Quasi-Polynomials, Z-Polyhedra and Parametric Integer Linear Programming

Rui-Juan Jing [1], Yuzhuo Lei [2], Marc Moreno Maza [2], Chirantan Mukherjee [2]

[2]Ontario Research Center for Computer Algebra (ORCCA), UWO, London, Ontario
[1]School of Mathematical Science, Jiangsu University, Zhenjiang, China

July 21, 2026

Acknowledgements

- This talk is based on research projects in which **many former and current ORCCA students** have played an essential role. By alphabetic order: Xiaohui Chen, Rui-Juan Jing, Yuzhuo Lei, Christopher Maligec, Chirantan Mukherjee, Delaram Talaashrafi, Linxiao Wang, Ning Xie, Rong Xiao and Haoze Yuan.

Acknowledgements

- This talk is based on research projects in which **many former and current ORCCA students** have played an essential role. By alphabetic order: Xiaohui Chen, Rui-Juan Jing, Yuzhuo Lei, Christopher Maligec, Chirantan Mukherjee, Delaram Talaashrafi, Linxiao Wang, Ning Xie, Rong Xiao and Haoze Yuan.
- This talk is based on **academic and industry collaborations** with Maplesoft, MIT/CSAIL, NVIDIA, Intel and IBM Canada, with funding support from Maplesoft, MITACS, IBM and NSERC of Canada, together with hardware/software from NVIDIA and Intel.

Acknowledgements

- This talk is based on research projects in which **many former and current ORCCA students** have played an essential role. By alphabetic order: Xiaohui Chen, Rui-Juan Jing, Yuzhuo Lei, Christopher Maligec, Chirantan Mukherjee, Delaram Talaashrafi, Linxiao Wang, Ning Xie, Rong Xiao and Haoze Yuan.
- This talk is based on **academic and industry collaborations** with Maplesoft, MIT/CSAIL, NVIDIA, Intel and IBM Canada, with funding support from Maplesoft, MITACS, IBM and NSERC of Canada, together with hardware/software from NVIDIA and Intel.
- Most of the algorithms presented in this software demo are **available in** MAPLE**'s** `PolyhedralSets` library.

Motivations and objectives

- Studying the integer solutions of linear systems of equations and inequalities is of practical importance in various areas of scientific computing:

Motivations and objectives

- Studying the integer solutions of linear systems of equations and inequalities is of practical importance in various areas of scientific computing:
  - **combinatorial optimization**, in particular, integer linear programming,

Motivations and objectives

- Studying the integer solutions of linear systems of equations and inequalities is of practical importance in various areas of scientific computing:
  - **combinatorial optimization**, in particular, integer linear programming,
  - **compiler optimization** in particular, the analysis, transformation, and scheduling of nested loops in computer programs.

Software architecture and history

Component	Release year	Key functionalities
`PolyhedralSets`	2015	Basic routines for polyhedra over $\mathbb{Q}$

Software architecture and history

Component	Release year	Key functionalities
`PolyhedralSets`	2015	Basic routines for polyhedra over $\mathbb{Q}$
`IntegerHull`	2022	fast computations of integer hulls

Software architecture and history

Component	Release year	Key functionalities
`PolyhedralSets`	2015	Basic routines for polyhedra over $\mathbb{Q}$
`IntegerHull`	2022	fast computations of integer hulls
`ZPolyhedralSets`	2023	$\mathbb{Z}$-polyhedra

Software architecture and history

Component	Release year	Key functionalities
`PolyhedralSets`	2015	Basic routines for polyhedra over $\mathbb{Q}$
`IntegerHull`	2022	fast computations of integer hulls
`ZPolyhedralSets`	2023	$\mathbb{Z}$-polyhedra
`FME_SatMat`	2025	FME, double-description method

Software architecture and history

Component	Release year	Key functionalities
`PolyhedralSets`	2015	Basic routines for polyhedra over $\mathbb{Q}$
`IntegerHull`	2022	fast computations of integer hulls
`ZPolyhedralSets`	2023	$\mathbb{Z}$-polyhedra
`FME_SatMat`	2025	FME, double-description method
`NumberOfIntegerPoints`	2025	Ehrhart polynomials

Software architecture and history

Component	Release year	Key functionalities
`PolyhedralSets`	2015	Basic routines for polyhedra over $\mathbb{Q}$
`IntegerHull`	2022	fast computations of integer hulls
`ZPolyhedralSets`	2023	$\mathbb{Z}$-polyhedra
`FME_SatMat`	2025	FME, double-description method
`NumberOfIntegerPoints`	2025	Ehrhart polynomials
`ValuesUnderConstrainsts`	2025	computations with piece-wise functions

Software architecture and history

Component	Release year	Key functionalities
`PolyhedralSets`	2015	Basic routines for polyhedra over $\mathbb{Q}$
`IntegerHull`	2022	fast computations of integer hulls
`ZPolyhedralSets`	2023	$\mathbb{Z}$-polyhedra
`FME_SatMat`	2025	FME, double-description method
`NumberOfIntegerPoints`	2025	Ehrhart polynomials
`ValuesUnderConstrainsts`	2025	computations with piece-wise functions
`QuasiPolynomials`	2025	computations with quasi-polynomials

Software architecture and history

Component	Release year	Key functionalities
`PolyhedralSets`	2015	Basic routines for polyhedra over $\mathbb{Q}$
`IntegerHull`	2022	fast computations of integer hulls
`ZPolyhedralSets`	2023	$\mathbb{Z}$-polyhedra
`FME_SatMat`	2025	FME, double-description method
`NumberOfIntegerPoints`	2025	Ehrhart polynomials
`ValuesUnderConstrainsts`	2025	computations with piece-wise functions
`QuasiPolynomials`	2025	computations with quasi-polynomials
`QuantifierEliminationOverZ`	new	QE over $\mathbb{Z}$ (back-engine)

Software architecture and history

Component	Release year	Key functionalities
`PolyhedralSets`	2015	Basic routines for polyhedra over $\mathbb{Q}$
`IntegerHull`	2022	fast computations of integer hulls
`ZPolyhedralSets`	2023	$\mathbb{Z}$-polyhedra
`FME_SatMat`	2025	FME, double-description method
`NumberOfIntegerPoints`	2025	Ehrhart polynomials
`ValuesUnderConstrainsts`	2025	computations with piece-wise functions
`QuasiPolynomials`	2025	computations with quasi-polynomials
`QuantifierEliminationOverZ`	new	QE over $\mathbb{Z}$ (back-engine)
`PresburgerFormulas`	new	QE over $\mathbb{Z}$ (front-engine)
`PILP`	new	parametric integer linear programming

Software architecture and history

Component	Release year	Key functionalities
`PolyhedralSets`	2015	Basic routines for polyhedra over $\mathbb{Q}$
`IntegerHull`	2022	fast computations of integer hulls
`ZPolyhedralSets`	2023	$\mathbb{Z}$-polyhedra
`FME_SatMat`	2025	FME, double-description method
`NumberOfIntegerPoints`	2025	Ehrhart polynomials
`ValuesUnderConstrainsts`	2025	computations with piece-wise functions
`QuasiPolynomials`	2025	computations with quasi-polynomials
`QuantifierEliminationOverZ`	new	QE over $\mathbb{Z}$ (back-engine)
`PresburgerFormulas`	new	QE over $\mathbb{Z}$ (front-engine)
`PILP`	new	parametric integer linear programming

- All these components are libraries, except `IntegerHull`, `NumberOfIntegerPoints` and `PILP` which are commands.

Software architecture and history

Component	Release year	Key functionalities
`PolyhedralSets`	2015	Basic routines for polyhedra over $\mathbb{Q}$
`IntegerHull`	2022	fast computations of integer hulls
`ZPolyhedralSets`	2023	$\mathbb{Z}$-polyhedra
`FME_SatMat`	2025	FME, double-description method
`NumberOfIntegerPoints`	2025	Ehrhart polynomials
`ValuesUnderConstrainsts`	2025	computations with piece-wise functions
`QuasiPolynomials`	2025	computations with quasi-polynomials
`QuantifierEliminationOverZ`	new	QE over $\mathbb{Z}$ (back-engine)
`PresburgerFormulas`	new	QE over $\mathbb{Z}$ (front-engine)
`PILP`	new	parametric integer linear programming

- All these components are libraries, except `IntegerHull`, `NumberOfIntegerPoints` and `PILP` which are commands.
- All libraries and commands are MAPLE code, except `FME_SatMat` which is is written in C/C++ in support of efficiency critical routines.

Plan

1. Overview
2. Integer hulls of polyhedra
3. Integer point counting for parametric polyhedra
4. Parametric Integer Linear Programming
5. Array Bound Checking via PILP
6. Concluding remarks

Plan

Integer hulls and lattices

- A subset $P \subseteq \mathbb{Q}^n$ is called a convex polyhedral set (or simply a polyhedral set) if
$$P = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$$
holds, for a matrix $A \in \mathbb{Q}^{m \times n}$ and a vector $\mathbf{b} \in \mathbb{Q}^m$, where n, m are positive integers.

