INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION INTERNE
N° 1330

A LIBRARY FOR $\mathcal{Z}$-POLYHEDRAL OPERATIONS

SUNDER PHANI KUMAR NOOKALA AND TANGUY RISSET

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES
Campus de Beaulieu – 35042 Rennes Cedex – France
Tél. : (33) 02 99 84 71 00 – Fax : (33) 02 99 84 71 71
http://www.irisa.fr

A Library for $\mathcal{Z}$-polyhedral Operations

Sunder Phani Kumar Nookala and Tanguy Risset

Thème 1 — Réseaux et systèmes
Projet COSI

Publication interne n° 1330 — Mai 2000 — 29 pages

Abstract: Polyhedra are commonly used for representing iteration domains of loop nests with unit stride: the iteration domain of a loop is associated with the set of integer points contained in a polyhedron. $\mathcal{Z}$-polyhedra are natural extension of polyhedra, in the sense that they represent iteration domains of loop nests with non-unit stride (they are polyhedra intersected with integral lattices). The polyhedral library (Polylib) has been developed for computing on polyhedra, it is now widely used in the automatic parallelization research community. This report describes the implementation of the extension of Polylib to $\mathcal{Z}$-polyhedra. We describe algorithms used for computing on lattices and $\mathcal{Z}$-polyhedra, and we provide technical documentation for the $\mathcal{Z}$-polyhedral library (data structures, functions available).

Key-words: polyhedra, lattice, $\mathcal{Z}$-polyhedra, loop nest, loop transformation

(Résumé : tsvp)

Centre National de la Recherche Scientifique
(UPRESSA 6074) Université de Rennes 1 – Insa de Rennes

Institut National de Recherche en Informatique
et en Automatique – unité de recherche de Rennes

Une bibliothèque pour les opérations $\mathcal{Z}$-polyèdrales

Résumé : Les polyèdres sont couramment utilisés pour représenter les espaces d'itération des boucles de pas 1 imbriquées: l'espace d'itération d'une boucle est associé avec l'ensemble des points entiers contenus dans un polyèdre. Les $\mathcal{Z}$-polyèdres sont une extension naturelle des polyèdres, en ce sens qu'il représentent les espaces d'itération des boucles imbriquées dont les pas peuvent être différents de 1 (ce sont des polyèdres intersectés avec des treillis entiers). La librairie polyédrique (Polylib) a été développée pour fournir une bibliothèque de calcul sur les polyèdres, elle est maintenant largement utilisée dans la communauté de recherche en parallélisation automatique. Ce rapport décrit l'implémentation de l'extension de Polylib aux $\mathcal{Z}$-polyèdres. Nous décrivons les algorithmes utilisés pour le calcul sur les treillis entiers et sur les $\mathcal{Z}$-polyèdres, et nous donnons une documentation technique de la librairie $\mathcal{Z}$-polyédrique (structure de données, fonctions disponibles).

Mots clés : polyèdre, treillis, $\mathcal{Z}$-polyèdre, boucles imbriquées, transformations de boucles

1 Introduction

Loop nests are some of the most time consuming structures of programs. The efficient compilation of loop nests has always been a major issue in the design of compilers. With the parallel computer generation, compilation should include parallelization. The loop parallelization research community now widely uses the abstraction of loop nests as polyhedra. The iteration domain of a loop nest can be represented by convex polyhedra (each integral point of the polyhedron represents a vector of iteration indices). Efficient computational kernel on convex polyhedra [Pug92, Fea92, Bal95, Tei93, Wil94] are now used in different research communities (automatic parallelization, high level VLSI design) as well as in prototypes of commercial compilers [GLW98]. This *polyhedral theory*, derives from the theory of linear and integer programming [MRTT53, Che68, Sch86, NW88].

The polyhedral library `Polylib` was a very successful tool. `Polylib` [Wil93] implemented operations on rational polyhedra efficiently. A convex polyhedron has two dual representations: intersection of half spaces, or combinations of vertices, rays and lines. These representations were introduced by Motzkin [MRTT53], and Chernikova showed how to go from one to the other [Che68]. Successive improvements of the Chernikova algorithm [FQ88, Le 92] provided an efficient computational kernel on convex polyhedra which led to the `Polylib`.

However, polyhedra are not powerful enough to solve certain important problems like, partitioning [TT93] or handling loop nests with non-unit stride [Hel94]. Teich and Thiele [TT93] formally introduced the images of polyhedra by affine mappings. The major problem with this class of sets is that it is very large and some operations are not possible because the affine mappings considered are generally not invertible. The restriction to image of polyhedra by invertible mapping was first studied by Ancourt [Anc91]. In [Anc91] she introduced *$\mathcal{Z}$-polyhedra* as intersection of polyhedra and lattices, as we will see, these sets also correspond to image of polyhedra by invertible functions.

In [QRR97] and [QRR96a] the implementation of a computational library was studied for $\mathcal{Z}$-polyhedra. The underlying motivation was the extension of the Alpha language to $\mathcal{Z}$-polyhedra and subsequently, the extension of automatic VLSI design towards a more general model. This report explains how the implementation of the $\mathcal{Z}$-polyhedral library was realized. First we detail each algorithm for computing on lattices or $\mathcal{Z}$-polyhedra. Some algorithms are new, like for instance the algorithm for lattice difference, most of them are gathered from different sources. Then we provide a technical documentation of the $\mathcal{Z}$-polyhedral library: how is it included (data structures) in `Polylib` and which functions are available.

2 Definitions

In this section we define some terms which we will be using frequently in the next sections.

Definition 1 Rational polyhedron *A rational polyhedron is a subset of Q^n defined by a finite set of affine inequalities.*

We will call *polyhedron* a set of integral points in a rational polyhedron. For instance, $Q_1 = \{i, j \mid 0 \le i \le 5, 0 \le 3j \le 20\}$ can be interpreted as a rational polyhedron or as a polyhedron (in that case, it contains 42 points, see figure 1).

Definition 2 Lattice *A lattice is a subset of Q^n defined by integral linear combinations of linearly independent rational vectors of Q^n, called generating vectors (or basis of the lattice) plus an affine vector.*

An *integral lattice* is a lattice whose generating vectors and affine part are integral. For instance, $L_1 = \{2i + 1, 3j + 5 \mid i, j \in \mathcal{Z}\}$ can be interpreted as an integral lattice: it is the lattice defined by any integral linear combinations of the vectors $(2, 0)$ and $(0, 3)$, plus the vector $(1, 5)$:

$$L_1 = \left\{ i \begin{pmatrix} 2 \\ 0 \end{pmatrix} + j \begin{pmatrix} 0 \\ 3 \end{pmatrix} + \begin{pmatrix} 1 \\ 5 \end{pmatrix} \mid i, j \in \mathcal{Z} \right\}.$$

The points $(3, 5), (3, 8), (5, 8)$ belong to lattice L_1 (see figure 1). An integral lattice can also be considered as the solution of a system of Diophantine equations. Here, L_1 is the set of (z_1, z_2) which are solutions of the following system of Diophantine equations (where i, j, z_1 and z_2 are the integral unknowns):

$$i \begin{pmatrix} 2 \\ 0 \end{pmatrix} + j \begin{pmatrix} 0 \\ 3 \end{pmatrix} + \begin{pmatrix} 1 \\ 5 \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$$

In this paper we represent a lattice as a set of generating vectors, plus the affine part. So, lattice L_1 will be represented as:

$$L_1 = \left(\begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 1 \\ 5 \end{pmatrix} \right)$$

Definition 3 True Dimension *The true dimension of a subset S of Z^n is the dimension of the minimum vector space that contains the convex hull of S. A subset of Z^n is full dimensional if its true dimension is n.*

Consider, for example, the polyhedron S of $\mathcal{Z}^2$: $S = \{i + j, i + j \mid i, j \in Z\}$. The true dimension of S is 1. From now on, whenever we talk of lattices we refer to *full dimensional, integral* lattices.

