U) + \text{conv}(V) = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$, with U, V minimal, and U pointed. We have another pair (U', V'), and let $P' = \text{cone}(U') + \text{conv}(V')$. We want to test if $P = P'$. We have the following:

$$
\begin{aligned}
AU' \leq \mathbf{0} &\Rightarrow \text{cone}(U') \subseteq \text{cone}(U) \\
AV' \leq \mathbf{b} &\Rightarrow \text{conv}(V') \subseteq P \\
U \sqsubseteq U' &\Rightarrow \text{cone}(U) \subseteq \text{cone}(U') \\
V \subseteq V' &\Rightarrow \text{conv}(V) \subseteq \text{conv}(V') \\
P' = P &\Rightarrow AU \leq \mathbf{0} \\
P' = P &\Rightarrow AV \leq \mathbf{b} \\
P' = P &\Rightarrow U \sqsubseteq U' \\
P' = P &\Rightarrow V \subseteq V'
\end{aligned}
$$

The first two lines imply that $P' \subseteq P$, while the next two imply that $P \subseteq P'$. We now have the ability to create an equivalence criteria.

Equivalence Criteria 3 (V-Cone $\to$ H-Cone)**.** *Say (U, V) is a minimal pair for $P = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$ (with pointed* $\text{cone}(U)$*), and suppose $P' = \text{cone}(U') + \text{conv}(V')$. Then*

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

Test 3 (H-Polyhedron $\to$ V-Polyhedron)**.** We now have a method for testing the program. First, we hand-craft an H-Polyhedron $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$ based on a minimal pair (U, V) for some pointed V-Polyhedron. We then run our program to get a pair (U', V'), with the alleged property that $\text{cone}(U') + \text{conv}(V') = \text{cone}(U) + \text{conv}(V)$. If we confirm Equivalence Criteria 3, then our program has succeeded.

4.4 Testing V-Polyhedron $\to$ H-Polyhedron

Now we suppose we have a V-Polyhedron $P_{U,V} = \text{cone}(U) + \text{conv}(V)$, and would like to test the program which returns a matrix-vector pair $A', \mathbf{b}'$ where supposedly $P_{U,V} = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{b}'\}$. Again, we will start off with a pair $A, \mathbf{b}$ where we know that $P_{U,V} = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$, where $A, \mathbf{b}$ satisfy some nice properties, and use those properties to test if $P_{U,V} = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{b}'\}$.

Definition 4.4.1 (Minimal H-Pair)**.** The pair $(A, \mathbf{b})$ is called *minimal* for $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$, if, for any row (A_i, b_i), letting $(A', \mathbf{b}')$ be $(A, \mathbf{b})$ with the i-th row deleted, $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\} \supset \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{b}'\}$.

To characterize Minimal H-Pairs, we will need a new form of the Farkas Lemma.

Theorem 4 (Farkas Lemma 2)**.**

$$(\exists \mathbf{t} \geq \mathbf{0})\mathbf{t}^T A = \mathbf{y},\ \mathbf{t}^T\mathbf{b} \leq c \Leftrightarrow \begin{cases} (\forall \mathbf{x}) A\mathbf{x} \leq \mathbf{0} \Rightarrow \mathbf{y}^T\mathbf{x} \leq 0 \ \textit{\textbf{and}} \\ (\forall \mathbf{x}) A\mathbf{x} \leq \mathbf{b} \Rightarrow \mathbf{y}^T\mathbf{x} \leq c \end{cases}$$

Proof. First we note that

$$\begin{aligned} &\exists \mathbf{t} \geq \mathbf{0},\ \mathbf{t}^T A = \mathbf{y},\ \mathbf{t}^T\mathbf{b} \leq c && \Leftrightarrow \\ &\exists \mathbf{t} \geq \mathbf{0},\ \mathbf{t}^T \begin{pmatrix} A \\ 0 \end{pmatrix} = \mathbf{y},\ \mathbf{t}^T \begin{pmatrix} \mathbf{b} \\ 1 \end{pmatrix} = c && \Leftrightarrow \\ &\exists \mathbf{t} \geq \mathbf{0},\ \mathbf{t}^T \begin{pmatrix} -\mathbf{b} & A \\ -1 & 0 \end{pmatrix} = (-c, \mathbf{y}) \end{aligned}$$

