Parametric Integer Linear Programming via Presburger Arithmetic

Chirantan Mukherjee *joint with R. J. Jing, Y. Lei and M. Moreno Maza*

Western Science

ORCCA
Ontario Research Centre for Computer Algebra

Plan

Plan

Step 1: Delinearization Problem

A loop accesses a **linearized array** `A[R]` whose true structure is a multi-dimensional array `B`$[x_1]$`[`x_2`]`. Is every access **within bounds**?

The Loop (`ex2.pip`)

`for` $x_1 = 0$ `to` p_2
  for $x_2 = 0$ to p_1
    `A`$[x_1 + x_2]$ `=` ...

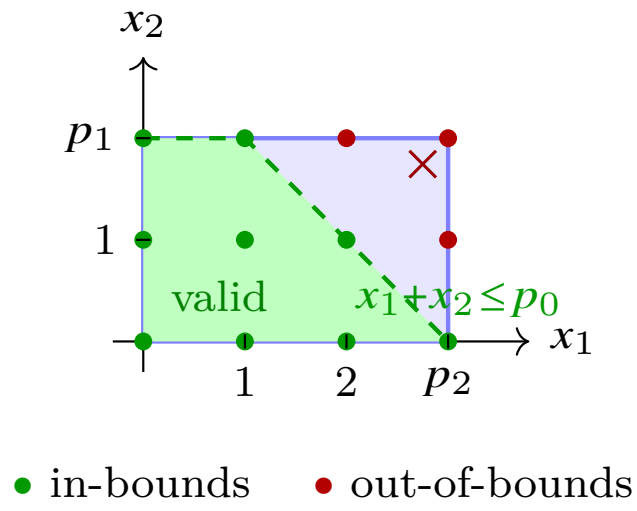
Delinearized:

`A[R]` $\mapsto$ `B`$[x_1][x_2]$

Delinearization map:

$$f_1(\mathbf{x}) = x_1, \qquad f_2(\mathbf{x}) = x_2$$

`B` is $(p_2{+}1) \times (p_1{+}1)$,
$p_0 \le p_1 \le p_2$.



Step 1: Delinearization Problem

A loop accesses a **linearized array** `A[R]` whose true structure is a multi-dimensional array `B[`x_1`][`x_2`]`. Is every access **within bounds**?

The Loop (`ex2.pip`)

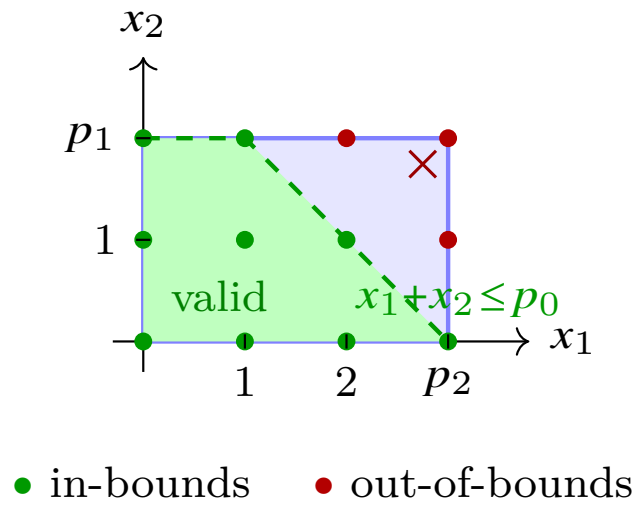
`for` $x_1 = 0$ `to` p_2
  for $x_2 = 0$ to p_1
    `A[`$x_1 + x_2$`] = ...`

Delinearized:

`A[R]` $\mapsto$ `B[`x_1`][`x_2`]`

Delinearization map:

$$f_1(\mathbf{x}) = x_1, \qquad f_2(\mathbf{x}) = x_2$$



`B` is $(p_2{+}1) \times (p_1{+}1)$,
$p_0 \le p_1 \le p_2$.

Validity question: for *all* (x_1, x_2) in iteration domain $\mathcal{D}$:

$$0 \;\le\; f_k(\mathbf{x}) \;\le\; M_k - 1 \qquad (k = 1, 2)$$

Since p_0, p_1, p_2 are runtime parameters $\Rightarrow$ this is a **parametric** problem.