Definition 4 $\mathcal{Z}$-polyhedron. *A $\mathcal{Z}$-polyhedron is the intersection of a polyhedron and a lattice.*

Alternatively, a $\mathcal{Z}$-polyhedron can be defined as the image of a polyhedron by an invertible, integral function. The conversion between the two representation can be done easily. Consider, for instance, the lattice L_1 and the polyhedron Q_1 discussed in the above examples, $Z_1 = L_1 \cap Q_1$ is a $\mathcal{Z}$-polyhedron. Z_1 can also be expressed as:

$$Z_1 = \{2i + 1, 3j + 5 \mid -1 \le 2i \le 4, -15 \le 9j \le 5\} \quad ,$$

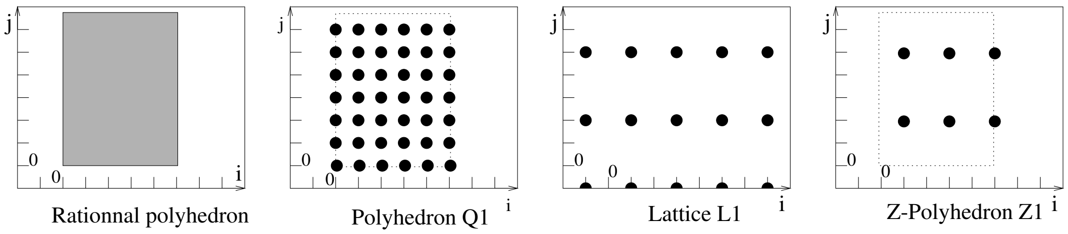


Figure 1: example of polyhedron $Q_1 = \{i,j \mid 0 \leq i \leq 5, 0 \leq 3j \leq 20\}$, lattice $L_1 = \{2i+1, 3j+5 \mid i,j \in \mathcal{Z}\}$ and $\mathcal{Z}$-polyhedron $Z_1 = Q_1 \cap L_1$ (the dotted line represent the shape of the original rational polyhedron).

which is the image of polyhedron $Q_2 = \{i,j \mid -1 \leq 2i \leq 4, -15 \leq 9j \leq 5\}$ by the function $(i,j \to 2i+1, 3j+5)$. Q_2 is obtained by taking the preimage of Q_1 by the function defining the lattice: $(i,j \to 2i+1, 3j+5)$.

Definition 5 **$\mathcal{Z}$-domain**. *A $\mathcal{Z}$-domain is a finite union of $\mathcal{Z}$-polyhedra.*

Definition 6 Hermite Normal Form. *A matrix of full row rank is said to be in Hermite Normal Form (HNF) if it has the form [B 0] where B is a nonsingular, lower triangular, non negative matrix, in which each row has a unique maximum entry located on the main diagonal of B.*

Theorem 1 Hermite Decomposition. *For any rational matrix M of full row rank, there exists a unique matrix H in Hermite Normal Form and an unimodular matrix U such that $M = HU$.*

Proof: [Sch86, p.45]

Consider the following matrices M, H and U, HU is the Hermite decomposition of M.

$$M = \begin{pmatrix} 1 & 2 & 3 \\ -3 & 2 & 0 \\ 1 & 0 & 0 \end{pmatrix} \; H = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 4 & 5 & 6 \end{pmatrix} \; U = \begin{pmatrix} 1 & 2 & 3 \\ -3 & 2 & 0 \\ 2 & -3 & -2 \end{pmatrix}$$

Theorem 2 Smith Decomposition. *If A is an $n \times n$ non-singular integer matrix, there exist unimodular matrices U and V such that:*

i. $UAV = \Delta$
ii. *Δ is a diagonal matrix with entries $\delta_i \in Z$,*
iii. $\delta_1 \mid \delta_2 \ \ldots \mid \delta_n$

Δ is unique and is called the Smith normal form of A.

Proof: see [Sch86, p. 50].

Consider again the matrix M above, its Smith normal decomposition is: $UMV = \Delta$ where:

$$\Delta = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 6 \end{pmatrix} \; U = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & -1 \\ 2 & 1 & 1 \end{pmatrix} \; V = \begin{pmatrix} 1 & -1 & 0 \\ 0 & -1 & 3 \\ 0 & 1 & -2 \end{pmatrix}$$

3 Operating on lattices

Before operating on $\mathcal{Z}$-polyhedra, we must be able to perform the basic operations (union, intersection, ...) on lattices. For some of these operation, we need to highlight properties of lattices. We start off by describing the canonical representation of a lattice and then we describe the algorithms for computing on lattices.

3.1 Unique Representation of a lattice

As we have briefly seen in section 2, an affine lattice of $\mathcal{Z}^n$ can be represented as a matrix and a vector, where the columns of the matrix are the generating vectors of the lattice. We will denote $L(A, a)$ (or simply (A, a)) the lattice whose generating vectors are the columns vectors of A and for which a is the constant vector:

$$L(A, a) = \{Ax + a \mid x \in \mathcal{Z}^n\}$$

As we consider only full dimensional integral lattices, matrix A will be *an $n \times n$ non singular, integral matrix*, and a is a vector of order n.

Proposition 1 Linear lattice canonical form. *Let A and A' be rational matrices of full row rank, with Hermite Normal Forms $[B\ 0]$ and $[B'\ 0]$, respectively. Then the columns of A generate the same linear lattice as that of A', if and only if $B = B'$.*

Proof: see [Sch86, p. 48].

Proposition 2 Affine lattice Normal Form. *Given a full dimensional integral affine lattice L in $\mathcal{Z}^n$, there exist a unique integral matrix H in Hermite Normal Form and a unique integral vector h such that L=L(H,h) with the property $0 \le h_i < H_{i,i}$ for $1 \le i \le n$.*

Proof: see [QRR96b].

For example consider L:

$$L = \left(\begin{pmatrix} 0 & 2 \\ 3 & 0 \end{pmatrix}, \begin{pmatrix} 6 \\ 4 \end{pmatrix} \right) = \{2j + 6, 3i + 4 \mid i, j \in \mathcal{Z}\} \quad ,$$

its unique normal form is

$$L = \left(\begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right) \quad .$$

Another important property is the fact that a basis of a sub-lattice of L can always be extracted from a particular basis of L by scaling the basis vectors:

Theorem 3 *Let L_1, L_2 be two linear lattices of dimension n such that $L_2 \subset L_1$. Then, there exists a basis $(b_1, \dots, b_n)$ of L_1 and integers $(a_1, \dots, a_n)$ such that $(a_1 b_1, \dots, a_n b_n)$ is a basis for L_2. Moreover, we may suppose that $a_i > 0$ $(1 \le i \le n)$ and, $a_i \mid a_{i+1}$ $(1 \le i < n)$.*

Proof: [Cas78, p.220]

This particular basis $(b_1, \dots, b_n)$ is not explicitly given, but one can notice that the matrix whose diagonal is $(a_1, \dots, a_n)$ is in Smith normal form. This is the idea that gives rise to the lattice difference algorithm presented in section 3.3.

3.2 Lattice intersection

Given two full dimensional, affine, integral lattices $L_1 = (A_1, b_1)$ and $L_2 = (A_2, b_2)$ in $\mathcal{Z}^n$, we look for an integral, affine lattice $L_3 = (A_3, b_3)$ such that $L_3 = L_1 \cap L_2$. We know that:

$$\begin{aligned} &\forall y_1 \in L_1, \exists x_1 \in \mathcal{Z}^n,\ A_1 x_1 + b_1 = y_1 \\ &\forall y_2 \in L_2, \exists x_2 \in \mathcal{Z}^n,\ A_2 x_2 + b_2 = y_2 \end{aligned}$$

If z is a point in L_3, there exist $x_1, x_2 \in \mathcal{Z}^n$ such that $z = A_1 * x_1 + b_1 = A_2 * x_2 + b_2$. These two equations form a system of Diophantine equations (Id_n represent the $n \times n$ identity matrix):

$$\begin{pmatrix} Id_n & -A_1 & 0 \\ Id_n & 0 & -A_2 \end{pmatrix} \begin{pmatrix} z \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \tag{1}$$

As A_1 and A_2 are of rank n, these $2n$ equations are linearly independent. As mentionned in [QRR96b], this intersection is either empty or full dimensional, hence the solution (if it exists) will be described by n free variables. Therefore, the form of the solution of 1 is:

$$\begin{pmatrix} z \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} S_1 \\ S_2 \\ S_3 \end{pmatrix} t + \begin{pmatrix} s_1 \\ s_2 \\ s_3 \end{pmatrix}\ ,\ t \in \mathcal{Z}^n \quad ,$$

where t is a vector of order n of free variables and S_1 is a $n \times n$ invertible matrix (the algorithm used for solving Diophantine equations is explained in appendix B). Hence we obtain the description of lattice L_3: $z = S_1 t + s_1,\ t \in \mathcal{Z}^n$ and $L_3 = L(S_1, s_1)$.