If we negate the right hand side of The Farkas Lemma, then we get that

$$\begin{aligned} &\exists \mathbf{t} \geq \mathbf{0},\ \mathbf{t}^T \begin{pmatrix} -\mathbf{b} & A \\ -1 & 0 \end{pmatrix} = (-c, \mathbf{y}) && \Leftrightarrow \\ &\forall \mathbf{x}, x_0 \begin{pmatrix} -\mathbf{b} & A \\ -1 & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ \mathbf{x} \end{pmatrix} \leq \mathbf{0} \Rightarrow \langle (-c, \mathbf{y}), (x_0, \mathbf{x}) \rangle \leq \mathbf{0} && \Leftrightarrow \\ &\forall x_0 \geq 0,\ A\mathbf{x} \leq x_0\mathbf{b} \ \Rightarrow\ \mathbf{y}^T\mathbf{x} \leq x_0 \cdot c \end{aligned}$$

Taking the case that $x_0 = 0$ and $x_0 > 0$ separately, you get the proposition. □

Proposition 4.4.1. *Suppose that $(A, \mathbf{b})$ is not minimal for some $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$. Then there is a row (A_i, b_i) for which the following holds. Let $(A', \mathbf{b}')$ be $(A, \mathbf{b})$ with the i-th row deleted. Then there is a $\mathbf{t}' \geq \mathbf{0}$ such that $\mathbf{t}'^T A' = A_i$, $\mathbf{t}'^T\mathbf{b}' \leq b_i$.*

Proof. Since $(A, \mathbf{b})$ is not minimal, there is such an (A_i, b_i) and $(A', \mathbf{b}')$ such that $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\} = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{b}'\}$. In the right hand side of Farkas Lemma 2, the conditions are satisfied with $A := A'$, $\mathbf{b} := \mathbf{b}'$, $\mathbf{y} := A_i$, $c := b_i$. □

Is the converse true? Does it hold that given a $(A, \mathbf{b})$ which is minimal, the implication of Proposition 4.4.1 fails? In general, no. For example, the hyperplane $\langle \mathbf{y}, \mathbf{x} \rangle = c$ has a minimal representation $\{\langle \mathbf{y}, \mathbf{x} \rangle \leq c,\ \langle -\mathbf{y}, \mathbf{x} \rangle \leq -c\}$, but the sum of the rows is 0, and so $\mathbf{t}' := (2, 1)$ satsifes the claim. In general, we need all $d+1$ rows to be *affinely independent*.

Definition 4.4.2 (Affine Dependence)**.** A set V of vectors is called *affinely independent* if, given any $\mathbf{y} \neq \mathbf{0}$, $c \in \mathbb{R}$, there is some $\mathbf{v} \in V$ such that $\mathbf{y}^T\mathbf{v} \neq c$.

Definition 4.4.3 (Full-Dimensional)**.** A polyhedron is called *full-dimensional* if it contains an affinely independent set of vectors.

Proposition 4.4.2. *If $P = \{\mathbf{x} \mid A\mathbf{x} \le \mathbf{b}\}$ is full dimensional, and $\mathbf{y}^T A = \mathbf{0}$ with $\mathbf{y} \ge \mathbf{0}$, then either $\mathbf{y} = \mathbf{0}$ or $\mathbf{y}^T\mathbf{b} > \mathbf{0}$.*

Proof. Say $\mathbf{y}^T A = \mathbf{0}$, $\mathbf{y} \ne \mathbf{0}$, and $\mathbf{y}^T\mathbf{b} = 0$. Then suppose $y_i \ne 0$. Then

$$y_i A_i + \sum_{j \ne i} y_j A_j = \mathbf{0}, \quad y_i b_i + \sum_{j \ne i} y_j a_j = 0$$

So, there are non-negative $\mathbf{t}_1, \mathbf{t}_2$ such that

$$\mathbf{t}_1^T A = -\mathbf{t}_2^T A, \quad \mathbf{t}_1^T\mathbf{b} = -\mathbf{t}_2^T\mathbf{b}$$