Integer hulls and lattices

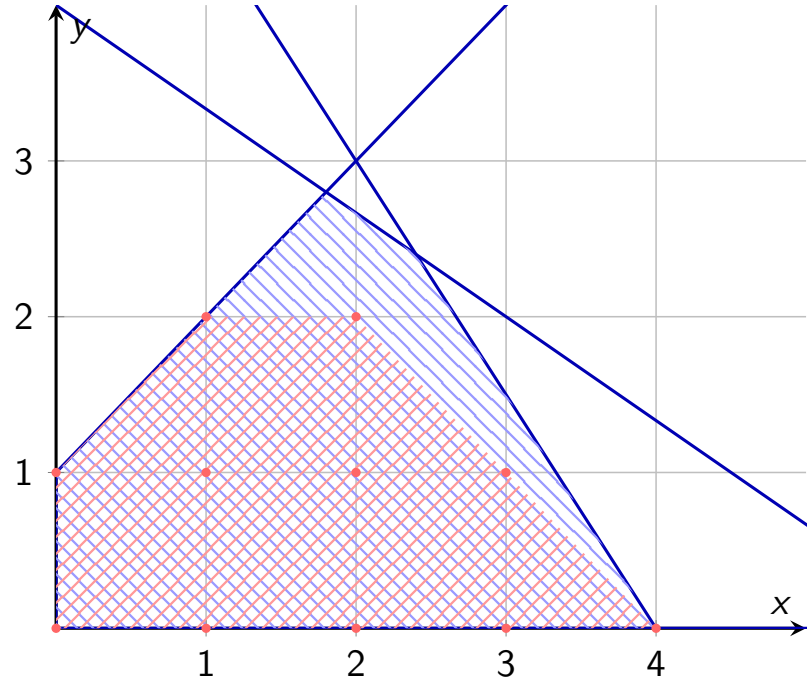
- A subset $P \subseteq \mathbb{Q}^n$ is called a convex polyhedral set (or simply a polyhedral set) if
$$P = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$$
holds, for a matrix $A \in \mathbb{Q}^{m \times n}$ and a vector $\mathbf{b} \in \mathbb{Q}^m$, where n, m are positive integers.
- We are interested in computing P_I the *integer hull* of P that is the smallest convex polyhedral set containing all the integer points of P.

Integer hulls and lattices

- A subset $P \subseteq \mathbb{Q}^n$ is called a convex polyhedral set (or simply a polyhedral set) if
$$P = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$$
holds, for a matrix $A \in \mathbb{Q}^{m \times n}$ and a vector $\mathbf{b} \in \mathbb{Q}^m$, where n, m are positive integers.
- We are interested in computing P_I the *integer hull* of P that is the smallest convex polyhedral set containing all the integer points of P.

$$\begin{cases} 0 \leq x \\ 0 \leq y \\ 3x + 2y \leq 12 \\ 2x + 3y \leq 12 \\ -x + y \leq 1 \end{cases}$$

$$\begin{cases} 0 \leq x \\ 0 \leq y \\ y \leq 2 \\ x + y \leq 4 \\ -x + y \leq 1 \end{cases}$$



Integer lattices

- A subset $L \subseteq \mathbb{Z}^d$ is called an integer lattice (or lattice) if
$$L = \{\mathbf{x} \in \mathbb{Z}^d \mid (\exists\, \mathbf{t} \in \mathbb{Z}^c)\ \mathbf{x} = A\mathbf{t} + \mathbf{b}\}$$
for $A \in \mathbb{Z}^{d \times c}$, $\mathbf{b} \in \mathbb{Z}^d$, $c \in \mathbb{Z}_{>0}$.

Integer lattices

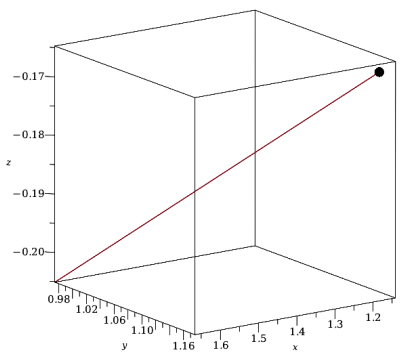
- A subset $L \subseteq \mathbb{Z}^d$ is called an integer lattice (or lattice) if
$$L = \{\mathbf{x} \in \mathbb{Z}^d \mid (\exists\, \mathbf{t} \in \mathbb{Z}^c)\ \mathbf{x} = A\mathbf{t} + \mathbf{b}\}$$
for $A \in \mathbb{Z}^{d\times c}$, $\mathbf{b} \in \mathbb{Z}^d$, $c \in \mathbb{Z}_{>0}$.
- A $\mathbb{Z}$-polyhedron is the intersection of a polyhedron $P \subseteq \mathbb{Q}^d$ and a lattice $L \subseteq \mathbb{Z}^d$, denoted $\mathbb{Z}\text{Polyhedron}(P, L)$.

Integer lattices

- A subset $L \subseteq \mathbb{Z}^d$ is called an integer lattice (or lattice) if
$$L = \{\mathbf{x} \in \mathbb{Z}^d \mid (\exists\, \mathbf{t} \in \mathbb{Z}^c)\ \mathbf{x} = A\mathbf{t} + \mathbf{b}\}$$
for $A \in \mathbb{Z}^{d\times c}$, $\mathbf{b} \in \mathbb{Z}^d$, $c \in \mathbb{Z}_{>0}$.
- A $\mathbb{Z}$-polyhedron is the intersection of a polyhedron $P \subseteq \mathbb{Q}^d$ and a lattice $L \subseteq \mathbb{Z}^d$, denoted $\mathbb{Z}\text{Polyhedron}(P, L)$.

Input system:

$$\begin{cases} 7x + 12y + 31z &= 17, \\ 3x + 5y + 14z &= 7, \\ x + y &\geq 0, \\ y - x &\leq 0. \end{cases}$$

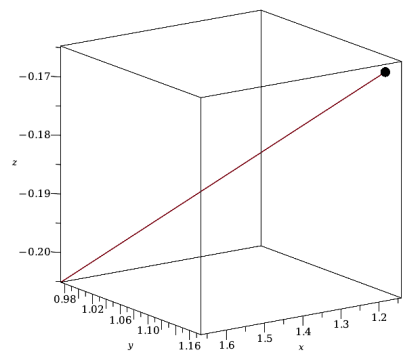


Integer lattices

- A subset $L \subseteq \mathbb{Z}^d$ is called an integer lattice (or lattice) if
$$L = \{\mathbf{x} \in \mathbb{Z}^d \mid (\exists\, \mathbf{t} \in \mathbb{Z}^c)\ \mathbf{x} = A\mathbf{t} + \mathbf{b}\}$$
for $A \in \mathbb{Z}^{d\times c}$, $\mathbf{b} \in \mathbb{Z}^d$, $c \in \mathbb{Z}_{>0}$.
- A $\mathbb{Z}$-polyhedron is the intersection of a polyhedron $P \subseteq \mathbb{Q}^d$ and a lattice $L \subseteq \mathbb{Z}^d$, denoted $\mathbb{Z}\text{Polyhedron}(P, L)$.

Input system:

$$\begin{cases} 7x + 12y + 31z &= 17, \\ 3x + 5y + 14z &= 7, \\ x + y &\geq 0, \\ y - x &\leq 0. \end{cases}$$



Equivalent Z-Polyhedron:

$$\begin{cases} x &= -13z - 1, \\ y &= 5z + 2, \\ z &\leq -1. \end{cases} \quad \text{and} \quad \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -13 \\ 5 \\ 1 \end{pmatrix}\mathbf{t} + \begin{pmatrix} -1 \\ 2 \\ 0 \end{pmatrix} \in \text{Lattice}\left(\begin{pmatrix} -13 \\ 5 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 2 \\ 0 \end{pmatrix}\right)$$

$\mathbb{Z}$-polyhedra

1. Recall: $\mathbb{Z}\text{Polyhedron}(P, L)$ is the intersection (in $\mathbb{Z}^d$) of the polyhedron $P \subseteq \mathbb{Q}^d$ and the lattice $L \subseteq \mathbb{Z}^d$.

$\mathbb{Z}$-polyhedra

1. Recall: $\mathbb{Z}\text{Polyhedron}(P, L)$ is the intersection (in $\mathbb{Z}^d$) of the polyhedron $P \subseteq \mathbb{Q}^d$ and the lattice $L \subseteq \mathbb{Z}^d$.
2. Denote by $x_1 < x_2 < \cdots < x_d$ the coordinates of $\mathbb{Z}^d$. We say that $\mathbb{Z}\text{Polyhedron}(P, L)$ is **normalized** if

$\mathbb{Z}$-polyhedra

1. Recall: $\mathbb{Z}\text{Polyhedron}(P, L)$ is the intersection (in $\mathbb{Z}^d$) of the polyhedron $P \subseteq \mathbb{Q}^d$ and the lattice $L \subseteq \mathbb{Z}^d$.
2. Denote by $x_1 < x_2 < \cdots < x_d$ the coordinates of $\mathbb{Z}^d$. We say that $\mathbb{Z}\text{Polyhedron}(P, L)$ is **normalized** if
   a. it is non-empty, and P is given by a system of linear inequalities of the form

$$\begin{cases} a_0 \leq x_1 \leq b_0 \\ a_1 \leq x_2 \leq b_1 \\ \vdots \quad \vdots \quad \vdots \\ a_{n-1} \leq x_d \leq b_{n-1}, \end{cases} \tag{2.1}$$

   where

$\mathbb{Z}$-polyhedra

1. Recall: $\mathbb{Z}\text{Polyhedron}(P, L)$ is the intersection (in $\mathbb{Z}^d$) of the polyhedron $P \subseteq \mathbb{Q}^d$ and the lattice $L \subseteq \mathbb{Z}^d$.
2. Denote by $x_1 < x_2 < \cdots < x_d$ the coordinates of $\mathbb{Z}^d$. We say that $\mathbb{Z}\text{Polyhedron}(P, L)$ is **normalized** if
   a. it is non-empty, and P is given by a system of linear inequalities of the form

$$\begin{cases} a_0 & \leq x_1 \leq & b_0 \\ a_1 & \leq x_2 \leq & b_1 \\ \vdots & \vdots & \vdots \\ a_{n-1} & \leq x_d \leq & b_{n-1}, \end{cases} \tag{2.1}$$

   where
   b. a_i (resp. b_i) is either $-\infty$ (resp. $+\infty$) or an expression of the form $\max(\ell_{i,1} \ldots \ell_{i,e_i})$ (resp. $\min(\ell_{i,1} \ldots \ell_{i,e_i})$), and