Step 2: Array Validity as a QE Problem

With fixed coefficients, each f_k is a concrete linear form. Validity becomes the **quantified formula**:

$$\Phi(\mathbf{p}) \;:=\; \forall (x_1, x_2) \in \mathcal{D}: 0 \le x_1 \le p_2 \;\wedge\; 0 \le x_2 \le p_1 \;\wedge\; x_1 + x_2 \le p_0$$

This is a **Quantifier Elimination (QE) over** $\mathbb{Z}$ problem: eliminate x_1, x_2 to find for which parameters $\mathbf{p}$ every access is safe.

Step 2: Array Validity as a QE Problem

With fixed coefficients, each f_k is a concrete linear form. Validity becomes the **quantified formula**:

$$\Phi(\mathbf{p}) \;:=\; \forall (x_1, x_2) \in \mathcal{D}\colon 0 \le x_1 \le p_2 \;\wedge\; 0 \le x_2 \le p_1 \;\wedge\; x_1 + x_2 \le p_0$$

This is a **Quantifier Elimination (QE) over** $\mathbb{Z}$ problem: eliminate x_1, x_2 to find for which parameters $\mathbf{p}$ every access is safe.

Equivalence to min/max: $\Phi(\mathbf{p})$ holds if and only if, for each k:

$$\min_{\mathbf{x} \in \mathcal{D}} f_k(\mathbf{x}) \;\ge\; 0 \qquad \text{and} \qquad \max_{\mathbf{x} \in \mathcal{D}} f_k(\mathbf{x}) \;\le\; M_k - 1$$

Why?

$\forall\, \mathbf{x} \in \mathcal{D}\colon\ f_k \ge 0 \iff \min_{\mathcal{D}} f_k \ge 0$

$\forall\, \mathbf{x} \in \mathcal{D}\colon\ f_k \le M_k - 1 \iff \max_{\mathcal{D}} f_k \le M_k - 1$

Key point:

Loop bounds p_0, p_1, p_2 are known *only at runtime*.

Computing min/max f_k over $\mathcal{D}$ with **parametric** bounds is exactly a **PILP problem**.

Step 3: From QE to PILP (`ex2.pip`)

PILP instance. max $g = x_1 + x_2$ subject to:

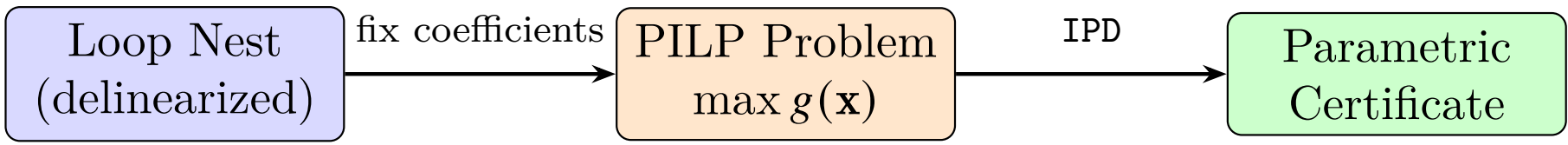
$$0 \le p_0 \le p_1 \le p_2, \qquad \begin{cases} 0 \le p_0 \le p_1 \le p_2 \\ 0 \le x_2 \le p_1 \\ 0 \le x_1 \le p_2 \\ 0 \le x_1 + x_2 \le p_0 \end{cases} \qquad x_1, x_2 \in \mathbb{Z}$$

PILP answer (parametric, all cases covered at compile time):

