

Computation and Manipulation of Enumerators of Integer Projections of Parametric Polytopes

Sven Verdoolaege *Kevin Woods*
Maurice Bruynooghe *Ronald Cools*

Report CW 392, March 2005



Katholieke Universiteit Leuven
Department of Computer Science

Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Computation and Manipulation of Enumerators of Integer Projections of Parametric Polytopes

Sven Verdoolaege *Kevin Woods*
Maurice Bruynooghe *Ronald Cools*

Report CW 392, March 2005

Department of Computer Science, K.U.Leuven

Abstract

Given a set of integer vectors defined by linear inequalities over a fixed number of variables, where some of the variables are considered as parameters, we consider two different ways of representing the number of elements in the set in terms of the parameters. The first is an explicit function which generalizes Ehrhart quasi-polynomials. The second is its corresponding generating function and generalizes the classical Ehrhart series. Both can be computed in polynomial time based on Barvinok's unimodular decomposition of cones. Furthermore, we can convert between the two representations in polynomial time.

In an extended problem, some of the variables that appear in the constraints may be existentially quantified and then the enumerated set corresponds to the projection of the integer points in a parametric polytope. We refer to this problem as the enumeration of the integer projection of a parametric polytope. This report shows how the enumeration of the integer projection of parametric polytopes can be reduced to the enumeration of parametric polytopes. Two approaches are described and experimentally compared. Differences are small, but both methods can solve problems that previously were considered very difficult to solve analytically.

Contents

Contents	1
List of Figures	2
List of Listings	3
List of Algorithms	3
List of Tables	3
1 Introduction	4
2 Preliminaries	5
2.1 Polyhedral Sets	5
2.2 Parametric Sets and their Enumerators	7
2.3 Generating Functions	8
2.4 Time Complexity	8
3 Parametric Counting Problems	9
3.1 Ehrhart Quasi-Polynomials	9
3.2 Vector Partition Functions	12
3.3 Parametric Polytopes	14
3.4 Parametric Projected Sets	22
4 Two Representations	23
5 Barvinok's Algorithm	25
5.1 Overview	25
5.2 Computing Generating Functions	27
5.2.1 Unimodular Cones	28
5.2.2 Brion's Theorem	29
5.2.3 Barvinok's Decomposition	31
5.2.4 Triangulation of Non-simplicial Cones	32
5.2.5 Decomposition of Simplicial Cones	36
5.2.6 Overview	39
5.3 Evaluating Generating Functions	43
5.4 Enumeration of Parametric Polytopes	47
6 Operations	52
6.1 Addition	52
6.2 Multiplication	54
6.3 Set Operations	57
6.4 Summation	57
6.5 Conversion	60
6.6 Evaluation	63
7 Projection	63
7.1 Shift and Subtract	64
7.2 Elimination	64
7.2.1 Unique Existential Variables	65
7.2.2 Redundant Existential Variables	66
7.2.3 Independent Splits	67
7.2.4 Overview	68

7.3	Parametric Integer Programming	68
7.4	Generating Functions	69
7.5	Line Removal	70
8	Optimizations	70
8.1	One-dimensional Polytopes	71
8.2	Simplification of Step-polynomials	72
9	Related Work	75
9.1	Pugh’s method	75
9.2	Clauss’s method	76
9.2.1	Interpolation and Degenerate Domains	76
9.2.2	Large Solution Size	77
9.2.3	Comparison	78
9.3	Other Techniques	78
10	Applications and Experiments	79
11	Conclusions and Future Work	81
A	Internal Representation	82
A.1	Existing Data Structures	82
A.2	Data Structures for Quasi-polynomials	83
A.3	Operations on Quasi-polynomials	85
A.4	Generating Functions	86
A.5	Counting Functions	87
A.6	Auxiliary Functions	88
B	Usage of the barvinok library	89
	References	92
	List of Symbols	97
	List of Acronyms	98
	Index	99

List of Figures

1	Two polyhedral complexes, (a) and (b), and two collections of polyhedra that are not polyhedral complexes, (c) and (d).	6
2	Dilations of the polytope $P = [0, \frac{1}{2}]$	10
3	Magic square.	11
4	The six cones defining the chamber decomposition of Example 10.	13
5	The chamber decomposition of Example 10.	13
6	Simple example program.	14
7	The number of points in P_4	14
8	More complicated example program.	15
9	Chamber decomposition of Example 12.	15
10	Cell and chamber decomposition.	18
11	The chamber decomposition and parametric vertices of the parametric polytope in Example 16.	20
12	Example Program.	23
13	Array elements accessed for $p = 3$	23

14	The set S from Example 20.	27
15	Barvinok Example. For each integer point (i, j) in the polytope T , there is a term $x_1^i x_2^j$ in the generating function $f(T; \mathbf{x})$	27
16	Supporting cone $\text{cone}(T, (0, 2))$ of polytope T at vertex $(0, 2)$	29
17	Intuitive explanation of Brion's theorem.	30
18	$P_{(3,4)}$ and its supporting cones.	31
19	A cone K and its polar K^*	32
20	The polytope P from Example 26 in thick lines and the supporting cone at the origin $\text{cone}(P, o)$ in dashed lines.	34
21	Slices of the cones C^λ and $C^{\lambda'}$ from Example 26 at $x = -1$ and the projections of their lower envelopes onto the $t = 0$ plane.	35
22	Slices of the cones C_\uparrow^λ and $C_\uparrow^{\lambda'}$ from Example 26 at $x = -1$ and the projections of their lower envelopes onto the $t = 0$ plane.	35
23	The triangulation of the supporting cone at the origin $\text{cone}(P, o)$ of the polytope P from Example 26.	36
24	Possible locations of \mathbf{w} with respect to the rays of a 3-dimensional cone.	37
25	Primal Unimodular Decomposition.	41
26	Dual Unimodular Decomposition.	42
27	The enumerator of $P_{\mathbf{p}}$, a step-polynomial in each chamber.	47
28	One-dimensional Example.	50
29	Intersection sets $A_1 \cap \mathbb{Q}_{\geq 0}^2$ and $A_2 \cap \mathbb{Q}_{\geq 0}^2$ for the alternative way in Example 32	52
30	Common refinement of chamber complexes with different outer walls.	54
31	Dual Unimodular Decomposition for the cone in Example 34.	57
32	Barvinok example indicator decomposition.	63
33	The set Q_4 from Example 40.	64
34	Decomposition of the set from Example 44.	67
35	Example of an answer generated by Pugh's method.	75
36	Geometrical representation of the chambers of Equation (4).	76
37	Matrix multiplication.	77
38	The quasi-polynomial $[1, 2]_p p^2 + 3p + \frac{5}{2}$	83
39	The quasi-polynomial $(1 + 2 \{\frac{p}{2}\}) p^2 + 3p + \frac{5}{2}$	84
40	Internal representation of $(\frac{3}{2} x_0^2 x_1^3 + 2 x_0^5 x_1^{-7}) / ((1 - x_0 x_1^{-3})(1 - x_1^2))$	87

List of Listings

1	Artificial pointer conversion example.	58
---	--	----

List of Algorithms

1	Barvinok's algorithm	40
2	Enumeration of sets with existential variables.	68

List of Tables

1	The contribution of each supporting cone to the constant term of the Laurent expansion of $f(P_{\mathbf{p}}, (t + 1, t + 1))$ about $t = 1$	46
2	The contribution of each supporting cone to the constant term of the Laurent expansion of $f(P_{\mathbf{p}}, (t + 1, u + 1))$ about $t = 1$ and $u = 1$	47
3	Construction of the piecewise step-polynomial from Example 32.	50
4	Comparison between the method of Clauss and Loechner (1998) and the method of Section 5.	78
5	Rule application distribution.	79

1 Introduction

Many compiler optimization techniques require the enumeration of certain kinds of objects as a substep, e.g., the number of times a statement is executed, the number of times a variable is accessed or the total number of accessed array elements. These counting problems can typically be formulated as the number of integer points that satisfy certain linear constraints. The general form of this type of problems is

$$c_S(\mathbf{p}) = \#S_{\mathbf{p}} = \# \left\{ \mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{y} \in \mathbb{Z}^{d'} : \mathbf{A}\mathbf{x} + \mathbf{D}\mathbf{y} + \mathbf{B}\mathbf{p} + \mathbf{c} \geq \mathbf{0} \right\},$$

where $\mathbf{x} \in \mathbb{Z}^d$ represent the objects to be counted, $\mathbf{p} \in \mathbb{Z}^n$ are parameters on which the count depends and $\mathbf{y} \in \mathbb{Z}^{d'}$ are extra existentially quantified variables that are used to further constrain the number of objects. If $d' = 0$, then S is the set of integer points in what is known as a rational parametric polytope. For $d' \neq 0$, S is the projection onto a lower-dimensional space of the integer points in such a parametric polytope. A technique for solving problems without existential variables ($d' = 0$) was proposed by Clauss and Loechner (1998) and implemented in `PolyLib`. The technique is worst-case exponential however, even for fixed dimension, and the current implementation even fails to produce a solution on some problems. Recent automata-based counting techniques (Boigelot and Latour 2004; Parker and Chatterjee 2004) solve problems without parameters ($n = 0$) but are also exponential. The technique of Pugh (1994) is intended to solve the general problem, but no implementation has been reported and the technique is also exponential.

Barvinok (1993) described a polynomial-time algorithm (for fixed dimension d) for computing the number of integer points in a fixed polytope ($n = 0$ and $d' = 0$), which was generalized by Barvinok and Pommersheim (1999) to parametric polytopes ($d' = 0$). Finally, Barvinok and Woods (2003) further generalized the technique to a large subclass of the problem class sketched above by considering the projection operation, but only to compute what is known as the generating function of $c_S(\mathbf{p})$ rather than the function itself. De Loera et al. (2004) were the first to implement the technique of Barvinok (1993) for enumerating fixed polytopes. De Loera et al. (2003a) extended the implementation to also compute the generating function of $c_S(\mathbf{p})$ for some specific subproblems.

In this report we present our implementation of the `barvinok` library which applies the algorithm of Barvinok and Pommersheim (1999), with some refinements inspired by the works of Clauss and Loechner (1998) and De Loera et al. (2004), to compute both an explicit representation of $c_S(\mathbf{p})$ or its generating function for general parametric polytopes ($d' = 0$) in polynomial time (for fixed dimension). We further describe some other operations that may be performed on both types of representations, as well as two techniques that handle more general enumeration problems. One of these techniques is polynomial but of limited scope and the other is general but worst-case exponential.

On a more theoretical level, we show that the explicit function representation and the generating function representation are interconvertible in polynomial time. Combined with the results of Barvinok and Woods (2003), this conversion results in a polynomial time algorithm for computing an explicit representation of $c_S(\mathbf{p})$ for the same subclass considered by Barvinok and Woods (2003). Our library currently does not implement this algorithm, however.

Section 2 introduces background information on polytopes and related sets; generating functions and time complexity. Section 3 discusses parametric counting problems and their relation, whereas Section 4 details the representation of both explicit functions and generating functions that we will use. The application of Barvinok’s algorithm to parametric polytopes is discussed in Section 5. Various operations are listed in Section 6, and projection is explained in some more detail in Section 7. Some optimization are performed in Section 8, while applications and related work are discussed in Sections 10 and 9. Concluding remarks follow in Section 11.

2 Preliminaries

In this section we introduce some concepts that we will need throughout this report. We start with polyhedral sets, i.e., polyhedra, special types of polyhedra, collections of polyhedra and projections of polyhedra. Then we define the enumerator of a set defined in terms of parameters. We touch upon the relation between a function on integer vectors and the corresponding generating function and we finish with a note on time complexity.

2.1 Polyhedral Sets

Definition 2.1 (Rational polyhedron) A rational polyhedron P is a subspace of \mathbb{Q}^d bounded by a finite number of hyperplanes.

$$P = \{ \mathbf{x} \in \mathbb{Q}^d \mid A\mathbf{x} \geq \mathbf{c} \}, \quad (1)$$

with $A \in \mathbb{Z}^{m \times d}$ and $\mathbf{c} \in \mathbb{Z}^m$.

Definition 2.2 (Affine hull) The affine hull of a set $X \subset \mathbb{Q}^d$ is the set

$$\text{aff } X = \left\{ \sum_i \lambda_i \mathbf{x}_i \mid \mathbf{x}_i \in X, \sum_i \lambda_i = 1 \right\}.$$

Definition 2.3 The dimension of a rational polyhedron $P \subset \mathbb{Q}^d$ is the dimension of its affine hull. Equivalently, it is equal to the dimension d of the ambient space \mathbb{Q}^d minus the number of linearly independent (implicit) equalities in the system $A\mathbf{x} \geq \mathbf{c}$.

Definition 2.4 A face F of a rational polyhedron P (1) is the intersection of P with $\{ \mathbf{x} \in \mathbb{Q}^d \mid A'\mathbf{x} = \mathbf{c}' \}$, where $A'\mathbf{x} \geq \mathbf{c}'$ is a subsystem of $A\mathbf{x} \geq \mathbf{c}$. If P has dimension n , then the $(n-1)$ -dimensional faces are called facets. The 0-dimensional faces are called vertices.

By convention, the empty set \emptyset is a (-1) -dimensional face of every polyhedron. Note that every vertex \mathbf{v} of P is an extremal point of P , i.e., \mathbf{v} is a point that cannot be expressed as a convex combination of other points in P .

Definition 2.5 (Rational polytope) A rational polytope is a bounded rational polyhedron.

Example 1 The interval $[0, 1]$ is a rational polytope:

$$[0, 1] = \{ i \mid i \geq 0 \wedge i \leq 1 \} = \{ \lambda_0 0 + \lambda_1 1 \mid \lambda_0, \lambda_1 \geq 0, \lambda_0 + \lambda_1 = 1 \}.$$

The bounding hyperplanes are the points 0 and 1.

Definition 2.6 (Ray) A ray of a set \mathcal{K} is a vector $\mathbf{r} \neq \mathbf{0}$ such that $\mathbf{x} \in \mathcal{K}$ implies $(\mathbf{x} + \mu\mathbf{r}) \in \mathcal{K}$ for all $\mu \geq 0$.

Definition 2.7 (Line) A line of a set \mathcal{K} is a vector $\mathbf{l} \neq \mathbf{0}$ such that $\mathbf{x} \in \mathcal{K}$ implies $(\mathbf{x} + \mu\mathbf{l}) \in \mathcal{K}$ for all μ .

Definition 2.8 (Positive hull) The positive hull of a set X is the set of all positive combinations of elements from X :

$$\text{pos } X = \left\{ \sum_i \lambda_i \mathbf{x}_i \mid \mathbf{x}_i \in X, \lambda_i \geq 0 \right\}.$$

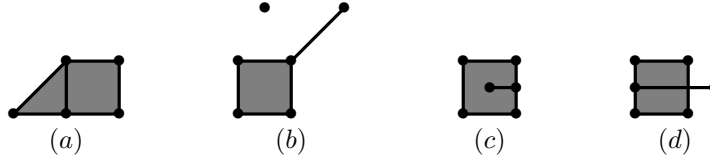


Figure 1: Two polyhedral complexes, (a) and (b), and two collections of polyhedra that are not polyhedral complexes, (c) and (d).

Definition 2.9 (Polyhedral cone) A polyhedral cone, or simply cone, is the positive hull of a set of elements, called its generators.

Definition 2.10 (Polyhedral complex) A polyhedral complex is a family of polyhedra such that each face of a member of the complex is also a member of the complex and such that any two members of the family intersect in a common, possibly empty, face. The support $|K|$ of a polyhedral complex K is the union of the polyhedra in K .

Example 2 Figure 1 shows four families of polyhedra. The first family (a) consists of a square and a triangle that share an edge as well as all edges and all vertices of the square and the triangle. The second family (b) consists of a square, its edges and vertices; a line segment connecting one of the vertices to another member point; and an isolated point. The third family (c) consists of the same square with its edges and vertices together with an additional line segment and its vertices. The fourth family (d) is similar to the second, except that the line segment intersects two edges and only one of these intersections is a member of the family. Only the first two families, (a) and (b) are polyhedral complexes. In the third (c), the intersection of the square and the line segment (i.e., the line segment itself) is not a face of the square. In the fourth (d), the intersections of the line segment with both the right edge of the square and the square itself are not part of the collection.

We will usually only be interested in the members of a complex of maximal dimension and we will therefore not make a strong distinction between these members of maximal dimension and the whole complex.

Definition 2.11 (Refinement) If K and K' are polyhedral complexes such that $|K| \subset |K'|$ and such that for every $R \in K$ there is a $R' \in K'$ with $R \supset R'$, then K' is a refinement of K . The common refinement of a family of polyhedra is a polyhedral complex such that its support is the union of the polyhedra and such that each member is the intersection of some of the polyhedra.

Example 3 Figure 5 on page 13 shows the common refinement of the family of cones in Figure 4.

Definition 2.12 (Subdivision) A subdivision of a polyhedron P is a polyhedral complex K such that $|K| = P$. An element of a subdivision of maximal dimension is called a cell.

Example 4 The first complex in Figure 1 is a subdivision containing two cells. The second complex is not a subdivision since its support is not a polyhedron. Figure 9 on page 15 shows a subdivision with 16 members: the whole polyhedron, the empty set, four 2-dimensional cells (three of which are infinite), three outer walls, four pairwise intersections of 2-dimensional cells and three vertices.

Definition 2.13 (Simplicial cone) A d -dimensional cone is simplicial if it is generated by d (linearly independent) generators.

Note that a 1- or 2-dimensional cone is always simplicial. Figure 20 on page 34 shows a 3-dimensional cone that is *not* simplicial since it is generated by the four points a , b , c and d .

Definition 2.14 (Triangulation) A triangulation of a cone C is a subdivision of C such that each element is a simplicial cone.

Note that triangulations are typically defined on polytopes rather than cones, but we will only need it for cones. Also note that some authors, e.g., Fortune (1992), call a subdivision a triangulation and a triangulation a proper triangulation.

Definition 2.15 (Projected set) A projected set S is a set of the form

$$S = \left\{ \mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{y} \in \mathbb{Z}^{d'} : A\mathbf{x} + B\mathbf{y} \geq \mathbf{c} \right\},$$

for some $A \in \mathbb{Z}^{m \times d}$, $B \in \mathbb{Z}^{m \times d'}$ and $\mathbf{c} \in \mathbb{Z}^m$.

Such projected sets have also been called integer projections of polyhedra (Pugh 1994). The “projection” refers to the fact that the set S in the above definition can be written as

$$S = \pi_d(\mathbb{Z}^{d+d'} \cap P),$$

with

$$P = \left\{ (\mathbf{x}, \mathbf{y}) \in \mathbb{Q}^{d+d'} \mid A\mathbf{x} + B\mathbf{y} \geq \mathbf{c} \right\}$$

and π_d the projection onto the first d dimensions. We call P the polyhedron defining S .

Definition 2.16 (Presburger set) A Presburger set is a set that can be described by a Presburger formula, which is a formula that consists of linear inequalities of integer variables, combined by existential and universal quantifiers, disjunction, conjunction and negation ($\exists, \forall, \vee, \wedge, \neg$).

Each Presburger set can be written as a union of projected sets, but the conversion can in general be very exponential (even for fixed dimension), which is why we make the distinction. The `Omega` library can be used to perform this conversion.

2.2 Parametric Sets and their Enumerators

In some of our sets, some of the variables, called the *parameters*, will be treated differently from the other variables. These parameters are used to create *parametric sets*, which represent collections of sets parametrized by the parameters. Such a parametric set can be modeled as a function $f : \mathbb{Z}^n \rightarrow 2^{\mathbb{Z}^d}$ from the parameter space to the set of polyhedra or projected sets or alternatively as a relation between the parameters and the elements of these sets, i.e., $S \subset \mathbb{Z}^n \times \mathbb{Z}^d$. We choose the latter alternative. We let $S_{\mathbf{p}} = \{ \mathbf{x} \in \mathbb{Z}^d \mid (\mathbf{p}, \mathbf{x}) \in S \} \subset \mathbb{Z}^d$. In particular, a *parametric polytope* is modeled by a polyhedron $P \subset \mathbb{Q}^n \times \mathbb{Q}^d$ such that for all $\mathbf{p} \in \mathbb{Q}^n$, the set $P_{\mathbf{p}}$ is a polytope.

Definition 2.17 (Enumerator) The enumerator c_S of a parametric set S is a function from the set of n -dimensional integer vectors \mathbb{Z}^n to the set of natural numbers \mathbb{N} . The function value at \mathbf{p} , denoted $c_S(\mathbf{p})$, is the number of integer points in the set $S_{\mathbf{p}}$.

$$\begin{aligned} c_S & : \mathbb{Z}^n & \rightarrow & \mathbb{N} \\ & \mathbf{p} & \mapsto & c_S(\mathbf{p}) = \#S_{\mathbf{p}} \end{aligned} \tag{2}$$

If $P \subset \mathbb{Q}^n \times \mathbb{Q}^d$ defines a parametric polytope, then we will also say that $c_P = \#(P_{\mathbf{p}} \cap \mathbb{Z}^d)$ is the enumerator of $P_{\mathbf{p}}$.

Note that Barvinok and Pommersheim (1999) call the parametric polytopes $P_{\mathbf{p}}$ the *fibers* of the projection of the polyhedron P onto the parameter space. Other authors denote by this term the sets $\{\mathbf{p}\} \times P_{\mathbf{p}}$. We will not use this terminology.

2.3 Generating Functions

Definition 2.18 (Generating function of a sequence) Let $\{a_i\}_{i=0}^{\infty}$ be a sequence of rational numbers $a_i \in \mathbb{Q}$, then the generating function of $\{a_i\}_{i=0}^{\infty}$ is a formal power series $A(x) \in \mathbb{Q}[[x]]$ with the a_i as coefficients, i.e.,

$$A(x) = \sum_{i=0}^{\infty} a_i x^i.$$

Such generating functions can naturally be extended to *multiple sequences* $\{a_{\mathbf{i}}\}_{\mathbf{i} \in \mathbb{N}^n}$. The corresponding generating functions are called *multiple generating functions* (Srivastava and Manocha 1984). Note that a (multiple) sequence is basically a (multivariate) function $a : \mathbb{N}^n \rightarrow \mathbb{Q}$.

The concept of a generating function can be further extended to correspond to (some) functions over integer vectors.

Definition 2.19 (Generating function) Let $a : \mathbb{Z}^n \rightarrow \mathbb{Q}$ be a function. The corresponding generating function, if it exists, is a function $A : \mathbb{C}^n \rightarrow \mathbb{C}$ such that A has a Laurent power series expansion

$$A(\mathbf{x}) = \sum_{\mathbf{s} \in \mathbb{Z}^n} a(\mathbf{s}) \mathbf{x}^{\mathbf{s}}$$

which converges on some non-empty open region of \mathbb{C}^n . We use the notation $\mathbf{x}^{\mathbf{s}} = x_1^{s_1} x_2^{s_2} \cdots x_n^{s_n}$.

Note that the convergence requirement means that not all generating functions from Definition 2.18 are also generating functions according to Definition 2.19. We must also be careful when speaking of a correspondence between a function and its generating function, because a generating function may have different Laurent power series expansions which converge on different regions of \mathbb{C}^n . For example, if $A(x) = \frac{1}{1-x}$ then

$$1 + x + x^2 + x^3 + \cdots \quad \text{and} \quad -x^{-1} - x^{-2} - x^{-3} - \cdots$$

are Laurent power series expansions convergent for $\|x\| < 1$ and $\|x\| > 1$, respectively. The first is the well-known geometric series, while the second follows from the identity

$$\frac{1}{1-x} = \frac{-x^{-1}}{1-x^{-1}},$$

or in general

$$\frac{1}{1-\mathbf{x}^{\mathbf{u}}} = \frac{-\mathbf{x}^{-\mathbf{u}}}{1-\mathbf{x}^{-\mathbf{u}}}. \quad (3)$$

That is, $A(x)$ is the generating function of both

$$a_1(s) = \begin{cases} 1 & \text{if } s \geq 0 \\ 0 & \text{if } s < 0 \end{cases} \quad \text{and} \quad a_2(s) = \begin{cases} 0 & \text{if } s \geq 0 \\ -1 & \text{if } s < 0, \end{cases}$$

but on different regions of \mathbb{C}^1 .

2.4 Time Complexity

Whenever we say that an algorithm has polynomial time complexity, we will mean that the time complexity is polynomial in the input size. This input size is the number of bits that is needed to represent the input (Schrijver 1986; Papadimitriou 1994). E.g., if the input is the implicit representation of a polyhedron $P \subset \mathbb{Q}^d$, i.e., a collection of m linear inequalities $A\mathbf{x} \geq \mathbf{c}$, then the input size is approximately

$$md + \sum_{i,j} \log_2 |a_{ij}| + \sum_i \log_2 |c_i|.$$

The algorithms will usually only be polynomial if we fix some size parameter, typically the dimension of the problem. E.g., if an algorithm has time-complexity $O(n^d)$, with n the input size and d the dimension of the problem then we will say that the computation time is polynomial in the input size, for fixed dimension d .

3 Parametric Counting Problems

Our main counting problem is the enumeration of parametric polytopes. We first consider two special cases, Ehrhart quasi-polynomials and vector partition functions. Then we consider the enumeration of parametric polytopes in some more detail, indicating the relation with vector partition functions and preparing for the application of Barvinok’s algorithm to parametric polytopes in Section 5. Finally, we briefly discuss the generalization to parametric projected sets.

3.1 Ehrhart Quasi-Polynomials

Consider a rational polytope $P \subset \mathbb{Q}^d$ and its dilations sP by a factor of $s \in \mathbb{N}$. Ehrhart (1962) showed that the number of integer points in sP as s varies is given by a special type of function called a *quasi-polynomial*. Many authors (e.g., Stanley 1986) use the following definition of a quasi-polynomial.

Definition 3.1 A function $f : \mathbb{Z} \rightarrow \mathbb{Q}$ is a (univariate) quasi-polynomial of period q if there exists a list of q polynomials $g_i \in \mathbb{Q}[T]$ for $0 \leq i < q$ such that

$$f(s) = g_i(s) \quad \text{if } s \equiv i \pmod{q}.$$

The functions g_i are called the constituents.

Ehrhart (1977) uses a different definition, based on *periodic numbers*.

Definition 3.2 A rational periodic number $U(p)$ is a function $\mathbb{Z} \rightarrow \mathbb{Q}$, such that there exists a period q such that $U(p) = U(p')$ whenever $p \equiv p' \pmod{q}$.

Definition 3.3 A (univariate) quasi-polynomial f of degree d is a function

$$f(n) = c_d(n) n^d + \cdots + c_1(n) n + c_0,$$

where $c_i(n)$ are rational periodic numbers. I.e., it is a polynomial expression of degree d with rational periodic numbers for coefficients. The period of a quasi-polynomial is the least common multiple (lcm) of the periods of its coefficients.

These two definition are easily seen to be equivalent. Ehrhart (1977) uses a list of q rational numbers enclosed in square brackets to represent periodic numbers.

Example 5 $U(p) = [1, 3/4]_p$ is a periodic number with period $q = 2$; $U(p) = 1$ if $p \pmod{2} \equiv 0$ and $U(p) = 3/4$ if $p \pmod{2} \equiv 1$. Furthermore,

$$\frac{1}{4}p^2 + p - \left[1, \frac{3}{4}\right]_p$$

is quasi-polynomial of degree 2 and period 2.

We can now formulate Ehrhart’s main theorem.

Theorem 3.4 Let $P \subset \mathbb{Q}^d$ be a rational polytope. The number of points in the dilations sP with $s \in \mathbb{N}$ is given by a degree- d quasi-polynomial. The period of the quasi-polynomial is a divisor of the lcm of the denominators of the vertices of P .

Note that Ehrhart actually considered more general problems, but the quasi-polynomials from Theorem 3.4 are the ones that are commonly known as *Ehrhart quasi-polynomials*. The corresponding generating function is called the *Ehrhart series*. If an Ehrhart quasi-polynomial has period 1, i.e., if it is an actual polynomial, then it is called an *Ehrhart polynomial*. In the compiler community, the term “Ehrhart polynomial” is also commonly used to refer to any enumerator of a parametric polytope. This terminology was introduced by Clauss and Loechner (1998). We will not use this term in this meaning here to avoid confusion with the established use of the term Ehrhart (quasi-)polynomial in the mathematical community (e.g., Stanley 1986; Diaz and Robins 1996; Beck et al. 2004; McAllister and Woods 2004; Miller and Sturmfels 2004; Woods 2004; Beck and Robins 2006).

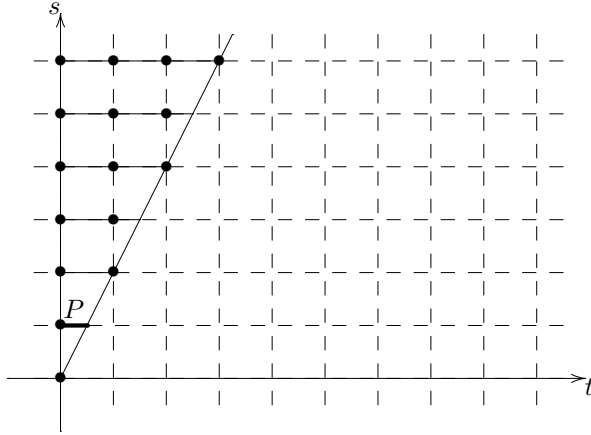


Figure 2: Dilations of the polytope $P = [0, \frac{1}{2}]$.

Example 6 Consider the polytope

$$P = \left[0, \frac{1}{2}\right] \in \mathbb{Q}^1,$$

shown in Figure 2. The number of integer points in P itself is 1. For dilations of the polytope sP , we have the general counting formula

$$c(s) = \left\lfloor \frac{s}{2} \right\rfloor + 1.$$

In Ehrhart's notation, this can be written as $c(s) = s/2 + [1, 1/2]_s$. Writing $c(s)$ as a list of polynomials $c_0(s)$, $c_1(s)$, we have $c_0(s) = s/2 + 1$ and $c_1(s) = s/2 + 1/2$. The corresponding Ehrhart series is

$$\begin{aligned} C(x) &= \frac{1}{(1-x)(1-x^2)} \\ &= (1+x+x^2+x^3+x^4+x^5\cdots)(1+x^2+x^4+x^6\cdots) \\ &= 1+x+2x^2+2x^3+3x^4+3x^5\cdots \end{aligned}$$

Example 7 As a slightly larger example, let $P \subset \mathbb{Q}^2$ be $[0, \frac{1}{2}] \times [0, \frac{1}{2}]$. Then

$$c_P(s) = \left\lfloor \frac{1}{2}s + 1 \right\rfloor^2, \text{ for } s \geq 0,$$

and we have that

$$C_P(x) = \sum_{s=0}^{\infty} c_P(s) x^s = \frac{2}{(1-x)(1-x^2)^2} - \frac{1}{(1-x)(1-x^2)}.$$

Example 8 As a more “practical” example, consider the problem of counting the number of magic squares (Yoshida 2004a; Beck and Robins 2006). An integer square matrix is magic if the sum of each row, the sum of each column and the sum of both diagonals are all equal to the same number s . Figure 3 shows a magic 4×4 square. Traditionally, the elements of a magic square are required to be distinct, but we will not consider this requirement here. The number of different magic squares with row, column and diagonal sums equal to a given

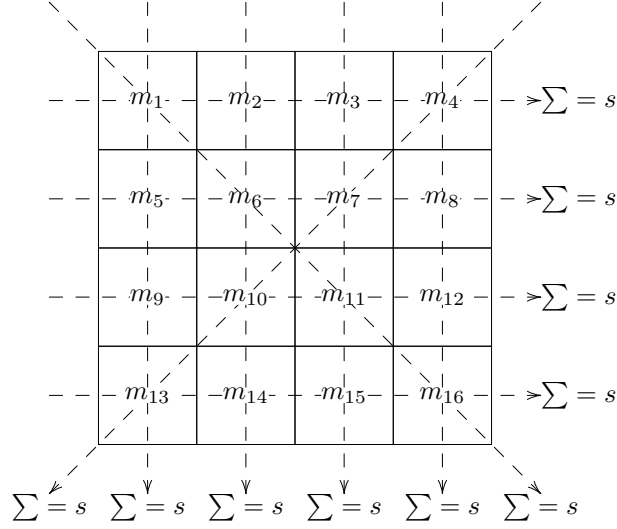


Figure 3: Magic square.

number s is equal to the number of nonnegative integers that satisfy the constraints

$$\begin{aligned}
 m_1 + m_2 + m_3 + m_4 &= s \\
 m_5 + m_6 + m_7 + m_8 &= s \\
 m_9 + m_{10} + m_{11} + m_{12} &= s \\
 m_{13} + m_{14} + m_{15} + m_{16} &= s \\
 m_1 + m_5 + m_9 + m_{13} &= s \\
 m_2 + m_6 + m_{10} + m_{14} &= s \\
 m_3 + m_7 + m_{11} + m_{15} &= s \\
 m_4 + m_8 + m_{12} + m_{16} &= s \\
 m_1 + m_6 + m_{11} + m_{16} &= s \\
 m_4 + m_7 + m_{10} + m_{13} &= s.
 \end{aligned}$$

Let $P \subset \mathbb{Q}^{16}$ be the polytope defined by $m_i \geq 0$ and the equations above with s replaced by 1. Then the number of magic squares with sum s is equal to the number of integer points in the dilation sP . Using our implementation based on the techniques from Section 5, we obtain this number as

$$\begin{aligned}
 \frac{1}{480}s^7 + \frac{7}{240}s^6 + \frac{89}{580}s^5 + \frac{65}{96}s^4 + \left(\frac{1}{48}\left\lfloor \frac{s}{2} \right\rfloor + \frac{377}{240}\right)s^3 + \\
 \left(\frac{1}{8}\left\lfloor \frac{s}{2} \right\rfloor + \frac{377}{160}\right)s^2 + \left(\frac{17}{48}\left\lfloor \frac{s}{2} \right\rfloor + \frac{523}{240}\right)s + \frac{3}{8}\left\lfloor \frac{s}{2} \right\rfloor + 1
 \end{aligned}$$

and the corresponding generating function as

$$\begin{aligned} & \frac{3x^2}{(1-x)^4(1-x^2)^2} + \frac{7x^4}{(1-x)^4(1-x^2)^3} + \frac{6x^3}{(1-x)^5(1-x^2)^2} + \\ & \frac{10x^5 + 4x^4 + 2x^3 + 2x^2}{(1-x)^5(1-x^2)^3} + \frac{5x^6 + 2x^4 + x^2}{(1-x)^4(1-x^2)^4} + \frac{4x^4 + 5x^3 + 4x^2}{(1-x)^6(1-x^2)^2} + \\ & \frac{\frac{105}{8}x + \frac{25}{8}}{(1-x)^5} + \frac{8x^3 + 9x^2}{(1-x)^7(1-x^2)} + \frac{\frac{469}{8}x^2 - 10x - \frac{157}{8}}{(1-x)^6} + \\ & \frac{92x^3 - \frac{107}{2}x^2 + \frac{11}{2}x + 34}{(1-x)^7} + \frac{\frac{93}{2}x^4 - \frac{101}{2}x^3 + 18x^2 - \frac{1}{2}x - \frac{35}{2}}{(1-x)^8} + \\ & \frac{1}{(1-x)^4} + \frac{-x^2}{(1-x)^6(1-x^2)}. \end{aligned}$$

The latter can be simplified to

$$\frac{x^8 + 4x^7 + 18x^6 + 36x^5 + 50x^4 + 36x^3 + 18x^2 + 4x + 1}{(1-x)^4(1-x^2)^4},$$

which can also be obtained using `LattE`, as reported by Yoshida (2004a, Appendix A.5.1). `LattE` cannot be used to obtain the explicit function, however. Rather, Yoshida (2004a) suggests to perform a partial Taylor expansion and to obtain an explicit function using interpolation.

3.2 Vector Partition Functions

In this section, we discuss a frequently occurring counting problem known as *vector partition functions*. Sturmfels (1995) mentions representation theory (Heckman 1982), commutative algebra (Stanley 1996), approximation theory (Dahmen and Micchelli 1988) and statistics (Diaconis and Gangolli 1995) as some of the areas in which vector partition functions occur. Vector partition functions are the natural extensions of the partition function $p(n)$, which is the number of ways a positive integer can be expressed as a sum of positive integers (see, e.g., Hardy and Wright 1979). As was shown by Euler (1770), the generating function of $p(n)$ is particularly easy,

$$1 + \sum_{n=1}^{\infty} p(n) t^n = \frac{1}{\prod_{n=1}^{\infty} (1 - t^n)}.$$

Definition 3.5 (Vector partition function) *Given $A \in \mathbb{N}^{n \times d}$ of rank n (in particular, $d \geq n$), the corresponding vector partition function $\phi_A : \mathbb{N}^n \rightarrow \mathbb{N}$ is such that $\phi_A(\mathbf{u})$ is the number of non-negative integer vectors $\boldsymbol{\lambda} \in \mathbb{N}^d$ such that $A\boldsymbol{\lambda} = \mathbf{u}$. Equivalently, it is defined as*

$$\frac{1}{\prod_{i=1}^d (1 - \mathbf{t}^{\mathbf{a}_i})} = \sum_{\mathbf{u} \in \mathbb{N}^n} \phi_A(\mathbf{u}) \mathbf{t}^{\mathbf{u}}$$

(with \mathbf{a}_i the columns of A).

Example 9 Let $c(s)$ be the vector partition function with $a_1 = 2$, $a_2 = 5$, i.e., $A = \begin{bmatrix} 2 & 5 \end{bmatrix}$. Then the generating function representation is

$$f(x) = \frac{1}{(1-x^2)(1-x^5)},$$

and it can be shown that the vector partition function itself can be written as

$$c(s) = \begin{cases} 0, & \text{if } s < 0 \\ \lfloor \frac{1}{2}s + 1 \rfloor + \lfloor -\frac{2}{5}s \rfloor, & \text{if } s \geq 0 \end{cases}.$$

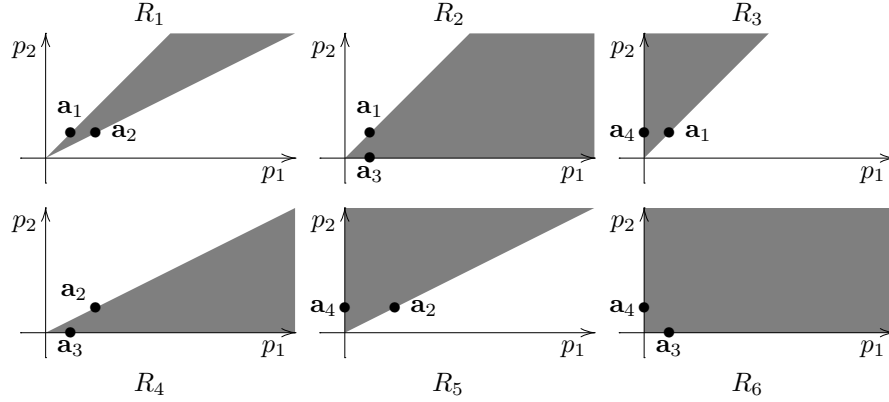


Figure 4: The six cones defining the chamber decomposition of Example 10.

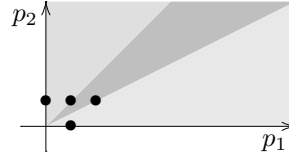


Figure 5: The chamber decomposition of Example 10. The different patterns mark the three full-dimensional chambers.

Sturmfels (1995) describes the general form of a vector partition function. It was shown previously by Blakley (1964) that there exists a finite decomposition of \mathbb{N}^n such that ϕ_A is a polynomial of degree $d - n$ on each piece. The description of Sturmfels (1995), however, is based on the geometric decomposition of \mathbb{N}^n into *chambers*, studied by Alekseevskaya et al. (1987). In each chamber, the vector partition function is the sum of a polynomial P of degree $d - n$ and a rational linear combination of some “corrector polynomials”.

In the remainder of this section, A is assumed surjective over \mathbb{Z} , i.e., $\mathbb{Z}A = \mathbb{Z}^n$. If A is not surjective, a $B \in \mathbb{Q}^{n \times n}$ needs to be chosen that defines an isomorphism from $\mathbb{Z}A$ onto \mathbb{Z}^n and then $\phi_A(\mathbf{s}) = \phi_{BA}(B\mathbf{s})$. We now define the decomposition into chambers.

Definition 3.6 (Chamber complex of a vector partition function) *The chamber complex of a vector partition function as defined in Definition 3.5 is the common refinement of all simplicial cones $\text{pos } A_\sigma$, i.e., the sets of nonnegative linear combinations of the columns of A_σ , where A_σ is the submatrix of A formed by the columns indexed by σ and $\sigma \subset \{1, \dots, d\}$ is such that $\#\sigma = \text{rank } A_\sigma = n$. A chamber C is a maximal cell in the chamber complex.*

Example 10 Consider the vector partition function corresponding to the generating function

$$C(\mathbf{x}) = \frac{1}{(1 - \mathbf{x}^{(1,1)})(1 - \mathbf{x}^{(2,1)})(1 - \mathbf{x}^{(1,0)})(1 - \mathbf{x}^{(0,1)}),}$$

i.e.,

$$c(\mathbf{p}) = \# \left\{ \boldsymbol{\lambda} \in \mathbb{N}^4 \mid \begin{bmatrix} 1 & 2 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \boldsymbol{\lambda} = \mathbf{p} \right\}.$$

The matrix A has 4 columns. Each 2×2 submatrix of A is of full rank. The chamber complex is therefore the common refinement of the six cones shown in Figure 4. For each of these cones, the columns of A that generate the cone are marked by a \bullet . The chamber decomposition is shown in Figure 5. It contains three full dimensional chambers, 4 one-dimensional chambers and the origin. As in the previous figure, the \bullet s refer to the columns of A .

```

for(i = 1; i <= N; i++)
  for(j = 1; j <= i; j++)
    S1;

```

Figure 6: Simple example program.

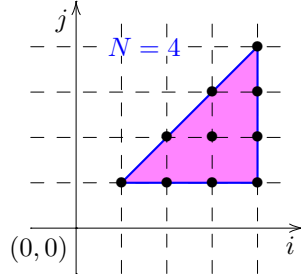


Figure 7: The number of points in P_4 .

Although we will not need it in our report, we mention the main result of Sturmfels (1995) on the shape of vector partition functions for completeness. For each chamber C there exists a polynomial P of degree $d - n$ in \mathbf{s} and for each $\sigma \in \Delta(C) := \{\sigma \subset \{1 \dots, n\} \mid C \subset \text{pos } A_\sigma\}$ there exists a polynomial Q_σ of degree $\#\sigma - n$ in \mathbf{s} and a function $\Omega_\sigma : \mathbb{Z}^n / \mathbb{Z}A_\sigma \rightarrow \mathbb{Q}$, with $\Omega_\sigma(0) = 0$, such that for all $\mathbf{s} \in C \cap \mathbb{N}^n$,

$$c(\mathbf{s}) = P(\mathbf{s}) + \sum_{\sigma \in \Delta(C_i)} \Omega_\sigma(\bar{\mathbf{s}}) Q_\sigma(\mathbf{s}),$$

where $\bar{\mathbf{s}}$ is the residue class of \mathbf{s} in $\mathbb{Z}^n / \mathbb{Z}A_\sigma$, i.e., $c(\mathbf{s})$ is a quasi-polynomial on C .

3.3 Parametric Polytopes

We start our discussion of the enumeration of parametric polytopes with a couple of examples.

Example 11 Consider the program in Figure 6 and suppose we want to know how many times the statement **S1** is executed. This count is equivalent to the number of integer points in the parametric polytope

$$P_N = \left\{ \binom{i}{j} \in \mathbb{Q}^2 \mid \begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 1 & -1 \end{pmatrix} \binom{i}{j} \geq \begin{pmatrix} 0 \\ -1 \\ 0 \\ 0 \end{pmatrix} (N) + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \right\}$$

The solution is obvious in this case,

$$\#\{(i, j) \in \mathbb{Z}^2 \mid 1 \leq i \leq N \wedge 1 \leq j \leq i\} = \frac{N(N+1)}{2}.$$

The solution is shown graphically for $N = 4$ in Figure 7.

Example 12 Consider the slightly more complicated program in Figure 8 and suppose again that we want to know how many times the statement **S1** is executed. Again, this number is equal to the number of integer points in some parametric polytope

$$P_{\binom{N}{M}} = \left\{ \binom{i}{j} \in \mathbb{Q}^2 \mid \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ -2 & -1 \end{pmatrix} \binom{i}{j} \geq \begin{pmatrix} 0 & 0 \\ 1 & -1 \\ -1 & 1 \\ 0 & 0 \\ -1 & 0 \end{pmatrix} \binom{N}{M} + \begin{pmatrix} 0 \\ 0 \\ -3 \\ 0 \\ 0 \end{pmatrix} \right\}. \quad (4)$$

```

for(i=max(0,N-M); i<=N-M+3; i++)
  for(j=0; j<=N-2*i; j++)
    S1;

```

Figure 8: More complicated example program.