$\mathbb{Z}$-polyhedra

1. Recall: $\mathbb{Z}\text{Polyhedron}(P, L)$ is the intersection (in $\mathbb{Z}^d$) of the polyhedron $P \subseteq \mathbb{Q}^d$ and the lattice $L \subseteq \mathbb{Z}^d$.
2. Denote by $x_1 < x_2 < \cdots < x_d$ the coordinates of $\mathbb{Z}^d$. We say that $\mathbb{Z}\text{Polyhedron}(P, L)$ is **normalized** if
   a. it is non-empty, and P is given by a system of linear inequalities of the form

$$\begin{cases} a_0 & \leq x_1 \leq & b_0 \\ a_1 & \leq x_2 \leq & b_1 \\ \vdots & \vdots & \vdots \\ a_{n-1} & \leq x_d \leq & b_{n-1}, \end{cases} \tag{2.1}$$

   where
   b. a_i (resp. b_i) is either $-\infty$ (resp. $+\infty$) or an expression of the form $\max(\ell_{i,1} \ldots \ell_{i,e_i})$ (resp. $\min(\ell_{i,1} \ldots \ell_{i,e_i})$), and
   c. each $\ell_{i,j} \in \mathbb{Q}[x_1, \ldots, x_{i-1}]$ with degree at most 1, so that

$\mathbb{Z}$-polyhedra

1. Recall: $\mathbb{Z}\text{Polyhedron}(P, L)$ is the intersection (in $\mathbb{Z}^d$) of the polyhedron $P \subseteq \mathbb{Q}^d$ and the lattice $L \subseteq \mathbb{Z}^d$.
2. Denote by $x_1 < x_2 < \cdots < x_d$ the coordinates of $\mathbb{Z}^d$. We say that $\mathbb{Z}\text{Polyhedron}(P, L)$ is **normalized** if
   a. it is non-empty, and P is given by a system of linear inequalities of the form

$$\begin{cases} a_0 & \leq x_1 \leq & b_0 \\ a_1 & \leq x_2 \leq & b_1 \\ \vdots & \vdots & \vdots \\ a_{n-1} & \leq x_d \leq & b_{n-1}, \end{cases} \tag{2.1}$$

   where
   b. a_i (resp. b_i) is either $-\infty$ (resp. $+\infty$) or an expression of the form $\max(\ell_{i,1} \ldots \ell_{i,e_i})$ (resp. $\min(\ell_{i,1} \ldots \ell_{i,e_i})$), and
   c. each $\ell_{i,j} \in \mathbb{Q}[x_1, \ldots, x_{i-1}]$ with degree at most 1, so that
   d. all the integer points of P are obtained by **back substitution**, that is, by specializing x_1 to every integer value v_1 in the interval (a_0, b_0), then by specializing x_2 to every integer value v_2 in the interval $(a_1(v_1), b_1(v_1))$, and so on.

$\mathbb{Z}$-polyhedra

1. Recall: $\mathbb{Z}$Polyhedron(P, L) is the intersection (in $\mathbb{Z}^d$) of the polyhedron $P \subseteq \mathbb{Q}^d$ and the lattice $L \subseteq \mathbb{Z}^d$.
2. Denote by $x_1 < x_2 < \cdots < x_d$ the coordinates of $\mathbb{Z}^d$. We say that $\mathbb{Z}$Polyhedron(P, L) is **normalized** if
   a. it is non-empty, and P is given by a system of linear inequalities of the form

$$\begin{cases} a_0 \leq x_1 \leq b_0 \\ a_1 \leq x_2 \leq b_1 \\ \vdots \quad \vdots \quad \vdots \\ a_{n-1} \leq x_d \leq b_{n-1}, \end{cases} \tag{2.1}$$

   where
   b. a_i (resp. b_i) is either $-\infty$ (resp. $+\infty$) or an expression of the form $\max(\ell_{i,1} \ldots \ell_{i,e_i})$ (resp. $\min(\ell_{i,1} \ldots \ell_{i,e_i})$), and
   c. each $\ell_{i,j} \in \mathbb{Q}[x_1, \ldots, x_{i-1}]$ with degree at most 1, so that
   d. all the integer points of P are obtained by **back substitution**, that is, by specializing x_1 to every integer value v_1 in the interval (a_0, b_0), then by specializing x_2 to every integer value v_2 in the interval $(a_1(v_1), b_1(v_1))$, and so on.
3. The algorithm IntegerPointDecomposition [7] decomposes any $\mathbb{Z}$-polyhedron into **normalized** $\mathbb{Z}$-polyhedra.

Plan

Generating functions of polyedra

- Consider a polyhedral set $P \subseteq \mathbb{Q}^d$ and map each integer point $\mathbf{e} = (e_1, \ldots, e_d)$ of P to the monomial $\mathbf{x}^{\mathbf{e}} = {x_1}^{e_1} \cdots {x_d}^{e_d}$

Generating functions of polyedra

- Consider a polyhedral set $P \subseteq \mathbb{Q}^d$ and map each integer point $\mathbf{e} = (e_1, \ldots, e_d)$ of P to the monomial $\mathbf{x}^{\mathbf{e}} = {x_1}^{e_1} \cdots {x_d}^{e_d}$
- The **generating function** of P is the formal Laurent series:

$$G(P, \mathbf{x}) = \sum_{\mathbf{e} \in P \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{e}}.$$

Generating functions of polyedra

- Consider a polyhedral set $P \subseteq \mathbb{Q}^d$ and map each integer point $\mathbf{e} = (e_1, \ldots, e_d)$ of P to the monomial $\mathbf{x}^{\mathbf{e}} = {x_1}^{e_1} \cdots {x_d}^{e_d}$
- The **generating function** of P is the formal Laurent series:
$$G(P, \mathbf{x}) = \sum_{\mathbf{e} \in P \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{e}}.$$
- If P is bounded, then $G(P, (1, \ldots, 1))$ counts its integer points.

Generating functions of polyedra

- Consider a polyhedral set $P \subseteq \mathbb{Q}^d$ and map each integer point $\mathbf{e} = (e_1, \ldots, e_d)$ of P to the monomial $\mathbf{x}^{\mathbf{e}} = {x_1}^{e_1} \cdots {x_d}^{e_d}$
- The **generating function** of P is the formal Laurent series:
$$G(P, \mathbf{x}) = \sum_{\mathbf{e} \in P \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{e}}.$$
- If P is bounded, then $G(P, (1, \ldots, 1))$ counts its integer points.
- If P is not bounded, then $G(P, \mathbf{x})$ is a formal Laurent series and can still be manipulated algorithmically.

Generating functions of polyedra

- Consider a polyhedral set $P \subseteq \mathbb{Q}^d$ and map each integer point $\mathbf{e} = (e_1, \ldots, e_d)$ of P to the monomial $\mathbf{x}^{\mathbf{e}} = {x_1}^{e_1} \cdots {x_d}^{e_d}$
- The **generating function** of P is the formal Laurent series:
$$G(P, \mathbf{x}) = \sum_{\mathbf{e} \in P \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{e}}.$$
- If P is bounded, then $G(P, (1, \ldots, 1))$ counts its integer points.
- If P is not bounded, then $G(P, \mathbf{x})$ is a formal Laurent series and can still be manipulated algorithmically.
- For $d = 2$, suppose P is the ray given by $y = 0$ and $x \geq 0$, then:

$$G(P, \mathbf{x}) =$$

Generating functions of polyedra

- Consider a polyhedral set $P \subseteq \mathbb{Q}^d$ and map each integer point $\mathbf{e} = (e_1, \ldots, e_d)$ of P to the monomial $\mathbf{x}^{\mathbf{e}} = {x_1}^{e_1} \cdots {x_d}^{e_d}$
- The **generating function** of P is the formal Laurent series:
$$G(P, \mathbf{x}) = \sum_{\mathbf{e} \in P \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{e}}.$$
- If P is bounded, then $G(P, (1, \ldots, 1))$ counts its integer points.
- If P is not bounded, then $G(P, \mathbf{x})$ is a formal Laurent series and can still be manipulated algorithmically.
- For $d = 2$, suppose P is the ray given by $y = 0$ and $x \geq 0$, then:
$$G(P, \mathbf{x}) = \sum_{n=0}^{n=\infty} (x, y)^{(n,0)} =$$

Generating functions of polyedra

- Consider a polyhedral set $P \subseteq \mathbb{Q}^d$ and map each integer point $\mathbf{e} = (e_1, \ldots, e_d)$ of P to the monomial $\mathbf{x}^{\mathbf{e}} = {x_1}^{e_1} \cdots {x_d}^{e_d}$
- The **generating function** of P is the formal Laurent series:
$$G(P, \mathbf{x}) = \sum_{\mathbf{e} \in P \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{e}}.$$
- If P is bounded, then $G(P, (1, \ldots, 1))$ counts its integer points.
- If P is not bounded, then $G(P, \mathbf{x})$ is a formal Laurent series and can still be manipulated algorithmically.
- For $d = 2$, suppose P is the ray given by $y = 0$ and $x \geq 0$, then:
$$G(P, \mathbf{x}) = \sum_{n=0}^{n=\infty} (x, y)^{(n,0)} = \sum_{n=0}^{n=\infty} x^n y^0 =$$

Generating functions of polyedra

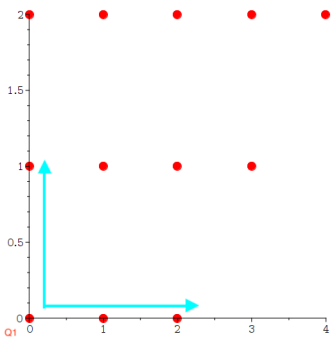
- Consider a polyhedral set $P \subseteq \mathbb{Q}^d$ and map each integer point $\mathbf{e} = (e_1, \ldots, e_d)$ of P to the monomial $\mathbf{x}^{\mathbf{e}} = {x_1}^{e_1} \cdots {x_d}^{e_d}$
- The **generating function** of P is the formal Laurent series:
$$G(P, \mathbf{x}) = \sum_{\mathbf{e} \in P \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{e}}.$$
- If P is bounded, then $G(P, (1, \ldots, 1))$ counts its integer points.
- If P is not bounded, then $G(P, \mathbf{x})$ is a formal Laurent series and can still be manipulated algorithmically.
- For $d = 2$, suppose P is the ray given by $y = 0$ and $x \geq 0$, then:
$$G(P, \mathbf{x}) = \sum_{n=0}^{n=\infty} (x, y)^{(n,0)} = \sum_{n=0}^{n=\infty} x^n y^0 = \sum_{n=0}^{n=\infty} x^n =$$