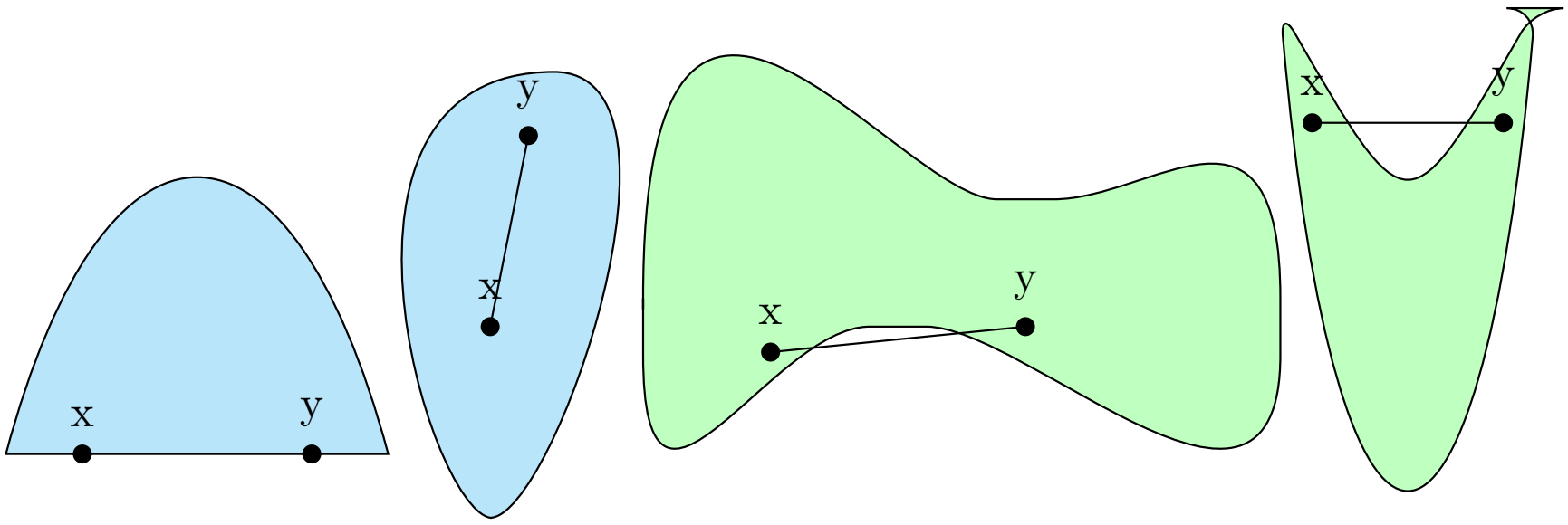
$$z_{\max} = p_1 + p_2, \qquad (x_1^*,\, x_2^*) = (p_2,\ p_1)$$

$$z_{\min} = p_0, \qquad (x_1^*,\, x_2^*) = (p_2,\ p_0 - p_2)$$

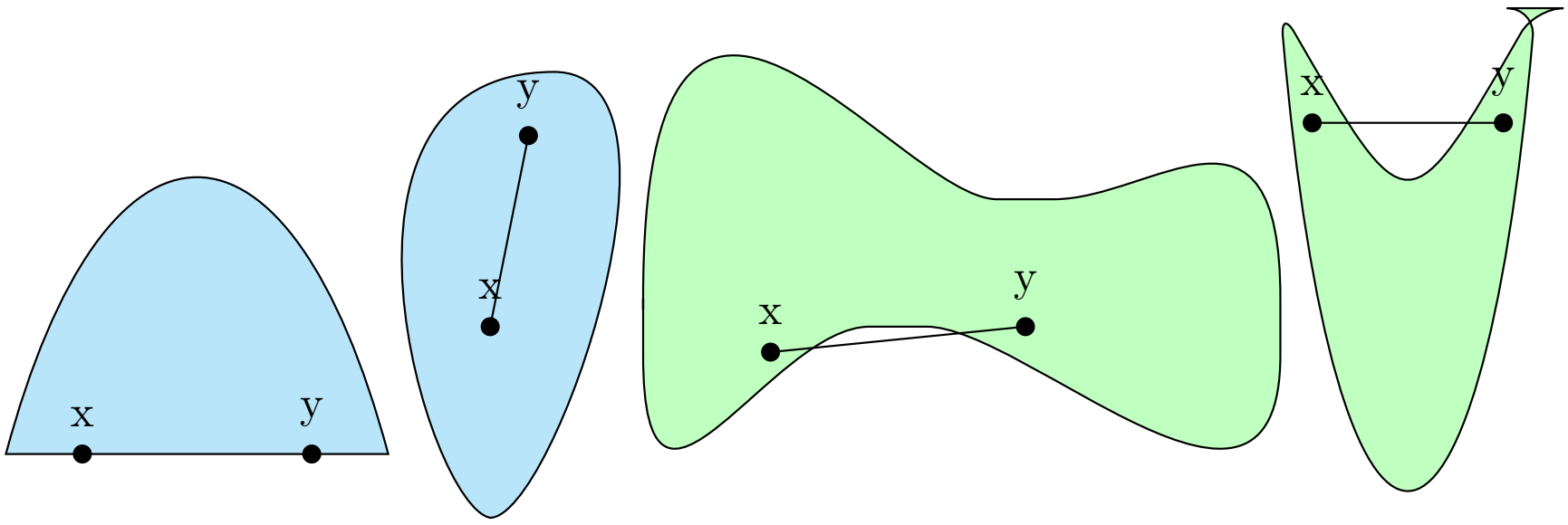
One compile-time run covers all runtime values of p_0, p_1, p_2. The minimum involves p_0 (the upper bound on $x_1 + x_2$), while the maximum is determined by p_1, p_2 alone.