It follows that, for any $\mathbf{x}$ satisfying $A\mathbf{x} \le \mathbf{b}$:

$$\mathbf{t}_1^T A\mathbf{x} \le \mathbf{t}_1^T\mathbf{b}, \quad -\mathbf{t}_2^T A\mathbf{x} \ge -\mathbf{t}_2^T\mathbf{b}$$

But then for any $\mathbf{x} \in P$ it must hold that $\mathbf{t}_1^T A\mathbf{x} = \mathbf{t}_1^T\mathbf{b}$. Since P is full dimensional, it must be that $\mathbf{t}_1 = \mathbf{0}$, then $\mathbf{y} = \mathbf{0}$. □

Proposition 4.4.3. *Suppose that $(A, \mathbf{b})$ is minimal for some full-dimensional $\{\mathbf{x} \mid A\mathbf{x} \le \mathbf{b}\}$. Then let (A_i, b_i) be any row. If, for some $\mathbf{t} \ge \mathbf{0}$ $\mathbf{t}^T A = A_i$, then either $\mathbf{t} = \mathbf{e}_i$ or $\mathbf{t}^T\mathbf{b} > b_i$.*

Proof. Suppose that $A_i = t_i A_i + \mathbf{t}'^T A'$, and $b_i = t_i b_i + \mathbf{t}'\mathbf{b}'$. If $0 \le t_i < 1$, then $A_i = \mathbf{t}'^T A'/(1 - t_i)$, and $b_i = \mathbf{t}'^T\mathbf{b}'/(1 - t_i)$. But then $(A, \mathbf{b})$ is not minimal (the i-th row may be deleted without changing the polyhedron). Say $1 \le t_i$. Then there is a non-negative $\mathbf{t}''$ with $\mathbf{t}''^T A = 0$, $\mathbf{t}''^T\mathbf{b} = 0$. By Proposition 4.4.2, $\mathbf{t}'' = \mathbf{0}$, and $\mathbf{t} = \mathbf{e}_i$. □

Now we intend to use these properties of full-dimensionality and minimality to let us reduce the problem to one of cones.

Proposition 4.4.4. *The following statements are equivalent:*

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

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:

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

Observe that, by the way these sets are constructed, $x_0 \ge 0$. If $x_0 = 0$, then we have $\{\mathbf{x} \mid A\mathbf{x} \le \mathbf{0}\} \ne \{\mathbf{x} \mid A'\mathbf{x} \le \mathbf{0}\}$, which, by Proposition 4.3.3 means that that A and A' don't create the same characterstic cone, so $\{\mathbf{x} \mid A\mathbf{x} \le \mathbf{b}\} \ne \{\mathbf{x} \mid A'\mathbf{x} \le \mathbf{b}'\}$. If $x_0 > 0$, then we have:

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

So $\{\mathbf{x} \mid A\mathbf{x} \le \mathbf{b}\} \ne \{\mathbf{x} \mid A'\mathbf{x} \le \mathbf{b}'\}$. □

Now, combining the results of 'Characterstic Cone' on page 40 and proposition 4.4.4, we have the following result:

Proposition 4.4.5. *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. $\operatorname{cone}\begin{pmatrix} -\mathbf{b}^T & -1 \\ A^T & \mathbf{0} \end{pmatrix} = \operatorname{cone}\begin{pmatrix} -\mathbf{b}'^T & -1 \\ A'^T & \mathbf{0} \end{pmatrix}$

Proposition 4.4.6. *If* $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$ *is minimal and full-dimensional, then either*

1. $\operatorname{cone}\begin{pmatrix} -\mathbf{b}^T & -1 \\ A^T & \mathbf{0} \end{pmatrix}$ *is minimal and pointed, or*

2. $\operatorname{cone}\begin{pmatrix} -\mathbf{b}^T \\ A^T \end{pmatrix}$ *is minimal and pointed, and* $\operatorname{cone}\begin{pmatrix} -\mathbf{b}^T \\ A^T \end{pmatrix} = \operatorname{cone}\begin{pmatrix} -\mathbf{b}^T & -1 \\ A^T & \mathbf{0} \end{pmatrix}$

Proof. To see that they are pointed, let

$$\operatorname{cone}\begin{pmatrix} -\mathbf{b}^T & -1 \\ A^T & \mathbf{0} \end{pmatrix}(\mathbf{t}, t_0) = \mathbf{0}$$

If $(\mathbf{t}, t_0)$ is non-zero, then $\mathbf{t}^T A = \mathbf{0}$, which by Proposition 4.4.2 means that $-\mathbf{t}^T\mathbf{b} < 0$. This means that $-\mathbf{t}^T\mathbf{b} - t_0 < 0$. So $(\mathbf{t}, t_0) = \mathbf{0}$.

