DISSERTATION

SOME ADVANCES IN THE POLYHEDRAL MODEL

Submitted by

Gautam Gupta

Department of Computer Science

In partial fulfillment of the requirements

for the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2010

COLORADO STATE UNIVERSITY

June 17, 2010

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY GAUTAM GUPTA ENTITLED SOME ADVANCES IN THE POLYHEDRAL MODEL BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Wim Bohm

Edwin K. P. Chong

Ross McConnell

Advisor : Sanjay Rajopadhye

Department Chair : Darrell Whitley

ABSTRACT OF DISSERTATION

SOME ADVANCES IN THE POLYHEDRAL MODEL

The polyhedral model is a mathematical formalism and a framework for the analysis and transformation of regular computations. It provides a unified approach to the optimization of computations from different application domains. It is now gaining wide use in optimizing compilers and automatic parallelization. In its purest form, it is based on a declarative model where computations are specified as equations over domains defined by "polyhedral sets". This dissertation presents two results. First is an analysis and optimization technique that enables us to simplify—reduce the asymptotic complexity—of such equations. The second is an extension of the model to richer domains called $\mathcal{Z}$-Polyhedra.

Many equational specifications in the polyhedral model have reductions—application of an associative and commutative operator to collections of values to produce a collection of answers. Moreover, expressions in such equations may also exhibit reuse where intermediate values that are computed or used at different index points are identical. We develop various compiler transformations to automatically exploit this reuse and simplify the computational complexity of the specification. In general, there is an infinite set of applicable simplification transformations. Unfortunately, different choices may result in equivalent specifications with different asymptotic complexity. We present an algorithm for the optimal application of simplification transformations resulting in an final specification with minimum complexity.

This dissertation also presents the $\mathcal{Z}$-Polyhedral model, an extension to the polyhedral model to more general sets, thereby providing a transformation framework for a larger set of regular computations. For this, we present a novel representation and interpretation of $\mathcal{Z}$-Polyhedra and prove a number of properties of the family of unions of $\mathcal{Z}$-Polyhedra that are required to extend the polyhedral model. Finally, we present value based dependence analysis and scheduling analysis for specifications in the $\mathcal{Z}$-Polyhedral model. These are direct extensions of the corre-

sponding analyses of specifications in the polyhedral model. One of the benefits of our results in the $\mathcal{Z}$-Polyhedral model is that our abstraction allows the reuse of previously developed tools in the polyhedral model with straightforward pre- and post-processing.

Gautam Gupta
Department of Computer Science
Colorado State University
Fort Collins, CO 80523
Summer 2010

ACKNOWLEDGEMENTS

I wouldn't have learnt the academic way if I hadn't met Sanjay. His practice in academia has been one of the purest that I have seen, with all its virtues and consciously accepted pitfalls. I am very grateful that I had the opportunity to observe and learn from him. Thank you, Dr. Sanjay Rajopadhye, for your faith, mentoring, patience and (fatherly) friendship with aspects extending well beyond this thesis. There is no one else with whom I would have ventured on this journey. And there are more things than I will list here for which I am thankful to you.

I want to thank Dr. Patrice Quinton, my advisor at IRISA in France, for his mentoring, support and direction.

At Colorado State University, I have had the fortune to learn from excellent teachers. I want to thank Dr. Edwin Chong, Dr. Wim Bohm and Dr. Darrell Whitley for their graduate courses. I want to further thank Dr. Bohm and Dr. Ross McConnell for their time and the extremely stimulating discussions outside the classroom. Your excitement is very contagious and helped keep the academic spirit alive in me.

I want to thank my thesis committee for their advice, patience and support. Additionally, I want to thank Dr. Louis Scharf for his advice and support as part of my committee for the first half of my thesis.

I want to thank Mr. Brijendra Sharma and DaeGon Kim for being my friend and partner when this thesis was coming to an end and I needed to take the next step. You proved me wrong—that I would not find another teacher after Sanjay. Thank you for that.

I want to thank all my colleagues at Colorado State University and IRISA, in particular Lakshminarayanan Renganarayana, Ramakrishna Upadrasta and Arnaud Carayol, for the knowledge I gained from them.

Christy Eylar from the International Office and Carol Calliham, Wayne Trzyna, Sharon Van Gorder, Kim Judith and Susan Short from the Computer Science Department helped me immensely and very patiently. I am very grateful to you for all your support.

Finally, I must thank my parents for their love and teachings. I must thank my wife and daughter for their love, understanding, support and selflessness. You make me the person I am.

DEDICATION

Many loved ones sacrificed for this to happen — my wife Smita, daughter Maahika, my brother, parents and in-laws. All my work is dedicated to them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

A significant focus of high-performance computing is towards *regular* computations. Such computations are found in dense linear algebra, in numerical solutions to PDEs in the simulation of physical phenomena, in image/video/signal processing applications, in bioinformatics, etc. Due to the enormous impact of these problems to their respective domains, each has traditionally merited specialized techniques and tools. Offering another point-of-view is the observation that most of these computations share vital properties. This enables the design of a common formalism and the development of a unified framework for analysis, transformations and optimizations of such computations. The work presented in this thesis is based on the *polyhedral model* which provides a mathematical interpretation and a program transformation framework for a class of such regular computations.

In computations, a majority of program execution time is spent in loops, thus, optimizing transformations typically focus on their efficient execution. Either informally with an operational interpretation, or formally through a mathematical abstraction, dependence analysis is crucial for any loop reordering as it provides the necessary validity conditions for program transformation. At a bird's eye view, in the polyhedral equational model, formal mathematical equations are extracted from loop nests through *value-based dependence analysis* [Fea91, Pug92, RF93, MAL93] that exposes the inherent ordering among computations in different loop iterations. On these mathematical specifications, analysis and transformations are performed eg. scheduling [Fea92a, Fea92b] and memory optimization analysis [DCD97, LF97, QR00, LLL01, TVSA01a, DSV05], etc. A final code generation step [QRW00, Bas02] provides (optimized) loop nests from these equations. Programs in the polyhedral model correspond to affine control loops (ACLs) extended with higher-level accumulation

operators (eg. summation $\sum$, product $\prod$, etc.). ACLs are an important class of loop kernels comprising significant parts of the SpecFP and PerfectClub benchmarks [BCG+03]. Furthermore, many computations can be naturally expressed as ACLs, *e.g.*, matrix multiplication, LU-decomposition, Cholesky factorization, Kalman filtering, as well as algorithms arising in image and video processing and in bio-informatics such as for RNA secondary structure prediction [LZP99]. The Berkeley View [ABC+06], a study by application, compiler and hardware specialists that outlines the most important problem domains for parallel platforms lists thirteen motifs from the entire field of computing that require attention. The generality of the polyhedral model can be seen in that it subsumes most of three of these motifs ("structured grids", "dense matrix" and "dynamic programming") and shares some concerns with "graphical models".

The polyhedral model abstracts computations in the different iterations of a loop program as integer points in multi-dimensional polyhedra[1] (and thus derives its name). Individual computations at these points are treated as atomic operations, and producer-consumer relationships determine dependences between their evaluation. The dependence of the consumer of values on the producer can be expressed as an affine function for polyhedral specifications. With such dependences and the affine boundary constraints of polyhedra, we can employ the power of linear algebra to design sophisticated static analyses and transformations. Across more than two decades, various techniques targeting automatic parallelization, hardware synthesis, program verification, memory optimization, and code optimization and generation have been designed for *polyhedral specifications*. Furthermore, to provide a convenient platform for devising analyses and program transformations on polyhedral specifications, an equational language was designed and implemented [Mau89]. The context of the research presented in this thesis is that of a compiler engine for mathematical specifications.

In addition to the formal/mathematical representation of polyhedral specifications, the geometrical nature of polyhedra provides us another point of view for these computations. As a result, we have three equivalent representations for scientific specifications. A major feature of the the polyhedral model is that we can simultaneously operate on these three abstractions and easily migrate results and interpretation from one to the other.

Example 1 Equivalent representations for regular, polyhedral specifications

---

[1]Informally, a polyhedron is a mathematical object with "flat" surfaces. These may be viewed as generalized multi-dimensional arrays, the bounds of which are given by arbitrary affine inequalities.

Representation 1: Mathematical Equation (and, with some syntactic sugar, a program in the polyhedral equational language)

$$Y_i = \sum_{j=1}^{i} \sum_{k=1}^{i} (A_{i,k} \times B_{k,j})$$

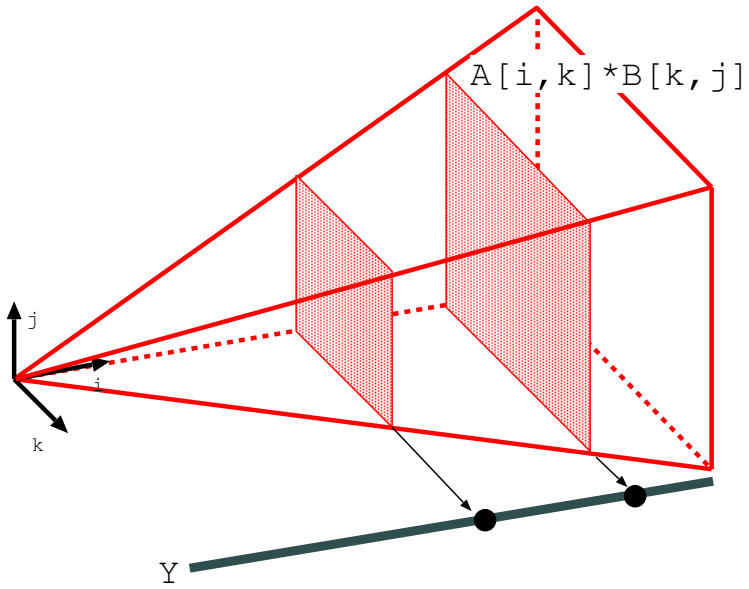
where the free index i belongs to the set $\{i | 1 \leq i \leq n\}$. The specification is complete if we give the equation along with the *domain* where it is defined. Henceforth, we will simply provide specifications as a list of equations along with the domains of definition of free indices, eg.

$$Y_i = \sum_{j=1}^{i} \sum_{k=1}^{i} (A_{i,k} \times B_{k,j}) \text{ for } \{i | 1 \leq i \leq n\} \tag{1.1}$$

Representation 2: Loop Nest

```
for i = 1 to n {
   Y[i] = 0;
   for j = 1 to i
      for k = 1 to i
         Y[i] += A[i,k] * B[k,j];
}
```

Representation 3: Geometric Interpretation



A subtle point here is that, even though we only see a double summation, this is a 3-D specification corresponding to the triply nested loop since there are three indices (i, j and k) in the scope of the computation $(A_{i,k} \times B_{k,j})$

The contributions in this thesis are broadly along two dimensions

1. Optimization and efficient execution of accumulations (called *reductions* in the polyhedral model).

2. Extensions to more general mathematical sets ($\mathcal{Z}$-polyhedra), thereby providing a transformation framework for a larger set of loop nests.

We will now illustrate and motivate these contributions.

Simplifying Reductions A reduction is an associative and commutative binary operator applied to a collection of values to produce a collection of results (eg. summation $\sum$, product $\prod$, etc.). Commutativity and associativity of the operator guarantees that we may change the order of the accumulation and obtain a semantically equivalent specification.

We will now elaborate the technique of simplifying reductions on the loop kernel provided in the example above. One can observe that it has cubic complexity. Such a deduction, however, is quite difficult for a compiler. The ability to do anything meaningful beyond that observation by a compiler has, typically, not even been considered. In this thesis, we will present a static analysis that identifies the asymptotic complexity of polyhedral specifications. Then, we provide a *simplification transformation* that reduces the complexity of accumulations through the sharing of partially computed values.

To elaborate, let us perform the following sequence of transformations to the equation in (1.1). In chapter 3, we will present this technique in much more detail. Recall that the equation in consideration is

$$Y_i = \sum_{j=1}^{i} \sum_{k=1}^{i} (A_{i,k} \times B_{k,j}) \text{ for } \{i | 1 \leq i \leq n\}$$

The following steps will optimize the complexity of this specification.

1. Change of the order of summation, since addition is associative and commutative.

$$Y_i = \sum_{k=1}^{i} \sum_{j=1}^{i} A_{i,k} \times B_{k,j} \text{ for } \{i | 1 \leq i \leq n\}$$

2. Distribution of multiplication over summation. Note, $A_{i,k}$ is independent of j, the index of the inner summation, thus can be treated as a constant for the summation along j.

$$Y_i = \sum_{k=1}^{i} \left(A_{i,k} \times \sum_{j=1}^{i} B_{k,j} \right) \text{ for } \{i | 1 \leq i \leq n\}$$

3. Introduction of the variable $Z_{i,k}$ to denote the result of the inner summation $\sum_{j=1}^{i} B_{k,j}$.

$$\begin{aligned} Y_i &= \sum_{k=1}^{i} A_{i,k} \times Z_{i,k} \text{ for } \{i | 1 \leq i \leq n\} \\ Z_{i,k} &= \sum_{j=1}^{i} B_{k,j} \text{ for } \{i,k | 1 \leq i \leq n, 1 \leq k \leq i\} \end{aligned}$$

Note that we still have a system of equations that has cubic complexity for the evaluation of $Z_{i,k}$. However, once $Z_{i,k}$ is computed, the evaluation for Y_i requires quadratic time. This can be seen in the following implementation of the equations.

```
for i = 1 to n
   for k = 1 to i {
      Z[i,k] = 0;
      for j = 1 to i
         Z[i,k] += B[k,j];
   }


for i = 1 to n {
   Y[i] = 0;
   for k = 1 to i
      Y[i] += A[i,k] * Z[i,k];
}
```

4. Detection of a scan. Our simplification occurs when we identify that, by pulling the last term out, $Z_{i,k} = \sum_{j=1}^{i} B_{k,j}$ can be written as

$$Z_{i,k} = B_{k,i} + \sum_{j=1}^{i-1} B_{k,j}$$

where for $1 \leq k < i$, $\sum_{j=1}^{i-1} B_{k,j}$ has already been calculated and stored in $Z_{i-1,k}$.
Thus, we may express $Z_{i,k}$ as a function of $Z_{i-1,k}$, essentially simplifying an $O(n)$ computation (for a single value of Z) to constant time.

$$Z_{i,k} = B_{k,i} + Z_{i-1,k} \text{ for } 1 \leq k < i$$

5. Our final simplified specification is

$$
\begin{aligned}
Y_i &= \sum_{k=1}^{i} A_{i,k} \times Z_{i,k} \text{ for } \{i | 1 \leq i \leq n\} \\
Z_{i,k} &= \begin{cases} 1 \leq k < i & B_{k,i} + Z_{i-1,k} \\ k = i & \sum_{j=1}^{i} B_{k,j} \end{cases} \text{ for } \{i, k | 1 \leq i \leq n, 1 \leq k \leq i\}
\end{aligned}
$$

The code for this specification has a doubly nested loop and its complexity can be verified to be $\Theta(n^2)$.

```
for i = 1 to n {
   Z[i,i] = 0;
   for j = 1 to i
      Z[i,i] += B[i,j];
   for k = 1 to i-1
      Z[i,k] = Z[i-1,k] + B[k,i];
   Y[i] = 0;
   for k = 1 to i
     Y[i] += A[i,k] * Z[i,k];
}
```

In our characterization, there is an infinite space of such transformations, essentially corresponding to the sharing of (temporary) values along different vectors in a multi-dimensional linear space. Often partially optimized specifications can be further simplified. In general, these transformations may result in final specifications with differing complexities. Thus, a valid concern is how one may pick a good sequence of simplification transformations to apply. To address this issue, we also provide an algorithm with a proof of optimality assuring that the suggested sequence of transformations will yield a resultant specification with minimum complexity.

All the mathematical equations in the derivation above are polyhedral specifications and programs in our equational language. The simplification can thus be implemented as an optimizing transformation in a compiler for polyhedral specifications.

The $\mathcal{Z}$-polyhedral Model The transformation presented above involved a sequence of steps, each operating on a valid equational specification in the polyhedral model. Therefore, each individual step must also produce a valid specification (with the exception of the final code generation

to an imperative program). This is possible only if the family of polyhedral specifications is closed under the family of transformations. Programs in the polyhedral model consist of

- variables representing collections of values defined at integer points in polyhedral domains (unions of polyhedra), and
- affine dependences between computations.

Thus, the constraints on valid program transformations may be expressed as

1. The family of polyhedral domains is closed under program transformations.
2. The family of affine functions is closed under program transformations.

With these constraints, program transformations in the polyhedral model essentially become combinations of

- set operations (viz. union, intersection, difference) on polyhedral domains, and
- preimage and (restricted) image by affine functions.

If we generalize domains and/or the family of functions such that these two operations maintain closure, we would be able to retain existing techniques, yet generalize the polyhedral model. The second contribution of this thesis is the $\mathcal{Z}$-polyhedral model, an extension of the polyhedral model, that addresses its following limitations. Unsurprisingly, these limitations pertain to either the strictness of polyhedra, or the nature of affine functions. These are,

- Loop programs with a non-unit stride fall outside the scope of the polyhedral model. This is an important class of programs [Ram95, LP94, Xue94, FLVG95] arising in various situations, such as the red-black SOR computation for solving partial differential equations. More importantly, tiled programs generally encode information of the tile origins and lattice. Any subsequent analysis of such programs for the purpose of porting to different architectures requires the more general dependence analysis presented here.

  For similar mathematical reasons, *non-unimodular* transformations are also disallowed in the polyhedral model. Non-unimodular transformations are required for the derivation of parallel architectures with *periodic* processor activity, such as *multi-rate arrays* [LR94] and bidirectional systolic arrays.

- The polyhedral model does not satisfy closure under image by arbitrary affine functions. Thus, reduction operations with arbitrary projections cannot be expressed in the model. Also, the use-set of a dependence (requiring the image of a domain by the dependence function) cannot be precisely characterized.

As a simple example, consider the following loop program

```
for i = 1 to N
  A[i] = (i%2==0 ? f(A[i/2]) : 0);
```

where `f` is an arbitrary point-wise function.

This program exhibits a dependence pattern that is richer than the affine dependences of the polyhedral model. In other words, it is impossible to write an equivalent program in the polyhedral equational model that can perform the required computation. One may consider replacing the variable A by two variables X and Y corresponding to the even and odd points of A. However, the definition of X now becomes $X[2i] = X[i]$ and $X[2i-1] = Y[i]$. The domain of evaluation of these two branches in the definition of X have *holes*, thus cannot be written as a (union of) polyhedra. This mathematical set is called the $\mathcal{Z}$-polyhedra which is the intersection of an integer polyhedron and an *affine integer lattice*.

In addition to expressibility, a model over $\mathcal{Z}$-polyhedra also enables more sophisticated analyses and transformations by providing more information in the specifications *viz.* pertaining to lattices. Consider the following loop

```
for i = 1 ... n
   if ((i%2==0)||(i%3==0)) X[i] = f(X[i-1]);
```

where `f` is an arbitrary point-wise function.

Note that the assignment to `X[i]` is not executed at all loop iterations. For any integer j, when $i = 6j + 1$ or $i = 6j + 5$, the statement guard `((i%2==0)||(i%3==0))` fails. However, any polyhedral approach is incapable to factor this information, therefore, conservatively assumes that the statement `X[i] = f(X[i-1])` is executed at all loop iterations. As all statement instances depend on the value obtained from the previous iteration, this abstraction clearly imposes a sequential execution order. However, if we were to use the information about the "holes", we may obtain the constant-time parallelization of this $\Theta(n)$ loop given below.

```
forall i = 2 ... n step 6
   X[i] = f(X[i-1]);
forall i = 6 ... n step 6
   X[i] = f(X[i-1]);
forall i = 3 ... n step 6
   X[i] = f(X[i-1]);
forall i = 4 ... n step 6
   X[i] = f(X[i-1]);
```

The iteration spaces of the each of the four loops is similar to the original loop, but restricted to certain periodic hops. More precisely, it is the intersection of the original polyhedron $i \in \{1, \ldots, n\}$ and a lattice (with a periodicity of 6). This parallelization requires

1. (value-based) dependence analysis to determine precise producer-consumer relationships,
2. scheduling analysis to determine the space-time mapping of computations and
3. code generation of equations on variables defined over $\mathcal{Z}$-polyhedral domains.

Prior to the results presented in this thesis, only code-generation had been presented for $\mathcal{Z}$-polyhedral domains [Bas04].

In chapter 4, we present the foundations of a simple and unified solution to the limitations of the polyhedral model through the extension of the equational language to domains that are unions of $\mathcal{Z}$-polyhedra, providing proofs of the required closure properties for $\mathcal{Z}$-polyhedral domains. We present the $\mathcal{Z}$-polyhedral model using a new representation and interpretation for $\mathcal{Z}$-polyhedra and the associated family of dependences (called *affine lattice functions*) that are a generalization of affine functions. In addition to the required closure properties needed to extend the polyhedral model, we also show that $\mathcal{Z}$-polyhedra are closed under arbitrary images by affine lattice functions. As a corollary, we prove that unions of *Linearly Bound Lattices* (LBLs)[TT93] that are mathematical sets obtained from arbitrary affine images of polyhedra, widely assumed to be a richer class of domains, are in fact equivalent to unions of $\mathcal{Z}$-polyhedra. This also shows that Presburger sets are equivalent in expressivity to unions of $\mathcal{Z}$-polyhedra.

Since the family of $\mathcal{Z}$-polyhedral domains and corresponding affine lattice functions are strict supersets of the family of polyhedral domains and affine functions respectively, the $\mathcal{Z}$-polyhedral model is a strict generalization of the polyhedral model. An important feature of our work

on the $\mathcal{Z}$-polyhedral model and on analyses for $\mathcal{Z}$-polyhedral specifications is that tools and techniques developed for the polyhedral model can be easily reused in the $\mathcal{Z}$-polyhedral model with straightforward pre- and post-processing.

Value-based Dependence analysis in the $\mathcal{Z}$-polyhedral Model In order to ensure the semantic validity of transformations (such as the space-time mapping corresponding to a schedule) that change their execution order it is essential that the transformations respect dependences between statement instances. True dependences between different computations in a mathematical or equational specification are relatively easy to derive. They are given as affine functions between consumers and producers of values in multi-dimensional space. However, to extract equations from loop nests in an imperative language, we need to analyze code that has memory effects including false dependences resulting from write-after-write and write-after-read. True dependences are also non-trivial to derive from such loop nests since values at a given memory location may be written by multiple statements, most often at different iterations. To derive the dependence, we need to find the last producer before the value is consumed.

The extraction of dependences from loop nests is known as *value-based dependence analysis*. Feautrier showed how ACLs can be analyzed to produce exact dependence relations. These result in specifications in the polyhedral model. In chapter 5, we present dependence analysis for a more general class of programs than ACLs. These programs contain:

1. loops with non-unit stride
2. modulo operations, integer divisions, max and min in loop bounds, guards and memory access functions, and
3. existential variables (implemented with flags) within guards.

In other words, this analysis can be applied to loop programs that have statements defined over Presburger sets. The result of this dependence analysis are specifications in the $\mathcal{Z}$-polyhedral model.

Scheduling in the $\mathcal{Z}$-polyhedral Model In chapter 6, we present scheduling analysis for the automatic parallelization of more general programs in the $\mathcal{Z}$-polyhedral model. Our key contributions are

1. deriving precedence (causality) constraints for programs written in the $\mathcal{Z}$-polyhedral model,

2. formulation of an *integer linear program* to obtain a schedule that minimizes latency, and

3. the generalization of the scheduling problem to multi-dimensional schedules (i.e., schedules where the "time instants" are multidimensional vectors under lexicographic order, corresponding naturally to nested loops).

An important feature of our formulation is that it finds schedules that can be used to construct a program transformation to obtain an equivalent specification in the $\mathcal{Z}$-polyhedral model. This has been a major drawback of previous methods that involved space-time transformations with rational coefficients.

The remainder of this thesis is organized as follows. In the next chapter, we present the mathematical background on polyhedra, affine functions, lattices and $\mathcal{Z}$-polyhedra . Along with mathematical background, we will also describe the equational language for high-level specifications. Then we will present our technical results in subsequent chapters on simplifying reductions, the $\mathcal{Z}$-polyhedral model, value-based dependence analysis in the $\mathcal{Z}$-polyhedral model, and finally scheduling in the $\mathcal{Z}$-polyhedral model. Finally, we discuss future and related work, and present our conclusions.

Chapter 2

Mathematical Background

Here, we will review some mathematical background on matrices and describe the mathematical objects and terminology used in the following chapters. As a convention, we denote matrices with the upper-case letters and vectors with the lower-case. All our matrices and vectors have integer elements. We denote the identity matrix by I.

We use the following concepts and properties of matrices

- An *affine function* f is of the form $(z \to Tz + t)$ where T is an $n \times m$ matrix and t is an n-vector.
- The kernel of a matrix T, written as $\ker(T)$ is the set of all vectors z such that $Tz = 0$. We define $\ker(f)$, the kernel of an *affine function* $f \equiv (z \to Tz + t)$, as the kernel of its linear part $\ker(T)$.
- f^{-1} denotes relational inverse.
- The relations $\supset$ and $\subset$ are strict and $\supseteq$ and $\subseteq$ are non-strict. The terms "superset" and "subset" denote the non-strict relations.
- The column (resp. row) *rank* of a matrix T is the maximal number of linearly independent columns (resp. rows) of T.
- A matrix is *unimodular* if it is square and its determinant is either 1 or -1.
- Two matrices L and L' are said to be *column equivalent* or *right equivalent* if there exists a unimodular matrix U such that $L = L'U$.
- A unique representative element in each set of matrices that are column equivalent is the one in *Hermite Normal Form* [Her51].