Generating functions of polyedra

- Consider a polyhedral set $P \subseteq \mathbb{Q}^d$ and map each integer point $\mathbf{e} = (e_1, \ldots, e_d)$ of P to the monomial $\mathbf{x}^{\mathbf{e}} = {x_1}^{e_1} \cdots {x_d}^{e_d}$
- The **generating function** of P is the formal Laurent series:
$$G(P, \mathbf{x}) = \sum_{\mathbf{e} \in P \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{e}}.$$
- If P is bounded, then $G(P, (1, \ldots, 1))$ counts its integer points.
- If P is not bounded, then $G(P, \mathbf{x})$ is a formal Laurent series and can still be manipulated algorithmically.
- For $d = 2$, suppose P is the ray given by $y = 0$ and $x \geq 0$, then:
$$G(P, \mathbf{x}) = \sum_{n=0}^{n=\infty} (x, y)^{(n,0)} = \sum_{n=0}^{n=\infty} x^n y^0 = \sum_{n=0}^{n=\infty} x^n = \frac{1}{1-x}.$$

Generating functions of polyedra

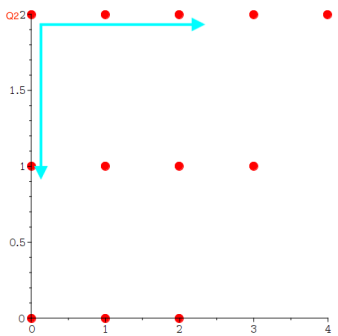
- Consider a polyhedral set $P \subseteq \mathbb{Q}^d$ and map each integer point $\mathbf{e} = (e_1, \ldots, e_d)$ of P to the monomial $\mathbf{x}^{\mathbf{e}} = {x_1}^{e_1} \cdots {x_d}^{e_d}$
- The **generating function** of P is the formal Laurent series:
$$G(P, \mathbf{x}) = \sum_{\mathbf{e} \in P \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{e}}.$$
- If P is bounded, then $G(P, (1, \ldots, 1))$ counts its integer points.
- If P is not bounded, then $G(P, \mathbf{x})$ is a formal Laurent series and can still be manipulated algorithmically.
- For $d = 2$, suppose P is the ray given by $y = 0$ and $x \geq 0$, then:
$$G(P, \mathbf{x}) = \sum_{n=0}^{n=\infty} (x, y)^{(n,0)} = \sum_{n=0}^{n=\infty} x^n y^0 = \sum_{n=0}^{n=\infty} x^n = \frac{1}{1-x}.$$
- Still for $d = 2$, $G(P, \mathbf{x})$ is computed as the sum of the generating functions of its vertex cones, thanks to Brion's theorem (1988) [2].



$G(P, \mathbf{x}) \quad = \quad G(Q_1, \mathbf{x}) + \qquad + \qquad +$

Generating functions of polyedra

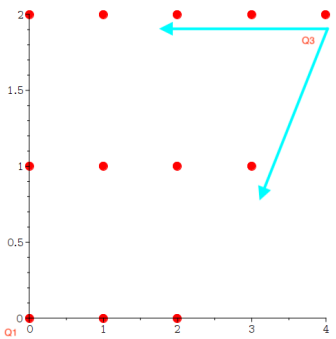
- Consider a polyhedral set $P \subseteq \mathbb{Q}^d$ and map each integer point $\mathbf{e} = (e_1, \ldots, e_d)$ of P to the monomial $\mathbf{x}^{\mathbf{e}} = {x_1}^{e_1} \cdots {x_d}^{e_d}$
- The **generating function** of P is the formal Laurent series:
$$G(P, \mathbf{x}) = \sum_{\mathbf{e} \in P \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{e}}.$$
- If P is bounded, then $G(P, (1, \ldots, 1))$ counts its integer points.
- If P is not bounded, then $G(P, \mathbf{x})$ is a formal Laurent series and can still be manipulated algorithmically.
- For $d = 2$, suppose P is the ray given by $y = 0$ and $x \geq 0$, then:
$$G(P, \mathbf{x}) = \sum_{n=0}^{n=\infty} (x, y)^{(n,0)} = \sum_{n=0}^{n=\infty} x^n y^0 = \sum_{n=0}^{n=\infty} x^n = \frac{1}{1-x}.$$
- Still for $d = 2$, $G(P, \mathbf{x})$ is computed as the sum of the generating functions of its vertex cones, thanks to Brion's theorem (1988) [2].



$$G(P, \mathbf{x}) \quad = \quad G(Q_1, \mathbf{x}) + G(Q_2, \mathbf{x}) + \qquad\qquad +$$

Generating functions of polyedra

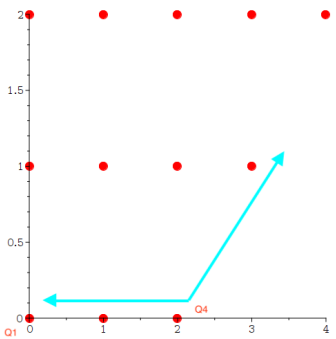
- Consider a polyhedral set $P \subseteq \mathbb{Q}^d$ and map each integer point $\mathbf{e} = (e_1, \ldots, e_d)$ of P to the monomial $\mathbf{x}^{\mathbf{e}} = {x_1}^{e_1} \cdots {x_d}^{e_d}$
- The **generating function** of P is the formal Laurent series:
$$G(P, \mathbf{x}) = \sum_{\mathbf{e} \in P \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{e}}.$$
- If P is bounded, then $G(P, (1, \ldots, 1))$ counts its integer points.
- If P is not bounded, then $G(P, \mathbf{x})$ is a formal Laurent series and can still be manipulated algorithmically.
- For $d = 2$, suppose P is the ray given by $y = 0$ and $x \geq 0$, then:
$$G(P, \mathbf{x}) = \sum_{n=0}^{n=\infty} (x, y)^{(n,0)} = \sum_{n=0}^{n=\infty} x^n y^0 = \sum_{n=0}^{n=\infty} x^n = \frac{1}{1-x}.$$
- Still for $d = 2$, $G(P, \mathbf{x})$ is computed as the sum of the generating functions of its vertex cones, thanks to Brion's theorem (1988) [2].



$$G(P, \mathbf{x}) \quad = \quad G(Q_1, \mathbf{x}) + G(Q_2, \mathbf{x}) + G(Q_3, \mathbf{x}) +$$

Generating functions of polyedra

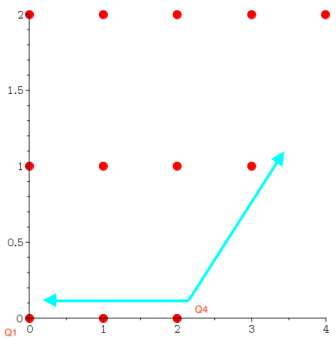
- Consider a polyhedral set $P \subseteq \mathbb{Q}^d$ and map each integer point $\mathbf{e} = (e_1, \ldots, e_d)$ of P to the monomial $\mathbf{x}^{\mathbf{e}} = {x_1}^{e_1} \cdots {x_d}^{e_d}$
- The **generating function** of P is the formal Laurent series:
$$G(P, \mathbf{x}) = \sum_{\mathbf{e} \in P \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{e}}.$$
- If P is bounded, then $G(P, (1, \ldots, 1))$ counts its integer points.
- If P is not bounded, then $G(P, \mathbf{x})$ is a formal Laurent series and can still be manipulated algorithmically.
- For $d = 2$, suppose P is the ray given by $y = 0$ and $x \geq 0$, then:
$$G(P, \mathbf{x}) = \sum_{n=0}^{n=\infty} (x, y)^{(n,0)} = \sum_{n=0}^{n=\infty} x^n y^0 = \sum_{n=0}^{n=\infty} x^n = \frac{1}{1-x}.$$
- Still for $d = 2$, $G(P, \mathbf{x})$ is computed as the sum of the generating functions of its vertex cones, thanks to Brion's theorem (1988) [2].



$$G(P, \mathbf{x}) \quad = \quad G(Q_1, \mathbf{x}) + G(Q_2, \mathbf{x}) + G(Q_3, \mathbf{x}) + G(Q_4, \mathbf{x})$$

Generating functions of polyedra

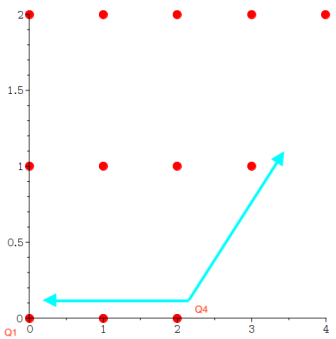
- Consider a polyhedral set $P \subseteq \mathbb{Q}^d$ and map each integer point $\mathbf{e} = (e_1, \ldots, e_d)$ of P to the monomial $\mathbf{x}^{\mathbf{e}} = {x_1}^{e_1} \cdots {x_d}^{e_d}$
- The **generating function** of P is the formal Laurent series:
$$G(P, \mathbf{x}) = \sum_{\mathbf{e} \in P \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{e}}.$$
- If P is bounded, then $G(P, (1, \ldots, 1))$ counts its integer points.
- If P is not bounded, then $G(P, \mathbf{x})$ is a formal Laurent series and can still be manipulated algorithmically.
- For $d = 2$, suppose P is the ray given by $y = 0$ and $x \geq 0$, then:
$$G(P, \mathbf{x}) = \sum_{n=0}^{n=\infty} (x, y)^{(n,0)} = \sum_{n=0}^{n=\infty} x^n y^0 = \sum_{n=0}^{n=\infty} x^n = \frac{1}{1-x}.$$
- Still for $d = 2$, $G(P, \mathbf{x})$ is computed as the sum of the generating functions of its vertex cones, thanks to Brion's theorem (1988) [2].