As an example, consider the two lattices:

$$L_1 = \{3i + 5, 4i + j \mid i, j \in \mathcal{Z}\}\ ,\quad L_2 = \{2j, 3j \mid i, j \in \mathcal{Z}\} \quad .$$

They can be represented as:

$$L_1 = \left(\begin{pmatrix} 3 & 0 \\ 4 & 1 \end{pmatrix}, \begin{pmatrix} 5 \\ 0 \end{pmatrix} \right)\ ,\quad L_2 = \left(\begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right)$$

The intersection of L_1 and L_2 is:

$$L_1 \cap L_2 = \left(\begin{pmatrix} 6 & 0 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 2 \\ 0 \end{pmatrix} \right)$$

3.3 Lattice difference

Given two lattices, L_1 and L_2 in $\mathcal{Z}^n$, we want to find the difference $L_1 - L_2$. Unfortunately, the difference of two lattices is not necessary a lattice. In this section we show that the difference of two lattices can always be expressed as a finite union of lattices.

First, note that Theorem 3 can be trivially extended to affine lattices because it only concerns the linear part of lattices. Next, note that, when we compute $L_1 - L_2$, we can suppose that $L_2 \subset L_1$ because $L_1 - L_2 = L_1 - (L_1 \cap L_2)$ and $(L_1 \cap L_2) \subset L_1$. Now, using this assumption and Theorem 3 we show how to express lattice L_1 as a union of lattices based on the basis of L_2.

Suppose that we have the representation of L_1 and L_2 in basis B_1 and B_2 that satisfy Theorem 3: $L_1 = L(B_1, f_1)$ and $L_2 = L(B_2, f_2)$, where B_1 and B_2 are such that: if $B_1 = (b_1, \ldots, b_n)$, then $B_2 = (a_1 b_1, \ldots, a_n b_n)$ where $(a_1, \ldots, a_n)$ are integers satisfying the properties of Theorem 3.

Consider the set $\mathcal{D}_{a_1,\ldots,a_n} = \{(j_1, \ldots, j_n) \mid 1 \leq j_k \leq a_k - 1 \;\; \forall k, \; 1 \leq k \leq n\}$, and let L_1' be:

$$L_1' = \bigcup_{(j_1,\ldots,j_n) \in \mathcal{D}_{a_1,\ldots,a_n}} (B_2, f_1 + j_1 b_1 + \ldots + j_n b_n) \tag{2}$$

We first prove that $L_1 = L_1'$. Let $x \in L_1(B_1, f_1)$, there exist integers $(y_1, \ldots, y_n)$ such that $x = \sum_{i=1}^n y_i b_i + f_1$. Let us note $q_i = y_i \; div \; a_i$ and $j_i = y_i \; mod \; a_i$ (hence, $0 \leq j_i < a_i$ and $(j_1, \ldots, j_n) \in \mathcal{D}_{a_1,\ldots,a_n}$). We have: $x = \sum_{i=1}^n q_i a_i b_i \; + \; \sum_{i=1}^n j_i b_i + f_1$, hence, $x \in L(B_2, f_1 + \sum_{i=1}^n j_i b_i) \subset L_1'$

Conversely, let $x \in L_1'$, there exist an $y \in \mathcal{Z}^n$ and $(j_1, \ldots, j_n)$ with $0 \leq j_k \leq a_k - 1, \;\; \forall k, \; 1 \leq k \leq n$, such that $x = B_2 y + \sum_{i=1}^n j_i b_i + f_1$. As $B_2 = (a_1 b_1, \ldots, a_n b_n)$, x can be expressed as: $x = \sum_{i=1}^n (a_i y_i + j_i) b_i + f_1 = \sum_{i=1}^n a_i' b_i + f_1$. Hence, $x \in L_1$ and we get $L_1 = L_1'$.

Thus we have been able to rewrite the lattice $L_1 = (B_1, f_1)$ as a union of lattices. Now, we will show that $L_2 = (B_2, f_2)$ is one of the lattices in the union. Consider $x \in L_2$, there exists $y_1, \ldots, y_n \in \mathcal{Z}$ such that $x = \sum_{i=1}^n (y_i a_i b_i) + f_2$. Since $x \in L_1$, there exists $y_1', \ldots, y_n' \in \mathcal{Z}$ such that $x = \sum_{i=1}^n (b_i y_i') + f_1$. Rewriting the above two equations we get:

$$\sum_{i=1}^n (b_i y_i') + f_1 = \sum_{i=1}^n (a_i b_i y_i) + f_2$$

or,

$$f_2 = \sum_{i=1}^n (b_i y_i') - \sum_{i=1}^n (a_i b_i y_i) + f_1 = \sum_{i=1}^n ((y_i' - a_i y_i) b_i) + f_1 \quad .$$

If we introduce $f_2' = \sum_{i=1}^{n}((y_i' - a_i y_i) \ mod \ a_i)b_i + f_1$, It can be easily proved that $L(B_2, f_2) = L(B_2, f_2')$ (this operation correspond to shifts of a finite number of each basis vector). Now, $L(B_2, f_2')$ is one of the component of the union of equation (2), hence L_2 is one of the lattices in the union of lattices. Since the union of equation (2) is disjoint, removing that lattice from the union we get the lattice difference $L_1 - L_2$.

Until now we have assumed that we have access to the Basis B_1, B_2. Below we describe a method to find B_1 and B_2 which have the properties required by theorem 3. Let $L_1 = (H_1, a_1)$ and $L_2 = (H_2, a_2)$, where H_1 and H_2 are the generating vectors (or the basis) and a_1 and a_2 are the affine parts. Since H_1 and B_1 must generate the same lattice, they differ only by a unimodular matrix. Therefore we have:

$$B_1 = H_1 U_1 \quad \text{and similarly,} \quad B_2 = H_2 U_2$$

We know, from Theorem 3 that, $B_1 \Delta = B_2$ where Δ is a diagonal matrix such that $\delta_i \mid \delta_{i+1}$. Putting together the above three equations, we have:

$$\Delta = U_1^{-1} H_1^{-1} H_2 U_2$$

As Δ is in Smith normal form, the above equation is the Smith normal decomposition of $H_1^{-1} H_2$. Hence, by using the Smith normal decomposition algorithm, we can find the basis B_1, B_2 and the lattice difference.

For example, let us take the same lattices L_1 and L_2:

$$L_1 = \left(\begin{pmatrix} 3 & 0 \\ 4 & 1 \end{pmatrix}, \begin{pmatrix} 5 \\ 0 \end{pmatrix} \right) \quad \text{and} \quad L_2 = \left(\begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right)$$

The lattice difference $L_1 - L_2$ is the union of the following lattices (note that the only "missing" lattice in $L_1 \cap L_2$):

$$\left(\begin{pmatrix} 6 & 0 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \end{pmatrix} \right) \cup \left(\begin{pmatrix} 6 & 0 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 5 \\ 1 \end{pmatrix} \right) \cup \left(\begin{pmatrix} 6 & 0 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \end{pmatrix} \right)$$
$$\cup \left(\begin{pmatrix} 6 & 0 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 5 \\ 2 \end{pmatrix} \right) \cup \left(\begin{pmatrix} 6 & 0 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 5 \\ 0 \end{pmatrix} \right)$$

3.4 Lattice preimage

A lattice can be considered as a set generated by linearly independent vectors, or as the solution to a system of Diophantine equations. Using the first definition of a lattice, we were able to prove that the preimage of a lattice by an affine, rational function is also a lattice [QRR96b]. In this section, we describe an algorithm to find the preimage of a lattice using the latter definition.

Definition 7 *The preimage of a lattice L_1 by an affine rational function $g : Q^n \to Q^l$ can be defined as $\{x \in \mathcal{Z}^n \mid \exists\, y \in L_1, g(x) = y\}$*

Note here that this is *not* the real preimage of the lattice. As the preimage of an integral lattice by a rational function may result in a rational lattice, we explicitly intersect this preimage with $\mathcal{Z}^n$ in order to stay in the set of integral lattices.