Definition 1 *An $n \times m$ matrix H with column rank d is in* Hermite Normal Form *(*HNF*), if*

1. *for columns $2, \ldots, d$, the first non-zero element is positive, and below the first positive element for the previous column.*
   $\exists i_1, \ldots, i_d, 1 \leq i_1 < \ldots < i_d \leq n$: $H_{i_j,j} > 0$.
   The first non-zero element of the first column is also positive
   $\forall i, j | 1 \leq j \leq d, 1 \leq i < i_j$: $H_{i,j} = 0$.
2. *The first positive entry in columns $1, \ldots, d$ is the maximal entry on its row. All elements are non-negative in this row.*
   $\forall j, l | 1 \leq l < j \leq d$: $0 \leq H_{i_j,l} < H_{i_j,j}$.
3. *columns $d+1, \ldots m$ are zero-columns*
   $\forall i, j | d+1 \leq j \leq m, 1 \leq i \leq n$: $H_{i,j} = 0$

$$\begin{pmatrix} \blacktriangledown & 0 & 0 & 0 & 0 \\ \bigstar & 0 & 0 & 0 & 0 \\ \blacktriangle & \blacktriangledown & 0 & 0 & 0 \\ \bigstar & \bigstar & 0 & 0 & 0 \\ \blacktriangle & \blacktriangle & \blacktriangledown & 0 & 0 \\ \bigstar & \bigstar & \bigstar & 0 & 0 \\ \bigstar & \bigstar & \bigstar & 0 & 0 \\ \blacktriangle & \blacktriangle & \blacktriangle & \blacktriangledown & 0 \\ \bigstar & \bigstar & \bigstar & \bigstar & 0 \\ \bigstar & \bigstar & \bigstar & \bigstar & 0 \end{pmatrix}$$

A template of a matrix in HNF is provided above. In the template, both ▼ and ▲ are non-negative elements. ▼ denotes the maximum element in the corresponding row, ▲ denotes elements that are not the maximum element and ★ denotes any integer.

For every matrix A, there exists a unique matrix H that is in HNF and column equivalent to A *i.e.*, there exists a unimodular matrix U such that $A = HU$. Note that the provided definition of the Hermite normal form does not require the matrix A to have full row rank.

There is a related normal form called the *Smith normal form* [Smi61].

Definition 2 *An $n \times m$ matrix S with rank d is in* Smith Normal Form *(SNF) , if*

1. *S is a diagonal matrix.*
2. *$1 \leq i \leq d$: $S_{i,i} > 0$.*
3. *$1 \leq i \leq d-1$: $S_{i,i}$ divides $S_{i+1,i+1}$.*
4. *$d+1 \leq i \leq \min(n, m)$: $S_{i,i} = 0$*

For every matrix, A, there exists a unique matrix S that is in SNF such that $A = VSU$ where V and U are unimodular matrices.

2.1 Mathematical Objects

The mathematical objects introduced here are used to abstract iteration domains and dependences between computations. Next we will present a language for the specification of equations over these domains and with such dependences. The basic mathematical objects manipulated by our methods are integer polyhedra.

2.1.1 Integer Polyhedra

An *integer polyhedron*, $\mathcal{P}$ is a subset of $\mathbb{Z}^n$ the elements of which satisfy a finite number of affine inequalities (also called affine constraints or just constraints when there is no ambiguity) with integer coefficients. We follow the convention that the affine constraint c_i is given as $(a_i^T z + \alpha_i \geq 0)$ where $z, a_i \in \mathbb{Z}^n, \alpha_i \in \mathbb{Z}$. The integer polyhedron, $\mathcal{P}$, satisfying the set of constraints $\mathcal{C} = \{c_1, \ldots, c_b\}$ is often written as $\{z \in \mathbb{Z}^n | Qz + q \geq 0\}$ where $Q = (a_1 \ldots a_b)^T$ is an $b \times n$ integer matrix[1] and $q = (\alpha_1 \ldots \alpha_b)^T$ is an integer b-vector.

Example 2 *Consider the equation*

$$Y_i = \sum_{j=1}^{i} \sum_{k=1}^{i} A_{i,k} \times B_{k,j}, 1 \leq i \leq 10$$

The domain of the variables Y, A and B are, respectively, the sets $\{i | 1 \leq i \leq 10\}$, $\{i, k | 1 \leq i \leq 10, 1 \leq k \leq i\}$ and $\{k, j | 1 \leq k \leq 10, 1 \leq j \leq 10\}$. These sets are polyhedra and deriving the aforementioned representation simply requires us to obtain, through elementary algebra, all affine constraints of the correct form, yielding $\{i | i - 1 \geq 0, -i + 10 \geq 0\}$, $\{i, k | i - 1 \geq 0, -i + 10 \geq 0, k - 1 \geq 0, i - k \geq 0\}$ and $\{k, j | k - 1 \geq 0, -k + 10 \geq 0, j - 1 \geq 0, -j + 10 \geq 0\}$, respectively. Nevertheless, these are less intuitive and in our presentation, we will not conform to the precise formalities of representation.

A subtle point to note here is that elements of polyhedral sets are tuples of integers. The index variables i, j and k are simply place holders and can very well be substituted by other unused indices. For example, the domain of B could as well be specified by $\{i, j | 1 \leq i \leq 10, 1 \leq j \leq 10\}$.

We shall use the following notation and properties of integer polyhedra and affine constraints.

- For any two coefficients β and β' where $\beta, \beta' \geq 0$ and $\beta + \beta' = 1$, $\beta z + \beta' z'$ is said to be a *convex combination* of z and z'. If z and z' are two iteration points in an integer polyhedron, $\mathcal{P}$, then any convex combination of z and z' that has all integer elements is also in $\mathcal{P}$.
- The *inverse* of the constraint $(a^T z + \alpha \geq 0)$ is $(a^T z + \alpha \leq 0)$. Note that the domains corresponding to a constraint and its inverse are non-disjoint.

[1] When Q and/or q is rational, we can appropriately multiply the constraints to get integer elements

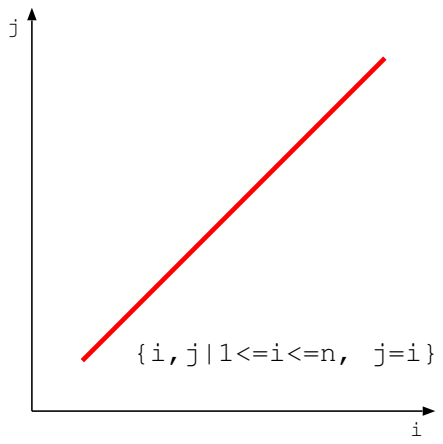


Figure 2.1: Number of dimensions of the polyhedron = 1, Number of dimensions of space = 2, The number of dimensions of the polyhedron is one less than the number of dimensions of space as it has one linearly independent saturated constraint (viz, {j = i}).

- The constraint $c \equiv (a^T z + \alpha \geq 0)$ of $\mathcal{P}$ is said to be *saturated iff* $c' \cap \mathcal{P} = \mathcal{P}$ where c' is the inverse of c. Alternatively, the property $(a^T z + \alpha = 0) \cap \mathcal{P} = \mathcal{P}$ holds for saturated constraints $c \equiv (a^T z + \alpha \geq 0)$.

- The *lineality space* of $\mathcal{P}$ is defined as the linear part of the largest affine subspace contained in $\mathcal{P}$. It is the largest linear subspace along which $\mathcal{P}$ extends indefinitely. It is given by the kernel of Q or ker(Q).

- The *context* of $\mathcal{P}$ is defined as the linear part of the smallest affine subspace that contains $\mathcal{P}$. If the saturated constraints of $\mathcal{P}$ in $\mathcal{C}$, are the rows of $\{Q_0 z + q_0 \geq 0\}$, then it is ker(Q_0). Specifically, if there are no equalities in the constraint representation of $\mathcal{P}$, then $\mathcal{P}$ has context that is the universe.

- The number of dimensions of a polyhedron $\mathcal{P}$, written as $\#(\mathcal{P})$, is the number of dimensions of space (*viz.* n) minus the number of its linearly independent saturated constraints. For an example, refer to Figure 2.1.

2.1.2 Parameterized Family of Integer Polyhedra

In Eq. (1.1), repeated below for convenience

$$Y_i = \sum_{j=1}^{i} \sum_{k=1}^{i} (A_{i,k} \times B_{k,j}) \text{ for } \{i | 1 \leq i \leq n\}$$

the domain of Y is given by the set $\{i|1 \leq i \leq n\}$. Intuitively, the variable n is seen as a size parameter that indicates the problem instance under consideration. If we associate every iteration point in the domain of Y with the appropriate problem instance, the domain of Y would be described by the set $\{n, i|0 \leq i \leq n\}$. Thus, a parameterized family of integer polyhedra (or *parameterized integer polyhedra*) is an integer polyhedron where some indices are interpreted as size parameters. We define $\mathcal{L}'$ as the linear space spanned by *program indices* (indices other than size parameters). For the domain of Y, the space spanned by program indices is $\{i\}$.

We may also interpret a parameterized family of integer polyhedra as the (possibly infinite) set of integer polyhedra obtained by exhaustively assigning valid constant values to all size parameters. The elements of a parameterized integer polyhedra are called its instances. We identify size parameters by omitting them from the index list in the set notation of a domain eg. $\{i|1 \leq i \leq n\}$.

2.1.3 Affine Images of Integer Polyhedra (LBLs)

Consider the integer polyhedron $\mathcal{P} = \{z \in \mathbb{Z}^m | Qz + q \geq 0\}$ and the affine function $f \equiv (z \rightarrow Tz + t)$ where Q and T are $b \times m$ and $n \times m$ matrices respectively and q and t are a b-vector and n vector respectively. The image of $\mathcal{P}$ by f is of the form $\{Tz + t | Qz + q \geq 0, z \in \mathbb{Z}^m\}$. Such sets are also called linearly bound lattices (or LBLs) by Teich and Thiele, who first studied such iteration spaces [TT93]. The family of LBLs is a strict superset of the family of integer polyhedra. Clearly, every integer polyhedra is an LBL with $T = I$ and $t = 0$. However, not all LBLs are integer polyhedra as we will see in Chapter 4.

2.1.4 Affine Lattices

Often, the domain over which an equation is specified, or the iteration space of a loop program, does not contain every integer point that satisfies a set of affine constraints.

Example 3 *Consider the red-black* SOR *for the iterative computation of partial differential equations as described in a standard text* [WA99]. *Iterations in the (i, j)-plane are divided into "red" points and "black" points, similar to the layout of squares in a chess board. First, black points (at even $i + j$) are computed using the four neighbouring red points (at odd $i + j$), then the red points are computed using its four neighbouring black points. These two phases are repeated until convergence. Introducing an additional dimension, k to denote the iterative application of the two phases, we get the following equation*

```
for k = 1 to m {
   // Copying boundary values from the previous iteration
   forall i = 0 to n {
      C[i,0,k] = C[i,0,k-1];
      C[i,n,k] = C[i,n,k-1];
   }
   forall j = 1 to n-1 {
      C[0,j,k] = C[0,j,k-1];
      C[n,j,k] = C[n,j,k-1];
   }
   // Stencil Computation
   start = 1; // int to hold the starting point for j
   forall i = 1 to n-1 {
      forall j = start to n-1 step 2
         C[i,j,k] = 0.25 * ( C[i-1,j,k-1] + C[i+1,j,k-1] +
                             C[i,j-1,k-1] + C[i,j+1,k-1] );
      start = 3 - start;
   }
   start = 2;
   forall i = 1 to n-1 {
      forall j = start to n-1 step 2
         C[i,j,k] = 0.25 * ( C[i-1,j,k] + C[i+1,j,k] +
                             C[i,j-1,k] + C[i,j+1,k] );
      start = 3 - start;
   }
}
```

Figure 2.2: Imperative loop implementing the red-black SOR.

$$C_{i,j,k} = \begin{cases} i+j \text{ even}, 1 \le i < n, 1 \le j < n: & \frac{1}{4}(C_{i-1,j,k-1} + C_{i+1,j,k-1} + \\ & C_{i,j-1,k-1} + C_{i,j+1,k-1}) \quad \textit{// black} \\ i+j \text{ odd}, 1 \le i < n, 1 \le j < n: & \frac{1}{4}(C_{i-1,j,k} + C_{i+1,j,k} + \\ & C_{i,j-1,k} + C_{i,j+1,k}) \quad \textit{// red} \\ i = 0 | i = n | j = 0 | j = n: & C_{i,j,k-1} \end{cases} \tag{2.1}$$

where the domain of C is $\{i,j,k | 0 \le i \le n, 0 \le j \le n, 0 \le k \le m\}$, n, m are size parameters and $C[i,j,0]$ forall i,j is given as input. The imperative loop nest that implements this equation is given in figure 2.2.

We see that the first (resp. second) branch of the equation is not defined over all iteration points in $\{i,j,k | 1 \le i < n, 1 \le j < n, 1 \le k \le m\}$ but only over points that additionally satisfy $(i+j)$ `mod` $2 = 0$ (resp. $(i+j)$ `mod` $2 = 1$). For the first branch of the equation, this additional constraint is satisfied by iteration points that can be expressed as integer linear combinations of the vectors $\left\{ \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 2 \end{pmatrix} \right\}$. The vectors $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 2 \end{pmatrix}$ are the generators of the lattice on which these iteration points lie.

In the second branch of the equation, the additional constraint is satisfied by iteration points that can be expressed as the following affine combination.

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} i' + \begin{pmatrix} 0 \\ 2 \end{pmatrix} j' + \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Formally, the lattice generated by a matrix L is the set of all integer linear combinations of the columns of L. If the columns of a matrix are linearly independent, they constitute a *basis* of the generated lattice. The lattices generated by two dimensionally identical matrices are equal *iff* the matrices are column equivalent. In general, the lattices generated by two arbitrary matrices are equal *iff* the submatrices corresponding to the non-zero columns in their Hermite normal forms are equal.

In the previous example, we need a generalization of the lattices generated by a matrix, additionally allowing offsets by constant vectors. These are called *affine lattices*. An affine lattice is a subset of $\mathbb{Z}^n$ of the following form $\{Lz+l|z \in \mathbb{Z}^m\}$ where L and l are an $n \times m$ matrix and n-vector respectively. We call z the *coordinates* in the particular representation of the affine lattice. A representation of an affine lattice will be denoted by $\mathcal{L}$.

The affine lattices $\{Lz + l|z \in \mathbb{Z}^m\}$ and $\{L'z' + l'|z' \in \mathbb{Z}^{m'}\}$ are equal *iff* the lattices generated by L and L' are equal and $l' = Lz_0 + l$ for some constant vector $z_0 \in \mathbb{Z}^m$. The latter requirement basically enforces that the offset of one lattice lies on the other lattice. An affine lattice represented by $\{Lz + l|z \in \mathbb{Z}^m\}$ is in canonical form if

1. L has full column rank, and
2. $\begin{pmatrix} 1 & 0 \\ l & L \end{pmatrix}$ is in Hermite normal form.

2.1.5 $\mathcal{Z}$-Polyhedra

A $\mathcal{Z}$-polyhedron, $\mathcal{ZP}$, is the intersection of an integer polyhedron and an affine lattice. Recall the set of iteration points defined by the second branch of the equation for the red-black SOR. As we saw above, these iteration points lie on an affine lattice in addition to satisfying a set of affine constraints. The set of these iteration points is precisely a $\mathcal{Z}$-polyhedron. When the affine lattice is the canonical lattice, $\mathbb{Z}^n$, a $\mathcal{Z}$-polyhedron is also an integer polyhedron. However, we cannot always express a $\mathcal{Z}$-polyhedron as a union of integer polyhedra, thus, the family of unions of $\mathcal{Z}$-polyhedra strictly contains the family of unions of integer polyhedra.

Given the integer polyhedron $\{y \in \mathbb{Z}^n|Qy + q \geq 0\}$ and the affine lattice $\{Lz + l|z \in \mathbb{Z}^m\}$ where Q and L are $b \times n$ and $n \times m$ matrices respectively and q and l are a b-vector and n-vector

respectively such that both the integer polyhedron and the affine lattice lie in $\mathbb{Z}^n$, we have as their intersection, the following $\mathcal{Z}$-polyhedron.

$$\mathcal{ZP} = \{y \in \mathbb{Z}^n | Qy + q \geq 0\} \cap \{Lz + l | z \in \mathbb{Z}^m\}$$

The integer polyhedron and the affine lattice intersect when $y = Lz+l$. Replacing y by $Lz+l$, we get

$$\begin{aligned} \mathcal{ZP} &= \{Lz + l | Q(Lz + l) + q \geq 0, z \in \mathbb{Z}^m\} \\ &= \{Lz + l | (QL)z + (q + Ql) \geq 0, z \in \mathbb{Z}^m\} \\ &= \{Lz + l | Q'z + q' \geq 0, z \in \mathbb{Z}^m\} \end{aligned} \tag{2.2}$$

where $Q' = QL$ and $q' = q + Ql$. The final set representation of a $\mathcal{Z}$-polyhedron is in the form of an affine image of an integer polyhedron, thus, all $\mathcal{Z}$-polyhedra are LBLs. As convention, in the representation of a $\mathcal{Z}$-polyhedron as an affine image of an integer polyhedron as in (2.2), we will call L and l the generator and the offset respectively.

2.2 Equational Language

Consider Eq. (1.1), repeated here for convenience.

$$Y_i = \sum_{j=1}^{i} \sum_{k=1}^{i} (A_{i,k} \times B_{k,j}) \text{ for } \{i | 1 \leq i \leq n\}$$

In chapter 1, we saw that its simplification transforms the accumulation expression in the *rhs* of the equation. Quite normally, one would expect the simplification technique to be able to decrease the asymptotic complexity of the following equation as well.

$$Y_i = 5 + \sum_{j=1}^{i} \sum_{k=1}^{i} (A_{i,k} \times B_{k,j}) \text{ for } \{i | 1 \leq i \leq n\} \tag{2.3}$$

Generalizing, one could reasonably expect that the simplification technique would be able to reduce the complexity of evaluation of the accumulation sub-expression, $\sum_{j=1}^{i} \sum_{k=1}^{i} (A_{i,k} \times B_{k,j})$, irrespective of its "level". This motivates the need for a homogeneous treatment of subexpressions at any level. Essentially, we wish to treat expressions as first class objects and associate domains to arbitrary subexpressions. At the lowest level of specification, a subexpression is a variable (or a constant) associated with a domain.

The treatment of expressions as first class objects leads to the design of a functional language where programs are a finite list of (mutually recursive) equations of the form `Var` = `Expr` where

Expression	**Syntax**	**Domain**
Constants	Constant name or symbol	$\mathcal{D}_C$
Variables	`V`	$\mathcal{D}_{\texttt{V}}$
Operators	$\texttt{op}(\texttt{Expr}_1, \ldots, \texttt{Expr}_M)$	$\bigcap_{i=1}^{M} \mathcal{D}_{\texttt{Expr}_i}$
Case	$\texttt{case}\,\texttt{Expr}_1; \ldots; \texttt{Expr}_M\,\texttt{esac}$	$\biguplus_{i=1}^{M} \mathcal{D}_{\texttt{Expr}_i}$
Restriction	$\mathcal{D}' : \texttt{Expr}$	$\mathcal{D}' \cap \mathcal{D}_{\texttt{Expr}}$
Dependence	$\texttt{Expr}.f$	$f^{-1}(\mathcal{D}_{\texttt{Expr}})$
Reductions	$\texttt{reduce}(\oplus, f, \texttt{Expr})$	$f(\mathcal{D}_{\texttt{Expr}})$

Table 2.1: Expressions: Syntax and Domains.
The domain of an expression A is denoted by $\mathcal{D}_A$. The operator, `op`, may be written in infix notation if it binary. $\biguplus$ denotes disjoint union and f^{-1} denotes relational inverse.

both `Var` and `Expr` denote mappings from their respective domains to a set of values quite similar to multidimensional arrays. A variable is defined by at most one equation. Expressions are constructed by the rules given in Table 2.1 (column 2). The domains of all variables are declared, the domains of constants are $\mathbb{Z}^0$ by default, and the domains of expressions are derived by the rules given in column 3 of Table 2.1. The function specified in a dependence expression is called the *dependence function* (or simply a *dependence*) and the function specified in a reduction is called the *projection function* (or simply a *projection*).

In this language, Ex. (2.3) would be concretely written as

```
Y=5.(i->) + reduce(+,(i,j,k->i),A.(i,j,k->i,k)*B.(i,j,k->k,j))
```

In the equation above,

1. `5` is a constant expression. The dependence on `5` is defined for all values of `i`.

2. The domains of the variables `Y`, `A` and `B` would be declared, respectively, as the sets $\{i|1 \leq i \leq n\}$, $\{i,k|1 \leq i \leq n, 1 \leq k \leq i\}$ and $\{k,j|1 \leq k \leq n, 1 \leq j \leq n\}$.

3. The reduce expression is the accumulation in Ex. (2.3). Summation is expressed by the reduction operator + (Other possible reduction operators are $*$, `max`, `min`, `or`, `and`, `xor` etc.).

   (a) The projection function `(i,j,k->i)` specifies that the accumulation is over the space spanned by j and k resulting in values in the one-dimensional space spanned by i.

(b) A subtle and important detail is that the accumulated expression is defined over a domain in three-dimensional space, spanned by i, j and k (This information is implicit in standard mathematical specifications as in Eq. (2.3)).

4. The accumulated expression (say E) is an operator expression equal to the product of the value of A at $[i,k]$ and B at $[k,j]$. We have

$$E[i,j,k] = A[i,k] \times B[k,j]$$

In functional notation, the required dependences of the operator expression on A and B are expressed through dependence expressions `A.(i,j,k->i,k)` and `B.(i,j,k->k,j)` respectively.

5. The equation does not contain any `case` or restrict constructs. For an example of these two constructs, refer to the following equation

```
A  =  case
      {2j|1 ≤ 2j ≤ n}:A.(2j->j)
      {2j − 1|1 ≤ 2j − 1 ≤ n}:0
      esac
```

where the domain of A is $\{i|1 \leq i \leq n\}$. There are two branches of the `case` expression, each of which is a restriction expression. Note that this is not a program in the polyhedral model since both the restriction domains have "holes". This is a valid specification in the $\mathcal{Z}$-polyhedral model.

2.2.1 Semantics

At this point, we intuitively understand the semantics of expressions. Here, we provide the formal semantics of expressions over their domains of definition. At the iteration point z in its domain, the value of

- a constant expression is the associated constant.
- a variable is either provided as input or given by an equation; in the latter case, it is the value, at z, of the expression on its *rhs*.
- an operator expression is the result of applying `op` on the values of its arguments at z. `op` is an arbitrary, point-wise, single valued function.

- a case expression is the value at z of that branch whose domain contains z. Branches of a case expression are defined over disjoint domains to ensure that the case expression is not under- or over-defined.
- a restriction over `E` is the value of `E` at z.
- the dependence expression $\texttt{E}.f$ is the value of `E` at $f(z)$.
- $\texttt{reduce}(\oplus, f, \texttt{E})$ is the application of $\oplus$ on the values of `E` at all points in its domain $\mathcal{D}_{\texttt{E}}$ that map to z by f. Since $\oplus$ is an associative and commutative binary operator, we may choose any order of application of $\oplus$.

It is often convenient to have a variable defined either entirely as input, or only by an equation. The former is called an *input variable* and the latter is a *computed variable*. So far, all our variables have been of these two kinds only. Since the family of parameterized polyhedra is closed under difference, it is always possible to transform any specification to have only input and computed variables.

A constraint on specifications in this equational language is that every program instance in a parameterized specification is independent, thus, all functions should map consumer iterations to producer iterations within the same program instance.

Our presentation of the equational language is based on ALPHA [Mau89, Le 92] that allows the specification of equations in the polyhedral model. ALPHA relies on a library for manipulating polyhedra called Polylib [Wil93].

2.2.2 Evaluation

The default evaluation for our specifications is demand-driven or lazy. The main advantage of this non-strictness is that it "frees the programmer from many concerns about evaluation order" [HPF99]. However, it is well known that demand driven execution can be quite inefficient.

For the polyhedral model, there has been extensive research and there exist well established techniques on the automatic scheduling of computations [KMW67, Lam74, Fea92a, Fea92b, DRV00, GRQ02]. Scheduling is the task of assigning an execution time to each computation so that precedence constraints are satisfied. The resultant schedule can then be used to generate code [QRW00, Bas02] that follows a strict evaluation order (and thus, is more efficient). Furthermore, the generated code may be optimized for parallelism [Fea92b] and/or for memory [DCD97, LF97, QR00, LLL01, TVSA01b, DSV05].

2.2.3 Context Domain

Consider the following specification

$$X = \text{reduce}(+, (i, j \to i), E)$$

where $\mathcal{D}_E = \{i, j | 0 \leq j \leq i\}$ and $\mathcal{D}_X = \{i | 0 \leq i \leq 100\}$.

The domain of the reduce expression ($\mathcal{D}_R$, say) is $i \geq 0$ and so if we study it in isolation, we would believe that there are an infinite number of values that have to be computed. However, the domain of X is just $\{i | 0 \leq i \leq 100\}$. Hence, the equation for X only needs a subset of values of the reduction.

The set of iteration points at which the value of an expression is actually needed is called the *context domain* of an expression [Dup97, GR06]. One can always restrict any sub-expression in an equational specification to its context domain and retain equivalent semantics. The context domain of an expression is derived from its "parent" (in the AST) by the following rules.

The context domain $\mathcal{X}_{\mathtt{E}}$ of the expression `E` is

- $\mathcal{D}_{\mathtt{V}} \cap \mathcal{D}_{\mathtt{E}}$ in the equation `V = E`. The context domain of the expression in the r.h.s. of an equation is the intersection of its domain with the domain of declaration of the variable in the l.h.s.

- $\mathcal{X}_{\mathtt{E}'}$, the context domain of the parent $\mathtt{E}'$, if $\mathtt{E}'$ is $\text{op}(\ldots, \mathtt{E}, \ldots)$.

- $\mathcal{D}_{\mathtt{E}} \cap \mathcal{X}_{\mathtt{E}'}$ when $\mathtt{E}'$ is case $\ldots, \mathtt{E}, \ldots$ esac. The context domain of the case expression, $\mathcal{X}_{\mathtt{E}'}$, is the domain over which its branches are accessed. The context domain of any branch is the context domain of the case expression intersected with the domain of the particular branch.

- $\mathcal{X}_{\mathtt{E}'}$ the context domain of the parent $\mathtt{E}'$, when $\mathtt{E}'$ is the restriction $\mathcal{D}' : \mathtt{E}$.