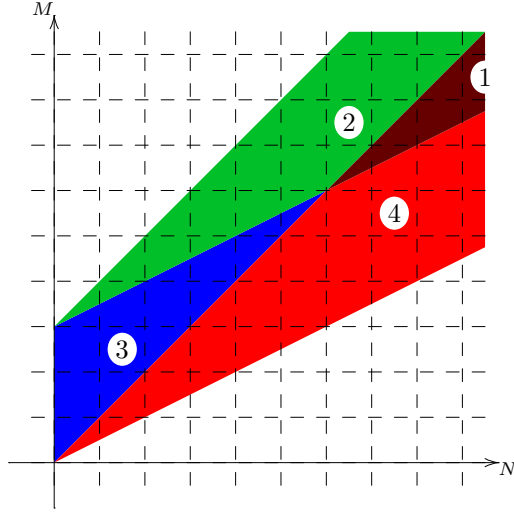


Figure 9: Chamber decomposition of Example 12.

In this example, the solution is no longer obvious. Using the techniques explained in Section 5, we obtain the solution

$$c_P = \begin{cases} -4N + 8M - 8 & \text{if } M \leq N \leq 2M - 6 & (1) \\ MN - 2N - M^2 + 6M - 8 & \text{if } N \leq M \leq N + 3 \wedge N \leq 2M - 6 & (2) \\ \frac{N^2}{4} + \frac{3}{4}N + \frac{1}{2} \lfloor \frac{N}{2} \rfloor + 1 & \text{if } 0 \leq N \leq M \wedge 2M \leq N + 6 & (3) \\ \frac{N^2}{4} - MN - \frac{5}{4}N + M^2 + 2M + \frac{1}{2} \lfloor \frac{N}{2} \rfloor + 1 & \text{if } M \leq N \leq 2M \leq N + 6. & (4) \end{cases}$$

As in the case of vector partition functions, we obtain different solutions in different regions of the parameter space. Figure 9 shows the chamber decomposition of the parameter space. Note that in contrast to the case of vector partition functions, the chambers are not necessarily cones.

Although Ehrhart (1977) already considered some problems in two parameters with solutions containing chambers (without using the term “chamber”), the general enumeration of parametric polytopes was first described by Clauss and Loechner (1998). The results are similar to those of Sturmfels (1995) for vector partition functions. Again, the enumerator can be described by a collection of “chambers”, each with an associated quasi-polynomial. Before stating the main results of Clauss and Loechner (1998) in more detail, we first extend the definition of chamber complexes and describe how to compute the (parametric) vertices of a parametric polytope. We end with a slight digression on the vertex enumeration algorithm.

We start with a seemingly independent definition of chamber complexes from Rambau (1996).

Definition 3.7 (Chamber complex of a polytope projection) Let P be a polytope, $P \subset \mathbb{Q}^{n+d}$, and let π_n be the projection onto the first n dimensions. The chamber complex of (P, π_n) is the polyhedral subdivision of the polytope $Q = \pi_n(P)$ formed by the closures of the connected components of the set

$$Q \setminus \{ \pi_n(F) \mid F \text{ a face of } P \text{ with } \dim F < \dim Q \}$$

together with all their faces.

We now consider our extension of Definition 3.6 to parametric polytopes.

Definition 3.8 (Chamber complex of a parametric polytope) Let $P \subset \mathbb{Q}^n \times \mathbb{Q}^d$ be a polyhedron such that P defines a parametric polytope. The chamber complex of P is the common refinement of projections onto the first n dimensions of the (generic) k -dimensional faces of the polyhedron P , with $k := \dim \pi_n(P)$.

It is clear that Definition 3.7 is a special case of Definition 3.8. Replacing the word “polytope” by the word “polyhedron” in Definition 3.7 yields essentially the same definition as Definition 3.8. To see that Definition 3.8 is also an extension of Definition 3.6, write the vector partition problem

$$\{ \boldsymbol{\lambda} \in \mathbb{N}^d \mid A\boldsymbol{\lambda} = \mathbf{u} \}$$

as the intersection of \mathbb{Z}^d and the parametric polytope $P_{\mathbf{u}}$, with

$$P = \left\{ (\mathbf{u}, \boldsymbol{\lambda}) \in \mathbb{Q}^n \times \mathbb{Q}^d \mid \begin{bmatrix} -I & A \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \boldsymbol{\lambda} \end{bmatrix} = \mathbf{0} \wedge \boldsymbol{\lambda} \geq \mathbf{0} \right\}.$$

First note that the projection of the polyhedron P onto the first n dimensions is simply $\text{pos } A$, which is, by definition, of dimension n . The n -dimensional faces are the intersections of P with d supporting hyperplanes. The n equalities are always included in this set of hyperplanes and the remaining $d - n$ correspond to the inequalities $\lambda_j \geq 0$, which we fix to $\lambda_j = 0$. The n remaining λ_j s form a subset σ and are generically linearly independent. The projection of the corresponding n -face is simply $\text{pos } A_\sigma$.

Clauss and Loechner (1998) describe an algorithm for computing the full-dimensional chambers in the chamber complex of a parametric polytope.¹ First note that we may assume that $\dim \pi_n(P) = n$. Otherwise, we may replace the parameters by k linearly independent parameters and substitute the old parameters for the new after the computation. The algorithm maintains a list (R_i) of n -dimensional regions, with pairwise intersections of dimension at most $n - 1$. Initially, the list contains a single region, corresponding to one of the n -faces. In each step, a new n -face with projection R' is considered. For each R_i in the current list, if $\dim(R_i \cap R') = n$ then R_i is replaced by two new regions $R_i \cap R'$ and $\overline{R_i} \setminus R'$, where \overline{S} is the closure of S . After scanning the whole list, the region $R' \setminus \bigcup_i R_i$ is added. If any of the set differences is empty, the region is discarded. Note that some of the intermediate regions may not be polyhedra but rather unions of polyhedra. In Section 6.1 we discuss some different possibilities of representing chamber complexes.

Example 13 Consider the parametric polytope

$$P = \left\{ (\mathbf{p}, \boldsymbol{\lambda}) \in \mathbb{Q}^2 \times \mathbb{Q}^4 \mid \begin{bmatrix} 1 & 2 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \boldsymbol{\lambda} = \mathbf{p} \wedge \boldsymbol{\lambda} \geq \mathbf{0} \right\},$$

which corresponds to the vector partition function from Example 10. A 2-face is formed by intersecting P with two hyperplanes of the form $\lambda_i = 0$. Selecting $\lambda_1 = 0$ and $\lambda_2 = 0$, we have

$$\left\{ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \lambda_3 \\ \lambda_4 \end{bmatrix} = \mathbf{p} \wedge \begin{bmatrix} \lambda_3 \\ \lambda_4 \end{bmatrix} \geq \mathbf{0} \right\}.$$

¹Clauss and Loechner (1998) use the term “validity domain” instead of “chamber”.

Projection onto the parameter space yields

$$R_6 = \{ (p_1, p_2) \in \mathbb{Q}^2 \mid p_1 \geq 0 \wedge p_2 \geq 0 \},$$

which is shown in Figure 4. The initial list is then simply (R_6) . Selecting $\lambda_1 = 0$ and $\lambda_3 = 0$, we obtain

$$R_5 = \{ (p_1, p_2) \in \mathbb{Q}^2 \mid 2p_2 \geq p_1 \geq p_2 \}$$

through a similar computation. We have $R_6 \cap R_5 = R_5$, $\overline{R_6 \setminus R_5} = R_4$ and $\overline{R_5 \setminus R_6} = \emptyset$. The new list is then (R_5, R_4) . Considering R_4 does not change the list, but R_3 changes the list to (R_3, R_1, R_4) . Further considering R_2 and R_1 again does not change the list and so the chambers are R_3, R_1 and R_4 .

The number of chambers is polynomial if the dimensions d and n are fixed. This follows from the following well-known lemma (see, e.g., Buck 1943, Edelsbrunner 1987 or Matoušek 2002, Section 6.1).

Lemma 3.9 *Let $\Phi(m, n) = \binom{m}{0} + \binom{m}{1} + \dots + \binom{m}{n}$. Then m hyperplanes in \mathbb{Q}^n decompose the space into at most $\Phi(m, n)$ polyhedral cells. Furthermore, if we fix n , then there is a polynomial time algorithm which, given m hyperplanes in \mathbb{Q}^n , computes the defining inequalities for each of these cells.*

Proof We prove both parts by induction on m . Certainly the statement is true for $m=0$. Suppose we have a collection of m hyperplanes $\mathcal{H}_1, \dots, \mathcal{H}_m$, and assume that these decompose \mathbb{Q}^n into at most $\Phi(m, n)$ polyhedral cells whose defining inequalities may be determined in polynomial time. Let us then add a new hyperplane \mathcal{H}_{m+1} , which will split some of the old cells in two. The cells that it splits correspond exactly to the cells that the m hyperplanes $\mathcal{H}_i \cap \mathcal{H}_{m+1} \subset \mathcal{H}_{m+1}$, for $1 \leq i \leq m$, decompose the $(n-1)$ -dimensional space \mathcal{H}_{m+1} into. Inductively, there are at most $\Phi(m, n-1)$ of these cells in \mathcal{H}_{m+1} , and their descriptions may be computed in polynomial time. Therefore, the hyperplanes $\mathcal{H}_1, \dots, \mathcal{H}_{m+1}$ decompose \mathbb{Q}^n into at most $\Phi(m, n) + \Phi(m, n-1) = \Phi(m+1, n)$ cells, and we may compute their descriptions in polynomial time. \square

Each full-dimensional chamber is a union of (adjacent) cells. Furthermore, there are only polynomially many n -faces (for fixed dimensions). The number of chambers is therefore also polynomial for fixed dimensions.

Example 14 Consider the parametric polytope

$$P = \{ (\mathbf{p}, x) \in \mathbb{Q}^2 \times \mathbb{Q} \mid 0 \leq x \leq 10 \wedge 0 \leq x + p_1 \leq 20 \wedge 0 \leq x + p_2 \leq 20 \}.$$

Both the cell and the chamber decomposition of the parameter space are shown in Figure 10. Note that this parameter space is 2-dimensional. The hyperplanes are therefore lines and are depicted in the figure using dashed lines. In all, there are 11 hyperplanes dividing the parameter space into 44 (full-dimensional) cells, 12 of which lie inside the projection of P onto the parameter space. There are 7 (full-dimensional) chambers, which are delineated by thick lines in the figure.

We turn now to the (parametric) vertices of a parametric polytope. We will assume however that the parametric polytope is full-dimensional. If P is not full-dimensional, then we may first transform it into another polytope which has the same number of integer points but which is embedded in a lower dimensional space. In particular, the ambient space of the transformed polytope has the same dimension as P and so the transformed polytope is full-dimensional. If P is of dimension $n+d-l$, with $l \geq 1$, then its description contains an equality $\langle \mathbf{a}, \mathbf{x} \rangle = \langle \mathbf{b}, \mathbf{p} \rangle + c$, with $\langle \cdot, \cdot \rangle$ the standard inner product. Let $\mathbf{a}' = \mathbf{a}/g$, with g the greatest common divisor (gcd) of the elements in \mathbf{a} . Note that $\mathbf{a} \neq \mathbf{0}$, since we assume that $\dim \pi_n P = n$. \mathbf{a}'^T can be extended to a unimodular matrix U (Bik 1996).² Let $P'_\mathbf{p} = UP_\mathbf{p}$. Since U and U^{-1} are unimodular, there is a one-to-one correspondence between the integer points in $P_\mathbf{p}$ and those in $P'_\mathbf{p}$ and so the number

²A unimodular matrix is an integer matrix with determinant 1 or -1 .

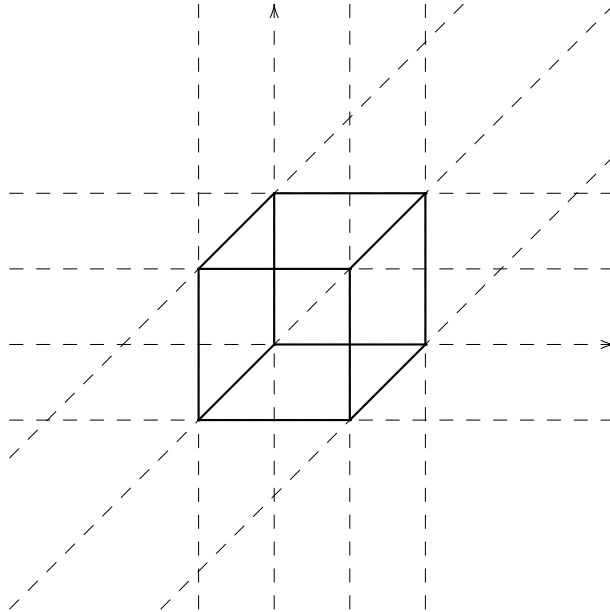


Figure 10: Cell and chamber decomposition. The cells are delineated by dashed lines (the hyperplanes); the chambers are delineated by thick lines.

of points in both polytopes is the same, i.e., $c_{P'} = c_P$. Furthermore, the first coordinate x'_1 of $P'_\mathbf{p}$ is independent of the other coordinates since $gx'_1 = g(\mathbf{a}', \mathbf{x}) = \langle \mathbf{b}, \mathbf{p} \rangle + c$ by construction of U and so $P'_\mathbf{p}$ is the product of $P''_\mathbf{p} = \{ (\langle \mathbf{b}, \mathbf{p} \rangle + c)/g \}$ and some $P_\mathbf{p}^{(1)} \in \mathbb{Q}^{d-1}$. Therefore we can factorize $P_\mathbf{p}$ and compute the number of points in $P_\mathbf{p}$ as the product of those in $P''_\mathbf{p}$ and $P_\mathbf{p}^{(1)}$, i.e., $c_P = c_{P'} = c_{P''} \cdot c_{P^{(1)}}$. The number of integer points in $P''_\mathbf{p}$ is zero or one depending on the parameters and can be represented by a periodic number.³ Repeating the above l times, yields a $P^{(l)} \in \mathbb{Q}^n \times \mathbb{Q}^{d-l}$ of full dimension. Obviously, we may also remove all equalities simultaneously through a single unimodular transformation. Also note that since the unimodular transformation is independent of the parameters, the projection of the faces of P onto the parameter space does not change. In particular, the chamber decomposition remains intact.

Example 15 Consider once more the vector partition function

$$c(\mathbf{p}) = \# \left\{ \boldsymbol{\lambda} \in \mathbb{N}^4 \mid \begin{bmatrix} 1 & 2 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \boldsymbol{\lambda} = \mathbf{p} \right\}$$

from Example 10. For a given $\mathbf{p} \in \mathbb{Z}^n$, the solution set

$$P_\mathbf{p} = \left\{ \boldsymbol{\lambda} \in \mathbb{Q}^4 \mid \boldsymbol{\lambda} \geq \mathbf{0} \text{ and } \begin{bmatrix} 1 & 2 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \boldsymbol{\lambda} = \mathbf{p} \right\}$$

is a two dimensional polytope in \mathbb{Q}^4 , so it is helpful to convert it to a full-dimensional polytope in \mathbb{Q}^2 (without changing the number of integer points). To do this, extend the matrix to

$$M = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

³This periodic number may depend on several parameters. Such periodic numbers will be defined in Definition 3.12 as the obvious extension of Definition 3.2.

which is unimodular (that is, it has determinant ± 1 and so, as a linear transformation, it bijectively maps \mathbb{Z}^4 to \mathbb{Z}^4), and perform the change of coordinates $\boldsymbol{\lambda} \mapsto \boldsymbol{\lambda}' = M\boldsymbol{\lambda}$. Then

$$\begin{aligned} c(\mathbf{p}) &= \# \left\{ \boldsymbol{\lambda}' \in \mathbb{Z}^4 \mid M^{-1}\boldsymbol{\lambda}' \geq \mathbf{0} \text{ and } \begin{bmatrix} 1 & 2 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} M^{-1}\boldsymbol{\lambda}' = \mathbf{p} \right\} \\ &= \# \left\{ \boldsymbol{\lambda}' \in \mathbb{Z}^4 \mid M^{-1}\boldsymbol{\lambda}' \geq \mathbf{0} \text{ and } \lambda'_1 = p_1, \lambda'_2 = p_2 \right\} \\ &= \# \left\{ (\lambda'_3, \lambda'_4) \in \mathbb{Z}^2 \mid \begin{bmatrix} -1 & 2 & 1 & -2 \\ 1 & -1 & -1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ \lambda'_3 \\ \lambda'_4 \end{bmatrix} \geq \mathbf{0} \right\}. \end{aligned}$$

In this case P'' is simply $\{(p_1, p_2, \lambda'_1, \lambda'_2) \in \mathbb{Q}^2 \times \mathbb{Q}^2 \mid \lambda'_1 = p_1 \wedge \lambda'_2 = p_2\}$ and so $c_{P''} = 1$.

For computing the parametric vertices, we basically follow the algorithm of Loechner and Wilde (1997), but use a simpler terminology.⁴ Let $P_{\mathbf{p}} = \{\mathbf{x} \in \mathbb{Q}^d \mid A\mathbf{x} \geq B\mathbf{p} + \mathbf{c}\}$ be a parametric polytope of dimension d with n parameters. The vertices are the 0-faces of $P_{\mathbf{p}}$, i.e., each vertex is the unique element in the intersection $V_{\mathbf{p}}$ of $P_{\mathbf{p}}$ with some set $\{\mathbf{x} \in \mathbb{Q}^d \mid A'\mathbf{x} = B'\mathbf{p} + \mathbf{c}'\}$, where $A'\mathbf{x} \geq B'\mathbf{p} + \mathbf{c}'$ is a subsystem of $A\mathbf{x} \geq B\mathbf{p} + \mathbf{c}$ such that the dimension of this intersection is 0. I.e.,

$$V_{\mathbf{p}} = \{\mathbf{v}(\mathbf{p})\} = \{\mathbf{x} \in \mathbb{Q}^d \mid A'\mathbf{x} = B'\mathbf{p} + \mathbf{c}' \wedge A''\mathbf{x} \geq B''\mathbf{p} + \mathbf{c}''\},$$

where $A''\mathbf{x} \geq B''\mathbf{p} + \mathbf{c}''$ are the remaining inequalities of $P_{\mathbf{p}}$. Since $P_{\mathbf{p}}$ is of dimension d , at least d equalities are needed. In fact, all vertices can be obtained by intersecting $P_{\mathbf{p}}$ with all possible combinations of d linearly independent equalities, i.e., such that A' is non-singular. The vertex $\mathbf{v}(\mathbf{p})$ is obtained as the solution of the linear system $A'\mathbf{x} = B'\mathbf{p} + \mathbf{c}'$. Obviously, $\mathbf{v}(\mathbf{p})$ is a d -vector of affine functions of the parameters. Such a solution is only an actual vertex of $P_{\mathbf{p}}$ if the set $V_{\mathbf{p}}$ is non-empty. This set may be non-empty only for some values of the parameters. We say that $\mathbf{v}(\mathbf{p})$ is an *active vertex* of $P_{\mathbf{p}}$ for these values of the parameters, i.e., those that belong to the following polyhedron:

$$R = \{\mathbf{p} \in \mathbb{Q}^n \mid A''\mathbf{v}(\mathbf{p}) \geq B''\mathbf{p} + \mathbf{c}''\}.$$

Note that R is the projection onto the parameter space of V , the polyhedron defining the parametric (0-dimensional) polytope $V_{\mathbf{p}}$, and that V is an n -face of the polyhedron P . It follows that the chambers are exactly those regions with a fixed set of active vertices. Since the number of linearly independent sets of d constraints from the original set of m constraints is at most $\binom{m}{d}$, the total number of parametric vertices is polynomial in the number of constraints (for fixed d) and therefore polynomial in the input size.

Example 16 Consider the parametric polytope

$$P = \left\{ (\mathbf{p}, \mathbf{t}) \in \mathbb{Q}^2 \times \mathbb{Q}^2 \mid \begin{bmatrix} -1 & 2 \\ 1 & -1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \mathbf{p} + \begin{bmatrix} 1 & -2 \\ -1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{t} \geq \mathbf{0} \right\},$$

which is the set we obtained in Example 15. The (parametric) vertices of $P_{\mathbf{p}}$ can be obtained as the intersections of pairs of (parametric) facets of $P_{\mathbf{p}}$. The facets $t_1 = 0$ and $t_1 - 2t_2 = p_1 - 2p_2$, for example, intersect at the point $\mathbf{v}_1 = (0, -p_1/2 + p_2)$. That is, we have chosen the subsystem

$$\begin{bmatrix} -1 & 2 \\ 0 & 0 \end{bmatrix} \mathbf{p} + \begin{bmatrix} 1 & -2 \\ 1 & 0 \end{bmatrix} \mathbf{t} \geq \mathbf{0}$$

and we have obtained

$$V_{\mathbf{p}} = \{(0, -p_1/2 + p_2)\}.$$

Notice that the vertex corresponds to the affine transformation

$$T_1 : \mathbb{Q}^2 \rightarrow \mathbb{Q}^2 : \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \mapsto \begin{bmatrix} 0 \\ -p_1/2 + p_2 \end{bmatrix}.$$

⁴The idea of using this simpler terminology is due to Rachid Seghir (Seghir 2004).

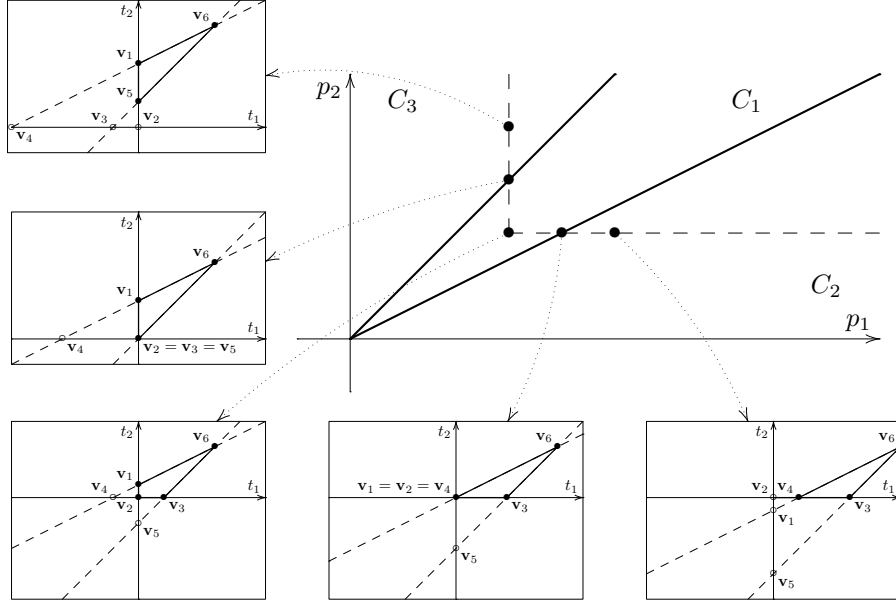


Figure 11: The chamber decomposition and parametric vertices of the parametric polytope in Example 16.

The remaining constraints

$$\begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix} \mathbf{p} + \begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ -p_1/2 + p_2 \end{bmatrix} \geq \mathbf{0}$$

simplify to $p_1 \geq 0$ and $2p_2 \geq p_1$. The vertex is therefore “active” (i.e., actually a vertex of $P_{\mathbf{p}}$) only when $2p_2 \geq p_1 \geq 0$ (for all other values of \mathbf{p} , $\mathbf{v}_1 \notin P_{\mathbf{p}}$). We similarly find the possible vertices $\mathbf{v}_2 = (0, 0)$, $\mathbf{v}_3 = (p_1 - p_2, 0)$, $\mathbf{v}_4 = (p_1 - 2p_2, 0)$, $\mathbf{v}_5 = (0, -p_1 + p_2)$ and $\mathbf{v}_6 = (p_1, p_2)$, active on the domains $2p_2 \geq p_1 \geq p_2$, $p_1 \geq p_2 \geq 0$, $p_1 \geq 2p_2 \geq 0$, $p_2 \geq p_1 \geq 0$, and $p_1, p_2 \geq 0$, respectively. The chambers are formed by the common refinement of these activity domains. We find, as in Example 13,

$$\begin{aligned} C_1 &= \{ \mathbf{p} \mid 2p_2 \geq p_1 \geq p_2 \} \\ C_2 &= \{ \mathbf{p} \mid p_1 \geq 2p_2 \geq 0 \} \\ C_3 &= \{ \mathbf{p} \mid p_2 \geq p_1 \geq 0 \}. \end{aligned}$$

Let V_C be the set of vertices active on chamber C , then

$$\begin{aligned} V_{C_1} &= \{ \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_6 \} \\ V_{C_2} &= \{ \mathbf{v}_3, \mathbf{v}_4, \mathbf{v}_6 \} \\ V_{C_3} &= \{ \mathbf{v}_1, \mathbf{v}_5, \mathbf{v}_6 \}. \end{aligned}$$

The top-right part of Figure 11 shows the chamber decomposition of the parameter space. The chambers in this figure are obviously the same as those in Figure 5. For five points \mathbf{p} in the parameter space, Figure 11 also shows the corresponding polytope $P_{\mathbf{p}}$ together with both its active (\bullet) and inactive (\circ) vertices. These five pictures show how the vertices evolve as \mathbf{p} is moved along the dashed line in the parameter space.

From the discussion above we conclude the following proposition.

Proposition 3.10 (Decomposition) *Fix d and n . There exists a polynomial time algorithm, which, given a parametric polytope $P \subset \mathbb{Q}^n \times \mathbb{Q}^d$, decomposes \mathbb{Q}^n into chambers C_i and, for each i , computes a collection of affine transformations $T_{i1}, T_{i2}, \dots, T_{im_i} : \mathbb{Q}^n \rightarrow \mathbb{Q}^d$, such that, for $\mathbf{p} \in C_i$, the vertices of $P_{\mathbf{p}}$ are $T_{i1}(\mathbf{p}), T_{i2}(\mathbf{p}), \dots, T_{im_i}(\mathbf{p})$.*

We can now state the following theorem.

Theorem 3.11 (Clauss and Loechner 1998, Theorem 2) *The enumerator of a parametric polytope $P_{\mathbf{p}}$ of dimension d is a quasi-polynomial of degree d in \mathbf{p} on each of a set of chambers that form a subdivision of the parameter space. The period of the quasi-polynomial in a given chamber divides the denominators that appear in the affine transformations defining the vertices active on that chamber.*

In Section 5.4 we will see that the quasi-polynomials from Theorem 3.11 can be computed in polynomial time (for fixed dimensions). These quasi-polynomials are the natural extension of the univariate quasi-polynomials from Definition 3.3.

Definition 3.12 *A rational n -periodic number $U(\mathbf{p})$ is a function $\mathbb{Z}^n \rightarrow \mathbb{Q}$, such that there exist a period $\mathbf{q} = (q_1, \dots, q_n) \in \mathbb{N}^n$ such that $U(\mathbf{p}) = U(\mathbf{p}')$ whenever $p_i \equiv p'_i \pmod{q_i}$, for $1 \leq i \leq n$.*

Definition 3.13 *A quasi-polynomial of degree d in n variables \mathbf{p} is a polynomial expression of degree d in \mathbf{p} with rational n -periodic numbers for coefficients. The period of a quasi-polynomial is the lcm of the periods of its coefficients.*

In the parametric vertex enumeration algorithm above, several subsystems $A'\mathbf{x} \geq B'\mathbf{p} + \mathbf{c}'$ may result in the same parametric vertex. It is important to remove the duplicates from the resulting set of parametric vertices. We slightly digress here to consider how Loechner and Wilde (1997) removed these duplicates because it was implemented incorrectly in versions of `PolyLib` up until 5.20.0, resulting in the elimination of non-duplicate parametric vertices. They consider the $(d+n)$ -dimensional polyhedron P and search for all combinations of d constraints such that the subset of P saturating these constraints forms an n -face, which they check by counting the number of generators of P that saturate this set of constraints. If this number is at least $n+1$ and contains at least one vertex, then the intersection is an n -face of P . They perform a depth-first search on the (ordered) list of all constraints, which results in an implicit ordering on the solutions. To eliminate duplicate n -faces, they check whether a constraint that is not part of the selected constraints and that occurs earlier in the list of constraints than the last selected constraint already saturates the generators saturated by the selected set of constraints. If this is the case, then the same set of generators was also saturated by a previous solution, so the current solution is dropped. To ensure that the face is actually of dimension n and not of some higher dimension, the search algorithm only adds a constraint to the set of constraints to be saturated, if the number of generators saturated by both the new constraint and the already selected constraints is smaller than the number of generators saturated by just the already selected constraints. The idea is to only add a constraint that (as an equality) is linearly independent of the other selected constraints (as equalities), but they actually check whether the new equality is redundant with respect to the selected equalities and the remaining inequalities. Adding a test to see whether the considered constraint is actually linearly dependent of the already selected equalities before discarding it solves the problem. Our fix should be available in the next release of `PolyLib`.

Example 17 Consider the polyhedron $P \subset \mathbb{Q}^4 \times \mathbb{Q}$

$$\left\{ (\mathbf{x}, p) \mid \begin{bmatrix} 0 & 0 & -20 & 0 \\ 0 & 1 & 20 & 0 \\ 0 & -1 & 0 & 0 \\ 4 & -20 & 0 & 0 \\ -4 & 20 & 0 & 0 \\ 0 & 1 & 0 & 20 \\ 0 & 0 & 0 & -20 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 19 \\ -20 \\ 0 \\ -1 \\ 1 \\ -20 \\ 19 \end{bmatrix} p + \begin{bmatrix} 1 \\ 16 \\ 4 \\ 23 \\ -22 \\ 16 \\ 1 \end{bmatrix} \geq \mathbf{0} \right\}.$$

The parametric polytope P_p is 4-dimensional, so we need a subsystem of four linearly independent constraints to form a vertex of P_p (i.e., a 1-face of P). Saturating just *three* of the above constraints can yield a 1-dimensional face. In particular, intersecting P with

$$\left\{ (\mathbf{x}, p) \mid \begin{bmatrix} 0 & 0 & -20 & 0 \\ 0 & 1 & 20 & 0 \\ 4 & -20 & 0 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 19 \\ -20 \\ -1 \end{bmatrix} p + \begin{bmatrix} 1 \\ 16 \\ 23 \end{bmatrix} = \mathbf{0} \right\}.$$

yields the 1-dimensional face defined by the equalities

$$\begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 20 \end{bmatrix} \mathbf{x} + \begin{bmatrix} -21 \\ -1 \\ -19 \\ -19 \end{bmatrix} p + \begin{bmatrix} 363 \\ 17 \\ -1 \\ -1 \end{bmatrix} = \mathbf{0}$$

and the inequality $-p + 21 \geq 0$. Since this set of 3 constraints already saturates the minimum number of generators, selecting a fourth constraint would never saturate fewer, but still enough, generators and would therefore never result in a parametric vertex in the old implementation. In reality, both the systems

$$\begin{bmatrix} 0 & 0 & -20 & 0 \\ 0 & 1 & 20 & 0 \\ 4 & -20 & 0 & 0 \\ 0 & 1 & 0 & 20 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 19 \\ -20 \\ -1 \\ -20 \end{bmatrix} p + \begin{bmatrix} 1 \\ 16 \\ 23 \\ 16 \end{bmatrix} = \mathbf{0}$$

and

$$\begin{bmatrix} 0 & 0 & -20 & 0 \\ 0 & 1 & 20 & 0 \\ 4 & -20 & 0 & 0 \\ 0 & 0 & 0 & -20 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 19 \\ -20 \\ -1 \\ 19 \end{bmatrix} p + \begin{bmatrix} 1 \\ 16 \\ 23 \\ 1 \end{bmatrix} = \mathbf{0}$$

yield (the same) parametric vertices. The duplicate is correctly removed in the current implementation because the last constraint of the first system saturates all generators of the second system.

3.4 Parametric Projected Sets

In this section we briefly discuss the most general counting problems we will consider: the enumeration of parametric projected sets

$$S = \left\{ (\mathbf{p}, \mathbf{x}) \in \mathbb{Z}^n \times \mathbb{Z}^d \mid \exists \mathbf{y} \in \mathbb{Z}^{d'} : \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} \geq D\mathbf{p} + \mathbf{c} \right\}. \quad (5)$$

The enumerator can be written as

$$\begin{aligned} c_S(\mathbf{p}) &= \#S_{\mathbf{p}} \\ &= \# \left\{ \mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{y} \in \mathbb{Z}^{d'} : \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} \geq D\mathbf{p} + \mathbf{c} \right\} \\ &= \# \left(\pi_d \left(\mathbb{Z}^{d+d'} \cap \left\{ (\mathbf{x}, \mathbf{y}) \in \mathbb{Q}^{d+d'} \mid \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} \geq D\mathbf{p} + \mathbf{c} \right\} \right) \right). \end{aligned}$$

That is, c_S enumerates the number of elements in the projection of the integer points in a parametric polytope. We conclude this brief discussion with an example indicating that this problem is more difficult than the enumeration of a parametric polytope.

Example 18 Consider the example program shown in Figure 12, adapted from an example from Clauss (1997). Assume we want to know the total number of array elements accessed by the statement in the inner loop as a function of the symbolic parameter p . This problem is equivalent to counting the number of elements in the set

$$S_p = \{ l \in \mathbb{Z} \mid \exists i, j \in \mathbb{Z} : l = 6i + 9j - 7 \wedge 1 \leq j \leq p \wedge 1 \leq i \leq 8 \}, \quad (6)$$

which can be written as:

$$\left\{ l \in \mathbb{Z} \mid \exists \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Z}^2 : \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} l + \begin{bmatrix} -6 & -9 \\ 6 & 9 \\ 0 & 1 \\ 0 & -1 \\ 1 & 0 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \end{bmatrix} p + \begin{bmatrix} -7 \\ 7 \\ 1 \\ 0 \\ 1 \\ -8 \end{bmatrix} \right\}.$$

```

for (j = 1; j <= p; ++j)
  for (i = 1; i <= 8; ++i)
    a[6i+9j-7] = a[6i+9j-7] + 5;

```

Figure 12: Example Program.

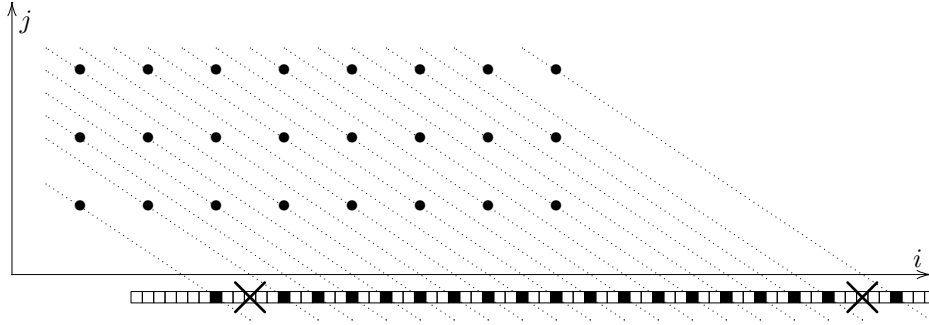


Figure 13: Array elements accessed for $p = 3$.

The equality in (6) has been rewritten here as a pair of inequalities to conform to (5).

Figure 13 shows the array elements that are accessed for $p = 3$. These elements do not correspond to the integer points in a polytope. Even after scaling by 3 it still contains two “holes” (marked by \times on the figure). These holes complicate the enumeration of such sets.

For $p = 3$, the set S_p contains 19 points, see Figure 13. In general, the number of points in S_p can be described by the function

$$c_S(p) = \begin{cases} 8 & \text{if } p = 1 \\ 3p + 10 & \text{if } p \geq 2 \end{cases}.$$

As in the case of parametric polytopes, the count is represented by different quasi-polynomials (in this case actual polynomials) on different chambers.

4 Two Representations

In the previous section we have discussed a variety of counting functions and in each case we have seen that we can represent this function as a set of quasi-polynomials, each associated to a chamber of the parameter space. For some of these counting problems we have also shown the corresponding generating function, which is sometimes easier to compute. In fact, it is trivial to compute for vector partition functions.

We will use the following representations for generating functions and explicit functions. Note that neither of these representations is unique, i.e., any generating function or explicit function can be represented in multiple ways.

Definition 4.1 *By a rational generating function $f(\mathbf{x})$, we will mean a function written in the form*

$$f(\mathbf{x}) = \sum_{i \in I} \alpha_i \frac{\mathbf{x}^{\mathbf{q}_i}}{\prod_{j=1}^{k_i} (1 - \mathbf{x}^{\mathbf{b}_{ij}})}, \quad (7)$$

where $\mathbf{x} \in \mathbb{C}^n$, $\alpha_i \in \mathbb{Q}$, $\mathbf{q}_i \in \mathbb{Z}^n$, and $\mathbf{b}_{ij} \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$.

Such a rational generating function is sometimes called *short* if the number of factors k_i in each term is bounded by a constant and if the number of elements in the index set I is polynomial in the input size of the enumerated set.

Definition 4.2 A step-polynomial $g : \mathbb{Z}^n \rightarrow \mathbb{Q}$ is a function written in the form

$$g(\mathbf{p}) = \sum_{j=1}^m \alpha_j \prod_{k=1}^{d_j} \lfloor \langle \mathbf{a}_{jk}, \mathbf{p} \rangle + b_{jk} \rfloor,$$

where $\alpha_j \in \mathbb{Q}$, $\mathbf{a}_{jk} \in \mathbb{Q}^n$, $b_{jk} \in \mathbb{Q}$, $\langle \cdot, \cdot \rangle$ is the standard inner product, and $\lfloor \cdot \rfloor$ is the greatest integer function. We say that the degree of $g(\mathbf{p})$ is $\max_j \{d_j\}$.

A piecewise step-polynomial $c : \mathbb{Z}^n \rightarrow \mathbb{Q}$ is a decomposition of \mathbb{Q}^n into polyhedral chambers C_i with corresponding functions $g_i : C_i \cap \mathbb{Z}^n \rightarrow \mathbb{Q}$ such that

1. $c(\mathbf{p}) = g_i(\mathbf{p})$, for $\mathbf{p} \in C_i \cap \mathbb{Z}^n$ and
2. each g_i is a step-polynomial.

We say that the degree of $c(\mathbf{p})$ is $\max_i \deg g_i$.

Example 19 Recall the parametric polytope $P_{\mathbf{p}}'' = \{(\langle \mathbf{b}, \mathbf{p} \rangle + c)/g\}$ we encountered in Section 3.3 when reducing a parametric polytope that may not be full-dimensional to a full-dimensional polytope. As we already mentioned, the number of integer points in $P_{\mathbf{p}}''$ is zero or one depending on the parameters. In particular, if g divides $\langle \mathbf{b}, \mathbf{p} \rangle + c$, then $c_{P''}(\mathbf{p}) = 1$. Otherwise, $c_{P''}(\mathbf{p}) = 0$. We can represent this as the step-polynomial

$$c_{P''} = \left\lfloor \frac{\langle \mathbf{b}, \mathbf{p} \rangle + c}{g} \right\rfloor - \left\lfloor \frac{\langle \mathbf{b}, \mathbf{p} \rangle + c}{g} - \frac{1}{g} \right\rfloor.$$

As we will explain in Section 8.2, we use “strides” to represent step-polynomials of this form in our implementation.

There are good reasons to consider both the explicit enumerator $c(\mathbf{p})$ and its generating function $C(\mathbf{x})$. An explicit function representation $c(\mathbf{p})$ has the advantage of being easily evaluated for a particular value of \mathbf{p} . Such a representation is therefore preferred in the compiler community (see, for example, Verdoolaege et al. 2004b). The most straightforward way of obtaining the coefficient of $\mathbf{x}^{\mathbf{p}}$, i.e., $c(\mathbf{p})$, from a rational generating function $C(\mathbf{x})$, on the other hand, is to expand the Laurent power series up to power \mathbf{p} , which is a process that is exponential in the size of \mathbf{p} .

The advantage of rational generating functions is that we may apply many computational tools to manipulate them and obtain information from them (see, for example, Barvinok and Woods 2003; Woods 2004). Consider, for example, summation. Let $C(\mathbf{x}, \mathbf{y})$ be the generation function of $c(\mathbf{p}, \mathbf{t})$, i.e.,

$$C(\mathbf{x}, \mathbf{y}) = \sum_{(\mathbf{p}, \mathbf{t})} c(\mathbf{p}, \mathbf{t}) \mathbf{x}^{\mathbf{p}} \mathbf{y}^{\mathbf{t}}.$$

Suppose we want to compute

$$d(\mathbf{t}) = \sum_{\mathbf{p}} c(\mathbf{p}, \mathbf{t}),$$

where we assume that for each \mathbf{t} , $c(\mathbf{p}, \mathbf{t})$ is non-zero only for a finite number of values of \mathbf{p} . To compute $d(\mathbf{t})$ explicitly we need to solve a set of new counting problems (see Section 6.4). To compute the corresponding generating function $D(\mathbf{y}) = C(\mathbf{1}, \mathbf{y})$, however, we “simply” need to plug in $\mathbf{1}$ for \mathbf{x} .

More importantly, Barvinok and Woods (2003) show that given a generating function that corresponds to a polytope, we may compute, in polynomial time, a generating function corresponding to the projection of this polytope.⁵ Using our results on the polynomial interconvertibility of piecewise step-polynomials and their rational generating functions (see Section 6.5), we will be able to show that we can compute the explicit enumerator of a parametric projected set in polynomial time. To the best of our knowledge, no other polynomial time algorithms to compute this enumerator have been proposed before.

⁵We will be more precise about this correspondence in the next sections.

5 Barvinok's Algorithm

In this section we explain Barvinok's algorithm for computing the number of integer points in a polytope and show how it can be applied to compute both the explicit enumerator of a parametric polytope and its generating function. We try to provide enough background information to make the algorithm understandable and focus on some implementation issues. For a more detailed theoretical discussion of the basic algorithm we refer to Barvinok and Pommersheim (1999). Many of the implementation details were inspired by De Loera et al. (2004) or Yoshida (2004a). Although most of this section is well-known, we are the first to combine the enumeration algorithm of Barvinok and Pommersheim (1999) with the chamber decomposition algorithm of Clauss and Loechner (1998). Furthermore, we extend the ideas of De Loera et al. (2004) for a practical implementation of Barvinok's algorithm to a much broader context. We start with a general overview of the algorithm, followed by a more detailed discussion of the two main substeps. In the final part, we show the application to parametric polytopes.

5.1 Overview

We first need a couple of definitions.

Definition 5.1 *Let $A \subset \mathbb{Q}^d$ be a set. The indicator function $[A]$ of A is defined by*

$$[A] : \mathbb{Q}^d \rightarrow \mathbb{Q} : [A](\mathbf{s}) = \begin{cases} 1 & \text{if } \mathbf{s} \in A \\ 0 & \text{if } \mathbf{s} \notin A \end{cases}$$

The reader with a background in functional programming may think of the function $[\cdot] : 2^{\mathbb{Q}^d} \rightarrow \mathbb{Q}^d \rightarrow \{0, 1\}$ as a curried version of the \ni -relation. The indicator function $[A]$ is then the partial evaluation of this function at A .

Definition 5.2 *Let S be an integer set such that the convex hull of S does not contain a line. The generating function of S is*

$$f(S; \mathbf{x}) = \sum_{\mathbf{s} \in S} \mathbf{x}^{\mathbf{s}}.$$

The generating function of a set $A \subset \mathbb{Q}^d$ is the generating function of the integer points in A

$$f(A; \mathbf{x}) = \sum_{\mathbf{s} \in A \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{s}},$$

provided A does not contain a line.

Note that the generating function of a set A can be written as

$$f(A; \mathbf{x}) = \sum_{\mathbf{s} \in \mathbb{Z}^d} [A](\mathbf{s}) \mathbf{x}^{\mathbf{s}},$$

i.e., it is the generating function of its indicator function restricted to the integer points. This function in turn can be seen as the enumerator of A when considered as a parametric set $A \subset \mathbb{Z}^d \times \mathbb{Z}^0$. The requirement for the sets to not contain any lines is needed to ensure that the series converge on some open region.

The generating functions of rational polyhedra satisfy the following properties.

Theorem 5.3 (Barvinok and Pommersheim 1999, Theorem 3.1)

- *If $P_1, \dots, P_k \subset \mathbb{Q}^d$ are rational polyhedra whose indicator functions satisfy a linear identity*

$$\sum_i \alpha_i [P_i] = 0,$$

then their generating function satisfy the same identity

$$\sum_i \alpha_i f(P_i; \mathbf{x}) = 0.$$

- If $\mathbf{m} + P$ is the translation of P by an integer vector $\mathbf{m} \in \mathbb{Z}^d$ then

$$f(P + \mathbf{m}; \mathbf{x}) = \mathbf{x}^{\mathbf{m}} f(P; \mathbf{x}).$$

- If P contains a line, then $f(P; \mathbf{x}) \equiv 0$

Note that the number of terms with coefficient 1 in the Laurent expansion of the generating function of a polytope P is equal to the number of integer points in P , i.e., $f(P; \mathbf{1}) = \#(A \cap \mathbb{Z}^d)$. The basic idea behind Barvinok's algorithm for counting the number of points in a polytope P is to first compute this generating function of P and then to evaluate it at $\mathbf{x} = \mathbf{1}$. This evaluation can be performed by computing the constant term of the Laurent expansion at $\mathbf{x} = \mathbf{1}$.