The preimage can be computed in the following way: Suppose $L = (A, b)$ is a lattice in $\mathcal{Z}^n$ and $g : Q^n \to Q^l$. Let L' be the preimage of L by g. Then,

$$L' = \{x \in \mathcal{Z}^l \mid \exists y \in L\ ,\ g(x) = y\} = \{x \in \mathcal{Z}^l \mid \exists z \in \mathcal{Z}^n\ ,\ g(x) = Az + b\}$$

Let G be the matrix representing the linear part of g and j its affine part. We have:

$$L' = \{x \in \mathcal{Z}^l \mid \exists z \in \mathcal{Z}^n\ ,\ Gx + j = Az + b\} = \{x \in \mathcal{Z}^l \mid \exists z \in \mathcal{Z}^n\ ,\ Gx - Az = b - j\}$$

Writing the above as a system of linear Diophantine equations, we get

$$\begin{pmatrix} G & -A \end{pmatrix} \begin{pmatrix} x \\ z \end{pmatrix} = \begin{pmatrix} b - j \end{pmatrix}$$

The above is a system of n linear Diophantine equations in $n + l$ unknowns. The above n equations are linearly independent. Hence we get a solution (if there is one) which will have l free variables. We get $x = A'x' + b'$, where $x' \in \mathcal{Z}^l$ and A' is a square $l \times l$ integral non singular matrix. Thus we have found the preimage of L by g

For instance, let $L_1 = \{6i, 3j \mid i, j \in \mathcal{Z}\}$ be a lattice. We want to find the preimage of this lattice by the function $f = (i, j, k \to i/2, j + k)$. The result is the lattice: $L_1' = \{12i, j, 2j + 3k \mid i, j, k \in \mathcal{Z}\}$

3.5 Lattice image

We define the image by an invertible f function as the preimage by f^{-1} (hence, again, it is not the real image, but the image intersection with $\mathcal{Z}^n$). The image of a lattice by a non invertible function is not computed because we do not need it for computing on $\mathcal{Z}$-polyhedra (the image of a $\mathcal{Z}$-polyhedron by such a function may not be a $\mathcal{Z}$-polyhedron).

4 Operating on $\mathcal{Z}$-polyhedra

In the previous section we have seen algorithms for lattice operations. In this section we will see how these operations have been used for implementing the computational kernel on $\mathcal{Z}$-domains.

4.1 $\mathcal{Z}$-polyhedra intersection

Given two $\mathcal{Z}$-polyhedra $Z_1 = (L_1 \cap Q_1)$ and $Z_2 = (L_2 \cap Q_2)$, the computation of their intersection uses the following property:

$$Z_3 = Z_1 \cap Z_2 = (L_1 \cap Q_1) \cap (L_2 \cap Q_2) = (L_1 \cap L_2) \cap (Q_1 \cap Q_2)$$

where $L_1 \cap L_2$ is a lattice and $Q_1 \cap Q_2$ is a polyhedron. We actually deal with $\mathcal{Z}$-domains which are union of $\mathcal{Z}$-polyhedra. Computing the intersection of two $\mathcal{Z}$-domains must be realized by union of the pairwise intersection of the underlying $\mathcal{Z}$-polyhedra.

4.2 $\mathcal{Z}$-polyhedra Difference

Given two $\mathcal{Z}$-polyhedra $Z_1 = (L_1 \cap Q_1)$ and $Z_2 = (L_2 \cap Q_2)$. We want to find the difference $Z_1 - Z_2$ (we note $\overline{A}$ for the complement of A):

$$\begin{aligned} Z_1 - Z_2 &= (L_1 \cap Q_1) - (L_2 \cap Q_2) \\ &= (L_1 \cap Q_1) \cap (\overline{L_2 \cap Q_2}) \\ &= (L_1 \cap Q_1) \cap (\overline{L_2} \cup \overline{Q_2}) \\ &= ((L_1 \cap Q_1) \cap \overline{L_2}) \cup ((L_1 \cap Q_1) \cap \overline{Q_2}) \\ &= ((L_1 \cap \overline{L_2}) \cap Q_1) \cup (L_1 \cap (Q_1 \cap \overline{Q_2})) \\ &= ((L_1 - L_2) \cap Q_1) \cup (L_1 \cap (Q_1 - Q_2)) \end{aligned}$$

In the previous section we have seen how to compute the lattice difference $L_1 - L_2$ and we already know how to compute $Q_1 - Q_2$, so we know all the terms in the above equation. Substituting them, we get the $\mathcal{Z}$-polyhedral difference. The extension to $\mathcal{Z}$-domains is shown by the following example:

Let Z_1 and Z_2 be two $\mathcal{Z}$-domains. Let Z_1 consist of $\mathcal{Z}$-polyhedra A and B and Z_2 consist of $\mathcal{Z}$-polyhedra C and D. Then,

$$Z_1 - Z_2 = ((A - C) - D) \cup ((B - C) - D)$$

4.3 $\mathcal{Z}$-domain image and preimage

Given a $\mathcal{Z}$-domain Z_1 and a function f, we can say that the image (or preimage) of the $\mathcal{Z}$-domain is the union of image (or preimage) of the underlying $\mathcal{Z}$-polyhedra by f. Given a $\mathcal{Z}$-polyhedron $Z = L \cap Q$ and an affine, rational function f (invertible in case of image) the procedure to find the preimage is as follows:

$$f^{-1}(Z) = f^{-1}(L \cap Q) = f^{-1}(L) \cap f^{-1}(Q)$$

Similarly for image it is,

$$f(Z) = f(L \cap Q) = f(L) \cap f(Q)$$

Remember that it is not the classical notion of image (or preimage), we take the intersection of the actual image (or preimage) and $\mathcal{Z}^n$ (n is the dimension of the image or preimage)

4.4 $\mathcal{Z}$-domain union

The union of two $\mathcal{Z}$-domains is just the concatenation of the two lists. Care is taken to minimize redundancy, i.e. in the list there are no equal $\mathcal{Z}$-polyhedra or no two $\mathcal{Z}$-polyhedra such that one is included in the other.

5 The $\mathcal{Z}$-polyhedral library

The $\mathcal{Z}$-polyhedral Library creates, operates on, and frees objects called *$\mathcal{Z}$-domains* made up of unions of $\mathcal{Z}$-polyhedra (described in the previous section). We start this section describing the way the affine space in represented in `Polylib`, they we discuss the data structures for lattices and $\mathcal{Z}$-domains. Then we go on to the operational side of the library and discuss briefly about the operations.

5.1 The homogeneous representation of affine spaces

`Polylib` manipulates *mixed inhomogeneous system of equations.* The terms inhomogeneous stands for the fact that it manipulates objects of an affine space (not a linear space). In [Wil93, Gol56], it is explained how to transform the inhomogeneous affine space of dimension n into an homogeneous vector space of dimension $n+1$. the mapping is

$$\mathcal{M} : x \longrightarrow \begin{pmatrix} \xi x \\ \xi \end{pmatrix}, \qquad \xi > 0 \quad .$$

With this mapping, a system $\mathcal{P} = \{x \mid Ax = b,\ Cx \geq d\}$ in the original inhomogeneous space is transformed into $\mathcal{C} = \{\tilde{x} \mid \tilde{A}\tilde{x} = 0,\ \tilde{C}\tilde{x} \geq 0\}$ where

$$\tilde{A} = (A \quad -b), \quad \tilde{x} = \begin{pmatrix} \xi x \\ \xi \end{pmatrix} \quad \text{and} \quad \tilde{C} = \begin{pmatrix} C & -d \\ 0 & 1 \end{pmatrix} \quad .$$

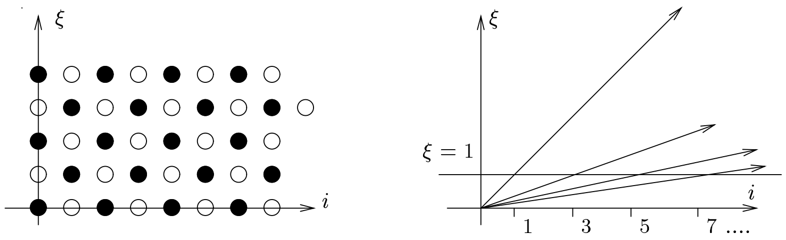
An intuitive representation of this mapping is the following: the set $\mathcal{P}$ can be seen as the intersection of the set $\mathcal{C}$ with the hyper-plane defined by the equality $\xi = 1$. This transformation has the advantage of simplification in the storage of the polyhedra (only cones are manipulated, hence only rays and lines are stored). It also simplifies computations.