Plan



Convex Convex Non-convex Non-convex



Convex Convex Non-convex Non-convex

A set S is called *convex* if the line joining any two points in S is in S, i.e.,

$$\forall x, y \in S, \forall \lambda \in [0,1], \lambda x + (1-\lambda) y \in S.$$

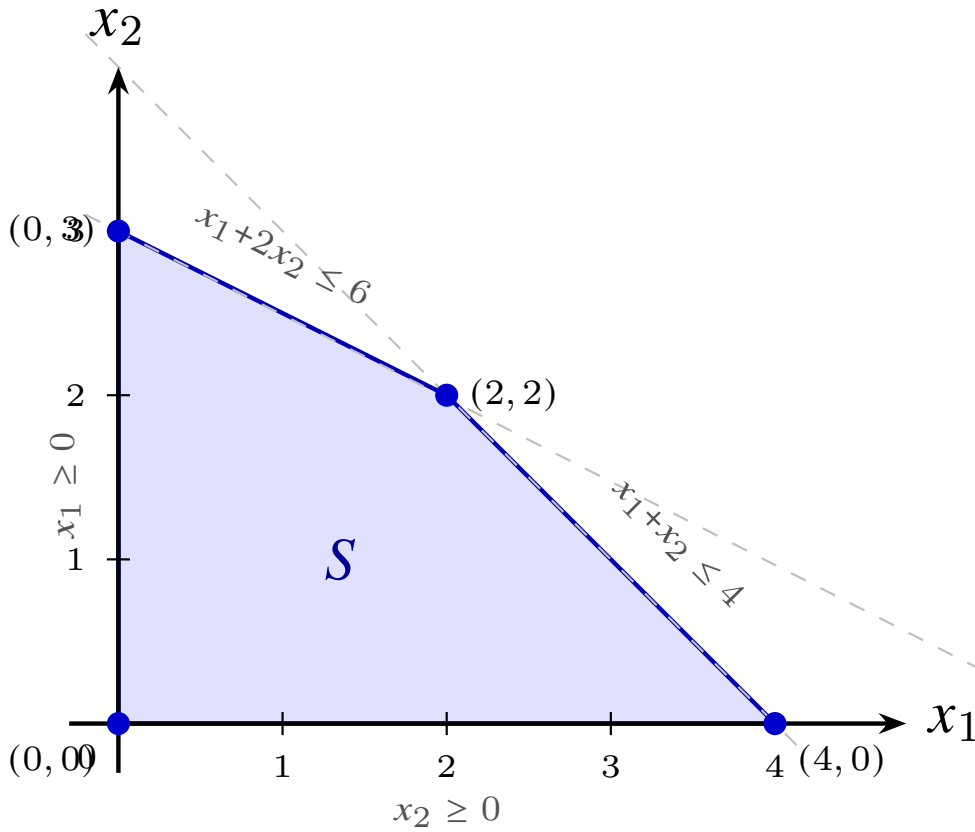
What is a Polyhedral Set?

A (convex) *polyhedral set* S is a set $\{\mathbf{x} \in \mathbb{R}^d \mid A\mathbf{x} \leq \mathbf{b}\}$, where $A \in \mathbb{R}^{m \times d}$ is a matrix and $\mathbf{b} \in \mathbb{R}^m$ is a vector.