- $f(\mathcal{X}_{\mathtt{E}'})$ if $\mathtt{E}'$ is $\mathtt{E}.f$. The dependence is such that `E'[z]` equals `E[f[z]]`. Therefore, if the domain over which $\mathtt{E}'$ is accessed is $\mathcal{X}_{\mathtt{E}'}$ the context domain of `E` is $f(\mathcal{X}_{\mathtt{E}'})$.

- $f_p^{-1}(\mathcal{X}_{\mathtt{E}'}) \cap \mathcal{D}_{\mathtt{E}}$ if $\mathtt{E}'$ is $\text{reduce}(\oplus, f_p, \mathtt{E})$. This is because points of the expression `E` that map to the result of the reduction at `z` belong to the set $f_p^{-1}(\mathtt{z})$. To obtain the precise set of points, we need to intersect $f_p^{-1}(\mathcal{X}_{\mathtt{E}'})$ with the domain of `E`.

The notion of context domains is important in the determination of computational complexity [GR06], since we may have expressions that are defined on a much larger domain than needed.

An isolated study of such expressions occurring in the *rhs* of an equation may provide us with an incorrect estimate of the complexity of the equation.

The context domain of the two subexpressions in the example above are.

- The context domain of the r.h.s expression is $\mathcal{D}_X \cap \mathcal{D}_R = \mathcal{D}_X$.
- The context domain of E, $\mathcal{X}_E$, is $\{i, j | 0 \leq i \leq 100\} \cap \mathcal{D}_E = \{i, j | 0 \leq j \leq i \leq 100\}$.

2.2.4 Normalization

Most analyses and transformations in the polyhedral model (eg. the simplification of reductions, scheduling etc.) need equations in a special form. Normalization is the transformation of a specification to an equivalent form containing only equations of the following types

1. $\mathtt{V} = \mathtt{E}$
2. $\mathtt{V} = \mathtt{reduce}(\oplus, f_p, \mathtt{E})$ where the expression $\mathtt{E}$ is of the form

$$\begin{array}{lll} \mathtt{case} & & \\ \ldots & \ldots & \\ \mathcal{D}_{V,i}: & \mathtt{op}(\ldots, \mathtt{U}.f, \ldots) & \\ \ldots & \ldots & \\ \mathtt{esac} & & \end{array} \tag{2.4}$$

   and $\mathtt{U}$ is a variable or a constant.

Such a normalization transformation[2] first introduces an equation for every `reduce` expression, replacing its occurrence with the corresponding variable. As a result, we get equations of the forms $\mathtt{V} = \mathtt{E}$ or $\mathtt{V} = \mathtt{reduce}(\oplus, f_p, \mathtt{E})$ where the expression $\mathtt{E}$ does not contain any `reduce` sub-expressions. Subsequently, these expressions are processed by a rewrite engine to obtain equivalent expressions the form specified in (2.4). The rules for the rewrite engine are given in table 2.2. Rules 1-4 "bubble" a single `case` expression to the outermost level, rules 5-7 then "bubble" a single restrict sub-expression to the second level, rule 8 gets the operator to the pen-ultimate level and rule 9 is a dependence composition obtain a single dependence at the inner-most level.

[2]More sophisticated normalization rules may be applied to express the interaction of `reduce` expressions with other subexpressions. However, these are unnecessary in the scope of this thesis.

#	Input	Output
1	$\texttt{case } E_1, \ldots, E_{k-1}, \texttt{ case } , E'_1, \ldots, E'_m \texttt{ esac}, E_{k+1}, \ldots, E_n \texttt{ esac}$	$\texttt{case } E_1, \ldots, E_{k-1}, E'_1, \ldots, E'_m, E_{k+1}, \ldots, E_n \texttt{ esac}$
2	$\mathcal{D}' : \texttt{ case } E_1, \ldots, E_n \texttt{ esac}$	$\texttt{case } \mathcal{D}' : E_1, \ldots, \mathcal{D}' : E_n \texttt{ esac}$
3	$\texttt{op}(E_1, \ldots, E_{k-1}, \texttt{ case } E'_1, \ldots, E'_m \texttt{ esac}, E_{k+1}, \ldots, E_n)$	$\texttt{case op}(E_1, \ldots, E_{k-1}, E'_1, E_{k+1}, \ldots, E_n), \ldots, \texttt{op}(E_1, \ldots, E_{k-1}, E'_m, E_{k+1}, \ldots, E_n) \texttt{ esac}$
4	$(\texttt{case } E_1, \ldots, E_n \texttt{ esac}).f$	$\texttt{case } E_1.f, \ldots, E_n.f \texttt{ esac}$
5	$\mathcal{D}_1 : \mathcal{D}_2 : E$	$(\mathcal{D}_1 \cap \mathcal{D}_2) : E$
6	$\texttt{op}(E_1, \ldots, E_{k-1}, \mathcal{D}' : E, E_{k+1}, \ldots, E_n)$	$\mathcal{D}' : \texttt{op}(E_1, \ldots, E_{k-1}, E, E_{k+1}, \ldots, E_n)$
7	$(\mathcal{D}' : E).f$	$f^{-1}(\mathcal{D}') : E$
8	$(\texttt{op}(E_1, \ldots, E_n)).f$	$\texttt{op}(E_1.f, \ldots, E_n.f)$
9	$(E.f_1).f_2$	$E.(f_1 \circ f_2)$

Table 2.2: Normalization Rules

Chapter 3

Simplifying Reductions

The example in Chapter 1 showed the simplification of a specification of cubic complexity to quadratic complexity. Here, we will elaborate on the simplification technique and formally present the transformation on reductions. Recall that a reduction is an associative and commutative operator ($+$, $\times$, max, min, and, or, xor etc.) applied to collections of values to produce a collection of answers (eg. $\sum$, $\prod$, etc.). Our collections of values and answers are defined by *polyhedral* sets, parameterized by size parameters.

The key enabler of the simplification technique is that mathematical specifications may exhibit *reuse* where intermediate values that are computed or used at different index points are identical. In this chapter, we develop various program transformations to exploit this reuse to simplify the computational complexity of evaluating reductions. In simplification, a major concern is that, in general, multiple transformations may be applied to a program that exhibits reuse. Furthermore, some transformations may, in turn, be parameterized by an infinite design space. How, then, should we choose our transformations to obtain an equivalent program with minimum complexity.

In this chapter, we present results to perform the simplification of reductions

1. *automatically*, in the sense that it can be implemented as a compiler optimization, and
2. *optimally*, in the sense that the choices that the compiler makes (from an infinite search space) cannot be further improved, even by an oracle guiding the search.

For this, we present an algorithm for the optimal application of simplification techniques to return an equivalent specification with minimum complexity.

In order, we will provide a precise characterization of the complexity of polyhedral equations; the mathematical foundations for simplification of reductions; and an algorithm for the optimal application of such program simplifications.

3.1 Intuition of Algorithm and Techniques

Consider an equation of the form below which is a generalization of accumulations in standard mathematical notation.

$$Y[f_p(z)] = \bigoplus E[z] \tag{3.1}$$

where $\bigoplus$ is an associative and commutative operator, E is an expression defined over a domain $\mathcal{D}_E$ and f_p is a many-to-one affine function.

$Y[f_p(z)]$ is the "accumulation", using the $\bigoplus$ operator, of the values of E at all points $z \in \mathcal{D}_E$ that have the same image $f_p(z)$. The *accumulation space* of the above equation is characterized by $\ker(f_p)$: any two points, z and z' that contribute to the same element of the result, Y, satisfy $z - z' \in \ker(f_p)$. For example, $Y_i = \sum_{j=1}^{i} F_{i,j}$ for $i = 1 \ldots n$ is the sum of columns of a triangular array. Note that "columns" in this index space are along the accumulation space spanned by the vector $(0, 1)^T$. Our formulation in Eq. (3.1) is more general than standard notation in the sense that vectors spanning the accumulation space may not always be the canonical basis vectors.

Now, let us examine the complexity of these accumulations. If E has a distinct value at all points in its domain, they must all be computed and no optimization is possible. However, consider the case where the expression E exhibits *reuse*: its value is the same at many points in $\mathcal{D}_E$. Reuse is characterized by $\ker(f_r)$, the kernel of a many-to-one affine function, f_r: the value of E at any two points in $\mathcal{D}_E$ is the same if their difference belongs to $\ker(f_r)$. We can denote this reuse by $E[z] = X[f_r(z)]$, where X is a variable with domain $\mathcal{D}_X$. Hence, the canonical equation that we analyze in the simplification of reductions is

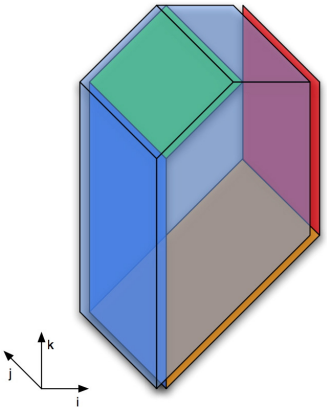
$$Y[f_p(z)] = \bigoplus X[f_r(z)] \tag{3.2}$$

Its nominal complexity is the cardinality of the domain $\mathcal{D}_E$ of $E[z] = X[f_r(z)]$.

The main idea behind our method is based on analyzing

1. $\mathcal{D}_E$, the domain of the expression E inside the reduction,
2. its reuse space, and
3. the accumulation space.

The reuse space is spanned by $\{(1,0,0)^T\}$ and the accumulation space is spanned by $\{(0,0,1)^T\}$. Translating $\mathcal{D}$ by $(1,0,0)^T$ exposes the left (blue) and top diagonal boundaries (green), and *exceeds* the right (red) and bottom diagonal (yellow) boundaries. Moreover, left and right boundaries contain the accumulation space. Each boundary is thus treated differently



- The left boundary is the *initialization* domain.
- The top diagonal boundary is the *addition* domain
- The bottom diagonal boundary is the *subtraction* domain.
- The right boundary is ignored (the projection along the accumulation space is outside the domain of the answer).
- The front, back and top horizontal boundaries are also ignored since translation is along these boundaries.

Figure 3.1: Illustration of the core algorithm for $Y_{i,j} = \sum_{k=i+1}^{\min(i+n,3n/2)} X_{j,k} \quad , 1 \leq i,j \leq n$

3.1.1 Core Algorithm

Let r_E be a vector in the reuse space of E and consider two adjacent instances of the answer variable, Y_z and Y_{z-r_Y} along $r_Y = f_p(r_E)$. The set of values that contribute to Y_z and Y_{z-r_Y} overlap. This would enable us to express Y_z in terms of Y_{z-r_Y}. Of course, there would be residual accumulations on values outside the intersection that must be "added" or "subtracted" accordingly. We may repeat this for other values of Y along r_Y. The simplification results from replacing the original accumulation by a recurrence on Y_{z-r_Y} and residual accumulations. For example, in the simple scan, $Y_i = \sum_{j=1}^{i} X_j$, the expression inside the summation $F_{i,j} = X_j$ has reuse along $(1,0)^T$. Taking $r_Y = (1)$, we get $Y_i = Y_{i-1} + F_{i,i} = Y_{i-1} + X_i$.

The geometric interpretation of the above reasoning is that we translate $\mathcal{D}_E$ by a vector in the reuse space of E. Let us call the translated domain $\mathcal{D}_{E'}$. The intersection of $\mathcal{D}_E$ and $\mathcal{D}_{E'}$ is precisely the domain of values over which the accumulation can be avoided. Their differences account for the residual accumulations. In the simple scan explained above the residual domain to be added is $\{i,j | 1 \leq i = j \leq n\}$ and the domain to be subtracted is empty. The residual accumulations to be added or subtracted are determined only by $\mathcal{D}_E$ and r_E.

This leads to Algorithm 1 (also see figure 3.1). Note that the algorithm assumes that the reduction operation admits an inverse, *i.e.*, "subtraction" is defined. If this is not the case, we

need to impose constraints on the direction of reuse to exploit: essentially requiring that the domain $\mathcal{D}^-$ is empty. This leads to a *feasible space* of exploitable reuse.

1. Choose r_E, a vector in $\ker(f_r)$, along which the reuse of E is to be exploited. In general, $\ker(f_r)$ is multidimensional and therefore there may be infinitely many choices.
2. Determine the domains $\mathcal{D}^0$, $\mathcal{D}^-$ and $\mathcal{D}^+$ corresponding to the domain of initialization, and the residual domains to subtract and to add, respectively. The choice of r_E is made such that the cardinality of these three domains are polynomials whose degree is strictly less than that for the original accumulation. This leads to simplification of the complexity.
3. For these three domains, $\mathcal{D}^0$, $\mathcal{D}^-$ and $\mathcal{D}^+$, define the three expressions, E^0, E^- and E^+ consisting of the original expression E, restricted to the appropriate subdomain.
4. Replace the original equation by the following recurrence

$$Y[f_p(z)] = \begin{cases} f_p(\mathcal{D}^0) & : \bigoplus E^0 \\ \mathcal{D}_Y - f_p(\mathcal{D}^0) & : Y[z - r_E] \oplus \\ & \quad (\bigoplus E^+) \ominus (\bigoplus E^-) \end{cases} \tag{3.3}$$

5. Apply steps 1-4 recursively on the residual reductions over E^0, E^- or E^+ if they exhibit further reuse.

Algorithm 1: Intuition of the Core Algorithm

3.1.2 Multidimensional Reuse

When the reuse space as well as the accumulation space are multidimensional, there are some interesting interactions. Consider the equation, $Y_i = \sum_{j=1}^{i-1} \sum_{k=1}^{i-j} X_j$ for $i \geq 2$. It has two-dimensional reuse (in the $\{i, k\}$ plane) and the accumulation is also two-dimensional (in the $\{j, k\}$ plane). Note that the two subspaces intersect, and this means that in the k direction, not only do all points have identical values, but they also all contribute to the same answer. From the bounds on the k summation we see that there are exactly $i - j$ such values, so the inner summation is just $(i - j) \times X_j$, because multiplication is a *higher order operator* for repeated addition of identical values (similar situations arise with other operators, e.g., power for multiplication, identity for the idempotent operator max, etc.). We have thus optimized the $\Theta(n^3)$ computation to $\Theta(n^2)$. However, our original equation had two dimensions of reuse and we may wonder whether further optimization is possible. In the new equation, $Y_i = \sum_{j=1}^{i-1} (i - j) \times X_j$, the reduction is over the product of two subexpressions, $(i-j)$ and X_j. They both have one dimension of reuse, respectively, in the $(1, 1)^T$ and $(1, 0)^T$ directions. However, their product does not have any reuse. Hence, no further optimization is possible for this equation.

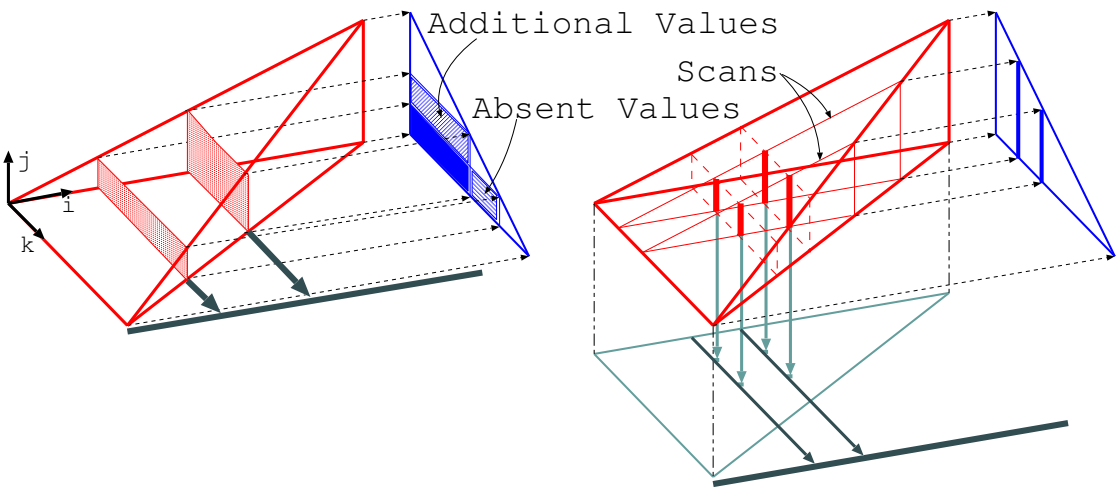


Figure 3.2: Geometric Interpretation for $Y_i = \bigoplus_{k=1}^{n-i} \bigoplus_{j=1}^{i} X_{j,k}$

On the other hand, if we had first exploited reuse along i, we would have obtained the simplified equation, $Y_i = Y_{i-1} + \sum_{j=1}^{i-1} X_j$ initialized with $Y_2 = X_1$. The residual reduction here is itself a scan, and we may recurse the algorithm to obtain $Y_i = Y_{i-1} + Z_i$ and $Z_i = Z_{i-1} + X_{i-1}$ initialized with $Z_2 = X_1$. Thus, our equation can be computed in linear time. This shows how the choice of reuse vectors to exploit, and their order affects the final simplification.

3.1.3 Decomposition of Accumulation

Consider the equation, $Y_i = \bigoplus_{k=1}^{n-i} \bigoplus_{j=1}^{i} X_{j,k}$ for $1 \leq i \leq n-1$. The one-dimensional reuse space is along $\{i\}$ and $\{j,k\}$ is the two-dimensional accumulation space. The set of points that contribute to the i-th result lie in an $i \times (n-i)$ rectangle of the two-dimensional input array X. Comparing successive rectangles, we see that as the width decreases from one to the other, the height increases (figure 3.2). If the operator $\oplus$ does not have an inverse, it seems that we may not be able to simplify this equation. This is not true: we can see that for each k we have an independent scan. The inner reduction $Z_{i,k} = \bigoplus_{j=1}^{i} X_{j,k}$, is actually just a family of scans, that can be done in quadratic time with $Z_{i,k} = Z_{i-1,k} + X_{i,k}$ initialized with $Z_{1,k} = X_{1,k}$. The outer reduction just accumulates columns of $Z_{i,k}$, which is also quadratic.

What we did in this example was to decompose the original reduction that was along the $\{j,k\}$ space into two reductions, the inner along the $\{j\}$ space yielding partial answers along the $\{i,k\}$ plane and the outer that combined these partial answers along the $\{k\}$ space. Note that

although the default choice of the decomposition—the innermost accumulation direction—of the $\{k, j\}$ space worked for this example, in general this is not the case. It is possible that the optimal solution may require a non-obvious decomposition, say along some diagonal. We encourage the reader to simplify[1] the following equation

$$Y_i = \bigoplus_{j=i}^{n+i} \bigoplus_{k=i}^{n+2i-j} X_{j,k}$$

3.1.4 Distributivity and Accumulation Decomposition

Let us now return to the motivating example on simplifying reductions presented in Chapter 1. Note that the methods presented so far are not applicable, since the body of the summation, $A_{i,k} * B_{j,k}$, has *no reuse*: at each point in the three dimensional $\{i, j, k\}$ space, the expression has a distinct value. However, the expression consists of two subexpressions, that individually have reuse and are combined with an operator, *viz.* multiplication, that distributes over the reduction operator (summation).

However, to distribute a subexpression outside any reduction, we need to ensure that it has a constant value at all the points that map to the same answer. We ensured this by reduction decomposition such that this property was satisfied for the inner reduction (along $\{j\}$). After distribution, the body of the inner summation exhibited reuse that was exploited for simplification of complexity.

3.2 Complexity

For the motivating example in Chapter 1, complexity was simply the number of dimensions but this is not strictly true. Consider the equation $A = B + C$ where the variables A, B and C are all defined over the domain $\{i, j | 1 \leq i \leq 10, 1 \leq j \leq n\}$. Within a program instance, these variables are indexed by both i and j and lie in 2-dimensional space. Note, however, the complexity of the equation is still $\Theta(n)$ as the number of operations performed are $10n$.

In this section, we will first formulate the complexity of equations as a function of the cardinalities of context domains of subexpressions on its *rhs*. Then, we will characterize the cardinality of a domain in terms of properties of integer polyhedra.

---

[1]Solution: The inner reduction would map all points for which $j + k = c$, for a given constant c, to the same partial answer.

3.2.1 Complexity of Equations

The equation $V = E$ defines a variable V over the domain $\mathcal{D}_V \cap \mathcal{D}_E$. Given the expression E, the complexity of the equation is the cardinality of $\mathcal{D}_V \cap \mathcal{D}_E$ — the context domain of E. However, there is also the complexity of evaluating E. The complexity of evaluating any expression given its immediate subexpressions is the maximum of the cardinality of its own context domain and the cardinalities of context domains of the subexpressions.

Above, we have a recursive definition for the asymptotic complexity of expressions. However, one may verify that a reduction is the only expression construct where the cardinality of the context domain of the subexpression (which is the expression that is reduced) is greater than the cardinality of the context domain of the parent expression. Thus, the asymptotic complexity of an equation is the maximum of the cardinality of the context domain of the expression on its *rhs* and the cardinalities of context domains of expressions inside all reduce subexpressions.

3.2.2 Additional Properties of Integer Polyhedra

The asymptotic complexity of equations is a function of the cardinalities of context domains. However, for any given program instance (specific parameter values) the cardinality is a constant for terminating programs. Nevertheless, the cardinalities of instances of the parametric family of context domains may increase unboundedly as a function of size parameters. We will now develop the mathematical machinery needed to define the complexity of program instances as a function of size parameters.

An integer polyhedron $\mathcal{P}$ is *unbounded* along a vector v *iff* for all constants k, $\mathcal{P} \cap \mathcal{P}' \neq \phi$ where $\mathcal{P}'$ is $\mathcal{P}$ translated by kv. For $\mathcal{P}$, let $\mathcal{L}_\mathcal{P}$ be the set of all such vectors.

Remark 1 *$\mathcal{L}_\mathcal{P}$ is a linear space.*

We will call $\mathcal{L}_\mathcal{P}$ the *effective linear subspace* of $\mathcal{P}$. $\mathcal{P}$ is said to be bounded or have constant thickness along any vector not in $\mathcal{L}_\mathcal{P}$.

We define the following properties on the constraints of $\mathcal{P}$.

- Given a constraint $c \equiv (a^T z + \alpha \geq 0)$, we say that $(a^T z + \alpha \leq \tau)$ for some non-negative integer constant τ, is an *effective inverse of c.*
- Given a constraint $c \equiv (a^T z + \alpha \geq 0)$, we say that $(a^T z + \beta \geq 0)$ for some integer constant β, is a *translation of c.*

- A constraint c of $\mathcal{P}$ is said to be *effectively saturated iff* $c' \cap \mathcal{P} = \mathcal{P}$ where c' is some effective inverse of c. Otherwise, it is said to be *effectively unsaturated*.
- The normals to all effectively unsaturated constraints of $\mathcal{P}$ are linearly independent of the set of normals to all effectively saturated constraints.

The following states how to obtain $\mathcal{L}_\mathcal{P}$ from the constraints of $\mathcal{P}$.

Remark 2 *The effective linear subspace of an integer polyhedron is the intersection of the kernels of its effectively saturated constraints.*

In concordance with the null intersection hypothesis, $\mathcal{L}_\mathcal{P}$ is the universe when $\mathcal{P}$ has no effectively saturated constraints.

3.2.2.1 Relevance to Complexity

We have seen that the complexity of equations is the cardinality of a certain domain. Here, we will relate the complexity of an equation in a parameterized specification to the properties defined above. We will assume that the domain whose cardinality we seek is an instance of a single parameterized integer polyhedron ($\mathcal{P}$, say).

It is assumed that program instances under consideration should not require the computation of values at an infinite number of index points. For this, we require that individual instances of the parameterized integer polyhedra $\mathcal{P}$ are bounded along all vectors[2] in $\mathcal{L}'$ (recall that $\mathcal{L}'$ is the space spanned by program indices). When $\mathcal{L}_\mathcal{P} \cap \mathcal{L}'$ is non-trivial, we know that even though the cardinality of any instance of $\mathcal{P}$ is finite, it increases unboundedly for large absolute values of size parameters. Since $\mathcal{P}$ is constrained by affine inequalities only, the cardinality of any instance of $\mathcal{P}$ is a polynomial in the size parameters [Ehr67, CLW97]. The degree of the polynomial is $\#(\mathcal{L}_\mathcal{P} \cap \mathcal{L}')$.

Since we are concerned with the simplification of the complexity of reductions only, we may allow a richer class of equations that define program instances over infinite domains. The weaker constraint the we need to impose arises from the semantics of a reduction over an expression E i.e., its value at any index point is the accumulation of a finite number of values of E. We can, therefore, not impose that instances of $\mathcal{P}$ are bounded along all vectors in $\mathcal{L}'$ and simply

[2]This is equivalent to the condition that there is not a non-trivial vector that lies in both $\mathcal{L}'$ and the characteristic cone of $\mathcal{P}$, given by $\{z | Qz \geq 0\}$

require that instances of the domain of E are bounded along all vectors in $\ker(f_p)$ where f_p is the projection function. For this generalized specification also, our measure of complexity will be the number of dimensions of $\mathcal{L}_{\mathcal{P}} \cap \mathcal{L}'$ since it captures the essence of the complexity involved in performing a reduction.

For the motivating example describing reduction simplification in Chapter 1, repeated here for convenience

$$Y_i = \sum_{j=1}^{i} \sum_{k=1}^{i} (A_{i,k} \times B_{k,j}) \quad , 1 \leq i \leq n$$

, the polyhedron over which the expression within the summation is defined lies in $\{i, j, k, n\}$. There are no effectively saturated constraints of this polyhedron, and thus $\mathcal{L}_{\mathcal{P}} = \{i, j, k, n\}$. There are three program indices, *viz.* i, j and k. Therefore, $\mathcal{L}' = \{i, j, k\}$. $\#(\mathcal{L}_{\mathcal{P}} \cap \mathcal{L}') = \#(\{i, j, k\})$ and thus the complexity of the reduction is $\Theta(n^3)$.

3.2.3 The Face Lattice

A *face* of $\mathcal{P}$ is an integer polyhedron that satisfies the set of constraints $\mathcal{C} \cup \mathcal{C}'$ where each constraint in $\mathcal{C}'$ is the inverse of some constraint in $\mathcal{C}$. We will identify each face with the set of saturated constraints in $\mathcal{C}$. Note, some unsaturated constraints in $\mathcal{C}$ may become redundant.

The *face lattice* of $\mathcal{P}$ is the set of all its faces ordered by set inclusion. $\mathcal{P}$ is the top element of this lattice. The bottom element is the face (possibly empty) for which all elements in $\mathcal{C}$ are saturated.