Any affine transformation $x \mapsto F.x + f$ is naturally extended to the linear transformation $\begin{pmatrix} \xi x \\ \xi \end{pmatrix} \mapsto \begin{pmatrix} F & f \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \xi x \\ \xi \end{pmatrix}$. Hence, in this system, all integral affine transformations manipulated in `Polylib` must have a $(0, 0, \ldots, 0, 1)$ as last row. However, the fact that the last element of this row is not one may be used for expressing rational transformation. Consider, for instance, the matrix $G = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$. If we consider this matrix as a representation of an affine transformation after mapping $\mathcal{M}$, it represents the following function:

$$x \xrightarrow{\mathcal{M}} \begin{pmatrix} \xi x \\ \xi \end{pmatrix} \xrightarrow{G} \begin{pmatrix} \xi x \\ 2\xi \end{pmatrix} = \begin{pmatrix} 2\xi \frac{x}{2} \\ 2\xi \end{pmatrix} \xrightarrow{\mathcal{M}^{-1}} \frac{x}{2}$$

Hence the function represented by $\begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$ is equivalent to the function represented by: $\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{pmatrix}$. This works because the matrices are supposed to be applied on cones pointed

$$M_1 = \begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix}$$



(a) (b)

Figure 2: We use M_1 to represent, in the homogeneous space, the set of odd integers. (a) is the set of point described by M_1 if M_1 is considered as a linear lattice in the 2-dimensional space, this set is *not* a pointed cone . (b) is the real set L'_1 represented by M_1 (i.e. when M_1 is considered as the representation, in the 2-dimensional homogeneous space, of a 1-dimensional affine lattice): an infinite union of rays.

on 0, i.e. if a vector $\begin{pmatrix} \xi x \\ \xi \end{pmatrix}$ is in a set manipulated, then any vector $\begin{pmatrix} \lambda\xi x \\ \lambda\xi \end{pmatrix}$, $\lambda \geq 0$ is in this set.

In our extension of `Polylib` to handle $\mathcal{Z}$-polyhedra, we have naturally chosen the `Polylib` matrices to represent affine lattices because of the direct correspondence between affine lattices and affine functions. For instance the set L_1 of odd integers correspond (canonically) to the affine function $i \mapsto 2i + 1$ (L_1 is the image of $\mathcal{Z}$ by this function which is in Hermite normal form). This affine function is represented in the homogeneous system by the following matrix: $M_1 = \begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix}$. One must be very careful with the following: the columns vectors of matrix M_1 *cannot* be considered generating vectors of a lattice in the homogeneous space of dimension 2. The set (lets call it L'_1) that we manipulate in the homogeneous space is definitely *not* a lattice, it is the union of infinitely many half-lines which start from 0 and contain each of the points of the lattice of dimension 1. This should be more clear by looking at figure 2, the original lattice correspond to the intersection of the set L'_1 with the hyper-plane $\xi = 1$.

This implies, in particular, that we cannot consider M_1 as the representation of a set on which we could perform matrix operations. For instance, in our representation of lattices, the matrix $\begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$ represents the set $\{x/2 \in \mathcal{Z} \mid x \in \mathcal{Z}\} = \{x \mid x \in \mathcal{Z}\}$. Hence it could also be represented by the matrix $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. The important thing to remember here is that we use matrices to represent lattices but they are not matrix representation of function, they correspond to a different data type.

5.2 Data Structures

5.2.1 Lattice

A full dimensional, integral lattice is completely characterized by an integral affine non-singular function. `Polylib` already has a data structure `Matrix` which represents an affine function. We can use it to represent affine lattice but, to avoid confusion between matrices and lattices which are different objects (section 5.1), we introduce a new type `Lattice`. An affine lattice in $\mathcal{Z}^n$ can be represented by an $(n+1) \times (n+1)$ `Polylib` matrix. The C data structure for lattice is:

```
typedef struct matrix {
  unsigned NbRows, NbColumns;
  int **p;
  int *p_Init;
} Matrix;

typedef Matrix Lattice;
```

For instance, the lattice $\{2i, 3j+2 \mid i,j \in \mathcal{Z}\}$, will be represented by an 3×3 matrix as follows:

$$\begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

5.2.2 Polyhedron

The representation used for a polyhedron is the one of `Polylib` (see [Wil93] for details): system of constraints and collection of rays and lines. The C data structure for a polyhedral domain is:

```
typedef struct polyhedron
{
  struct polyhedron *next;
  unsigned Dimension, NbConstraints, NbRays, NbEqualities, NbLines;
  int **Constraints;
  int **Rays;
  int *p_Init;
} polyhedron;
```

5.2.3 $\mathcal{Z}$-domain

A $\mathcal{Z}$-polyhedron is represented internally as the image of a polyhedron by an affine, invertible mapping. This representation facilitates extension of the Alpha language to sparse polyhedral model [QRR96a]. The C data structure for $\mathcal{Z}$-domain (union of $\mathcal{Z}$-polyhedra) is:

```
typedef struct ZPolyhedron
{
  Lattice *Lat;
  Polyhedron *P;
  struct ZPolyhedron *next;
} ZPolyhedron;
```

for example, the $\mathcal{Z}$-domain $\{2i \mid 0 \leq i \leq 5\}$ will be represented as:

```
ZPOLYHEDRON: Dimension 1

LATTICE:
2 2
    2    0
    0    1
POLYHEDRON Dimension:1
           Constraints:2  Equations:0  Rays:2  Lines:0
Constraints 2 3
Inequality: [    1    0 ]
Inequality: [   -1    5 ]
Rays 2 3
Vertex: [    0 ]/1
Vertex: [    5 ]/1
```

5.2.4 Union of Lattices

The C data structure to represent union of Lattices is:

```
typedef struct LatticeUnion
{
  Lattice *M;
  struct LatticeUnion *next;
} LatticeUnion;
```

5.3 Operations on $\mathcal{Z}$-polyhedra

We present here the functions available in the $\mathcal{Z}$-polyhedra library. Most of these functions do not modify their arguments (space is allocated for the result and return pointer on it). Some function (solveDiophantine, Affine hermite, AffineSmith, CanonicalForm) create new objects which are assigned to arguments (the arguments they modify where just null pointeurs). The only function that really modifies its argument is LatticeSimplify.

- **ZPolyhedron *ZPolyhedron_Alloc (Lattice *Lat, polyhedron *P)**
  This function takes as input the lattice Lat and the polyhedron P, allocates space for,

and returns the $\mathcal{Z}$-polyhedron corresponding to the image of the polyhedron P by the lattice Lat. If the input lattice Lat is not integral it *integralises* it (i.e. the lattice of the $\mathcal{Z}$-polyhedron returned is integral).

- **ZPolyhedron *ZDomainIntersection(ZPolyhedron *Z1, ZPolyhedron *Z2)**
  Returns the $\mathcal{Z}$-domain intersection of the $\mathcal{Z}$-domains $Z1$ and $Z2$. The dimensions of domains $Z1$ and $Z2$ must be equal.

- **ZPolyhedron *ZDomainUnion (ZPolyhedron *Z1, ZPolyhedron *Z2)**
  Returns the $\mathcal{Z}$-domain union of the $\mathcal{Z}$-domain $Z1$ and $Z2$. The dimensions of the $\mathcal{Z}$-domains $Z1$ and $Z2$ must be equal.

- **ZPolyhedron *ZDomainDifference (ZPolyhedron *Z1, ZPolyhedron *Z2)**
  Returns the $\mathcal{Z}$-domain difference, $Z1$ less $Z2$. The dimensions of the $\mathcal{Z}$-domains $Z1$ and $Z2$ must be equal.