To show that it they are minimal, by 'Minimal V-Cone Generators' on page 32, we only need to show that

$$\mathbf{t} \geq \mathbf{0},\ [(A_i = \mathbf{t}^T A,\ \mathbf{t}^T\mathbf{b} + t_0 = b_i) \Rightarrow \mathbf{t} = \mathbf{e}_i]$$

By Proposition 4.4.3, if $\mathbf{t} \neq \mathbf{e}_i$, and $\mathbf{t}^T A = A_i$, then $\mathbf{t}^T\mathbf{b} > b_i$, so $\mathbf{t}^T\mathbf{b} + t_0 > b_i$. So that means that if $\mathbf{t}^T A = A_i$, $\mathbf{t}^T\mathbf{b} = b_i$, then $\mathbf{t}$ mush be $\mathbf{e}_i$. If there is a $\mathbf{t} \neq \mathbf{0}$ such that $\mathbf{t}^T A = \mathbf{0}$, then it follows that $-\mathbf{t}^T\mathbf{b} < 0$, and that $\operatorname{cone}\begin{pmatrix} -\mathbf{b}^T \\ A^T \end{pmatrix} = \begin{pmatrix} -\mathbf{b}^T & -1 \\ A^T & \mathbf{0} \end{pmatrix}$. Then $\begin{pmatrix} -\mathbf{b}^T \\ A^T \end{pmatrix}$ is minimal. If there is no such $\mathbf{t}$, that is, if $\mathbf{t}^T A = 0$ then $\mathbf{t} = \mathbf{0}$, then $\begin{pmatrix} -\mathbf{b}^T & -1 \\ A^T & \mathbf{0} \end{pmatrix}$ is minimal. □

Proposition 4.4.7. *Say* $(A, \mathbf{b})$ *is minimal for full-dimensional* $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$. *Then if* $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\} = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{b}'\}$, *then* $(A, \mathbf{b}) \sqsubseteq (A', \mathbf{b}')$.

Proof. Proposition 4.4.5 shows that the equality of the H-Cones is predicated on the equality of the V-Cones $\operatorname{cone}\begin{pmatrix} -\mathbf{b} \\ A \end{pmatrix}$ and $\operatorname{cone}\begin{pmatrix} -\mathbf{b}' \\ A' \end{pmatrix}$. Proposition 4.4.6 shows that we end up with the situation that either $\operatorname{cone}\begin{pmatrix} -\mathbf{b} \\ A \end{pmatrix} = \operatorname{cone}\begin{pmatrix} -\mathbf{b}' \\ A' \end{pmatrix}$ with $\operatorname{cone}\begin{pmatrix} -\mathbf{b} \\ A \end{pmatrix}$ minimal, in which case $\begin{pmatrix} -\mathbf{b} \\ A \end{pmatrix} \sqsubseteq \begin{pmatrix} -\mathbf{b}' \\ A' \end{pmatrix}$, or $\operatorname{cone}\begin{pmatrix} -\mathbf{b} & -1 \\ A & 0 \end{pmatrix}$ is minimal. In this case, since for no $A_i \in A$ is $A_i \simeq \mathbf{0}$, we conclude that $\begin{pmatrix} -\mathbf{b} \\ A \end{pmatrix} \sqsubseteq \begin{pmatrix} -\mathbf{b}' \\ A' \end{pmatrix}$ □