3.2.4 The Thick Face Lattice (TFL)

Let $\mathcal{P}$ be an integer polyhedron satisfying the set of constraints given by $\mathcal{C}$. A *thick face* of $\mathcal{P}$ is an integer polyhedron that satisfies the set of constraints $\mathcal{C} \cup \mathcal{C}'$ where each constraint in $\mathcal{C}'$ is an effective inverse of some constraint in $\mathcal{C}$. Since there are infinitely many effective inverses for any constraint in $\mathcal{C}$, there are an infinite number of thick faces. However, we may partition them into a finite number of equivalence classes where each equivalence class is identified by the set of effectively saturated constraints in $\mathcal{C}$ of a thick face. Note, all thick faces within an equivalence class have exactly the same effective linear subspace.

The *thick face lattice* of $\mathcal{P}$ is the set of all these equivalence classes ordered by set inclusion. The lattice point containing $\mathcal{P}$, identified by the smallest set of effectively saturated constraints, is the top element of the thick face lattice. The bottom element is the lattice point identified by the set that has all the constraints in $\mathcal{C}$.

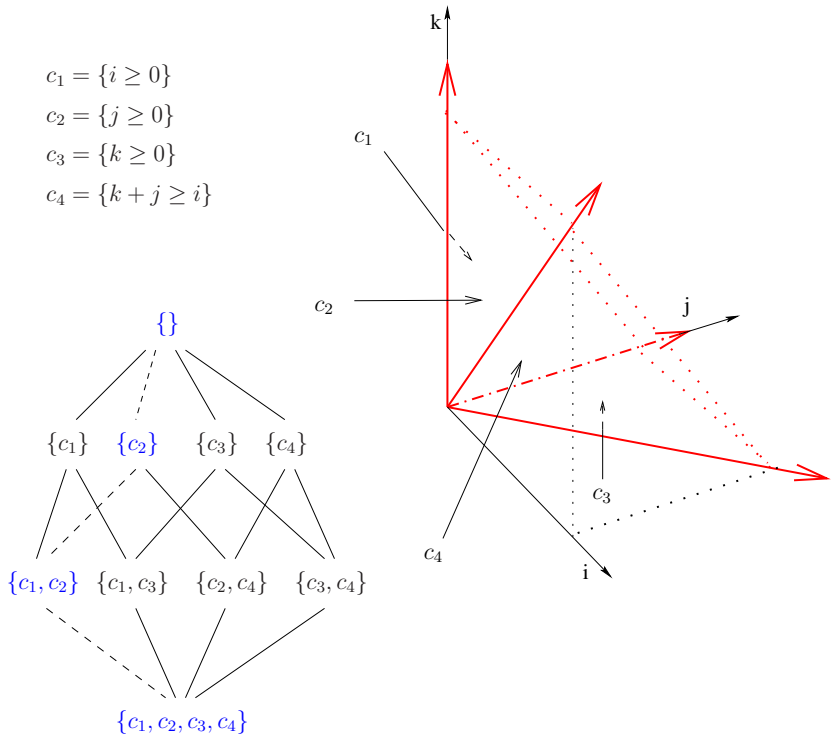


Figure 3.3: Thick Face Lattice

Figure 3.3 shows an example of a thick face lattice. Note that here there is no lattice point corresponding to the set $\{c_2, c_3\}$ because whenever both these constraints are effectively saturated, c_1 and c_4 too get effectively saturated. For the same reason there is not a lattice point corresponding to the set $\{c_1, c_4\}$. Similarly, there are no lattice points for which only three constraints are effectively saturated (the fourth one is simultaneously effectively saturated).

Thick faces from the lattice points in blue in figure 3.3 (connected by dotted lines) are shown in figure 3.4. Note, in figure 3.4d, the upper polyhedron is a thick face but the lower polyhedron is not. We will now define a set to which the lower polyhedron belongs.

3.2.5 Extended Thick Face Lattice (ETFL)

Consider an integer polyhedron $\mathcal{P}$ satisfying the set of constraints given by $\mathcal{C}$. A thick face of $\mathcal{P}$ satisfies the set of constraints $\mathcal{C} \cup \mathcal{C}'$ where each constraint in $\mathcal{C}'$ is an effective inverse of some constraint in $\mathcal{C}$. Translations of effective inverses simply correspond to another thick face. We will now define the set of *extended thick faces* of $\mathcal{P}$ where we, additionally, allow translations of constraints in $\mathcal{C}$.

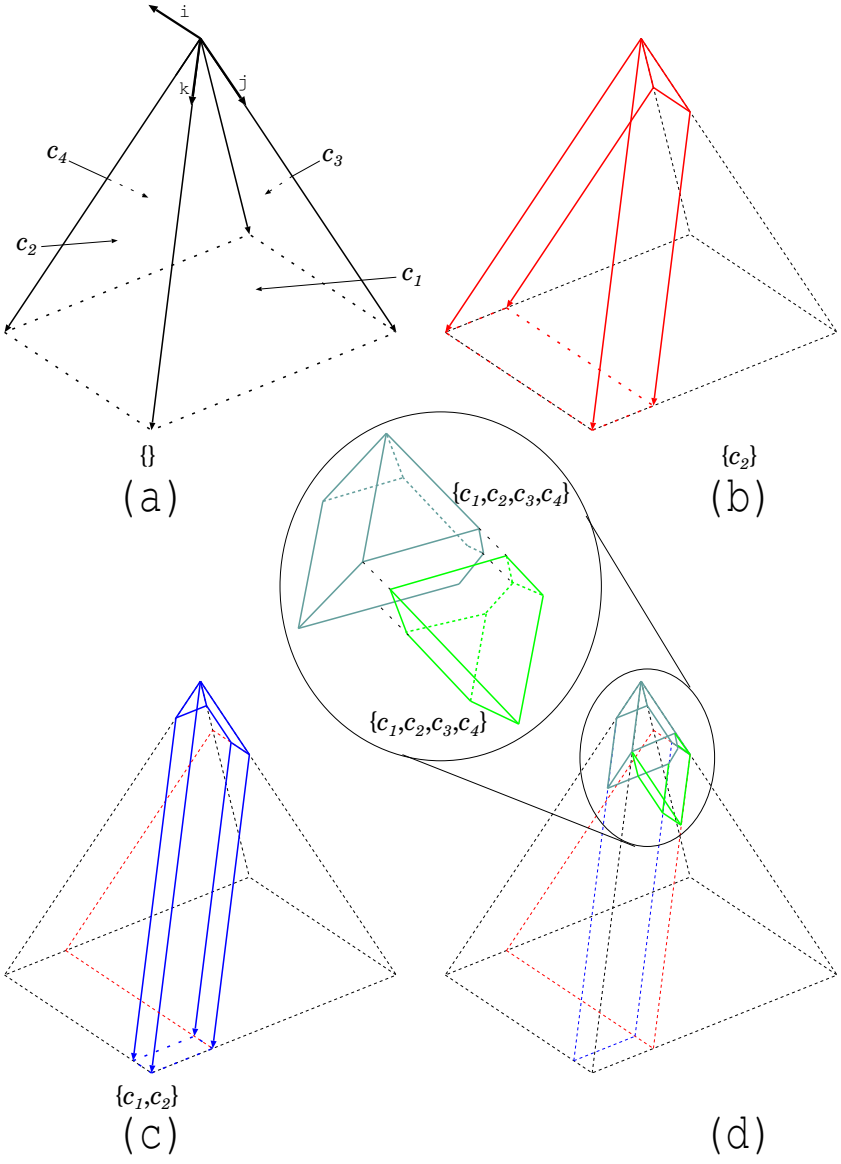


Figure 3.4: Thick Faces

An extended thick face is a non-empty integer polyhedron that satisfies the set of constraints $\mathcal{C}'' \cup \mathcal{C}'$. Constraints in $\mathcal{C}''$ are translations of constraints in $\mathcal{C}$ and each constraint in $\mathcal{C}$ has exactly one translation in $\mathcal{C}''$. We will partition the infinite set of extended thick faces into a finite number of equivalence classes where each equivalence class is identified by the set of effectively saturated constraints in $\mathcal{C}$.

Remark 3 *The effective linear subspace of all extended thick faces within an equivalence class is the same.*

The *effective thick face lattice* of $\mathcal{P}$ is the set of all these equivalence classes ordered by set inclusion. Again, the lattice point containing $\mathcal{P}$, identified by the smallest set of effectively saturated constraints, is the top element. The bottom element is the lattice point identified by the set that has all the constraints in $\mathcal{C}$.

The following remark on the extended thick faces of $\mathcal{P}$ will be used in our analysis.

Remark 4 *If $\mathcal{P}'$ is the translation of $\mathcal{P}$ along a non-trivial vector $r \in \mathcal{L}_{\mathcal{P}}$, then for the linear space $\mathcal{K}$, where $r \in \mathcal{K}$, the following properties hold*

- *$\mathcal{P} - \mathcal{P}' = \biguplus \mathcal{P}_i$ where each $\mathcal{P}_i$ is an extended thick face of $\mathcal{P}$ with effective linear subspace $\mathcal{L}_i$.*
- *For all $\mathcal{P}_i$, $\#(\mathcal{L}_{\mathcal{P}} \cap \mathcal{K}) > \#(\mathcal{L}_i \cap \mathcal{K})$.*
- *For at least one $\mathcal{P}_i$, $\#(\mathcal{L}_{\mathcal{P}} \cap \mathcal{K}) - \#(\mathcal{L}_i \cap \mathcal{K})$ is 1.*

These properties hold for the extended thick faces of $\mathcal{P}' - \mathcal{P}$ also, since $\mathcal{L}_{\mathcal{P}'} = \mathcal{L}_{\mathcal{P}}$ and the set of effective thick faces of $\mathcal{P}$ and $\mathcal{P}'$ are identical.

When $\mathcal{P}$ satisfies the set of constraints $\mathcal{C}$ and c_i is a constraint in $\mathcal{C}$ of the form $(a_i^T z + \alpha_i \geq 0)$, then

$$\mathcal{P} - \mathcal{P}' = \bigcup_{c_i \in \mathcal{C}} \mathcal{P} \cap (a_i^T z + \alpha_i < a_i^T r)$$

We can show that the residual extended thick faces, $\mathcal{P}_i$, have a one-one correspondence with elements from the set of non-empty thick faces $\mathcal{P} \cap (a_i^T z + \alpha_i < a_i^T r)$.

In our analysis, we will require that the set of non-redundant constraints of all extended thick faces within a lattice point of the ETFL is unique up to translations. As such this is not the case. Using the fact that we are performing a simplification of the asymptotic complexity of

reductions, we may restrict $\mathcal{P}$ to only "large" values of the size parameters and increase the number of redundant constraints for an extended thick face. We conjecture that with this restriction, a preprocessing of $\mathcal{P}$ may guarantee the requirement of our analysis.

3.3 Reduction Simplification

In this section, we present the simplification of a single reduction. We require equations of the form

$$X = \text{reduce}(\oplus, f_p, E)$$

where $\mathcal{D}_E$ is a single integer polyhedron and equal to $\mathcal{X}_E$. For simplicity of explanation, we have the reduction named by a computed variable X.

3.3.1 Preprocessing

As previously mentioned, we may restrict any expression to its context domain without any change in semantics.

We will use the following adaptation of a theorem by LeVerge [Le 92] to transform our reductions to be over expressions defined over single integer polyhedra.

Theorem 1 *A reduce expression of the following form*

$$X \quad = \quad reduce(\oplus, f_p, case\ E_1; E_2;\ esac)$$

is semantically equivalent to

$$\begin{aligned} X \quad = \quad & case \\ & \mathcal{D}_1 : X_1; \\ & \mathcal{D}_{12} : (X_1 \oplus X_2); \\ & \mathcal{D}_2 : X_2; \\ & esac; \end{aligned}$$

where $\mathcal{D}_{12} = f_p(\mathcal{D}_{E_1}) \cap f_p(\mathcal{D}_{E_2})$, $\mathcal{D}_1 = f_p(\mathcal{D}_{E_1}) - f_p(\mathcal{D}_{E_2})$, $\mathcal{D}_2 = f_p(\mathcal{D}_{E_2}) - f_p(\mathcal{D}_{E_1})$ *and* X_1 *and* X_2 *are computed variables defined over* $\mathcal{D}_1$ *and* $\mathcal{D}_2$ *respectively as follows*

$$\begin{aligned} X_1 &= reduce(\oplus, f_p, E_1) \\ X_2 &= reduce(\oplus, f_p, E_2) \end{aligned}$$

If $\mathcal{D}_E$ is a union of integer polyhedra, we can always rewrite E as case $E_1; E_2$ esac where $\mathcal{D}_{E_1}$ is one integer polyhedra in the union. By recursively applying the above theorem, we obtain reductions defined over single integer polyhedra. Note, if $\mathcal{D}_E = \mathcal{X}_E$, then all the values of X_1 and X_2 are needed to evaluate X and $\mathcal{D}_{E_1} = \mathcal{X}_{E_1}$ and $\mathcal{D}_{E_2} = \mathcal{X}_{E_2}$.

3.3.2 Sharing of Values

Consider a dependence expression E of the form

$$E'.f \tag{3.4}$$

The expression E has the same value at any two index points $z, z' \in \mathcal{D}_E$ if $f(z) = f(z')$ since they map to the same index point of E'. We will say that the value of E at these index points is *shared*. Note, two index points in $\mathcal{D}_E$ share a value if they differ by a vector in $\ker(f)$. Thus, values of E are shared along the linear space $\ker(f)$.

However, $\ker(f)$ may not be the maximal linear space along which values are shared in E. Observe, if in turn, index points in E' also share values, along $\ker(f')$ say, then a larger linear space along which values of E are shared is $\ker(f' \circ f)$. We denote the maximal linear space along which values are shared in E as $\mathcal{S}_E$, and call it the *share space* of E. Below, we list the relationship between the share space of expressions. We assume that the specification has been transformed to have just input and computed variables, and all reductions are over expressions defined on a single integer polyhedron. The share space, S_E is[3] equal to

- ϕ if E is a constant.
- ϕ is E is an input variable. We assume that program inputs have no sharing.
- $\mathcal{S}_{E'}$ if E is a computed variable defined by $E = E'$
- $\bigcap_{i=1}^{M} \mathcal{S}_{E_i}$ if E is $\mathrm{op}(E_1, \ldots, E_M)$
- $\bigcap_{i=1}^{M} \mathcal{S}_{E_i}$ if E is case $E_1, \ldots, E_M$ esac
- $\mathcal{S}_{E'}$ if E is $\mathcal{D}' : E'$

[3]This is not a constructive definition but enables us to compute it. We will not present here, the techniques to determine the share space of expressions and to optimally transform an equational specification for simplification.

- $\ker(T \circ f)$ if E is $E'.f$ and $\mathcal{S}_{E'} = \ker(T)$.
- $f_p(\ker(Q) \cap \mathcal{S}_{E'})$ if E is reduce$(\oplus, f_p, E')$ and $\mathcal{D}_{E'}$ is a single integer polyhedron $\mathcal{P} \equiv \{z | Qz + q \geq 0\}$.

Since a program has to be self sufficient, all dependences are within the same program instance. Therefore, the share space for a parameterized expression is a subset of the space spanned by the program indices $\mathcal{L}'$.

Any (parameterized) expression E that has a sharing $\mathcal{S}_E$ can be transformed to an expression of the form $E'.f$ where $\ker(f) = \mathcal{S}_E$ and f has full row rank. As a result, the expression E' has the trivial share space ($\mathcal{S}_{E'} = \phi$).

3.3.3 Simplification when an inverse operator exists

We will now analyze equations of the form

$$X = \text{reduce}(\oplus, f_p, E) \tag{3.5}$$

where $\mathcal{D}_E$ is a single integer polyhedron ($\mathcal{P}$, say) equal to $\mathcal{X}_E$ and $\mathcal{S}_E$ is the sharing in E. The following remark states how we can exploit the sharing of values in E along a vector $r_E \in \mathcal{S}_E \backslash \ker(f_p)$ to transform our reduction. We assume that $\oplus$ admits an inverse denoted by $\ominus$.

Remark 5 *An equation of the following form*

$$X = \textit{reduce}(\oplus, f_p, E)$$

is semantically equivalent to

$$
\begin{array}{rcl}
X & = & \\
\textit{case} & & \\
(\mathcal{D}_{add} - \mathcal{D}_{int}) & : & X_{add}; \\
(\mathcal{D}_{int} - (\mathcal{D}_{add} \cup \mathcal{D}_{sub})) & : & X.(z \to z - r_X); \\
(\mathcal{D}_{add} \cap (\mathcal{D}_{int} - \mathcal{D}_{sub})) & : & (X_{add} \oplus X.(z \to z - r_X)); \\
(\mathcal{D}_{sub} \cap (\mathcal{D}_{int} - \mathcal{D}_{add})) & : & (X.(z \to z - r_X) \ominus X_{sub}); \\
(\mathcal{D}_{add} \cap \mathcal{D}_{int} \cap \mathcal{D}_{sub}) & : & (X_{add} \oplus X.(z \to z - r_X) \ominus X_{sub}); \\
\textit{esac}; & & \\
X_{add} & = & \textit{reduce}(\oplus, f_p, (\mathcal{D}_E - \mathcal{D}_{E'}) : E) \\
X_{sub} & = & \textit{reduce}(\oplus, f_p, \\
 & & f_p^{-1}(\mathcal{D}_{int}) : (\mathcal{D}_{E'} - \mathcal{D}_E) : E')
\end{array}
$$

where $r_E \in \mathcal{S}_E \backslash ker(f_p)$ is a constant vector, $E' = E.(z \to z - r_E)$, $r_X = f_p(r_E)$, $\ominus$ is the inverse of $\oplus$, $\mathcal{D}_{add}$, $\mathcal{D}_{sub}$ and $\mathcal{D}_{int}$ denote the domains $f_p(\mathcal{D}_E - \mathcal{D}_{E'})$, $f_p(\mathcal{D}_{E'} - \mathcal{D}_E)$ and $f_p(\mathcal{D}_E \cap \mathcal{D}_{E'})$ respectively, and X_{add} and X_{sub} are defined over the domains $\mathcal{D}_{add}$ and $\mathcal{D}_{int} \cap \mathcal{D}_{sub}$ respectively.

We require that $r_E \notin \ker(f_p)$. This is because our technique involves the use of the value of X at an index point to simplify the computation at another and so in order to avoid a self-dependence, we must ensure that these index points are distinct (i.e., $r_X \neq 0$).

Also note, if we choose to apply the above transformation along a vector $r_E \in \ker(Q) \cap \mathcal{S}_E$ where $\ker(Q)$ is the lineality space of $\mathcal{P}$, then $\mathcal{D}_{\text{add}}$ and $\mathcal{D}_{\text{sub}}$ are both empty. So, $\mathcal{D}_{\text{add}} - \mathcal{D}_{\text{int}} = \phi$ and therefore the transformed specification for X does not have an initialization.

Remark 6 *The transformation in remark 5 cannot be applied to a reduction over the expression E along any vector in the intersection of the share space and the lineality space of E.*

However, the projection of $\ker(Q) \cap \mathcal{S}_E$ by f_p is precisely the share space of the entire reduce expression. We may simplify the complexity of the reduction by transforming it to have the trivial share space. Given that instances of $\mathcal{D}_E$ are bounded along all vectors in $\ker(f_p)$, this would result in a improvement of $\#(\ker(Q) \cap \mathcal{S}_E)$ in the degree of the polynomial representing the complexity of the reduction.

We will assume in the remainder of this chapter that such a simplification has been performed whenever applicable such that there is no sharing in the values of the reduction.

3.3.3.1 Complexity Analysis

We can see that the reduction in (3.5) has been replaced by two reductions, over expressions defined on $\mathcal{D}_E - \mathcal{D}_{E'}$ and $f_p^{-1}(\mathcal{D}_{\text{int}}) : (\mathcal{D}_{E'} - \mathcal{D}_E)$. Since there is no longer a reduction on the *rhs* of X, the complexity of its evaluation is now simply determined by the number of index points in its domain. However, the complexity of the transformed list of equations is the max of the cardinality of $\mathcal{D}_X$ and that of the domains of expressions inside the residual reductions for X_{add} and X_{sub}, $\mathcal{D}_E - \mathcal{D}_{E'}$ and $f_p^{-1}(\mathcal{D}_{\text{int}}) : (\mathcal{D}_{E'} - \mathcal{D}_E)$, respectively. Note, if $\mathcal{D}_E = \mathcal{X}_E$, then all the values of the residual reduction are needed to evaluate X and the domains of the expressions inside the residual reductions is equal to their context domains.

For simplicity of explanation, only for the case when an inverse exists, we will drop the domain restriction[4] $f_p^{-1}(\mathcal{D}_{\text{int}})$ on $\mathcal{D}_{E'} - \mathcal{D}_E$. Taking $\mathcal{K} = \mathcal{L}'$ in remark 4, we know that if r_E lies in $\mathcal{L}_{\mathcal{P}}$, then the max of the complexity of residual reductions is a polynomial whose degree is exactly one less than that of the original reduction. We will call the intersection of $\mathcal{S}_E$ and $\mathcal{L}_{\mathcal{P}}$ the *available reuse space* of the reduction and denote it by $\mathcal{R}_E$. $\mathcal{R}_E$ is the precise space along which we may exploit sharing to get improvements in asymptotic complexity since sharing of values along a vector in $\mathcal{S}_E \backslash \mathcal{R}_E$ is just along a constant number of index points. We will refer to the transformation in remark 5 along a vector in $\mathcal{R}_E \backslash \ker(f_p)$ as the *simplification transformation*.

3.3.3.2 Recursive Simplification

In general, both $\mathcal{D}_E - \mathcal{D}_{E'}$ and $\mathcal{D}_{E'} - \mathcal{D}_E$ are unions of integer polyhedra. By remark 4, they may be written as a disjoint union of extended thick faces of $\mathcal{D}_E$. Hence, using theorem 1, we will first transform both the residual reductions into a set of reduce expressions, each defined over a single extended thick face. The simplification transformation may be repeated on resultant reductions over expressions defined on extended thick faces if their available reuse space is non-trivial.

---

[4]Note that the domain restriction does not affect $\mathcal{L}_{\mathcal{P}}$, $\mathcal{S}_E$ and f_p.

3.3.4 Simplification in the absence of an inverse

So far, we assumed that an inverse operator exists. However, this may not always be the case *eg.* max, min. To be able to use the simplification transformation in such situations, we must ensure that there are no values of E to "subtract". Thus, the domain $\mathcal{D}_{\text{int}} \cap \mathcal{D}_{\text{sub}}$ of X_{sub} must be empty. The conditions for this are presented in the following remark.

Remark 7 *Let $\mathcal{P}$ be an integer polyhedron defined by the set of constraints $\mathcal{C}$ where $c_i \in \mathcal{C}$ is of the form $(a_i^T z + \alpha_i \geq 0)$, $\mathcal{H}_{\mathcal{P}}$ be the linear part of the smallest affine subspace that contains $\mathcal{P}$, $\mathcal{P}'$ be the translation of $\mathcal{P}$ along a non-trivial vector $r \in \mathcal{L}_{\mathcal{P}}$, f be an affine function, and $\mathcal{P}_{sub}$ and $\mathcal{P}_{int}$ be the integer polyhedra $f(\mathcal{P}' - \mathcal{P})$ and $f(\mathcal{P} \cap \mathcal{P}')$ respectively. Then $(\mathcal{P}_{int} \cap \mathcal{P}_{sub} = \phi)$ if all constraints c_i for which $a_i^T r < 0$ satisfy the following condition*

$$\mathcal{H}_{\mathcal{P}} \cap ker(f) \subseteq \mathcal{H}_{\mathcal{P}} \cap ker(c_i)$$

All constraints of $\mathcal{P}$ for which[5] $\mathcal{H}_{\mathcal{P}} \cap \ker(f) \subseteq \mathcal{H}_{\mathcal{P}} \cap \ker(c_i)$ will be called *boundary constraints*. We will denote the set of boundary constraints by $\mathcal{B}$. Remark 7 implies that X_{sub} is empty if all constraints for which $a_i^T r < 0$ are boundary constraints.

The following corollary of remark 7 identifies the set of reuse vectors along which we may employ the simplification transformation in the absence of an inverse operator.

Corollary 1 *The simplification transformation can be employed along a reuse vector $r_E \in \mathcal{R}_E \backslash ker(f_p)$ in the absence of an inverse operator if*

$$a_i^T r_E \geq 0, \forall c_i \in \mathcal{C} - \mathcal{B}$$

where $c_i \equiv (a_i^T z + \alpha_i \geq 0)$.

Note, the condition is trivially satisfied for all effectively saturated constraints of $\mathcal{P}$ (since $a_i^T z = 0, \forall z \in \mathcal{L}_{\mathcal{P}}$). We will call the intersection $a_i^T z \geq 0$ for all $c_i \in \mathcal{C} - \mathcal{B}$ where $z \in \mathcal{R}_E$, the *validity cone* of reuse and denote it by $\mathcal{V}_E$. The notion of the validity cone can be extended to the case when the an inverse exists. It is simply the entire available reuse space, $\mathcal{R}_E$.

In corollary 1, we have presented the conditions for applying the simplification transformation along a vector r_E. However, to perform the simplification transformation, we may choose any vector in $\mathcal{V}_E$. Thus, we may make the following claim.

---

[5] $\mathcal{H}_{\mathcal{P}}$ is the intersection of the kernels of all the saturated constraints of $\mathcal{C}$

Claim 1 *We can perform the simplification transformation on a reduction in the absence of an inverse operator if there exists a non-trivial vector in*

$$\mathcal{V}_E \backslash ker(f_p) \tag{3.6}$$

Remark 8 *Remark 7 and subsequently corollary 1 and claim 1 are both necessary and sufficient when* $\mathcal{L}_\mathcal{P} \cap ker(f_p) = ker(f_p)$.

When $\mathcal{L}_\mathcal{P} \cap \ker(f_p) \subset \ker(f_p)$, it is possible to replace the reduction by a finite number of reductions, each on a polyhedron having a greater number of equalities. This may increase the number of boundary constraints. We will study this case in detail later.