$$G(P, \mathbf{x}) \quad = \quad G(Q_1, \mathbf{x}) + G(Q_2, \mathbf{x}) + G(Q_3, \mathbf{x}) + G(Q_4, \mathbf{x})$$

Generating functions of polyedra

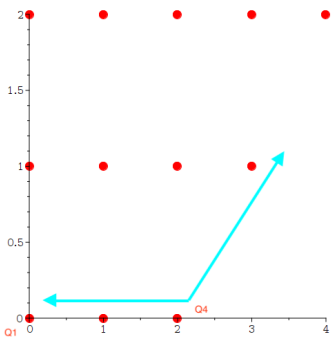
- Consider a polyhedral set $P \subseteq \mathbb{Q}^d$ and map each integer point $\mathbf{e} = (e_1, \ldots, e_d)$ of P to the monomial $\mathbf{x}^{\mathbf{e}} = {x_1}^{e_1} \cdots {x_d}^{e_d}$
- The **generating function** of P is the formal Laurent series:
$$G(P, \mathbf{x}) = \sum_{\mathbf{e} \in P \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{e}}.$$
- If P is bounded, then $G(P, (1, \ldots, 1))$ counts its integer points.
- If P is not bounded, then $G(P, \mathbf{x})$ is a formal Laurent series and can still be manipulated algorithmically.
- For $d = 2$, suppose P is the ray given by $y = 0$ and $x \geq 0$, then:
$$G(P, \mathbf{x}) = \sum_{n=0}^{n=\infty} (x, y)^{(n,0)} = \sum_{n=0}^{n=\infty} x^n y^0 = \sum_{n=0}^{n=\infty} x^n = \frac{1}{1-x}.$$
- Still for $d = 2$, $G(P, \mathbf{x})$ is computed as the sum of the generating functions of its vertex cones, thanks to Brion's theorem (1988) [2].



$$\begin{aligned} G(P, \mathbf{x}) &= G(Q_1, \mathbf{x}) + G(Q_2, \mathbf{x}) + G(Q_3, \mathbf{x}) + G(Q_4, \mathbf{x}) \\ &= \frac{1}{1-x}\frac{1}{1-y} + \frac{1}{1-x}\frac{y^2}{1-y^{-1}} + \frac{x^4 y^2}{(1-x^{-1})(1-x^{-1}y^{-1})} + \frac{x^2 y^0}{(1-xy)(1-x^{-1})} \end{aligned}$$

Generating functions of polyedra

- Consider a polyhedral set $P \subseteq \mathbb{Q}^d$ and map each integer point $\mathbf{e} = (e_1, \ldots, e_d)$ of P to the monomial $\mathbf{x}^{\mathbf{e}} = {x_1}^{e_1} \cdots {x_d}^{e_d}$
- The **generating function** of P is the formal Laurent series:
$$G(P, \mathbf{x}) = \sum_{\mathbf{e} \in P \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{e}}.$$
- If P is bounded, then $G(P, (1, \ldots, 1))$ counts its integer points.
- If P is not bounded, then $G(P, \mathbf{x})$ is a formal Laurent series and can still be manipulated algorithmically.
- For $d = 2$, suppose P is the ray given by $y = 0$ and $x \geq 0$, then:
$$G(P, \mathbf{x}) = \sum_{n=0}^{n=\infty} (x, y)^{(n,0)} = \sum_{n=0}^{n=\infty} x^n y^0 = \sum_{n=0}^{n=\infty} x^n = \frac{1}{1-x}.$$
- Still for $d = 2$, $G(P, \mathbf{x})$ is computed as the sum of the generating functions of its vertex cones, thanks to Brion's theorem (1988) [2].



$$\begin{aligned} G(P, \mathbf{x}) &= G(Q_1, \mathbf{x}) + G(Q_2, \mathbf{x}) + G(Q_3, \mathbf{x}) + G(Q_4, \mathbf{x}) \\ &= \frac{1}{1-x}\frac{1}{1-y} + \frac{1}{1-x}\frac{y^2}{1-y^{-1}} + \frac{x^4 y^2}{(1-x^{-1})(1-x^{-1}y^{-1})} + \frac{x^2 y^0}{(1-xy)(1-x^{-1})} \\ &= y^2 + xy^2 + x^2y^2 + x^3y^2 + x^4y^2 + y + xy + x^2y + \\ &\quad + x^3y + 1 + x + x^2. \end{aligned}$$

Quasi-polynomials

- The **univariate quasi-polynomial** $Q(s, m, f_0, \ldots, f_{m-1})$ is the function defined on $\mathbb{Z}$, and taking values in $\mathbb{Q}[X]$ given

Quasi-polynomials

- The **univariate quasi-polynomial** $Q(s, m, f_0, \ldots, f_{m-1})$ is the function defined on $\mathbb{Z}$, and taking values in $\mathbb{Q}[X]$ given
  - the **switch** $s \in \mathbb{Z}[X]$

Quasi-polynomials

- The **univariate quasi-polynomial** $Q(s, m, f_0, \ldots, f_{m-1})$ is the function defined on $\mathbb{Z}$, and taking values in $\mathbb{Q}[X]$ given
  - the **switch** $s \in \mathbb{Z}[X]$
  - the **modulo** m, a positive integer

Quasi-polynomials

- The **univariate quasi-polynomial** $Q(s, m, f_0, \ldots, f_{m-1})$ is the function defined on $\mathbb{Z}$, and taking values in $\mathbb{Q}[X]$ given
  - the **switch** $s \in \mathbb{Z}[X]$
  - the **modulo** m, a positive integer
  - the **constituents** m polynomials $f_0(k), \ldots, f_{m-1}(k) \in \mathbb{Q}[k]$

Quasi-polynomials

- The **univariate quasi-polynomial** $Q(s, m, f_0, \ldots, f_{m-1})$ is the function defined on $\mathbb{Z}$, and taking values in $\mathbb{Q}[X]$ given
  - the **switch** $s \in \mathbb{Z}[X]$
  - the **modulo** m, a positive integer
  - the **constituents** m polynomials $f_0(k), \ldots, f_{m-1}(k) \in \mathbb{Q}[k]$

Quasi-polynomials

- The **univariate quasi-polynomial** $Q(s, m, f_0, \ldots, f_{m-1})$ is the function defined on $\mathbb{Z}$, and taking values in $\mathbb{Q}[X]$ given
  - the **switch** $s \in \mathbb{Z}[X]$
  - the **modulo** m, a positive integer
  - the **constituents** m polynomials $f_0(k), \ldots, f_{m-1}(k) \in \mathbb{Q}[k]$

  so that, for $0 \leq i < m$, we have

$$Q(s, m, f_0, \ldots, f_{m-1})(k) \;=\; f_i \quad \Longleftrightarrow \quad s(k) \equiv i \mod m. \qquad (3.1)$$

Quasi-polynomials

- The **univariate quasi-polynomial** $Q(s, m, f_0, \ldots, f_{m-1})$ is the function defined on $\mathbb{Z}$, and taking values in $\mathbb{Q}[X]$ given
  - the **switch** $s \in \mathbb{Z}[X]$
  - the **modulo** m, a positive integer
  - the **constituents** m polynomials $f_0(k), \ldots, f_{m-1}(k) \in \mathbb{Q}[k]$

  so that, for $0 \leq i < m$, we have

$$Q(s, m, f_0, \ldots, f_{m-1})(k) \;=\; f_i \quad \Longleftrightarrow \quad s(k) \equiv i \mod m. \qquad (3.1)$$

- Propositions:
  1. For every quasi-polynomial $q_1 := Q(s, m, f_0, \ldots, f_{m-1})$, there exists a quasi-polynomial $q_2 := Q(X, m, g_1, \ldots, g_{m-1})$, such that we have $q_1 = q_2$. We denote by $\mathbb{Q}[X \mod m]$ the set of all univariate quasi-polynomials of the form $Q(X, m, f_0, \ldots, f_{m-1})$.

Quasi-polynomials

- The **univariate quasi-polynomial** $Q(s, m, f_0, \ldots, f_{m-1})$ is the function defined on $\mathbb{Z}$, and taking values in $\mathbb{Q}[X]$ given
  - the **switch** $s \in \mathbb{Z}[X]$
  - the **modulo** m, a positive integer
  - the **constituents** m polynomials $f_0(k), \ldots, f_{m-1}(k) \in \mathbb{Q}[k]$

  so that, for $0 \leq i < m$, we have

$$Q(s, m, f_0, \ldots, f_{m-1})(k) \;=\; f_i \quad \Longleftrightarrow \quad s(k) \equiv i \mod m. \tag{3.1}$$

- Propositions:
  1. For every quasi-polynomial $q_1 := Q(s, m, f_0, \ldots, f_{m-1})$, there exists a quasi-polynomial $q_2 := Q(X, m, g_1, \ldots, g_{m-1})$, such that we have $q_1 = q_2$. We denote by $\mathbb{Q}[X \mod m]$ the set of all univariate quasi-polynomials of the form $Q(X, m, f_0, \ldots, f_{m-1})$.
  2. For $q_1 := Q(X, m, f_0, \ldots, f_{m-1})$ and $q_2 := Q(X, m, g_0, \ldots, g_{m-1})$ in $\mathbb{Q}[X \mod m]$, we define the sum $q_1 + q_2$ and the product $q_1 \cdot q_2$ as

$$q_1 + q_2 = (X, m, f_0 + g_0, \ldots, f_{m-1} + g_{m-1}) \tag{3.2}$$

$$q_1 \cdot q_2 = (X, m, f_0 g_0, \ldots, f_{m-1} g_{m-1}). \tag{3.3}$$

Quasi-polynomials

- The **univariate quasi-polynomial** $Q(s, m, f_0, \ldots, f_{m-1})$ is the function defined on $\mathbb{Z}$, and taking values in $\mathbb{Q}[X]$ given
  - the **switch** $s \in \mathbb{Z}[X]$
  - the **modulo** m, a positive integer
  - the **constituents** m polynomials $f_0(k), \ldots, f_{m-1}(k) \in \mathbb{Q}[k]$

  so that, for $0 \leq i < m$, we have

$$Q(s, m, f_0, \ldots, f_{m-1})(k) \;=\; f_i \quad \Longleftrightarrow \quad s(k) \equiv i \mod m. \tag{3.1}$$