Say we know that $P = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\} = \text{cone}(U) + \text{conv}(V)$, with $(A, \mathbf{b})$ minimal, and P full-dimensional. We have another pair $(A', \mathbf{b}')$, and let $P' = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{b}'\}$. We want to test if $P = P'$. We have the following:

$$
\begin{aligned}
&A'U \leq \mathbf{0} && \Rightarrow \text{cone}(U') \subseteq \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{0}\} \\
&A'V \leq \mathbf{b} && \Rightarrow \text{conv}(V') \subseteq P' \\
&(A, \mathbf{b}) \sqsubseteq (A', \mathbf{b}') && \Rightarrow P' \subseteq P \\
&P' = P && \Rightarrow A'U \leq \mathbf{0} \\
&P' = P && \Rightarrow A'V \leq \mathbf{b} \\
&P' = P && \Rightarrow (A, \mathbf{b}) \sqsubseteq (A', \mathbf{b}')
\end{aligned}
$$

The first two lines imply that $P \subseteq P'$, so the first three mean that $P = P'$. We then have the following equivalence criteria:

Equivalence Criteria 4. *Say $(A, \mathbf{b})$ is a minimal pair for $P = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\} = \text{cone}(U) + \text{conv}(V)$, and suppose $P' = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{b}'\}$. Then*

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

Test 4 (V-Polyhedron $\to$ H-Polyhedron)**.** We now have a method for testing the program. First, we hand-craft a V-Polyhedron $\text{cone}(U) + \text{conv}(V)$ based on some minimal pair $(A, \mathbf{b})$, then run our program to get the pair $(A', \mathbf{b}')$, with the alleged property that $\text{cone}(U) + \text{conv}(V) = \{\mathbf{x} \mid A'\mathbf{x} \leq \mathbf{b}'\}$. If we confirm Equivalence Criteria 4, then our program has succeeded.

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 minimal 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 to be denoted $c \approx 0$.

```
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 to 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 to 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 to 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 to 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 to 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 minimal, 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 minimal, 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 minimal, 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 $P = \mathrm{cone}(U) + \mathrm{conv}(V) = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$ where A is minimal, 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);
}
```

Conclusion

In this thesis we have shown the proof of the Minkowski-Weyl Theorem. The proof is constructive in nature, showing how to create the sets promised by the theorem, and a basic implementation is given. While the proof is hardly elegant, it does show that the intuitive result of the theorem is true by brute force methods and does not require any advanced results. Indeed, the proof is from first principles, using the language of linear algebra.

The more interesting (and, perhaps, informative) part of the paper deals with creating a testing framework for the program. This presented an opportunity to discuss pointed and full-dimensional polyhedra, and their relation to minimal and essential representations of polyhedra. The characteristic and dual cones were defined with some useful properties proven, invoking the Farkas Lemma in a few different forms. As of now, the testing framework is open for extension, to include polyhedra for which minimal representations are not essentially unique.

It may be worth mentioning an anachronism presented in the text. The chapter on testing starts off by describing some sets that are chalked up to "wishful thinking," then minimality is introduced and the useful properties are derived. In earnest, the actual progression from wishful thinking to effective testing was a bit different. Initially, the requirements for testing were received. Then, the first idea was to create some cones, and check if one representation was a subset of another (modulo scaling, i.e. equivalence). This is the simplest, most immediate (and perhaps natural) answer to the challenge for testing. After this method was decided upon, the question then arose: "what properties of the sets representing the polyhedra are necessary to ensure the tests will work?" Then the properties were determined. It was only afterwards that it became clear that these properties were actually pointedness and full-dimensionality, at which point the presentation was altered to emphasize this. Without this alteration, the presentation would be akin to: "these seemingly arbitrary properties allow us to test the polyhedra in this manner," which is less pleasant to read than "these natural classes of polyhedra have useful properties which allow us to test our implementation on them."

It should also be mentioned that the algorithm here is not efficient. The intermediate representations of the polyhedra may be exponential in the size of the input and output. The "double description" method is a far better way to calculate the alternative representations desired, however the method is a bit more advanced and is better pursued after getting a decent grasp of the underlying problem.

The Farkas Lemma should have the last word, as it is a rather wonderful combinatorial compactification of much of the information of the Minkowski-Weyl Theorem. It's main contribution here was to show that minimal sets of H-Polyhedra do exist, and then allowed us to re-use some of the work we had done with V-Polyhedra to expedite the proofs of the testing methods.

Bibliography

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