We must ensure that $a_i^T r_E > 0$ for at least one constraint $c_i \equiv (a_i^T z + \alpha_i \geq 0)$, so that there is an initialization. This is always possible because the reduce expression has a trivial share space.

3.3.4.1 Recursive Simplification

In the absence of an inverse operator, too, we may recursively apply the simplification transformation on the resultant reductions over expressions defined on extended thick faces if their available reuse space is non-trivial.

3.3.5 Simplification along $\mathcal{R}_E \cap \ker(f_p)$

Because $r_X = f_p(r_E)$ is a dependence in the transformed equation, the simplification transformation can only be applied along a vector $r_E \notin \ker(f_p)$. Here, we will discuss the simplification of complexity that can be achieved through exploitation of shared values of E along a non-trivial $\mathcal{R}_E \cap \ker(f_p)$. This is the case when shared values of E contribute to the same index point of the reduce expression. We will consider two cases. The first is when the reduction operator is idempotent, *eg.* max or min, and the second is when there is a higher order operator, *eg.* $\times$ for $+$, *exponentiation* for $\times$. In our explanation, we will use the projection function f_c where $\ker(f_c) = \mathcal{R}_E \cap \ker(f_p)$.

3.3.5.1 Idempotence

When the reduction operator is idempotent, a single index point along $\mathcal{R}_E \cap \ker(f_p)$ is sufficient. The many-to-one function f_c collapses points along the space over which the reduction is unnecessary. We may replace our reduction over E by a reduce expression over its subset E' where $f_c(\mathcal{D}_E) = f_c(\mathcal{D}_{E'})$. The expression E' is typically chosen such that the degree of the polynomial

representing its cardinality is lower than that for the polynomial representing the cardinality of E, thus resulting in simplification.

3.3.5.2 Higher Order Operator

When the reduction operator $\oplus$ is not idempotent but has an associated higher order operator, say $\otimes$, we may again simplify the reduction. The shared value along $\mathcal{R}_E \cap \ker(f_p)$ may be "multiplied" (using $\otimes$) by the number of its occurrences. This is again the cardinality of an instance of a parameterized integer polyhedron. It is precisely given by the Ehrhart quasi-polynomial[6] [Ehr67, CLW97].

We assume that the simplification through use of the higher order operator is a terminal transformation (as seen in section 3.1.2). Thus, in order to obtain the maximum improvement in complexity, we will collapse along the entire space of $\mathcal{R}_E \cap \ker(f_p)$. If the complexity of an application of $\otimes$ is the same as the application of $\oplus$ then such a transformation would result in a decrease in the polynomial complexity by $\#(\mathcal{R}_E \cap \ker(f_p))$ (m, say) in its degree. On the other hand, if the implementation of such an operator is not provided, we could define one that applies $\oplus$ to k identical values in a logarithmic number of steps. The decrease in the complexity in this case is $\frac{n^m}{m \log n}$ where n denotes a size parameter.

3.4 Simplification Enhancing Transformations

In the previous section, we saw transformations that resulted in the simplification of reductions. Here, we will present transformations that, *per se*, do not simplify but enhance simplification.

3.4.1 Algebraic Transformations enhancing $\mathcal{R}_E$

When the reduction is over an expression E that is a binary expression over two subexpressions, we may be able to transform the reduction into simpler reductions over one or more of its subexpressions. Since, subexpressions can only have a larger or equal share space, such a transformation may expand the available reuse space and thus enhance simplification. These transformations have no side effects (on $\mathcal{D}_E$ or f_p) and so are aggressively applied prior to any simplification of the reduction.

---

[6] Calculation of the ehrhart quasi-polynomial is NP-hard in the number of constraints. However, its calculation can be strength reduced at compile time. Moreover, the number of constraints of a polyhedra is a constant.

3.4.1.1 Same Operator Simplification

A reduction of the form

$$\text{reduce}(\oplus, f_p, E_1 \oplus E_2)$$

is semantically equivalent to

$$\text{reduce}(\oplus, f_p, E_1) \oplus \text{reduce}(\oplus, f_p, E_2)$$

The complexity of evaluating the reduction over E is now the max of the complexity of evaluating the two resultant reductions.

3.4.1.2 Distributivity

Consider a reduction of the form

$$\text{reduce}(\oplus, f_p, E_1 \otimes E_2)$$

where $\otimes$ distributes over $\oplus$.

If one of the expressions is constant within the summation (E_1, say), we would be able to distribute it outside the reduction, as we have already seen in the motivating example in Chapter 1. For the expression E_1 to be constant within a reduction by the projection f_p, we require

$$\mathcal{H}_{\mathcal{P}} \cap \ker(f_p) \subseteq \mathcal{H}_{\mathcal{P}} \cap \mathcal{S}_{E_1}$$

After distribution, the resultant expression is

$$E_1 \otimes \text{reduce}(\oplus, f_p, E_2)$$

Distributing the entire expression E. If there exists the operator $\diamond$ with id, its identity element, such that $\diamond$ distributes over $\oplus$, then the expression E within the reduction may be replaced with $E \diamond \text{id}.f$ where f is the dependence from $\mathcal{L}'$ to $\mathbb{Z}^0$. We would be able to distribute E outside the reduction and the available reuse space of the resultant expression within the reduce expression ($\text{id}.f$) would become all of $\mathcal{L}_{\mathcal{P}} \cap \mathcal{L}'$.

3.4.2 Expression Transformation

We will now see how we may enhance simplification when E is a case expression, a restriction or a dependence. Again, these transformations have no side effects (on $\mathcal{D}_E$ or f_p) and so are applied whenever applicable.

3.4.2.1 Case expression

Using theorem 1, we may transform the reduction over a case expression into a set of reductions over its alternatives. Since, alternatives can only have a larger share space, such a transformation may expand the available reuse space, and thus enhance simplification. The complexity of evaluating the original reduction is the max of the complexity of evaluating all resultant reductions.

3.4.2.2 Restriction and Dependences

When the reduction is over an expression E of the form $\mathcal{D}' : E'$ (or $E'.f$), and E' is neither a variable nor a constant, then it is always possible to distribute the restriction (or dependence) inside the expression E'. This can be done automatically through a set of normalizing rules.

These transformations do not alter the share space of the expression within the reduction. However, they expose other top-level expressions which may possibly enhance simplification.

3.4.3 Reduction Decomposition

We will now introduce a transformation that has wide applicability in enhancing simplification.

An expression of the form

$$\text{reduce}(\oplus, f_p, E)$$

is semantically equivalent to

$$\text{reduce}(\oplus, f''_p, \text{reduce}(\oplus, f'_p, E))$$

where[7] $f_p = f''_p \circ f'_p$.

We will call this the *reduction decomposition*. The complexity of evaluating the transformed expression is the max of the complexity of the inner and outer reductions. In this chapter, we assume that the inner reduction has no sharing of answers and therefore has a strictly greater complexity.

Depending on its use, the reduction decomposition may or may not have side effects.

3.4.3.1 Reduction Decomposition without Side Effects

Here, we will see the use of reduction decomposition to weaken the condition for boundary constraints and distributivity. In both these cases, it enhances simplification (having more boundary

[7] A discussion of the legality of f''_p and f'_p is beyond the scope of this thesis

constraints also potentially expands the validity cone).

For a reduction over E defined on $\mathcal{D}_E = \mathcal{P}$, the expression E extends unboundedly only along $\mathcal{L}_\mathcal{P}$. For simplification, it suffices to consider the reduction along f'_p defined as $\ker(f'_p) = \mathcal{L}_\mathcal{P} \cap \ker(f_p)$ since f'_p collapses E along the space on which it is unbounded and where it contributes to a single index point of the reduce expression. The outer reduction along f''_p simply accumulates a constant number of values for every index point of the reduction. Other contributing factors to simplification and complexity, *viz.* $\mathcal{L}_\mathcal{P}$, $\mathcal{R}_E \cap \ker(f_p)$ and the cardinality of the reduce expression remain unchanged.

For the inner reduction, the condition for boundary constraints becomes

$$\mathcal{L}_\mathcal{P} \cap \ker(f'_p) \subseteq \mathcal{L}_\mathcal{P} \cap \ker(c)$$

Since now, $\ker(f'_p) = \mathcal{L}_\mathcal{P} \cap \ker(f_p)$, through this reduction decomposition, the condition for applying the simplification transformation in the absence of an inverse becomes both necessary and sufficient, as stated in remark 8.

With this reduction decomposition, the condition for distributivity becomes

$$\mathcal{L}_\mathcal{P} \cap \ker(f'_p) \subseteq \mathcal{L}_\mathcal{P} \cap \mathcal{R}_E$$

Since this reduction decomposition has no side effects, we will apply it whenever $\ker(f_p) \neq \mathcal{L}_\mathcal{P} \cap \ker(f_p)$.

3.4.3.2 Reduction Decomposition with Side Effects

Above, we saw the use of reduction decomposition to weaken the condition for boundary constraints and distributivity. Here again, we have the same motivation but will look at a more aggressive kind of reduction decomposition that has a side effect in the form of increase in the cardinality of the domain of the reduce expression and decrease of $\mathcal{R}_E \cap \ker(f_p)$.

We assume that the aforementioned reduction decomposition has already been performed.

Boundary Constraints are those for which $\mathcal{L}_\mathcal{P} \cap \ker(f_p) \subseteq \mathcal{L}_\mathcal{P} \cap \ker(c)$. We may transform a non-boundary constraint, c, into a boundary constraint of the domain of the expression inside the inner reduction through a decomposition by choosing f'_p such that

$$\ker(f'_p) = \ker(f_p) \cap \ker(c)$$

We had performed such a reduction decomposition in the example in section 3.1.3.

Distributivity of an expression E outside a reduction by a projection f_p is valid when $\mathcal{L}_\mathcal{P} \cap \ker(f_p) \subseteq \mathcal{L}_\mathcal{P} \cap \mathcal{R}_E$. We may decompose f_p into $f''_p \circ f'_p$ to distribute an expression with available reuse space R_E outside the inner reduction by choosing f'_p such that

$$\ker(f'_p) = \ker(f_p) \cap \mathcal{R}_E$$

We had performed such a reduction decomposition to enable distributivity in our motivating example in Chapter 1.

3.5 Optimality and Algorithm

We have seen transformations that simplify reductions and those that enable simplification. One of our main contributions in this chapter is an algorithm to choose the optimal transformations and their parameters from an infinite search space. We emphasize that our claim of optimality is only with respect to the transformations described here. There may be transformations that enable a greater decrease in complexity. Furthermore, there could be alternate specifications for a problem that may not even be expressed in our model *viz.* in our equational specification, matrix multiplication is $\Theta(n^3)$.

The problem of simplifying reductions exhibits optimal substructure such that the optimal solution to the problem contains within it, optimal solutions to subproblems. Secondly, all subproblems are independent. We conjecture that a preprocessing will guarantee that the set of non-redundant constraints of all extended thick faces within a lattice point in the ETFL is unique up to translations. We can then show that they will have the same complexity and thus the optimal simplification problem needs to be solved only once for reductions by the same projection over the same expressions restricted to different extended thick faces within a lattice point in the ETFL. This ensures overlapping subproblems. We will therefore formulate a dynamic programming algorithm, with complexity as the cost function, that chooses the optimal transformation at every step in the simplification.

For the dynamic programming in step 5 to work, we must show that from the infinite search space of parameters for the various transformations, we need to consider only a finite set of choices and the global optima can be reached through such choices. This is done below.

Simplification Transformation: Application of the simplification transformation on a reduction over E defined over the integer polyhedron $\mathcal{P}$, along a valid reuse vector r_E, results in

Input: An equational specification in the polyhedral model.

1. Preprocess to obtain a reduction over an expression whose domain is a single polyhedron and equal to its context domain.
2. Transform the reduction to have a trivial share space.
3. Perform the reduction decomposition without side effects.
4. Perform any of the following transformations, if applicable
   (a) Same operator simplification.
   (b) Distributivity.
   (c) Expression Transformation.
5. Repeat steps 1-4 till convergence.
6. Dynamic Programming Algorithm to optimally choose
   (a) The simplification transformation along some r_E.
   (b) A simplification using an idempotent or higher order operator whichever is applicable.
   (c) A reduction decomposition with side effects.
7. Repeat from step 1 on residual reductions until convergence.

Output: An equivalent specification of optimal complexity.

Algorithm 2: The Optimal Simplification Algorithm

residual reductions defined over $\mathcal{P} - \mathcal{P}'$ and $f_p^{-1}(\mathcal{P}_{\text{int}}) : \mathcal{P}' - \mathcal{P}$ where $\mathcal{P}'$ is the translation of $\mathcal{P}$ along r_E and $\mathcal{P}_{\text{int}} = f_p(\mathcal{P} \cap \mathcal{P}')$. For simplicity, we will again ignore the restriction $f_p^{-1}(\mathcal{P}_{\text{int}})$. It does not affect our claim of optimality. These are transformed to a set of residual reductions over extended thick faces, denoted by $\mathcal{P}_i$. From our conjecture, reductions by the same projection over the same expression restricted to different extended thick faces within a lattice point of the ETFL have the same complexity.

Given that the ETFL is finite, we only have a finite number of combinations of residual reductions to consider. Therefore, we may partition the infinite set of choices for the reuse vector r_E into a finite number of equivalence classes corresponding to the set of residual reductions. We then simply need to consider a representative element from each equivalence class.

As stated previously, there is a one-to-one correspondence between each $\mathcal{P}_i$ and the thick faces in $\mathcal{P} - \mathcal{P}' = \bigcup_{c_i \in \mathcal{C}} \mathcal{P} \cap (a_i^T z + \alpha_i < a_i^T r_E)$ and $\mathcal{P}' - \mathcal{P} = \bigcup_{c_i \in \mathcal{C} - \mathcal{B}} \mathcal{P}' \cap (a_i^T z + \alpha_i < 0)$ where the constraint c_i of $\mathcal{P}$ is of the form $a_i^T z + \alpha_i \geq 0$. Note that for c_i, $\mathcal{P} \cap (a_i^T z + \alpha_i < a_i^T r_E)$ is empty when $a_i^T r_E \leq 0$ and $\mathcal{P}' \cap (a_i^T z + \alpha_i < 0)$ is empty when $a_i^T r_E \geq 0$. The set of extended thick faces of $\mathcal{P}$ and $\mathcal{P}'$ are identical. We can therefore say that a residual reduction over an extended thick face of $\mathcal{P}$ corresponding to the effective saturation of $c_i \in \mathcal{C} - \mathcal{B}$ vanishes when $a_i^T r_E = 0$. If $c_i \in \mathcal{B}$, a residual reduction over the corresponding effective thick face vanishes when $a_i^T r_E \leq 0$. To have an initialization, we must ensure that $a_i^T r_E > 0$ for at least one constraint c_i. We now simply need to choose representative elements from all combinations of the intersection of a strict inequality $a_i^T r_E > 0$ corresponding to some constraint, a subset of hyperplanes $a_i^T r_E = 0, c_i \in \mathcal{C} - \mathcal{B}$ and half-spaces $a_i^T r_E \leq 0, c_i \in \mathcal{B}$, and $\mathcal{V}_E \backslash \ker(f_p)$.

After the simplification transformation, the complexity of evaluating a reduction is the max of the complexity of evaluating residual reductions (and the cardinality of its domain). Therefore, if we have an equivalence class of residual reductions that is a strict superset of another, then it may be ignored in the search for an optimal reuse direction.

Idempotence: Simplification exploiting reuse along $\ker(f_c) = \mathcal{R}_E \cap \ker(f_p)$ for an idempotent reduction operator transformed the reduction over E to one over E' where $f_c(\mathcal{D}_E) = f_c(\mathcal{D}_{E'})$. We restrict our simplification to unions of extended thick faces of $\mathcal{D}_E$ and do not consider arbitrary domains. Since the ETFL is finite, we only need to consider a finite number of unions.

Higher Order Operator: In our analysis,, simplification exploiting reuse along $\mathcal{R}_E \cap \ker(f_p)$ using a higher order operator is a terminal step. In order to get maximum improvement in

complexity, we collapse the entire space of $\mathcal{R}_E \cap \ker(f_p)$. Thus, there is only a single choice in the use of this transformation. Note that the improvement in complexity is the same for a reduction by the same projection over the same expression restricted to different extended thick faces within a lattice point of the ETFL.

Reduction Decomposition with Side Effects: Of the infinite possible decompositions of the projection f_p, we only need to consider a finite subset, since the transformation is needed to increase the set of boundary constraints and/or distribute a set of subexpressions outside the inner reduction. Both these sets are elements from a finite power set which is unique for reductions by the same projection over the same expression restricted to extended thick faces within a lattice point of the ETFL.

3.6 Conclusions

Here we have presented an automatic and optimal program transformation algorithm to achieve simplification of the computational complexity of programs that have reductions and whose values exhibit reuse. Using our techniques, we have been able to repeat the optimization, from $\Theta(n^4)$ to $\Theta(n^3)$, presented in [LZP99] of an algorithm for RNA secondary structure prediction.

Chapter 4

The $\mathcal{Z}$-Polyhedral Model

To address the limitations of the polyhedral model, we present the $\mathcal{Z}$-polyhedral model with unions of $\mathcal{Z}$-polyhedra as the family of domains and affine lattice functions (generalizations of affine functions) as the associated family of functions, and show that it satisfies all the required closure properties.

This chapter is divided into three parts.

1. In the first part, we present some previous results on $\mathcal{Z}$-polyhedra by Le Verge [Le 94] that provide (polynomial time) sufficient conditions for testing whether an LBL is a $\mathcal{Z}$-polyhedron.

2. Then we present the representation that is key in proving the required closure properties on unions of $\mathcal{Z}$-polyhedra. We will prove the completeness of our representation showing that it permits the specification of any $\mathcal{Z}$-polyhedron. To enhance the expressivity available to programmers, we give a relaxation of Le Verge's sufficient conditions to check if an LBL is also a $\mathcal{Z}$-polyhedron. Finally, we provide the interpretation of $\mathcal{Z}$-polyhedra in our representation, equivalence of two $\mathcal{Z}$-polyhedra, canonical form and parameterized $\mathcal{Z}$-polyhedra.

3. The third part provides proofs of all the required closure properties to generalize the polyhedral model. This include closure of the family of unions of $\mathcal{Z}$-polyhedra under union, intersection and difference. In this part, we also characterize the family of affine lattice functions as a strict superset of the family of affine functions. We then prove the closure of $\mathcal{Z}$-polyhedral domains under preimage and image by arbitrary affine lattice functions, and the closure of the family of affine lattice functions under composition. We also present clo-

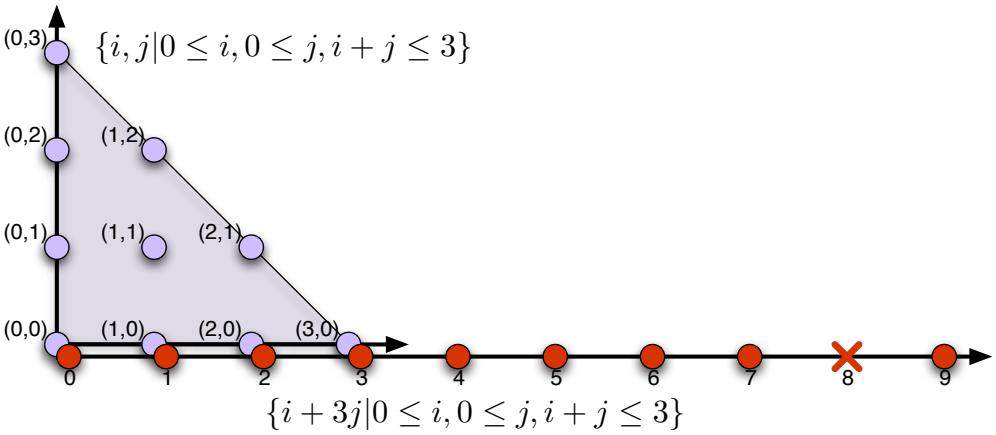


Figure 4.1: Image of the polyhedron $\{i,j|0 \leq i, 0 \leq j, i+j \leq 3\}$ by the affine function $(i,j \rightarrow i+3j)$

sure under change of basis, an important semantics-preserving transformation of equational specifications.

Our proof of closure in the $\mathcal{Z}$-polyhedral model under image by affine lattice functions overcomes the limitations on reductions and use-set characterizations of the polyhedral model. Also, the closure under image proves that Presburger sets and unions of LBLs are equivalent to unions of $\mathcal{Z}$-polyhedra.

4.1 Le Verge's work on $\mathcal{Z}$-Polyhedra

We will extensively use Le Verge's results on $\mathcal{Z}$-polyhedra [Le 94]. He showed that the family of polyhedra is strictly contained in the family of $\mathcal{Z}$-polyhedra which in turn is strictly contained in the family of LBLs. For an example of an LBL that is not a $\mathcal{Z}$-polyhedron, refer to Figure 4.1.

Le Verge also proved that membership testing in LBLs is $\mathcal{NP}$-complete and, even determining if an LBL is a $\mathcal{Z}$-polyhedron is at least $\mathcal{NP}$. He gave the following sufficient condition that may be verified in polynomial time for testing whether an LBL is a $\mathcal{Z}$-polyhedron: L has full column rank in the context, say $\ker(Q_0)$, of the integer polyhedron $\{z \in \mathbb{Z}^m | Q'z + q' \geq 0\}$ in (2.2). Mathematically, L has full column rank in context of the polyhedron *iff* $A = \begin{pmatrix} L \\ Q_0 \end{pmatrix}$ has full column rank.

We will also use the following proposition proved by him

Proposition 1 *Let M be an integral matrix. The following properties are equivalent:*

1. *there exists an integral matrix M' such that $M'M = I$;*

2. *there exists an integral matrix* N *such that* $\begin{pmatrix} M & N \end{pmatrix}$ *is unimodular;*

3. *The Hermite normal form of* M^T *is* $\begin{pmatrix} I & 0 \end{pmatrix}$

4.2 Representation of $\mathcal{Z}$-Polyhedra

Recall that a $\mathcal{Z}$-polyhedron can be represented as an affine image of an integer polyhedron (also called an LBL). However, not all affine images of integer polyhedra are $\mathcal{Z}$-polyhedra. Our representation of $\mathcal{Z}$-polyhedra is a special kind of an affine image of an integer polyhedron

$$\{Lz + l | Qz + q \geq 0, z \in \mathbb{Z}^m\} \tag{4.1}$$

where L has full column rank and $\mathcal{P}^c_{\mathcal{Z}} = \{z | Qz + q \geq 0, z \in \mathbb{Z}^m\}$ has a context that is the universe, $\mathbb{Z}^m$.

L and l are called the generator and offset, respectively, for the representation (say, $\mathcal{Z}$)[1] of the $\mathcal{Z}$-polyhedron. $\mathcal{P}^c_{\mathcal{Z}}$ is called the coordinate polyhedron associated with the particular representation. When L has no columns, ${\mathcal{P}_{\mathcal{Z}}}^c$ lies in $\mathbb{Z}^0$. We say two representations are equivalent if they correspond to the same set. Conversely, two sets are equal if they have equivalent representations.

The interpretation is that the $\mathcal{Z}$-polyhedral representation is said to be *based on* the affine lattice given by $\{Lz + l | z \in \mathbb{Z}^m\}$. Iteration points of the $\mathcal{Z}$-polyhedral domain are points of the affine lattice corresponding to valid coordinates. The set of valid coordinates is given by the coordinate polyhedron.

Our conditions on L and $\mathcal{P}^c_{\mathcal{Z}}$ in the representation are even stricter than Le Verge's sufficient conditions for an LBL to be a $\mathcal{Z}$-polyhedron. In addition to the guarantee that (4.1) is in fact a $\mathcal{Z}$-polyhedron, we ensure the following three critical properties.

1. *There is a one-to-one mapping from the coordinate polyhedron to the* $\mathcal{Z}$*-polyhedron.* Since L has full column rank, it admits a left inverse which can be used to construct the inverse map from iteration points in the $\mathcal{Z}$-polyhedron to points in the coordinate polyhedron. *By this property, a* $\mathcal{Z}$*-polyhedron is empty* iff *its coordinate polyhedron is empty.*

2. This inverse map is, in fact, a one-to-one map from all coordinates in $\mathbb{Z}^m$ to all points on the lattice $\{Lz + l | z \in \mathbb{Z}^m\}$. Therefore, a point exists in the $\mathcal{Z}$-polyhedron *iff* its corresponding

[1] We denote the set of iterations in a $\mathcal{Z}$-polyhedron by $\mathcal{ZP}$, and particular representations by $\mathcal{Z}$.

coordinate exists in ${\mathcal{P}_{\mathcal{Z}}}^c$. *Hence, two representations with the same affine image,* $Lz + l$, *are* equivalent iff *their coordinate polyhedra are equal.*

3. *Two representations,* $\{Lz+l|Qz+q \geq 0, z \in \mathbb{Z}^m\}$ *and* $\{L'z'+l'|Q'z'+q' \geq 0, z' \in \mathbb{Z}^{m'}\}$, *are* not equivalent *if the affine lattices represented by* $\{Lz+l|z \in \mathbb{Z}^m\}$ *and* $\{L'z'+l'|z' \in \mathbb{Z}^{m'}\}$ *are not equal.* This is because ${\mathcal{P}_{\mathcal{Z}}}^c$ has a context that is the universe, $\mathbb{Z}^m$, and L has full column rank.

Properties 2 and 3 are not guaranteed by Le Verge's sufficient conditions. With his conditions, it is possible for two $\mathcal{Z}$-polyhedra with the same affine image, $Lz + l$, to be equivalent even when their coordinate polyhedra differ, also, two representations may be equivalent even when the corresponding affine lattices in the representation are not equal. This is illustrated in the following example

Example 4 *Consider the following three representations of what we will see is the same* $\mathcal{Z}$*-polyhedra that satisfy Le Verge's sufficient conditions.*

$$\{2i, 0|1 \leq i \leq 10, i = j\}$$
$$\{2i, 0|1 \leq i \leq 10, j = 5\}$$
$$\{2i + j, j|1 \leq i \leq 10, j = 0\}$$

The first two representations have the same affine image

$$\left(\begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}\right)$$

but different coordinate polyhedra.

