$\mathbb{Z}$-Polyhedra and LBLs in PolyLib

Vincent Loechner, Dhimiter Riza

University of Strasbourg and Inria, France

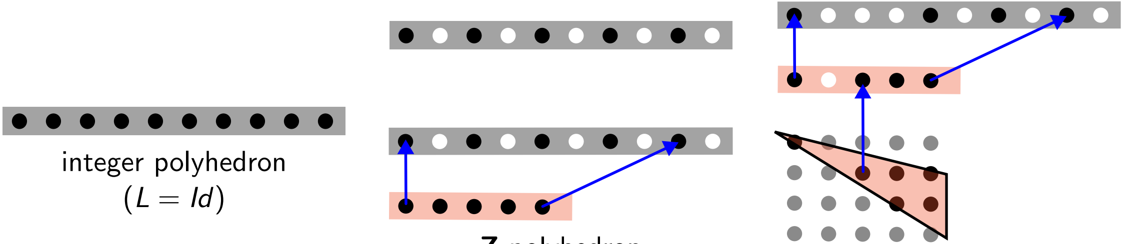
IMPACT Workshop, January 2026

Thanks to Patrice Quinton for making this happen!

Outline

Integer Sets

$$\mathcal{Z} = \left\{ z = Ly + l \;\middle|\; Cy + c \geq 0,\ y \in \mathbb{Z}^d \right\}$$



integer polyhedron
($L = Id$)

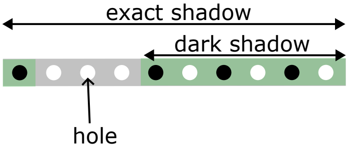
Z-polyhedron
(L is full-rank)

LBL
linearly bounded lattice

Integer polyhedra $\subset$ **Z**-polyhedra $\subset$ LBLs $\subset$ (unions of **Z**-polyhedra) $=$ (unions of LBLs)

Some History

- Elimination of integer variables in ILP: dark shadow and exact shadow [Pugh 1991]



Some History

- Elimination of integer variables in ILP: dark shadow and exact shadow [Pugh 1991]
- First implementation of **Z**-polyhedra in PolyLib in 2000
  [Quinton-Rajopadhye-Risset, 1997] [Nookala-Risset, 2000]

$$\mathcal{Z} = \text{lattice} \cap \text{rational polyhedron}$$

- Representation of **Z**-polyhedra and LBLs as the affine image of a full-dimensional integer coordinate polyhedron [Gautam-Rajopadhye, 2007]

$$\mathcal{Z} = \{Lz + l \mid Cz + c \geq 0, z \in \mathbb{Z}^d\}$$

- Same representation, but non-full dimensional integer coordinate polyhedron, at the cost of some more complex algorithms [Iooss-Rajopadhye, 2012]

→ libraries for manipulating unions of LBLs are either absent, limited or no longer maintained

Outline

Canonical Lattice Function

$$\mathcal{A} = \{Lz + l \mid z \in \mathbb{Z}^d\}$$

- To be in canonical form L and l have to satisfy the condition that the affine homogeneous matrix
$$\hat{L} = \begin{pmatrix} 1 & 0 \cdots 0 \\ l & L \end{pmatrix}$$
is in column left Hermite normal form (HNF)
- the zero columns of $\hat{L}$ are on its right
- ensures uniqueness of the (left) non zero columns of $\hat{L}$

Example

```
#offset
#     v1   v2   v3
  1    0    0    0         1    0    0    0         1    0    0    0
  2    2   18   20    =    0    2    0    0         1    1    9   10
  2    5   60   65        12    5   15    0        -1    0    1    1
  2    1    2    3        -6    1   -7    0         0    0    0    1


         M            =            H                        U
```

Normalized LBL

LBL:

$$\mathcal{Z} = \{Lz + l \mid z \in \mathcal{C}, z \in \mathbb{Z}^d\}$$

with a rational coordinate polyhedron:

$$\mathcal{C} = \{z \in \mathbb{Q}^d \mid Cz + c \geq 0\}$$

In normalized form:

1. canonical lattice function
2. no equalities in the rational coordinate polyhedron
   (but the integer set of points may contain implicit equalities)
3. the zero columns of L are eliminated if the sufficient condition that the dark shadow of the coordinate polyhedron covers its exact shadow is verified