Example ($d = 2$, $m = 4$)

$$A = \begin{pmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 0 \\ 0 \\ 4 \\ 6 \end{pmatrix}$$

$$\begin{aligned} x_1 &\geq 0 \\ x_2 &\geq 0 \\ x_1 + x_2 &\leq 4 \\ x_1 + 2x_2 &\leq 6 \end{aligned}$$



What is Linear Programming?

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

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

- $\mathbf{x}$ is called the *decision variable*
- $\mathbf{c}^\top \mathbf{x}$ is called the *objective function*
- $A\mathbf{x} \leq \mathbf{b}$ is the *feasible domain*
  - It forms a polyhedral set.

What is Integer Linear Programming?

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

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.

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

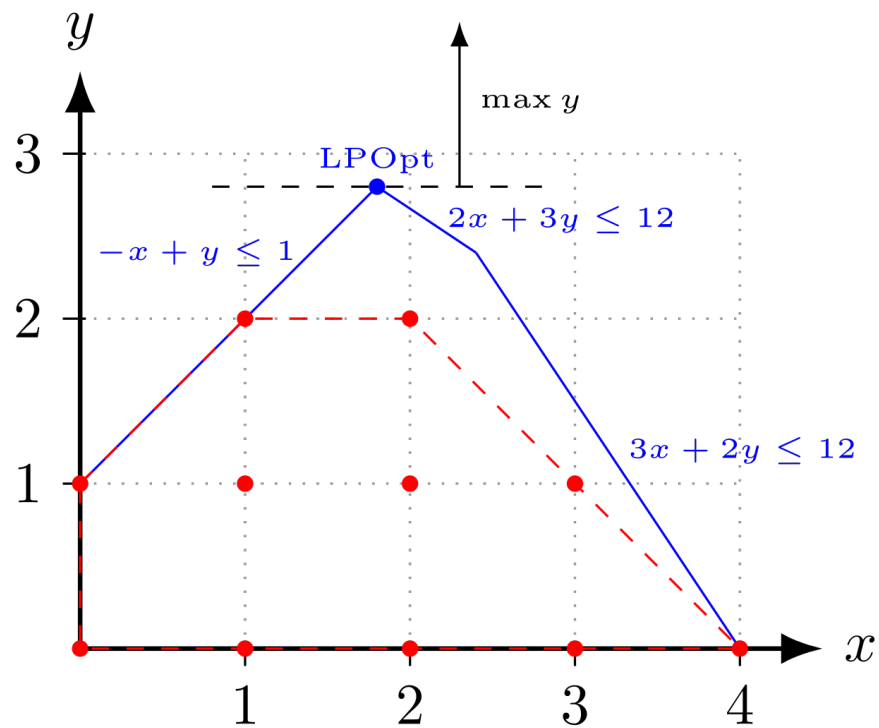


Figure: https://commons.wikimedia.org/w/index.php?curid=79967308

A subset $L \subseteq \mathbb{Z}^d$ is called an *integer lattice* (or *lattice*) if

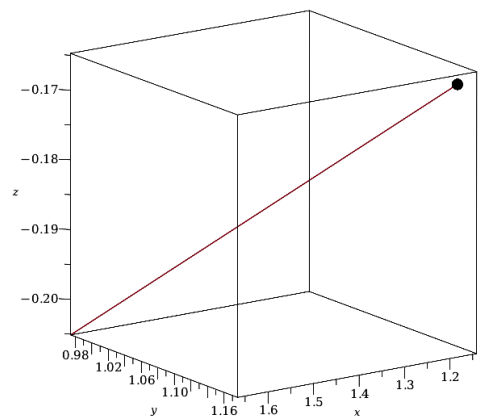
$$L = \{\mathbf{x} \in \mathbb{Z}^d \mid (\exists\, \mathbf{t} \in \mathbb{Z}^c)\ \mathbf{x} = A\mathbf{t} + \mathbf{b}\}$$

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 $\mathbb{Z}$Polyhedron(P, L).

Input:

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



Output:

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

Normalized $\mathbb{Z}$-Polyhedral set

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

1. it is non-empty, and P is given by a system of linear inequalities of the form

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

2. a_i (resp. b_i) is either $-\infty$ (resp. $+\infty$) or an expression of the form $\max(l_{i,1} \dots l_{i,e_i})$ (resp. $\min(l_{i,1} \dots l_{i,e_i})$), and
3. each $l_{i,j} \in \mathbb{Q}[x_1, \dots, x_{i-1}]$ with degree at most 1, so that
4. 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.