Example 20 Consider the one-dimensional polytope

$$S = \{s \mid s \geq 0 \wedge 2s \leq 13\},$$

shown in Figure 14. The generating function of S is

$$f(S; x) = x^0 + x^1 + x^2 + x^3 + x^4 + x^5 + x^6.$$

Evaluating this function at 1 yields

$$f(S; 1) = 1^0 + 1^1 + 1^2 + 1^3 + 1^4 + 1^5 + 1^6 = 7 = \#S,$$

as expected. This is not how Barvinok's algorithm would construct the generating function however. Consider instead the cone $K = \text{pos}\{1\}$. We have $K \cap \mathbb{Z} = \mathbb{N}$ and the generating function is then simply the standard geometric series

$$f(K; x) = \frac{1}{1-x} = \sum_{s=0}^{\infty} x^s.$$

This generating function contains more terms than we want, since it also contains the terms with power greater than or equal to 7. We can rectify this situation by subtracting the generating function of K shifted to $s = 7$, whence

$$\begin{aligned} f(S; \mathbf{x}) &= f(K; \mathbf{x}) - f(K + 7; \mathbf{x}) \\ &= \frac{x^0}{1-x} - \frac{x^7}{1-x} \\ &= \frac{x^0}{1-x} + \frac{x^6}{1-x^{-1}}, \end{aligned}$$

where in the last equality we applied (3). Note that the generating function of S is the sum of the generating functions of the (shifted) cones emanating from its "integer vertices" 0 and 6, with the second cone in the negative direction. The integer vertex appears as the power of x in the numerator of each generating function, whereas the rays of the cones appear as the power of x in the denominator of the generating functions. We will see in the remainder of this section that this holds in general. Computing the Laurent expansion at $x = 1$, we find

$$\frac{x^0}{1-x} = -\frac{x^0}{x-1} = -1(x-1)^{-1} + 0(x-1)^0 + \dots$$

and

$$\frac{x^6}{1-x^{-1}} = \frac{((x-1)+1)^7}{x-1} = 1(x-1)^{-1} + 7(x-1)^0 + \dots$$

The constant term in the Laurent expansion of $f(S; x)$ is therefore $f(S; 1) = 0 + 7 = 7$.

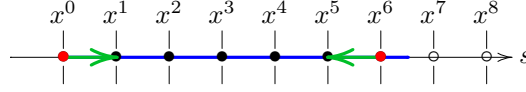


Figure 14: The set S from Example 20.

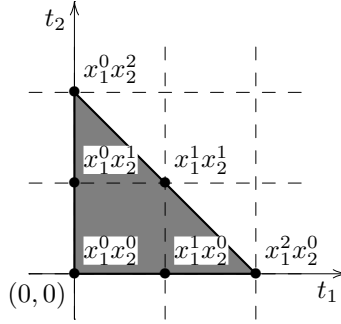


Figure 15: Barvinok Example. For each integer point (i, j) in the polytope T , there is a term $x_1^i x_2^j$ in the generating function $f(T; \mathbf{x})$.

Example 21 Consider the polytope

$$T = \{ \mathbf{t} \mid t_1 \geq 0 \wedge t_2 \geq 0 \wedge t_1 + t_2 \leq 2 \},$$

shown in Figure 15. The integer points in T are $(0, 0)$, $(1, 0)$, $(2, 0)$, $(0, 1)$, $(1, 1)$, and $(0, 2)$. The generating function of T is therefore

$$f(T; \mathbf{x}) = 1 + x_1 + x_1^2 + x_2 + x_1 x_2 + x_2^2.$$

Barvinok's algorithm, however, will produce this function in a different form which does not require the computation of all integer points in the polytope. In particular, it will compute $f(T; \mathbf{x})$ as the following short sum of rational functions:

$$f(T; \mathbf{x}) = \frac{x_1^2}{(1 - x_1^{-1})(1 - x_1^{-1}x_2)} + \frac{x_2^2}{(1 - x_2^{-1})(1 - x_1x_2^{-1})} + \frac{1}{(1 - x_1)(1 - x_2)}. \quad (8)$$

Notice that the general structure of this rational generating function is the same as in the previous example, with each vertex appearing as the power in the numerator of some term and the rays emanating from the same vertex appearing as the powers in the denominator. This construction process will be explained in more detail in the next section.

Given a parametric polytope $P_{\mathbf{p}}$, we can apply Barvinok's algorithm in two different ways to obtain either the explicit enumerator or its generating function.

- To obtain the explicit enumerator, first compute the chamber decomposition and the parametric vertices of $P_{\mathbf{p}}$. In each chamber, Barvinok's algorithm can be applied to the polytope defined by the parametric vertices active on that chamber. As we will see, the result is a piecewise step-polynomial.
- To obtain the generating function, first compute the generating function $f(P; (\mathbf{x}, \mathbf{y}))$ of the polyhedron $P = \{ (\mathbf{p}, \mathbf{t}) \mid \mathbf{t} \in P_{\mathbf{p}} \}$ and then compute the *partial* evaluation $c_P = f(P; (\mathbf{x}, \mathbf{1}))$. The result of this partial evaluation may be obtained as a rational generating function.

5.2 Computing Generating Functions

Following Barvinok and Pommersheim (1999), we will calculate the generating function $f(P_{\mathbf{p}} \cap \mathbb{Z}^d; \mathbf{x})$ as a rational generating function, and we will examine how it changes as \mathbf{p} varies within a

chamber. From this, we will calculate

$$c(\mathbf{p}) = \#(P_{\mathbf{p}} \cap \mathbb{Z}^d) = f(P_{\mathbf{p}} \cap \mathbb{Z}^d; \mathbf{1}).$$

5.2.1 Unimodular Cones

As a first step, we examine how to compute the generating function of an easy set: the integer points in a *unimodular cone*. We will then reduce the general case of a polyhedron to that of unimodular cones.

Definition 5.4 (Unimodular cone) *A (parametric) rational (shifted) polyhedral cone*

$$K = \{ \mathbf{x} \in \mathbb{Q}^d \mid C\mathbf{x} \geq \boldsymbol{\gamma} \}$$

is unimodular if $C \in \mathbb{Z}^{d \times d}$ is a unimodular matrix, i.e., if the rows \mathbf{c}_i of C form a basis for the lattice \mathbb{Z}^d . The righthand side may be parametric, i.e., $\boldsymbol{\gamma} = B\mathbf{p} + \mathbf{d}$, with $B \in \mathbb{Q}^{d \times n}$ and $\mathbf{d} \in \mathbb{Q}^d$.

We first consider the case where $\boldsymbol{\gamma}$ is zero. The unimodular cone K may then be written as

$$K = \{ \lambda_1 \mathbf{u}_1 + \lambda_2 \mathbf{u}_2 + \cdots + \lambda_d \mathbf{u}_d \mid \boldsymbol{\lambda} \geq \mathbf{0} \},$$

with $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_d$ the rays of K , i.e., $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_d$ are the columns of $U = C^{-1}$ since they satisfy $C\mathbf{u}_i = \mathbf{0}$. Clearly, U is also a unimodular matrix. We then have that

$$K \cap \mathbb{Z}^d = \{ \mathbf{s} = \lambda_1 \mathbf{u}_1 + \lambda_2 \mathbf{u}_2 + \cdots + \lambda_d \mathbf{u}_d \mid \boldsymbol{\lambda} \in \mathbb{Z}_{\geq 0}^d \},$$

and therefore

$$\begin{aligned} f(K; \mathbf{x}) &= \frac{1}{(1 - \mathbf{x}^{\mathbf{u}_1})(1 - \mathbf{x}^{\mathbf{u}_2}) \cdots (1 - \mathbf{x}^{\mathbf{u}_d})} \\ &= \left(\sum_{\lambda_1 \geq 0} (\mathbf{x}^{\mathbf{u}_1})^{\lambda_1} \right) \left(\sum_{\lambda_2 \geq 0} (\mathbf{x}^{\mathbf{u}_2})^{\lambda_2} \right) \cdots \left(\sum_{\lambda_d \geq 0} (\mathbf{x}^{\mathbf{u}_d})^{\lambda_d} \right), \end{aligned} \tag{9}$$

since there is exactly one value of $\boldsymbol{\lambda}$ for each $\mathbf{s} \in K \cap \mathbb{Z}^d$ due to the unimodularity of U .

In the general case, where $\boldsymbol{\gamma}$ is not necessarily zero, $K = \mathbf{v} + \text{pos} \{ \mathbf{u}_i \}$, where $\mathbf{v} = U\boldsymbol{\gamma}$ is the vertex of K . According to Theorem 5.3, translation of a polyhedron by some integer vector \mathbf{m} corresponds to a multiplication of $f(K; \mathbf{x})$ by $\mathbf{x}^{\mathbf{m}}$. However, \mathbf{v} may not be integer. To obtain the correct multiplication factor, we consider the *fundamental parallelepiped*.

Definition 5.5 (Fundamental parallelepiped) *Let $K = \mathbf{v} + \text{pos} \{ \mathbf{u}_i \}$ be a (shifted) cone, then the fundamental parallelepiped Π of K is*

$$\Pi = \mathbf{v} + \left\{ \sum_i \alpha_i \mathbf{u}_i \mid 0 \leq \alpha_i < 1 \right\}.$$

Each (integer) point in K can be written as the sum of an (integer) point in Π and a positive integer combination of the rays \mathbf{u}_i of K . Since K is unimodular, Π contains a single integer point

$$\mathbf{w} = \sum_i \lceil \gamma_i \rceil \mathbf{u}_i = \sum_i (\gamma_i \mathbf{u}_i + \{-\gamma_i\} \mathbf{u}_i) = \mathbf{v} + \sum_i \alpha_i \mathbf{u}_i,$$

where $\lceil \cdot \rceil$ is the least integer function and $\{\cdot\}$ is the fractional part. To see that \mathbf{w} is unique, consider another point \mathbf{w}' in Π , $\mathbf{w}' = \mathbf{v} + \sum_i \lambda'_i \mathbf{u}_i$. Then $\sum (\lambda_i - \lambda'_i) \mathbf{u}_i \in \mathbb{Z}^d$ and so $\sum (\lambda_i - \lambda'_i) \in \mathbb{Z}^d$ or $\lambda_i = \lambda'_i$. We therefore have that

$$f(K; \mathbf{x}) = \frac{\mathbf{x}^{\mathbf{w}}}{(1 - \mathbf{x}^{\mathbf{u}_1})(1 - \mathbf{x}^{\mathbf{u}_2}) \cdots (1 - \mathbf{x}^{\mathbf{u}_d})}, \tag{10}$$

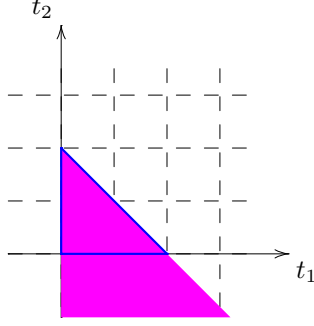


Figure 16: Supporting cone $\text{cone}(T, (0, 2))$ of polytope T at vertex $(0, 2)$.

with

$$\mathbf{w} = \sum_i \lceil \gamma_i \rceil \mathbf{u}_i = - \sum_i \lfloor -\gamma_i \rfloor \mathbf{u}_i. \quad (11)$$

This greatest integer function in the definition of \mathbf{w} is where the greatest integer function in our step-polynomial will come from. Note also that the denominator of this generating function does not depend on γ , only on the \mathbf{c}_i .

5.2.2 Brion's Theorem

We want to reduce our problem, finding the generating function $f(P_{\mathbf{p}} \cap \mathbb{Z}^d; \mathbf{x})$ where $P_{\mathbf{p}}$ is a polyhedron, to the easy problem of finding the generating function for a unimodular cone. The general case of a polyhedron can be reduced to (not necessarily unimodular) cones using Brion's Theorem (Brion 1988) and then the case of general cones can be further reduced to that of unimodular cones by applying Barvinok's unimodular decomposition (Barvinok 1994).

Before we can formulate Brion's theorem, we need to define the concept of a supporting cone.

Definition 5.6 (Supporting cone) Let $\mathbf{v}(\mathbf{p})$ be a (parametric) vertex of a (parametric) polyhedron $Q_{\mathbf{p}}$

$$Q_{\mathbf{p}} = \{ \mathbf{x} \in \mathbb{Q}^d \mid C\mathbf{x} \geq B\mathbf{p} + \mathbf{d} \},$$

with $C \in \mathbb{Z}^{d \times d}$, $B \in \mathbb{Z}^{d \times n}$ and $\mathbf{d} \in \mathbb{Z}^d$. Let $I_{\mathbf{v}} = \{ i \mid \langle \mathbf{c}_i, \mathbf{v}(\mathbf{p}) \rangle = \langle \mathbf{b}_i, \mathbf{p} \rangle + d_i \}$ be the set of active constraints on $\mathbf{v}(\mathbf{p})$. The supporting cone $\text{cone}(Q_{\mathbf{p}}, \mathbf{v}(\mathbf{p}))$ of $Q_{\mathbf{p}}$ at $\mathbf{v}(\mathbf{p})$ is the (shifted) cone bounded by the facets of $Q_{\mathbf{p}}$ that contain $\mathbf{v}(\mathbf{p})$, i.e.,

$$\text{cone}(Q_{\mathbf{p}}, \mathbf{v}(\mathbf{p})) = \{ \mathbf{x} \in \mathbb{Q}^d \mid \langle \mathbf{c}_i, \mathbf{x} \rangle \geq \langle \mathbf{b}_i, \mathbf{p} \rangle + d_i \text{ for } i \in I_{\mathbf{v}} \}.$$

Note that the vertex of $\text{cone}(Q_{\mathbf{p}}, \mathbf{v}(\mathbf{p}))$ depends on the parameters \mathbf{p} , but that its rays do not.

Example 22 Consider the polytope T from Example 21. The constraint active on the vertex $\mathbf{v} = (0, 2)$ are $t_1 \geq 0$ and $t_1 + t_2 \leq 2$. The supporting cone at $\mathbf{v} = (0, 2)$ is therefore

$$\text{cone}(T, \mathbf{v}) = \{ \mathbf{t} \mid t_1 \geq 0 \wedge t_1 + t_2 \leq 2 \}$$

and is shown in Figure 16.

Theorem 5.7 (Brion's Theorem) The generating function of a polyhedron P is equal to the sum of the generating functions of its supporting cones,

$$f(P_{\mathbf{p}}; \mathbf{x}) = \sum_{\mathbf{v}(\mathbf{p}) \text{ a vertex of } P_{\mathbf{p}}} f(\text{cone}(P_{\mathbf{p}}, \mathbf{v}(\mathbf{p})); \mathbf{x}).$$

For a proof of Brion's theorem, we refer to Brion (1988), Barvinok and Pommersheim (1999) or Barvinok (2002). We will limit ourselves to a more intuitive explanation on an example polytope.

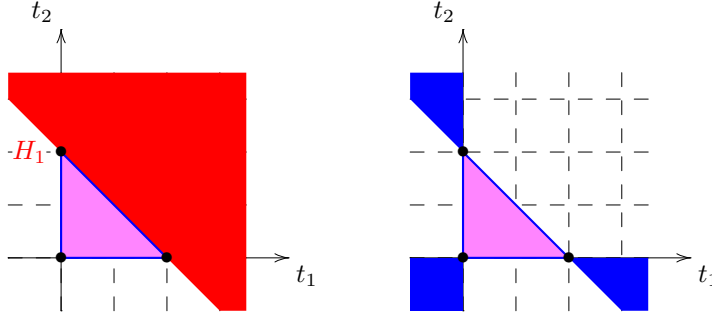


Figure 17: Intuitive explanation of Brion's theorem.

Example 23 Consider the polytope T from Example 21. The indicator function of the whole plane except T is

$$[\mathbb{Q}^2] - [T] = [H_1] + [H_2] + [H_3] - [H_1 \cap H_2] - [H_2 \cap H_3] - [H_3 \cap H_1],$$

where each H_i is the (open) halfspace on the opposite site of T bounded by one of the edges of T . For example, H_1 is shown on the left of Figure 17. When translating this identity to the corresponding generating function, we may ignore indicator functions of polyhedra that contain a line since the corresponding generating function is 0. We therefore have

$$[T] \equiv [H_1 \cap H_2] + [H_2 \cap H_3] + [H_3 \cap H_1] \pmod{\mathcal{L}},$$

where \mathcal{L} is the subspace of indicator functions $[P]$ of polyhedra that contain lines and $f \equiv g \pmod{\mathcal{L}}$ if f is equal to g up to multiples of elements from \mathcal{L} . The intersections of pairs of halfspaces in this formula are shown on the right of Figure 17. The generating function of such an intersection of a pair of halfspaces, as in the formula above, is the same as the generating function of the corresponding supporting cone, e.g.,

$$\begin{aligned} f(H_2 \cap H_3; \mathbf{x}) &= \frac{x_1^{-1} x_2^{-1}}{(1 - x_1^{-1})(1 - x_2^{-1})} \\ &= \frac{1}{(1 - x_1)(1 - x_2)} \\ &= f(\text{cone}(T, \mathbf{v}_1); \mathbf{x}), \end{aligned}$$

where we applied (10) in the first and in the final equality and (3) in the second. Recall that there may be different Laurent power series expansions that converge to the same generating function on different regions of \mathbb{C}^n . The numerator $x_1^{-1} x_2^{-1}$ appears in the first equation because H_2 and H_3 are *open* halfspaces. We finally have

$$f(T; \mathbf{x}) = \sum_i f(\text{cone}(T, \mathbf{v}_i); \mathbf{x}),$$

as desired.

Example 24 Consider the parametric polytope $P_{\mathbf{p}}$ from Example 16. The polytope corresponding to $\mathbf{p} = (3, 4) \in C_3$ is shown in Figure 18 together with the supporting cones at each active vertex. This value for \mathbf{p} corresponds to the top left part of Figure 11. Note that the rays of each supporting cone do not change as long as \mathbf{p} remains within chamber C_3 . Brion's theorem tells us that the generating function for $P_{\mathbf{p}}$ is the sum of the generating functions for these three support cones. Note that the supporting cones at \mathbf{v}_1 , \mathbf{v}_5 and \mathbf{v}_6 have rays $\{(2, 1), (0, -1)\}$, $\{(1, 1), (0, 1)\}$ and $\{(-2, -1), (-1, -1)\}$ respectively and so the supporting cones at \mathbf{v}_5 and \mathbf{v}_6 are unimodular, but the one at \mathbf{v}_1 is not.

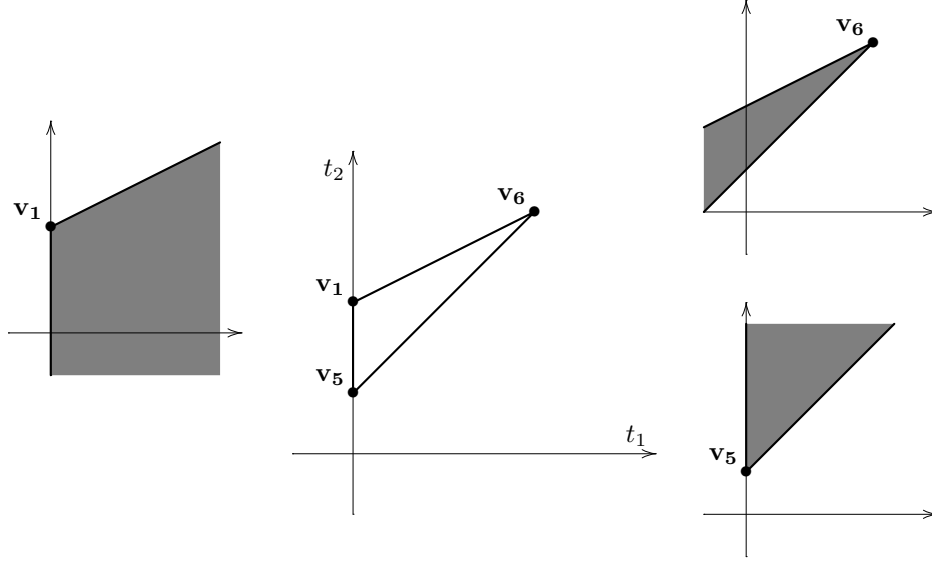


Figure 18: $P_{(3,4)}$ and its supporting cones.

5.2.3 Barvinok's Decomposition

Barvinok's decomposition will decompose a cone $K \subset \mathbb{Q}^d$ into a signed sum of unimodular cones

$$[K] = \sum_{i \in I} \epsilon_i [K_i],$$

with $K_i \subset \mathbb{Q}^d$ unimodular, $\epsilon_i \in \{-1, 1\}$ and $|I|$ bounded by a polynomial in the input size of K . In principle the sum on the right would also contain lower dimensional cones. This can be avoided, though, by using what is known as Brion's polarization trick (Brion 1988). The idea is to compute the decomposition not of K itself but of its polar K^* . This *polar cone* K^* is the set defined by

$$K^* = \{ \mathbf{y} \mid \forall \mathbf{x} \in K : \langle \mathbf{x}, \mathbf{y} \rangle \geq 0 \},$$

where we use the definition of polar cone as also used by e.g., Barvinok and Pommersheim (1999) and Wilde (1993). Other authors, e.g., Grünbaum (1967) and Schrijver (1986) define the polar of K to be $-K^*$.

Example 25 Figure 19 shows the cone

$$K = \{ \mathbf{x} \mid x_1 \geq 0 \wedge x_1 + x_2 \leq 0 \}$$

on the left. The generators of this cone are $\mathbf{r}_1 = (0, -1)$ and $\mathbf{r}_2 = (1, -1)$. If \mathbf{y} is such that $\langle \mathbf{r}_1, \mathbf{y} \rangle \geq 0$ and $\langle \mathbf{r}_2, \mathbf{y} \rangle \geq 0$ then $\langle \mathbf{x}, \mathbf{y} \rangle \geq 0$ for all $\mathbf{x} \in K$ since any element of K is a nonnegative linear combination of \mathbf{r}_1 and \mathbf{r}_2 . The polar cone is therefore

$$K^* = \{ \mathbf{y} \mid -y_2 \geq 0 \wedge y_1 - y_2 \geq 0 \}$$

and is shown on the right of the same figure.

An interesting property of polarization is that any linear equality that holds between a set of cones $K_j \subset \mathbb{Q}^d$ also holds between their polars (Barvinok and Pommersheim 1999, Corollary 2.8), i.e.,

$$\sum_{i=1}^m \alpha_i [K_i] = 0 \Leftrightarrow \sum_{i=1}^m \alpha_i [K_i^*] = 0.$$

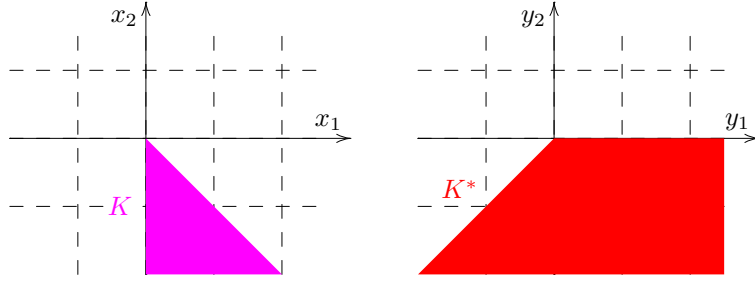


Figure 19: A cone K and its polar K^* .

We can therefore decompose the polar of K into a signed sum of full-dimensional cones K_j and some lower-dimensional cones F ,

$$[K^*] = \sum_j \epsilon_j [K_j] + \sum_F \epsilon_F [F]$$

and polarize back to obtain the decomposition of K itself,

$$[K] = \sum_j \epsilon_j [K_j^*] + \sum_F \epsilon_F [F^*] \equiv \sum_j \epsilon_j [K_j^*] \pmod{\mathcal{L}}$$

where we may ignore the polars of the lower-dimensional cones since the polar of such a cone contains a line. An example of polarization will follow after we have explained Barvinok’s decomposition. Unless otherwise stated we will assume from now on that we are working on the polar of the cone we want to decompose and that we may therefore ignore lower-dimensional parts in the decomposition.

Using `PolyLib`, the polarization of a cone is particularly easy to implement. The polar of a cone is the original cone with the rays and constraints interchanged, both of which are maintained in `PolyLib`’s internal representation of a polyhedron. The function `Polyhedron.Polarize` polarizes its argument, which is assumed to be a cone with apex $\mathbf{0}$, in place by interchanging its rays and constraints.

5.2.4 Triangulation of Non-simplicial Cones

The core of Barvinok’s unimodular decomposition algorithm takes a simplicial cone as input and produces a signed unimodular decomposition of the cone. If a supporting cone is not a (shifted) simplicial cone, then we first need to *triangulate* it. Efficient triangulation algorithms were described by Aurenhammer and Klein (2000) and Lee (1997). De Loera et al. (2004) use the Delaunay triangulation in their `LattE` implementation (De Loera et al. 2003b).

We explain this Delaunay triangulation, or the more general regular triangulation, in some more detail. We follow the exposition of De Loera (1995), which is in turn based on Lee (1991), but apply it to cones instead of polytopes. A *regular triangulation* of a cone $C \subset \mathbb{Q}^d$ is a triangulation obtained through the following procedure. Let $\{\mathbf{u}_i\}_{i \in I}$ be the generators of C . Consider the cone

$$C^\lambda = \text{pos}\{(\mathbf{u}_i, \lambda_i)\} \subset \mathbb{Q}^{d+1},$$

where the extra coordinate λ_i of each generator is called the “height” of the generator. The process of adding such an extra coordinate is called “lifting”. If C^λ is defined by the system of linear inequalities

$$C^\lambda = \{\mathbf{x} \in \mathbb{Q}^{d+1} \mid A\mathbf{x} \geq \mathbf{0}\},$$

then a *lower face* is a face

$$\{\mathbf{x} \in C^\lambda \mid A'\mathbf{x} = \mathbf{0}\},$$

where $A'\mathbf{x} \geq \mathbf{0}$ is a subsystem of $A\mathbf{x} \geq \mathbf{0}$ such that the $(d+1)$ st coordinate $a'_{i,d+1}$ is positive for each row \mathbf{a}'_i in A' . The polyhedral complex formed by the lower faces is called the *lower envelope*. The projection of the lower envelope onto the first d dimensions is a subdivision of C . For generic choices of λ this subdivision will be a triangulation. The *Delaunay triangulation* is a regular triangulation with heights $\lambda_i = \sum_j u_{i,j}^2$.

This triangulation is implemented in the `triangularize_cone` method.⁶ The most straightforward way of computing the lower envelope is to simply compute the implicit representation of the polyhedron generated by the origin and the rays $\{(\mathbf{u}_i, \lambda_i)\}$ using a tool such as `PolyLib`. This implicit representation will contain both the lower and the upper envelope and we then simply ignore the upper envelope. As a minor optimization, our implementation computes the implicit representation of the polyhedron with the additional ray $(\mathbf{0}, 1)$. This means that rather than computing the facets in both the lower and the upper envelope, we only compute the lower facets plus some vertical facets. In general, this decreases the number of facets, especially for higher dimensional cones. We first compute the Delaunay triangulation and if this fails to produce an actual triangulation, we perform a fixed number of attempts with random heights. In principle we should provide a backup procedure that never fails, but in practice we have seen that in all cases the first attempt with random heights yields a valid triangulation.

Example 26 Consider the polytope

$$P = \text{conv} \left\{ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} \right\},$$

shown in Figure 20. The supporting cone at the origin $C = \text{cone}(P, o)$ is not simplicial since it has four extremal rays,

$$C = \text{pos} \left\{ \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \right\}.$$

Let us call these rays a, b, c and d , respectively. The Delaunay triangulation would use heights $\lambda = (1, 2, 3, 2)$, i.e.,

$$C^\lambda = \text{pos} \left\{ \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 0 \\ 2 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 1 \\ 3 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ 1 \\ 2 \end{bmatrix} \right\}.$$

Random heights could be $\lambda' = (6, 3, 5, 5)$, i.e.,

$$C^{\lambda'} = \text{pos} \left\{ \begin{bmatrix} -1 \\ 0 \\ 0 \\ 6 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 0 \\ 3 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 1 \\ 5 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ 1 \\ 5 \end{bmatrix} \right\}.$$

We call the lifted rays a', b', c' and d' .

Figure 21 shows slices of these cone at $x = -1$ and the projection of the lower envelope onto the $t = 0$ plane. Note that the right figure is scaled by $\frac{1}{2}$ in the t dimension. The implicit representations are

$$C^\lambda = \left\{ \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} \mid [1 \quad -1 \quad -1 \quad 1] \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} = 0 \wedge \begin{bmatrix} 0 & -1 & -2 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -2 & -1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} \geq \mathbf{0} \right\}$$

and

$$C^{\lambda'} = \left\{ \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} \mid \begin{bmatrix} 6 & 3 & 1 & 1 \\ -6 & -3 & 2 & -1 \\ -6 & 0 & -1 & -1 \\ 3 & 0 & -2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} \geq \mathbf{0} \right\}.$$

⁶`triangulate_cone` would have been a better name.

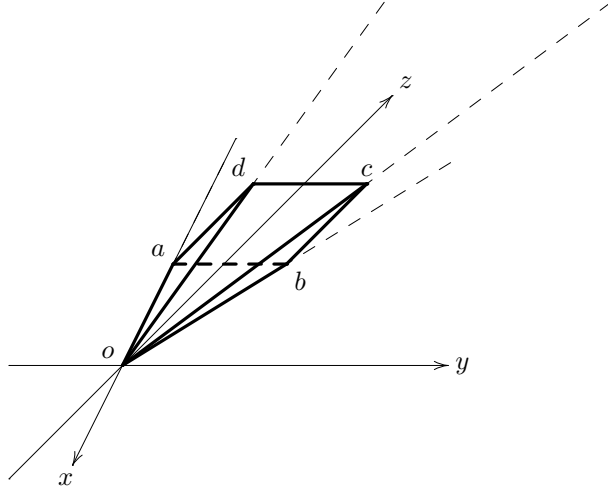


Figure 20: The polytope P from Example 26 in thick lines and the supporting cone at the origin $\text{cone}(P, o)$ in dashed lines.

Note that C^λ is only 3-dimensional and so the lower envelope of C^λ is simply C^λ itself. It is clear that this will not give us the desired triangulation. The cone $C^{\lambda'}$ has two lower facets $[6 \ 3 \ 1 \ 1] \mathbf{x} = 0$ and $[3 \ 0 \ -2 \ 1] \mathbf{x} = 0$, saturating the origin and the rays $\{a', b', d'\}$ and $\{b', c', d'\}$ respectively. Projecting these lower facets onto $t = 0$ yields the desired triangulation.

Consider now the cones C_{\uparrow}^λ and $C_{\uparrow}^{\lambda'}$ with the extra ray, i.e.,

$$C_{\uparrow}^\lambda = \text{pos} \left\{ \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 0 \\ 2 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 1 \\ 3 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ 1 \\ 2 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right\}$$

and

$$C_{\uparrow}^{\lambda'} = \text{pos} \left\{ \begin{bmatrix} -1 \\ 0 \\ 0 \\ 6 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 0 \\ 3 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 1 \\ 5 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ 1 \\ 5 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right\}.$$

These cones are shown in Figure 22. The implicit representations are

$$C_{\uparrow}^\lambda = \left\{ \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} \mid \begin{bmatrix} 1 & -1 & -1 & 1 \\ -1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} \geq \mathbf{0} \right\}$$

and

$$C_{\uparrow}^{\lambda'} = \left\{ \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} \mid \begin{bmatrix} 0 & 0 & 1 & 0 \\ 6 & 3 & 1 & 1 \\ 3 & 0 & -2 & 1 \\ -1 & -1 & 0 & 0 \\ -1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} \geq \mathbf{0} \right\}.$$

Note that both cones are now 4-dimensional and have the same lower envelope as their bounded counterparts. Also note that the number of constraints has increased in this example, but this is atypical.

The resulting triangulation of the supporting cone at the origin is shown in Figure 23.

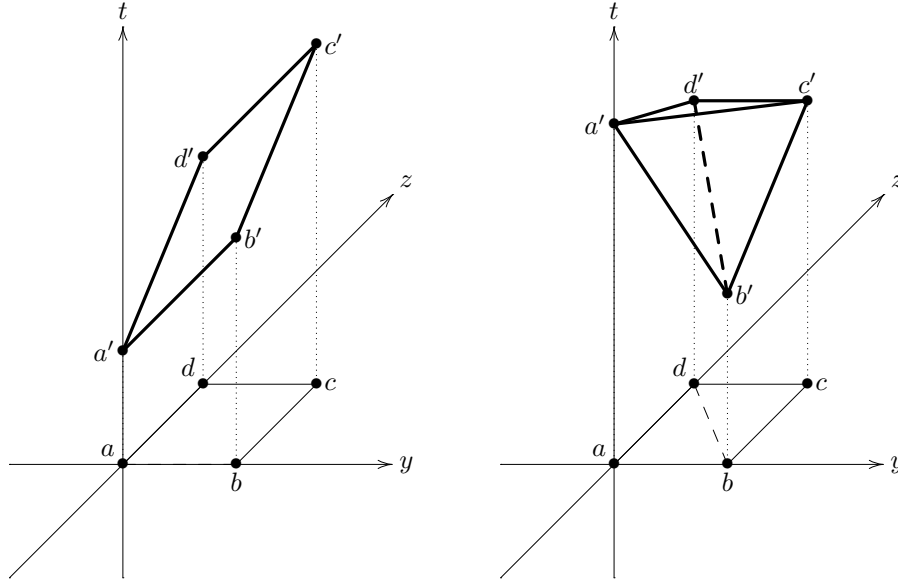


Figure 21: Slices of the cones C^λ (on the left) and $C^{\lambda'}$ (on the right) from Example 26 at $x = -1$ and the projections of their lower envelopes onto the $t = 0$ plane.

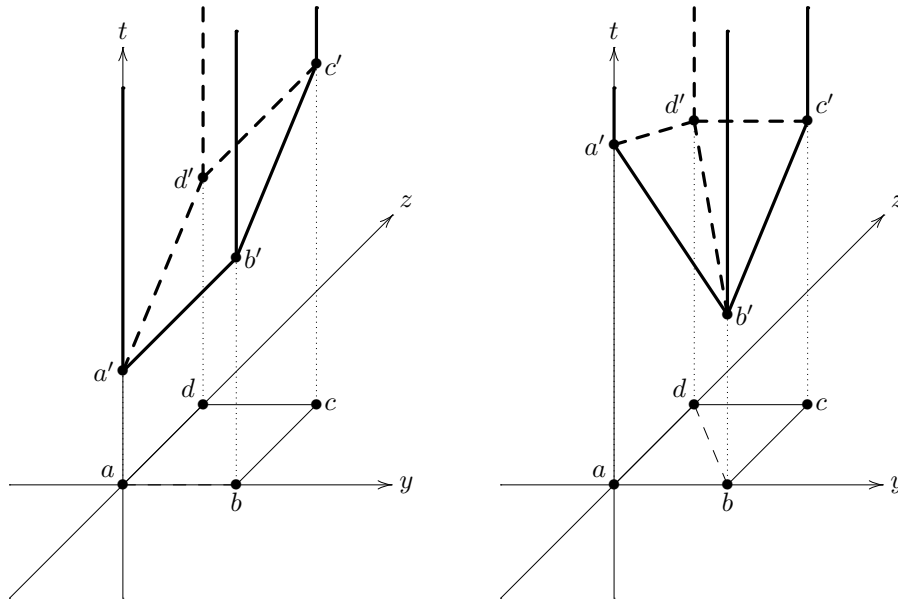


Figure 22: Slices of the cones C_\uparrow^λ (on the left) and $C_\uparrow^{\lambda'}$ (on the right) from Example 26 at $x = -1$ and the projections of their lower envelopes onto the $t = 0$ plane.

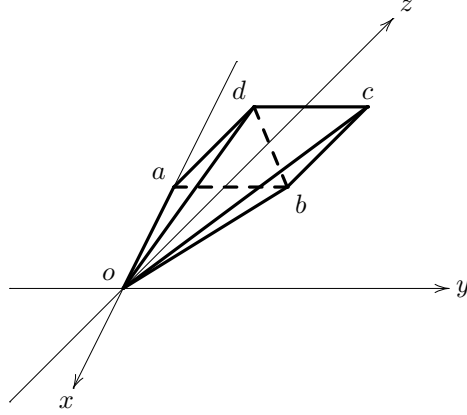


Figure 23: The triangulation of the supporting cone at the origin $\text{cone}(P, o)$ of the polytope P from Example 26.

5.2.5 Decomposition of Simplicial Cones

As mentioned before, Barvinok's decomposition will decompose a (simplicial) cone K , with $K = \text{pos} \{ \mathbf{u}_i \}_{i=1}^d \subset \mathbb{Q}^d$, into a signed sum of unimodular cones

$$[K] = \sum_{i \in I} \epsilon_i [K_i]. \quad (12)$$

With a slight abuse of notations, we let K denote both the cone $K \subset \mathbb{Q}^d$ itself and the matrix $K \in \mathbb{Z}^{d \times d}$ with the d linearly independent generators of the cone K as columns. If K is unimodular, i.e., $|\det K| = 1$, then there is nothing to decompose. Otherwise $|\det K| > 1$ and we decompose K into d cones K_i , each with $|\det K_i| < |\det K|$. Successively decomposing the resulting cones, we obtain the desired result. Barvinok (1994) proves that we may compute the whole decomposition in polynomial time. Here, we will mainly focus on *how* to compute the decomposition. Our exposition of this decomposition process basically follows De Loera et al. (2004).

To perform the decomposition, we need an integer vector \mathbf{w} that has small coefficients when written as a linear combination of the generators of K . In particular, we want

$$\mathbf{w} = \sum_{i=1}^d \alpha_i \mathbf{u}_i \quad |\alpha_i| \leq |\det(K)|^{-1/d}. \quad (13)$$

Replacing a generator of K by such a \mathbf{w} will yield a cone with a smaller determinant. The existence of a suitable \mathbf{w} is guaranteed by the application of Minkowski's First Theorem (Schrijver 1986) to the convex body

$$B = \left\{ \sum_{i=1}^d \alpha_i \mathbf{u}_i \mid \forall i : |\alpha_i| \leq |\det(K)|^{-1/d} \right\}.$$

This convex body is just the closure of the fundamental parallelepiped Π scaled by $2|\det(K)|^{-1/d}$ in each dimension and centered around the origin. The scaling and the fact that the volume of Π is $|\det K|$ ensure that the volume of B is 2^d . Since B is a centrally symmetric convex body with volume $\geq 2^d$, Minkowski's First Theorem tells us that B contains a non-zero integer vector \mathbf{w} , as required.

For $1 \leq j \leq d$, let K_j be the cone generated by the generators of K with \mathbf{u}_j replaced by \mathbf{w} , i.e.,

$$K_j = \text{pos} \left(\{ \mathbf{u}_i \}_{i=1}^d \setminus \{ \mathbf{u}_j \} \cup \{ \mathbf{w} \} \right). \quad (14)$$

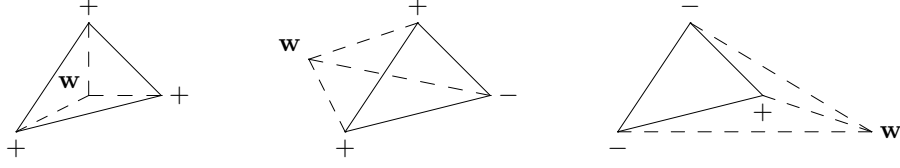


Figure 24: Possible locations of \mathbf{w} with respect to the rays of a 3-dimensional cone. The figure shows a section of the cones.

Then

$$|\det K_j| \leq |\alpha_j| |\det K| \leq |\det K|^{(d-1)/d} < |\det K|.$$

To see that we can use these K_j to perform a decomposition, rearrange the \mathbf{u}_i such that for all $1 \leq i \leq k$ we have $\alpha_i \leq 0$ and for all $k+1 \leq i \leq d$ we have $\alpha_i > 0$. We may assume $k < d$; otherwise replace $\mathbf{w} \in B$ by $-\mathbf{w} \in B$. We have

$$\mathbf{w} + \sum_{i=1}^k (-\alpha_i) \mathbf{u}_i = \sum_{i=k+1}^d \alpha_i \mathbf{u}_i$$

or

$$\sum_{i=0}^k \beta_i \mathbf{u}_i = \sum_{i=k+1}^d \alpha_i \mathbf{u}_i, \quad (15)$$

with $\mathbf{u}_0 = \mathbf{w}$, $\beta_0 = 1$ and $\beta_i = -\alpha_i \geq 0$ for $1 \leq i \leq k$. Any two \mathbf{u}_j and \mathbf{u}_l on the same side of the equality are on opposite sides of the affine hull H of the other \mathbf{u}_i s since there exists a convex combination of \mathbf{u}_j and \mathbf{u}_l on this hyperplane. In particular, since α_j and α_l have the same sign, we have

$$\frac{\alpha_j}{\alpha_j + \alpha_l} \mathbf{u}_j + \frac{\alpha_l}{\alpha_j + \alpha_l} \mathbf{u}_l \in H.$$

The corresponding cones K_j and K_l (with $K_0 = K$) therefore intersect in a common face $F \subset H$. Let

$$K' := \text{pos} \left(\{ \mathbf{u}_i \}_{i=1}^d \cup \{ \mathbf{w} \} \right),$$

then any $\mathbf{x} \in K'$ lies both in some cone K_i with $0 \leq i \leq k$ and in some cone K_i with $k+1 \leq i \leq d$. (Just subtract an appropriate multiple of Equation (15).) The cones $\{K_i\}_{i=0}^k$ and $\{K_i\}_{i=k+1}^d$ therefore both form a triangulation of K' and hence

$$[K'] = [K] + \sum_{i=1}^k [K_i] = \sum_{i=k+1}^d [K_i],$$

where, as usual, we ignore lower-dimensional faces. Figure 24 shows the possible configurations in the case of a 3-dimensional cone.

To find a suitable \mathbf{w} , we essentially use the method proposed by Dyer and Kannan (1997), which is also used by De Loera et al. (2004). Let $A = K^T$, i.e., A is the matrix with the generators of K as rows. From (13) it follows that if we can find an integer vector $\mathbf{w}^T = \boldsymbol{\alpha}^T A$ such that $\|\boldsymbol{\alpha}\|_\infty$ (the L^∞ -norm of $\boldsymbol{\alpha}$) is minimal, i.e., $\max_i |\alpha_i|$ is minimal, then $\mathbf{w} \in B$. Let $\Delta = |\det A|$, then looking for a small rational vector $\boldsymbol{\alpha}^T = \mathbf{w}^T A^{-1}$ with \mathbf{w} integer is equivalent to searching for a small integer vector $\boldsymbol{\lambda} = \Delta \boldsymbol{\alpha}$ with $\boldsymbol{\lambda}^T = \Delta \mathbf{w}^T A^{-1}$. A well-known tool for finding a short integer vector in a lattice is Lenstra, Lenstra and Lovasz' basis reduction algorithm (LLL). Given an integer basis for a lattice, e.g., ΔA^{-1} , it will produce a new integer basis, say A' , for the same lattice, but with "short" basis vectors that are nearly orthogonal (Lenstra et al. 1982; Grötschel et al. 1988; Schrijver 1986; Cohen 1993). Shortness here refers to the L^2 -norm, i.e., the Euclidean norm, rather than the L^∞ -norm, so Dyer and Kannan (1997) propose to search over linear combinations

with small integer coefficients $\boldsymbol{\mu}$ of the reduced basis vectors. Since A' and ΔA^{-1} generate the same lattice, $A' = U(\Delta A^{-1})$ for some unimodular matrix U . We have

$$\begin{aligned}\boldsymbol{\lambda}^T &= \boldsymbol{\mu}^T A' \\ \boldsymbol{\lambda}^T &= \boldsymbol{\mu}^T U(\Delta A^{-1}) \\ \boldsymbol{\lambda}^T &= \Delta \mathbf{w}^T A^{-1} \\ \Delta^{-1} \boldsymbol{\lambda}^T A &= \mathbf{w}^T \\ \boldsymbol{\alpha}^T A &= \mathbf{w}^T,\end{aligned}$$

with $\mathbf{w}^T = \boldsymbol{\mu}^T U$. As reported by De Loera et al. (2004), the search for $\boldsymbol{\mu}$ may be very expensive, however. We therefore only consider values of $\boldsymbol{\mu}$ equal to a standard basis vector \mathbf{e}_j , i.e., we only consider the rows of U as possible values for \mathbf{w}^T . This appears to work very well in practice in the sense that we always obtain a valid reduction. Note that Δ may be any integer value that ensures that ΔA^{-1} is an integer matrix.

The search for a short vector is implemented in `cone::short_vector`. We basically call the LLL procedure from the NTL library (Shoup 2004) with the standard parameters. Given a matrix B with a basis for a lattice as rows, this procedure will modify B to contain the reduced basis and will additionally produce the unimodular transformation matrix U . The standard parameters ensure that the Euclidian length of the first basis vector is no more than 2^{d-1} times that of the shortest vector in the lattice. We then search for the row from the new B with smallest L^∞ -norm and assign it to $\boldsymbol{\lambda}$. The corresponding row from U is assigned to \mathbf{w} . If no element of $\boldsymbol{\lambda}$ is strictly positive, then we replace \mathbf{w} by $-\mathbf{w}$ to ensure that the right-hand side of (15) has at least one term.