Also, the first two representations correspond to the affine lattice spanned by the single vector $\begin{pmatrix} 2 \\ 0 \end{pmatrix}$. *The third representation corresponds to a different affine lattice, spanned by the vectors* $\begin{pmatrix} 2 \\ 0 \end{pmatrix}$ *and* $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

All the three, however, correspond to the same $\mathcal{Z}$*-polyhedron that would be expressed as* $\{2i|1 \leq i \leq 10\}$ *in our representation.*

However, our stricter requirement does not mean that we accept a restricted set of objects or limit the expressibility provided to the programmer. We will first show that every $\mathcal{Z}$-polyhedron has a representation that satisfies the condition that L has full column rank. Subsequently, we will present a transformation that converts representations in which L has full column rank to those that additionally satisfy the condition that their coordinate polyhedron has the entire

universe as its context. Collectively, these proofs show that all $\mathcal{Z}$-polyhedra can be expressed in our representation.

Then, we will extend the result of Le Verge by relaxing his sufficient condition for an LBL to be a $\mathcal{Z}$-polyhedron. He requires $A = \begin{pmatrix} L \\ Q_0 \end{pmatrix}$, where $\ker(Q_0)$ is the context of the coordinate polyhedron, to have full column rank (equivalently, $\ker(A) = \{0\}$). In our relaxation, we simply require $\ker(A) \subseteq \ker(Q)$. To show that LBLs satisfying our weaker conditions are also $\mathcal{Z}$-polyhedra, we will provide a transformation that converts representations satisfying $\ker(A) \subseteq \ker(Q)$ to equivalent representations satisfying the more stringent requirement that $\ker(A) = \{0\}$.

Since it is at least $\mathcal{NP}$ to determine if an LBL is a $\mathcal{Z}$-polyhedron and our conditions take polynomial time to verify, these are also *sufficient* conditions just like Le Verge's.

4.2.1 Completeness of the Representation

Here we will show that every $\mathcal{Z}$-polyhedron can be represented in the required form presented in (4.1). Consider the $\mathcal{Z}$-polyhedron, $\mathcal{ZP}$ say, which is the intersection of the integer polyhedron $\mathcal{P} = \{y \in \mathbb{Z}^n | Qy + q \geq 0\}$ and the affine lattice represented by $\mathcal{L} = \{Lz + l | z \in \mathbb{Z}^m\}$ where Q and L are $b \times n$ and $n \times m$ matrices respectively and q and l are a b-vector and n-vector respectively. Note that both $\mathcal{P}$ and the affine lattice represented by $\mathcal{L}$ are in $\mathbb{Z}^n$.

We will show the completeness of the representation in two steps, first by showing that any $\mathcal{Z}$-polyhedron can be represented in a form where the generator has full column rank. Then, we will transform this representation to the required form in (4.1) that additionally satisfies the condition that the coordinate polyhedron has the entire universe as its context.

Step 1: Let L have rank d and $H = \begin{pmatrix} H' & 0 \end{pmatrix}$ be its Hermite normal form such that H' is an $n \times d$ matrix of full column rank and $L = HU$ where U is a unimodular matrix. The representation $\mathcal{L}$ can be written as $\{HUz + l | z \in \mathbb{Z}^m\}$. Since U is unimodular, $\mathcal{L}$ is equivalent to $\{Hz' + l | z' \in \mathbb{Z}^m\}$ where $z' = Uz$.

Now consider the $\mathcal{Z}$-polyhedron$\mathcal{ZP}$ obtained from the intersection of the affine lattice represented by $\mathcal{L}$ and the polyhedron $\mathcal{P}$.

$$\mathcal{ZP} \quad = \quad \{Hz' + l | z' \in \mathbb{Z}^m\} \cap \{y \in \mathbb{Z}^n | Qy + q \geq 0\}$$

The integer polyhedron and affine lattice intersect when $y = Hz' + l$. Replacing y by $Hz' + l$, we get

$$\begin{aligned}\mathcal{ZP} &= \{Hz' + l | Q(Hz' + l) + q \geq 0, z' \in \mathbb{Z}^m\} \\ &= \{\begin{pmatrix} H' & 0 \end{pmatrix} z' + l | Q \begin{pmatrix} H' & 0 \end{pmatrix} z' + (q + Ql) \geq 0, z' \in \mathbb{Z}^m\} \\ &= \{H'z'' + l | Q'z'' + q' \geq 0, z'' \in \mathbb{Z}^d\}\end{aligned}$$

where $Q' = QH'$, $q' = q + Ql$ and $z'' = \begin{pmatrix} I & 0 \end{pmatrix} z'$. Now, we have a representation where the generator, H', has full column rank. The representation can be brought to the required form of (4.1) using the following transformation.

Step 2: To show the completeness of our representation scheme, we need to prove that every representation of the form

$$\{Lz + l | Qz + q \geq 0, z \in \mathbb{Z}^m\}$$

where L has full column rank can be transformed to the required form in (4.1).

Let $\mathcal{P}^c_{\mathcal{Z}} = \{z | Qz + q \geq 0, z \in \mathbb{Z}^m\}$ be the coordinate polyhedron associated with the representation. Let the smallest affine subspace that contains $\mathcal{P}^c_{\mathcal{Z}}$ be $\{z | Tz = t\}$ where T is an $n \times m$ matrix and t is an n-vector. The context of $\mathcal{P}^c_{\mathcal{Z}}$ is given by $\ker(T)$. Let T have rank d such that $\mathcal{P}^c_{\mathcal{Z}}$ is a $m-d$ dimensional polyhedron embedded in $\mathbb{Z}^m$. Since, L has full column rank, iterations in the $\mathcal{Z}$-polyhedron have a one-to-one correspondence with coordinate points in ${\mathcal{P}_{\mathcal{Z}}}^c$ and so the $\mathcal{Z}$-polyhedron is also $m - d$ dimensional. In the following steps, we will use the equalities in $Tz = t$ to obtain an equivalent representation with a coordinate polyhedron in lower dimensional space, and appropriately massaged generators with full column rank.

Let S be the Smith normal form of T such that $T = VSU$, where V and U are unimodular matrices. We have $VSUz = t$. Since, U is unimodular, we may change the coordinates to $z' = Uz$ to get the following equivalent representation of the $\mathcal{Z}$-polyhedron

$$\{L'z' + l | Q'z' + q \geq 0, z' \in \mathbb{Z}^m\} \tag{4.2}$$

where $L' = LU^{-1}$ has full column rank and $Q' = QU^{-1}$.

We have $Sz' = t'$ where $t' = V^{-1}t$. By the definition of SNF, S is of the form $\begin{pmatrix} S' & 0 \\ 0 & 0 \end{pmatrix}$ where S' is a diagonal $d \times d$ matrix. If any of the $(d+1)^{\text{th}}, \ldots, n^{\text{th}}$ component of t' is non-zero then the coordinate polyhedron is empty implying the corresponding $\mathcal{Z}$-polyhedron is empty. Proceeding otherwise, if t'' is the vector constructed from the first d elements of t' and $z' = \begin{pmatrix} z_1 \\ z'' \end{pmatrix}$, we

have $\begin{pmatrix} S' & 0 \end{pmatrix} \begin{pmatrix} z_1 \\ z'' \end{pmatrix} = t''$ or

$$\begin{pmatrix} I & 0 \end{pmatrix} \begin{pmatrix} z_1 \\ z'' \end{pmatrix} = (S')^{-1}t''$$

These give us d equalities for the first d elements of z'. Again, if any elements of the vector $(S')^{-1}t''$ are rational, then the coordinate polyhedron is empty and therefore the corresponding $\mathcal{Z}$-polyhedron is empty. Otherwise, substituting $\begin{pmatrix} (S')^{-1}t'' \\ z'' \end{pmatrix}$ for z' in (4.2), we get the following equivalent representation

$$\{L''z'' + l' | Q''z'' + q' \geq 0, z'' \in \mathbb{Z}^{m-d}\} \tag{4.3}$$

with $L' = \begin{pmatrix} L_1 & L'' \end{pmatrix}$, $l' = l + L_1(S')^{-1}t''$, $Q' = \begin{pmatrix} Q_1 & Q'' \end{pmatrix}$ and $q' = q + Q_1(S')^{-1}t''$.

Note that L' and therefore L'' have full column rank. Since the $\mathcal{Z}$-polyhedron is $m - d$ dimensional and the coordinate polyhedron in (4.3) lies in $\mathbb{Z}^{m-d}$, therefore, it necessarily must have a context that is the entire universe.

4.2.2 Expressivity

Here, we will present the expressivity available to the programmer of specifications in the $\mathcal{Z}$-polyhedral model.

Relaxation of the Sufficient Condition Le Verge proved that an LBL of the form

$$\{Lz + l | Qz + q \geq 0, z \in \mathbb{Z}^m\}$$

is a $\mathcal{Z}$-polyhedron when $\ker(A) = \{0\}$ where $A = \begin{pmatrix} L \\ Q_0 \end{pmatrix}$ and $\ker(Q_0)$ is the context of the polyhedron $\{z | Qz + q \geq 0, z \in \mathbb{Z}^m\}$.

We will present a relaxation, only requiring $\ker(A) \subseteq \ker(Q)$, by providing a transformation that converts such representations to equivalent representations satisfying $\ker(A) = \{0\}$. In addition to providing greater expressivity to the programmer, this theorem is crucial to prove closure of unions of $\mathcal{Z}$-polyhedra under image.

Example 5 *Consider the following representation.*

$$\{i + j | 1 \leq i + j \leq 10\}$$

The generator, say L, in the representation is $\begin{pmatrix} 1 & 1 \end{pmatrix}$ *and its kernel is the subspace spanned by the vector* $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$. *The context of the coordinate polyhedron is the entire universe, therefore,*

$ker(Q_0) = \mathbb{Z}^2$, giving $ker(A) = ker(L) \cap ker(Q_0) = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$. Thus, Le Verge's sufficient conditions are not satisfied. However, this too corresponds to a $\mathcal{Z}$-polyhedron and may be transformed to $\{i|1 \leq i \leq 10\}$. Note that $ker(Q)$, where $\{Qz + q \geq 0\}$ are the constraints of the coordinate polyhedron, is also equal to $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$, thus $ker(A) \subseteq ker(Q)$.

The following theorem claims that more general representations, only satisfying $\ker(A) \subseteq \ker(Q)$, are also $\mathcal{Z}$-polyhedra.

Theorem 2 *The set $\{Lz+l|Qz+q \geq 0, z \in \mathbb{Z}^m\}$ is a $\mathcal{Z}$-polyhedron when $ker(A) \subseteq ker(Q)$ where $A = \begin{pmatrix} L \\ Q_0 \end{pmatrix}$ and $ker(Q_0)$ specifies the context of the coordinate polyhedron.*

Proof Our proof involves providing the transformation to an equivalent representation satisfying Le Verge's sufficient conditions; having the form $\{L'z' + l|Q'z' + q \geq 0, z' \in \mathbb{Z}^{m'}\}$ where L' has full column rank in the context of the coordinate polyhedron.

Let A be of rank d and $H = \begin{pmatrix} H' & 0 \end{pmatrix}$ be its Hermite normal form such that H' is the submatrix of full column rank corresponding to the first d columns of H. From definition, there exists a unimodular matrix U such that $A = HU$. We have $AU^{-1} = \begin{pmatrix} H' & 0 \end{pmatrix}$ where U^{-1} is the unimodular inverse of U.

Let $U = \begin{pmatrix} V_1 \\ V_2 \end{pmatrix}$ and $U^{-1} = \begin{pmatrix} V_1' & V_2' \end{pmatrix}$ where V_2' is the column submatrix of U^{-1} corresponding to the zero-columns of $\begin{pmatrix} H' & 0 \end{pmatrix}$ and $V_1V_1' = I$, $V_1V_2' = 0$, $V_2V_1' = 0$ and $V_2V_2' = I$.

Let us construct a matrix, W such that

$$W \begin{pmatrix} V_1' & V_2' \end{pmatrix} = \begin{pmatrix} V_1' & 0 \end{pmatrix}$$

or

$$\begin{aligned} W &= \begin{pmatrix} V_1' & 0 \end{pmatrix} U \\ &= V_1'V_1 \end{aligned}$$

Since $AV_2' = 0$ and $\ker(A) \subseteq \ker(L)$, $LV_2' = 0$. Since $U^{-1} = \begin{pmatrix} V_1' & V_2' \end{pmatrix}$ is a unimodular basis and the coordinates of z along V_2' do not affect Lz, we have

$$Lz + l = LWz + l$$

Also since, $\ker(A) \subseteq \ker(Q)$ we get $Qz + q = QWz + q$.

Substituting in $\{Lz + l|Qz + q \geq 0, z \in \mathbb{Z}^m\}$ we get $\{LWz + l|QWz + q \geq 0, z \in \mathbb{Z}^m\}$ or

$$\{LV_1'V_1z + l|QV_1'V_1z + q \geq 0, z \in \mathbb{Z}^m\} \tag{4.4}$$

The Hermite normal form for V_1 is $\begin{pmatrix} I & 0 \end{pmatrix}$ by proposition 1 since $V_1 V_1' = V_1'^T V_1^T = I$. For unimodular matrix U', we have

$$\begin{aligned} \{V_1 z | z \in \mathbb{Z}^m\} &= \{\begin{pmatrix} I & 0 \end{pmatrix} U' z | z \in \mathbb{Z}^m\} \\ &= \{\begin{pmatrix} I & 0 \end{pmatrix} z' | z' \in \mathbb{Z}^m\} \\ &= \{z'' | z'' \in \mathbb{Z}^d\} \end{aligned}$$

Denoting $V_1 z$ by z'' in (4.4), we get the following equivalent representation

$$\{LV_1' z'' + l | QV_1' z'' + q \geq 0, z'' \in \mathbb{Z}^d\} \tag{4.5}$$

We will now show that (4.5) is such that its generator, LV_1', has full column rank in the context, say $\ker(Q_0')$, of its coordinate polyhedron $\{z'' | QV_1' z'' + q \geq 0, z'' \in \mathbb{Z}^d\}$. Note, $\ker(Q_0 V_1')$ is a superset of $\ker(Q_0')$.

$$\begin{aligned} \ker \begin{pmatrix} LV_1' \\ Q_0' \end{pmatrix} &\subseteq \ker \begin{pmatrix} LV_1' \\ Q_0 V_1' \end{pmatrix} \\ &= \ker \left(\begin{pmatrix} L \\ Q_0 \end{pmatrix} V_1' \right) \\ &= \ker(AV_1') \\ &= \ker(H') \\ &= \{0\} \end{aligned}$$

since H' has full column rank.

This equivalent representation is a $\mathcal{Z}$-polyhedron since it satisfies Le Verge's sufficient conditions. ∎

Henceforth we will assume that all representations of $\mathcal{Z}$-polyhedra conform to our scheme presented in (4.1) where the generator has full column rank and the coordinate polyhedron has full context.

4.2.3 Interpretation

The conventional way to interpret our representation of $\mathcal{Z}$-polyhedra as

$$\{Lz + l | Qz + q \geq 0, z \in \mathbb{Z}^m\}$$

is similar to the definition of an LBL, as an affine image of an integer polyhedron.

We would like to motivate an alternate view in which $\{Lz + l | z \in \mathbb{Z}^m\}$ in the $\mathcal{Z}$-polyhedral representation is interpreted as a representation of an affine lattice. The $\mathcal{Z}$-polyhedral representation is said to be *based on* the representation of the affine lattice. The set of valid coordinates is given by the coordinate polyhedron. Iteration points of the $\mathcal{Z}$-polyhedral domain are points of the affine lattice corresponding to valid coordinates for the particular representation.

4.2.4 Equivalence

Our representation for $\mathcal{Z}$-polyhedra is such that any two $\mathcal{Z}$-polyhedral representations based on the same representation of an affine lattice are equivalent *iff* their corresponding coordinate polyhedra are equal. We will now study the equivalence of $\mathcal{Z}$-polyhedral representations based on different representations of the same affine lattice. Recall that in our representation scheme, $\mathcal{Z}$-polyhedral representations on different affine lattices are necessarily different. Consider the representation of a $\mathcal{Z}$-polyhedron

$$\{Lz + l | Qz + q \geq 0, z \in \mathbb{Z}^m\} \tag{4.6}$$

Let $\{L'z' + l' | z' \in \mathbb{Z}^m\}$ be a different representation of the affine lattice represented by $\{Lz + l | z \in \mathbb{Z}^m\}$. As a consequence of our representation scheme, L' necessarily has the same number of columns as L. By definition $L' = LU$ and $l' = Lz_0 + l$ for some constant vector $z_0 \in \mathbb{Z}^m$. The relationship between the coordinates in the two lattices is simply

$$\begin{aligned} Lz + l &= L'z' + l' \\ &= LUz' + Lz_0 + l \\ &= L(Uz' + z_0) + l \end{aligned}$$

Since L has full column rank, we have $z = Uz' + z_0$. Substituting for z in (4.6), we get the following equivalent representation

$$\{L'z' + l' | Q(Uz' + z_0) + q \geq 0, z' \in \mathbb{Z}^m\}$$

With this characterization, we have precisely decomposed the problem of equivalence of $\mathcal{Z}$-polyhedral representations to the problem of equivalence of representations of affine lattices and the equality of polyhedra. This is a direct consequence of our representation scheme. In previous works [QRR96, QRR97], the equivalence of two representations was unsolved.

4.2.5 Canonical Form

The representation of a $\mathcal{Z}$-polyhedron is not unique. However, a $\mathcal{Z}$-polyhedral representation is said to be in canonical form when it is based on an affine lattice representation in canonical form. We may choose any previously used canonical form for the representation of the coordinate polyhedron (having context that is the entire universe, therefore, satisfying a unique set of non-redundant constraints). Recall that, a property of our representation scheme is that two $\mathcal{Z}$-polyhedral representations based on the same affine lattice are equivalent *iff* their coordinate polyhedra are equal. Thus, the test for equality of two $\mathcal{Z}$-polyhedral representations in canonical form is trivial.

4.2.6 Parameterized $\mathcal{Z}$-Polyhedra

A parameterized $\mathcal{Z}$-polyhedron is a $\mathcal{Z}$-polyhedron where some rows of its corresponding affine lattice are interpreted as size parameters. Similar to parameterized integer polyhedra, elements of a parameterized $\mathcal{Z}$-polyhedron are called its instances and identified by the vector of size parameters.

For the sake of explanation, and without loss of generality, we may impose that the rows that denote size parameters are before all non-parameter rows. Such a $\mathcal{Z}$-polyhedron in its canonic representation has the important property that all points of the coordinate polyhedron with identical values of the first few indices belong to the same instance of the parameterized $\mathcal{Z}$-polyhedron.

Example 6 *Consider the $\mathcal{Z}$-polyhedron given by the intersection of the polyhedron $\{p, i | 0 \leq i \leq p\}$ and the lattice*[2] *$\{j + k, j - k\}$. It may be written as*

$$\{j + k, j - k | 0 \leq j - k \leq j + k\} = \{j + k, j - k | 0 \leq k \leq j\}$$

Now, suppose the first index, p, in the polyhedron is the size parameter. Similarly, the first row in the lattice $\{j + k, j - k\}$ corresponding to the $\mathcal{Z}$-polyhedron is the size parameter. The Hermite normal form of this lattice is $\{j', j' + 2k'\}$. The equivalent $\mathcal{Z}$-polyhedron is

$$\{j', j' + 2k' | k' \leq 0 \leq j' + 2k'\}$$

---

[2]For both the polyhedron and the affine lattice, the specification of the space $\mathbb{Z}^2$ is redundant. It can be derived from the number of indices and is therefore dropped for the sake of brevity.

The iterations of this $\mathcal{Z}$-polyhedron belong to the same program instance iff *they have the same coordinate index j'. Note that valid values of the parameter row trivially have a one-to-one correspondence with value of j'; identity being the required bijection. In the general case, however, this is not the case. Nevertheless, the required property remains invariant. For example, consider the following $\mathcal{Z}$-polyhedron with the first two rows considered as size parameters.*

$$\{m, m+2n, i+m, j+n | 0 \leq i \leq m; 0 \leq j \leq n\}$$

Here, valid values of the parameter rows have a one-to-one correspondence with the values of m and n but it is impossible to obtain identity as the required bijection.

4.3 Unions of $\mathcal{Z}$-Polyhedra and Closure Properties

Domains in the $\mathcal{Z}$-polyhedral model are unions of $\mathcal{Z}$-polyhedra where each element in the union is expressed in the representation discussed in section 4.2. To be an instance of the equational language, unions of $\mathcal{Z}$-polyhedra must be closed under intersection, union and difference and image and preimage by the family of functions. Here we will show closure under intersection and difference (The union of a finite set of unions of $\mathcal{Z}$-polyhedra is trivially a union of $\mathcal{Z}$-polyhedra).

In section 4.3.3 we will define the family of functions and then demonstrate closure of the family of unions of $\mathcal{Z}$-polyhedra under image and preimage by the family of functions.

4.3.1 Intersection

From elementary set theory, the intersection of two unions of $\mathcal{Z}$-polyhedra , given as $\mathcal{D} = \bigcup_i \mathcal{ZP}_i$ and $\mathcal{D}' = \bigcup_j \mathcal{ZP}'_j$, equals the union of intersections of two $\mathcal{Z}$-polyhedra as follows

$$\mathcal{D} \cap \mathcal{D}' = (\bigcup_i \mathcal{ZP}_i) \cap (\bigcup_j \mathcal{ZP}'_j) = \bigcup_{i,j} (\mathcal{ZP}_i \cap \mathcal{ZP}'_j)$$

Thus, we only need to show that the intersection of two $\mathcal{Z}$-polyhedra is a union of $\mathcal{Z}$-polyhedra. As a matter of fact, it is precisely a single $\mathcal{Z}$-polyhedron . Let the two $\mathcal{Z}$-polyhedra be represented by, say $\mathcal{Z}$ and $\mathcal{Z}'$ as follows

$$\begin{aligned} \mathcal{Z} &= \{Lz + l | Qz + q \geq 0, z \in \mathbb{Z}^m\} \\ \mathcal{Z}' &= \{L'z' + l' | Q'z' + q' \geq 0, z' \in \mathbb{Z}^{m'}\} \end{aligned}$$

The intersection of $\mathcal{Z}$-polyhedra represented by $\mathcal{Z}$ and $\mathcal{Z}'$ relies on the intersection of the affine lattices represented by $\mathcal{L} = \{Lz + l | z \in \mathbb{Z}^m\}$ and $\mathcal{L}' = \{L'z' + l' | z' \in \mathbb{Z}^{m'}\}$ on which they

are based. The intersection of affine lattices represented by $\mathcal{L}$ and $\mathcal{L}'$ is an affine lattice, say represented by $\mathcal{L}'' = \{L''z'' + l''|z'' \in \mathbb{Z}^{m''}\}$ where L'' has full column rank. The affine lattice represented by $\mathcal{L}''$ may be empty, in which case the corresponding $\mathcal{Z}$-polyhedron is empty. If $\mathcal{L}''$ represents a non-empty affine lattice, we have the following relationships. Note, L'' may have fewer columns than either L or L'.

$$L'' = LS, l'' = Ls + l$$
$$L'' = L'S', l'' = L's' + l'$$

where S and S' are matrices and s and s' are vectors.

Taking the intersection of the $\mathcal{Z}$-polyhedra represented by $\mathcal{Z}$ and $\mathcal{Z}'$ with the affine lattice represented by $\mathcal{L}''$ we get

$$\begin{aligned} \mathcal{Z} \cap \mathcal{L}'' &= \{L''z'' + l''|Q(Sz'' + s) + q \geq 0, z'' \in \mathbb{Z}^{m''}\} \\ \mathcal{Z}' \cap \mathcal{L}'' &= \{L''z'' + l''|Q'(S'z'' + s') + q' \geq 0, z'' \in \mathbb{Z}^{m''}\} \end{aligned}$$

Since, they are based on the same representation of the affine lattice, the intersection of these two $\mathcal{Z}$-polyhedra simply requires the intersection of their coordinate polyhedra.

$$\{L''z'' + l''| \begin{pmatrix} QS \\ Q'S' \end{pmatrix} z'' + \begin{pmatrix} q + Qs \\ q' + Q's' \end{pmatrix} \geq 0, z'' \in \mathbb{Z}^{m''}\}$$

Note, the coordinate polyhedra of this intersection may not have the entire universe as its context, in which case, we would bring it to the required representation through the transformation in step 2 presented in section 4.2.1.

4.3.2 Difference

From set theory, the difference of two unions of $\mathcal{Z}$-polyhedra, given as $\mathcal{D} = \bigcup_i \mathcal{Z}\mathcal{P}_i$ and $\mathcal{D}' = \bigcup_j \mathcal{Z}\mathcal{P}'_j$, equals the union of differences of two $\mathcal{Z}$-polyhedra as follows

$$\mathcal{D} - \mathcal{D}' = (\bigcup_i \mathcal{Z}\mathcal{P}_i) - (\bigcup_j \mathcal{Z}\mathcal{P}'_j) = \bigcup_i \left(\bigcap_j (\mathcal{Z}\mathcal{P}_i - \mathcal{Z}\mathcal{P}'_j) \right)$$

If we show that the difference of two $\mathcal{Z}$-polyhedra is a union of $\mathcal{Z}$-polyhedra, we may use the result on closure of domains under intersection presented in the previous section and claim closure under difference.

Let the two $\mathcal{Z}$-polyhedra be represented by, say $\mathcal{Z}$ and $\mathcal{Z}'$ as follows

$$\begin{aligned} \mathcal{Z} &= \{Lz + l|Qz + q \geq 0, z \in \mathbb{Z}^m\} \\ \mathcal{Z}' &= \{L'z' + l'|Q'z' + q' \geq 0, z' \in \mathbb{Z}^{m'}\} \end{aligned}$$

Points in $\mathcal{Z} - \mathcal{Z}'$ lie on the affine lattice represented by represented by $\mathcal{L} = \{Lz + l | z \in \mathbb{Z}^m\}$. Additionally, they may or may not belong to the affine lattice represented by $\mathcal{L}' = \{L'z' + l' | z' \in \mathbb{Z}^{m'}\}$. We will decompose the problem of finding the difference of $\mathcal{Z}$-polyhedra represented by $\mathcal{Z}$ and $\mathcal{Z}'$ into finding the intersection and difference of the affine lattices represented by $\mathcal{L} = \{Lz + l | z \in \mathbb{Z}^m\}$ and $\mathcal{L}' = \{L'z' + l' | z' \in \mathbb{Z}^{m'}\}$ on which they are based, and operations on the coordinate polyhedra of the $\mathcal{Z}$-polyhedra based on these lattices. Therefore, the difference of $\mathcal{Z}$-polyhedra represented by $\mathcal{Z}$ and $\mathcal{Z}'$ is a union of $\mathcal{Z}$-polyhedra, elements of which may be defined on either the intersection or the difference of the affine lattices.