The algorithm IntegerPointDecomposition [3] decomposes any $\mathbb{Z}$-polyhedron into *normalized* $\mathbb{Z}$-polyhedra.

Example of a Normalized $\mathbb{Z}$-Polyhedron

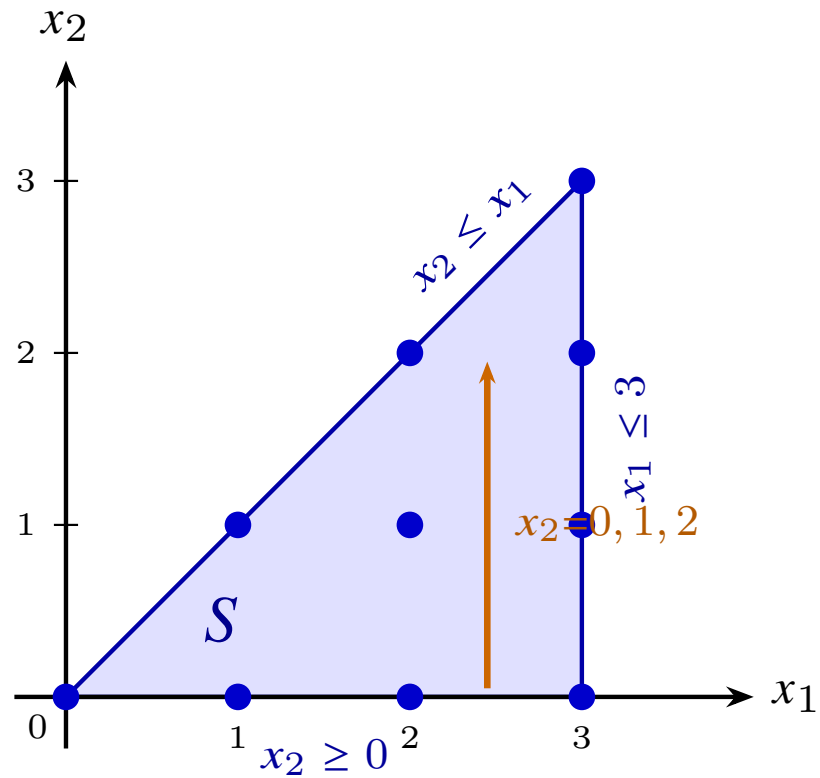
Example. Let $d = 2$, $L = \mathbb{Z}^2$ (full lattice), and

$$P = \left\{ (x_1, x_2) \in \mathbb{Q}^2 \;\middle|\; 0 \le x_1 \le 3, \quad 0 \le x_2 \le x_1 \right\}.$$

Normalized form:

$$\underbrace{0}_{a_0} \le x_1 \le \underbrace{3}_{b_0}, \quad \underbrace{0}_{a_1} \le x_2 \le \underbrace{x_1}_{b_1(x_1)}$$

- $a_0 = 0$, $b_0 = 3$ are constants
- $a_1 = 0$, $b_1 = x_1 \in \mathbb{Q}[x_1]$



Back-substitution:

1. Specialize $x_1 \in \{0, 1, 2, 3\}$
2. For each x_1, specialize $x_2 \in \{0, 1, \ldots, x_1\}$

Integer points (10 total):

$$\left\{ (x_1, x_2) \in \mathbb{Z}^2 \;\middle|\; 0 \le x_2 \le x_1 \le 3 \right\}$$

Plan

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

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 $\mathbb{Z}\text{Polyhedron}(P_i, L_i)$ 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 $\mathbb{Z}\text{Polyhedron}(P_i, L_i)$ 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:

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

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 $\mathbb{Z}\text{Polyhedron}(P_i, L_i)$ 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 $\mathbb{Z}$Polyhedron(P_i, L_i) has at most one upper bound for z, reading off 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 Max$(\mathbf{y}) = +\infty$. If $q_i = 0$, then z is constant and Max$(\mathbf{y}) = f_i(\mathbf{y})$ directly.