The method `decomposer::decompose` computes Barvinok's decomposition and then calls the method `decomposer::handle` on each resulting unimodular cone. The method maintains a list of non-unimodular cones, initialized to the original cone, if it is indeed not unimodular. As long as this list is non-empty, the method calls `cone::short_vector` to obtain both the short vector and the vector $\boldsymbol{\lambda}$. For each $1 \leq j \leq d$ such that $\lambda_j \neq 0$ it constructs the cone K_j (14) and either adds it to the list or sends it to `decomposer::handle`. If $\lambda_j = 0$, then \mathbf{w} contains no contribution of \mathbf{u}_j and so K_j is not of full dimension and may be ignored. To obtain the final sign ϵ_i in the decomposition (12), note that if (and only if) α_j is negative during a single decomposition then the signs of $\det K$ and $\det K_j$ differ. The final sign is then simply the sign of the original cone multiplied by the sign of the unimodular cone. Note that in our implementation, $\boldsymbol{\mu}$ usually, but not always, ends up being equal to \mathbf{e}_1 .

Example 27 For a (trivial) example where $\boldsymbol{\mu} \neq \mathbf{e}_1$, consider the cone with as generators the rows of

$$A = \begin{bmatrix} -2 & 1 \\ 2 & 3 \end{bmatrix}.$$

Let

$$B = (\det A) A^{-1} = -8A^{-1} = \begin{bmatrix} 3 & -1 \\ -2 & -2 \end{bmatrix}.$$

LLL yields

$$B' = \begin{bmatrix} 3 & -1 \\ -2 & -2 \end{bmatrix} \quad U = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

In this case, the second row of B' is smaller than the first in both the L^2 and L^∞ -norm and we choose $\boldsymbol{\lambda}^T = [-2 \quad -2]$. Note that $\boldsymbol{\alpha}^T = \Delta^{-1} \boldsymbol{\lambda}^T = \frac{1}{8} [-2 \quad -2] = [-\frac{1}{4} \quad -\frac{1}{4}]$ and its coefficients satisfy the constraint (13) as $\frac{1}{4} \leq \frac{1}{\sqrt{8}}$.

For a slightly less trivial example, consider the matrix

$$B = \begin{bmatrix} 238 & 0 & 0 & -119 \\ -476 & -1 & 102 & 71 \\ -238 & 0 & 119 & 0 \\ -357 & -1 & 102 & 71 \end{bmatrix}.$$

LLL now yields

$$B' = \begin{bmatrix} 0 & -7 & 0 & 21 \\ 0 & 15 & 17 & 6 \\ 0 & -20 & 17 & -8 \\ 119 & 0 & 0 & 0 \end{bmatrix} \quad U = \begin{bmatrix} 4 & -1 & -6 & 8 \\ -9 & 1 & 13 & -16 \\ 12 & -2 & -17 & 22 \\ 0 & -1 & 0 & 1 \end{bmatrix}.$$

Here, the first row of B' is shorter than the second row in L^2 -norm ($\sqrt{490} < \sqrt{550}$), but larger in L^∞ -norm ($21 > 17$). Note that the first coordinate of $\boldsymbol{\lambda} = [0 \ 15 \ 17 \ 6]^T$ is zero and so the decomposition at this level consists of three cones rather than four.

Example 28 Continuing on this theme, consider the cone with as generators the rows of

$$A = \begin{bmatrix} 6 & 1 & -6 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}.$$

Let

$$B = 6A^{-1} = \begin{bmatrix} 1 & -6 & -1 \\ 0 & 0 & 6 \\ 0 & -6 & 0 \end{bmatrix}.$$

LLL yields

$$B' = \begin{bmatrix} -1 & 0 & 1 \\ 3 & 0 & 3 \\ 0 & -6 & 0 \end{bmatrix} \quad U = \begin{bmatrix} -1 & 0 & 1 \\ 3 & 1 & -3 \\ 0 & 0 & 1 \end{bmatrix}.$$

We have $\boldsymbol{\lambda} = [-1 \ 0 \ 1]^T$ and $\mathbf{w} = [-1 \ 0 \ 1]^T$. The decomposition at this level therefore consists of two cones rather than three. It is clear that replacing the second row of A by \mathbf{w}^T would yield a singular matrix and, hence, a lower-dimensional cone.

5.2.6 Overview

Having described Barvinok's decomposition into unimodular cones, we now return to the problem of computing the generating function of a (possibly parametric) polytope, which, according to Brion's Theorem, is equal to the sum of the generating functions of its supporting cones. Computing the generating function of a supporting cone $\text{cone}(P_{\mathbf{p}}, \mathbf{v}(\mathbf{p}))$ is simply a matter of translating to the origin, $K = \text{cone}(P_{\mathbf{p}}, \mathbf{v}(\mathbf{p})) - \mathbf{v}(\mathbf{p})$, computing the unimodular decomposition (12) of K ,

$$[K] = \sum_{i \in I} \epsilon_i [K_i],$$

translating back to $\mathbf{v}(\mathbf{p})$

$$[\text{cone}(P_{\mathbf{p}}, \mathbf{v}(\mathbf{p}))] = \sum_{i \in I} \epsilon_i [K_i + \mathbf{v}(\mathbf{p})]$$

and constructing the corresponding generating functions for the (shifted) unimodular cones as in (10). Note that this is equivalent to first computing the generating functions (9) of the K_i and then multiplying each with the appropriate $\mathbf{x}^{\mathbf{w}^i}$ according to (11), where $\boldsymbol{\gamma}$ is such that

$$\mathbf{v} = \sum_i \gamma_i \mathbf{u}_i = K\boldsymbol{\gamma}, \tag{16}$$

i.e.,

$$\boldsymbol{\gamma}(\mathbf{p}) = K^{-1}\mathbf{v}(\mathbf{p}). \tag{17}$$

Since the coordinates of $\mathbf{v}(\mathbf{p})$ are affine combinations of the parameters \mathbf{p} , then so are the coordinates of $\boldsymbol{\gamma}$. Hence, \mathbf{w} is a step-polynomial in \mathbf{p} of degree one. We have the following proposition, a rephrasing of Theorem 4.4 of Barvinok and Pommersheim (1999).

Algorithm 1 Barvinok's algorithm

1. For each vertex $\mathbf{v}_i(\mathbf{p})$ of $P_{\mathbf{p}}$
 - (a) Determine supporting cone $\text{cone}(P_{\mathbf{p}}, \mathbf{v}_i(\mathbf{p}))$
 - (b) Let $K = \text{cone}(P_{\mathbf{p}}, \mathbf{v}_i(\mathbf{p})) - \mathbf{v}_i(\mathbf{p})$
 - (c) Decompose K into unimodular cones $\{\epsilon_j, K_j\}$
 - i. Polarize K into K^*
 - ii. Triangulate K^*
 - iii. Decompose each simplicial cone in triangulation
 - iv. Polarize back
 - (d) For each K_j
 - i. Determine $f(K_j; \mathbf{x})$
 - (e) $f(\text{cone}(P_{\mathbf{p}}, \mathbf{v}_i(\mathbf{p})); \mathbf{x}) = \sum_j \epsilon_j \mathbf{x}^{E(\mathbf{v}_i(\mathbf{p}), K_j)} f(K_j; \mathbf{x})$
 2. For each chamber C of $P_{\mathbf{p}}$
 - (a) $f(P_{\mathbf{p}}; \mathbf{x}) = \sum_{\mathbf{v}_i \in C} f(\text{cone}(P_{\mathbf{p}}, \mathbf{v}_i(\mathbf{p})); \mathbf{x})$
 - (b) Evaluate $f(P_{\mathbf{p}}; \mathbf{1})$
-

Proposition 5.8 *Fix d . There exists a polynomial time algorithm, which, given a parametric polyhedron $P \subset \mathbb{Q}^n \times \mathbb{Q}^d$ and a polyhedral chamber C such that for $\mathbf{p} \in C$ the vertices of $P_{\mathbf{p}} = \{\mathbf{t} \in \mathbb{Q}^d \mid (\mathbf{p}, \mathbf{t}) \in P\}$ are given by affine transformations $T_1(\mathbf{p}), T_2(\mathbf{p}), \dots, T_m(\mathbf{p})$, computes the generating functions*

$$f(P_{\mathbf{p}} \cap \mathbb{Z}^d; \mathbf{x}) = \sum_{i \in I} \epsilon_i \frac{\mathbf{x}^{\mathbf{w}_i(\mathbf{p})}}{(1 - \mathbf{x}^{\mathbf{b}_{i1}})(1 - \mathbf{x}^{\mathbf{b}_{i2}}) \dots (1 - \mathbf{x}^{\mathbf{b}_{id}})},$$

where $\epsilon \in \{-1, 1\}$, $\mathbf{b}_{ij} \in \mathbb{Z}^d \setminus \{0\}$, and each coordinate of $\mathbf{w}_i(\mathbf{p}) : \mathbb{Z}^n \rightarrow \mathbb{Z}^d$ is a step-polynomial of degree one, for each i .

An overview of Barvinok's algorithm, applied to parametric polytopes, is shown in Algorithm 1. The function $E(\mathbf{v}(\mathbf{p}), K)$ in step 1e returns the step-polynomial that determines the unique point in the fundamental parallelepiped of $\mathbf{v}(\mathbf{p}) + K$. The notation $\mathbf{v}_i \in C$ in step 2a means that vertex \mathbf{v}_i is active in chamber C . The evaluation in step 2b is explained in Section 5.3.

Example 29 We continue with the parametric polytope $P_{\mathbf{p}}$ from Examples 16 and 24. In particular, we consider chamber C_3 (see Figure 11). The supporting cones in this chamber are as shown in Figure 18. Recall that the supporting cones at \mathbf{v}_5 and \mathbf{v}_6 are unimodular, but that the one at \mathbf{v}_1 is not. We therefore need to apply Barvinok's unimodular decomposition to the cone $K = \text{cone}(P_{\mathbf{p}}, \mathbf{v}_1) - \mathbf{v}_1$.

We first consider a "primal decomposition", i.e., without polarization. The generators of K are $(2, 1)$ and $(0, -1)$, i.e.,

$$A = \begin{bmatrix} 2 & 1 \\ 0 & -1 \end{bmatrix}.$$

We have $\Delta = |\det A| = |\det K| = 2$ and so

$$\Delta A^{-1} = \begin{bmatrix} 1 & 1 \\ 0 & -2 \end{bmatrix}.$$

LLL yields

$$\begin{bmatrix} 1 & 1 \\ 0 & -2 \end{bmatrix} = UA' = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

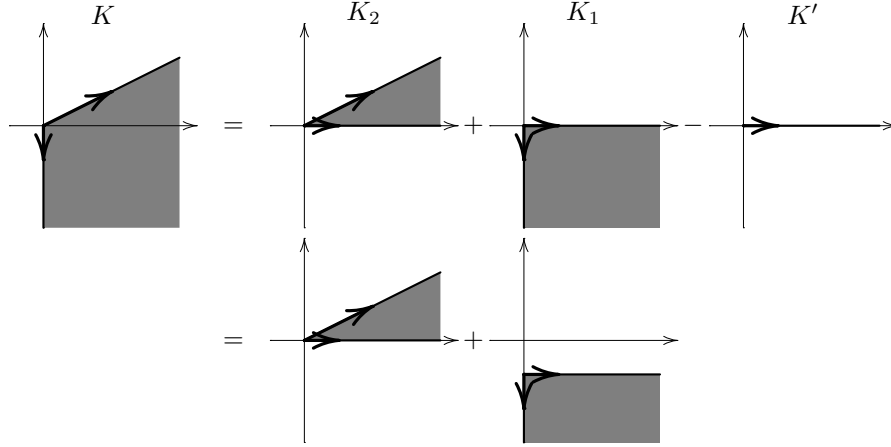


Figure 25: Primal Unimodular Decomposition.

Both rows of A' have the same L^∞ -norm, so we arbitrarily choose the first row $\lambda^T = [1 \ 1]$, i.e., $\mu = \mathbf{e}_1$, and the corresponding row from U is $\mathbf{w}^T = [1 \ 0]$. We obtain

$$[K] = [K_1] + [K_2] - [K_1 \cap K_2],$$

with the generators of K_1 , K_2 and $K' = K_1 \cap K_2$

$$K_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad K_2 = \begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix} \quad \text{and} \quad K' = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

This unimodular decomposition is shown in Figure 25, with K_2 on the left of K_1 for reasons that will become clear when we consider the dual decomposition. Note that we cannot ignore the lower-dimensional cone in this case since we are computing the primal decomposition. The generating function for K is then

$$f(K; \mathbf{x}) = \frac{1}{(1 - \mathbf{x}^{(1,0)})(1 - \mathbf{x}^{(2,1)})} + \frac{1}{(1 - \mathbf{x}^{(1,0)})(1 - \mathbf{x}^{(0,-1)})} - \frac{1}{1 - \mathbf{x}^{(1,0)}}$$

which can be simplified to

$$f(K; \mathbf{x}) = \frac{1}{(1 - \mathbf{x}^{(1,0)})(1 - \mathbf{x}^{(2,1)})} + \frac{\mathbf{x}^{(0,-1)}}{(1 - \mathbf{x}^{(1,0)})(1 - \mathbf{x}^{(0,-1)})}. \quad (18)$$

The final row in Figure 25 shows a “decomposition” that corresponds directly to this generating function.

Let us now consider the dual decomposition, shown in Figure 26. The second row shows the original cone K on the left-hand side. The polar of this cone is

$$K^* = \text{pos} \left\{ \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\},$$

which is shown on the left-hand side of the top row and which we now decompose to obtain the right-hand side. We have

$$A = \begin{bmatrix} 1 & -2 \\ 1 & 0 \end{bmatrix}.$$

We again have $\Delta = |\det A| = |\det K| = 2$ and so

$$\Delta A^{-1} = \begin{bmatrix} 0 & 2 \\ -1 & 1 \end{bmatrix}.$$

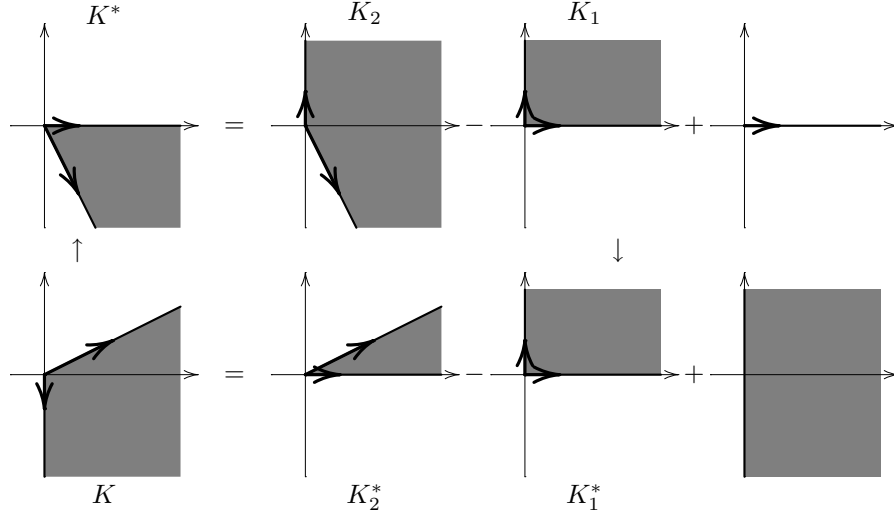


Figure 26: Dual Unimodular Decomposition.

LLL now yields

$$\begin{bmatrix} 0 & 2 \\ -1 & 1 \end{bmatrix} = UA' = \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}.$$

Again, both rows of A' have the same L^∞ -norm, so we arbitrarily choose the first row $\lambda^T = [-1 \ 1]$, i.e., $\mu = \mathbf{e}_1$, and the corresponding row from U is $\mathbf{w}^T = [0 \ 1]$. We obtain

$$[K^*] \equiv [K_2] - [K_1],$$

with the generators of K_1 and K_2

$$K_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \text{and} \quad K_2 = \begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix}.$$

In the figure we have also shown the lower-dimensional intersection, but this can be ignored since polarizing back yields a cone containing a line, as shown in the second row of the figure. We finally have

$$[K] \equiv [K_2^*] - [K_1^*],$$

with

$$K_1^* = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad K_2^* = \begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix}.$$

The generating function for K is then

$$f(K; \mathbf{x}) = \frac{1}{(1 - \mathbf{x}^{(1,0)})(1 - \mathbf{x}^{(2,1)})} - \frac{1}{(1 - \mathbf{x}^{(1,0)})(1 - \mathbf{x}^{(0,1)})}, \quad (19)$$

which is equivalent to (18) through application of (3).

To obtain the generating function of $K + \mathbf{v}_1 = \text{cone}(P_s, \mathbf{v}_1)$, Equation (19) still needs to be translated to $\mathbf{v}_1(\mathbf{p}) = (0, -p_1/2 + p_2)$ using (11). From (17) we have

$$\mathbf{v}_1 = \begin{bmatrix} 0 \\ -\frac{p_1}{2} + p_2 \end{bmatrix} = \gamma_1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \gamma_2 \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \gamma.$$

We have

$$\gamma = \begin{bmatrix} 1 & -2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ -\frac{p_1}{2} + p_2 \end{bmatrix} = \begin{bmatrix} p_1 - 2p_2 \\ -\frac{p_1}{2} + p_2 \end{bmatrix}.$$

Note that this corresponds to

$$K_2^* + \mathbf{v}_1 = \left\{ \mathbf{t} \in \mathbb{Q}^2 \mid t_2 \geq -\frac{p_1}{2} + p_2 \wedge t_1 - 2t_2 \geq p_1 - 2p_2 \right\}.$$

We conclude from (11) that

$$\mathbf{w} = -\lfloor -\gamma_1 \rfloor \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \lfloor -\gamma_2 \rfloor \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} -2\lfloor \frac{p_1}{2} - p_2 \rfloor + p_1 - 2p_2 \\ -\lfloor \frac{p_1}{2} - p_2 \rfloor \end{bmatrix}.$$

Handling the second term similarly, we obtain

$$f((K + \mathbf{v}_1); \mathbf{x}) = \frac{\mathbf{x}^{(-2\lfloor \frac{p_1}{2} - p_2 \rfloor + p_1 - 2p_2, -\lfloor \frac{p_1}{2} - p_2 \rfloor)}}{(1 - \mathbf{x}^{(1,0)})(1 - \mathbf{x}^{(2,1)})} - \frac{\mathbf{x}^{(0, -\lfloor \frac{p_1}{2} - p_2 \rfloor)}}{(1 - \mathbf{x}^{(1,0)})(1 - \mathbf{x}^{(0,1)})}. \quad (20)$$

5.3 Evaluating Generating Functions

Now that we know how to compute $f(P_{\mathbf{s}}; \mathbf{x})$, all that remains is to evaluate it at $\mathbf{x} = \mathbf{1}$. We use the following lemma, which is a special case of the monomial substitution theorem (Barvinok and Woods 2003, Theorem 2.6). We provide a slightly different proof, which lends itself more easily to an implementation. It is an extension of an idea from De Loera et al. (2004), which is in itself a variation of the idea used by Barvinok (1994). De Loera et al. (2004) only consider the case of a full evaluation, i.e., the case where m in the lemma is 0.

Lemma 5.9 (Specialization) *Let us fix k . There exists a polynomial time algorithm which, given a rational generating function $f(\mathbf{x})$ of the form (7), with $k_i \leq k$ and $|I|$ bounded by a polynomial, and an m with $0 \leq m \leq n$ such that $g(\mathbf{z}) := f(z_1, \dots, z_m, 1, \dots, 1)$ is an analytic function on some nonempty open subset of \mathbb{C}^m , computes $g(\mathbf{z})$ in the same form, i.e.,*

$$g(\mathbf{z}) = \sum_{i \in I'} \beta_i \frac{\mathbf{z}^{\mathbf{w}_i}}{\prod_{j=1}^{k'_i} (1 - \mathbf{z}^{\mathbf{d}_{ij}})} \quad (21)$$

where $k'_i \leq k$, $|I'|$ is bounded by a polynomial, $\mathbf{z} \in \mathbb{C}^m$, $\beta_i \in \mathbb{Q}$, $\mathbf{w}_i \in \mathbb{Z}^m$, and $\mathbf{d}_{ij} \in \mathbb{Z}^m \setminus \{\mathbf{0}\}$.

Furthermore, if the vectors \mathbf{b}_{ij} and the numbers α_i in (7) are fixed, but the vectors \mathbf{q}_i vary, then the vectors \mathbf{d}_{ij} are fixed, \mathbf{w}_i each differ by a constant vector from some \mathbf{q}_i , and β_i are each a polynomial of degree at most k in the coordinates of some \mathbf{q}_i .

Proof The case $m = n$ is trivial, so we will assume $m < n$. Note that we cannot simply plug in the values 1, since $(z_1, \dots, z_m, 1, \dots, 1)$ may be a pole of some of the terms in (7). In fact, if $m = 0$, then it will be a pole of all those terms. For each n -dimensional vector \mathbf{v} , we write \mathbf{v}' for the first m components of \mathbf{v} and \mathbf{v}'' for the remaining $n - m$ components. Consider

$$h(t) = f(z_1, \dots, z_m, (t+1)^{\lambda_1}, \dots, (t+1)^{\lambda_{n-m}}),$$

as a function of t only (i.e., $\mathbf{z} = (z_1, \dots, z_m)$ are treated as symbolic constants), where $\boldsymbol{\lambda} \in \mathbb{Z}^{n-m}$ is such that for each $i \in I$ and $j \leq k_i$, either $\mathbf{b}'_{ij} \neq \mathbf{0}$ or $\langle \mathbf{b}''_{ij}, \boldsymbol{\lambda} \rangle \neq 0$. Such a $\boldsymbol{\lambda}$ can be found in polynomial time by choosing an appropriate point from the ‘‘moment curve’’ as do Barvinok and Pommersheim (1999, Algorithm 5.2). This *moment curve* is $m(\tau) = (1, \tau, \tau^2, \dots, \tau^{n-1})$. For each of the \mathbf{b}_{ij} the inner product with $m(\tau)$ is a polynomial of degree at most $n - 1$ and thus has at most $n - 1$ zeros. Therefore, enumerating the points on the moment curve $m(0), m(1), \dots, m(k)$ will yield a valid $\boldsymbol{\lambda}$ for k at most $n(n - 1)|I| + 1$.

The function $g(\mathbf{z})$ is then simply the constant term in the Laurent power series expansion of

$$h(t) = \sum_{i \in I} h_i(t)$$

about $t = 0$. This is the sum of the constant terms in the Laurent power series expansions of

$$h_i(t) = \alpha_i \frac{\mathbf{z}^{\mathbf{q}'_i} (t+1)^{a_i}}{\prod_{j=1}^{k_i} (1 - \mathbf{z}^{\mathbf{b}'_{ij}} (t+1)^{v_{ij}})},$$

where we let $a_i = \langle \mathbf{q}'_i, \boldsymbol{\lambda} \rangle \in \mathbb{Z}$, and $v_{ij} = \langle \mathbf{b}'_{ij}, \boldsymbol{\lambda} \rangle \in \mathbb{Z}$. It only remains to show that this constant term is indeed of the form (21) and that we can compute all of these terms in polynomial time.

Consider a particular $h_i(t)$. For ease of notation, we will omit the i subscript on newly introduced variables. Partition the factors in the denominator based on the variables that appear in each factor. Let r be the number of factors with $v_{ij} \neq 0$ but $\mathbf{b}'_{ij} = \mathbf{0}$ and s the number of factors with both $v_{ij} \neq 0$ and $\mathbf{b}'_{ij} \neq \mathbf{0}$. The remaining $k_i - r - s$ factors have $v_{ij} = 0$. In the special space that $r = 0$ (in particular, this requires $m \neq 0$), then $t = 0$ is not a pole and we can simply plug in 0 for t . We obtain a single term in the sum of the form (21) with $k'_i = k_i$. Otherwise, $h_i(t)$ has a pole of order r at $t = 0$ and we must compute the coefficient of t^r in the Taylor series expansion of $t^r h_i(t)$, which is analytic at $t = 0$. Reorder the factors if necessary and write

$$h_i(t) = \frac{C(\mathbf{z})(t+1)^{a_i}}{\prod_{j=1}^r ((t+1)^{\beta_j} - 1) \prod_{j=r+1}^{r+s} ((t+1)^{\gamma_j} - \mathbf{z}^{\boldsymbol{\alpha}_j})} = C(\mathbf{z}) \frac{P(t)}{t^r \prod Q_j(t; \mathbf{z})},$$

where $C(\mathbf{z})$ is a symbolic constant that collects the factors in both numerator and denominator that are independent of t and where we have multiplied both numerator and denominator with either -1 or $-\mathbf{z}^{\boldsymbol{\alpha}_j} = -\mathbf{z}^{\mathbf{b}'_{ij}}$ for some j' to ensure that all powers in the denominator β_j and γ_j are positive. That is, we applied (3) on those factors with $v_{ij} < 0$. This is needed to ensure that the expansions of each $h_i(t)$ converge on a common region.

Following De Loera et al. (2004), we use the technique outlined by Henrici (1974, 241–247) (who applies it to compute the residue of a function, i.e., the coefficient of the term t^{-1}) to compute the coefficient of t^r in $P(t)/\prod Q_j(t; \mathbf{z})$. To compute the coefficients c_l in

$$\frac{P(t)}{Q(t)} =: c_0 + c_1 t + c_2 t^2 + \dots,$$

expand $P(t)$ and $Q(t)$ as

$$\begin{aligned} P(t) &=: a_0 + a_1 t + a_2 t^2 + \dots \\ Q(t) &=: b_0 + b_1 t + b_2 t^2 + \dots \end{aligned}$$

and apply the recurrence relation

$$c_l = \frac{1}{b_0} \left(a_l - \sum_{i=1}^l b_i c_{l-i} \right).$$

To compute the coefficient of t^r in $P(t)/\prod Q_j(t; \mathbf{z})$, we apply the above for each factor in the denominator, in each iteration replacing $P(t)$ by the result of the previous iteration. Note that we only need to keep track of the first $r+1$ coefficients. First divide by the first r factors, which are independent of \mathbf{z} . The constant terms of the remaining factors are of the form $1 - \mathbf{z}^{\boldsymbol{\alpha}_j}$. Only expressions of this kind will ever appear in a denominator. After the first division, the largest power in a denominator of $c_l^{[1]}$ is $(1 - \mathbf{z}^{\boldsymbol{\alpha}_j})^{r+1}$. Each subsequent division increases the total power of all factors in the denominator by one. The total power of factors in the denominator of $c_l^{[s]}$ will therefore be $r+s$ and so remains constant. The number of terms in $c_l^{[s]}$ is also clearly bounded by a constant $s(k)$, i.e., $|I'| \leq s(k)|I|$.

Note that based on the binomial theorem (see, e.g., Graham et al. 1989)

$$(1+t)^r = \sum_{j=0}^r \binom{r}{j} t^j$$

the coefficients of the numerator $P(t)$ are polynomial expressions in a_i , which is itself a linear combination of the coefficients of \mathbf{q}_i . The maximal degree of such a polynomial expression is $r \leq k$. The lemma is proved. \square

Note that as argued by De Loera et al. (2004), a $\boldsymbol{\lambda}$ from the moment curve may not be the most appropriate choice to use in an implementation since it is likely to have large coefficients. They therefore propose to construct a random vector with small coefficients and check whether $\langle \mathbf{b}_{ij}'' , \boldsymbol{\lambda} \rangle \neq 0$ for all i and j . (Or rather $\langle \mathbf{b}_{ij} , \boldsymbol{\lambda} \rangle \neq 0$, since $m = 0$ in their case.) Only after a fixed number of failed attempts would the implementation fall back onto the moment curve.

Both of these strategies have the disadvantage however that all the terms in (7) need to be available before the constant term of the first term can be computed. This may induce a large memory bottleneck. The authors of `LattE` have therefore also implemented an alternative strategy where a random vector with larger coefficients is constructed at the beginning of the computation (Yoshida 2004b). If the coefficients are large enough, then the probability of having constructed an incorrect vector is close to zero. The disadvantage of this technique is that the coefficients are larger and that the computation has to be redone completely in the unlikely event the vector was incorrect.

We propose a different strategy which does not require all terms to be available, nor does it require the use of large coefficients. We simply repeatedly apply Lemma 5.9 for m' from $n - 1$ down to m . In each application, we can simply use $\lambda = 1$, which is known to be valid in any case.

Versions of `barvinok` up to version 0.11 implemented the method using the random vector with small coefficients. Version 0.16 and newer support both the method using the random vector with large coefficients and the incremental approach for enumerating parametric polytopes. The latter is the default and may be switched off using the `--disable-incremental` configure option. The effect on memory usage can be dramatic in some cases. For example, enumerating the non-parametric polytope `long_4D`, included in the distribution, using version 0.11 requires nearly 170 MB, while version 0.16 needs less than 1.6 MB. For small-dimensional problems, the difference between the large random vector and the incremental approach is small, both in computation time and in the final resulting step-polynomial.

Example 30 Consider the rational generating function (8) from Example 21

$$f(z, x) = \frac{x^2}{(1-x^{-1})(1-x^{-1}z)} + \frac{z^2}{(1-z^{-1})(1-xz^{-1})} + \frac{1}{(1-x)(1-z)},$$

where we renamed x_1 and x_2 to x and z respectively to fit the notation of the lemma. We want to compute $g(z) = f(z, 1)$. Substituting $x = t + 1$, we obtain

$$f(z, t+1) = \frac{(t+1)^2}{(1-(t+1)^{-1})(1-(t+1)^{-1}z)} + \frac{z^2}{(1-z^{-1})(1-(t+1)z^{-1})} + \frac{1}{(1-(t+1))(1-z)}.$$

In the notation of the proof, the final term has $r = 1$ and $s = 0$. Since the coefficient of t^1 in the numerator is 0, the contribution of this term is 0. The second term has $r = 0$ and so we can simply plug in 1 to obtain

$$\frac{z^2}{(1-z^{-1})^2}.$$

The first term has $r = 1$ and $s = 1$ and has negative powers of t in the denominator, so we first rewrite it using (3) to

$$\frac{(t+1)^4}{t(t+(1-z))}.$$

The numerator is

$$(t+1)^4 \equiv 1 + 4t \pmod{t^2},$$

i.e., $a_0 = 1$ and $a_1 = 4$, while the denominator has $b_0 = 1 - z$ and $b_1 = 1$. We find

$$c_0 = \frac{1}{1-z} \quad \text{and} \quad c_1 = \frac{1}{1-z} \left(4 - \frac{1}{1-z} \right).$$

Vertex \mathbf{v}_i	Constant term of $f((K_{ij} + \mathbf{v}_i); (t+1, t+1))$
$\mathbf{v}_1 = (0, -p_1/2 + p_2)$	$\frac{p_1^2}{6} + \frac{p_1 p_2}{3} - p_1 \left\lfloor \frac{p_1}{2} \right\rfloor - \frac{p_1}{2} - \frac{p_2}{3} + \left\lfloor \frac{p_1}{2} \right\rfloor^2 + \left\lfloor \frac{p_1}{2} \right\rfloor + \frac{2}{9}$ $-\frac{1}{2} \left\lfloor \frac{p_1}{2} \right\rfloor^2 - \frac{p_2^2}{2} + \left\lfloor \frac{p_1}{2} \right\rfloor p_2 - \frac{1}{2} \left\lfloor \frac{p_1}{2} \right\rfloor + \frac{p_2}{2}$
$\mathbf{v}_2 = (0, 0)$	0
$\mathbf{v}_3 = (p_1 - p_2, 0)$	$-\frac{p_1^2}{4} + \frac{p_1 p_2}{2} - \frac{p_2^2}{4} + \frac{1}{8}$
$\mathbf{v}_4 = (p_1 - 2p_2, 0)$	$\frac{p_1^2}{6} - \frac{2p_1 p_2}{3} - \frac{p_1}{2} + \frac{2p_2^2}{3} + p_2 + \frac{2}{9}$
$\mathbf{v}_5 = (0, -p_1 + p_2)$	$\frac{p_1^2}{4} - \frac{p_1 p_2}{2} + \frac{p_1}{2} + \frac{p_2^2}{4} - \frac{p_2}{2} + \frac{1}{8}$
$\mathbf{v}_6 = (p_1, p_2)$	$\frac{p_1^2}{6} + \frac{p_1 p_2}{6} + \frac{p_1}{2} + \frac{p_2^2}{12} - \frac{p_2}{2} + \frac{47}{72}$

Table 1: The contribution of each supporting cone to the constant term of the Laurent expansion of $f(P_{\mathbf{p}}, (t+1, t+1))$ about $t = 1$.

Summing the three contributions, we obtain

$$g(z) = 0 + \frac{z^2}{(1-z^{-1})^2} + \left(\frac{4}{1-z} + \frac{1}{(1-z)^2} \right).$$

Example 31 We continue with Equation (20) from Example 29 on page 40:

$$f((K + \mathbf{v}_1); \mathbf{x}) = \frac{\mathbf{x}^{(-2\lfloor \frac{p_1}{2} \rfloor - p_2 + p_1 - 2p_2, -\lfloor \frac{p_1}{2} \rfloor - p_2)}}{(1 - \mathbf{x}^{(1,0)})(1 - \mathbf{x}^{(2,1)})} - \frac{\mathbf{x}^{(0, -\lfloor \frac{p_1}{2} \rfloor - p_2)}}{(1 - \mathbf{x}^{(1,0)})(1 - \mathbf{x}^{(0,1)})}.$$

We first apply Lemma 5.9 with $m = 0$, i.e., we compute the value $f((K + \mathbf{v}_1); \mathbf{1})$ by performing a suitable variable substitution $x_i = (1+t)^{\lambda_i}$ and computing the constant term in the corresponding Laurent series at $t = 0$. Since $(1, 1)$ is not orthogonal to any of the powers in the denominator, we can use the substitution $\mathbf{x} = (t+1, t+1)$. Applied to the second term in (20), we obtain

$$-\frac{(1+t)^{-\lfloor \frac{p_1}{2} \rfloor - p_2}}{(1 - (1+t))(1 - (1+t))}.$$

Since the denominator, in this case, is exactly t^2 , the constant term in the Laurent expansion is simply the coefficient of t^2 in the expansion of the numerator, i.e.,

$$-\frac{(-\lfloor \frac{p_1}{2} \rfloor + p_2)(-\lfloor \frac{p_1}{2} \rfloor + p_2 - 1)}{2} = -\frac{1}{2} \left\lfloor \frac{p_1}{2} \right\rfloor^2 - \frac{p_2^2}{2} + \left\lfloor \frac{p_1}{2} \right\rfloor p_2 - \frac{1}{2} \left\lfloor \frac{p_1}{2} \right\rfloor + \frac{p_2}{2}.$$

The first term and the other vertices are handled similarly. The results, the generating function for each supporting cone, are listed in Table 1. The final step-polynomial in each chamber is computed using Brion's Theorem as the sum of the appropriate step-polynomials from this table. The final result is shown in Figure 27.

Let us now apply Lemma 5.9 incrementally. We first substitute $t+1$ for x_1 in the second term of (20) and obtain

$$-\frac{x_2^{-\lfloor \frac{p_1}{2} \rfloor + p_2}}{(1 - (t+1))(1 - x_2)}.$$

Since the denominator is exactly t here and t does not appear in the numerator, the contribution of this term is 0. This is not very insightful, so let us also look at the first term of (20). Performing the same substitution, we obtain

$$\frac{(t+1)^{-2\lfloor \frac{p_1}{2} \rfloor + p_1} x_2^{-\lfloor \frac{p_1}{2} \rfloor + p_2}}{(1 - (t+1))(1 - (t+1)^2 x_2)} = \frac{(t+1)^{-2\lfloor \frac{p_1}{2} \rfloor + p_1} x_2^{-\lfloor \frac{p_1}{2} \rfloor + p_2 - 1}}{t(t^2 + 2t + (1 - x_2^{-1}))}.$$

Again we need to compute the coefficient of t . We have $a_0 = 1$, $a_1 = -2\lfloor \frac{p_1}{2} \rfloor + p_1$, $b_0 = 1 - x_2^{-1}$ and $b_1 = 2$. We find

$$c_0 = \frac{1}{1 - x_2^{-1}} \quad \text{and} \quad c_1 = \frac{1}{1 - x_2^{-1}} \left(-2 \left\lfloor \frac{p_1}{2} \right\rfloor + p_1 - \frac{2}{1 - x_2^{-1}} \right).$$

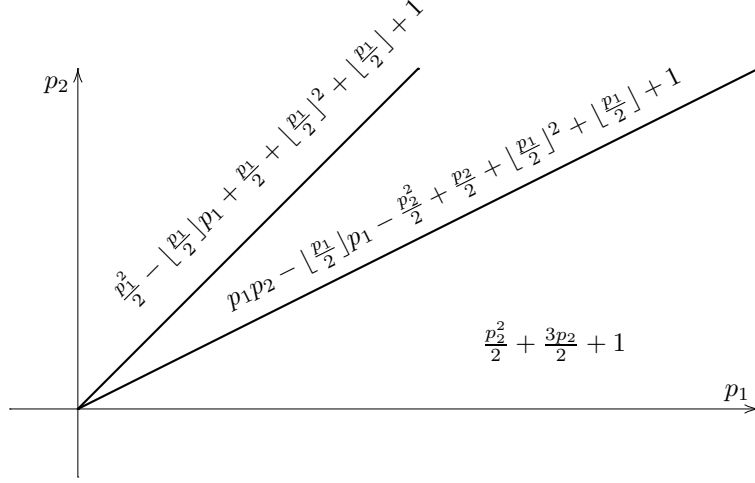


Figure 27: The enumerator of $P_{\mathbf{p}}$, a step-polynomial in each chamber.

Vertex \mathbf{v}_i	Constant term of $f((K_{ij} + \mathbf{v}_i); (t+1, u+1))$
$\mathbf{v}_1 = (0, -p_1/2 + p_2)$	$p_1p_2 - p_1 \lfloor \frac{p_1}{2} \rfloor - p_2^2 - p_2 + \lfloor \frac{p_1}{2} \rfloor^2 + \lfloor \frac{p_1}{2} \rfloor$
$\mathbf{v}_2 = (0, 0)$	0
$\mathbf{v}_3 = (p_1 - p_2, 0)$	0
$\mathbf{v}_4 = (p_1 - 2p_2, 0)$	0
$\mathbf{v}_5 = (0, -p_1 + p_2)$	$\frac{p_1^2}{2} - p_1p_2 + \frac{p_1}{2} + \frac{p_2^2}{2} - \frac{p_2}{2}$
$\mathbf{v}_6 = (p_1, p_2)$	$\frac{p_2^2}{2} + \frac{3p_2}{2} + 1$

Table 2: The contribution of each supporting cone to the constant term of the Laurent expansion of $f(P_{\mathbf{p}}, (t+1, u+1))$ about $t=1$ and $u=1$.

Partial specialization therefore yields

$$\left(-2 \lfloor \frac{p_1}{2} \rfloor + p_1\right) \frac{x_2^{-\lfloor \frac{p_1}{2} \rfloor + p_2 - 1}}{1 - x_2^{-1}} - 2 \frac{x_2^{-\lfloor \frac{p_1}{2} \rfloor + p_2 - 1}}{(1 - x_2^{-1})^2}.$$

Performing a second substitution $x_2 = u+1$ we obtain

$$\left(-2 \lfloor \frac{p_1}{2} \rfloor + p_1\right) \frac{(u+1)^{-\lfloor \frac{p_1}{2} \rfloor + p_2}}{u} - 2 \frac{(u+1)^{-\lfloor \frac{p_1}{2} \rfloor + p_2 + 1}}{u^2}$$

and the contribution of this term is therefore

$$\left(-2 \lfloor \frac{p_1}{2} \rfloor + p_1\right) \left(-\lfloor \frac{p_1}{2} \rfloor + p_2\right) - 2 \frac{\left(-\lfloor \frac{p_1}{2} \rfloor + p_2 + 1\right) \left(-\lfloor \frac{p_1}{2} \rfloor + p_2\right)}{2},$$

as shown in the first row of Table 2. The other rows are obtained similarly. The final result is the same as in the first case and is still shown in Figure 27.

5.4 Enumeration of Parametric Polytopes

Combining the results from Section 5.2 and 5.3, we have the following two propositions describing the enumerators of a parametric polytope, either as a piecewise step-polynomial or as a rational generating function. The first is a combination of Barvinok and Pommersheim (1999, Theorem 5.3)

and Proposition 3.10 (or Barvinok and Pommersheim 1999, Theorem 10.6; see below) and was also discussed by Verdoolaege et al. (2004b). The second is an immediate consequence of results from Barvinok and Pommersheim (1999).

Proposition 5.10 *Fix n and d . There is a polynomial time algorithm which, given a parametric polytope $P \subset \mathbb{Q}^n \times \mathbb{Q}^d$, computes the piecewise step-polynomial*

$$c(\mathbf{p}) = \#(P_{\mathbf{p}} \cap \mathbb{Z}^d)$$

with degree at most d .

Proof Given a parametric polytope $P \subset \mathbb{Q}^n \times \mathbb{Q}^d$, apply Proposition 3.10 to obtain the chamber decomposition $\{C_i\}$. For each chamber C_i , apply Proposition 5.8 to obtain the corresponding generating function of $P_{\mathbf{p}}$, for $\mathbf{p} \in C_i$. The result is a collection of polyhedral regions C_i such that, for $\mathbf{p} \in C_i$,

$$f(P_{\mathbf{p}} \cap \mathbb{Z}^d; \mathbf{x}) = \sum_j \frac{\mathbf{x}^{\mathbf{w}_j(\mathbf{p})}}{(1 - \mathbf{x}^{\mathbf{u}_{j1}})(1 - \mathbf{x}^{\mathbf{u}_{j2}}) \cdots (1 - \mathbf{x}^{\mathbf{u}_{jd}})},$$

where $\mathbf{u}_{jl} \in \mathbb{Z}^d \setminus \{\mathbf{0}\}$ and the coordinates of $\mathbf{w}_j : \mathbb{Z}^n \rightarrow \mathbb{Z}^d$ are piecewise step-polynomials of degree one. All that remains is to use Lemma 5.9 with $m = 0$ to compute $c_i(\mathbf{p}) := f(P_{\mathbf{p}} \cap \mathbb{Z}^d; \mathbf{1})$ as a step-polynomial in \mathbf{p} , and we have $c(\mathbf{p}) = c_i(\mathbf{p})$, for $\mathbf{p} \in C_i$. \square

Proposition 5.11 *Fix n and d . There is a polynomial time algorithm which, given a parametric polytope $P \subset \mathbb{Q}^n \times \mathbb{Q}^d$ such that*

$$f(\mathbf{x}) = \sum_{\mathbf{p} \in \mathbb{Z}^n} c(\mathbf{p}) \mathbf{x}^{\mathbf{p}}$$

converges on some nonempty open subset of \mathbb{C}^n , computes $f(\mathbf{x})$ as a rational generating function of the form (7) with the k_i at most $n + d$.

Proof Given a parametric polytope $P \subset \mathbb{Q}^n \times \mathbb{Q}^d$, apply Theorem 4.4 of Barvinok and Pommersheim (1999) (see Proposition 5.8) directly on P (that is, not considering P as a parametric polytope but as a polyhedron in its own right) to obtain the rational generating function

$$f(P \cap \mathbb{Z}^{n+d}; \mathbf{x}, \mathbf{y}) = \sum_{(\mathbf{p}, \mathbf{t}) \in P \cap \mathbb{Z}^{n+d}} \mathbf{x}^{\mathbf{p}} \mathbf{y}^{\mathbf{t}}$$

in polynomial time. Then

$$\sum_{\mathbf{p} \in \mathbb{Z}^n} c(\mathbf{p}) \mathbf{x}^{\mathbf{p}} = f(P \cap \mathbb{Z}^{n+d}; \mathbf{x}, \mathbf{1}).$$

We may perform this substitution $\mathbf{y} = \mathbf{1}$ in polynomial time using Lemma 5.9. The result is in the form (7). \square

For completeness, we now sketch an alternative for obtaining a result similar to that of Proposition 5.10. This alternative is based on Barvinok and Pommersheim (1999, Theorem 10.6), who propose to compute a partition Q_j of \mathbb{Q}^n and then to compute the rational generating function of the enumerator of the parametric polytope $P_{\mathbf{p}}$ on each Q_j . As in Proposition 5.10, an explicit function may then be obtained from this result by specializing each of these rational generating functions at $\mathbf{1}$.