Let us first consider the intersection of the affine lattices. The intersection of affine lattices represented by $\mathcal{L}$ and $\mathcal{L}'$ is an affine lattice, say represented by $\mathcal{L}'' = \{L''z'' + l'' | z'' \in \mathbb{Z}^{m''}\}$ where L'' has full column rank. The affine lattice represented by $\mathcal{L}''$ may be empty, in which case all $\mathcal{Z}$-polyhedra corresponding to the intersection are empty. If $\mathcal{L}''$ represents a non-empty affine lattice, we have the following relationships.

$$L'' = LS, l'' = Ls + l$$
$$L'' = L'S', l'' = L's' + l'$$

where S and S' are matrices and s and s' are vectors.

Taking the intersection of the $\mathcal{Z}$-polyhedra represented by $\mathcal{Z}$ and $\mathcal{Z}'$ with the affine lattice represented by $\mathcal{L}''$ we get

$$\begin{aligned} \mathcal{Z} \cap \mathcal{L}'' &= \{L''z'' + l'' | Q(Sz'' + s) + q \geq 0, z'' \in \mathbb{Z}^{m''}\} \\ \mathcal{Z}' \cap \mathcal{L}'' &= \{L''z'' + l'' | Q'(S'z'' + s') + q' \geq 0, z'' \in \mathbb{Z}^{m''}\} \end{aligned}$$

Since, they are based on the same representation of the affine lattice, the difference of these two $\mathcal{Z}$-polyhedra simply requires the difference of their coordinate polyhedra. The resultant is a union of $\mathcal{Z}$-polyhedra, each element of which (indexed by k) can be represented by

$$\{L''z'' + l'' | Q''_k z'' + q''_k \geq 0, z'' \in \mathbb{Z}^{m''}\}$$

where $\{z'' | Q(Sz'' + s) + q \geq 0, z'' \in \mathbb{Z}^{m''}\} - \{z'' | Q'(S'z'' + s') + q' \geq 0, z'' \in \mathbb{Z}^{m''}\}$ is a union of polyhedra of the form $\{z'' | Q''_k z'' + q''_k \geq 0, z'' \in \mathbb{Z}^{m''}\}$.

Let the obtained union of $\mathcal{Z}$-polyhedra be denoted by $\mathcal{D}_{\text{int}}$. The coordinate polyhedra of elements in this union may not have the entire universe as its context, in which case, we would bring them to the required representation through the transformation in step 2 presented in section 4.2.1.

Now, let us consider the difference of the affine lattices. The difference of affine lattices represented by $\mathcal{L}$ and $\mathcal{L}'$ is a union of non-empty affine lattices (indexed by h), say represented by $\mathcal{L}_h^\# = \{L_h^\# z_h^\# + l_h^\# | z_h^\# \in \mathbb{Z}^{m_h^\#}\}$ where each $L_h^\#$ has full column rank. We have the following relationships.

$$L_h^\# = LS_h, l_h^\# = Ls_h + l$$

where S_h is a matrix and s_h is a vector. The intersection of the $\mathcal{Z}$-polyhedra represented by $\mathcal{Z}$ with the affine lattices represented by $\mathcal{L}_h^\#$ can be represented by

$$\mathcal{Z} \cap \mathcal{L}_h^\# = \{L_h^\# z_h^\# + l_h^\# | QS_h z_h^\# + (q + Qs_h) \geq 0, z_h^\# \in \mathbb{Z}^{m_h^\#}\}$$

Let this union be denoted by $\mathcal{D}_{\text{diff}}$. The coordinate polyhedra of elements in this union may not have the entire universe as its context, in which case, we would bring them to the required representation through the transformation in step 2 presented in section 4.2.1.

Finally, we get $\mathcal{Z} - \mathcal{Z}' = \mathcal{D}_{\text{int}} \cup \mathcal{D}_{\text{diff}}$

4.3.3 Affine Functions on a Lattice

We will now define the family of functions over unions of $\mathcal{Z}$-polyhedra. An (standard) affine function is of the form $(z \to Tz + t)$ where T is an $n \times m$ matrix and t is an n vector. *Affine lattice functions or affine functions on a lattice* are of the form $(Kz + k \to Rz + r)$, where K has full column rank. Such functions provide a mapping from the iteration $Kz + k$ to the iteration $Rz + r$. We have imposed that K have full column rank to guarantee that $(Kz + k \to Rz + r)$ be a function and not a relation, mapping any point in its domain to a unique point in its range. All standard affine functions are also affine lattice functions.

4.3.4 Preimage

The preimage of a union of $\mathcal{Z}$-polyhedra is the union of the preimage of individual $\mathcal{Z}$-polyhedra. We therefore only need to show that the preimage of a single $\mathcal{Z}$-polyhedron, represented by say $\mathcal{Z}$, is a union of $\mathcal{Z}$-polyhedra . As a matter of fact, it is precisely a single $\mathcal{Z}$-polyhedron. Let the representation $\mathcal{Z}$ be

$$\{Lz + l | Qz + q \geq 0, z \in \mathbb{Z}^m\}$$

Let us find the preimage of $\mathcal{Z}$ by the function represented as $(Kz' + k \to Rz' + r)$. By the definition of an affine lattice function, K has full column rank, m' say. However, R need not have full column rank.

Let R have rank d and $H = \begin{pmatrix} H' & 0 \end{pmatrix}$ be its Hermite normal form such that H' has full column rank and $R = HU$. We may rewrite the preimage function as

$$(Kz' + k \rightarrow \begin{pmatrix} H' & 0 \end{pmatrix} Uz' + r)$$

By introducing coordinates $z'' = Uz'$ we get the equivalent function

$$(KU^{-1}z'' + k \rightarrow \begin{pmatrix} H' & 0 \end{pmatrix} z'' + r)$$

Let us express z'' as $\begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$ and KU^{-1} as $\begin{pmatrix} K_1 & K_2 \end{pmatrix}$ such that z_1 are the first d coordinates of z'' and K_1 is the matrix formed by the first d columns of KU^{-1}. The affine lattice function can be rewritten as

$$(K_1 z_1 + K_2 z_2 + k \rightarrow H' z_1 + r)$$

If the iteration point $H'z_1 + r$ does not lie in the $\mathcal{Z}$-polyhedron for a certain value of z_1, its preimage does not exist. Specifically, a preimage exists only for iteration points in the intersection the $\mathcal{Z}$-polyhedron represented by $\mathcal{Z}$ and the affine lattice represented by $\{H'z_1 + r | z_1 \in \mathbb{Z}^d\}$.

Let the intersection of the affine lattices represented by $\{H'z_1 + r | z_1 \in \mathbb{Z}^d\}$ and $\{Lz + l | z \in \mathbb{Z}^m\}$ be the affine lattice, say represented by $\mathcal{L} = \{L^{\#}z^{\#} + l^{\#} | z^{\#} \in \mathbb{Z}^{m^{\#}}\}$ where $L^{\#}$ has full column rank. The affine lattice represented by $\mathcal{L}$ may be empty, in which case the preimage is also empty. If $\mathcal{L}$ represents a non-empty affine lattice, we have the following relationships.

$$L^{\#} = LS, l^{\#} = Ls + l$$
$$L^{\#} = H'S', l^{\#} = H's' + r$$

where S and S' are matrices and s and s' are vectors.

Taking the intersection of the $\mathcal{Z}$-polyhedron represented by $\mathcal{Z}$ and the affine lattice represented by $\mathcal{L} = \{L^{\#}z^{\#} + l^{\#} | z^{\#} \in \mathbb{Z}^{m^{\#}}\}$, we get the following $\mathcal{Z}$-polyhedron.

$$\mathcal{Z} \cap \mathcal{L} = \{L^{\#}z^{\#} + l^{\#} | QSz^{\#} + (q + Qs) \geq 0, z^{\#} \in \mathbb{Z}^{m^{\#}}\} \tag{4.7}$$

The preimage by the function exists only for iterations in the $\mathcal{Z}$-polyhedron given above. Therefore, we may safely restrict the function to map to iteration points in the lattice $\{L^{\#}z^{\#} + l^{\#} | z^{\#} \in \mathbb{Z}^{m^{\#}}\}$. We will first characterize values of z_1 for which $H'z_1 + r$ lies in the lattice $\{L^{\#}z^{\#} + l^{\#} | z^{\#} \in \mathbb{Z}^{m^{\#}}\}$.

Substituting for $L^{\#}$ and $l^{\#}$, we have

$$\begin{aligned} L^{\#}z^{\#} + l^{\#} &= H'S'z^{\#} + H's' + r \\ &= H'(S'z^{\#} + s') + r \end{aligned}$$

This equals $H'z_1 + r$ to get $z_1 = S'z^{\#} + s'$. Substituting for z_1, we get the following restricted function by which the preimage is desired.

$$(K_1(S'z^{\#} + s') + K_2 z_2 + k \rightarrow H'(S'z^{\#} + s') + r)$$

Since $H'(S'z^{\#} + s') + r = L^{\#}z^{\#} + l^{\#}$, the preimage is

$$\left\{ \begin{pmatrix} K_1S' & K_2 \end{pmatrix} \begin{pmatrix} z^{\#} \\ z_2 \end{pmatrix} + (k + K_1 s') \right. \\ \left. \left| \begin{pmatrix} QS & 0 \end{pmatrix} \begin{pmatrix} z^{\#} \\ z_2 \end{pmatrix} + (q + Qs) \geq 0, \begin{pmatrix} z^{\#} \\ z_2 \end{pmatrix} \in \mathbb{Z}^{m^{\#}+m'-d} \right\} \tag{4.8}$$

Since, $\begin{pmatrix} K_1 & K_2 \end{pmatrix}$ has full column rank, $\begin{pmatrix} K_1S' & K_2 \end{pmatrix}$ will have full column rank *iff* K_1S' has full column rank. Since $L^{\#} = H'S'$ and both $L^{\#}$ and H' have full column rank, S' must necessarily have full column rank, which in turn implies that K_1S' has full column rank. The coordinate polyhedra for (4.8) may not have the entire universe as its context, in which case, we would bring it to the required representation through the transformation in step 2 presented in section 4.2.1.

Discussion We wish to mention that it is necessary to consider the general case where the affine lattice in the range of a function may not necessarily be a subset of the affine lattice on which the $\mathcal{Z}$-polyhedron is based. If otherwise, we had restricted the function to have its range defined over the appropriate affine sublattice (which would have significantly simplified the calculations presented above), the semantic equivalence of $\mathcal{Z}$-polyhedra based solely on the set of iteration points, would fail. For example, the function $(i \rightarrow i)$ would be defined on the $\mathcal{Z}$-polyhedron represented by $\{i | i \geq 0, i \in \mathbb{Z}\}$ but would not be defined on the equivalent union of two $\mathcal{Z}$-polyhedra represented by $\{2i | i \geq 0, i \in \mathbb{Z}\}$ and $\{2i + 1 | i \geq 0, i \in \mathbb{Z}\}$.

4.3.5 Change of Basis

Before we study arbitrary images of unions of $\mathcal{Z}$-polyhedra by affine lattice functions, we will present a restricted case, called the change of basis, in which

1. the affine lattice function represented as $(Kz' + k \rightarrow Rz' + r)$ is such that K as well as R have full column rank, m'.

2. For each $\mathcal{Z}$-polyhedron , $\{L_i z_i + l_i | Q_i z_i + q_i \geq 0, z_i \in \mathbb{Z}^{m_i}\}$, in the union, its corresponding affine lattice $\{L_i z_i + l_i | z_i \in \mathbb{Z}^{m_i}\}$ is a sublattice of the affine lattice $\{Kz' + k | z' \in \mathbb{Z}^{m'}\}$ in the domain of the affine lattice function.

The change of basis is a frequently used space reindexing function to perform semantically equivalent transformations of specifications. We will first present the image of a $\mathcal{Z}$-polyhedron by the change of basis function. Then, we will discuss the transformation of an equational specification.

By property 2 above,

$$L_i = KS_i, l_i = Ks_i + k$$

where S_i are matrices and s_i are vectors. We may safely restrict the change of basis of a $\mathcal{Z}$-polyhedron to be defined over the affine sublattice corresponding to the $\mathcal{Z}$-polyhedron. The restricted change of basis is

$$(K(S_i z_i + s_i) + k \to R(S_i z_i + s_i) + r)$$

which is equivalent to

$$(L_i z_i + l_i \to RS_i z_i + (r + Rs_i))$$

The image can be represented as

$$\{RS_i z_i + (r + Rs_i) | Q_i z_i + q_i \geq 0, z_i \in \mathbb{Z}^{m_i}\}$$

Note, RS_i has full column rank since both R and S_i have full column rank. Also the coordinate polyhedron is identical to the original $\mathcal{Z}$-polyhedron, and so has the entire universe as its context.

The transformation of the specification under a change of basis, f of the variable X is the following.

1. Replace the domain of X by its image under f.
2. Replace all occurrences of X on the *rhs* of any equation by $X.f$
3. For the equation defining X, add a dependence by f^{-1} on the expression on its *rhs*.

4.3.6 Image

We will now discuss images of unions of $\mathcal{Z}$-polyhedra by affine functions on lattices. The image of a union of $\mathcal{Z}$-polyhedra by a function is the union of images of individual $\mathcal{Z}$-polyhedra.

We already know that the image of a $\mathcal{Z}$-polyhedron by an arbitrary affine function is an LBL which is a more general class of objects than $\mathcal{Z}$-polyhedra. Here, we will prove a surprising result that any LBL can be expressed as a union of $\mathcal{Z}$-polyhedra. As a corollary, Presburger sets can be expressed as a union of $\mathcal{Z}$-polyhedra (existential variables may be eliminated by taking the image by affine lattice functions).

Consider the image of the $\mathcal{Z}$-polyhedron represented as $\mathcal{Z} = \{Lz + l | Qz + q \geq 0, z \in \mathbb{Z}^m\}$ by the affine lattice function represented as $(Kz' + k \rightarrow Rz' + r)$ where K has full column rank, m' say. By definition, the function provides a mapping from the iteration point $Kz' + k$ to the iteration point $Rz' + r$. The image $Rz' + r$ only exists for those values of z' for which the lattice point $Kz' + k$ lies in the $\mathcal{Z}$-polyhedron represented by $\mathcal{Z}$. Thus, an image only exists for points in the intersection of the $\mathcal{Z}$-polyhedron represented by $\mathcal{Z}$ and the affine lattice $\{Kz' + k | z' \in \mathbb{Z}^{m'}\}$.

Consider the intersection of the affine lattices represented by $\mathcal{L} = \{Lz + l | z \in \mathbb{Z}^m\}$ and $\mathcal{L}' = \{Kz' + k | z' \in \mathbb{Z}^{m'}\}$. Let this be the affine lattice, say represented by $\mathcal{L}'' = \{L''z'' + l'' | z'' \in \mathbb{Z}^{m''}\}$ where L'' has full column rank. The affine lattice represented by $\mathcal{L}''$ may be empty, in which case the image is also empty. If $\mathcal{L}''$ represents a non-empty lattice, we have the following relationships.

$$L'' = LS, l'' = Ls + l$$
$$L'' = KS', l'' = Ks' + k$$

where S and S' are matrices and s and s' are vectors.

Taking the intersection of the $\mathcal{Z}$-polyhedron represented by $\mathcal{Z}$ and the affine lattice represented by $\mathcal{L}'' = \{L''z'' + l'' | z'' \in \mathbb{Z}^{m''}\}$, we get the following $\mathcal{Z}$-polyhedron.

$$\mathcal{Z} \cap \mathcal{L}'' = \{L''z'' + l'' | QSz'' + (q + Qs) \geq 0, z'' \in \mathbb{Z}^{m''}\} \tag{4.9}$$

An image by the function exists only for iterations in the $\mathcal{Z}$-polyhedron given above. Therefore, we may safely restrict the function to map iteration points in the lattice $\{L''z'' + l'' | z'' \in \mathbb{Z}^{m''}\}$. The restricted function is

$$(K(S'z'' + s') + k \rightarrow R(S'z'' + s') + r)$$

which is equivalent to

$$(L''z'' + l'' \rightarrow RS'z'' + (r + Rs')) \tag{4.10}$$

With equations (4.9) and (4.10), the image may be represented as

$$\{Tz'' + t | Q'z'' + q' \geq 0, z'' \in \mathbb{Z}^{m''}\} \tag{4.11}$$

where $T = RS'$, $t = (r + Rs')$, $Q' = QS$ and $q' = (q + Qs)$

The set in (4.11) is not necessarily a $\mathcal{Z}$-polyhedron since there is no guarantee on the rank of T. We will now provide an algorithm that transforms such a set into a union of $\mathcal{Z}$-polyhedra.

Let the context of the coordinate polyhedron $\mathcal{P} = \{z'' | Q'z'' + q' \geq 0, z'' \in \mathbb{Z}^{m''}\}$ be given by $\ker(Q_0)$. Using the polynomial time condition for $\mathcal{Z}$-polyhedra testing by Le Verge, if $\ker(A) \subseteq \ker(Q')$ for $A = \begin{pmatrix} T \\ Q_0 \end{pmatrix}$, the set in (4.11) is a $\mathcal{Z}$-polyhedron by theorem 2.

When this condition fails, we will iterate the following steps (potentially producing multiple subsets), successively reducing the number of dimensions of the context of the coordinate polyhedron, $\ker(Q_0)$, until the conditions for theorem 2 are satisfied. In the limiting case, the context of the coordinate polyhedron is its lineality space, $\ker(Q')$, and $\ker(A) = \ker(T) \cap \ker(Q_0) \subseteq \ker(Q')$ is trivially satisfied. Upon the termination of this algorithm, we have a union of $\mathcal{Z}$-polyhedra.

Pick any constant vector v in $\ker(A) \backslash \ker(Q')$. Let $\mathcal{P}'$ be the translation of $\mathcal{P}$ along v such that

$$\mathcal{P} - \mathcal{P}' = \bigcup_{c_i \in \mathcal{C}} \mathcal{P} \cap (a_i^T z'' + \alpha_i < a_i^T v) \tag{4.12}$$

where $\mathcal{P}$ satisfies the set of constraints $\mathcal{C}$ and c_i is a constraint in $\mathcal{C}$ of the form $(a_i^T z'' + \alpha_i \geq 0)$. If $\mathcal{P} - \mathcal{P}'$ is an empty union, then we choose to translate along $-v$. We are guaranteed that one of the directions results in a non-empty union since $v \notin \ker(Q')$. In the remainder, we will assume that the translation along v results in a non-empty union.

The key insight into our proof is that the image of $\mathcal{P}$ by the affine lattice function equals the image of $\mathcal{P} - \mathcal{P}'$ by the affine lattice function. This is true since any element $z_1 \in \mathcal{P} \cap \mathcal{P}'$ is of the form $z_0 + \gamma v$ where $z_0 \in \mathcal{P} - \mathcal{P}'$ and γ is a constant. Since v lies in $\ker(T)$, its image satisfies the following property

$$Tz_1 + t = T(z_0 + \gamma v) + t = Tz_0 + t$$

For each non-empty element in the union of polyhedra in (4.12), create a union of polyhedra of the form

$$\mathcal{P}_{i,j} = \mathcal{P} \cap (a_i^T z'' + \alpha_i = \beta_j)$$

where $\beta_j \in \{0, \ldots, a_i^T v - 1)$. Now we claim that if the context of $\mathcal{P}_{i,j}$ is $\ker(Q_{0_{i,j}})$ then

$$\ker \begin{pmatrix} T \\ Q_{0_{i,j}} \end{pmatrix} \subset \ker \begin{pmatrix} T \\ Q_0 \end{pmatrix} \tag{4.13}$$

where the inclusion is strict. This is because

1. a_i^T is linearly independent of the rows of $\begin{pmatrix} T \\ Q_0 \end{pmatrix}$ since $\begin{pmatrix} T \\ Q_0 \end{pmatrix} v = 0$ and $a_0^T v \neq 0$.
2. a_i^T is a row of $Q_{0_{i,j}}$ and not a row of Q_0.

One iteration of this transformation returns an equivalent representation of the set in (4.11), that is a union of form

$$\bigcup_{i,j} \{Tz'' + t | Q'_{i,j} z'' + q'_{i,j} \geq 0, z'' \in \mathbb{Z}^{m''}\}$$

where $\mathcal{P}_{i,j} = \{z'' | Q'_{i,j} z'' + q'_{i,j} \geq 0, z'' \in \mathbb{Z}^{m''}\}$

We are guaranteed that the algorithm will eventually terminate as a consequence of the strict inclusion presented in (4.13).

Finally, we must to mention that this result does not violate the complexity results for deciding whether an LBL is a $\mathcal{Z}$-polyhedron since there may be an exponential number of elements in our union.

Example 7 *Let us now apply this algorithm to the* LBL *given in Figure 4.1. The* LBL[3] *is the image of the integer polyhedron* $\{i, j | 0 \leq i, 0 \leq j, i + j \leq 3\}$ *by the function* $(i, j \rightarrow i + 3j)$. *Since the lattice corresponding to the* $\mathcal{Z}$*-polyhedron and the lattice in the domain of the function are identical, we do not need to restrict either the* $\mathcal{Z}$*-polyhedron or the affine lattice function. Also* $ker(Q_0)$ *is the entire universe and* $ker(Q') = \{0\}$.

Picking the vector $v = \begin{pmatrix} 3 \\ -1 \end{pmatrix}$ *in* $ker(T)$ *we get* $\mathcal{P} - \mathcal{P}'$ *as a union of three integer polyhedra that, in addition to the constraints of* $\mathcal{P}$, *satisfy* $(j = 0)$, $(i + j = 3)$ *and* $(i + j = 2)$ *respectively. The image of these three integer polyhedra result in the sets* $\{i | 0 \leq i \leq 3\}$, $\{3 + 2i | 0 \leq i \leq 3\}$ *and* $\{2 + 2i | 0 \leq i \leq 2\}$ *respectively, each of which is a* $\mathcal{Z}$*-polyhedron. Their union equals the given* LBL. *Figure 4.2A shows the resultant union of* $\mathcal{Z}$*-polyhedra. Figure 4.2B shows the union of* $\mathcal{Z}$*-polyhedra that would be obtained if we had chosen the vector* $-v$.

4.3.7 Function Composition

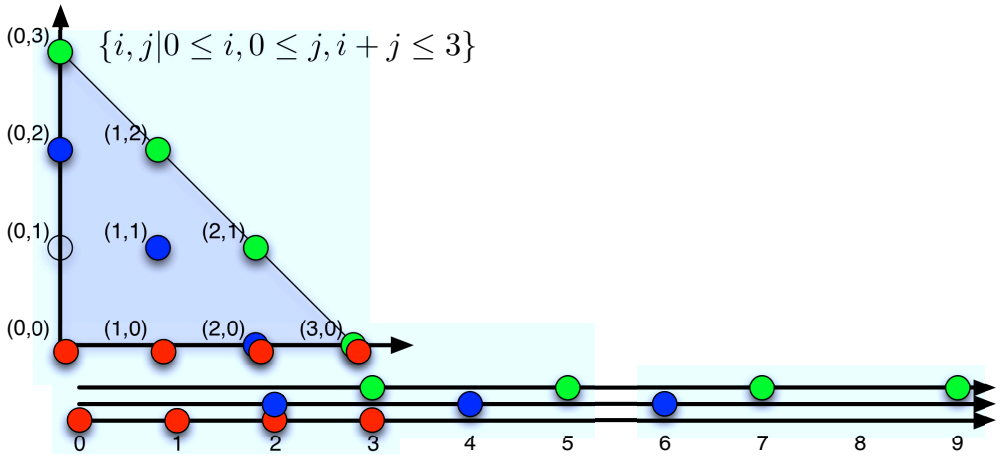
An important property of the family of affine lattice functions is that it is closed under function composition. Consider the function composition $f \circ g$, where

$$\begin{aligned} f &\equiv (Kz_1 + k \rightarrow Rz_1 + r) \\ g &\equiv (Tz_2 + t \rightarrow Vz_2 + v) \end{aligned}$$

In the function composition, $f \circ g$, the domain of f is the range of g. By the definition of an affine lattice function, K has full column rank. However, V need not have full column rank. Let V have rank d and $H = \begin{pmatrix} H' & 0 \end{pmatrix}$ be its Hermite normal form such that H' has full column rank and $V = HU$.

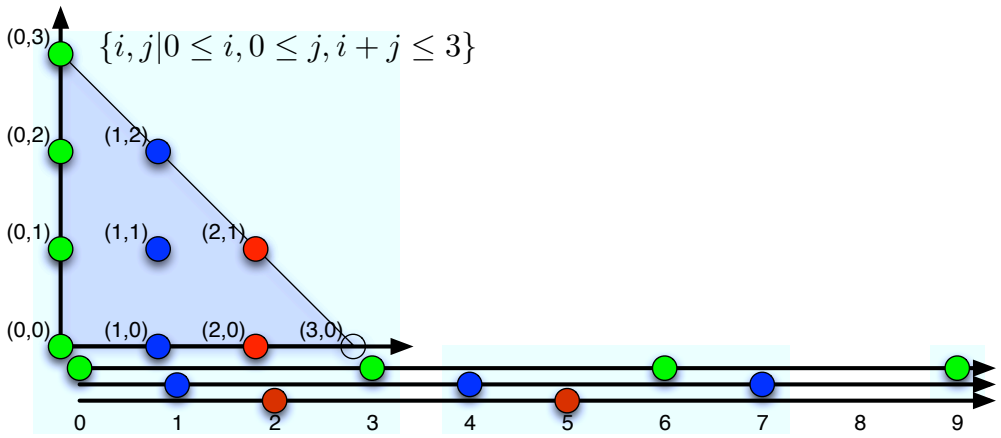
---

[3]The LBL has a constant number of iterations, therefore can be trivially expressed as a union of $\mathcal{Z}$-polyhedra or even a union of integer polyhedra. However, in this example, we will elaborate the steps of our algorithm (applicable to infinite or parameterized sets).