- Propositions:
  1. For every quasi-polynomial $q_1 := Q(s, m, f_0, \ldots, f_{m-1})$, there exists a quasi-polynomial $q_2 := Q(X, m, g_1, \ldots, g_{m-1})$, such that we have $q_1 = q_2$. We denote by $\mathbb{Q}[X \mod m]$ the set of all univariate quasi-polynomials of the form $Q(X, m, f_0, \ldots, f_{m-1})$.
  2. For $q_1 := Q(X, m, f_0, \ldots, f_{m-1})$ and $q_2 := Q(X, m, g_0, \ldots, g_{m-1})$ in $\mathbb{Q}[X \mod m]$, we define the sum $q_1 + q_2$ and the product $q_1 \cdot q_2$ as

$$q_1 + q_2 = (X, m, f_0 + g_0, \ldots, f_{m-1} + g_{m-1}) \tag{3.2}$$

$$q_1 \cdot q_2 = (X, m, f_0 g_0, \ldots, f_{m-1} g_{m-1}). \tag{3.3}$$

  3. Equipped with this addition and multiplication, the set $\mathbb{Q}[X \mod m]$ is a commutative ring.

Quasi-polynomials

- The **univariate quasi-polynomial** $Q(s, m, f_0, \ldots, f_{m-1})$ is the function defined on $\mathbb{Z}$, and taking values in $\mathbb{Q}[X]$ given
  - the **switch** $s \in \mathbb{Z}[X]$
  - the **modulo** m, a positive integer
  - the **constituents** m polynomials $f_0(k), \ldots, f_{m-1}(k) \in \mathbb{Q}[k]$

  so that, for $0 \leq i < m$, we have

$$Q(s, m, f_0, \ldots, f_{m-1})(k) \;=\; f_i \quad \Longleftrightarrow \quad s(k) \equiv i \mod m. \tag{3.1}$$

- Propositions:
  1. For every quasi-polynomial $q_1 := Q(s, m, f_0, \ldots, f_{m-1})$, there exists a quasi-polynomial $q_2 := Q(X, m, g_1, \ldots, g_{m-1})$, such that we have $q_1 = q_2$. We denote by $\mathbb{Q}[X \mod m]$ the set of all univariate quasi-polynomials of the form $Q(X, m, f_0, \ldots, f_{m-1})$.
  2. For $q_1 := Q(X, m, f_0, \ldots, f_{m-1})$ and $q_2 := Q(X, m, g_0, \ldots, g_{m-1})$ in $\mathbb{Q}[X \mod m]$, we define the sum $q_1 + q_2$ and the product $q_1 \cdot q_2$ as

$$q_1 + q_2 = (X, m, f_0 + g_0, \ldots, f_{m-1} + g_{m-1}) \tag{3.2}$$

$$q_1 \cdot q_2 = (X, m, f_0 g_0, \ldots, f_{m-1} g_{m-1}). \tag{3.3}$$

  3. Equipped with this addition and multiplication, the set $\mathbb{Q}[X \mod m]$ is a commutative ring.
  4. Unless $m = 1$, this ring has zero divisors and is isomorphic to the direct product of m copies of $\mathbb{Q}[X]$.

What is Integer Linear Programming?

$$\max(\textit{or}\ \min)\ \mathbf{c}^\top \mathbf{x} \quad \text{s.t.} \quad A\mathbf{x} \leq \mathbf{b}$$

where $\mathbf{x} \in \mathbb{Z}^d$, $A \in \mathbb{R}^{m \times d}$ is a matrix and $\mathbf{b} \in \mathbb{R}^m$ is a vector.

What is Integer Linear Programming?

$$\max(\textit{or}\ \min)\ \mathbf{c}^\top \mathbf{x} \quad \text{s.t.} \quad A\mathbf{x} \le \mathbf{b}$$

where $\mathbf{x} \in \mathbb{Z}^d$, $A \in \mathbb{R}^{m \times d}$ is a matrix and $\mathbf{b} \in \mathbb{R}^m$ is a vector.

$$\max_{x,y \in \mathbb{Z}} \ y$$
$$-x + y \le 1$$
$$3x + 2y \le 12$$
$$2x + 3y \le 12$$
$$x \ge 0, y \ge 0$$

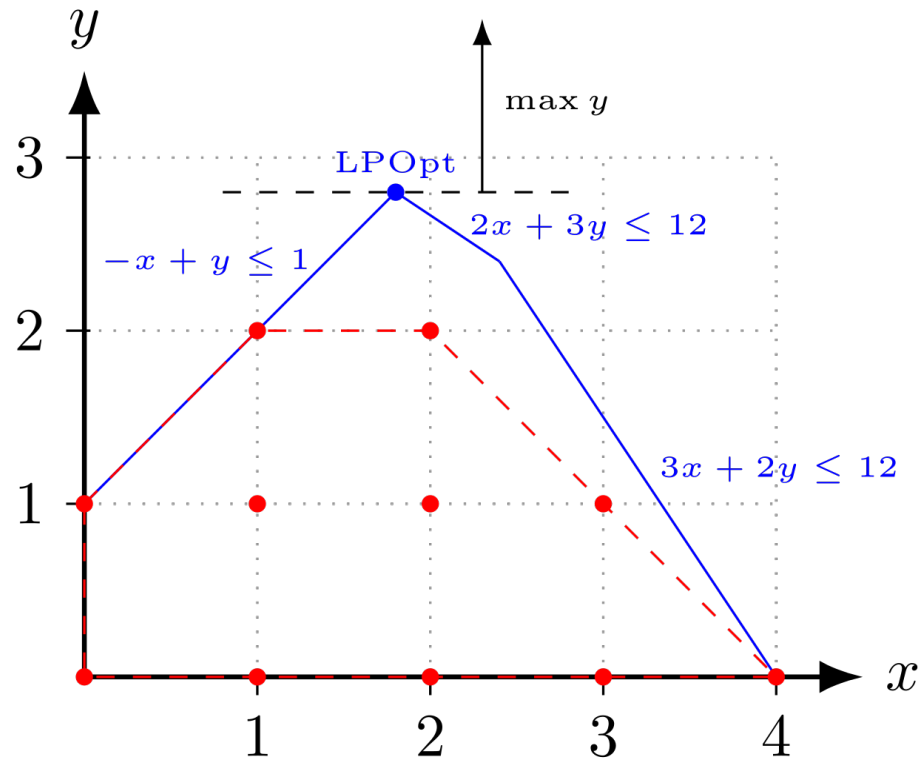


Figure: https://commons.wikimedia.org/w/index.php?curid=79967308

Parametric Integer Linear Programming (PILP)

Given a *parametric* $\mathbb{Z}$*-polyhedron*

$$\mathcal{P}(\mathbf{y}) = \{\mathbf{x} \in \mathbb{Z}^m \mid M\mathbf{x} \leq N\mathbf{y} + \mathbf{p}, \begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix} \in L\}$$

where $M \in \mathbb{Q}^{s\times m}$, $N \in \mathbb{Q}^{s\times n}$, $\mathbf{p} \in \mathbb{Q}^s$, $\mathbf{y}$ is a vector of parameters taking values in $\mathbb{Z}^n$, and $s \in \mathbb{Z}_{\geq 0}, m \in \mathbb{Z}_{>0}, n \in \mathbb{Z}_{>0}$ and L is a lattice of $\mathbb{Z}^{m+n}$.

Parametric Integer Linear Programming (PILP)

Given a *parametric* $\mathbb{Z}$*-polyhedron*

$$\mathcal{P}(\mathbf{y}) = \{\mathbf{x} \in \mathbb{Z}^m \mid M\mathbf{x} \leq N\mathbf{y} + \mathbf{p},\ \begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix} \in L\}$$

where $M \in \mathbb{Q}^{s\times m}$, $N \in \mathbb{Q}^{s\times n}$, $\mathbf{p} \in \mathbb{Q}^s$, $\mathbf{y}$ is a vector of parameters taking values in $\mathbb{Z}^n$, and $s \in \mathbb{Z}_{\geq 0}, m \in \mathbb{Z}_{>0}, n \in \mathbb{Z}_{>0}$ and L is a lattice of $\mathbb{Z}^{m+n}$.

Goal:

We want to compute two functions:

$$\begin{aligned} \text{Max}&: \mathbb{Z}^n \to \mathbb{Z} \\ &\mathbf{y} \mapsto \max\{g(\mathbf{x},\mathbf{y}) \mid \mathbf{x} \in \mathcal{P}(\mathbf{y})\} \end{aligned}$$

and

$$\begin{aligned} X_{\max}&: \mathbb{Z}^n \to 2^{\mathbb{Z}^n} \\ &\mathbf{y} \mapsto \{\mathbf{x} \mid g(\mathbf{x},\mathbf{y}) = \text{Max}(\mathbf{y}),\ \mathbf{x} \in \mathcal{P}(\mathbf{y})\} \end{aligned}$$

Reduction to *IntegerPointDecomposition* (*IPD*)

Key idea: introduce an auxiliary variable z and consider

$$F(z, \mathbf{y}) \equiv \exists\, \mathbf{x} \in \mathbb{Z}^m : z = g(\mathbf{x}, \mathbf{y}) \wedge \mathbf{x} \in \mathcal{P}(\mathbf{y})$$

Then $\mathrm{Max}(\mathbf{y}) = \max\{z \in \mathbb{Z} \mid F(z, \mathbf{y})\}$.