Rather than directly computing the parametric vertices, Barvinok and Pommersheim (1999, Theorem 10.6) propose the following algorithm to obtain the partition with corresponding rational generating functions. As in the proof of Proposition 5.11, compute the generating function of $P \subset \mathbb{Q}^{n+d}$

$$f(P \cap \mathbb{Z}^{n+d}; \mathbf{x}, \mathbf{y}) = \sum_{(\mathbf{p}, \mathbf{t}) \in P \cap \mathbb{Z}^{n+d}} \mathbf{x}^{\mathbf{p}} \mathbf{y}^{\mathbf{t}} = \sum_{\mathbf{p} \in \mathbb{Z}^n} \sum_{\mathbf{t} \in P_{\mathbf{p}} \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{p}} \mathbf{y}^{\mathbf{t}}.$$

The formula above shows that to obtain the generating function of $P_{\mathbf{p}}$, i.e.,

$$\sum_{\mathbf{t} \in P_{\mathbf{p}} \cap \mathbb{Z}^d} \mathbf{y}^{\mathbf{t}}$$

we need to select the terms with a fixed value of \mathbf{p} and then to specialize at $\mathbf{x} = \mathbf{1}$. Applying identity (3) if needed, we may assume that all terms in (7) converge on a common open set of \mathbb{C}^{n+d} . Let $k := n + d$ and $\mathbf{z} := (\mathbf{x}, \mathbf{y})$. We have

$$f(P; \mathbf{z}) = \sum_{i \in I} \alpha_i \sum_{\boldsymbol{\mu} \in \mathbb{N}^k} \mathbf{z}^{\mathbf{w}_i + B_i \boldsymbol{\mu}},$$

with B_i the matrix with \mathbf{b}_{ij} as columns. To select only those terms with a fixed power \mathbf{p} for the first n variables, we intersect \mathbb{N}^k with $A_i(\mathbf{p}) = \{\boldsymbol{\mu} \mid \pi_n(\mathbf{w}_i + B_i \boldsymbol{\mu}) = \mathbf{p}\}$. The generating function of $P_{\mathbf{p}}$ is then the specialization at $\mathbf{x} = \mathbf{1}$ of

$$\sum_{i \in I} \alpha_i \sum_{\boldsymbol{\mu} \in \mathbb{N}^k \cap A_i(\mathbf{p})} \mathbf{z}^{\mathbf{w}_i + B_i \boldsymbol{\mu}}.$$

To compute this generating function, we compute for each i the generating function

$$f(A_i(\mathbf{p}) \cap \mathbb{Q}_{\geq 0}^k; \mathbf{u}) = \sum_{\boldsymbol{\mu} \in \mathbb{N}^k \cap A_i(\mathbf{p})} \mathbf{u}^{\boldsymbol{\mu}}$$

using Proposition 5.8, perform the substitution $u_j = \mathbf{z}^{\mathbf{b}_{ij}}$ and multiply the result with $\mathbf{z}^{\mathbf{w}_i}$. The substitution is safe since B_i is a unimodular matrix by construction. The end result is a (possibly) different generating function for each element in the ‘‘common refinement’’ of the chamber decompositions of each $A_i(\mathbf{p}) \cap \mathbb{Q}_{\geq 0}^k$.

Example 32 Consider the parametric polytope

$$P_p = \{t \mid t \geq 0 \wedge 2t \leq p + 6 \wedge t \leq p\}.$$

This polytope is shown in Figure 28 for different values of p . Since P_p is a one-dimensional parametric polytope, each of its vertices is determined by a single constraint, whereas the projections of the edges onto the parameter space $t = 0$ yield the chamber decomposition. The vertices, their domains of activity, the generating function of their supporting cones and the corresponding Laurent coefficients are shown in Table 3. The results are comparable to those from Example 20. The complete piecewise step-polynomial is then

$$c_P = \begin{cases} p + 1 & \text{if } 0 \leq p \leq 6 \\ \lfloor \frac{p}{2} \rfloor + 4 & \text{if } 6 \leq p \end{cases}. \quad (22)$$

Now consider the corresponding polyhedron

$$P = \{(p, t) \mid t \in P_p\} = \{(p, t) \mid t \geq 0 \wedge 2t \leq p + 6 \wedge t \leq p\}.$$

The generating function of P is

$$f(P; (z, x)) = \frac{z^0 x^0}{(1 - z^1)(1 - z^1 x^1)} + \frac{z^6 x^6}{(1 - z^{-1} x^{-1})(1 - z^2 x^1)},$$

where the z variable corresponds to the p dimension and the x variable to the t dimension. (Note that both supporting cones are unimodular.) Specialization is particularly easy in this example and we obtain the final rational generating function

$$C_P(z) = f(P; (z, 1)) = \frac{z^0}{(1 - z^1)^2} + \frac{z^6}{(1 - z^{-1})(1 - z^2)}.$$

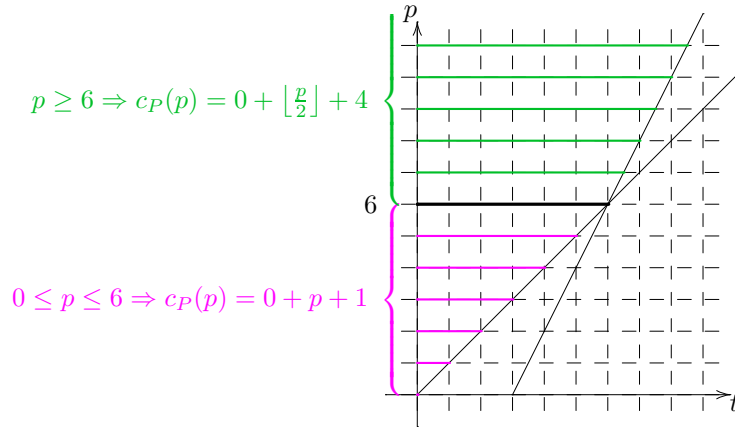


Figure 28: One-dimensional Example.

Constraint	$t \geq 0$	$t \leq p$	$2t \leq p + 6$
Vertex v	0	p	$\frac{p}{2} + 3$
Active if	$0 \leq p$	$0 \leq p \leq 6$	$6 \leq p$
$f(\text{cone}(P_p, v); x)$	$\frac{x^0}{1-x}$	$\frac{x^p}{1-x^{-1}}$	$\frac{x^{\lfloor \frac{p}{2} \rfloor + 3}}{1-x^{-1}}$
Laurent coefficient	0	$p + 1$	$\lfloor \frac{p}{2} \rfloor + 4$

Table 3: Construction of the piecewise step-polynomial from Example 32.

To verify that this rational generating function indeed corresponds to the explicit function in (22), we expand about $z = 0$ and obtain

$$\begin{aligned}
C_P(z) &= 1 + 2z + 3z^2 + 4z^3 + 5z^4 + 6z^5 + 7z^6 + 8z^7 + 9z^8 + 10z^9 + \\
&\quad 11z^{10} + 12z^{11} + 13z^{12} + 14z^{13} + 15z^{14} + 16z^{15} + \dots \\
&\quad + \\
&\quad -z^7 - z^8 - 2z^9 - 2z^{10} - 3z^{11} - 3z^{12} - 4z^{13} - 4z^{14} + \dots \\
&= 1 + 2z + 3z^2 + 4z^3 + 5z^4 + 6z^5 + 7z^6 + 7z^7 + 8z^8 + 8z^9 + \\
&\quad 9z^{10} + 9z^{11} + 10z^{12} + 10z^{13} + 11z^{14} + 11z^{15} + \dots
\end{aligned}$$

as expected.

Just for completeness, let us also look at the alternative way of obtaining an explicit function. The result is essentially the same, but the computation process is slightly more complicated. Again, we start from the generating function of P , but we need to make all terms converge on a common region first. We apply (3) and obtain

$$f(P; (z, x)) = \frac{z^0 x^0}{(1-z^1)(1-z^1 x^1)} - \frac{z^7 x^7}{(1-z^1 x^1)(1-z^2 x^1)}.$$

The first term has

$$\mathbf{v}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \text{and} \quad \mathbf{B}_1 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

and so

$$\begin{aligned}
A_1 \cap \mathbb{Q}_{\geq 0}^2 &= \left\{ \boldsymbol{\mu} \in \mathbb{Q}_{\geq 0}^2 \mid [1 \ 0] \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \boldsymbol{\mu} \right) = p \right\} \\
&= \left\{ \boldsymbol{\mu} \in \mathbb{Q}^2 \mid \mu_1 + \mu_2 = p \wedge \mu_1 \geq 0 \wedge \mu_2 \geq 0 \right\}.
\end{aligned}$$

This set is shown on the left of Figure 29 and is non-empty as long as $p \geq 0$. Its generating function (for $p \geq 0$) is

$$f(A_1 \cap \mathbb{Q}_{\geq 0}^2; \mathbf{u}) = \frac{u_1^0 u_2^p}{1 - u_1^1 u_2^{-1}} + \frac{u_1^p u_2^0}{1 - u_1^{-1} u_2^1}.$$

Substituting

$$\mathbf{u} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \mathbf{y},$$

i.e., $u_1 = y_1^1 y_2^0$ and $u_2 = y_1^1 y_2^1$ we obtain

$$\frac{y_1^p y_2^p}{1 - y_1^0 y_2^{-1}} + \frac{y_1^p}{1 - y_1^0 y_2^1}.$$

Multiplication with $y_1^0 y_2^0$ has no effect and specialization at $y_1 = 1$ yields

$$\frac{y_2^p}{1 - y_2^{-1}} + \frac{1}{1 - y_2^1}. \tag{23}$$

Now let us look at the second term. We have

$$\mathbf{v}_2 = \begin{bmatrix} 7 \\ 7 \end{bmatrix} \quad \text{and} \quad \mathbf{B}_2 = \begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix}$$

and so

$$\begin{aligned}
A_2 \cap \mathbb{Q}_{\geq 0}^2 &= \left\{ \boldsymbol{\mu} \in \mathbb{Q}_{\geq 0}^2 \mid [1 \ 0] \left(\begin{bmatrix} 7 \\ 7 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix} \boldsymbol{\mu} \right) = p \right\} \\
&= \left\{ \boldsymbol{\mu} \in \mathbb{Q}^2 \mid 7 + \mu_1 + 2\mu_2 = p \wedge \mu_1 \geq 0 \wedge \mu_2 \geq 0 \right\}.
\end{aligned}$$

This set is shown on the right of Figure 29 and is non-empty as long as $p \geq 7$. Its generating function (for $p \geq 7$) is

$$f(A_2 \cap \mathbb{Q}_{\geq 0}^2; \mathbf{u}) = \frac{u_1^{p-2} \lfloor \frac{p-7}{2} \rfloor - 7 \lfloor \frac{p-7}{2} \rfloor}{1 - u_1^2 u_2^{-1}} + \frac{u_1^{p-7} u_2^0}{1 - u_1^{-2} u_2^1}.$$

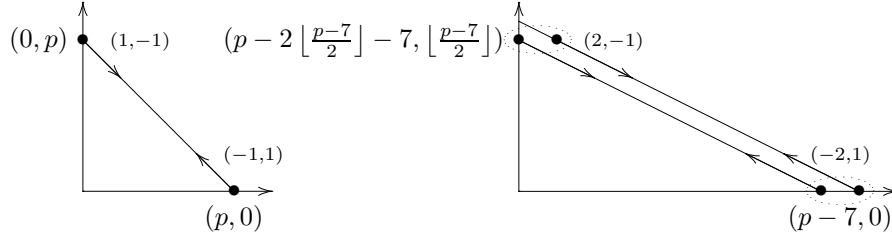


Figure 29: Intersection sets $A_1 \cap \mathbb{Q}_{\geq 0}^2$ and $A_2 \cap \mathbb{Q}_{\geq 0}^2$ for the alternative way in Example 32 .

Substituting

$$\mathbf{u} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} \mathbf{y},$$

i.e., $u_1 = y_1^1 y_2^1$ and $u_2 = y_1^2 y_2^1$ we obtain

$$\frac{y_1^{p-7} y_2^{p - \lfloor \frac{p-7}{2} \rfloor - 7}}{1 - y_1^0 y_2^1} + \frac{y_1^{p-7} y_2^{p-7}}{1 - y_1^1 y_2^{-1}}.$$

Multiplication with $-y_1^7 y_2^7$ and specialization at $y_1 = 1$ yields

$$-\frac{y_2^{p - \lfloor \frac{p-7}{2} \rfloor}}{1 - y_2^1} - \frac{y_2^p}{1 - y_2^{-1}}. \quad (24)$$

Combining the two terms, we have for $0 \leq p \leq 6$ simply Equation (23) and for $p \geq 7$ the sum of Equation (23) and Equation (24), i.e.,

$$f(P_p; y_2) = \frac{1}{1 - y_2^1} - \frac{y_2^{p - \lfloor \frac{p-7}{2} \rfloor}}{1 - y_2^{-1}}.$$

Specializing these rational generating functions at $y_2 = 1$ yields a function that is equivalent to the piecewise step-polynomial in (22).

6 Operations

In this section, we consider some operations that we may perform on enumerators, both piecewise step-polynomials and rational generating functions, of parametric sets. In particular, we consider the addition and multiplication of enumerators, operations that correspond to set operations on the corresponding sets, the summation of an enumerator over part of the parameter space, the interconversion of piecewise step-polynomials and rational generating functions and the evaluation of an enumerator at a particular parameter value.

6.1 Addition

We start with the simplest operation, the addition of two or more enumerators. That is, given the enumerators of two or more sets, we want to know the total number of points these sets. Note that this may be different from the number of points in the union of the sets, if the sets intersect. Addition of piecewise step-polynomials and of rational generating functions may both be performed in polynomial time. It follows that the same holds for any \mathbb{Q} -linear combination of piecewise step-polynomials or rational generating functions.

The sum of a set of rational generating functions can trivially be computed in polynomial time. The index set in the summation of the sum is simply the union of the index sets in the summations of the terms. For piecewise step-polynomials we need the following lemma.

Lemma 6.1 *Fix d . There is a polynomial time algorithm which, given piecewise step-polynomials $c_i : \mathbb{Z}^d \rightarrow \mathbb{Q}$, computes $c(\mathbf{s}) = \sum_i c_i(\mathbf{s})$ as a piecewise step-polynomial.*

Proof Suppose $c_i(\mathbf{s})$ are given as piecewise step-polynomials, and let $c(\mathbf{s}) = \sum_i c_i(\mathbf{s})$. We would like to compute $c(\mathbf{s})$ as a piecewise step-polynomial. For each i , let $\{\langle \mathbf{a}_{ij}, \mathbf{x} \rangle \leq b_{ij}\}_j$ be the collection of linear inequalities that define the chambers of the piecewise step-polynomial representation of $c_i(\mathbf{s})$. By Lemma 3.9, we can compute in polynomial time all cells in \mathbb{Q}^n determined by the collection of all inequalities $\{\langle \mathbf{a}_{ij}, \mathbf{x} \rangle \leq b_{ij}\}_{i,j}$. These are subsets of the full-dimensional chambers in the piecewise step-polynomial representation of $c(\mathbf{s})$. Within a particular chamber, each $c_i(\mathbf{s})$ is defined by

$$c_i(\mathbf{s}) = \sum_{j=1}^{n_i} \alpha_{ij} \prod_{k=1}^{d_{ij}} \lfloor \langle \mathbf{a}_{ijk}, \mathbf{s} \rangle + b_{ijk} \rfloor,$$

where $\alpha_{ij} \in \mathbb{Q}$, $\mathbf{a}_{ijk} \in \mathbb{Q}^d$, and $b_{ijk} \in \mathbb{Q}$, and so $c(\mathbf{s}) = \sum_i c_i(\mathbf{s})$ in this chamber is simply of sum of such functions. \square

We must be careful here how we represent a chamber complex. One possibility is to keep track of all chambers, that is, the full-dimensional chambers as well as all their (common) faces. In this case, the chambers may be considered as open polyhedra and then the associated step-polynomial is only valid in this open chamber. Using this representation, there is no problem with the lemma above. This is also conceptually the nicest option, but it is not ideal from an implementation perspective.

Another option is to only keep track of the full-dimensional chambers. In this case, the chambers should be considered as closed polyhedra. Otherwise, the piecewise step-polynomial would not be defined on common faces, i.e., on lower-dimensional chambers. This means that the step-polynomials on adjacent chambers should be equivalent when restricted to their common faces. This holds for the result of Proposition 5.10, but it may not hold for the result of the lemma above if the “outer walls” of all chamber complexes in the sum do not coincide. If a full-dimensional chamber of one chamber complex is cut by an outer wall of another chamber complex, then of the two resulting adjacent chambers, one will have a contribution to the associated step-polynomial from both complexes, whereas the other will only have a contribution from the first complex, since the second piece-wise step-polynomial is 0 outside the complex. Since the step-polynomial associated to the outer chamber of the second complex is not necessarily 0 on its outer wall, the two chambers in the new complex may have *different* step-polynomials on their common face. One way of solving this issue is to shift all outer walls such that they do not include any integer points (obviously without adding any new integer points to the complex). This ensures that the two adjacent chambers in the new complex will not have conflicting associated step-polynomials since their common face will not include any integer points.

A third option is to insulate the outer chambers with double glazing, keeping the inner pane at room temperature and the outer pane at zero. When computing the common refinement of two chamber complexes, the space in between is then simply ignored. Technically, the result is no longer a chamber complex, but we can still ensure that each integer point is inside at least one full-dimensional chamber. In our implementation, we ensure that each integer point is inside *exactly* one full-dimensional “chamber”. That is, we actually compute a *partition* of the integer points. The sum of two piecewise step-polynomials is implemented in `eadd_partitions`. In our implementation, we also allow chambers to be unions of polyhedra. That is, if C and D are chambers from different complexes such that $C \setminus D$ is not a polyhedron, then we still allow $C \setminus D$ to be a chamber. Otherwise we would have to compute a subdivision of $C \setminus D$.

Example 33 Consider the two chamber complexes, each with a single chamber, in the top row of Figure 30. The next two rows show three valid common refinements corresponding to the different choices of representing chamber complexes listed above, i.e., the first maintains a piecewise step-polynomial for each chamber in the chamber complex; the second only maintains piecewise step-polynomials for the full-dimensional chambers, but ensures that the intersection of two possibly conflicting chambers does not contain any integer points; and the

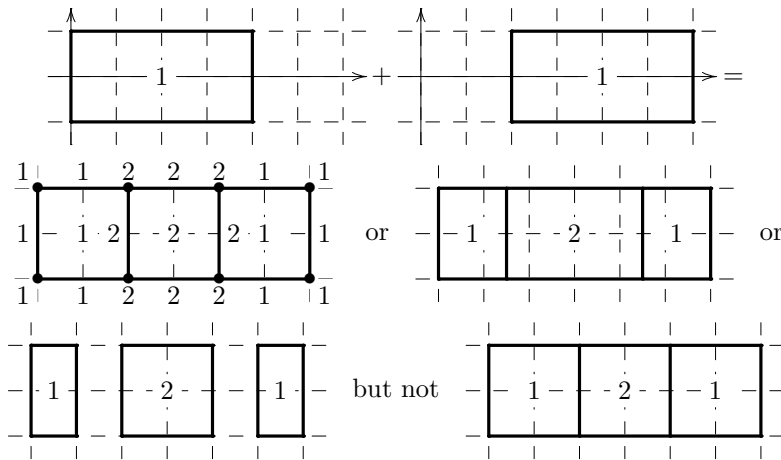


Figure 30: Common refinement of chamber complexes with different outer walls.

third maintains a partition of the integer points. The final case, which also maintains only full-dimensional chambers but allows them to intersect in integer points, is invalid since the function on the common inner wall is 1 or 2 depending on which of the intersecting chambers you consider. The annotations 1 and 2 refer to the piecewise step-polynomials and are placed inside (for two-dimensional chambers) or next to the chamber to which they are associated.

6.2 Multiplication

Suppose we are given two enumerators $c_1(\mathbf{p})$ and $c_2(\mathbf{p})$ and we want to compute their product $d(\mathbf{p}) = c_1(\mathbf{p}) \cdot c_2(\mathbf{p})$. This is especially useful if the image of either of the two factors is $\{0, 1\}$, i.e., if the enumerator corresponds to the indicator function of some set S . In this way we can construct a selection operator. That is, let the enumerator c_2 correspond to the indicator function of S . Then

$$d(\mathbf{p}) = c_1(\mathbf{p}) \cdot c_2(\mathbf{p}) = \begin{cases} c_1(\mathbf{p}) & \text{if } \mathbf{p} \in S \\ 0 & \text{otherwise.} \end{cases}$$

In particular, if both c_1 and c_2 correspond to indicator functions, say of S_1 and S_2 respectively, then d corresponds to the indicator function of their intersection $S_1 \cap S_2$. We call selection with the complement of a set, i.e., $d(\mathbf{p}) = c_1(\mathbf{p}) \cdot (1 - c_2(\mathbf{p}))$, “masking out”.

The corresponding operation on generating functions is called the *Hadamard product* (Barvinok and Pommersheim 1999) of $C_1(\mathbf{x})$ and $C_2(\mathbf{x})$, denoted $C_1(\mathbf{x}) \star C_2(\mathbf{x})$. That is, if

$$C_1(\mathbf{x}) = \sum_{\mathbf{p} \in \mathbb{Z}^d} c_1(\mathbf{p}) \mathbf{x}^{\mathbf{p}} \quad \text{and} \quad C_2(\mathbf{x}) = \sum_{\mathbf{p} \in \mathbb{Z}^d} c_2(\mathbf{p}) \mathbf{x}^{\mathbf{p}}$$

then

$$D(\mathbf{x}) = C_1(\mathbf{x}) \star C_2(\mathbf{x}) = \sum_{\mathbf{p} \in \mathbb{Z}^d} c_1(\mathbf{p}) \cdot c_2(\mathbf{p}) \mathbf{x}^{\mathbf{p}} = \sum_{\mathbf{p} \in \mathbb{Z}^d} d(\mathbf{p}) \mathbf{x}^{\mathbf{p}}.$$

Both the Hadamard product of rational generating functions (Barvinok and Pommersheim 1999, Theorem 10.2) and the product of piecewise step-polynomials may be computed in polynomial time. The latter can be proved as in Lemma 6.1 for addition. In this case, there are no problems with respect to the representation of the chamber complexes since the support of the chamber complex of the product is the *intersection* of the supports of the chambers complexes of the factors. Outer walls in the initial complexes will therefore remain outer walls in the final chamber complex (or will be demolished).

To ensure a polynomial time complexity, however, we may only apply the (Hadamard) product operation a *fixed* number of times. The reason is that in case of a product of piecewise step-polynomials the degree of the step-polynomials may increase. Generally, the maximal degree will be the sum of the degree of the factors. If we perform more than a fixed number of multiplications, then this degree is no longer a constant in terms of the dimension. In the case of rational generating functions, the number of factors in the denominator of each term will in general be the sum of the number of factors for each factor in the Hadamard product. Again, if we perform more than a fixed number of Hadamard products, then we may no longer assume that this number of factors is bounded by a constant. An additional problem is that the number of terms in the sum may grow exponentially on repeated application of the Hadamard product.

In some cases, the Hadamard product may still be performed without increase in the number of factors in the denominators. To see this, let us look at the computation of Hadamard products (Barvinok and Pommersheim 1999, Proof of Theorem 10.2) in slow motion. It is sufficient to know how to perform the Hadamard product on rational generating functions with a single term,

$$C_1(\mathbf{x}) = \frac{\mathbf{x}^{\mathbf{v}}}{\prod_{i=1}^{k_1} (1 - \mathbf{x}^{\mathbf{a}_i})} \quad \text{and} \quad C_2(\mathbf{x}) = \frac{\mathbf{x}^{\mathbf{w}}}{\prod_{i=1}^{k_2} (1 - \mathbf{x}^{\mathbf{b}_i})}, \quad (25)$$

with $\mathbf{v}, \mathbf{w}, \mathbf{a}_i, \mathbf{b}_i \in \mathbb{Z}^d$. Expansion as multiple geometric series yields

$$C_1(\mathbf{x}) = \sum_{\boldsymbol{\mu} \in \mathbb{N}^{k_1}} \mathbf{x}^{\mathbf{v} + \sum_{i=1}^{k_1} \mu_i \mathbf{a}_i} = \sum_{\mathbf{m}} c_1(\mathbf{m}) \mathbf{x}^{\mathbf{m}}$$

with

$$c_1(\mathbf{m}) = \# \left\{ \boldsymbol{\mu} \in \mathbb{N}^{k_1} \mid \mathbf{m} = \mathbf{v} + \sum_{i=1}^{k_1} \mu_i \mathbf{a}_i \right\}$$

and similarly

$$C_2(\mathbf{x}) = \sum_{\boldsymbol{\nu} \in \mathbb{N}^{k_2}} \mathbf{x}^{\mathbf{w} + \sum_{i=1}^{k_2} \nu_i \mathbf{b}_i} = \sum_{\mathbf{m}} c_2(\mathbf{m}) \mathbf{x}^{\mathbf{m}}$$

with

$$c_2(\mathbf{m}) = \# \left\{ \boldsymbol{\nu} \in \mathbb{N}^{k_2} \mid \mathbf{m} = \mathbf{w} + \sum_{i=1}^{k_2} \nu_i \mathbf{b}_i \right\}.$$

The product $d(\mathbf{m})$ of $c_1(\mathbf{m})$ and $c_2(\mathbf{m})$ is then simply

$$d(\mathbf{m}) = \# \left\{ (\boldsymbol{\mu}, \boldsymbol{\nu}) \in \mathbb{N}^{k_1+k_2} \mid \mathbf{m} = \mathbf{v} + \sum_{i=1}^{k_1} \mu_i \mathbf{a}_i \wedge \mathbf{m} = \mathbf{w} + \sum_{i=1}^{k_2} \nu_i \mathbf{b}_i \right\}.$$

This is the enumerator of a parametric polytope in $\mathbb{Q}^{k_1+k_2}$ with $2d$ equalities. To compute the generating function $D(\mathbf{x})$ of this enumerator (see Proposition 5.11, we need to consider the polyhedron

$$\left\{ (\mathbf{m}, \boldsymbol{\mu}, \boldsymbol{\nu}) \in \mathbb{Q}^{d+k_1+k_2} \mid \begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\nu} \end{bmatrix} \geq \mathbf{0} \wedge \mathbf{m} = \mathbf{v} + \sum_{i=1}^{k_1} \mu_i \mathbf{a}_i \wedge \mathbf{m} = \mathbf{w} + \sum_{i=1}^{k_2} \nu_i \mathbf{b}_i \right\}, \quad (26)$$

a polyhedron in $\mathbb{Q}^{d+k_1+k_2}$. If the $2d$ equalities are linearly independent, then the dimension of this polyhedron will be $k_1 + k_2 - d$ and so the number of factors in the denominators will be at most $k_1 + k_2 - d$.

In particular, if $C_1(\mathbf{x})$ and $C_2(\mathbf{x})$ are the generating functions of polyhedra, then $k_1 = k_2 = d$ and the $2d$ equalities are linearly independent since both \mathbf{a}_i and \mathbf{b}_i are the generators of a unimodular cone in this case. The number of factors therefore remains constant at d . This should not be surprising as the Hadamard product in this case is just the generating function of the intersection, which is itself a d -polyhedron. Note that the corresponding piecewise step-polynomial will just have the polyhedron itself as single chamber with as associated step-polynomial the constant 1.

A slightly more interesting case is selection with a polyhedron. In the world of piecewise step-polynomials, such selection is performed by simply intersecting each chamber with the polyhedron. The equivalent on generating functions is slightly more expensive, but still cheap in the sense that the number of factors in the denominators does not increase.

Example 34 Suppose we are given the following two rational generating functions

$$\begin{aligned} C_1(x) &= \frac{1}{(1-x)^2} = 1 + 2x + 3x^2 + 4x^3 + 5x^4 + 6x^5 + 7x^6 + \dots \\ C_2(x) &= \frac{1}{1-x^2} = 1 + x^2 + x^4 + x^6 + \dots \end{aligned}$$

and that we want to compute their Hadamard product. The corresponding piecewise step-polynomials are

$$\begin{aligned} c_1(p) &= p + 1 \quad \text{if } p \geq 0 \\ c_2(p) &= \left\lfloor \frac{p}{2} \right\rfloor - \left\lfloor \frac{p-1}{2} \right\rfloor \quad \text{if } p \geq 0 \end{aligned}$$

and so their product is simply

$$d(p) = c_1(p) \cdot c_2(p) = (p+1) \left(\left\lfloor \frac{p}{2} \right\rfloor - \left\lfloor \frac{p-1}{2} \right\rfloor \right) \quad \text{if } p \geq 0.$$

To compute the corresponding rational generating function $D(x)$ as the Hadamard product $C_1(\mathbf{x}) \star C_2(\mathbf{x})$, we see that in the notation of Equation (25) we have $v = 0$, $k_1 = 2$, $a_1 = a_2 = 1$ and $w = 0$, $k_2 = 1$, $b_1 = 2$. The polyhedron that needs to be enumerated (26) is

$$P = \left\{ (m, \boldsymbol{\mu}, \nu) \in \mathbb{Q}^{1+2+1} \mid \begin{bmatrix} \boldsymbol{\mu} \\ \nu \end{bmatrix} \geq \mathbf{0} \wedge m = \mu_1 + \mu_2 \wedge m = 2\nu \right\}.$$

After some simplification, we can solve this enumeration problem manually. First note that we can eliminate μ_2 using $m = \mu_1 + \mu_2$ and replace $\mu_2 \geq 0$ by $m - \mu_1 \geq 0$. The second equality is more tricky to eliminate. Suppose that we simply replace m by 2ν and consider ν to be the parameter. We have

$$P' = \{ (\nu, \mu_1) \in \mathbb{Q}^{1+1} \mid \mu_1 \geq 0 \wedge 2\nu - \mu_1 \geq 0 \wedge \nu \geq 0 \}.$$

The result of the enumeration of P' is a generating function

$$F(y) = \sum_{\nu \geq 0} f(\nu) y^\nu. \quad (27)$$

This generating function is not the generating function $D(x)$ that we seek, however. For $m = 2\nu$, there is a one-to-one correspondence between the integer points in P and those in P' and so $d(m) = f(\nu)$; for $m = 2\nu + 1$, on the other hand, the polyhedron P contains no integer points and so $d(m) = 0$. That is,

$$d(m) = \begin{cases} f\left(\frac{m}{2}\right) & \text{if } m = 2\nu \\ 0 & \text{if } m = 2\nu + 1. \end{cases}$$

Substituting $y = x^2$ in (27), we obtain

$$\begin{aligned} F(x^2) &= \sum_{\nu \geq 0} f(\nu) (x^2)^\nu \\ &= \sum_{m=2\nu \geq 0} f\left(\frac{m}{2}\right) x^m \\ &= \sum_{m=2\nu \geq 0} f\left(\frac{m}{2}\right) x^m + \sum_{m=2\nu+1 \geq 0} 0 x^m \\ &= \sum_{m \geq 0} d(m) x^m \\ &= D(x). \end{aligned}$$

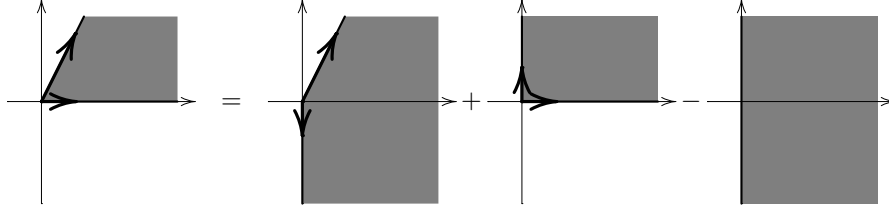


Figure 31: Dual Unimodular Decomposition for the cone in Example 34.

In other words, we can simply enumerate P' as $F(y)$ and subsequently substitute x^2 for y to obtain $D(x) = F(x^2)$.

The polyhedron P' is a non-unimodular cone. The corresponding decomposition is shown in Figure 31, where the final “term” is only shown as an intuitive explanation of this decomposition. According to the decomposition, the generating function of P' is

$$f(P'; \mathbf{y}) = \frac{1}{(1 - y_1 y_2^2)(1 - y_2^{-1})} + \frac{1}{(1 - y_1)(1 - y_2)}.$$

The enumerator of P' is $F(y_1) = f(P'; y_1, 1)$. Using the specialization technique of Lemma 5.9 on page 43, we obtain

$$F(y_1) = \left(\frac{1}{1 - y_1} + \frac{2y_1}{(1 - y_1)^2} \right) + (0).$$

The final generating function is therefore

$$D(x) = F(x^2) = \frac{1}{1 - x^2} + \frac{2x^2}{(1 - x^2)^2} = 1 + 3x^2 + 5x^4 + 7x^6 + \dots.$$

Multiplication of piecewise step-polynomials is implemented in `emul_partitions`. Masking on piecewise step-polynomials is implemented in `emask`.

6.3 Set Operations

Set operations, i.e., intersection, union and set difference, are direct applications of multiplication. Let $S_1, S_2 \subset \mathbb{Q}^n \times \mathbb{Q}^0$, i.e., let all of the following enumerators correspond to indicators. We have

$$\begin{aligned} c_{S_1 \cap S_2}(\mathbf{p}) &= c_{S_1}(\mathbf{p}) \cdot c_{S_2}(\mathbf{p}) \\ c_{S_1 \cup S_2}(\mathbf{p}) &= c_{S_1}(\mathbf{p}) + c_{S_2}(\mathbf{p}) - c_{S_1 \cap S_2}(\mathbf{p}) \\ c_{S_1 \setminus S_2}(\mathbf{p}) &= c_{S_1}(\mathbf{p}) - c_{S_1 \cap S_2}(\mathbf{p}) = c_{S_1}(\mathbf{p}) \cdot (1 - c_{S_2}(\mathbf{p})) \end{aligned}$$

and similarly

$$\begin{aligned} C_{S_1 \cap S_2}(\mathbf{x}) &= C_{S_1}(\mathbf{x}) \star C_{S_2}(\mathbf{x}) \\ C_{S_1 \cup S_2}(\mathbf{x}) &= C_{S_1}(\mathbf{x}) + C_{S_2}(\mathbf{x}) - C_{S_1 \cap S_2}(\mathbf{x}) \\ C_{S_1 \setminus S_2}(\mathbf{x}) &= C_{S_1}(\mathbf{x}) - C_{S_1 \cap S_2}(\mathbf{x}). \end{aligned}$$

6.4 Summation

Suppose we are given an enumerator $c(\mathbf{p})$ in n variables and we want to know the sum of all function values over some region of the parameter space. This region will typically only involve some of the parameters and the resulting sum will then be a function of the remaining parameters. In particular, we consider a set $S \subset \mathbb{Q}^l$ with $l \leq n$ and such that for each $\mathbf{p}' \in \mathbb{Q}^{n-l}$ there are

Listing 1: Artificial pointer conversion example.

```

p = a;
for (i = 0; i <= 99; ++i)
  for (j = i; j <= 99; ++j) {
    p += j * ((j-i)/4);
    *p = 0;
  }

```

only finitely many elements $\mathbf{p}'' \in S$ such that $c(\mathbf{p}', \mathbf{p}'')$ is non-zero. The *summation* of $c(\mathbf{p})$ over S is defined as

$$d(\mathbf{p}') = \sum_{\mathbf{p}'' \in S} c(\mathbf{p}', \mathbf{p}'').$$

Equivalently,

$$D(\mathbf{y}) = \sum_{\mathbf{p}'} d(\mathbf{p}') \mathbf{y}^{\mathbf{p}'} = \sum_{\mathbf{p}'} \sum_{\mathbf{p}'' \in S} c(\mathbf{p}', \mathbf{p}'') \mathbf{x}^{\mathbf{p}'} = C'(\mathbf{y}, \mathbf{1}), \quad (28)$$

with

$$C'(\mathbf{y}, \mathbf{z}) = C'(\mathbf{x}) = C(\mathbf{x}) \star f(\mathbb{Q}^{n-l} \times S; \mathbf{x}). \quad (29)$$

Such a summation may be useful to enumerate unions of parametric polytopes without computing a disjoint union first. We can first consider all of the $n + d$ variables to be parameters; enumerate all parametric polytopes to obtain enumerators in these $n + d$ variables; compute the intersection as in the previous section and finally sum over the d non-parameters. Another possible application of summation is pointer conversion or array recovery (van Engelen and Gallivan 2001; Franke and O'Boyle 2003), a process in which array accesses through pointers are converted to explicit array references.

Example 35 Consider the artificial program in Listing 1 and suppose we want to replace accesses to the array \mathbf{a} through the pointer \mathbf{p} by explicit accesses to \mathbf{a} . To obtain an explicit indexation in term of the iterators, we need to accumulate the increments to pointer \mathbf{p} over all previous iterations. That is, we need to sum the increment over all $0 \leq i' \leq 99$ and $i' \leq j' \leq 99$ such that $(i', j') \preccurlyeq (i, j)$. Note that the increment is the step-polynomial

$$c(i, j, i', j') = j' \left\lfloor \frac{j' - i'}{4} \right\rfloor.$$

The accumulated increment is therefore

$$d(i, j) = \sum_{\substack{(i', j') \in S \\ (i', j') \preccurlyeq (i, j)}} c(i, j, i', j'), \quad (30)$$

with $S = \{(i', j') \mid 0 \leq i' \leq 99 \wedge i' \leq j' \leq 99\}$.

We can split the summation process into two steps. The first is the selection specified in (29), which we may perform both on piecewise step-polynomials and rational generating functions in polynomial time as in Section 6.2. Note that if S is not a polyhedron, in particular $S \neq \mathbb{Z}^l$, then the degree of the piecewise step-polynomial or the number of factors in the denominators of the rational generating function may increase by a constant amount. The second step then summates over the whole of \mathbb{Z}^l .

Proposition 6.2 *Fix n and k . Let $c(\mathbf{p})$ with $\mathbf{p} \in \mathbb{Z}^n$ be a piecewise step-polynomial of degree k , then the summation of $c(\mathbf{p})$ over \mathbb{Z}^l may be computed in polynomial time. Likewise, let $C(\mathbf{x})$ with $\mathbf{x} \in \mathbb{C}^n$ be a rational generating function with at most k factors in each denominator, then the summation of $C(\mathbf{x})$ over \mathbb{Z}^l may be computed in polynomial time.*

Proof The case of rational generating functions is easy. By Lemma 5.9, we may perform the specialization specified in (28) in polynomial time.

For piecewise step-polynomials, it then suffices to prove the result for functions of the form

$$c(\mathbf{p}) = \begin{cases} \prod_{j=1}^k \lfloor \langle \mathbf{a}_j, \mathbf{p} \rangle + b_j \rfloor, & \text{for } \mathbf{p} \in C \\ 0, & \text{for } \mathbf{p} \notin C \end{cases},$$

where C is a rational polyhedron, $\mathbf{a}_j \in \mathbb{Q}^n$, and $b_j \in \mathbb{Q}$, because all piecewise step-polynomials may be written as linear combinations of functions of this form. Assuming that the value of each affine functional $\lfloor \langle \mathbf{a}_j, \mathbf{p} \rangle + b_j \rfloor$ is positive over the whole chamber C , the function value of $c(\mathbf{p})$ is equal to the number of integer points in a hyperrectangle with sides of length $\lfloor \langle \mathbf{a}_j, \mathbf{p} \rangle + b_j \rfloor - 1$, if the value of each affine functional is greater than or equal to 1, and zero if the value of any of the affine functionals is strictly less than 1. We therefore add k new variables t_j corresponding to the dimensions of such a hyperrectangle and consider the parametric polytope $Q \subset \mathbb{Q}^{n-l} \times \mathbb{Q}^{l+k}$, specified by

$$Q = \{ (\mathbf{p}', \mathbf{p}'', \mathbf{t}) \in C \times \mathbb{Q}^k \mid 1 \leq t_j \leq \langle \mathbf{a}_j, \mathbf{p} \rangle + b_j, \text{ for } 1 \leq j \leq k \}.$$

Then $d(\mathbf{p}') = \#Q_{\mathbf{p}'}$, which we may compute as a piecewise step-polynomial using Proposition 5.10. If any affine functional is not positive over the whole chamber then we may assume that it has a lower or an upper bound. (Otherwise split the (infinite) chamber according to the orthants of the parameter space.) If it has a lower bound, then let

$$m_j := \min_{\mathbf{p} \in C} \lfloor \langle \mathbf{a}_j, \mathbf{p} \rangle + b_j \rfloor \in \mathbb{Z}_{\leq 0}$$

and replace $\lfloor \langle \mathbf{a}_j, \mathbf{p} \rangle + b_j \rfloor$ by $\lfloor \langle \mathbf{a}_j, \mathbf{p} \rangle + b_j - m_j \rfloor + m_j$. If it only has an upper bound then replace $\lfloor \langle \mathbf{a}_j, \mathbf{p} \rangle + b_j \rfloor$ by

$$-1 - \left\lfloor \langle -\mathbf{a}_j, \mathbf{p} \rangle + b_j - \frac{1}{m} \right\rfloor$$

first, where m is the common denominator of a_{jk} and b_j . □

Example 36 Consider the summation in Equation (30) from the previous example. For simplicity, we will combine selection and summation and write the summation as the sum of two summations over polyhedra:

$$d(i, j) = \sum_{\substack{(i', j') \in S \\ i' < i}} c(i, j, i', j') + \sum_{\substack{(i', j') \in S \\ i' = i \wedge j' \leq j}} c(i, j, i', j').$$

We will continue with the first of these two summation. The ‘‘chamber’’ is simply the whole of the parameter domain intersected with $\mathbb{Q}^2 \times S$ and the constraint $i' \leq i - 1$,

$$C = \{ (i, j, i', j') \mid 0 \leq i \leq 99 \wedge i \leq j \leq 99 \wedge 0 \leq i' \leq 99 \wedge i' \leq j' \leq 99 \wedge i' \leq i - 1 \}.$$

The parametric polytope Q that needs to be enumerated to compute the summation is then

$$Q = \{ (i, j, i', j', t_1, t_2) \in C \times \mathbb{Q}^2 \mid 1 \leq t_1 \leq j' \wedge 4 \leq 4t_2 \leq j' - i' \}.$$

Note that $j' - i'$ is never negative in C and so we do not need to shift the step-polynomial or split the chamber.

Note that as in Section 6.2, we need to be careful how we represent chamber complexes. Keeping a list of overlapping full-dimensional chambers would result in the points in their common intersections being counted twice on a straightforward application of the above proposition. As we mentioned in Section 6.2, we actually represent chamber complexes as a partition of the integer points. Summation on piecewise step-polynomials is implemented in `esum`.

6.5 Conversion

The operations we have seen so far are all polynomial for both piecewise step-polynomials and rational generating functions. In principle, it would have been sufficient to only consider either piecewise step-polynomials or rational generating functions as we may convert between them in polynomial time, as we will show in the following theorem. This is especially important for operations for which there was no previously known polynomial method for performing them on one of these representations. In particular, as we will discuss in Section 7.4, a polynomial algorithm for projection on rational generating functions was described by Barvinok and Woods (2003), but to the best of our knowledge, no polynomial algorithm was ever described for performing projection on piecewise step-polynomials.

Theorem 6.3 *Fix n and k . There is a polynomial time algorithm which, given a rational generating function $f(\mathbf{x})$ in the form (7) with n variables and each $k_i \leq k$ and given $\mathbf{l} \in \mathbb{Z}^n$ such that $\langle \mathbf{l}, \mathbf{b}_{ij} \rangle \neq 0$ for all i and j , computes the piecewise step-polynomial $c : \mathbb{Z}^n \rightarrow \mathbb{Q}$ with degree at most k such that*

$$C(\mathbf{x}) = \sum_{\mathbf{p} \in \mathbb{Z}^n} c(\mathbf{p}) \mathbf{x}^{\mathbf{p}}$$

is the Laurent power series expansion of $C(\mathbf{x})$ that is convergent on a neighborhood of $\mathbf{e}^{\mathbf{l}} = (e^{\mathbf{l}_1}, e^{\mathbf{l}_2}, \dots, e^{\mathbf{l}_d})$.

Conversely, there is a polynomial time algorithm which, given a piecewise step-polynomial $c : \mathbb{Z}^n \rightarrow \mathbb{Q}$ of degree at most k such that $C(\mathbf{x}) = \sum_{\mathbf{p} \in \mathbb{Z}^n} c(\mathbf{p}) \mathbf{x}^{\mathbf{p}}$ converges on some nonempty open subset of \mathbb{C}^n , computes the rational generating function $C(\mathbf{x})$ in the form (7) with $k_i \leq k$.

Proof In both directions, we reduce the problem to a set of counting problems to which we apply either Proposition 5.10 or Proposition 5.11.

We first consider the conversion of rational generating functions to piecewise step-polynomials. By Lemma 6.1 it suffices to consider rational generating functions with a single term, i.e.,

$$C(\mathbf{x}) = \alpha \frac{\mathbf{x}^{\mathbf{v}}}{(1 - \mathbf{x}^{\mathbf{a}_1})(1 - \mathbf{x}^{\mathbf{a}_2}) \dots (1 - \mathbf{x}^{\mathbf{a}_d})}.$$

Furthermore, we may assume that $\langle \mathbf{l}, \mathbf{a}_i \rangle < 0$ for all i (otherwise apply the identity (3)) and that $\alpha = 1$ and $\mathbf{v} = \mathbf{0}$, because if $c'(\mathbf{p})$ is a piecewise step-polynomial representation of the generating function $C'(\mathbf{x})$, then $\alpha \cdot c'(\mathbf{p} - \mathbf{v})$ is a piecewise step-polynomial representation of $\alpha \mathbf{x}^{\mathbf{v}} C'(\mathbf{x})$.