- **ZPolyhedron *ZDomainSimplify (ZPolyhedron *Z1)**
  Returns the simplified representation of the $\mathcal{Z}$-domain $Z1$. It attempts to convexize unions of polyhedra when they correspond to the same lattices and to simplify union of lattices when they correspond to the same polyhedra (this function is still under development).

- **ZPolyhedron *ZDomainImage (ZPolyhedron *Z1, Matrix *Func)**
  Returns the image of the $\mathcal{Z}$-domain $Z1$ under the *invertible, affine, rational* transformation Func. Since the Func is invertible, the matrix representing the function must be non-singular and the number of rows of the function must be equal to the number of rows in the matrix representing the lattice of $Z1$.

- **ZPolyhedron *ZDomainPreimage (ZPolyhedron *Z1, Matrix *Func)**
  Returns the preimage of the $\mathcal{Z}$-domain $Z1$ under affine transformation Func. The number of rows of matrix representing the function Func must be equal to the number of rows of the matrix representing the lattice of $Z1$.

- **ZPolyhedron *EmptyZPolyhedron (unsigned Dimension)**
  Returns the empty $\mathcal{Z}$-polyhedron of dimension $Dimension$.

- **ZPolyhedron *ZDomainCopy (ZPolyhedron *Z)**
  Returns a copy of the $\mathcal{Z}$-domain Z.

- **void ZDomainFree (ZPolyhedron *Z)**
  Frees the memory used of the $\mathcal{Z}$-domain Z.

- **Bool isEmptyZPolyhedron (ZPolyhedron *Z)**
  Returns True (1) if Z is empty, else it returns False (0).

- **Bool ZDomainIncludes (ZPolyhedron *Z1, ZPolyhedron *Z2)**
  Returns True if $Z1$ is included in $Z2$, else returns False.

- **void CanonicalForm (ZPolyhedron *Z, ZPolyhedron **CF, Matrix **Basis)**
  Returns the $\mathcal{Z}$-polyhedron Z in canonical form: CF (for the $\mathcal{Z}$-polyhedron in canonical form) and Basis (for the basis with respect to which CF is in canonical form see [QRR96b]).
  Note: This function has a meaning only when Z is a $\mathcal{Z}$-polyhedron and not a $\mathcal{Z}$-domain, even when the input is a $\mathcal{Z}$-domain, it returns a $\mathcal{Z}$-domain in which only the first $\mathcal{Z}$-polyhedron in the list is in Canonical Form

5.4 Lattice operations

In the previous section we gave the prototypes for the functions on $\mathcal{Z}$-domains. As $\mathcal{Z}$-polyhedra, lattices are closed under the operations intersection, image by an invertible affine function and preimage by an affine function and are not closed under union and difference. We do not provide a complete set of function for the union of lattice because we do not directly need it for manipulating $\mathcal{Z}$-domains, however, some of the functions below return a union of lattice.

- **Lattice *LatticeIntersection(Lattice *A, Lattice *B)**

  The function takes as input two lattices A and B and returns their intersection. The dimensions of A and B should be the same.

- **LatticeUnion *LatticeDifference (Lattice *A, Lattice *B)**

  Returns the Union of lattices that constitute the difference $A-B$. As, we have already seen the Difference of two lattices in not a lattice but a union of lattices, so this function returns a union of lattices.

- **Lattice *LatticeImage (Lattice *A, Matrix *F)**

  Returns the image of the lattice A by the *invertible, affine, rational* function F.

- **Lattice *LatticePreimage (Lattice *A, Matrix *F)**

  Returns the preimage of the lattice A by the *affine, rational* function F.
  Note: - The Image and preimage of the lattices are not the actual image and preimage in the classical notion. We deal only with integral lattices, so given a rational lattice we extract the integral lattice contained in the rational lattice.

- **Lattice *EmptyLattice (unsigned Dimension)**

  Returns the empty lattice of dimension $Dimension-1$. In our representation, a empty lattice is a lattice in which the generating vectors are zeroes and the affine part is also zero. For example a empty lattice of dimension 2 will be represented as

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- **Bool isEmptyLattice (Lattice *A)**

  Returns True if A is empty else returns False.

- **LatticeUnion *LatticeSimplify (LatticeUnion *Head)**

  Returns a simplified list of lattices, given a union of lattices.

  Note: The above function expects as input a LatticeUnion in which all the lattices are in affine Hermite form and also it modifies the input.

- **void AffineHermite (Lattice *A, Lattice **H, Lattice **U)**

  Returns the affine Hermite normal form of the affine lattice A. The unique affine Hermite form of a lattice is stored in H and the unimodular matrix corresponding to $A = H * U$ is stored in the matrix U. (see Appendix C).

5.5 Other operations

- **int SolveDiophantine (Matrix *A, Matrix **U, Vector **V)**

  This function takes a system of linear Diophantine equations in the form $A'x + b' = 0$ (here A is a `Polylib` matrix which must be constructed this way: $A = \begin{pmatrix} A' & b \\ 0 & 1 \end{pmatrix}$) and solves it. It allocates space for the result which is stored in matrix U and vector V. The procedure also returns the rank of A if a solution exists, else it returns -1. Warning, the solution matrices (U and V) are not in `Polylib` format. The general form of the solution is obtained by $x = Ut + V$ where t is a vector of free variables (For a detailed description of the algorithm and an example see Appendix B).

- **void AffineSmith (Matrix *A, Matrix **U, Matrix **V, Matrix **Delta)**

  In this function A is a `Polylib` matrix, i.e. it represents an affine function. Given such a matrix, it returns the affine Smith normal form Delta of A (see appendix C) and unimodular matrices U and V such that $A = U \times \text{Delta} \times V$. (For a detailed description of the algorithm and an example see Appendix C).

6 Conclusion

The implementation of the $\mathcal{Z}$-polyhedral library represent a major step toward the achievement of an efficient silicon or FPGA compiler. It will permit to handle partitioning, operators sharing, red/black types algorithms, etc. The current implementation has been tested on Unix, Linux and WindowsNT platform. It has also been interfaced with MMAlpha and $\mathcal{Z}$-polyhedral operations are now available with a user friendly interface (omega like). The next important step to realize will be the extension of all function of MMAlpha to $\mathcal{Z}$-polyhedra. This should be easy because, in MMAlpha, most of the operations on domains are done via calls to Domlib (the interface between MMAlpha and `Polylib`). However some points have to be carefully checked:

- The cyclic scheduling used in MMAlpha is tightly linked to the fact that polyhedra can be approximated by rational polyhedra (as it uses the fundamental theorem of duality which holds for rational polyhedra). The extension of the schedule algorithm to $\mathcal{Z}$-polyhedra has to be studied precisely.
- The fact that polyhedra and rational polyhedra were both represented by `Polylib` polyhedra led to a mix between them. In particular, in MMAlpha, the preimage of a polyhedron by any function always results in a polyhedron. This is *not* true in general. This is true for rational polyhedra but, for instance, the preimage of $\mathcal{Z}$ by $(i \rightarrow i/2)$ is a $\mathcal{Z}$-polyhedron and not a polyhedron (it is $\{2i \mid i \in \mathcal{Z}\}$).

References

[Anc91] C. Ancourt. *Génération automatique de codes de transfert pour multiprocesseurs à mémoires locales.* PhD thesis, Université de Paris VI, 1991.

[Bal95] F. Balasa. *Background Memory Allocation for Multi-Dimensional Signal Processing.* PhD thesis, Katholieke Universiteit Leuven, 1995.

[Cas78] J.W.S. Cassels. *Rational Quadratic Forms.* London Mathematical Society Monographs N°13. Academic Press, 1978.

[Che68] N.V. Chernikova. Algorithm for discovering the set of all the solution of a linear programming problem. *U.S.S.R Computational Mathematics and Mathematical Physics*, 8(6):282–293, 1968.

[Dar93] A. Darte. *Techniques de parallélisation automatique de nids de boucles.* PhD thesis, LIP ENS-Lyon, 1993.

[Fea92] P. Feautrier. Some efficient solutions to the affine scheduling problem, part I, one dimensional time. *Int. J. of Parallel Programming*, 21(5):313–348, October 1992.

[FQ88] F. Fernández and P. Quinton. Extension of chernikova's algorithm for solving general mixed linear programming problems. Technical Report 437, IRISA - Rennes (France), 1988.