Properties:

- if matrix L has no zero columns (on the right) then the LBL is a **Z**-polyhedron
- non-uniqueness

Computing the Normalized Form of an LBL

1. Canonicalize the lattice function
   - compute $L = HU$
   - H is the new lattice function
   - the new coordinate polyhedron is its preimage by U.
2. Eliminate the equalities of the coordinate polyhedron

   (and reduce the dimension of L)
   - compute the integer kernel of the equalities matrix: $K = HNF(ker(Eq))$
   - the new lattice function is: LK
   - the new coordinate polyhedron is its preimage by K
3. Eliminate zero columns of L

   For each zero columns of L:
   - compute the dark shadow and the exact shadow
   - if the exact shadow is contained in the dark shadow remove the column from L and project the (rational) coordinate polyhedron along the corresponding dimension

Normalized Form of Unions of LBLs

A union of LBLs is a list of pairs:

$$(L, D)$$

where L is a PolyLib matrix (in homogeneous form) and D a PolyLib domain.

The normalized form of a union of LBLs is defined as:

- each lattice matrix L of the list is in affine HNF
- it appears at most once in the list
  (coordinate polyhedra are merged into polyhedral domains)
- there are no empty coordinate polyhedra in the list
  (unless the whole union is a single empty LBL)

Note: non-uniqueness because different lattice decomposition can represent the same set

Outline

Lattice Affine HNF

```
void AffineHermite(Matrix *A, Matrix **H, Matrix **U)
```

$\rightarrow$ compute the affine left HNF of homogeneous matrix `A`, such that $\hat{A} = \hat{H}\hat{U}$

constant moved as first column (constant dimension)
and homogeneous dimension moved as first row

$$\hat{A} = \begin{pmatrix} 1 & 0 \cdots 0 \\ a & A \end{pmatrix}$$

Algorithm:

- basic HNF engine of PolyLib (similar to Gaussian elimination)

Lattice Intersection

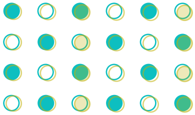
```
Matrix *LatticeIntersection(Matrix *A, Matrix *B)
```

$\rightarrow$ compute the intersection of `A` and `B`

Algorithm:

1. build:

$$T = \left(\begin{array}{cc|cc} 1 & 0 \ldots 0 & 1 & 0 \ldots 0 \\ a & A & b & B \\ \hline 1 & 0 \ldots 0 & 0 & 0 \ldots 0 \\ a & A & 0 & 0 \ldots 0 \end{array}\right)$$



2. compute H the left HNF of T
3. the bottom right matrix of H is the intersection of the two lattices (zero columns above)

Lattice Complement / Difference

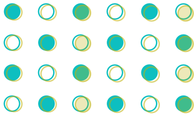
```
LatticeUnion *LatticeDifference(Matrix *A, Matrix *B)
```

→ compute the difference (`A` $\setminus$ `B`)

→ `A` and `B` must be of exact same dimensions (else the difference might not be finite)

→ the result is a **union** of lattices

→ if `A` is `NULL`, compute the $\mathtt{B}^c$



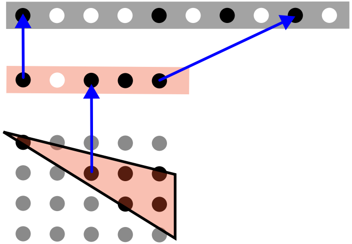
Algorithm:

- first compute the intersection $\mathtt{J} = \mathtt{A} \cap \mathtt{B}$
- take `J` out of `A`: build the matrix spreading all points of `A` that are not part of `J`,
  by scanning the pivots of `A` and generating alternative rows not intersecting `J`.