We expand $C(\mathbf{x})$ as a product of infinite geometric series,

$$C(\mathbf{x}) = \prod_{i=1}^k (1 + \mathbf{x}^{\mathbf{a}_i} + \mathbf{x}^{2\mathbf{a}_i} + \dots).$$

Then

$$C(\mathbf{e}^{\mathbf{l}}) = \prod_{i=1}^k (1 + e^{\langle \mathbf{l}, \mathbf{a}_i \rangle} + e^{2\langle \mathbf{l}, \mathbf{a}_i \rangle} + \dots),$$

and this expansion is convergent on a neighborhood of $\mathbf{e}^{\mathbf{l}}$, since $\langle \mathbf{l}, \mathbf{a}_i \rangle < 0$. We see that we are looking to compute the enumeration

$$c(\mathbf{p}) = \#\{ \boldsymbol{\lambda} = (\lambda_1, \lambda_2, \dots, \lambda_k) \in \mathbb{Z}_{\geq 0}^k \mid \mathbf{p} = \lambda_1 \mathbf{a}_1 + \lambda_2 \mathbf{a}_2 + \dots + \lambda_k \mathbf{a}_k \}.$$

Let P be the parametric polytope

$$P = \{ (\mathbf{p}, \boldsymbol{\lambda}) \in \mathbb{Q}^n \times \mathbb{Q}^k \mid \boldsymbol{\lambda} \geq \mathbf{0} \text{ and } \mathbf{p} = \lambda_1 \mathbf{a}_1 + \dots + \lambda_k \mathbf{a}_k \}. \quad (31)$$

Then

$$c(\mathbf{p}) = \#\{ \boldsymbol{\lambda} \in \mathbb{Z}^k \mid (\mathbf{p}, \boldsymbol{\lambda}) \in P \},$$

which can be computed as a piecewise step-polynomial using Proposition 5.10. This proves the first half of the theorem.

The second half may be proved as in Proposition 6.2 with $l = 0$ and applying Proposition 5.11 on the parametric polytope Q instead of Proposition 5.10. \square

Note that if we are given a rational generating function and want to compute a function $c(\mathbf{p})$ which we know is only nonzero for \mathbf{p} in some polyhedron Q such that Q does not contain any straight lines, then we may take any \mathbf{l} such that $\langle \mathbf{l}, \mathbf{b}_{ij} \rangle \neq 0$ for all i, j and such that

$$Q \cap \{\mathbf{x} \in \mathbb{Q}^n \mid \langle \mathbf{l}, \mathbf{x} \rangle \geq 0\} \quad (32)$$

is bounded. Such an \mathbf{l} will give us the desired Laurent power series expansion $\sum_{\mathbf{p}} c(\mathbf{p}) \mathbf{x}^{\mathbf{p}}$. If such a polyhedron Q is known, then it can be advantageous to intersect P from Equation (31) with $Q \times \mathbb{Q}^k$. Some of these intersections may be empty, avoiding the cost of enumerating them and, more significantly, avoiding the cost of adding them together to form step-polynomials outside Q that are known to be zero.

Example 37 Take the parametric polytope `g1` from the `PolyLib` distribution. Directly computing the enumerator function yields a piecewise step-polynomial with 6 “chambers”. First computing the corresponding generating function yields a rational generating function with 56 terms, 46 of which are non-zero only outside the projection Q of `g1` onto its parameter space. Converting the rational generating function to a piecewise step-polynomial using the method above and taking into account the context Q takes 1.5s in total. The resulting piecewise step-polynomial has 32 “chambers”. If we ignore Q , then summing the piecewise step-polynomials corresponding to the first 10 terms in the rational generating function already yields a piecewise step-polynomial with 114 chambers. Adding the piecewise step-polynomial corresponding to the 11th term takes over one hour. Note that as we explain in Section 6.1, we allow our chambers to be unions of polyhedra. The main bottleneck during the computation of the sum of two piecewise step-polynomials appears to be the function `DomainDifference` from `PolyLib`.

Example 38 Consider the function

$$C(\mathbf{x}) = \frac{1}{(1 - \mathbf{x}^{(1,1)})(1 - \mathbf{x}^{(2,1)})(1 - \mathbf{x}^{(1,0)})(1 - \mathbf{x}^{(0,1)})},$$

which is the generating function of the vector partition function

$$c(\mathbf{s}) = \# \left\{ \boldsymbol{\lambda} \in \mathbb{N}^4 \mid \begin{bmatrix} 1 & 2 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \boldsymbol{\lambda} = \mathbf{s} \right\}.$$

This is the same as the example of Beck (2004, Section 4). The conversion of the rational generating function to the corresponding piecewise step-polynomial was discussed in Examples 10, 16, 24, 29 and 31. The final result is shown in Figure 27.

Example 39 Consider once more the generating function $f(T; \mathbf{x})$ (8) from Example 21. If we consider T as element of $\mathbb{Q}^2 \times \mathbb{Q}^0$, i.e., if we consider $T_{\mathbf{t}}$ to be a parametric polytope of dimension 0 with 2 parameters, then $f(T; \mathbf{x})$ is also the generating function of the enumerator of T . Let us convert this generating function to an explicit piecewise step-polynomial in slow motion. We choose $\mathbf{l} = (-1, 1)$ and obtain

$$f(T; \mathbf{x}) = \frac{x_1^2(-x_1)(-x_1x_2^{-1})}{(1-x_1)(1-x_1x_2^{-1})} + \frac{x_2^2}{(1-x_2^{-1})(1-x_1x_2^{-1})} + \frac{-x_2^{-1}}{(1-x_1)(1-x_2^{-1})}.$$

For the first term we get

$$\begin{aligned}
\frac{x_1^2(-x_1)(-x_1x_2^{-1})}{(1-x_1)(1-x_1x_2^{-1})} &= \frac{x_1^4x_2^{-1}}{(1-x_1)(1-x_1x_2^{-1})} \\
&= x_1^4x_2^{-1} \left(\sum_{u \geq 0} x_1^u \right) \left(\sum_{v \geq 0} (x_1x_2^{-1})^v \right) \\
&= x_1^4x_2^{-1} \sum_{u \geq 0, v \geq 0} 1 x_1^{u+v} x_2^{-v} \\
&= x_1^4x_2^{-1} \sum_{-v \geq 0, u+v \geq 0} 1 x_1^u x_2^v \\
&= \sum_{-v \geq 0, u+v \geq 0} 1 x_1^{u+4} x_2^{v-1} \\
&= \sum_{v \leq -1, u+v \geq 3} 1 x_1^u x_2^v
\end{aligned}$$

Similarly

$$\begin{aligned}
\frac{x_2^2}{(1-x_2^{-1})(1-x_1x_2^{-1})} &= x_2^2 \sum_{u \geq 0, v \geq 0} 1 x_1^v x_2^{-u-v} \\
&= \sum_{u \geq 0, -v-u \geq 0} 1 x_1^u x_2^{v+2} \\
&= \sum_{u \geq 0, v+u \leq 2} 1 x_1^u x_2^v
\end{aligned}$$

and

$$\begin{aligned}
\frac{-x_2^{-1}}{(1-x_1)(1-x_2^{-1})} &= \sum_{u \geq 0, v \leq 0} -1 x_1^u x_2^{v-1} \\
&= \sum_{u \geq 0, v \leq -1} -1 x_1^u x_2^v
\end{aligned}$$

The result is then $-[u \geq 0, v \leq -1] + [u + v \geq 3, v \leq -1] + [u + v \leq 2, u \geq 0] = [u \geq 0, v \geq 0, u + v \leq 2]$, where we use the notation $[constraints]$ as a shorthand for the function

$$1 \quad \text{if } constraints.$$

I.e., the final piecewise step-polynomial (after removal of chambers where the function value is zero) is

$$1 \quad \text{if } u \geq 0, v \geq 0, u + v \leq 2.$$

The sum of the piecewise step-polynomials above is shown graphically in figure 32, with the domain of the first piecewise step-polynomial, which contributes negatively to the final piecewise step-polynomial, marked by the shaded area and the domains of the other two piecewise step-polynomials, which contribute positively to the final piecewise step-polynomial, marked by thick lines.

The conversion of a rational generating function to a piecewise step-polynomial is implemented in `gen_fun::operator value *`. In our implementation, we assume that Q from Equation (32) has only lexico-positive rays, if any. This is not a real restriction since Q may not contain any lines anyway. Under this assumption, the vector \mathbf{l} in Theorem 6.3 need not be computed explicitly. Instead the requirement $\langle \mathbf{l}, \mathbf{a}_i \rangle < 0$ for all i may be replaced by the requirement that all \mathbf{a}_i be lexico-positive. This is ensured during the addition of a term to a rational generating function in `gen_fun::add`.

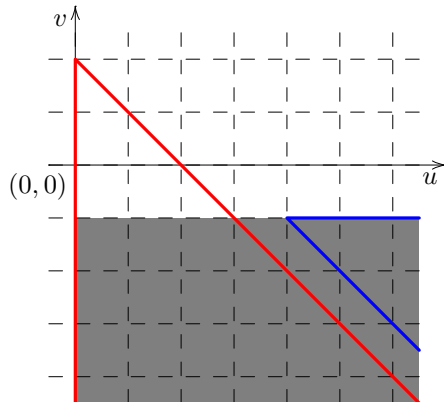


Figure 32: Barvinok example indicator decomposition.

6.6 Evaluation

As we discussed in Section 4, evaluation of a piecewise step-polynomial c at $\mathbf{p} \in \mathbb{Z}^n$ is trivial. Just find a chamber containing \mathbf{p} and evaluate the associated step-polynomial at \mathbf{p} . The most straightforward way of obtaining the coefficient of $\mathbf{x}^{\mathbf{p}}$, i.e., $c(\mathbf{p})$, from a rational generating function $C(\mathbf{x})$, on the other hand, would be to expand the Laurent power series up to power \mathbf{p} , which is a process that is exponential in the size of \mathbf{p} . A better way is to first convert the rational generating function to a piecewise step-polynomial using the results from Section 6.5. This can be slightly improved by immediately taking into account the value of \mathbf{p} during the enumeration of the polytope P in (31), reducing the parametric enumeration to a non-parametric enumeration.

7 Projection

In this section we consider techniques for enumerating parametric projected sets as discussed in Section 3.4. We will mainly be interested in computing an explicit enumeration function. It is clear that these enumerators can also be represented as piecewise step-polynomials. It is not obvious, however, that they may be computed in polynomial time.

We consider four techniques for enumerating sets with existential variables. The first technique, proposed by Claus (1996) and discussed in Section 7.1, is polynomial, but only works for the case of a single existential variable, i.e., the case where only a single variable is projected out. Section 7.2 explains a new technique, which is more general and still polynomial, but which does not handle all cases. It does seem to handle most cases that occur in compiler optimization, though (Verdoolaege et al. 2004a). The third technique, proposed by Boulet and Redon (1998a) and discussed in Section 7.3, is worst-case exponential, but works in general. The original proposal combined Parametric Integer Programming (PIP) (Feautrier 1988) with the method of Claus and Loechner (1998). Replacing the latter by the method from Section 5 yields a technique that seems to work very well in practice (Verdoolaege et al. 2004a). The fourth method, explained in Section 7.4, is based on a polynomial technique by Barvinok and Woods (2003) to compute rational generating functions of parametric sets, provided that the polyhedron representing the parametric set is in fact a polytope. This technique has not yet been implemented, however. Finally, in Section 7.5, we briefly discuss a technique for reducing the enumeration of parametric sets that are projections of polyhedra containing lines to the enumeration of parametric sets that are projections of polyhedra without lines. Such a reduction could be useful as a preprocessing step for a technique based on generating functions since the generating function of a polyhedron containing a line is zero.

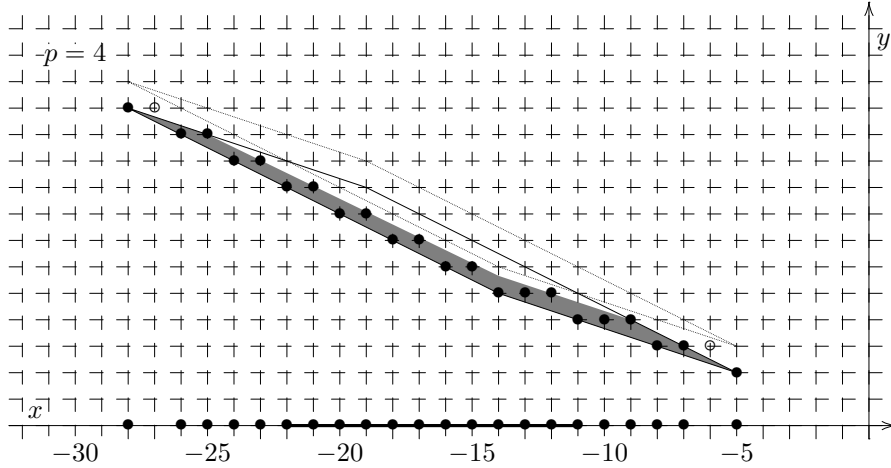


Figure 33: The set Q_4 from Example 40.

7.1 Shift and Subtract

We start with a very simple technique that works in case of a single existential variable. Let $S_{\mathbf{p}} \subset \mathbb{Z}^d$ be a parametric set with a single existential variable. Then $S_{\mathbf{p}} = \pi_d(\mathbb{Z}^{d+1} \cap P_{\mathbf{p}})$, with $P_{\mathbf{p}} \subset \mathbb{Q}^{d+1}$ some parametric polytope. Consider the set

$$Q_{\mathbf{p}} = P_{\mathbf{p}} \setminus (P_{\mathbf{p}} + \mathbf{e}_{d+1}),$$

with \mathbf{e}_{d+1} the $(d+1)$ st standard basis vector. We have $S_{\mathbf{p}} = \pi_d(\mathbb{Z}^{d+1} \cap Q_{\mathbf{p}})$ and for every point $\mathbf{x} \in S_{\mathbf{p}}$, there is exactly one point $(\mathbf{x}, y) \in Q_{\mathbf{p}}$. The number of points in $S_{\mathbf{p}}$ is therefore the same as that of $Q_{\mathbf{p}}$, i.e., $c_S(\mathbf{p}) = c_Q(\mathbf{p})$. In general, $Q_{\mathbf{p}}$ is a union of polytopes. By first writing $Q_{\mathbf{p}}$ as a disjoint union of polytopes, we may simply apply the techniques from Section 5. This technique was proposed by Clauss (1996), who called $Q_{\mathbf{p}}$ the *thick facets* of $P_{\mathbf{p}}$, and was recently implemented by Seghir (2002) as `Polyhedron_Image_Enumerate`, available in `PolyLib`. Barvinok and Pommersheim (1999, Example 10.4) apply the same technique on rational generating functions. In this case, we may opt not to compute $Q_{\mathbf{p}}$ explicitly, but rather compute $f(P \setminus (P + \mathbf{e}_{n+d+1}); \mathbf{x})$ using the techniques from Section 6.3. For both representations, we may also compute the enumerator of $S_{\mathbf{p}}$ as the difference of the enumerators of $P_{\mathbf{p}}$ and $P_{\mathbf{p}} \cap (P_{\mathbf{p}} + \mathbf{e}_{d+1})$.

Example 40 Consider the set

$$S'_p = \{x \in \mathbb{Z} \mid \exists y \in \mathbb{Z} : -x - p \leq 2y \leq -x - 1 \wedge -x + 1 \leq 3y \leq -x + 8\}.$$

We have $S'_p = \pi_1(P_p)$ with

$$P_p = \{(x, y) \in \mathbb{Z}^2 \mid -x - p \leq 2y \leq -x - 1 \wedge -x + 1 \leq 3y \leq -x + 8\}.$$

Therefore, $c_S(p) = c_Q(p)$ with $Q_p = P_p \setminus (P_p + (0, 1))$. The set Q_4 is shown in Figure 33.

7.2 Elimination

The elimination technique tries to eliminate as many existentially quantified variables as possible, either by projecting them away or removing the quantifier, if this does not affect the count, or by splitting the original problem into a number of smaller problems that have a higher chance of seeing an existential variable eliminated. If any existential variables remain then other techniques, such as the one in Section 7.3, can be used on the simplified problem(s).

7.2.1 Unique Existential Variables

The existential quantifiers introduced by tools that automatically extract counting problems from source code can sometimes be redundant. This occurs when for each \mathbf{x} in the corresponding set, there is at most one y_i that satisfies the constraints. In such a case, the existential quantifier for y_i can be omitted without affecting the count of the set.

Many such cases can be detected when there is a constraint that involves y_i but none of the other existential variables, which we denote by $\bar{\mathbf{y}}$. Without loss of generality, we will assume the constraint establishes a lower bound on the variable y_i , i.e., it is of the form

$$n_l y_i + \langle \mathbf{a}_l, \mathbf{x} \rangle + \langle \mathbf{b}_l, \mathbf{p} \rangle + c_l \geq 0 \quad (33)$$

with $n_l \in \mathbb{N}$, and $\langle \mathbf{v}, \mathbf{w} \rangle$ the inner product of \mathbf{v} and \mathbf{w} . Combining this constraint with an upper bound

$$-n_u y_i + \langle \mathbf{a}_u, \mathbf{x} \rangle + \langle \bar{\mathbf{d}}_u, \bar{\mathbf{y}} \rangle + \langle \mathbf{b}_u, \mathbf{p} \rangle + c_u \geq 0 \quad (34)$$

we obtain

$$-n_u (\langle \mathbf{a}_l, \mathbf{x} \rangle + \langle \mathbf{b}_l, \mathbf{p} \rangle + c_l) \leq n_u n_l y_i \leq n_l (\langle \mathbf{a}_u, \mathbf{x} \rangle + \langle \bar{\mathbf{d}}_u, \bar{\mathbf{y}} \rangle + \langle \mathbf{b}_u, \mathbf{p} \rangle + c_u). \quad (35)$$

The number of distinct integer values for $n_u n_l y_i$ is given by the upper bound minus the lower bound plus one. If this number is smaller than $n_u n_l$, then the two constraints admit at most one integer value for y_i . That is, if

$$n_l (\langle \mathbf{a}_u, \mathbf{x} \rangle + \langle \bar{\mathbf{d}}_u, \bar{\mathbf{y}} \rangle + \langle \mathbf{b}_u, \mathbf{p} \rangle + c_u) + n_u (\langle \mathbf{a}_l, \mathbf{x} \rangle + \langle \mathbf{b}_l, \mathbf{p} \rangle + c_l) + 1 \leq n_l n_u \quad (36)$$

for all integer values that satisfy the constraints, then y_i is uniquely determined by \mathbf{x} and \mathbf{p} and can therefore be treated as a regular variable, without existential quantification. It is independent of the other existential variables because of our assumption that one of the constraints does not involve these other variables. Condition (36) can easily be checked by adding the negation to the existing set of constraints and testing for satisfiability. Note that it is sufficient to find one such pair to be able to drop the existential quantification of the variable.

Example 41 Consider the set S_p

$$\{x \in \mathbb{Z} \mid \exists y \in \mathbb{Z} : x + 3y \leq 8 \wedge x + 2y + 1 \leq 0 \wedge x + 2y + p \geq 0 \wedge x + 3p + 11 \leq 0\}.$$

Since there is only a single existential variable, all constraints are independent of the “other existential variables”. Using $x + 2y + p \geq 0$ and $-x - 3y + 8 \geq 0$ as constraints, condition (36) yields

$$x + 3p + 17 \leq 6. \quad (37)$$

All elements of the set satisfy this constraint so we can remove the existential quantification and the set S_p then corresponds to the integer points in the parametric polytope P_p

$$\{(x, y) \in \mathbb{Q}^2 \mid x + 3y \leq 8 \wedge x + 2y + 1 \leq 0 \wedge x + 2y + p \geq 0 \wedge x + 3p + 11 \leq 0\}.$$

The number of points can be written as

$$c_S(p) = c_P(p) = \begin{cases} 5 & \text{if } p \geq 3 \\ -\frac{3}{4}p^2 + \frac{15}{4}p + \frac{1}{2}\lfloor \frac{1}{2}p \rfloor & \text{if } 1 \leq p \leq 2 \end{cases}.$$

Even if there is no single existential variable that is unique, some linear combination of existential variables may still be unique. To avoid enumerating all possible combinations, we only consider this case if we have two constraints that are “parallel in the existential space”, i.e., such that $\mathbf{d}_l = n_l \mathbf{d}$ and $\mathbf{d}_u = -n_u \mathbf{d}$ for some positive integers n_l and n_u and an integer vector \mathbf{d} with greatest common divisor (gcd) 1. We compute condition (36) from (33) and (34) with y_i replaced by $\langle \mathbf{d}, \mathbf{y} \rangle$ ($\bar{\mathbf{d}}_u$ is $\mathbf{0}$ in this case). If this condition holds, we perform a change of basis such that $y'_1 = \langle \mathbf{d}, \mathbf{y} \rangle$, which we now know to be unique. Such a change of basis can be obtained through transformation by the unimodular extension of \mathbf{d} (Bik 1996).

Example 42 Consider the set S_p (6) from Example 18. This set satisfies the equality $l = 6i + 9j - 7$, which means that $2i + 3j$ is unique. Transforming this set using the unimodular extension of $\mathbf{d} = (2, 3)$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ -1 & -1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

we obtain

$$S_p = \{ l \in \mathbb{Z} \mid \exists x, y \in \mathbb{Z} : l = 3x - 7 \wedge -x - p \leq 2y \leq -x - 1 \wedge -x + 1 \leq 3y \leq -x + 8 \}.$$

Since equation $l = 3x - 7$ provides an upper and a lower bound on x that are equal, Equation (36) is trivially satisfied and $\exists x$ can be removed. Since l is now redundant, we removed it for simplicity:

$$S'_p = \{ x \in \mathbb{Z} \mid \exists y \in \mathbb{Z} : -x - p \leq 2y \leq -x - 1 \wedge -x + 1 \leq 3y \leq -x + 8 \}. \quad (38)$$

7.2.2 Redundant Existential Variables

Consider once more a lower bound on the existential variable y_i :

$$n_l y_i + \langle \mathbf{c}_l, \mathbf{w} \rangle \geq 0,$$

where we used $\mathbf{c}_l := (\mathbf{a}_l, \bar{\mathbf{d}}_l, \mathbf{b}_l, c_l)$ and $\mathbf{w} := (\mathbf{x}, \bar{\mathbf{y}}, \mathbf{p}, 1)$ for brevity. Since we are only interested in integer values of y_i , this is equivalent to

$$n_u(n_l y_i + \langle \mathbf{c}_l, \mathbf{w} \rangle) + n_u - 1 \geq 0,$$

for any positive integer n_u . Similarly, for an upper bound we obtain

$$n_l(-n_u y_i + \langle \mathbf{c}_u, \mathbf{w} \rangle) + n_l - 1 \geq 0.$$

The range in (35) can therefore be expanded to

$$-n_u \langle \mathbf{c}_l, \mathbf{w} \rangle - n_u + 1 \leq n_u n_l y_i \leq n_l \langle \mathbf{c}_u, \mathbf{w} \rangle + n_l - 1.$$

If this range is larger than $n_u n_l$, i.e., if

$$n_l \langle \mathbf{c}_u, \mathbf{w} \rangle + n_u \langle \mathbf{c}_l, \mathbf{w} \rangle + n_l - 1 + n_u - 1 + 1 \geq n_l n_u, \quad (39)$$

then there is *at least* one integer value for each given value of the other variables. If this holds for all pairs of constraints, then variable y_i does not restrict the solutions in any way and can simply be eliminated. This is known as the Omega test (Pugh 1991; Pugh 1992). Note that unlike the case of unique existential variables, the constraints need not be independent of the other existential variables.

Example 43 Consider the set

$$S_p = \{ x \in \mathbb{Z} \mid \exists y \in \mathbb{Z} : -x - p \leq 2y \leq -x - 1 \wedge x \leq -11 \wedge -x + 1 \leq 3y \leq -x + 8 \wedge x + 3p + 10 \geq 0 \wedge p \geq 3 \}.$$

This set is shown (■) in Figure 34. Pairwise combining the two upper and two lower bounds to form condition (39), we obtain $2p + 1 \geq 4$, $26 \geq 9$, $-x - 1 \geq 6$ and $x + 20 + 3p \geq 6$. All of these are true in S_p . (Note that in practice we would use the lcm of n_l and n_u instead of their product.) Variable y can therefore be eliminated and we obtain

$$S_p = \{ x \in \mathbb{Z} \mid x \leq -11 \wedge p \geq 3 \wedge x + 3p + 10 \geq 0 \}.$$

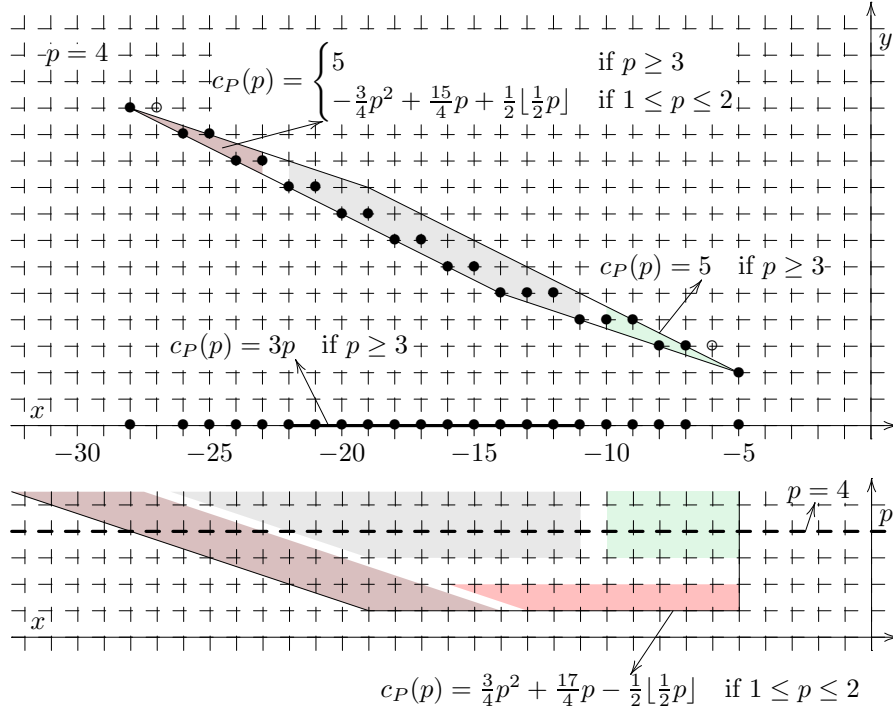


Figure 34: Decomposition of the set from Example 44.

7.2.3 Independent Splits

If neither of the two heuristics above apply, we can split the set into two or more parts by cutting the polyhedron in the combined space along a hyperplane. We only consider hyperplanes that are independent of the existential variables. This ensures that the enumerator of the original set is the sum of the enumerators of the parts. A cut that depends on existential variables, on the other hand, would result in sets that may intersect, requiring the computation of a disjoint union.

In particular, we consider all pairs of a lower and an upper bound on an existential variable that do not depend on other existential variables, i.e., they are of the form (33). If neither condition (36) nor condition (39) is satisfied over the whole set, then we cut off that part of the set where condition (36) does hold. In the remaining part, condition (39) holds for this particular pair of constraints. Since the number of pairs of constraints is polynomial in the input size, the number of sets we split off is also polynomial and so the whole technique, if it applies, is polynomial in the input size (for fixed dimension). As a special case, this technique always applies if there is only a single existential variable.

Example 44 Consider once more the set S'_p (38) from Example 42. The bottom of Figure 34 shows the projection of the corresponding polyhedron in the combined data-parameter space onto the xp -plane and the top shows the xy -slice at $p = 4$. The two constraints we considered in Example 41 also appear in this set. Condition (37) does not hold for the whole set, but instead is used to cut off the part that we considered in Example 41. This is the leftmost part (■) in Figure 34. Using the other constraints, we further split off $p \leq 2$ and $x \geq -10$. The remaining part is the set discussed in Example 43. The complete enumerator is the sum of the enumerators of the individual pieces, i.e.,

$$c_S(p) = c_{S'}(p) = \begin{cases} 3p + 10 & \text{if } p \geq 3 \\ 8p & \text{if } 1 \leq p \leq 2. \end{cases}$$

Algorithm 2 Enumeration of sets with existential variables.

enumerate_set(S)

1. If the number of existential variables is zero, then
 - (a) $E = \text{enumerate_polyhedron}(S)$
 - (b) return E
 2. If there is a redundant existential variable, then
 - (a) Eliminate the variable to obtain S'
 - (b) $E = \text{enumerate_set}(S')$
 - (c) return E
 3. Otherwise, if there is a unique existential variable, then
 - (a) Remove the quantification to obtain S'
 - (b) $E = \text{enumerate_set}(S')$
 - (c) return E
 4. Otherwise, if an independent split can be performed, then
 - (a) Split S into S_1 and S_2
 - (b) $E_1 = \text{enumerate_set}(S_1)$
 - (c) $E_2 = \text{enumerate_set}(S_2)$
 - (d) return $E_1 + E_2$
-

7.2.4 Overview

Algorithm 2 shows how the different reduction rules in this section can be combined. If none of the rules apply, then we may use the technique outlined in the next section.

7.3 Parametric Integer Programming

Boulet and Redon (1998a) propose to compute the enumerator of a parametric set with existential variables in two steps. First, PIP (Feautrier 1988) is used to eliminate the existential variables, after which the method of Clauss and Loechner (1998) is used to enumerate the resulting set of linear inequalities. In this section, we describe this technique in some more detail, but obviously use the method from Section 5 rather than the method of Clauss and Loechner (1998) to enumerate the resulting parametric polytopes.

PIP is a technique for computing the lexicographical minimum of a parametric polytope as a function of the parameters. The solution is defined by rational linear expressions in both the original parameters and possibly some extra parameters, defined as the lower integer parts of rational linear expressions of other parameters. Different solutions may exist in different parts of the parameter space, each defined by linear inequalities in the parameters (both original and extra).

To see how parametric integer programming helps in the enumeration of parametric sets, consider such a parametric set

$$S_{\mathbf{p}} = \left\{ \mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{y} \in \mathbb{Z}^{d'} : A\mathbf{x} + D\mathbf{y} + B\mathbf{p} + \mathbf{c} \geq \mathbf{0} \right\}.$$

with d regular variables, d' existential variables and n parameters. Compute the lexicographical minimum of the d' existential variables where both the regular variables and the original

parameters are considered as parameters, i.e.,

$$\underline{\mathbf{y}}_{(\mathbf{x}, \mathbf{p})} = \text{lexmin} \left\{ \mathbf{y} \in \mathbb{Z}^{d'} \mid \mathbf{A}\mathbf{x} + D\mathbf{y} + B\mathbf{p} + \mathbf{c} \geq \mathbf{0} \right\}.$$

Replacing \mathbf{y} by $\underline{\mathbf{y}}_{(\mathbf{x}, \mathbf{p})}$ in the definition of $S_{\mathbf{p}}$ does not change the number of solutions. However, $\underline{\mathbf{y}}_{(\mathbf{x}, \mathbf{p})}$ is unique (it satisfies Equation (36)) and the quantifier can be dropped. The extra parameters that may appear in the solution can be handled by considering them as extra (unique) existential variables in the set $S_{\mathbf{p}}$.

The advantage of using PIP is that it always applies and that it introduces d' equalities, reducing the total dimension. The disadvantage of PIP is that it is worst-case exponential, even for fixed dimension, and that it adds extra existential variables, increasing the total dimension. The total dimension is important since the enumeration technique for parametric polytopes is only polynomial for fixed dimension. Whether the final dimension is greater or smaller than the dimension of the original problem depends on whether the number of extra variables is greater or smaller than the number of existential variables in the original problem.

Example 45 Consider once more the set S'_p (38) from Example 42. The solution of

$$\underline{y}_{(x,p)} = \text{lexmin} \{ y \in \mathbb{Z} \mid -x - p \leq 2y \leq -x - 1 \wedge -x + 1 \leq 3y \leq -x + 8 \}$$

is

$$\underline{y}_{(x,p)} = \begin{cases} 1 - x - \lfloor \frac{2-2x}{3} \rfloor & \text{if } x + 3p + 2 \geq 0 \\ -x - \lfloor \frac{p-x}{2} \rfloor & \text{otherwise} \end{cases}.$$

The set S'_p can thus be written as the (disjoint) union of two sets $S_p^1 \sqcup S_p^2$. E.g., S_p^1 is defined as

$$S_p^1 = \{ x \in \mathbb{Z} \mid \exists y, q \in \mathbb{Z}^2 : y = 1 - x - q \wedge 2 - 2x \leq 3q \leq 4 - 2x \wedge x + 3p + 2 \geq 0 \wedge -x - p \leq 2y \leq -x - 1 \wedge -x + 1 \leq 3y \leq -x + 8 \},$$

where q is the new “parameter” $q = \lfloor (2 - 2x)/3 \rfloor$. Note that both S_p^1 and S_p^2 have exactly one additional (unique) existential variable, which means that the total dimension remains constant in this example.

7.4 Generating Functions

Barvinok and Woods (2003) describe a polynomial time algorithm for computing the generating function of the projection of a polytope. Combining their result with our polynomial time conversion algorithm from Section 6.5, we may compute the enumerator of a parametric set as a piecewise step-polynomial in polynomial time as in the following proposition. To the best of our knowledge, this yields the first polynomial time algorithm to compute such enumerators. We have not yet implemented this algorithm, however, and a proper implementation may still prove to be a challenge. Furthermore, the results of Barvinok and Woods (2003) only apply to polytopes, i.e., *bounded* polyhedra. In general we also want to handle parametric sets that are projections of parametric polyhedra, as long as the parametric set itself is bounded. A partial solution is proposed in the next section.

Proposition 7.1 *Let n , d , and m be fixed. There is a constant $k = k(n, d, m)$ and a polynomial time algorithm which, given a polytope $P \subset \mathbb{Q}^n \times \mathbb{Q}^d \times \mathbb{Q}^m$, computes the piecewise step-polynomial*

$$c(\mathbf{p}) = \# \{ \mathbf{t} \in \mathbb{Z}^d \mid \exists \mathbf{u} \in \mathbb{Z}^m : (\mathbf{p}, \mathbf{t}, \mathbf{u}) \in P \}$$

with degree at most k .

Proof Let

$$S = \{ (\mathbf{p}, \mathbf{t}) \in \mathbb{Z}^n \times \mathbb{Z}^d \mid \exists \mathbf{u} \in \mathbb{Z}^m : (\mathbf{p}, \mathbf{t}, \mathbf{u}) \in P \}.$$

Then we may compute, in polynomial time, the generating function

$$f(S; \mathbf{x}, \mathbf{y}) = \sum_{(\mathbf{p}, \mathbf{t}) \in S} \mathbf{x}^{\mathbf{p}} \mathbf{y}^{\mathbf{t}},$$

using Theorem 1.7 of Barvinok and Woods (2003). Next we compute $C_S(\mathbf{x}) = f(S; \mathbf{x}, \mathbf{1})$ using Lemma 5.9, and the $c_S(\mathbf{p})$ that we desire to compute is the piecewise step-polynomial representation of this generating function. Applying Theorem 6.3, the proof follows. \square

7.5 Line Removal

In this section, we consider one way of reducing the enumeration of a polyhedron containing lines to the enumeration of a polyhedron without lines. This could be useful if we want to use generating function as part of the enumeration process, since the generating function of a polyhedron containing a line is zero and will not yield any useful information. Suppose we want to enumerate $S_{\mathbf{p}}$. Let P be a parametric polyhedron

$$P = \left\{ (\mathbf{p}, \mathbf{y}, \boldsymbol{\epsilon}) \in \mathbb{Q}^{(n+d+d')} \mid A\mathbf{y} + D\boldsymbol{\epsilon} + C\mathbf{p} + \mathbf{b} \geq 0 \right\}$$

such that $S_{\mathbf{p}} = \pi_d(\mathbb{Z}^{d+d'} \cap P_{\mathbf{p}})$. Suppose further that P contains a line $(\mathbf{c}, \mathbf{a}, \mathbf{b})$, i.e.,

$$(\mathbf{p}, \mathbf{y}, \boldsymbol{\epsilon}) \in P \Leftrightarrow (\mathbf{p} + \mathbf{c}, \mathbf{y} + \mathbf{a}, \boldsymbol{\epsilon} + \mathbf{b}) \in P$$

or

$$c_P(\mathbf{p}) = c_P(\mathbf{p} + \mathbf{c}).$$

If $\mathbf{c} = \mathbf{0}$ then either $\mathbf{a} \neq \mathbf{0}$, which is impossible since $S_{\mathbf{p}}$ is bounded, or $\mathbf{b} \neq \mathbf{0}$, but then the existential variable is redundant and may be removed as in Section 7.2. So we may assume that $\mathbf{c} \neq \mathbf{0}$. Without loss of generality, assume that $c_1 > 0$. Let P' be the slice of P with p_1 between 0 and $c_1 - 1$, i.e.,

$$P' = P \cap \left\{ (\mathbf{p}, \mathbf{y}, \boldsymbol{\epsilon}) \in \mathbb{Q}^{(n+d+d')} \mid 0 \leq p_1 \leq c_1 - 1 \right\}$$

and let $S'_{\mathbf{p}} = \pi_d(\mathbb{Z}^{d+d'} \cap P'_{\mathbf{p}})$. For any other value of \mathbf{p} , the number of points in $P_{\mathbf{p}}$ is equal to the number of points in $P'_{\mathbf{p}'}$ with \mathbf{p}' such that $0 \leq p'_1 \leq c_1 - 1$ and such that \mathbf{p} and \mathbf{p}' differ in an integer multiple of \mathbf{c} , i.e.,

$$c_S(\mathbf{p}) = c_{S'} \left(\mathbf{p} - \left\lfloor \frac{p_1}{c_1} \right\rfloor \mathbf{c} \right).$$

If $c_{S'}(\mathbf{p})$ is a piecewise step-polynomial, then we can also represent $c_S(\mathbf{p})$ as a piecewise step-polynomial if we allow the pieces to be slightly more general than the usual chamber complex. If $c_{S'} = \{ (D'_i, f'_i) \}_{i \in I}$ and $c_S = \{ (D_i, f_i) \}_{i \in I}$, then

$$D_i = \bigcup_{j=-\infty}^{\infty} D'_i + j\mathbf{c} \quad \text{and} \quad f_i(\mathbf{p}) = f'_i(\mathbf{p} - j\mathbf{c}).$$

One way of representing such D_i is to add extra variables q and \mathbf{r} to D'_i as well as the linear equalities $\mathbf{p} = q\mathbf{c} + \mathbf{r}$ and the linear inequality $0 \leq r_1 \leq c_1 - 1$. Each f_i is then a step-polynomial in the \mathbf{r} , i.e., $f_i(\mathbf{p}, q, \mathbf{r}) = f'_i(\mathbf{r})$.

In principle, rays can be removed in a similar way. This is not advisable, however, since, unlike lines, the number of rays is not bounded by a constant.

8 Optimizations

We have seen how to compute the enumerator of parametric polytope as a piecewise step-polynomial in Section 5 and in Section 6 we have seen how to perform various operations on these piecewise step-polynomials. In this section, we will first discuss a trivial optimization for enumerating one-dimensional polytopes and prisms. In the second part, we will discuss various simplifications that may be performed on piecewise step-polynomials.

8.1 One-dimensional Polytopes

If $P_{\mathbf{p}}$ is one-dimensional then applying Barvinok's method is overkill. A one-dimensional polytope has two vertices $l(\mathbf{p}) \leq u(\mathbf{p})$ and the number of integer points inside $P_{\mathbf{p}}$ is simply

$$\lfloor u(\mathbf{p}) \rfloor - \lceil l(\mathbf{p}) \rceil + 1 = \lfloor u(\mathbf{p}) \rfloor + \lfloor -l(\mathbf{p}) \rfloor + 1.$$

Example 46 Consider the parametric polytope

$$P_p = \{ t \mid t \geq 0 \wedge 2t \leq p + 6 \wedge t \leq p \}$$

from Example 32. In the chamber $\{0 \leq p \leq 6\}$, the vertices are $l = 0$ and $u = p$. The number of integer points is therefore $\lfloor p \rfloor + \lfloor -0 \rfloor + 1$. In the chamber $\{6 \leq p\}$, the vertices are $l = 0$ and $u = \frac{p}{2} + 3$. The number of integer points is therefore $\lfloor \frac{p}{2} + 3 \rfloor + \lfloor -0 \rfloor + 1$.

Even if $P_{\mathbf{p}}$ is not itself one-dimensional, it may still contain a one-dimensional *factor*, i.e., $P_{\mathbf{p}}$ may be a prism. We factor out such one-dimensional factors in `ParamPolyhedronReduce`. Ideally, we should factorize $P_{\mathbf{p}}$ completely (Halbwachs et al. 2003). Since the dimension of each factor is smaller than that of $P_{\mathbf{p}}$, we could greatly reduce the computation time by calculating the number of points in each factor separately and multiplying the piecewise step-polynomials afterward as in Section 6.2. That is, we write $P_{\mathbf{p}}$ as

$$P_{\mathbf{p}} = \prod_i P_{\mathbf{p}}^i$$

and then

$$c_P(\mathbf{p}) = \prod c_{P^i}(\mathbf{p}).$$

Example 47 Consider the polytope defined by the following linear inequalities in `PolyLib` notation (see Section B).

```

23 14
1 0 0 0 -1 0 0 0 0 0 0 0 0 8
1 0 1 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 -1 0 0 0 0 0 0 0 3
1 0 0 0 0 0 -1 0 0 0 0 0 0 3
1 0 0 0 1 0 0 0 0 0 0 0 0 0
1 0 0 -1 0 0 0 0 0 0 0 0 0 8
1 0 -1 0 0 0 0 0 0 0 0 0 0 8
1 0 0 0 0 0 1 0 0 0 0 0 0 0
1 0 0 0 0 1 0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 -1 0 0 8
1 0 0 0 0 0 0 -1 0 0 0 0 0 10
1 0 0 0 0 0 0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 -1 0 3
1 0 0 0 0 0 0 0 0 0 0 0 -1 3
1 0 0 0 0 0 0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0 0 -1 0 0 0 8
1 0 0 0 0 0 0 0 -1 0 0 0 0 8
1 0 0 0 0 0 0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0 0 0 0 1 0 0
1 -1 0 0 0 0 0 1 0 0 0 0 0 -1

```

This 12-dimensional (non-parametric) polytope has 23 constraints and 3072 vertices. It contains 7482689280 integer points. Using the method from Section 5, it takes 50s to enumerate this polytope. Factoring out the one-dimensional factors, the problem reduces to

$$136048896 \cdot \#\{(x, y) \mid -x + y - 1 \geq 0, -y + 10 \geq 0, x \geq 0\}.$$

The total computation time in this case is 5s. The reason that the computation time is still high is that `PolyLib` insists on computing all vertices of the polytope.

Example 48 Let us consider an example where some of the coefficients defining the polyhedron are themselves parameters, i.e., an example with some *quadratic* constraints. Note that is just an example. Our implementation does not support quadratic constraints and Barvinok’s algorithm does not apply if the generators of the supporting cones depend on parameters.

Consider the following parametric polytope, which is part of an example from Boulet and Redon (1998a),

$$P_{(n,m)} = \{ (i_1, i_2, j_1, j_2, q) \mid 1 \leq j_1 \leq n \wedge 1 \leq j_2 \leq m \wedge i_1 = j_1 \wedge 1 \leq i_2 \leq m \wedge j_2 - 1 = 8q \},$$

where the number 8 refers to number of processor elements in each direction in the processor array (of size 8×8). Let us enumerate this set with this number parameterized by N . We can immediately eliminate i_1 and then factorize $P_{(n,m)}$ into three parametric polytopes $P_{(n,m)} = Q_{(n,m)} \times R_{(n,m)} \times S_{(n,m)}$ with

$$\begin{aligned} Q_{(n,m)} &= \{ j_1 \mid 1 \leq j_1 \leq n \} \\ R_{(n,m)} &= \{ (j_2, q) \mid 1 \leq j_2 \leq m \wedge j_2 - 1 = Nq \} \\ S_{(n,m)} &= \{ i_2 \mid 1 \leq i_2 \leq m \}. \end{aligned}$$

Obviously, $c_Q = n$ and $c_S = m$. We make $R_{(n,m)}$ full-dimensional by performing a change of basis $j = j_2 - Nq$ and obtain

$$\begin{aligned} R'_{(n,m)} &= \{ j \mid j = 1 \} \\ R''_{(n,m)} &= \{ q \mid 0 \leq Nq \leq m - 1 \}. \end{aligned}$$

The enumerator of $P_{(n,m)}$ is then

$$\begin{aligned} c_P(n, m) &= c_Q(n, m) \cdot c_{R'}(n, m) \cdot c_{R''}(n, m) \cdot c_S(n, m) \\ &= n \cdot 1 \cdot \left(\left\lfloor \frac{m-1}{N} \right\rfloor + 1 \right) \cdot m \\ &= \left(\left\lfloor \frac{m-1}{N} \right\rfloor + 1 \right) nm. \end{aligned}$$

8.2 Simplification of Step-polynomials

Many (piecewise) step-polynomials represent the same function on \mathbb{Z}^n . As a trivial example, $\lfloor \frac{x}{2} + \frac{1}{4} \rfloor$ and $\lfloor \frac{x}{2} \rfloor$ represent the same function on the integer numbers. In particular, there are many ways of representing the zero function. Ideally we would want to detect (and remove) such zero functions and compute a unique piecewise step-polynomial representation for each function on \mathbb{Z}^n . Failing this, we present here some simplifications on piecewise step-polynomial that we have implemented. We explain the simplifications in terms of fractional parts $\{x\} = x - \lfloor x \rfloor$ rather than floors since, as explained in Appendix A.2, this is how we currently internally represent quasi-polynomials. This representation was chosen because of the more direct correspondence to quasi-polynomials defined in terms of periodic numbers. Some other simplifications that are more intimately tied to the internal representation are explained in Appendix A.3.