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** [2, 7]
   - 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, 7]
   - 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]

Example: `ex2.pip` Problem Setup

Variables: $x_1, x_2 \in \mathbb{Z}$, **Parameters:** $p_0, p_1, p_2 \in \mathbb{Z}$,
Objective: $g(\mathbf{x}) = x_1 + x_2$

Constraints:

$$0 \le p_0 \le p_1 \le p_2, \quad 0 \le x_2 \le p_1, \quad 0 \le x_1 \le p_2, \quad 0 \le x_1 + x_2 \le p_0$$

Introduce auxiliary variable z with $z = x_1 + x_2$, and place z *first* in the lexicographic ordering so that the first output coordinate gives the optimal objective value.

IPD produces a single $\mathbb{Z}\text{Polyhedron}(P, L)$ where:

$$P := \begin{cases} x_1 = z - x_2 \\ x_2 - p_1 \le 0 \\ -z + p_0 \le 0 \\ p_0 - p_1 - p_2 \le 0 \\ z - p_2 - p_1 \le 0 \\ z - x_2 - p_2 \le 0 \end{cases} \qquad L := \left(\begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \right)$$

Example: `ex2.pip` Bounds on z and Optimizer

From P, we read the bounds on z:

$$p_0 \;\le\; z \;\le\; p_1 + p_2$$

Maximum

Upper bound $z \le p_1 + p_2$ gives
$z_{\max} = p_1 + p_2$.
From $z - x_2 - p_2 \le 0$:

$$p_1 + p_2 - x_2 - p_2 \le 0 \;\Rightarrow\; x_2 \ge p_1$$

Together with $x_2 \le p_1$: $x_2^* = p_1$.
From $x_1 = z - x_2$:

$$x_1^* = (p_1 + p_2) - p_1 = p_2$$

$$\boxed{z_{\max} = p_1 + p_2, \quad (x_1^*, x_2^*) = (p_2,\, p_1)}$$

Minimum

Lower bound $z \ge p_0$ gives
$z_{\min} = p_0$.
From $0 \le x_1 \le p_2$, take $x_1^* = p_2$.
Then $x_2 = z_{\min} - x_1$:

$$x_2^* = p_0 - p_2$$

The constraint $x_2 \le p_1$ requires
$p_0 - p_2 \le p_1$, i.e. $p_0 \le p_1 + p_2$,
already enforced by P.

$$\boxed{z_{\min} = p_0, \quad (x_1^*, x_2^*) = (p_2,\, p_0 - p_2)}$$

Example: `sor1d.pip` Problem Setup

Variables: $x_1, x_2, x_3, x_4 \in \mathbb{Z}$, **Parameters:** $p_0, p_1 \in \mathbb{Z}$,
Objective: $g(\mathbf{x}) = x_1 + x_2 + x_3 + x_4$

Constraints:

$$
\begin{array}{rrrr}
0 \le p_0 & 0 \le x_4 - 2 & 0 \le -2x_2 + x_3 & 0 \le x_2 \\
0 \le p_1 & 0 \le -2x_1 + 2x_3 + x_4 - 4 & 0 \le -2x_1 + 2x_3 + p_1 - 5 & 0 \le -2x_1 + 4x_2 + p_1 - 3 \\
0 = -x_1 + 2 & 0 \le -x_4 + p_1 - 1 & 0 \le -x_3 + p_0 & 0 \le -2x_2 + p_0 \\
0 = -x_2 + 1 & 0 \le 2x_1 - 2x_3 - x_4 + 5 & 0 \le 2x_2 - x_3 + 1 & 0 \le 2x_1 - 4x_2 + 3 \\
0 = -x_3 + 2 & 0 \le 2x_1 + 1 & 0 \le 2x_1 - 2x_3 + 3 & 0 \le -2x_1 + 2p_0 + p_1 - 5 \\
0 = -x_4 + 4 & 0 \le x_3 - 1 & &
\end{array}
$$

Introduce auxiliary variable z with $z = x_1 + x_2 + x_3 + x_4$

IPD produces a single $\mathbb{Z}$Polyhedron(P, L) where:

$$
P := \begin{cases} x_1 = 2 \\ x_2 = 1 \\ x_3 = 2 \\ x_4 = 4 \end{cases}
\qquad
L := \left(\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \\ 2 \\ 4 \end{pmatrix} \right)
$$