[GLW98] Martin Griebl, Christian Lengauer, and Sabine Wetzel. Code generation in the polytope model. In *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques (PACT'98)*, pages 106–111. IEEE Computer Society Press, 1998.

[Gol56] A.J. Goldman. Resolution and separation theormes for polyhedral convex sets. In *Linear inequlities and related systems.* Princeton University, 1956.

[Hel94] P.C. Held. Hipars: a tool for automatic conversion of nested loop programs into single assignment programs. Technical Report Dept. Electrical Engineering, Delft University of Technology, 1994.

[Le 92] H. Le Verge. A note on Chernikova's algorithm. RR 635, IRISA, IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France, Feb 1992.

[MRTT53] T.S. Motzkin, H. Raiffa, G.L. Thompson, and R.M. Thrall. The double description method. Theodore S. Motzkin: Selected Papers, 1953. reprinted in D.Cantor, B. Gordon and B. Rothschild, eds, Birkhauser, Boston, 1983.

[NW88] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-interscience, 1988.

[Pug92] W. Pugh. The omega test: a fast and practical integer programming algorithm for depedence analysis. *Communications of the ACM*, 8:102–114, August 1992.

[QRR96a] P. Quinton, S. V. Rajopadhye, and T. Risset. Extension of the ALPHA language to recurrences on sparse periodic domains. In *Int. Conf. on Application Specific Array Processors*, 1996.

[QRR96b] P. Quinton, S. V. Rajopadhye, and T. Risset. On manipulating $\mathcal{Z}$-polyhedra. Technical Report 1016, IRISA, Rennes, France, 1996.

[QRR97] P. Quinton, S. V. Rajopadhye, and T. Risset. On manipulating z-polyhedra using a canonical representation. *Parallel Processing Letters*, 7(2):181–194, June 1997.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, New York, 1986.

[Tei93] J. Teich. *A compiler for Application-Specific Processor Arrays*. PhD thesis, Universität des Saarlandes, 1993.

[TT93] J. Teich and L. Thiele. Partitioning of processor arrays: A piecewise regular approach. *Integration: The VLSI Journal*, 14(3):297–332, Feb 1993.

[Wil93] D. Wilde. A library for doing polyhedral operations. Technical Report 785, IRISA, Rennes, France, 1993.

[Wil94] D. Wilde. The alpha language. Technical Report 827, IRISA, Rennes, France, Dec 1994.

A Example C-PROGRAM

The following is an example program written to demonstrate how the library is called from a C-Program. This program read two $\mathcal{Z}$-polyhedra and a matrix, computes the intersection of the two $\mathcal{Z}$-polyhedra and the image of the first $\mathcal{Z}$-polyhedron by the function. The first section has the code itself, the second section is the input for the program, and the last section is the output.

A.1 Program Code

```
/*
  file example.c

  compile this file this way:

       OBJS= vector.o polyhedron.o matrix.o errormsg.o NormalForms.o
             Lattice.o Zpolyhedron.o Matop.o SolveDio.o

       example:  example.c $(OBJS)
         gcc  -Wall example example.c $(OBJS)

 run it this way:

       example <input
*/

#include "types.h"
#include "ztypes.h"
#include "vector.h"
#include "matrix.h"
#include "polyhedron.h"
#include "Zpolyhedron.h"

int main ()
{
  Matrix *a, *b;
  Polyhedron *P;
  ZPolyhedron *Z1, *Z2, *Z3, *Z4;

  a = Matrix_Read ();
  b = Matrix_Read ();
  P = Constraints2Polyhedron (b, 200);
  Z1 = ZPolyhedron_Alloc (a, P);
```

```
  Matrix_Free (a);
  Matrix_Free (b);
  Domain_Free (P);

  a = Matrix_Read ();
  b = Matrix_Read ();
  P = Constraints2Polyhedron (b, 200);
  Z2 = ZPolyhedron_Alloc (a, P);

  Matrix_Free (a);
  Matrix_Free (b);
  Domain_Free (P);

  Z3 = ZDomainIntersection (Z1, Z2);
  printf ("\nZ3 = Z1 and Z2");
  ZDomainPrint (stdout, "%5d", Z3);

  a = Matrix_Read ();
  Z4 = ZDomainImage (Z1, a);
  printf ("\nZ4 = image (Z1 by a)");
  ZDomainPrint (stdout, "%5d", Z4);

  Matrix_Free (a);
  ZDomain_Free (Z1);
  ZDomain_Free (Z2);
  ZDomain_Free (Z3);
  ZDomain_Free (Z4);

  return 0;
}
```

A.2 Input file

```
3 3
2 0 0
0 3 0
0 0 1
4 4
1 0 1 -1
1 -1 0 6
1 0 -1 7
1 1 0 -2
```

```
3 3
6 0 0
0 5 0
0 0 1
4 4
1 1 0 -1
1 -1 0 3
1 0 -1 5
1 0 1 -2
3 3
2 0 0
0 3 0
0 0 1
```

A.3 Program Output

```
Z3 = Z1 and Z2
ZPOLYHEDRON: Dimension 2

LATTICE:
3 3
    6    0    0
    0   15    0
    0    0    1
POLYHEDRON Dimension:2
           Constraints:4  Equations:0  Rays:4  Lines:0
Constraints 4 4
Inequality: [    0   -5    7 ]
Inequality: [   -1    0    2 ]
Inequality: [    0    3   -2 ]
Inequality: [    1    0   -1 ]
Rays 4 4
Vertex: [    5    7 ]/5
Vertex: [   10    7 ]/5
Vertex: [    6    2 ]/3
Vertex: [    3    2 ]/3

Z4 = image (Z1 by a)
ZPOLYHEDRON: Dimension 2

LATTICE:
3 3
```

```
    4    0    0
    0    9    0
    0    0    1
POLYHEDRON Dimension:2
           Constraints:4  Equations:0  Rays:4  Lines:0
Constraints 4 4
Inequality: [    0    1   -1 ]
Inequality: [   -1    0    6 ]
Inequality: [    0   -1    7 ]
Inequality: [    1    0   -2 ]
Rays 4 4
Vertex: [    2    1 ]/1
Vertex: [    6    1 ]/1
Vertex: [    6    7 ]/1
Vertex: [    2    7 ]/1
```

B Solving Diophantine equations

In this section we explain the algorithm that we use to solve a system of linear Diophantine equations. Diophantine equations are integral equations for which only integral solution are considered. Consider a system of m linear Diophantine equations in n variables.

$$Ax + b = 0 \tag{3}$$

A is a $m \times n$ matrix (i.e. m equations, n variable) and b is a vector of order m. The general solution of this equation (if it exists) has the following form: $x = Vt + v_0$ where V is a $n \times q$ matrix, and v_0 is a vector of order n. t is a q-vector of free variables used to described all the solutions (v_0 can be seen as a particular solution of equation (3) and V describes the general solution of $Ax = 0$). If we note l the rank of matrix A, we have the relation: $q = n - l$.