Note that it is in principle possible to check whether a piecewise step-polynomial is equal to the zero function. If the piecewise step-polynomial is given, then this can even be done in polynomial time. We just convert the piecewise step-polynomial to its corresponding rational generating function and then use the algorithm of Woods (2004) to check whether this rational generating function is zero. We have not implemented this functionality.

In our internal representation, we use an additional element, which we call *strides*. Strides are used to represent enumerators that depend on the residue class modulo some integer of a linear expression in the parameters. The residue class is selected using a fractional part $f(\mathbf{p}) = \{h(\mathbf{p})\}$ of a degree-1 polynomial $h(\mathbf{p})$. A stride has two branches representing a quasi-polynomial, one e_1 for the selected residue class and one e_2 for all the other residue classes. (Strides may be nested.) The value of the stride

$$e' = [f = 0] \cdot e_1 + [f \neq 0] \cdot e_2$$

is

$$e'(\mathbf{p}) = \begin{cases} e_1(\mathbf{p}) & \text{if } f(\mathbf{p}) = 0 \\ e_2(\mathbf{p}) & \text{otherwise} \end{cases}$$

Note that a stride is equivalent to

$$e_1(\mathbf{p}) + \left(\{h(\mathbf{p})\} - \left\{ h(\mathbf{p}) - \frac{1}{m} \right\} + \frac{m-1}{m} \right) (e_2(\mathbf{p}) - e_1(\mathbf{p})),$$

with m the common denominator of the coefficients of $h(\mathbf{p})$. Often, the stride representation is more succinct.

Example 49 Consider the parametric polytope

$$P_p = \{x \mid 4x = p\}.$$

The enumerator is

$$c_P(p) = \left[\left\{ \frac{p}{4} \right\} = 0 \right] \cdot 1$$

or

$$c_P(p) = 1 - \left(\left\{ \frac{p}{4} \right\} - \left\{ \frac{p}{4} - \frac{1}{4} \right\} + \frac{3}{4} \right).$$

We first describe some relatively cheap simplifications implemented in `reduce_value`. They are only *relatively* cheap because the simplification may induce a reordering of the internal structure representing the quasi-polynomial. The reason is that, as explained in Appendix A.3, we enforce a fixed nesting order on the internal representation.

- “Normalization” of arguments of fractional parts

If m is the common denominator of the coefficients of the argument h of a fractional part, then we have the following identity:

$$\{h\} = \frac{m-1}{m} - \left\{ -h - \frac{1}{m} \right\}.$$

We can therefore ensure that the leading coefficient is less than or equal to $\frac{1}{2}$. If it is equal to $\frac{1}{2}$, then we can do the same for next coefficient(s). This normalization allows us to for example simplify $\left\{ \frac{3}{4}p \right\} + \left\{ \frac{1}{4}p + \frac{3}{4} \right\}$ to $\frac{3}{4}$ since we may rewrite $\left\{ \frac{3}{4}p \right\}$ into $\frac{3}{4} - \left\{ \frac{1}{4}p + \frac{3}{4} \right\}$.

- Move out “constant part” of constant from fractional part

If the denominator of the constant in the argument of the fractional part does not divide the denominator m of the other coefficients in the same argument, then the difference with the nearest multiple of $\frac{1}{m}$ may be moved out of the fractional part. E.g., $\left\{ \frac{1}{4}p + \frac{5}{8} \right\} = \left\{ \frac{1}{4}p + \frac{1}{2} \right\} + \frac{1}{8}$.

- Elimination of variables using equalities in chambers

Although all chambers are initially of full dimension, addition or multiplication of quasi-polynomials may result in lower-dimensional chambers. The resulting equalities can be used to eliminate some variables. E.g., if $p = q$ inside a chamber, then $p - q$ can be simplified to 0.

- Reduction of the arguments of fractional parts using stride information

This is similar to the previous simplification, except that we use identities modulo an integer.

Example 50 Suppose we know that $\{\frac{1}{4}i\} = 0$. Then we may perform the following simplification.

$$\begin{aligned}\left\{\frac{1}{8}i + \frac{7}{8}\right\} &= \frac{7}{8} - \left\{\frac{7}{8}i + \frac{1}{8} - \frac{1}{8}\right\} \\ &= \frac{7}{8} - \left\{\frac{1}{8}i + 3\frac{1}{4}i + \frac{1}{8} - \frac{1}{8}\right\} \\ &= \frac{7}{8} - \left\{\frac{1}{8}i\right\}\end{aligned}$$

Note that this type of simplification has so far only been partially implemented.

We can also use negative information. That is, if we know that $\{\frac{E}{m}\} \neq 0$, with E a degree-1 polynomial with integer coefficients, then we can replace $\{\frac{E}{m}\}$ by $\{\frac{E-1}{m}\} + \frac{1}{m}$.

The following simplifications have been implemented in the function `evaluate_range_reduction`. As the name suggests they depend on the range of values that a functional g attains over a chamber C .

- If there exists an $i \in \mathbb{Z}$ such that $i \leq g(C) < i + 1$ then $\{g(\mathbf{p})\}$ may be replaced by $g(\mathbf{p}) - i$.
- If there exists an $i \in \mathbb{Z}$ such that $i < g(C) < i + 1$ then

$$[\{f(\mathbf{p})\} = 0] \cdot e_1 + [\{f(\mathbf{p})\} \neq 0] \cdot e_2$$

may be replaced by e_2 .

- If there exists an $i \in \mathbb{Z}$ such that $i = g(C)$ then

$$[\{f(\mathbf{p})\} = 0] \cdot e_1 + [\{f(\mathbf{p})\} \neq 0] \cdot e_2$$

may be replaced by e_1 .

- If there exists an $i \in \mathbb{Z}$ such that $i - 1 < g(C) < i + 1$ then

$$[\{f(\mathbf{p})\} = 0] \cdot e_1 + [\{f(\mathbf{p})\} \neq 0] \cdot e_2$$

may be replaced by

$$[\{f(\mathbf{p})\} = 0] \cdot e'_1 + [\{f(\mathbf{p})\} \neq 0] \cdot e_2$$

where e'_1 the reduction of e_1 using $g(\mathbf{p}) = i$.

- If there exists an $i \in \mathbb{Z}$ such that $i \leq g(C) < i + 2$ then $g'(\mathbf{p}) := g(\mathbf{p}) - i$ is such that $0 \leq g'(C) < 2$ and so

$$g'(\mathbf{p}) - \{g'(\mathbf{p})\} = \lfloor g'(\mathbf{p}) \rfloor = \lfloor g'(\mathbf{p}) \rfloor^2 = g'(\mathbf{p}) - 2\{g'(\mathbf{p})\}g'(\mathbf{p}) + \{g'(\mathbf{p})\}^2$$

on C . In other words,

$$\{g'(\mathbf{p})\}^2 = -g'(\mathbf{p})^2 + (2\{g'(\mathbf{p})\} + 1)g'(\mathbf{p}) - \{g'(\mathbf{p})\}.$$

on C and so we may replace $\{g(\mathbf{p})\}^2$ by

$$-(g(\mathbf{p}) - i)^2 + (2\{g(\mathbf{p})\} + 1)(g(\mathbf{p}) - i) - \{g(\mathbf{p})\}.$$

This may look more complicated, but we have reduced the degree of the polynomial in the fractional parts.

Note that $\{g(\mathbf{p}) + \langle \mathbf{k}, \mathbf{p} \rangle\}$ for any $\mathbf{k} \in \mathbb{Z}^d$ represents the same function as $\{g(\mathbf{p})\}$. To simplify as much as possible, we would have to check the above conditions for any value of \mathbf{k} . In our current implementation, we only perform the checks for a single representative.

The most expensive simplification is implemented in `evaluate_combine`. The basic idea is to replace two chambers with the same quasi-polynomial by a single chamber which is the union of the two original chambers. For any pair of chambers with their associated quasi-polynomials (C_1, e_1) and (C_2, e_2) , the function attempts to reduce e_2 in the context C_1 . If the result is e_1 , then the pair of chambers is replaced by the single chamber $C_1 \cup C_2$ with associated quasi-polynomial e_1 .

```

for(i=1;1000*i<=N;i++)
  for(j=1;1000*j<=N;j++)
    S1;

```

(a) How many times does S1 execute?

$$\left\{ \begin{array}{ll} \frac{N^2}{1000000} & \text{if 1000 divides } N \\ \frac{(N-1)N}{1000000} - \frac{N-1}{1000000} & \text{if 1000 divides } N - 1 \\ \dots & \\ \frac{(N-999)N}{1000000} - \frac{999N-998001}{1000000} & \text{if 1000 divides } N - 999 \end{array} \right.$$

(b) Solution computed by hand using the method of Pugh (1994)

$$\left\lfloor \frac{N}{1000} \right\rfloor^2$$

(c) Solution generated by our method

Figure 35: Example of an answer generated by Pugh’s method. The number of different cases in Pugh’s answer is as large as the factor of *i* and *j* in the program in (a) (1000 in this example). Therefore, the solution size of Pugh’s method is exponentially large.

9 Related Work

In the compiler community, two methods are often cited for enumerating parametric sets: Pugh (1994) and Clauss and Loechner (1998). We discuss these methods in some more detail and briefly mentions some other related techniques.

9.1 Pugh’s method

Pugh’s method is a general technique for enumerating Presburger sets. It consists of a set of simplification and rewrite rules and the application of a set of standard summation formulas for some base cases. In contrast to our technique and that of Clauss and Loechner (1998), his technique does not appear to have ever been implemented. Furthermore, the description of the method of Pugh (1994) fails to indicate which rewrite rules to use when several are applicable. We are therefore unable to systematically compare our results to those that would or would not be obtained using that method. Application by hand on the example in Figure 35 shows that even for parametric polytopes, the solution may be exponentially large, even for fixed dimensions.

Pugh (1994) proposes two algorithms to eliminate variables, one from Pugh (1992), resulting in a collection of possibly overlapping sets, and a new one resulting in disjoint sets. The basic idea behind both algorithms is to use the Omega test to detect the part of the polytope where projection is exact, i.e., where the inverse image of the projection contains at least one integer point. The projection of this part is called the “dark shadow”. The remaining parts of the projection are collected in possibly overlapping “splinters”. The number of splinters is linear (in the new algorithm even quadratic) in some of the coefficients that appear in the input and so exponential in the input size. The new algorithm avoids this overlap, but it is formulated for a single variable. It is not at all clear whether successively applying the algorithm twice to eliminate two variables would still yield disjoint sets. For eliminating a single variable, the method from Section 7.1 is more appropriate. Note that the new algorithm contains a check that is essentially the same as our constraint (36) for detecting unique existential variables. If the check holds, then the algorithm still eliminates the variable, using exponential splintering no less, even though, as we pointed out, this is not needed for counting purposes.

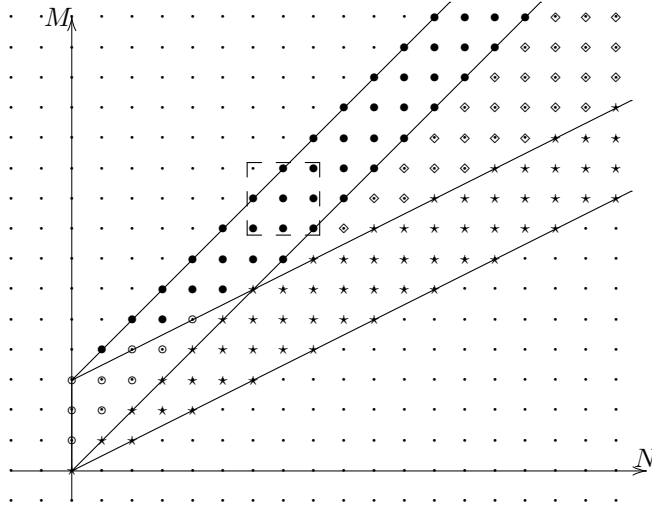


Figure 36: Geometrical representation of the chambers of Equation (4). The points in the different chambers are shown by different symbols. The domain marked by \bullet is degenerate.

9.2 Claus's method

As already explained in Section 3.3, our technique and Claus's method share the decomposition into chambers, but whereas our technique produces polynomially sized quasi-polynomials, the method of Claus and Loechner (1998) can produce exponentially sized quasi-polynomials or sometimes no solution at all.

9.2.1 Interpolation and Degenerate Domains

Based on the knowledge of the structure of the solution (Theorem 3.11), Claus and Loechner (1998) calculate the number of points in a set of instances of $P_{\mathbf{p}}$ for fixed values of \mathbf{p} in a given chamber, called *initial countings*, and then calculate the quasi-polynomial for this chamber through interpolation. They represent periodic numbers by lookup-tables, containing a separate value for each residue class (see Appendix A.1), and during their calculations they directly determine the elements in these lookup-tables. To interpolate a d -dimensional Ehrhart polynomial with periods q_i their algorithm requires $\prod_{i=1}^n (d+1)q_i$ initial countings. Since the implementation is based on Vandermonde interpolation, it searches for fixed parameter values located in a hyperrectangle. However, it is not always possible to find a hyperrectangle of the correct size that is completely inside a given chamber.

For example, consider the chamber $N \leq M \leq N + 3 \wedge N \leq 2M - 7$ from the polytope in Equation (4). This chamber is geometrically represented by \bullet s in Figure 36. For this chamber, the period in both dimensions is 1, and the implementation of Claus's method in `PolyLib` (Loechner 1999) searches for a solution of the following form

$$aN^2M^2 + bN^2M + cN^2 + dNM^2 + eNM + fN + gM^2 + hM + i. \quad (40)$$

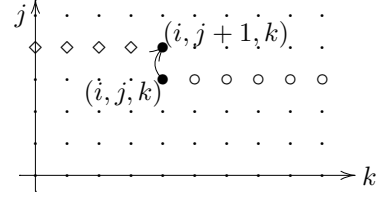
To find the nine unknown values, Claus's method looks for a 3×3 rectangle in the chamber for which it can compute initial countings. As is clear from Figure 36, however, no such rectangle can be found and the method fails to compute the solution. The chambers where this problem occurs are known as *degenerate domains* and occur in practice, as reported by, e.g., Turjan et al. (2002) and Beyls (2004). This problem could in principle be solved by considering other interpolation techniques such as those used by Manocha (1993) or by adding an extra parameter (Nootaert et al. 2005).

```

do i = 0, 199
  do j = 0, 199
    s = 0
    do k = 0, 199
      s = s + A[i][k] * B[k][j]
    enddo
    C[i][k] = s
  enddo
enddo

```

(a) Source code



(b) Intermediate accesses

Figure 37: Matrix multiplication.

9.2.2 Large Solution Size

Since the periods q_i are bounded only by the *value* of the coefficients in the input, they can be *exponential* in terms of the input size and so the worst-case computation time for the Ehrhart polynomial in a single chamber is exponential even for fixed dimension. Since Clauss’s method is based on the lookup-table representation of periodic numbers, the output size is also exponential in the input size.

Consider, for example, the program in Figure 37 (matrix multiplication). Suppose we want to count the number of distinct Translation Lookaside Buffer (TLB) pages accessed between two consecutive accesses to the same TLB page. This count is an indication of the number of TLB page misses that can be expected and is called the reuse distance (Beyls 2004).

For simplicity, we will assume that $A[i][k]$ and $B[k][j]$ access different TLB pages and we will concentrate on $A[i][k]$. We assume that A is a 200×200 matrix, which is laid out in column major order, and starts at address zero. Furthermore, an element size of 4 bytes is assumed. As such, $A[i][k]$ is located at address $4 \times (200k + i)$.

Iterations (i, j, k) and $(i, j + 1, k)$ access the same array element $A[i][k]$. Figure 37b shows the iterations that are executed between these two iterations: iterations $(i, j, k + 1 \dots 199)$ (\circ on the figure) and iterations $(i, j + 1, 0 \dots k - 1)$ (\diamond on the figure). The set of TLB pages accessed by the \circ -iterations can be described as

$$S_1 = \left\{ t \mid \exists k' : t = \left\lfloor \frac{800k' + 4i}{L} \right\rfloor \wedge 0 \leq i, j, k \leq 199 \wedge k + 1 \leq k' \leq 199 \right\},$$

where i, j and k are parameters. Assuming page size $L = 4096$, this can be written as a set of linear constraints:

$$S_1 = \{ t \mid \exists k' : 1024t \leq 200k' + i \leq 1024t + 1023 \wedge 0 \leq i, j, k \leq 199 \wedge k + 1 \leq k' \leq 199 \}$$

and further simplified to (e.g., using Ω (Kelly et al. 1996))

$$S_1 = \{ t \mid 0 \leq i \wedge 1024t - 39800 \leq i \leq 199 \wedge 0 \leq k \leq 198 \wedge 0 \leq j \leq 199 \wedge i + 200k \leq 823 + 1024t \}.$$

We obtain a similar expression S_2 for the \diamond -iterations. The total count of TLB pages is $\#(S_1 \cup S_2) = \#S_1 + \#(S_2 \setminus S_1)$. Concentrating on S_1 , we see that it is a one-dimensional polytope and using PolyLib we can find out that its vertices are

$$\frac{i}{1024} + 25\frac{k}{128} - \frac{823}{1024} \quad \text{and} \quad \frac{i}{1024} + \frac{4975}{128}.$$

Since the dimension of this polytope is $d = 1$ and the periods are $q_i = 1024$, $q_j = 1$ and $q_k = 128$, the interpolation method requires $2^3 \cdot 1024 \cdot 128$ initial countings. If we assume we need two bytes to represent a value, then we need up to 256KiB just to store a single periodic number in the

	#C	#DD	interpolation	Barvinok	
				table	fractional
e16	4	0	16.076s	0.844s	0.702s
isnm	2	0	5.153s	0.038s	0.027s
g14	6	2	(0.688s)	0.040s	0.040s
RD1	2	1	(151.524s)	0.224s	0.068s
RD2	1	0	26.920s	3.207s	0.026s
CME	5	?	∞	∞	0.333s

Table 4: Comparison between the method of Clauss and Loechner (1998) and the method of Section 5.

output. Using our own technique, we obtain the following equivalent, but much shorter solution in polynomial time (for fixed dimensions):

$$\left\lfloor \frac{i + 888}{1024} \right\rfloor - \left\lfloor \frac{i + 200k + 199}{1024} \right\rfloor + 39.$$

Although the degeneracy problem can in principle be solved, the problem of an exponentially-sized output in the worst-case is intrinsic to this approach.

A number of proposed compiler methods (e.g., Beyls 2004; Franke and O’Boyle 2003; Loechner et al. 2002; Turjan et al. 2002) hard-code the resulting quasi-polynomials in the program they are optimizing. For these optimizations, large quasi-polynomials result in large binaries, making the optimizations less interesting, especially in an embedded systems context. Using our method, the size of the resulting quasi-polynomials does not grow exponentially.

9.2.3 Comparison

Table 4 shows a small comparison between the method of Clauss and Loechner (1998) (“interpolation”) to the method of Section 5 (“Barvinok”). The first column shows the number of chambers, the second the number of degenerate domains (only relevant for the interpolation method) and the remaining columns show the computation times for the interpolation method, our method when using lookup-tables for periodic numbers and our method when using fractional parts for periodic numbers. For all three methods, computations were performed in exact long integer arithmetic using `GMP`. It should be noted that our library currently requires computation in exact long integer arithmetic, whereas the interpolation method as implemented in `PolyLib` can also use 32-bit or 64-bit integers. Computation with fixed-size integer can be significantly faster.

The first three polytopes can be found in the `PolyLib` distribution (Loechner 1999). The first two contain some moderate-sized periodic numbers. Our method is clearly faster than interpolation and the fractional part representation is also slightly faster than the periodic number representation. The third polytope results in a few degenerate domains for the interpolation method. Again, our method is faster even though it produces extra results when compared to the interpolation method, which simply fails on its degenerate domains. The next two polytopes appear in the context of reuse distances (Beyls 2004). The speed improvements are even more dramatic in these cases. RD2 is the example discussed in the previous section. The final polytope is based on the Cache Miss Equations (CME) (Ghosh et al. 1999). For this polytope, both the interpolation method and our method with periodic number representation had to be aborted because they slowly used up all available memory. The experiments were performed on an Athlon MP 1500+ with 512MiB internal memory.

9.3 Other Techniques

`LatTE` (De Loera et al. 2003b) was the first known implementation of Barvinok’s counting algorithm. It initially only counted the number of points in non-parametric polytopes, but it has been

type	Chatterjee	Balasa	Boulet
Sets	8+13	4	1
Fixed	0+2	14	5
Change+Fixed	0+0	0	2
Unique	8+9	0	0
Change+Unique	0+0	0	0
Redundant	0+0	2	1
Split	0+0	0	0
PIP	0+0	0	0

Table 5: Rule application distribution for polytopes originating from cache miss analysis (Chatterjee), memory size estimation (Balasa) and communication volume computation (Boulet).

extended to also compute the Ehrhart series of a polytope (see Section 3.1) by what the authors call the homogenized Barvinok algorithm (De Loera et al. 2003a).

The technique proposed by Clauss (1996) and implemented by Seghir (2002) for removing a single existential variable was discussed in Section 7.1. The general, but worst case exponential technique for removing existential variables using PIP (Feautrier 1988) proposed by Boulet and Redon (1998a) was discussed in Section 7.3. They proposed to use the method of Clauss and Loechner (1998) to enumerate the resulting parametric polytopes. The appendix of their technical report (Boulet and Redon 1998b) indicates that using this method, their combined technique cannot compute the enumerator fully automatically. Meister (2004) proposes a similar technique using his more general periodic polyhedra instead of PIP. No implementation has been reported.

Beck (2004) describes a general technique for computing vector partition functions from their generating functions. For each variable in the generating function, he computes partial fractions and then determines the constant term of the Laurent expansion. The latter part is similar to our specialization technique from Section 5.3, except that we need the expansion at 1, whereas Beck (2004) needs the expansion at 0. He does not provide a complexity analysis, but standard techniques for computing partial fractions (Henrici 1974) are exponential, even for fixed dimensions.

Recently some advances have been made towards automata-based counting (Boigelot and Lator 2004; Parker and Chatterjee 2004). They handle the general case of Presburger formulas but they do not support symbolic parameters. Not surprisingly, these techniques are exponential, even for fixed dimensions. Preliminary experiments have shown that in the intersection of the application domains, i.e., for non-parametric polytopes, our method is as fast or faster (up to a factor 100 in some exceptional cases) than Parker’s, except for polytopes with a large number (say thousands) of vertices. For such polytopes, the homogenized Barvinok algorithm as implemented by `LattE` would be more appropriate.⁷

10 Applications and Experiments

In this section we list some applications that require the enumeration of parametric polytopes or more general parametric sets and that therefore may benefit from our implementation. We also mention some experimental results, comparing the method of Section 7.2 to that of Section 7.3. One of these experiments counts the number of times a particular rule from Section 7.2 was used. Table 5 shows the results. The row “Fixed” refers to the special case of a unique existential variable determined by an equality; “Change” refers to a change of basis. The individual test sets in the columns are explained below. For more experimental results, we refer to Section 9.2, Verdoolaege et al. (2004b) and Verdoolaege et al. (2004a). The last reference is also the source of the experiments in this section.

⁷This version embeds a polytope P in a single cone with a polar that has few rays if P has few facets (De Loera et al. 2004).

PIP	3.16s	53.34s	82.95s	74.43s	4.20s	56.07s	87.62s	87.62s
Rules	3.67s	53.52s	80.72s	68.02s	4.14s	56.23s	88.03s	80.14s

Table 6: Computation time for Chatterjee’s sets.

Enumeration problems occur frequently in the context of cache analysis. Clauss (1996) was the first to show the applicability of Ehrhart’s theory in computer science and includes an example that extends the techniques of Ferrante et al. (1991) for counting the number of cache lines accessed by a loop to parameterized loops. The CME of Ghosh et al. (1999) give a detailed representation of cache behavior. The authors refer to Clauss (1996) as a possible way to help the automation of their technique. Beyls (2004) proposes a technique based on reuse distance and requiring the enumeration of parametric sets.

Chatterjee et al. (2001) propose a technique, which, in contrast to the use of CME, is exact. They model cache behavior using Presburger formulas and provide a (large) formula (Chatterjee et al. 2001, Figure 4), parametrized by the start address of an array, that represents the number of conflict misses in an example program. Table 6 shows the computation times on a 2.66GHz Pentium4 machine for the 8 disjuncts in this formula each considered as a separate set. Each of these disjuncts contains a single existentially quantified variable. The computation time for these sets is larger than for trivial examples, but still reasonable. Using the method of Clauss and Loechner (1998) (see Section 9.2) to enumerate one of the resulting parametric polytopes did not produce a result even after 15 hours. To count the number of solutions to the whole formula, we first compute the disjoint union using *Omega*, which only takes a fraction of a second. The resulting 13 sets exhibit computation times comparable to those in Table 6. The rules that are applied are shown in Table 5, with the original disjuncts on the left and the disjoint sets on the right.

Another typical counting problem is the calculation of the memory requirements of a program (Balasa et al. 1995; Anantharaman and Pande 1998; Zhao and Malik 2000). The basic idea is to count the number or array elements that are live at a certain point during the execution. The maximum number of live array elements is then the total memory requirement. Under certain conditions, the live elements can be described by linear equations. To the best of our knowledge, none of these techniques have used exact parametric counting, presumably because it was considered to costly. For example, Zhao and Malik (2000) do refer to Clauss and Loechner (1998) for enumerating the parametric polytopes that result when considering loops with parametric bounds, but then continue to develop their own heuristics. Balasa et al. (1995) have an example where they count the number of array elements accessed by 4 references in a motion estimation loop kernel, for a number of different values of the symbolic loop bounds. We considered the same loop kernel, but handled the symbolic loop bounds parametrically, thereby obtaining a single solution for all possible values of the symbolic loop bounds. The execution times for the four references using Clauss’s method (after removing equalities; see below) were respectively 1.38s, 0.01s, 1.41s and 1.41s. Using our method, we obtained the symbolic results in respectively 0.06s, 0.01s, 0.07s and 0.04s.

Boulet and Redon (1998a) and Heine and Slowik (2000) consider the problem of data distribution on parallel systems. They use enumerators of parametric sets in their computations of the communication volume. Boulet and Redon (1998b) report that manual intervention is needed to efficiently calculate the enumerator of the communication volume of a particular program on an 8×8 processor array. Indeed, directly applying Clauss’s method on the output from PIP, as they apparently propose, leads to a computation time of 713s. If we consider the same problem on a 64×64 array, the computation time even increases to 6855s. The output from PIP contains a few equalities, however, and the implementation of Clauss’s method apparently does not exploit those. Removing these equalities in a preprocessing step, we obtain the more reasonable times of 0.04s (8×8) and 1.43s (64×64). Using our own method, which removes equalities automatically, instead of Clauss’s we obtain a result in 0.01s for both sizes. Our heuristics also work for this

example, as shown in Table 5.

Lisper (2003) uses parametric counting in his Worst-Case Execution Time (WCET) analysis to obtain safe upper bounds of execution times in the context of real-time systems. In his example, he uses the techniques by Pugh (1994), but an actual implementation might benefit from using our method. Also in the context of real-time systems, Braberman et al. (2003) use enumerators of parametric polytopes to automatically determine the size of the memory region associated with a scope in function of the arguments of the `java` method that defines the scope. Scoped memory (Bollella and Gosling 2000) is used here to avoid the unpredictable behavior of regular garbage collection on real-time systems. In their conversion of pointers to arrays (array recovery), Franke and O’Boyle (2003) count the number of times a pointer is incremented inside a loop nest. If these loops have parametrized bounds, then this results in a parametric counting problem.

Loechner et al. (2002) try to reorder the data in memory according to the order in which it is accessed. The access order is a function of the array indices and can be computed as the enumerator of a parametric polytope. In the context of converting `matlab` programs to process networks, Turjan et al. (2002) use a similar idea to compute their rank function, which returns the number of iteration points of a for-statement executed before a given iteration point, as an enumerator. In the next step of converting the process networks to hardware (descriptions), Derrien et al. (2003) also use parametric counting to optimize the resulting hardware. Earlier, Rijpkema et al. (1999) used parametric enumerations during the linearization step of their method for converting parameterized `matlab` programs to process networks. In a similar context, Bednara et al. (2002) encounter parametric counting problems during optimization of the control of VLSI or FPGA-based processor arrays.

11 Conclusions and Future Work

We have presented the first known implementation of Barvinok’s enumeration algorithm applied to parametric polytopes. The resulting enumerator can be obtained either as a piecewise step-polynomial or as a rational generating function and both can be obtained in polynomial time. Furthermore, each representation can be converted into the other in polynomial time. For piecewise step-polynomials this is a significant improvement over known, exponential, methods for enumerating parametric polytopes, making a whole class of known optimization techniques practically usable. For rational generating functions this is a major extension of an earlier implementation of Barvinok’s algorithm for the subproblem of computing Ehrhart series.

We have also considered the extension to projections of the integer points in parametric polytopes. We have presented a polynomial-time algorithm (without implementation) for computing the enumerator of such sets as a piecewise step-polynomial by combining earlier results on rational generating functions with our polynomial-time conversion algorithm. We have further presented two alternative methods (with implementation), one new and one proposed before, for reducing such problems to parametric polytope enumeration problems. The first is not generally applicable and the second is worst-case exponential, but both are easier to implement and appear to work fairly well in practice. This reduction, together with our polynomial method for enumerating parametric polytopes, yields a method for enumerating parametric projected sets that can solve many problems that were previously considered very difficult or even unsolvable.

The resulting piecewise step-polynomials, though polynomially sized, can in some cases still be relatively large, even with the simplifications we perform already. Further research into additional simplifications is still needed. A more detailed study of the relative benefits of piecewise step-polynomials and rational generating functions, in particular a thorough evaluation of the assumption that piecewise step-polynomials are more appropriate for compiler optimization problems is also a point of further research. Finally, an actual implementation of the polynomial-time algorithm for enumerating parametric projected sets would be very interesting.

A Internal Representation

Our `barvinok` library is built on top of `PolyLib` (Wilde 1993; Loechner 1999). In particular, it reuses the implementations of the algorithm of Loechner and Wilde (1997) for computing parametric vertices and the algorithm of Clauss and Loechner (1998) for computing chamber decompositions. Initially, our library was meant to be a replacement for the algorithm of Clauss and Loechner (1998), also implemented in `PolyLib`, for computing quasi-polynomials. To ease the transition of application programs we tried to reuse the existing data structures as much as possible.

A.1 Existing Data Structures

Inside `PolyLib` integer values are represented by the `Value` data type. Depending on a configure option, the data type may either be a 32-bit integer, a 64-bit integer or an arbitrary precision integer using `GMP`. The `barvinok` library requires that `PolyLib` is compiled with support for arbitrary precision integers.

The basic structure for representing (unions of) polyhedra is a `Polyhedron`.

```
typedef struct polyhedron {
    unsigned Dimension, NbConstraints, NbRays, NbEq, NbBid;
    Value **Constraint;
    Value **Ray;
    Value *p_Init;
    int p_Init_size;
    struct polyhedron *next;
} Polyhedron;
```

The attribute `Dimension` is the dimension of the ambient space, i.e., the number of variables. The attributes `Constraint` and `Ray` point to two-dimensional arrays of constraints and generators, respectively. The number of rows is stored in `NbConstraints` and `NbRays`, respectively. The number of columns in both arrays is equal to `1+Dimension+1`. The first column of `Constraint` is either 0 or 1 depending on whether the constraint is an equality (0) or an inequality (1). The number of equalities is stored in `NbEq`. If the constraint is $\langle \mathbf{a}, \mathbf{x} \rangle + c \geq 0$, then the next columns contain the coefficients a_i and the final column contains the constant c . The first column of `Ray` is either 0 or 1 depending on whether the generator is a line (0) or a vertex or ray (1). The number of lines is stored in `NbBid`. Let d be the lcm of the denominators of the coordinates of a vertex \mathbf{v} , then the next columns contain dv_i and the final column contains d . For a ray, the final column contains 0. The field `next` points to the next polyhedron in the union of polyhedra. It is 0 if this is the last (or only) polyhedron in the union. For more information on this structure, we refer to Wilde (1993).

Quasi-polynomials are represented using the `evaluate` and `enode` structures.

```
typedef enum { polynomial, periodic, evector } enode_type;

typedef struct _evaluate {
    Value d; /* denominator */
    union {
        Value n; /* numerator (if denominator != 0) */
        struct _enode *p; /* pointer (if denominator == 0) */
    } x;
} evaluate;

typedef struct _enode {
    enode_type type; /* polynomial or periodic or evector */
    int size; /* number of attached pointers */
    int pos; /* parameter position */
}
```

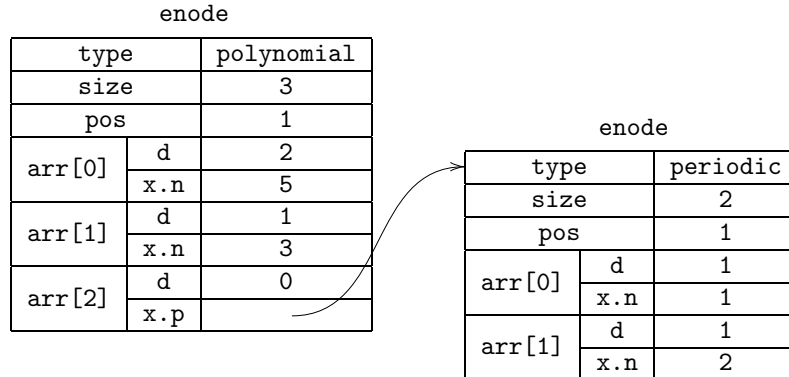


Figure 38: The quasi-polynomial $[1, 2]_p p^2 + 3p + \frac{5}{2}$.

```

    evaluate arr[1];          /* array of rational/pointer */
} enode;

```

If the field `d` of an `evaluate` is zero, then the `evaluate` is a placeholder for a pointer to an `enode`, stored in `x.p`. Otherwise, the `evaluate` is a rational number with numerator `x.n` and denominator `d`. An `enode` is either a `polynomial` or a `periodic`, depending on the value of `type`. The length of the array `arr` is stored in `size`. For a `polynomial`, `arr` contains the coefficients. For a `periodic`, it contains the values for the different residue classes modulo the parameter indicated by `pos`. For a `polynomial`, `pos` refers to the variable of the polynomial. The value of `pos` is 1 for the first parameter. That is, if the value of `pos` is 1 and the first parameter is p , and if the length of the array is l , then in case it is a `polynomial`, the `enode` represents

$$\text{arr}[0] + \text{arr}[1]p + \text{arr}[2]p^2 + \dots + \text{arr}[l-1]p^{l-1}.$$

If it is a `periodic`, then it represents

$$[\text{arr}[0], \text{arr}[1], \text{arr}[2], \dots, \text{arr}[l-1]]_p.$$

Note that the elements of a `periodic` may themselves be other `periodics` or even `polynomials`. In our library, we only allow the elements of a `periodic` to be other `periodics` or rational numbers. The chambers and their corresponding quasi-polynomial are stored in `Enumeration` structures.

```

typedef struct _enumeration {
    Polyhedron *ValidityDomain; /* constraints on the parameters */
    evaluate EP;                /* dimension = combined space */
    struct _enumeration *next; /* Ehrhart Polynomial, corresponding
                               to parameter values inside the
                               domain ValidityDomain below */
} Enumeration;

```

For more information on these structures, we refer to Loechner (1999).

Example 51 Figure 38 is a skillful reconstruction of Figure 2 from Loechner (1999). It shows the contents of the `enode` structures representing the quasi-polynomial $[1, 2]_p p^2 + 3p + \frac{5}{2}$.

A.2 Data Structures for Quasi-polynomials

Similarly to Loechner (1999), we do not represent our quasi-polynomials internally as step-polynomials, but as polynomials with periodic numbers for coefficients. However, we also allow our periodic numbers to be represented by fractional parts of degree-1 polynomials rather than

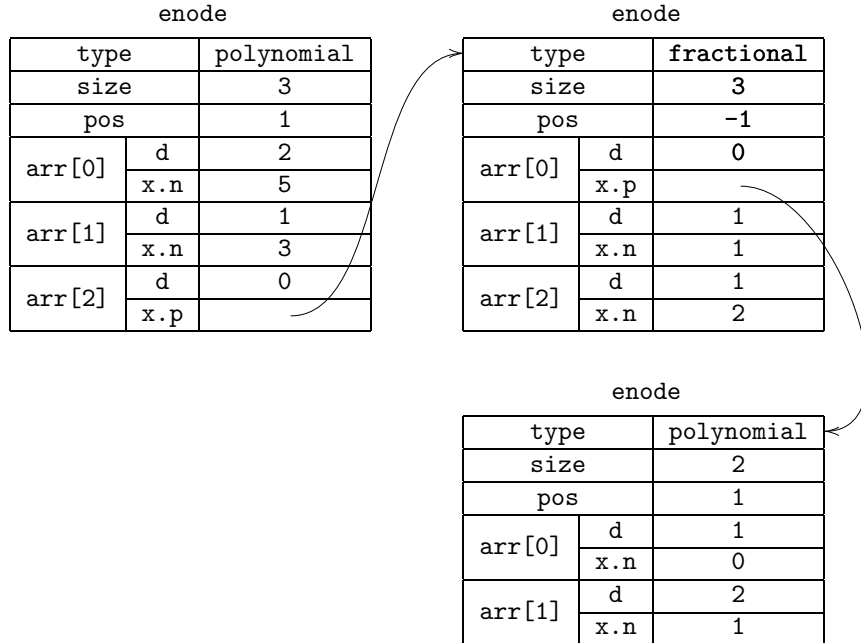


Figure 39: The quasi-polynomial $(1 + 2 \{\frac{p}{2}\}) p^2 + 3p + \frac{5}{2}$.

an explicit enumeration using the `periodic` type. By default, the current version of `barvinok` uses `periodics`, but this can be changed through the `--enable-fractional` configure option. In the latter case, the quasi-polynomial using fractional parts can also be converted to an actual step-polynomial using `evalvalue_frac2floor`, but this is not fully supported yet.

For reasons of compatibility,⁸ we shoehorned our representations for piecewise step-polynomials into the existing data structures. To this effect, we introduced four new types, `fractional`, `relation`, `partition` and `flooring`.

```
typedef enum { polynomial, periodic, evector, fractional, relation,
              partition, flooring } enode_type;
```

The field `pos` is not used in most of these additional types and is therefore set to `-1`.

The types `fractional` and `flooring` represent polynomial expressions in a fractional part or a floor respectively. The generator is stored in `arr[0]`, while the coefficients are stored in the remaining array elements. That is, an `enode` of type `fractional` represents

$$\text{arr}[1] + \text{arr}[2] \{\text{arr}[0]\} + \text{arr}[3] \{\text{arr}[0]\}^2 + \dots + \text{arr}[l-1] \{\text{arr}[0]\}^{l-2}.$$

An `enode` of type `flooring` represents

$$\text{arr}[1] + \text{arr}[2] \lfloor \text{arr}[0] \rfloor + \text{arr}[3] \lfloor \text{arr}[0] \rfloor^2 + \dots + \text{arr}[l-1] \lfloor \text{arr}[0] \rfloor^{l-2}.$$

Example 52 The internal representation of the quasi-polynomial

$$\left(1 + 2 \left\{\frac{p}{2}\right\}\right) p^2 + 3p + \frac{5}{2}$$

is shown in Figure 39.

The `relation` type is used to represent strides. In particular, if the value of `size` is 2, then the value of a `relation` is (in pseudo-code):

⁸Also known as laziness.

```
(value(arr[0]) == 0) ? value(arr[1]) : 0
```

If the size is 3, then the value is:

```
(value(arr[0]) == 0) ? value(arr[1]) : value(arr[2])
```

The type of `arr[0]` is typically `fractional`.

Finally, the `partition` type is used to represent piecewise quasi-polynomials. We prefer to encode this information inside `evaluate` rather than using `Enumerations` since we want to perform the same kinds of operations on both quasi-polynomials and piecewise quasi-polynomials. An `enode` of type `partition` may not be nested inside another `enode`. The size of the array is twice the number of “chambers”. Pointers to chambers are stored in the even slots, whereas pointer to the associated quasi-polynomials are stored in the odd slots. To be able to store pointers to chambers, the definition of `evaluate` was changed as follows.

```
typedef struct _evaluate {
    Value d;          /* denominator */
    union {
        Value n;     /* numerator (if denominator > 0) */
        struct _enode *p; /* pointer (if denominator == 0) */
        Polyhedron *D; /* domain (if denominator == -1) */
    } x;
} evaluate;
```

Note that we allow a “chamber” to be a union of polyhedra as discussed in Section 6.1. Chambers with extra variables, i.e., those of Section 7.5, are only partially supported. The field `pos` is set to the actual dimension, i.e., the number of parameters.

A.3 Operations on Quasi-polynomials

In this section we discuss some of the more important operations on `evaluate` provided by the `barvinok` library. Some of these operations are extensions of the functions from `PolyLib` with the same name.

```
void eadd(evaluate *e1, evaluate *res);
void emul (evaluate *e1, evaluate *res );
```

The functions `eadd` and `emul` takes two (pointers to) `evaluate` `e1` and `res` and computes their sum and product respectively. The result is stored in `res`, overwriting (and deallocating) the original value of `res`. It is an error if exactly one of the arguments of `eadd` is of type `partition` (unless the other argument is 0). The addition and multiplication operations are described in Sections 6.1 and 6.2 respectively.

The function `eadd` is an extension of the function `new_eadd` from Seghir (2002). Apart from supporting the additional types from Appendix A.2, the new version also additionally imposes an order on the nesting of different `enodes`. Without such an ordering, `evaluate` could be constructed representing for example

$$(0y^0 + (0x^0 + 1x^1)y^1)x^0 + (0y^0 - 1y^1)x^1,$$

which is just a funny way of saying 0.

```
void eor(evaluate *e1, evaluate *res);
```

The function `eor` implements the union operation from Section 6.3. Both arguments are assumed to correspond to indicator functions.

```
evaluate *esum(evaluate *E, int nvar);
```

The function `esum` performs the summation operation from Section 6.4. The piecewise step-polynomial represented by `E` is summated over its first `nvar` variables. Note that `E` must be zero or of type `partition`. The function returns the result in a newly allocated `evaluate`. Note also that `E` needs to have been converted from `fractionals` to `floorings` using the function `evaluate_frac2floor`.

```
void evaluate_frac2floor(evaluate *e);
```

This function also ensures that the arguments of the `floorings` are positive in the relevant chambers. It currently assumes that the argument of each `fractional` in the original `evaluate` has a minimum in the corresponding chamber.

```
double compute_evaluate(evaluate *e, Value *list_args);
Value *compute_poly(Enumeration *en, Value *list_args);
```

The functions `compute_evaluate` and `compute_poly` evaluate a (piecewise) quasi-polynomial at a certain point. The argument `list_args` points to an array of `Values` that is assumed to be long enough. The `double` return value of `compute_evaluate` is inherited from `PolyLib`.

```
void print_evaluate(FILE *DST, evaluate *e, char **pname);
```

The function `print_evaluate` dumps a human-readable representation to the stream pointed to by `DST`. The argument `pname` points to an array of character strings representing the parameter names. The array is assumed to be long enough.

```
int eequal(evaluate *e1, evaluate *e2);
```

The function `eequal` return true (1) if its two arguments are structurally identical. I.e., it does *not* check whether the two (piecewise) quasi-polynomial represent the same function.

```
void reduce_evaluate (evaluate *e);
```

The function `reduce_evaluate` performs some simplifications on `evaluate`s. Here, we only describe the simplifications that are directly related to the internal representation. Some other simplifications are explained in Section 8.2. If the highest order coefficients of a `polynomial`, `fractional` or `flooring` are zero (possibly after some other simplifications), then the size of the array is reduced. If only the constant term remains, i.e., the size is reduced to 1 for `polynomial` or to 2 for the other types, then the whole node is replaced by the constant term. Additionally, if the argument of a `fractional` has been reduced to a constant, then the whole node is replaced by its partial evaluation. A `relation` is similarly reduced if its second branch or both its branches are zero. Chambers with zero associated quasi-polynomials are discarded from a `partition`.

A.4 Generating Functions

The representation of rational generating functions uses some basic types from the NTL library for representing arbitrary precision integers (`ZZ`) as well as vectors (`vec_ZZ`) and matrices (`mat_ZZ`) of such integers. Each term in a rational generating function is represented by a `short_rat` structure.

```
struct short_rat {
    struct {
        /* rows: terms in numerator */
        mat_ZZ  coeff;
        mat_ZZ  power;
    } n;
    struct {
        /* rows: factors in denominator */
        mat_ZZ  power;
    } d;
};
```

short_rat		
n.coeff	3	2
	2	1
n.power	2	3
	5	-7
d.power	1	-3
	0	2

Figure 40: Internal representation of $(\frac{3}{2}x_0^2x_1^3 + 2x_0^5x_1^{-7})/((1-x_0x_1^{-3})(1-x_1^2))$.

The fields `n` and `d` represent the numerator and the denominator respectively. Note that in our implementation we combine terms with the same denominator. In the numerator, each row of `coeff` and `power` represents a single such term. The matrix `coeff` has two columns, one for the numerator and one for the denominator of the rational coefficient α_i of each term. The columns of `power` correspond to the powers of the variables. In the denominator, each row of `power` corresponds to the power \mathbf{b}_{ij} of a factor in the denominator.