Example: `sor1d.pip` Bounds on z and Optimizer

Because IPD yields a *single feasible point* $(x_1, x_2, x_3, x_4) = (2,\ 1,\ 2,\ 4)$ valid under the parameter constraint $p_0 \geq 2,\ p_1 \geq 5$, the objective value is *uniquely determined*:

$$z = x_1 + x_2 + x_3 + x_4 = 2 + 1 + 2 + 4 = 9.$$

Hence $z_{\min} = z_{\max} = 9$ and both optimizers coincide.

Maximum

The polyhedron P fixes all variables:

$$x_1^* = 2, \quad x_2^* = 1, \quad x_3^* = 2, \quad x_4^* = 4.$$

There is no slack; the unique feasible point *is* the maximizer.

$$\boxed{z_{\max} = 9,\ (x_1^*, x_2^*, x_3^*, x_4^*) = (2, 1, 2, 4)}$$

Minimum

Identically, the lower bound on z also evaluates to 9:

$$z_{\min} = z_{\max} = 9$$

under $p_0 \geq 2$ and $p_1 \geq 5$.

For parameter values *not* satisfying $p_0 \geq 2$ and $p_1 \geq 5$, IPD reports **no solution**

$$\boxed{z_{\min} = 9,\ (x_1^*, x_2^*, x_3^*, x_4^*) = (2, 1, 2, 4)}$$

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.

Test	Feautrier	Barvinok	Sample Point	Locus
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

Plan

Conclusion

How our algorithm differs from ISL and Barvinok:

ISL (in `C`) [2, 7]	**Barvinok** (in `C`) [1, 7]	**IPD** (in `Maple`) [4, 5]
Parametric simplex; lexicographic order by default; requires *separate* calls for min and max	Rational generating functions; max via negation of `lexmin`; *two* separate calls needed	Both min *and* max extracted from the **same** $\mathbb{Z}$-polyhedron decomposition

---

Two kinds of optimizer output our algorithm produces:

- **Sample Point Method:** picks a concrete integer witness $(x_1^*, \ldots, x_n^*)$; existence guaranteed by normalization of the IPD output; may require an auxiliary ILP on hard instances
- **Locus Method:** back-substitutes through the normalized $\mathbb{Z}$-polyhedron to express each x_i^* as a *closed-form function of the parameters*; avoids auxiliary ILP, slightly faster in practice

Future:

- Non-linear PILP for compiler scheduling.
- Develop a `C` library for computing $\mathbb{Z}$-Polyhedral set and thus for doing PILP.

References I

[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] P. Feautrier. "Parametric integer programming". In: *RAIRO. Recherche opérationnelle* 22.3 (1988), pp. 243–268.

[3] R. J. Jing and M. Moreno Maza. "Computing the Integer Points of a Polyhedron, I: Algorithm". In: *Proceedings of CASC*. 2017, pp. 225–241.

[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.

References II

[5] R.-J. Jing, Y. Lei, C. F. S. Maligec, M. Moreno Maza, and C. Mukherjee. "Quantifier Elimination Over the Integers". In: *Proceedings of the 2025 International Symposium on Symbolic and Algebraic Computation*. ISSAC '25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 353–362. ISBN: 9798400720758. DOI: 10.1145/3747199.3747580. URL: https://doi.org/10.1145/3747199.3747580.

[6] M. Moreno Maza, R.-J. Jing, Y. Lei, and C. Mukherjee. *Quantifier Elimination Over the Integers*. SSRN Preprint. Available at SSRN: https://ssrn.com/abstract=6178290. 2025. DOI: 10.2139/ssrn.6178290. URL: https://dx.doi.org/10.2139/ssrn.6178290.

References III

[7] S. Verdoolaege. "*isl*: An Integer Set Library for the Polyhedral Model". In: *Mathematical Software - ICMS 2010, Third International Congress on Mathematical Software, Kobe, Japan, September 13-17, 2010. Proceedings*. Ed. by K. Fukuda, J. van der Hoeven, M. Joswig, and N. Takayama. Vol. 6327. Lecture Notes in Computer Science. Springer, 2010, pp. 299–302.