In the $\mathcal{Z}$-polyhedral library, we solve the equation using the Hermite normal decomposition. In the following, we suppose without loss of generality that, if the rank of A is l, then the first l rows of A are independent vectors (it is always possible to permute the rows of A such that this condition is met). The Hermite normal decomposition of A is: $A = HU$ where H is a $m \times n$ lower triangular matrix of the following form:

$$H = \begin{pmatrix} h_{11} & & & \\ \vdots & \ddots & & 0 \\ h_{l1} & \dots & h_{ll} & \\ & \cdots & & 0 \end{pmatrix}$$

Now, we can write $Ax + b = 0$ as $HUx + b = 0$. Since U is a unimodular matrix Ux is an integral vector for all $x \in \mathcal{Z}^n$. Let us call it x': $x' = Ux$. Now, to solve equation (3), we only have to solve equation (4):

$$Hx' + b = 0 \quad . \tag{4}$$

As H is a lower triangular matrix, we can solve equation (4) using forward substitution. The only thing to check at each step is that no rational solution is encountered (i.e. each time a constant is divided by the pivot it must be an exact division). We find the first l components of x' this way: $x'_1, \dots, x'_l$, then we check that the remaining $m - l$ equations are consistent with our solution. The remaining $n - l$ components of x' can take any value (because the last $n - l$ columns of H are null). Hence, the general solution of equation (4) is (if it exists):

$$x' = \begin{pmatrix} x'_1 \\ \vdots \\ x'_l \\ t_1 \\ \vdots \\ t_{n-l} \end{pmatrix} = \begin{pmatrix} x'_1 \\ \vdots \\ x'_l \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ t_1 \\ \vdots \\ t_{n-l} \end{pmatrix}, \qquad \text{where } (t_1, \dots, t_{n-l}) \in \mathcal{Z}^{n-l}$$

Hence we get the expression of x (solution of equation (3)):

$$x = U^{-1}x' = U^{-1}\begin{pmatrix} x'_1 \\ \vdots \\ x'_l \\ 0 \\ \vdots \\ 0 \end{pmatrix} + U^{-1}\begin{pmatrix} 0 \\ \vdots \\ 0 \\ t_1 \\ \vdots \\ t_{n-l} \end{pmatrix} = V\begin{pmatrix} t_1 \\ \vdots \\ t_{n-l} \end{pmatrix} + v_0, \quad \text{where } (t_1, \ldots, t_{n-l}) \in \mathcal{Z}^{n-l}.$$

V is a $n \times n - l$ matrix composed of the $n - l$ last columns of U^{-1}. v_0 is a constant vector of order n.

For example, consider the following set of equations:

$$\begin{array}{l} 2a + 3b - 4c + d = 10 \\ a - b + c + d = 8 \\ a + d = 11 \end{array} \tag{5}$$

The matrix representing the above system of equations, which is the input to SolveDiophantine is

$$\begin{pmatrix} 2 & 3 & -4 & 1 & -10 \\ 1 & -1 & 1 & 1 & -8 \\ 1 & 0 & 0 & 1 & -11 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

System (5) has the following properties:

- the rank of the system is 3 (hence, SolveDiophantine will return 3). As there are four variables, there will be one column for matrix U,
- a particular solution of system (5) is $V = (-13, 0, -3, 24)$,
- the general solution of the linear system corresponding to (5) is: $x = (t, t, t, -t)$, $\forall t \in \mathcal{Z}$.

the object pointed by U and V after the call to SolveDiophantine are (note that there is no homogeneous dimension):

```
U   :  Matrix -> NbRows = 4, NbColumns = 1.
                p[0] = 1, p[1] = 1, p[2] = 1, p[3] = -1
V   : Vector -> Size -> 4
               p[0] = -13, p[1] = 0, p[2] = -3, p[3] = 24
```

The general solution is

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \end{pmatrix} y + \begin{pmatrix} -13 \\ 0 \\ -3 \\ 24 \end{pmatrix}$$

C Affine Smith and Hermite forms

Hermite and Smith normal form are widely used as linear decomposition. In this section we define the corresponding affine decomposition and indicate how to compute them in the homogeneous representation used in `Polylib`

C.1 Affine Hermite normal form

The affine Hermite normal form we use has been defined in [QRR96b], we recall its definition here:

Theorem 4 (affine Hermite normal form) *Given a full dimensional affine lattice (L, l), there exists a unique matrix H in Hermite normal form and a unique vector h such that such that $(L, l) = L(H, h)$, with the property $0 \leq h_i < H_{ii}\ \forall i$.*

Proof: see [QRR96b].

The proof gave a way to find the affine Hermite normal form, but it was more convenient to use the Hermite normal form already implemented by Alain Darte [Dar93]. This could be done using what we call the *homogenize/de-homogenize* transformation. For a `Polylib` matrix we call *homogenization* the transformation that consists in permuting rows and columns in such a way that the last column becomes the first and the last row become the first (other rows and columns are shifted by one). This transformation corresponds to a permutation of the indices of the homogoneous space: the homogeneous dimension (ξ) is the first dimension, not the last. *de-homogenization* performs the reverse transformation. The affine Hermite normal form for a `Polylib` matrix is computed with the following algorithm:

1. homogenize the matrix
2. compute the (linear) Hermite normal form
3. de-homogenize the matrix (change the unimodular matrix accordingly)

One can easily check that the property of theorem 4 will be met by the result of this procedure. Moreover the matrix will be in correct form for `Polylib`. Indeed the first row of the matrix once homogenized is $(1, 0, \ldots, 0)$, hence the first row will not be modified by the computation of the Hermite normal form (hopefully, no operation will add an index to the constant). Hence the last row of the result (after de-homogenization) will be $(0, \ldots, 0, 1)$.

Consider for instance the affine function $(i, j \to 2i + 4j + 1, 5i + 18)$ its affine Hermite decomposition is computed this way:

$$\begin{pmatrix} 2 & 4 & 1 \\ 5 & 0 & 18 \\ 0 & 0 & 1 \end{pmatrix} \overset{homogenize}{\rightarrow} \begin{pmatrix} 1 & 0 & 0 \\ 1 & 2 & 4 \\ 18 & 5 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 8 & 5 & 10 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 1 & 0 & -1 \end{pmatrix}$$

$$\overset{de-homogenize}{\rightarrow} \begin{pmatrix} 2 & 0 & 1 \\ 5 & 10 & 8 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

C.2 Affine Smith normal form

The affine Smith normal form could be defined the same way, however there are several possibilities for the constant coefficients. In the Hermite normal decomposition, the extension to affine matrices was natural (and as we saw, it is directly implemented with a standard Hermite decomposition). In the Smith normal decomposition, it seems more difficult to choose one decomposition among all the possible ones, and it may happen that the one we give must be modified for some specific application. We chose to define the affine Smith normal form with a property which is close to our definition of the affine Hermite normal form:

Theorem 5 affine Smith Decomposition. *If (A, a) is an $n \times n$ non-singular integer affine function, there exist unimodular matrices U and V and a n-vector v such that:*

i. $(A, a) = (U, 0)(\Delta, \delta)(V, v)$
ii. *Δ is a diagonal matrix with integral entries $\Delta_{i,i}$, δ is an integral vector such that $0 \leq \delta_i \leq \Delta_{i,i}$*
iii. $\Delta_{1,1} \mid \Delta_{2,2} \ \ldots \mid \Delta_{n,n}$

(Δ, δ) is unique and is called the affine Smith normal form of A.

Sketch of proof: This decomposition can be obtained by combining the *homogenize* transformation with the standard Smith decomposition (see the algorithm below). The uniqueness comes from the uniqueness of the standard Smith normal decomposition.

The algorithm we use for computing the affine Smith normal decomposition of an affine function is :

1. homogenize the matrix
2. compute its Smith normal decomposition
3. de-homogenize the three resulting matrices
4. perform transfers of the affine parts such that the one of U is null and the one of Δ has the property of Theorem 5.

Consider for instance the same affine function $(i, j \to 2i + 4j + 1, 5i + 18)$ its affine Smith decomposition is computed this way (the changes in the affine part are highlighted in bold characters):

$$
\begin{aligned}
\begin{pmatrix} 2 & 4 & 1 \\ 5 & 0 & 18 \\ 0 & 0 & 1 \end{pmatrix} \overset{homogenize}{\rightarrow} & \begin{pmatrix} 1 & 0 & 0 \\ 1 & 2 & 4 \\ 18 & 5 & 0 \end{pmatrix} \\
= & \begin{pmatrix} 1 & 0 & 0 \\ 1 & 2 & 1 \\ 18 & 5 & 2 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 20 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -8 \\ 0 & 0 & 1 \end{pmatrix} \\
\overset{de-homogenize}{\rightarrow} & \begin{pmatrix} 2 & 1 & 1 \\ 5 & 2 & 18 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -8 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\
= & \begin{pmatrix} 2 & 1 & \mathbf{0} \\ 5 & 2 & \mathbf{0} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & \mathbf{16} \\ 0 & 20 & \mathbf{-31} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -8 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\
= & \begin{pmatrix} 2 & 1 & 0 \\ 5 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & \mathbf{0} \\ 0 & 20 & \mathbf{9} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -8 & \mathbf{16} \\ 0 & 1 & \mathbf{-2} \\ 0 & 0 & 1 \end{pmatrix} \\
= & (U,0)(\Delta,\delta)(V,v)
\end{aligned}
$$