Example 53 Figure 40 shows the internal representation of

$$\frac{\frac{3}{2}x_0^2x_1^3 + 2x_0^5x_1^{-7}}{(1-x_0x_1^{-3})(1-x_1^2)}.$$

The whole rational generating function is represented by a `gen_fun` structure.

```
struct gen_fun {
  std::vector< short_rat * > term;
  Polyhedron *context;

  void add(ZZ& cn, ZZ& cd, vec_ZZ& num, mat_ZZ& den);
  void print(unsigned int nparam, char **param_name);
  operator evaluate *();

  gen_fun(Polyhedron *C = NULL) : context(C) {}
  ~gen_fun();
};
```

The method `gen_fun::add` adds a new term to the rational generating function. It makes all powers in the denominator lexico-positive, orders them in lexicographical order and inserts the new term in `term` according to the lexicographical order of the combined powers in the denominator. The method `gen_fun::operator evaluate *` performs the conversion from rational generating function to piecewise step-polynomial explained in Section 6.5. The `Polyhedron context` is the superset of all points where the enumerator is non-zero used during this conversion, i.e., it is the set Q from (32). If `context` is `NULL` the maximal allowed context is assumed, i.e., the maximal region with lexico-positive rays.

A.5 Counting Functions

We provide essentially three different counting functions: one for non-parametric polytopes, one for parametric polytopes and one for parametric sets with existential variables.

```
void barvinok_count(Polyhedron *P, Value* result, unsigned NbMaxCons);
```

The function `barvinok_count` enumerates the non-parametric polytope `P` and returns the result in the `Value` pointed to by `result`, which needs to have been allocated and initialized. The argument `NbMaxCons` is passed to various `PolyLib` functions and defines the maximum size of a table used in the double description computation in the `PolyLib` function `Chernikova`. In earlier versions of `PolyLib`, this parameter had to be conservatively set to a high number to ensure successful operation, resulting in significant memory overhead. Our change to allow this table to grow dynamically is available in recent versions of `PolyLib`. In these versions, the value no longer indicates the maximal table size, but rather the size of the initial allocation. This value may be set to 0.

The function `barvinok_enumerate` for enumerating parametric polytopes was meant to be a drop-in replacement of the corresponding `Polyhedron_Enumerate` function in `PolyLib`. Unfortunately, the latter has been changed to accept an extra argument in recent versions of `PolyLib` as shown below.

```
Enumeration* barvinok_enumerate(Polyhedron *P, Polyhedron* C, unsigned MaxRays);
extern Enumeration *Polyhedron_Enumerate ( Polyhedron *P, Polyhedron *C,
                                         unsigned MAXRAYS, char **pname );
```

The argument `MaxRays` has the same meaning as the argument `NbMaxCons` above. The argument `P` refers to the $(d+n)$ -dimensional polyhedron defining the parametric polytope. The argument `C` is an n -dimensional polyhedron containing extra constraints on the parameter space. Its primary use is to indicate how many of the dimensions in `P` refer to parameters as any constraint in `C` could equally well have been added to `P` itself. Note that the dimensions referring to the parameters should appear *last*. The result is a newly allocated `Enumeration`. As an alternative we also provide a function (`barvinok_enumerate_ev`) that returns an `evaluate`.

```
evaluate* barvinok_enumerate_ev(Polyhedron *P, Polyhedron* C, unsigned MaxRays);
```

For enumerating parametric sets with existentially quantified variables, we provide two functions: `barvinok_enumerate_e` and `barvinok_enumerate_pip`.

```
evaluate* barvinok_enumerate_e(Polyhedron *P,
                              unsigned exist, unsigned nparam, unsigned MaxRays);
evaluate *barvinok_enumerate_pip(Polyhedron *P,
                                unsigned exist, unsigned nparam, unsigned MaxRays);
```

The first function tries the simplification rules from Section 7.2 before resorting to the method based on PIP from Section 7.3. The second function immediately applies the technique from Section 7.3. The argument `exist` refers to the number of existential variables, whereas the argument `nparam` refers to the number of parameters. The order of the dimensions in `P` is: counted variables first, then existential variables and finally the parameters.

The function `barvinok_series` enumerates parametric polytopes in the form of a rational generating function. The polyhedron `P` is assumed to have only lexico-positive rays.

```
gen_fun * barvinok_series(Polyhedron *P, Polyhedron* C, unsigned MaxRays);
```

A.6 Auxiliary Functions

In this section we briefly mention some auxiliary functions available in the `barvinok` library.

```
void Polyhedron_Polarize(Polyhedron *P);
```

The function `Polyhedron_Polarize` polarizes its argument and is explained in Section 5.2.

```
Matrix * unimodular_complete(Vector *row);
```

The function `unimodular_complete` extends `row` to a unimodular matrix using the algorithm of Bik (1996).

```
int DomainIncludes(Polyhedron *Pol1, Polyhedron *Pol2);
```

The function `DomainIncludes` extends the PolyLib function `PolyhedronIncludes` to unions of polyhedra. It checks whether its first argument is a superset of its second argument.

```
Polyhedron *DomainConstraintSimplify(Polyhedron *P, unsigned MaxRays);
```

The function `DomainConstraintSimplify` returns a newly allocated `Polyhedron` that contains the same integer points as its first argument but possibly has simpler constraints. Each constraint $g\langle \mathbf{a}, \mathbf{x} \rangle \geq c$ is replaced by $\langle \mathbf{a}, \mathbf{x} \rangle \geq \left\lceil \frac{c}{g} \right\rceil$, where g is the gcd of the coefficients in the original constraint. The `Polyhedron` pointed to by `P` is destroyed.

```
Polyhedron* Polyhedron_Project(Polyhedron *P, int dim);
```

The function `Polyhedron_Project` projects `P` onto its last `dim` dimensions.

B Usage of the barvinok library

In this section, we describe some of the application programs provided by the `barvinok` library, available from <http://freshmeat.net/projects/barvinok/>. For compilation instructions we refer to the `README` file included in the distribution.

The program `barvinok_count` enumerates a non-parametric polytope. It takes one polytope in PolyLib notation as input and prints the number of integer points in the polytope and the time taken by both “manual counting” and Barvinok’s method. The PolyLib notation corresponds to the internal representation of `Polyhedrons` as explained in Appendix A.1. The first line of the input contains the number of rows and the number of columns in the `Constraint` matrix. The rest of the input is composed of the elements of the matrix. Recall that the number of columns is two more than the number of variables, where the extra first columns is one or zero depending on whether the constraint is an inequality (≥ 0) or an equality ($= 0$). The next columns contain the coefficients of the variables and the final column contains the constant in the constraint. E.g., the set $S = \{s \mid s \geq 0 \wedge 2s \leq 13\}$ from Example 20 on page 26 corresponds to the following input and output.

```
> cat S
2 3

1 1 0
1 -2 13
> ./barvinok_count < S
POLYHEDRON Dimension:1
          Constraints:2 Equations:0 Rays:2 Lines:0
Constraints 2 3
Inequality: [  1  0 ]
Inequality: [ -2 13 ]
Rays 2 3
Vertex: [  0 ]/1
Vertex: [ 13 ]/2
manual:  7
User: 0.01; Sys: 0
Barvinok:  7
User: 0; Sys: 0
```

The program `cdd2polylib.pl` can be used to convert a polytope in `cdd` (Fukuda 1993) notation to PolyLib notation.

The program `barvinok_enumerate` enumerates a parametric polytope. It takes two polytopes in PolyLib notation as input, optionally followed by a list of parameter names. The two polytopes refer to arguments P and C of the corresponding function. (See Appendix A.5.) The following example was taken by Loechner (1999) from Loechner (1997, Chapter II.2).

```

> cat loechner
# Dimension of the matrix:
7 7
# Constraints:
# i j k P Q cte
1 1 0 0 0 0 0 # 0 <= i
1 -1 0 0 1 0 0 # i <= P
1 0 1 0 0 0 0 # 0 <= j
1 1 -1 0 0 0 0 # j <= i
1 0 0 1 0 0 0 # 0 <= k
1 1 -1 -1 0 0 0 # k <= i-j
0 1 1 1 0 -1 0 # Q = i + j + k

# 2 parameters, no constraints.
0 4
> ./barvinok_enumerate < loechner
POLYHEDRON Dimension:5
          Constraints:6 Equations:1 Rays:5 Lines:0
Constraints 6 7
Equality:  [  1  1  1  0 -1  0 ]
Inequality: [  0  1  1  1 -1  0 ]
Inequality: [  0  1  0  0  0  0 ]
Inequality: [  0  0  1  0  0  0 ]
Inequality: [  0 -2 -2  0  1  0 ]
Inequality: [  0  0  0  0  0  1 ]
Rays 5 7
Ray:  [  1  0  1  1  2 ]
Ray:  [  1  1  0  1  2 ]
Vertex: [  0  0  0  0  0 ]/1
Ray:  [  0  0  0  1  0 ]
Ray:  [  1  0  0  1  1 ]
POLYHEDRON Dimension:2
          Constraints:1 Equations:0 Rays:3 Lines:2
Constraints 1 4
Inequality: [  0  0  1 ]
Rays 3 4
Line:  [  1  0 ]
Line:  [  0  1 ]
Vertex: [  0  0 ]/1
      - P + Q >= 0
      2P - Q >= 0
      1 >= 0

( -1/2 * P^2 + ( 1 * Q + 1/2 )
  * P + ( -3/8 * Q^2 + ( -1/2 * {( 1/2 * Q + 0 )
} + 1/4 )
  * Q + ( -5/4 * {( 1/2 * Q + 0 )
} + 1 )
)

```

```

)
    Q >= 0
    P - Q - 1 >= 0
    1 >= 0

( 1/8 * Q^2 + ( -1/2 * {( 1/2 * Q + 0 )
} + 3/4 )
* Q + ( -5/4 * {( 1/2 * Q + 0 )
} + 1 )
)

```

The output corresponds to

$$\begin{cases} -\frac{1}{2}P^2 + PQ + \frac{1}{2}P - \frac{3}{8}Q^2 + (\frac{1}{4} - \frac{1}{2}\{\frac{1}{2}Q\})Q + 1 - \frac{5}{4}\{\frac{1}{2}Q\} & \text{if } P \leq Q \leq 2P \\ \frac{1}{8}Q^2 + (\frac{3}{4} - \frac{1}{2}\{\frac{1}{2}Q\}) - \frac{5}{4}\{\frac{1}{2}Q\} & \text{if } 0 \leq Q \leq P - 1. \end{cases}$$

The program `barvinok_enumerate_e` enumerates a parametric projected set. It takes a single polytopes in PolyLib notation as input, followed by two lines indicating the number or existential variables and the number of parameters and optionally followed by a list of parameter names. The syntax for the line indicating the number of existential variables is the letter E followed by a space and the actual number. For indicating the number of parameters, the letter P is used. The following example corresponds to Example 18 on page 22.

```

> cat projected
5 6
#   k   i   j   p   cst
1   0   1   0   0   -1
1   0  -1   0   0    8
1   0   0   1   0   -1
1   0   0  -1   1    0
0  -1   6   9   0   -7

E 2
P 1
> ./barvinok_enumerate_e <projected
POLYHEDRON Dimension:4
          Constraints:5  Equations:1  Rays:4  Lines:0
Constraints 5 6
Equality:  [  1  -6  -9   0   7 ]
Inequality: [  0   1   0   0  -1 ]
Inequality: [  0  -1   0   0   8 ]
Inequality: [  0   0   1   0  -1 ]
Inequality: [  0   0  -1   1   0 ]
Rays 4 6
Vertex: [  50   8   1   1 ]/1
Ray:    [   0   0   0   1 ]
Ray:    [   9   0   1   1 ]
Vertex: [   8   1   1   1 ]/1
exist: 2, nparam: 1
      P  -3 >= 0
      1  >= 0

( 3 * P + 10 )
      P  -1 >= 0
      - P + 2 >= 0

```

```
( 8 * P + 0 )
```

The program `barvinok_series` enumerates a parametric polytope in the form of a rational generating function. The input of this program is the same as that of `barvinok_enumerate`, except that the input polyhedron is assumed to be full-dimensional. The following is an example of Petr Lisoněk.

```
> cat petr
4 6
1 -1 -1 -1 1 0
1 1 -1 0 0 0
1 0 1 -1 0 0
1 0 0 1 0 -1

0 3
n
> ./barvinok_series < petr
POLYHEDRON Dimension:4
      Constraints:5  Equations:0  Rays:5  Lines:0
Constraints 5 6
Inequality: [ -1 -1 -1 1 0 ]
Inequality: [ 1 -1 0 0 0 ]
Inequality: [ 0 1 -1 0 0 ]
Inequality: [ 0 0 1 0 -1 ]
Inequality: [ 0 0 0 0 1 ]
Rays 5 6
Ray: [ 1 1 1 3 ]
Ray: [ 1 1 0 2 ]
Ray: [ 1 0 0 1 ]
Ray: [ 0 0 0 1 ]
Vertex: [ 1 1 1 3 ]/1
POLYHEDRON Dimension:1
      Constraints:1  Equations:0  Rays:2  Lines:1
Constraints 1 3
Inequality: [ 0 1 ]
Rays 2 3
Line: [ 1 ]
Vertex: [ 0 ]/1
(n^3)/((1-n) * (1-n) * (1-n^2) * (1-n^3))
```

References

- Alekseevskaya, T. V., I. M. Gel'fand, and A. V. Zelevinskii (1987). Distribution of real hyperplanes and the partition function connected with it. *Doklady Akademii Nauk SSSR* 297(6), 1289–1293. [13]
- Anantharaman, S. and S. Pande (1998). Compiler optimizations for real time execution of loops on limited memory embedded systems. In *The 19th IEEE Systems Symposium (RTSS98)*. [80]
- Aurenhammer, F. and R. Klein (2000). Voronoi diagrams. In J. Sack and G. Urrutia (Eds.), *Handbook of Computational Geometry, Chapter V*, pp. 201–290. Elsevier Science Publishing. [SFB Report F003-092, TU Graz, Austria, 1996]. [32]

- Balasa, F., F. Catthoor, and H. De Man (1995, June). Background memory area estimation for multidimensional signal processing systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 3(2), 157–172. [80]
- Barvinok, A. (1993, November). A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. In *34th Annual Symposium on Foundations of Computer Science*, pp. 566–572. IEEE. [4]
- Barvinok, A. (1994). Computing the Ehrhart polynomial of a convex lattice polytope. *Discrete Comput. Geom.* 12, 35–48. [29, 36, 43]
- Barvinok, A. (2002). *A Course in Convexity*, Volume 54 of *Graduate Studies in Mathematics*. Providence, RI: American Mathematical Society. [29]
- Barvinok, A. and J. Pommersheim (1999). An algorithmic theory of lattice points in polyhedra. *New Perspectives in Algebraic Combinatorics* 38, 91–147. [4, 7, 25, 27, 29, 31, 39, 43, 47, 48, 54, 55, 64]
- Barvinok, A. and K. Woods (2003, April). Short rational generating functions for lattice point problems. *J. Amer. Math. Soc.* 16, 957–979. [4, 24, 43, 60, 63, 69, 70]
- Beck, M. (2004). The partial-fractions method for counting solutions to integral linear systems. *Discrete Comp. Geom.* 32, 437–446. (special issue in honor of Louis Billera). [61, 79]
- Beck, M., J. A. De Loera, M. Develin, J. Pfeifle, and R. P. Stanley (2004, February). Coefficients and Roots of Ehrhart Polynomials. AIM 2004-1. [9]
- Beck, M. and S. Robins (2006). *Computing the Continuous Discretely. Integer-point Enumeration in Polyhedra*. Springer Undergraduate Texts in Mathematics. Springer. to appear. [9, 10]
- Bednara, M., F. Hannig, and J. Teich (2002). Generation of distributed loop control. In *In Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation (SAMOS)*, Volume 2268 of *Lecture Notes in Computer Science*, pp. 154–170. [81]
- Beyls, K. (2004). *Software Methods to Improve Data Locality and Cache Behavior*. Ph. D. thesis, Ghent University. [76, 77, 78, 80]
- Bik, A. J. C. (1996). *Compiler Support for Sparse Matrix Computations*. Ph. D. thesis, University of Leiden, The Netherlands. [17, 65, 88]
- Blakley, G. R. (1964). Combinatorial remarks on partitions of a multipartite number. *Duke Math. J.* 31(2), 335–340. [13]
- Boigelot, B. and L. Latour (2004, February). Counting the solutions of Presburger equations without enumerating them. *Theoretical Computer Science* 313(1), 17–29. [4, 79]
- Bollella, G. and J. Gosling (2000). The real-time specification for Java. *Computer* 33(6), 47–54. [81]
- Boulet, P. and X. Redon (1998a). Communication pre-evaluation in HPF. In *EUROPAR'98*, Volume 1470 of *LNCS*, pp. 263–272. Springer Verlag. [63, 68, 72, 79, 80]
- Boulet, P. and X. Redon (1998b). Communication pre-evaluation in HPF. Technical report, Université des Sciences et Technologies de Lille. AS-182. [79, 80]
- Braberman, V., D. Garbervetsky, and S. Yovine (2003, October). On synthesizing parametric specifications of dynamic memory utilization. Technical Report TR-2004-03, VERIMAG. [81]
- Brion, M. (1988). Points entiers dans les polyèdres convexes. *Annales Scientifiques de l'École Normale Supérieure. Quatrième Série* 21(4), 653–663. [29, 31]

- Buck, R. (1943). Partition of space. *American Mathematical Monthly* 50(9), 541–544. [17]
- Chatterjee, S., E. Parker, P. J. Hanlon, and A. R. Lebeck (2001). Exact analysis of the cache behavior of nested loops. In *Proceedings of the ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation*, pp. 286–297. ACM Press. [80]
- Clauss, P. (1996). Counting solutions to linear and nonlinear constraints through Ehrhart polynomials: Applications to analyze and transform scientific programs. In *International Conference on Supercomputing*, pp. 278–285. [63, 64, 79, 80]
- Clauss, P. (1997). Handling memory cache policy with integer points counting. In *European Conference on Parallel Processing*, pp. 285–293. [22]
- Clauss, P. and V. Loechner (1998, July). Parametric analysis of polyhedral iteration spaces. *Journal of VLSI Signal Processing* 19(2), 179–194. [3, 4, 9, 15, 16, 21, 25, 63, 68, 75, 76, 78, 79, 80, 82]
- Cohen, H. (1993). *A Course in Computational Algebraic Number Theory*. Springer-Verlag New York, Inc. [37]
- Dahmen, W. and C. A. Micchelli (1988). The number of solutions to linear diophantine equations and multivariate splines. *Trans. Amer. Math. Soc.* 308, 509–532. [12]
- De Loera, J. A. (1995, May). *Triangulations of Polytopes and Computational Algebra*. Ph. D. thesis, Cornell University. [32]
- De Loera, J. A., D. Haws, R. Hemmecke, P. Huggins, B. Sturmfels, and R. Yoshida (2003a, July). Short rational functions for toric algebra and applications. <http://arxiv.org/abs/math.CO/0307350>. [4, 79]
- De Loera, J. A., D. Haws, R. Hemmecke, P. Huggins, J. Tauzer, and R. Yoshida (2003b, November). A user’s guide for latte v1.1. software package LatTE is available at <http://www.math.ucdavis.edu/~latte/>. [32, 78]
- De Loera, J. A., R. Hemmecke, J. Tauzer, and R. Yoshida (2004). Effective lattice point counting in rational convex polytopes. *The Journal of Symbolic Computation* 38(4), 1273–1302. [4, 25, 32, 36, 37, 38, 43, 44, 45, 79]
- Derrien, S., A. Turjan, C. Zissulescu, B. Kienhuis, and E. Deprettere (2003). Deriving efficient control in Kahn process network. In *Proc. of the "Int. Workshop on Systems, Architectures, Modeling, and Simulation, (SAMOS 2003)"*. [81]
- Diaconis, P. and A. Gangolli (1995). Rectangular arrays with fixed margins. In *Discrete probability and algorithms (Minneapolis, MN, 1993)*, Volume 72 of *IMA Vol. Math. Appl.*, pp. 15–41. New York: Springer. [12]
- Diaz, R. and S. Robins (1996). The ehrhart polynomial of a lattice n-simplex. [9]
- Dyer, M. and R. Kannan (1997). On barvinok’s algorithm for counting lattice points in fixed dimension. *Mathematics of Operations Research* 22(3), 545–549. [37]
- Edelsbrunner, H. (1987). *Algorithms in Combinatorial Geometry*. Springer-Verlag New York, Inc. [17]
- Ehrhart, E. (1962). Sur les polyèdres rationnels homothétiques à n dimensions. *C. R. Acad. Sci. Paris* 254, 616–618. [9]
- Ehrhart, E. (1977). *Polynômes arithmétiques et Méthode des Polyèdres en Combinatoire*, Volume 35 of *International Series of Numerical Mathematics*. Basel/Stuttgart: Birkhauser Verlag. [9, 15]

- Euler, L. (1770). De partitione numerorum in partes tam numero quam specie datas. *Novi Commentarii academiae scientiarum Petropolitanae* 14, 168–187. [12]
- Feautrier, P. (1988). Parametric integer programming. *Operationnelle/Operations Research* 22(3), 243–268. [63, 68, 79]
- Ferrante, J., V. Sarkar, and W. Thrash (1991, August). On estimating and enhancing cache effectiveness. In U. Banerjee, D. Gelernter, A. Nicolau, and D. Padua (Eds.), *Proceedings of the Fourth International Workshop on Languages and Compilers for Parallel Computing*, Volume 589 of *Lecture Notes in Computer Science*, pp. 328–343. Springer-Verlag. [80]
- Fortune, S. (1992). Voronoi diagrams and delaunay triangulations. In D.-Z. Du and F. Hwang (Eds.), *Computing in Euclidean Geometry*, World Scientific, Lecture Notes Series on Computing – Vol. 1, pp. 193–234. World Scientific. [7]
- Franke, B. and M. O’Boyle (2003, May). Array recovery and high-level transformations for DSP applications. *ACM Transactions on Embedded Computing Systems* 2(2), 132–162. [58, 78, 81]
- Free Software Foundation, Inc.. GMP. Available from <ftp://ftp.gnu.org/gnu/gmp>. []
- Fukuda, K. (1993). cdd.c: C-implementation of the double description method for computing all vertices and extremal rays of a convex polyhedron given by a system of linear inequalities. Technical report, Department of Mathematics, Swiss Federal Institute of Technology, Lausanne, Switzerland. program available from <http://www.ifor.math.ethz.ch/fukuda/fukuda.html>. [89]
- Ghosh, S., M. Martonosi, and S. Malik (1999). Cache miss equations: a compiler framework for analyzing and tuning memory behavior. *ACM Transactions on Programming Languages and Systems* 21(4), 703–746. [78, 80]
- Graham, R. L., D. E. Knuth, and O. Patashnik (1989). *Concrete Mathematics*. Addison-Wesley. [44]
- Grötschel, M., L. Lovász, and A. Schrijver (1988). *Geometric Algorithms and Combinatorial Optimization*. Berlin: Springer. [37]
- Grünbaum, B. (1967). *Convex Polytopes*. John Wiley & Sons. [31]
- Halbwachs, N., D. Merchat, and C. Parent-Vigouroux (2003, June). Cartesian factoring of polyhedra in linear relation analysis. In *Static Analysis Symposium, SAS’03*, San Diego. LNCS 2694, Springer Verlag. [71]
- Hardy, G. H. and E. M. Wright (1979). *An Introduction to the Theory of Numbers* (Fifth ed.). Oxford University Press. [12]
- Heckman, G. J. (1982). Projections of orbits and asymptotic behavior of multiplicities for compact connected Lie groups. *Inventiones Mathematicae* 67(2), 333–356. [12]
- Heine, F. and A. Slowik (2000). Volume driven data distribution for NUMA-machines. In *Proceedings from the 6th International Euro-Par Conference on Parallel Processing*, pp. 415–424. [80]
- Henrici, P. (1974). *Applied and Computational Complex Analysis*. Pure and applied mathematics. New York: Wiley-Interscience [John Wiley & Sons]. Volume 1: Power series—integration—conformal mapping—location of zeros, Pure and Applied Mathematics. [44, 79]
- Kelly, W., V. Maslov, W. Pugh, E. Rosser, T. Shpeisman, and D. Wonnacott (1996, November). The Omega calculator and library. Technical report, University of Maryland. [77]
- Lee, C. W. (1991). Regular triangulations of convex polytopes. *Applied Geometry and Discrete Mathematics — The Victor Klee Festschrift 4*, 443–456. [32]

- Lee, C. W. (1997). Subdivisions and triangulations of polytopes. In *Handbook of discrete and computational geometry*, pp. 271–290. CRC Press, Inc. [32]
- Lenstra, A. K., H. W. Lenstra, and L. Lovász (1982). Factoring polynomials with rational coefficients. *Mathematische Annalen* 261(4), 515–534. [37]
- Lisper, B. (2003, July). Fully automatic, parametric worst-case execution time analysis. In J. Gustafsson (Ed.), *Proc. Third International Workshop on Worst-Case Execution Time (WCET) Analysis*, Porto, pp. 77–80. [81]
- Loechner, V. (1997). *Contribution à l'étude des polyèdres paramétrés et applications en parallélisation automatique*. Ph. D. thesis, University Louis Pasteur, Strasbourg. [90]
- Loechner, V. (1999, March). Polylib: A library for manipulating parameterized polyhedra. Technical report, ICPS, Université Louis Pasteur de Strasbourg, France. [76, 78, 82, 83, 90]
- Loechner, V. and D. K. Wilde (1997, December). Parameterized polyhedra and their vertices. *International Journal of Parallel Programming* 25(6), 525–549. [19, 21, 82]
- Loechner, V., B. Meister, and P. Clauss (2002). Precise data locality optimization of nested loops. *J. Supercomput.* 21(1), 37–76. [78, 81]
- Manocha, D. (1993). Multipolynomial resultant algorithms. *Journal of Symbolic Computation* 15(2), 99–122. [76]
- Matoušek, J. (2002). *Lectures on Discrete Geometry*, Volume 212 of *Graduate Texts in Mathematics*. New York: Springer-Verlag. [17]
- McAllister, T. B. and K. Woods (2004, March). The minimum period of the ehrhart quasipolynomial of a rational polytope. [9]
- Meister, B. (2004, July). Projecting periodic polyhedra for loop nest analysis. In M. Gerndt and E. Kereku (Eds.), *Proceedings of the 11th Workshop on Compilers for Parallel Computers (CPC 04)*, pp. 13–24. [79]
- Miller, E. and B. Sturmfels (2004). *Combinatorial Commutative Algebra*. Springer Graduate Texts in Math. Springer. [9]
- Nootaert, B., K. Beyls, and E. D'Hollander (2005, January). On the calculation of Erhart polynomials in degenerate domains. Technical Report R105.001, Ghent University. [76]
- Papadimitriou, C. (1994). *Computational Complexity*. Reading, MA: Addison-Wesley Publishing Company. [8]
- Parker, E. and S. Chatterjee (2004, April). An automata-theoretic algorithm for counting solutions to Presburger formulas. In *Compiler Construction 2004*, Volume 2985 of *Lecture Notes in Computer Science*, pp. 104–119. [4, 79]
- Pugh, W. (1991). The Omega test: a fast and practical integer programming algorithm for dependence analysis. In *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, pp. 4–13. ACM Press. [66]
- Pugh, W. (1992). A practical algorithm for exact array dependence analysis. *Communications of the ACM* 35(8), 102–114. [66, 75]
- Pugh, W. (1994). Counting solutions to Presburger formulas: How and why. In *SIGPLAN Conference on Programming Language Design and Implementation (PLDI'94)*, pp. 121–134. [4, 7, 75, 81]
- Rambau, J. (1996, October). *Polyhedral Subdivisions and Projections of Polytopes*. Ph. D. thesis, Fachbereich Mathematik, TU-Berlin, Shaker-Verlag, Aachen. [15]

- Rijkema, E., E. Deprettere, and B. Kienhuis (1999). Compilation from matlab to process networks. In *Second International Workshop on Compiler and Architecture Support for Embedded Systems (CASES'99)*. [81]
- Schrijver, A. (1986). *Theory of Linear and Integer Programming*. John Wiley & Sons. [8, 31, 36, 37]
- Seghir, R. (2002, June). Dénombrément des point entiers de l'union et de l'image des polyédres paramétrés. Master's thesis, ICPS, Université Louis Pasteur de Strasbourg, France. [64, 79, 85]
- Seghir, R. (2004). Personal communication. [19]
- Shoup, V. (2004). NTL. Available from <http://www.shoup.net/ntl/>. [38]
- Srivastava, H. M. and H. L. Manocha (1984). *A Treatise on Generating Functions*. Ellis Horwood Series: Mathematics and its Applications. Chichester: Ellis Horwood Ltd. [8]
- Stanley, R. P. (1986). *Enumerative Combinatorics*, Volume 1. Cambridge University Press. [9]
- Stanley, R. P. (1996). *Combinatorics and Commutative Algebra, Second edition*, Volume 41 of *Progress in Mathematics*. Birkhäuser, Boston. [12]
- Sturmfels, B. (1995). On vector partition functions. *J. Comb. Theory Ser. A* 72(2), 302–309. [12, 13, 14, 15]
- Turjan, A., B. Kienhuis, and E. Deprettere (2002, July). A compile time based approach for solving out-of-order communication in Kahn process networks. In *IEEE 13th International Conference on Application-specific Systems, Architectures and Processors (ASAP'2002)*. [76, 78, 81]
- van Engelen, R. A. and K. A. Gallivan (2001). An efficient algorithm for pointer-to-array access conversion for compiling and optimizing dsp applications. In *Innovative Archts. for Future Gen. High-Perf. Processors and Systems*, pp. 80–89. IEEE. [58]
- Verdoolaege, S., K. Beyls, M. Bruynooghe, and F. Catthoor (2004a, October). Experiences with enumeration of integer projections of parametric polytopes. Report CW 395, K.U.Leuven, Department of Computer Science. [63, 79]
- Verdoolaege, S., R. Seghir, K. Beyls, V. Loechner, and M. Bruynooghe (2004b, September). Analytical computation of Ehrhart polynomials: Enabling more compiler analyses and optimizations. In *Proceedings of International Conference on Compilers, Architectures, and Synthesis for Embedded Systems, Washington D.C.*, pp. 248–258. [24, 48, 79]
- Wilde, D. K. (1993). A library for doing polyhedral operations. Technical Report 785, IRISA, Rennes, France. <http://www.irisa.fr/EXTERNE/bibli/pi/pi785.html>. [31, 82]
- Woods, K. (2004, December). Computing the period of an Ehrhart quasi-polynomial. [9, 24, 72]
- Yoshida, R. (2004a). *Barvinok's Rational Functions: Algorithms and Applications to Optimization, Statistics, and Algebra*. Ph. D. thesis, UC-Davis. [10, 12, 25]
- Yoshida, R. (2004b). Personal communication. [45]
- Zhao, Y. and S. Malik (2000, October). Exact memory size estimation for array computations. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 8(5), 517–521. [80]

List of Symbols

c_S The enumerator of the integer points in the set S , see equation (2), page 7

- K A (typically simplicial) cone or the matrix with the generators of the cone as columns, page 36
- π_d The projection onto the first d dimensions

List of Acronyms

- CME..... Cache Miss Equations
- GCD greatest common divisor
- LCM least common multiple
- LLL..... Lenstra, Lenstra and Lovasz' basis reduction algorithm
- PIP Parametric Integer Programming
- TLB..... Translation Lookaside Buffer
- WCET..... Worst-Case Execution Time

Index

- disable-incremental, 45
- enable-fractional, 84

- active vertex, **19**
- add
 - gen_fun::, *see* gen_fun::add
- affine hull, **5**
- Alekseevskaya, T. V., 13, *92*
- Anantharaman, S., 80, *92*
- approximation theory, 12
- arr, 83
- array recovery, 58, 81
- Aurenhammer, F., 32, *92*

- Balasa, F., 80, *93*
- barvinok, 2, 4, 45, 82, 84, 85, 88, 89
 - availability, 89
- Barvinok, A., 4, 7, 24, 25, 27, 29, 31, 36, 39, 43, 47, 48, 54, 55, 60, 63, 64, 69, 70, *93*
- barvinok_count, 88, 89
- barvinok_enumerate, 88, 90, *92*
- barvinok_enumerate_e, 88, 91
- barvinok_enumerate_ev, 88
- barvinok_enumerate_pip, 88
- barvinok_series, 88, *92*
- Beck, M., 9, 10, 61, 79, *93*
- Bednara, M., 81, *93*
- Beyls, K., 76–78, 80, *93, 96, 97*
- Bik, A. J. C., 17, 65, 88, *93*
- Blakley, G. R., 13, *93*
- Boigelot, B., 4, 79, *93*
- Bollella, G., 81, *93*
- Boulet, P., 63, 68, 72, 79, 80, *93*
- Braberman, V., 81, *93*
- Brion’s polarization trick, 31
- Brion’s Theorem, 29
- Brion, M., 29, 31, *93*
- Bruynooghe, M., *97*
- Buck, R., 17, *94*

- Catthoor, F., *93, 97*
- cdd, 89
- cdd2polylib.pl, 89
- cell, **6**
- chamber, **13**
- chamber complex, 53, 59
 - of a parametric polytope, **16**
 - of a polytope projection, **15**
 - of a vector partition function, **13**
- Chatterjee, S., 4, 79, 80, *94, 96*

- Chernikova, 88
- Clauss, P., 3, 4, 9, 15, 16, 21, 22, 25, 63, 64, 68, 75, 76, 78–80, 82, *94, 96*
- coeff, 87
- Cohen, H., 37, *94*
- common refinement, **6**
- commutative algebra, 12
- complex
 - chamber, *see* chamber complex
 - polyhedral, *see* polyhedral complex
- compute_evaluate, 86
- compute_poly, 86
- cone, **6**
 - polar, *see* polar cone
 - polyhedral, *see* polyhedral cone
 - simplicial, *see* simplicial cone
 - supporting, *see* supporting cone
 - unimodular, *see* unimodular cone
- cone::short_vector, 38
- constituent, 9
- Constraint, 82, 89
- context, 87
- corrector polynomial, 13
- curve
 - moment, *see* moment curve

- d, 83, 87
- D’Hollander, E., *96*
- Dahmen, W., 12, *94*
- dark shadow, 75
- De Loera, J. A., 4, 25, 32, 36–38, 43–45, 78, 79, *93, 94*
- De Man, H., *93*
- decompose
 - decomposer::, *see* decomposer::decompose
- decomposer::decompose, 38
- decomposer::handle, 38
- degenerate domain, 76
- degree
 - of a piecewise step-polynomial, **24**
 - of a step-polynomial, **24**
- Delaunay triangulation, 32, **33**
- Deprettere, E., *94, 97*
- Derrien, S., 81, *94*
- Develin, M., *93*
- Diaconis, P., 12, *94*
- Diaz, R., 9, *94*
- difference
 - set, *see* set difference
- Dimension, 82
- dimension, **5**

domain
 degenerate, *see* degenerate domain
 validity, *see* validity domain
 DomainConstraintSimplify, 89
 DomainDifference, 61
 DomainIncludes, 89
 Dyer, M., 37, 94

 eadd, 85
 eadd_partitions, 53
 Edelsbrunner, H., 17, 94
 eequal, 86
 Ehrhart polynomial, 9
 Ehrhart quasi-polynomial, 9
 Ehrhart series, 9, 79
 Ehrhart, E., 9, 15, 94
 emask, 57
 emul, 85
 emul_partitions, 57
 enode, 82–85
 Enumeration, 83, 85, 88
 enumerator, 7
 envelope
 lower, *see* lower envelope
 eor, 85
 esum, 59, 86
 Euler, L., 12, 95
 evalue, 82, 83, 85, 86, 88
 evalue_combine, 74
 evalue_frac2ffloor, 84, 86
 evalue_range_reduction, 74

 face, 5
 lower, *see* lower face
 facet, 5
 thick, *see* thick facet
 Feautrier, P., 63, 68, 79, 95
 Ferrante, J., 80, 95
 fiber, 7
 flooring, 84, 86
 Fortune, S., 7, 95
 fractional, 84–86
 fractional part, 72
 Franke, B., 58, 78, 81, 95
 Free Software Foundation, Inc., 95
 Fukuda, K., 89, 95
 function
 generating, *see* generating function
 indicator, *see* indicator function
 partition, *see* partition function
 fundamental parallelepiped, 28, 36

 g, 4
 g1, 61

 Gallivan, K. A., 58, 97
 Gangolli, A., 12, 94
 Garbervetsky, D., 93
 Gel'fand, I. M., 92
 gen_fun, 87
 gen_fun::add, 62, 87
 gen_fun::operator evaluate *, 62, 87
 generating function, 8
 multiple, *see* multiple generating function
 of a sequence, 8
 of an integer set, 25
 rational, *see* rational generating function
 generator, 5
 geometric series, 8, 26
 Ghosh, S., 78, 80, 95
 GMP, 78, 82
 Gosling, J., 81, 93
 Grünbaum, B., 31, 95
 Graham, R. L., 44, 95
 Grötschel, M., 37, 95

 Hadamard product, 54
 Halbwachs, N., 71, 95
 handle
 decomposer::, *see* decomposer::handle
 Hanlon, P. J., 94
 Hannig, F., 93
 Hardy, G. H., 12, 95
 Haws, D., 94
 Heckman, G. J., 12, 95
 height, 32
 Heine, F., 80, 95
 Hemmecke, R., 94
 Henrici, P., 44, 79, 95
 Huggins, P., 94
 hull
 affine, *see* affine hull
 positive, *see* positive hull

 implicit representation, 8
 indicator function, 25, 54
 initial counting, 76
 input size, 8
 integer projection, 7
 interpolation
 Vandermonde, *see* Vandermonde interpolation
 intersection, 54, 57

 Kannan, R., 37, 94
 Kelly, W., 77, 95
 Kienhuis, B., 94, 97
 Klein, R., 32, 92

Knuth, D. E., 95
 Latour, L., 4, 79, 93
 LattE, 12, 32, 45, 78, 79
 Lebeck, A. R., 94
 Lee, C. W., 32, 95, 96
 Lenstra, A. K., 37, 96
 Lenstra, H. W., 96
 Lenstra, Lenstra and Lovasz' basis reduction algorithm (LLL), 37–40, 42
 lifting, 32
 line, 5
 Lisoněk, P., 92
 Lisper, B., 81, 96
 LLL, 38
 Loechner, V., 3, 4, 9, 15, 16, 19, 21, 25, 63, 68, 75, 76, 78–83, 90, 94, 96, 97
 long_4D, 45
 Lovász, L., 95, 96
 lower envelope, 33
 lower face, 32

 magic square, 10
 Malik, S., 80, 95, 97
 Manocha, D., 76, 96
 Manocha, H. L., 8, 97
 Martonosi, M., 95
 masking out, 54
 Maslov, V., 95
 mat_ZZ, 86
 matlab, 81
 Matoušek, J., 17, 96
 McAllister, T. B., 9, 96
 Meister, B., 79, 96
 Merchat, D., 95
 Micchelli, C. A., 12, 94
 Miller, E., 9, 96
 Minkowski's First Theorem, 36
 moment curve, 43
 monomial substitution, 43
 multiple generating function, 8
 multiple sequence, 8

 n, 87
 NbBid, 82
 NbConstraints, 82
 NbEq, 82
 NbRays, 82
 new_eadd, 85
 next, 82
 Nootaert, B., 76, 96
 NTL, 38, 86

 O'Boyle, M., 58, 78, 81, 95

 Omega, 7, 77, 80
 Omega test, 66, 75
 operator evaluate *
 gen_fun::, *see* gen_fun::operator evaluate *
 outer wall, 53

 Pande, S., 80, 92
 Papadimitriou, C., 8, 96
 parallelepiped
 fundamental, *see* fundamental parallelepiped
 parameter, 7
 Parametric Integer Programming (PIP), 63, 68, 69, 79, 80, 88
 parametric polytope, 7
 chamber complex, *see* chamber complex of a parametric polytope
 parametric set, 7
 parametric vertex, 17
 ParamPolyhedron_Reduce, 71
 Parent-Vigouroux, C., 95
 Parker, E., 4, 79, 94, 96
 partition, 84–86
 partition function, 12
 vector, *see* vector partition function
 Patashnik, O., 95
 period, 9, 21
 periodic, 83, 84
 periodic number, 9, 21
 Pfeifle, J., 93
 piecewise step-polynomial, 24
 pointer conversion, *see* array recovery
 polar cone, 31
 polynomial
 step-, *see* step-polynomial
 polyhedral complex, 6
 polyhedral cone, 6
 Polyhedron, 82, 87, 89
 polyhedron
 rational, *see* rational polyhedron
 Polyhedron.Enumerate, 88
 Polyhedron.Image.Enumerate, 64
 Polyhedron.Polarize, 32, 88
 Polyhedron.Project, 89
 PolyhedronIncludes, 89
 PolyLib, 4, 21, 32, 33, 61, 64, 71, 76, 78, 82, 85, 86, 88–91
 polynomial
 corrector, *see* corrector polynomial
 Ehrhart, *see* Ehrhart polynomial
 polynomial, 83, 86
 polynomial time complexity, 8
 polytope
 parametric, *see* parametric polytope

- rational, *see* rational polytope
- polytope projection
 - chamber complex, *see* chamber complex
 - of a polytope projection
- Pommersheim, J., 4, 7, 25, 27, 29, 31, 39, 43, 47, 48, 54, 55, 64, 93
- pos, 83–85
- positive hull, **5**
- power, 87
- power series, 8
- Presburger formula, 7
- Presburger set, **7**
- print_evalue, 86
- prism, 71
- product
 - Hadamard, *see* Hadamard product
- projected set, **7**
- projection
 - integer, *see* integer projection
- proper triangulation, 7
- Pugh, W., 4, 7, 66, 75, 81, 95, 96

- quadratic constraint, 72
- quasi-polynomial
 - Ehrhart, *see* Ehrhart quasi-polynomial
- quasi-polynomial, **9, 21**

- Rambau, J., 15, 96
- rational generating function, **23**
 - short, *see* short rational generating function
- rational polyhedron, **5**
- rational polytope, **5**
- rational unimodular cone, **28**
- Ray, 82
- ray, **5**
- Redon, X., 63, 68, 72, 79, 80, 93
- reduce_evalue, 73, 86
- refinement, **6**
- regular triangulation, **32**
- relation, 84, 86
- representation theory, 12
- residue, 44
- Rijpkema, E., 81, 97
- Robins, S., 9, 10, 93, 94
- Rosser, E., 95

- Sarkar, V., 95
- Schrijver, A., 8, 31, 36, 37, 95, 97
- Seghir, R., 19, 64, 79, 85, 97
- selection, 54
- sequence, 8
 - multiple, *see* multiple sequence
- series
 - Ehrhart, *see* Ehrhart series
 - geometric, *see* geometric series
- set
 - parametric, *see* parametric set
 - Presburger, *see* Presburger set
 - projected, *see* projected set
- set difference, 57
- shadow
 - dark, *see* dark shadow
- short rational generating function, 23
- short_rat, 86
- short_vector
 - cone::, *see* cone::short_vector
- Shoup, V., 38, 97
- Shpeisman, T., 95
- simplicial cone, **6, 13, 32**
- size
 - input, *see* input size
- size, 83, 84
- Slowik, A., 80, 95
- specialization, 43
- splinter, 75
- Srivastava, H. M., 8, 97
- Stanley, R. P., 9, 12, 93, 97
- statistics, 12
- step-polynomial, **24**
 - piecewise, *see* piecewise step-polynomial
- stride, 24, **72, 84**
- Sturmfels, B., 9, 12–15, 94, 96, 97
- subdivision, **6**
- substitution
 - monomial, *see* monomial substitution
- summation, **58**
- support, **6**
- supporting cone, **29**

- Tauzer, J., 94
- Teich, J., 93
- term, 87
- thick facet, **64**
- Thrash, W., 95
- time complexity, 8
- triangularize_cone, 33
- triangulation, **7**
 - Delaunay, *see* Delaunay triangulation
 - proper, *see* proper triangulation
 - regular, *see* regular triangulation
- Turjan, A., 76, 78, 81, 94, 97
- type, 83

- unimodular cone
 - rational, *see* rational unimodular cone
- unimodular matrix, 28, 88
- unimodular_complete, 88

- union, 57, 85
- validity domain, 16
- Value, 82, 86, 88
- van Engelen, R. A., 58, 97
- Vandermonde interpolation, 76
- vec_ZZ, 86
- vector partition function, **12**, 79
 - chamber complex, *see* chamber complex
 - of a vector partition function
- Verdoolaege, S., 24, 48, 63, 79, 97
- vertex, **5**
 - active, *see* active vertex
 - parametric, *see* parametric vertex
- wall
 - outer, *see* outer wall
- Wilde, D. K., 19, 21, 31, 82, 96, 97
- Wonnacott, D., 95
- Woods, K., 4, 9, 24, 43, 60, 63, 69, 70, 72, 93, 96, 97
- Wright, E. M., 12, 95
- x.n, 83
- x.p, 83
- Yoshida, R., 10, 12, 25, 45, 94, 97
- Yovine, S., 93
- Zelevinskiĭ, A. V., 92
- Zhao, Y., 80, 97
- Zissulescu, C., 94
- ZZ, 86