Single LBL image

```
LBL *sLBLImage(LBL *A, Matrix *M)
```

→ compute the image of `A` by matrix `M`



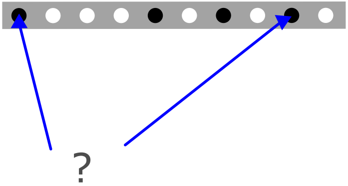
Algorithm:

- let `A = (L, D)`
- return the normalized single LBL defined by `(M L, D)`

Single LBL preimage

```
LBL *sLBLPreimage(LBL *A, Matrix *M)
```

$\rightarrow$ compute the preimage of `A` by matrix `M`



Algorithm:

- if `M` is integer invertible: return the LBL image by $\mathtt{M}^{-1}$ of `A`
- else: compute the LBL as the points z' that verify $Mz' + m = Lz + l$, with $z \in \mathtt{D}$ and $\mathtt{A} = \mathtt{(L,\ D)}$ and normalize the result to remove the equalities.

Single LBL Intersection

```
LBL *sLBLIntersection(LBL *A, LBL *B)
```

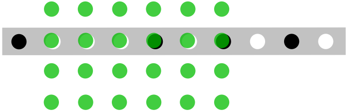
→ compute the intersection A ∩ B

Algorithm:

- if A and B are Z-domains (their lattices contain no zero column):
  return LBL($L_A \cap L_B$, preimage of hull(A) ∩ hull(B))
- else:
  explicitly compute the LBL spread by lattice L_A of the set of points z verifying

$$L_A z + l_A = L_B z' + l_B,\ z \in D_A,\ z' \in D_B$$

  and normalize the result.

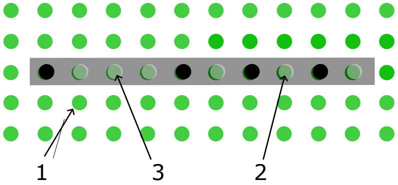


Single LBL Complement

```
LBL *sLBLComplement(LBL *A)
```

→ compute the complement of A: $\mathtt{A}^c = \mathbb{Z}^d \backslash \mathtt{A}$

→ used to compute the difference:
$A \backslash B = A \cap B^c$



Algorithm: return the union of

1. LBL($\mathbb{Z}^d$, `DomainComplement(hull(A))`)
2. LBL($\mathtt{L}_A{}^c$, Universe(dim))
3. LBL($\mathbb{Z}^d$, holes(A))

Single LBL to $\mathbf{Z}$-Domain

```
LBL *sLBL2ZDomain(LBL *A)
```

→ Compute the union of $\mathbf{Z}$-polyhedra that is equal to the LBL `A`.

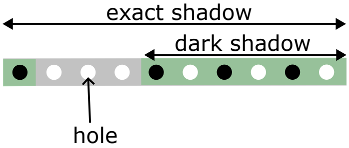
→ If the lattice function of `A` has some zero columns, they will be removed and the coordinate polyhedron will be split into a union of polyhedra.

Algorithm:

- compute the exact shadow E and the dark shadow D of `A`
- compute the holes in $E - D$

→ implies scanning all points of the coordinate polyhedron that can potentially be holes, check for integer solution

- return LBL($L_\emptyset$, E - holes)



Unions of LBLs: User Exposed Functions

```
LBL *LBLAlloc   (Matrix *Lat, Polyhedron *Domain)
void LBLFree    (LBL *A)
void LBLPrint   (FILE *fp, char *format, LBL *A)
LBL *LBLCopy    (LBL *A)
LBL *EmptyLBL   (int dimension)
LBL *UniverseLBL(int dimension)
Bool isEmptyLBL (LBL *A)

void LBLCanonical    (LBL *A)
LBL *LBLUnion        (LBL *A, LBL *B)
LBL *LBLImage        (LBL *A, Matrix *M)
LBL *LBLPreimage     (LBL *A, Matrix *M)
LBL *LBLIntersection (LBL *A, LBL *B)
LBL *LBLDifference   (LBL *A, LBL *B)
void LBLSimplifyEmpty(LBL *A)         // remove integer-empty polyhedra
Bool LBLIncluded     (LBL *A, LBL *B) // if simple checks fail,
                                      // check emptyness of the difference
LBL *LBL2ZDomain     (LBL *A)
```

Outline

Conclusion

- fast, robust, unified library implementation for manipulating **Z**-polyhedra, LBLs, and their union.
- Try it: https://github.com/vincentloechner/polylib/tree/LBL
- ongoing work: provide functions to facilitate enumeration and scanning

Outline