Reduction to *IntegerPointDecomposition* (*IPD*)

Key idea: introduce an auxiliary variable z and consider

$$F(z, \mathbf{y}) \equiv \exists\, \mathbf{x} \in \mathbb{Z}^m : z = g(\mathbf{x}, \mathbf{y}) \wedge \mathbf{x} \in \mathcal{P}(\mathbf{y})$$

Then $\mathrm{Max}(\mathbf{y}) = \max\{z \in \mathbb{Z} \mid F(z, \mathbf{y})\}$.

Algorithm outline:

1. Apply *IPD* to $\begin{pmatrix} \mathbf{x} \\ z \\ \mathbf{y} \end{pmatrix} \in \bigvee_{i=1}^{e} \mathbb{Z}\mathrm{Polyhedron}(P_i, L_i)$ with variable ordering $x_1 > \cdots > x_m > z > y_1 > \cdots > y_n$
2. Obtain normalized $\mathbb{Z}\mathrm{Polyhedron}(P_i, L_i)$ such that each P_i has *at most one upper bound for* z
3. For each $\mathbb{Z}\mathrm{Polyhedron}(P_i, L_i)$, extract $\mathrm{Max}(\mathbf{y})$ as a piecewise quasi-polynomial

Reducing to One Upper Bound for z

After IPD, each normalized $\mathbb{Z}$Polyhedron(P_i, L_i) may have *several* upper bounds for z: $b_1 z \leq A_1(\mathbf{y})$ and $b_2 z \leq A_2(\mathbf{y})$.

Reducing to One Upper Bound for z

After IPD, each normalized $\mathbb{Z}$Polyhedron(P_i, L_i) may have *several* upper bounds for z: $b_1 z \le A_1(\mathbf{y})$ and $b_2 z \le A_2(\mathbf{y})$.

The idea: Let $l = \text{lcm}(b_1, b_2)$, $c_1 = l/b_1$, $c_2 = l/b_2$. The two upper bounds together say:

$$lz \le c_1 A_1(\mathbf{y}) \quad \text{and} \quad lz \le c_2 A_2(\mathbf{y})$$

This is equivalent to a *case split* on $\mathbf{y}$:

- **Region Δ_{12}:** $c_1 A_1(\mathbf{y}) \le c_2 A_2(\mathbf{y})$ where only the bound $b_1 z \le A_1(\mathbf{y})$ is active
- **Region Δ_{21}:** $c_2 A_2(\mathbf{y}) \le c_1 A_1(\mathbf{y})$ where only the bound $b_2 z \le A_2(\mathbf{y})$ is active

Reducing to One Upper Bound for z

After IPD, each normalized $\mathbb{Z}$Polyhedron(P_i, L_i) may have *several* upper bounds for z: $b_1 z \leq A_1(\mathbf{y})$ and $b_2 z \leq A_2(\mathbf{y})$.

The idea: Let $l = \text{lcm}(b_1, b_2)$, $c_1 = l/b_1$, $c_2 = l/b_2$. The two upper bounds together say:

$$lz \leq c_1 A_1(\mathbf{y}) \quad \text{and} \quad lz \leq c_2 A_2(\mathbf{y})$$

This is equivalent to a *case split* on $\mathbf{y}$:

- **Region Δ_{12}:** $c_1 A_1(\mathbf{y}) \leq c_2 A_2(\mathbf{y})$ where only the bound $b_1 z \leq A_1(\mathbf{y})$ is active
- **Region Δ_{21}:** $c_2 A_2(\mathbf{y}) \leq c_1 A_1(\mathbf{y})$ where only the bound $b_2 z \leq A_2(\mathbf{y})$ is active

On each region, IPD is called again to produce normalized $\mathbb{Z}$-polyhedra. This time with *at most one upper bound* for z.

Reducing to One Upper Bound for z

After IPD, each normalized $\mathbb{Z}$Polyhedron(P_i, L_i) may have *several* upper bounds for z: $b_1 z \leq A_1(\mathbf{y})$ and $b_2 z \leq A_2(\mathbf{y})$.

The idea: Let $l = \text{lcm}(b_1, b_2)$, $c_1 = l/b_1$, $c_2 = l/b_2$. The two upper bounds together say:

$$lz \leq c_1 A_1(\mathbf{y}) \quad \text{and} \quad lz \leq c_2 A_2(\mathbf{y})$$

This is equivalent to a *case split* on $\mathbf{y}$:

- **Region Δ_{12}:** $c_1 A_1(\mathbf{y}) \leq c_2 A_2(\mathbf{y})$ where only the bound $b_1 z \leq A_1(\mathbf{y})$ is active
- **Region Δ_{21}:** $c_2 A_2(\mathbf{y}) \leq c_1 A_1(\mathbf{y})$ where only the bound $b_2 z \leq A_2(\mathbf{y})$ is active

On each region, IPD is called again to produce normalized $\mathbb{Z}$-polyhedra. This time with *at most one upper bound* for z.

The process repeats until all pieces have at most one upper bound for z.

Extracting Max from Each Piece

Once every $\mathbb{Z}\text{Polyhedron}(P_i, L_i)$ has at most one upper bound for z, reading off $\text{Max}(\mathbf{y})$ is straightforward.

Extracting Max from Each Piece

Once every $\mathbb{Z}\text{Polyhedron}(P_i, L_i)$ has at most one upper bound for z, reading off $\text{Max}(\mathbf{y})$ is straightforward.

The idea: Because the $\mathbb{Z}$Polyhedron is *normalized*, the lattice L_i expresses z as a linear combination of a free integer parameter t and the $\mathbf{y}$-variables:

$$z = q_i \cdot t + f_i(\mathbf{y}), \quad t \in \mathbb{Z}, \quad q_i \in \mathbb{Q}$$

To maximize z, we simply maximize t, which is bounded above by the single upper bound $b_i z \leq A_i(\mathbf{y})$:

$$t \leq \left\lfloor \frac{A_i(\mathbf{y}) - b_i f_i(\mathbf{y})}{b_i q_i} \right\rfloor$$

Plugging the maximum t back in gives

$$\text{Max}(\mathbf{y}) = q_i \left\lfloor \frac{A_i(\mathbf{y}) - b_i f_i(\mathbf{y})}{b_i q_i} \right\rfloor + f_i(\mathbf{y})$$

a *piecewise quasi-polynomial* in $\mathbf{y}$, valid over the region of P_i.

Extracting Max from Each Piece

Once every $\mathbb{Z}\text{Polyhedron}(P_i, L_i)$ has at most one upper bound for z, reading off $\text{Max}(\mathbf{y})$ is straightforward.

The idea: Because the $\mathbb{Z}$Polyhedron is *normalized*, the lattice L_i expresses z as a linear combination of a free integer parameter t and the $\mathbf{y}$-variables:

$$z = q_i \cdot t + f_i(\mathbf{y}), \quad t \in \mathbb{Z}, \quad q_i \in \mathbb{Q}$$

To maximize z, we simply maximize t, which is bounded above by the single upper bound $b_i z \leq A_i(\mathbf{y})$:

$$t \leq \left\lfloor \frac{A_i(\mathbf{y}) - b_i f_i(\mathbf{y})}{b_i q_i} \right\rfloor$$

Plugging the maximum t back in gives

$$\text{Max}(\mathbf{y}) = q_i \left\lfloor \frac{A_i(\mathbf{y}) - b_i f_i(\mathbf{y})}{b_i q_i} \right\rfloor + f_i(\mathbf{y})$$

a *piecewise quasi-polynomial* in $\mathbf{y}$, valid over the region of P_i.

Edge cases: If no upper bound exists on z, then $\text{Max}(\mathbf{y}) = +\infty$. If $q_i = 0$, then z is constant and $\text{Max}(\mathbf{y}) = f_i(\mathbf{y})$ directly.

Two Strategies for Finding the Optimizer

Given Max $= z^*$, they differ in *how* the optimizer $(x_1^*, \ldots, x_m^*)$ is recovered.

Sample Point Method

1. Set $z = z^*$
2. **Pick a concrete integer point** from the resulting feasible slice.
3. Return that point as $(x_1^*, \ldots, x_m^*)$.

Key idea: existence of a solution is guaranteed by normalization; any witness suffices.
Drawback: may require solving an auxiliary ILP to find the witness, which can be expensive on hard instances.

Locus Method

1. Set $z = z^*$
2. **Back-substitute** through the polyhedron structure to express each x_i^* as a closed-form function of the parameters $p_0, p_1, \ldots$
3. Return the full parametric optimizer locus.

Key idea: normalization of IPD output enables direct symbolic back-substitution without search.
Advantage: avoids auxiliary ILP; slightly faster in practice.

The Three PILP Implementations

For each parametric $\mathbb{Z}$-polyhedron $\mathcal{P}(\mathbf{y})$, we solve both $\min_{\mathbf{x}\in\mathcal{P}(\mathbf{y})} g(\mathbf{x})$ and $\max_{\mathbf{x}\in\mathcal{P}(\mathbf{y})} g(\mathbf{x})$, where $g(\mathbf{x}) = x_1 + \cdots + x_m$.

The Three PILP Implementations

For each parametric $\mathbb{Z}$-polyhedron $\mathcal{P}(\mathbf{y})$, we solve both $\min_{\mathbf{x}\in\mathcal{P}(\mathbf{y})} g(\mathbf{x})$ and $\max_{\mathbf{x}\in\mathcal{P}(\mathbf{y})} g(\mathbf{x})$, where $g(\mathbf{x}) = x_1 + \cdots + x_m$.

1. **islpip by Feautrier [3, 8]**
   - Lexicographic by default; introduce $z = g(\mathbf{x})$ and place z *first* to optimize a general objective
   - Output: each line gives [params] $\rightarrow$ variable values | parameter constraints

The Three PILP Implementations

For each parametric $\mathbb{Z}$-polyhedron $\mathcal{P}(\mathbf{y})$, we solve both $\min_{\mathbf{x}\in\mathcal{P}(\mathbf{y})} g(\mathbf{x})$ and $\max_{\mathbf{x}\in\mathcal{P}(\mathbf{y})} g(\mathbf{x})$, where $g(\mathbf{x}) = x_1 + \cdots + x_m$.

1. **islpip by Feautrier [3, 8]**
   - Lexicographic by default; introduce $z = g(\mathbf{x})$ and place z *first* to optimize a general objective
   - Output: each line gives [params] $\rightarrow$ variable values | parameter constraints
2. **lexmin by Verdoolaege / Barvinok [1, 8]**
   - Based on rational generating functions, not parametric simplex
   - Computes only lexicographic *minima*; to maximize $g(\mathbf{x})$, introduce $z = -g(\mathbf{x})$, minimize, then negate

The Three PILP Implementations

For each parametric $\mathbb{Z}$-polyhedron $\mathcal{P}(\mathbf{y})$, we solve both $\min_{\mathbf{x}\in\mathcal{P}(\mathbf{y})} g(\mathbf{x})$ and $\max_{\mathbf{x}\in\mathcal{P}(\mathbf{y})} g(\mathbf{x})$, where $g(\mathbf{x}) = x_1 + \cdots + x_m$.

1. **islpip by Feautrier [3, 8]**
   - Lexicographic by default; introduce $z = g(\mathbf{x})$ and place z *first* to optimize a general objective
   - Output: each line gives [params] $\rightarrow$ variable values | parameter constraints
2. **lexmin by Verdoolaege / Barvinok [1, 8]**
   - Based on rational generating functions, not parametric simplex
   - Computes only lexicographic *minima*; to maximize $g(\mathbf{x})$, introduce $z = -g(\mathbf{x})$, minimize, then negate
3. **IntegerPointDecomposition [6]**
   - Run *IPD* once with $z = g(\mathbf{x})$ introduced
   - *Both* min and max extracted from the same $\mathbb{Z}$-polyhedron decomposition
   - Implemented in `Maple` *QuantifierEliminationOverZ* [4, 5]

Test	Feautrier (C)	Barvinok (C)	Sample Point (Maple)	Locus (Maple)
boulet	0.032	0.021	2.514	2.493
bouleti	0.039	error	13.069	12.795
cg1	0.024	0.027	1.205	1.167
cnt_sum2	1.114	not supported	>30 mins	>30 mins
difficult	0.079	0.555	1136.988	1133.288
esced	0.051	0.026	394.609	390.996
ex	0.027	0.030	0.900	0.900
ex2	0.027	0.030	0.907	0.900
fimmel	0.038	0.038	0.644	0.640
jcomplex	583.280	not supported	>30 mins	>30 mins
max	0.026	0.030	1.437	1.397
negative	0.021	0.028	0.521	0.521
phideo	0.764	>30 mins	>30 mins	>30 mins
seghir–e1	0.068	error	1278.216	1265.387
seghir–e3	0.031	0.067	73.585	73.353
seghir–e4	0.045	11.675	>30 mins	>30 mins
seghir–e5	0.147	2.533	>30 mins	>30 mins
seghir–e6	0.045	0.366	447.975	445.734
seghir–e7	0.029	0.111	55.668	55.795
seghir–e8	0.031	0.075	925.595	903.961
seghir–e9	0.332	>30 mins	>30 mins	>30 mins
small	0.018	0.026	1.209 (ILP)	1.200 (ILP)
sor1d	0.024	0.022	0.455	0.453
square	0.027	0.029	0.916	0.958
sven	0.024	0.019	0.228	0.278
tobi	0.026	0.025	45.504	45.355

Array Bound Checking: Setup & F_2 Analysis

Input Setup

Triply-Nested Loop

for $(i_1 = 0;\ i_1 \le R_1;\ i_1{+}{+})$
  for $(i_2 = 0;\ i_2 \le R_2 - i_1;\ i_2{+}{+})$
    for $(i_3 = 0;\ i_3 \le R_3 - i_1;\ i_3{+}{+})$

$\tilde{A}[i_1 + i_2 + 1][i_1 + 2i_2 + 1][2i_1 + i_3] \leftarrow \ldots$

- **Domain ($\mathcal{D}$):** $\triangle$-prism
- $(i_1, i_2, i_3) \in \mathbb{Z}^3$
- $0 \le i_1 \le R_1$
- $0 \le i_2 \le R_2 - i_1$
- $0 \le i_3 \le R_3 - i_1$
- **Reference Functions:**

$$f_2(i_1, i_2, i_3) = i_1 + 2i_2 + 1$$
$$f_3(i_1, i_2, i_3) = 2i_1 + i_3$$

Formulation & Solving

Safety Condition via PILP

$$\underbrace{\max_{(i_1,i_2,i_3)\in\mathcal{D}} f_2}_{F_2} \le M_2 \wedge \underbrace{\max_{(i_1,i_2,i_3)\in\mathcal{D}} f_3}_{F_3} \le M_3$$

Exact Piecewise Maxima (F_2)

$$F_2 = \begin{cases} 2R_2 + 1 & \text{if } R_2 \ge 1 \\ 1 & \text{if } R_1 = 0, R_2 = 0 \end{cases}$$

- Max achieved at $i_1 = 0$ (due to $i_2 \le R_2 - i_1$).
- **Naive cuboid:** $R_1 + 2R_2 + 1$ (over-conservative).

F_3 Analysis & Final Safety Checks

Input Setup

Triply-Nested Loop

for $(i_1 = 0;\ i_1 \le R_1;\ i_1{+}{+})$
  for $(i_2 = 0;\ i_2 \le R_2 - i_1;\ i_2{+}{+})$
    for $(i_3 = 0;\ i_3 \le R_3 - i_1;\ i_3{+}{+})$

$\tilde{A}[i_1 + i_2 + 1][i_1 + 2i_2 + 1][2i_1 + i_3] \leftarrow \dots$

- **Domain ($\mathcal{D}$):** $\triangle$-prism
- $(i_1, i_2, i_3) \in \mathbb{Z}^3$
- $0 \le i_1 \le R_1$
- $0 \le i_2 \le R_2 - i_1$
- $0 \le i_3 \le R_3 - i_1$
- **Reference Functions:**

$$f_2(i_1, i_2, i_3) = i_1 + 2i_2 + 1$$
$$f_3(i_1, i_2, i_3) = 2i_1 + i_3$$

Key Trade-off

Increasing i_1 increases the $2i_1$ term but reduces the available range for i_3. Optimal boundary depends on which constraint binds.

Formulation & Solving

Exact Piecewise Maxima (F_3)

$$F_3 = \begin{cases} R_1 + R_3 & \text{if } R_1 \le R_3 \\ 2R_3 & \text{if } R_1 > R_3 \end{cases}$$

Final Runtime Conditions

$$2R_2 + 1 \le M_2$$
and
$$\begin{cases} R_1 + R_3 \le M_3 & \text{if } R_1 \le R_3 \\ 2R_3 \le M_3 & \text{if } R_1 > R_3 \end{cases}$$

Concluding remarks

Summary and notes

1. We have presented software libraries for computing integer hulls and $\mathbb{Z}$-polyhedra.
2. They support integer point counting of parametric polyhedra (= computing Ehrhart polynomials) as well as Presburger arithmetic (= quantifier elimination over the integers).
3. One can replace this QE problems with a number of problem instances from parametric integer linear programming (PILP).
4. The algorithm IntegerPointDecomposition plays an essential role.
5. Most of these functionalities are available in Maple 2025, except QE and PILP.

References

[1] A. Barvinok and K. Woods. "Short Rational Generating Functions for Lattice Point Problems". In: Journal of the American Mathematical Society 16.4 (2003), pp. 957–979.

[2] M. Brion. "Points entiers dans les polyedres convexes". In: Annales scientifiques de l'École normale supérieure. Vol. 21. 4. 1988, pp. 653–663.

[3] P. Feautrier. "Parametric integer programming". In: RAIRO. Recherche opérationnelle 22.3 (1988), pp. 243–268. URL: http://www.numdam.org/item?id=RO_1988__22_3_243_0.

[4] R.-J. Jing, Y. Lei, C. F. S. Maligec, M. M. Maza, and C. Mukherjee. "Integer Hulls, Z-Polyhedra and Presburger Arithmetic in Action". In: ACM Commun. Comput. Algebra 59.3 (Jan. 2026), pp. 41–46. ISSN: 1932-2232. DOI: 10.1145/3787957.3787958. URL: https://doi.org/10.1145/3787957.3787958.

[5] R.-J. Jing, Y. Lei, C. F. S. Maligec, M. Moreno Maza, and C. Mukherjee. “Quantifier Elimination Over the Integers”. In: ISSAC ’25 (2025), pp. 353–362. DOI: 10.1145/3747199.3747580. URL: https://doi.org/10.1145/3747199.3747580.

[6] R.-J. Jing, Y. Lei, M. Moreno Maza, and C. Mukherjee. “Quantifier Elimination Over the Integers”. In: To appear. DOI: 10.2139/ssrn.6178290. URL: https://ssrn.com/abstract=6178290.

[7] R. Jing and M. Moreno Maza. “Computing the Integer Points of a Polyhedron, I: Algorithm”. In: CASC 2017, Proceedings. Vol. 10490. LNCS. Springer, 2017, pp. 225–241.

[8] S. Verdoolaege. “isl: An Integer Set Library for the Polyhedral Model”. In: Lecture Notes in Computer Science 6327 (2010). Ed. by K. Fukuda, J. van der Hoeven, M. Joswig, and N. Takayama, pp. 299–302.