$\{3+2i|0 \leq i \leq 3\} \cup \{2+2i|0 \leq i \leq 2\} \cup \{i|0 \leq i \leq 3\}$

(A)



$\{3i|0 \leq i \leq 3\} \cup \{3i+1|0 \leq i \leq 2\} \cup \{3i+2|0 \leq i \leq 1\}$

(B)

Figure 4.2: LBL as a union of $\mathcal{Z}$-polyhedra

We may rewrite g as

$$(Tz_2 + t \to \begin{pmatrix} H' & 0 \end{pmatrix} Uz_2 + v)$$

By introducing the coordinates $z^{\#} = Uz_2$ we get

$$g \equiv (TU^{-1}z^{\#} + t \to \begin{pmatrix} H' & 0 \end{pmatrix} z^{\#} + v)$$

Let us express $z^{\#}$ as $\begin{pmatrix} z' \\ z'' \end{pmatrix}$ and TU^{-1} as $\begin{pmatrix} T_1 & T_2 \end{pmatrix}$ such that z' are the first d coordinates of $z^{\#}$ and T_1 is the matrix formed by the first d columns of TU^{-1}. We can now rewrite g as

$$(T_1 z' + T_2 z'' + t \to H' z' + v) \tag{4.14}$$

Let the intersection of the domain of f and the range of g be the lattice represented as $\{Lz + l | z \in \mathbb{Z}^m\}$ where L has full column rank such that

$$\begin{array}{l} L = KS_1, l = Ks_1 + k \\ L = H'S_2, l = H's_2 + v \end{array}$$

where S_1, S_2 are matrices and s_1 and s_2 are vectors. Now consider the restricted functions, f' and g' which are subset of f and g respectively such that the domain of f' and the range of g' is the lattice represented by $\{Lz + l | z \in \mathbb{Z}^m\}$.

$$\begin{aligned} f' &\equiv (K(S_1 z + s_1) + k \to R(S_1 z + s_1) + r) \\ g' &\equiv (T_1(S_2 z + s_2) + T_2 z'' + t \to H'(S_2 z + s_2) + v) \end{aligned}$$

Since $K(S_1 z + s_1) + k = H'(S_2 z + s_2) + v = Lz + l$, we may rewrite f' and g' as

$$(Lz + l \to R(S_1 z + s_1) + r)$$

and

$$(T_1(S_2 z + s_2) + T_2 z'' + t \to Lz + l)$$

respectively. As the composition, $f \circ g$ is defined only when the domain of f is the range of g, it equals

$$\begin{aligned} f' \circ g' &\equiv (T_1 S_2 z + T_2 z'' + (t + T_1 s_2) \to RS_1 z + (r + Rs_1)) \\ &\equiv \left(\begin{pmatrix} T_1 S_2 & T_2 \end{pmatrix} \begin{pmatrix} z \\ z'' \end{pmatrix} + (t + T_1 s_2) \to \begin{pmatrix} RS_1 & 0 \end{pmatrix} \begin{pmatrix} z \\ z'' \end{pmatrix} + (r + Rs_1) \right) \end{aligned}$$

which is also an affine lattice function.

4.4 Conclusions

We presented a novel representation and interpretation of $\mathcal{Z}$-polyhedra and prove the various closure properties of the family of unions of $\mathcal{Z}$-polyhedra required to extend the polyhedral model. In addition, we prove closure of domains in the $\mathcal{Z}$-polyhedral model under images by arbitrary affine functions which had been a major limitation of the polyhedral model. As a corollary, we have proved that Presburger sets and unions of LBLs, widely assumed to be richer classes, are equivalent to unions of $\mathcal{Z}$-polyhedra.

Chapter 5

$\mathcal{Z}$-Polyhedral Model: Value-Based Dependence Analysis

Value based dependence analysis involves the extraction of ordering constraints between statement instances at different loop iterations to guarantee the validity of optimizing transformations. The analysis needs to take into account complex memory effects including false dependences resulting from write-after-write and write-after-read. True dependences are also non-trivial to derive from loop nests since multiple values, usually produced by different loop iterations of different statements, may be stored at a particular memory location. We must find the last producer before the consumption of a value to derive the dependence.

Currently, exact dependences can be derived from ACLs [Fea91] to produce equations in the polyhedral model. In this chapter, we will to analyze a more general class of loop programs to yield the more general specifications of the $\mathcal{Z}$-polyhedral model. We will consider loop programs with

1. non-unit stride
2. modulo operations, integer divisions, max and min in loop bounds, guards and memory access functions, and
3. existential variables in guards. Existential variables may be implemented by flags which get set if a loop executes at least one iteration.

This is joint work with DaeGon Kim.

5.1 Relevant Mathematical Background

Before proceeding to the technical contributions of this chapter, let us recall some background that will be needed and present the relationship between Presburger sets and $\mathcal{Z}$-polyhedra.

An Integer Polyhedra is a subset of $\mathbb{Z}^n$ the elements of which satisfy a finite number of affine inequalities with integer coefficients. A parameterized integer polyhedron is an integer polyhedron where some indices are interpreted as size parameters.

An Affine Lattices generated by a constant matrix L and the constant offset vector l, is the set of all vectors obtained by adding the constant l to integer linear combinations of the columns of L. An affine lattice is a subset of $\mathbb{Z}^n$ and can be represented as $\{Lz + l | z \in \mathbb{Z}^m\}$ where L and l are an $n \times m$ matrix and n-vector respectively. We call z the coordinates of the affine lattice.

A $\mathcal{Z}$-Polyhedron is the intersection of an integer polyhedron and an affine lattice. We use the following representation of $\mathcal{Z}$-polyhedra.

$$\{Lz + l | Qz + q \geq 0, z \in \mathbb{Z}^m\}$$

where the columns of L constitute a basis of the affine lattice $\{Lz + l | z \in \mathbb{Z}^m\}$ and valid values of z are given by the *coordinate polyhedron* $\{z | Qz + q \geq 0, z \in \mathbb{Z}^m\}$ of the $\mathcal{Z}$-polyhedron. Iteration points of the $\mathcal{Z}$-polyhedral domain are points of the affine lattice corresponding to valid coordinates.

A parameterized $\mathcal{Z}$-polyhedron is a $\mathcal{Z}$-polyhedron where some rows of its corresponding affine lattice are interpreted as size parameters. We can always express a $\mathcal{Z}$-polyhedron such that its size parameters have a one-to-one correspondence with certain indices of its coordinate polyhedron.

5.1.1 Presburger Sets

Presburger (integer) formulae consists of affine inequalities over integer variables and logical operators combining affine inequalities. Formally, a Presburger formula is defined as follows [Mac99]:

$$f = a \mid \exists x.f \mid f \wedge f \mid f \vee f \mid \neg f$$

where

1. $a = t < t$

Presburger Set Construction Rule	Corresponding operation on a union of $\mathcal{Z}$-polyhedra
Presburger atom $a < a'$	A half space whose constraint is $a < a'$
$f_1 \wedge f_2$	$\mathcal{D}_{f_1} \cap \mathcal{D}_{f_2}$
$f_1 \vee f_2$	$\mathcal{D}_{f_1} \cup \mathcal{D}_{f_2}$
$\neg\ f$	$\mathcal{U} \setminus \mathcal{D}_f$
$\exists x\ .\ f$	image of $\mathcal{D}_f$ by the function $((z, x) \rightarrow z)$

Table 5.1: Conversion rules from a Presburger set to a union of $\mathcal{Z}$-polyhedra; f, f_1 and f_2 are Presburger formulae and $\mathcal{D}_f$, $\mathcal{D}_{f_1}$ and $\mathcal{D}_{f_2}$ are the corresponding unions of $\mathcal{Z}$-polyhedra

2. $t = 0 \mid 1 \mid x \mid t + t \mid -t$, and

3. x represents an integer variable.

The Presburger atom a is an single affine expression over integer variables denoting a half-space, which is trivially an integer polyhedron. The universe, $\mathcal{U}$, is also an integer polyhedron. The family of unions of integer polyhedra is closed under intersection, union and difference. In the absence of the existential qualifier, a Presburger formula f can be expressed as a union of integer polyhedra.

The existential qualifier can be viewed as a projection from a higher dimensional space that eliminates the variable. The family of unions of integer polyhedra is not closed under projections. However, recall that in chapter 4, we showed that the family of unions of $\mathcal{Z}$-polyhedra is closed under intersection, union and difference as well as image by generalized affine functions. As a result, any Presburger set can always be written as a union of $\mathcal{Z}$-polyhedra. The formal conversion rules are given in Table 5.1. The existential qualification on variable x can be viewed as the image of the set $\mathcal{D}_f$ satisfying the formula f by the affine function that eliminates the variable x. Note that a point z exists in the set described by $\exists x.f$ if and only if $(z, x) \in \mathcal{D}_f$.

To derive the iteration space of more general loops than ACLs, we will first express them as Presburger sets. Then, using the conversion rules presented above, we will obtain an equivalent representation of the iteration space as a union of $\mathcal{Z}$-polyhedra.

5.2 Motivating Example for $\mathcal{Z}$-Polyhedral Analysis

Consider the following loop program.

```
for i = 0 to N
  for j = 0 to N
    X[i,j] = (i%2==0?X[i,j-1]:Y[i-1,j]);
```

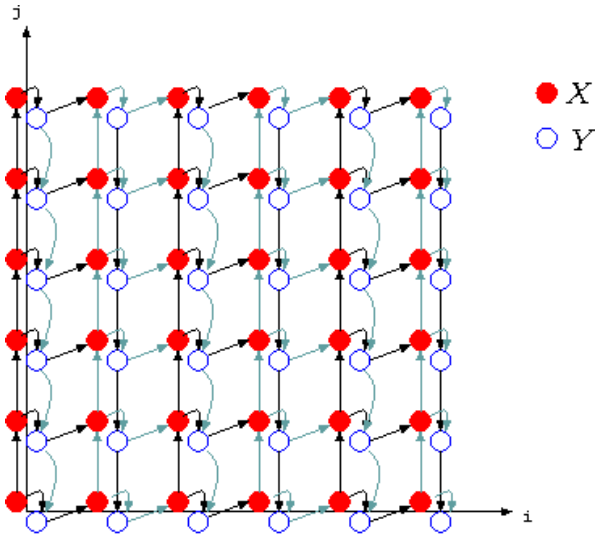


Figure 5.1: Motivating Example: Data dependence graph for $N = 5$ in the Polyhedral model

```
for j = N downto 0
  Y[i,j] = (i%2==0?X[i,j]:Y[i,j+1]);
```

As explained previously, extracting data dependence relations is the first and most critical step to guarantee the validity of any transformations on the loop program. The semantics of the program will not be preserved if dependences are not respected by transformations.

The data dependence information obtained by [Fea91] is shown in Figure 5.1. A node in data dependence graph represents an iteration point of a statement, and the arrow between nodes denotes data-flow between two operations.

Say, we want to parallelize this computation. Because of the vertical dependence on X, the execution of X will be in increasing order of the j index. However, the execution of Y will be in decreasing order of j. Together with the other dependences between X and Y, this computation must be done sequentially exactly like the original loops, that is, for each i, first compute X in the increasing order of j and then compute Y in the decreasing order of j.

In the figure 5.1, true dependences are denoted with black arrows and represent data-dependences in the input program. False dependences are an artifact of the approximation used by the dependence analysis; in this case w.r.t. the domain of definition of the branches of the conditional expression. If we consider only the true dependences in the specification (as

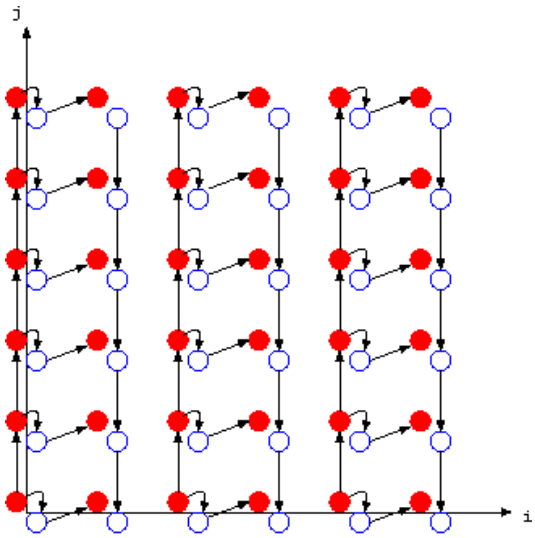


Figure 5.2: Motivating Example: Data dependence graph for $N = 5$ in the $\mathcal{Z}$-polyhedral model

shown in figure 5.2), we can see that the following (parallel) execution order is also valid.

$$\lambda_X(i,j) = \begin{cases} i \text{ even} & : \quad j \\ i \text{ odd} & : \quad j+2 \end{cases}$$

$$\lambda_Y(i,j) = \begin{cases} i \text{ even} & : \quad j+1 \\ i \text{ odd} & : \quad N-j \end{cases}$$

Here, the schedules $\lambda_X(i,j)$ (respectively, $\lambda_Y(i,j)$) represents the time instant at which the computation $X[i,j]$ (respectively, $Y[i,j]$) is executed. To derive this schedule, we need to be able to extract the precise dependences that are over $\mathcal{Z}$-polyhedral domains. The remainder of this chapter will present the technique for this.

The parallel execution order is realized by the loop program given below (for the sake of simplicity, let us assume that N is odd).

```
for j = 0 to N
  forall i to (N/2)
    Y[2i+1,N-j]=Y[2i+1,N-j+1];
  forall i to (N/2)
    X[2i,j]=X[2i,j-1];
  forall i to (N/2)
    Y[2i,j]=X[2i,j];
  forall i to (N/2)
```

```
X[2i+1,j]=Y[2i,j];
```

5.3 Illustrating Example

Here, we illustrate the problem that we wish to solve and our approach with the help of the following example. We will express iteration spaces as unions of $\mathcal{Z}$-polyhedra so that further analyses and transformations, especially those that can be generalized from the polyhedral equational model to unions of $\mathcal{Z}$-polyhedra, can be applied. In this example, our goal is to extract the data dependence relations between statement iterations in the following loop nest.

```
for i=2 to 2N
  C[i]=0;                        --- R
for i=1 to N
  for j=(i div 2) to N step 2
    C[i+j]=C[i+j]+A[i]*B[j];   --- S
```

Here, `div` represents integer division (also denoted by `/`). The iteration space $\mathcal{D}$ of a statement is the set of all valid loop indices of its surrounding loops. The iteration space $\mathcal{D}_R$ of statement R is $\{i \mid 2 \leq i \leq 2N\}$. It is well known that the iteration space of a statement in ACLs can be represented by a parameterized integer polyhedron. However, $\mathcal{D}_S$, the iteration space of statement S, cannot be represented by a polyhedron because of integer division on lower bounds and the non-unit stride.

However, we can write $\mathcal{D}_S$ as the following set:

$$\mathcal{D}_S = \{i, j \mid 1 \leq i \leq N; (i \text{ div } 2) \leq j \leq N; (j - (i \text{ div } 2)) \bmod 2 = 0\}$$

Now, we will derive a Presburger set to express the iteration space by introducing new variables and using the division rule. We introduce a new integer variable $\alpha = i$ div 2.

$$\mathcal{D}_S = \{i, j \mid \exists \alpha, 1 \leq i \leq N; \alpha \leq j \leq N; (j - \alpha) \bmod 2 = 0; \alpha = i \text{ div } 2\}$$

By the division rule, we replace $\alpha = i$ div 2 with $0 \leq i - 2\alpha \leq 1$.

$$\mathcal{D}_S = \{i, j \mid \exists \alpha, 1 \leq i \leq N; \alpha \leq j \leq N; (j - \alpha) \bmod 2 = 0; 0 \leq i - 2\alpha \leq 1\}$$

The modular operation $(j - \alpha) \bmod 2 = 0$ can be removed by introducing a new integer variable β and setting $j - \alpha = 2\beta$. Finally, we get the following Presburger set.

$$\mathcal{D}_S = \{i, j \mid \exists \alpha, \beta, 1 \leq i \leq N; \alpha \leq j \leq N; j - \alpha = 2\beta; 0 \leq i - 2\alpha \leq 1\}$$

We can view this Presburger set as the image of

$$\mathcal{T} = \{i, j, \alpha, \beta \mid 1 \leq i \leq N; \alpha \leq j \leq N; j - \alpha = 2\beta; 0 \leq i - 2\alpha \leq 1\}$$

by the projection $(i, j, \alpha, \beta \rightarrow i, j)$. An important point to note is that $(i, j, \alpha, \beta) \in \mathcal{T}$ if and only if $(i, j) \in \mathcal{D}_S$.

In this example, the execution of statement S at the iteration (i', j') requires the value of the array C at $[i' + j']$. We want to find *the* most recent instance $(i, j) \in \mathcal{D}_S$ of S that wrote its result to that location of the array C. Also, the answer (i, j) should be a function of (i', j').

First, we find the set of all instances that write to $C[i' + j']$ before the execution of the (i', j') instance. This set can be described by the following four conditions:

1. $(i, j) \in \mathcal{D}_S$ or $(i, j, \alpha, \beta) \in \mathcal{T}$,
2. $(i', j') \in \mathcal{D}_S$ or $(i', j', \alpha', \beta') \in \mathcal{T}$,
3. $i + j = i' + j'$ and
4. $i \leq i' - 1$

Now, we express this set as a single polyhedron $\mathcal{P}$ indexed by $(N, i', j', i, j, \alpha, \beta, \alpha', \beta')$ where (N, i', j') are parameters. Once we find the lexicographic maximum as a function of (N, i', j'), the first two indices will be the computation that writes the value read by the (i', j') instance. The parameterized lexicographic maximum of $\mathcal{P}$ can be obtained by the PIP solver [Fea88].

Given a valid (i', j') we are assured that (i, j) belongs $\mathcal{D}_S$ by the nature of the constraints posed above. However, we need to devise a mechanism to determine whether (i', j') belongs $\mathcal{D}_S$. To verify whether $(i'j') \in \mathcal{D}_S$, we need to find a point $(i', j', \alpha', \beta') \in \mathcal{T}$. Alternatively, we must explicitly represent the set without the use of extra variables. This iteration space can be represented directly, as a union of $\mathcal{Z}$-polyhedra.

This iteration space $\mathcal{D}_S$ for $N = 7$ is shown in Figure 5.3. By investigating the figure, we see that a translation from any valid iteration point along the $(2, 1)$ and $(0, 2)$ directions results in either a valid point, or is outside the polyhedral closure of the domain. However, note that it is impossible to move a black point to a white point with the $(2, 1)$ and $(0, 2)$ generators. The black and white points correspond to two different affine lattices. The set of black points can be described as follows:

$$\mathcal{B} = \{2y_1 + 1, y_1 + 2y_2 \mid 1 \leq 2y_1 + 1 \leq N; y_1 + 2y_2 \leq N; 0 \leq y_2\} \tag{5.1}$$

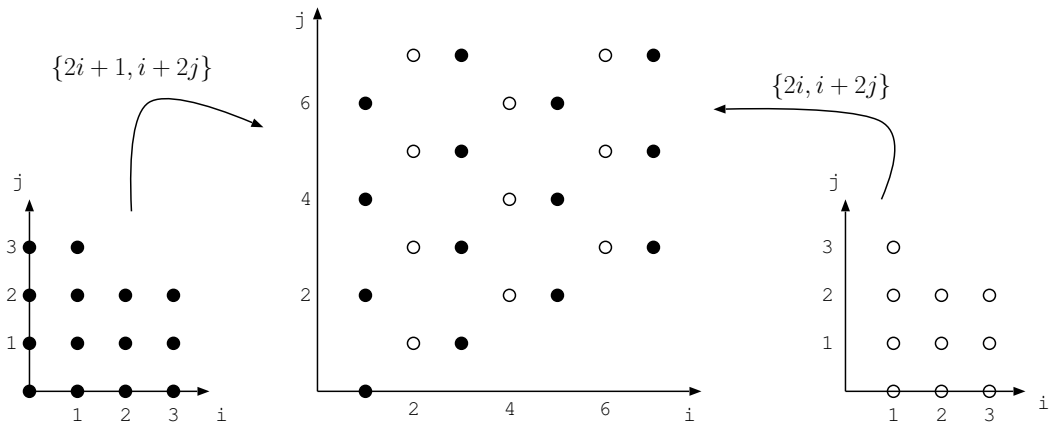


Figure 5.3: Illustrating Example: The diagram in the center is the iteration space, $\mathcal{D}_S$, of the statement S which is a union of two $\mathcal{Z}$-polyhedra; the leftmost object is the coordinate polyhedron of the black points in $\mathcal{D}_S$; and the rightmost object is the coordinate polyhedron of the white points in $\mathcal{D}_S$.

The set of white points can be described as:

$$\mathcal{W} = \{2y_1, y_1 + 2y_2 \mid 1 \leq 2y_1 \leq N; y_1 + 2y_2 \leq N; 0 \leq y_2\} \tag{5.2}$$

This iteration space can be computed in a systematic way, i.e., by computing the image of $\mathcal{T}$ by the function $(i, j, \alpha, \beta) \rightarrow (i, j)$. The image of polyhedra by affine functions can be represented as a union of $\mathcal{Z}$-polyhedra. More generally, any arbitrary Presburger set can be expressed by a union of $\mathcal{Z}$-polyhedra.

Finally, what is remaining is to express the lexicographic maximum obtained by the PIP solver in terms of y_1 and y_2 instead of i' and j' for each of the two $\mathcal{Z}$-polyhedra in the iteration space of statement S. This can be achieved by simply replacing i' by $2y_1 + 1$ and j' by $y_1 + 2y_2$ for $\mathcal{B}$ and i' by $2y_1$ and j' by $y_1 + 2y_2$ for $\mathcal{W}$.

The value at $C[i' + j']$ could also have been written by statement R in which case following the lines of [Fea91], we would repeat the above analysis for a producer at R.

5.4 Problem Definition and ILP Formulation

In this section, we want to answer the following three questions:

1. What is the set, $\mathcal{D}$, of valid iteration vectors over which a statement is executed, and what is its preferred representation.

2. For the execution of a statement at a certain iteration (given as a vector of loop indices

```
                Input program Grammar

Program     : Stmt*
Stmt        : Loop | If | Assignment
Loop        : for Index = ALExpr to ALExpr step PINT
                 Stmt*
If          : if '(' ABExprs ')'
                 Stmt*
ArrayRef    : Var[ALExpr*]
Assignment  : ArrayRef = func '(' ArrayRef* ')'
ALBExprs    : ALBExprs '&&' ALBExprs |
              ALBExprs '||' ALBExprs |
              exists '(' Index ',' ALBExprs ')' |
              ALBExpr
ALBExpr     : ALExpr Relation ALExpr
Relation    : '>' | '<' | '==' | '=<' | '=>'
ALExpr      : ALExpr '+' Factor |
              ALExpr '-' Factor |
              ALExpr mod PINT |
              ALExpr div PINT |
              max '(' ALExpr* ')' |
              min '(' ALExpr* ')'
Factor      : PINT Index | Index | PINT
```

Figure 5.4: Input program grammar. `Index` is an identifier; `ALExpr` (called affine lattice expressions) are functions of outer indices and size parameters; `PINT` is a positive integer; `ArrayRef` is a reference to a multidimensional array that is accessed by an affine lattice expression of outer indices and size parameters; `func is` a strict, side-effect free function; `div` is a integer division (also denoted by $'/'$); and `mod` is a modular operation (also denoted by $'\%'$).

and size parameters) requiring the value of a memory location, which instance of which statement produces the value.

3. How can we represent the iteration vector of the producer in terms of indices in the representation of $\mathcal{D}$.

5.4.1 Input programs

Here, we describe the class of programs that can be analyzed by our technique. This class contains ACLs but, in addition, allows loops with non-unit stride and arbitrarily nested mod, integer division, max and min operations on surrounding indices in loop bounds, guards and memory access functions. In addition, we also allow existential quantifiers in guards. The grammar for our input programs is given in Figure 5.4.

We now present two example programs: a stencil computation for red-black SOR, and a contrived example to illustrate the ILP formulation.

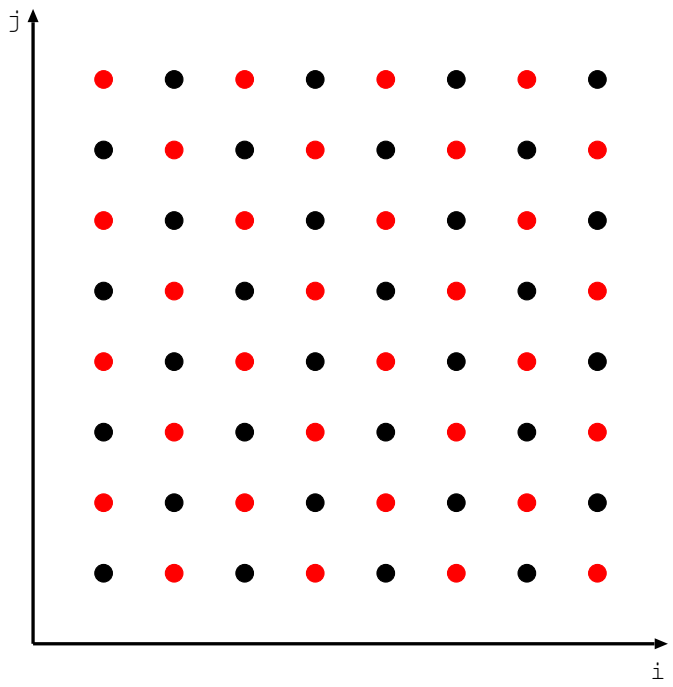


Figure 5.5: Red-Black SOR. For a certain t iteration, black points are executed. At the next iteration of t, the red points are executed.

Example 8 *Stencil computation (*RED-BLACK SOR*). The computation pattern is described in figure 5.5.*

```
for t = 0 to 2M
  for i = 1 to N-1
    for j = (t+i-1)%2+1 to N-1 step 2
      X[i,j]=(X[i-1,j]+X[i+1,j]+X[i,j-1]+X[i,j+1])/4;
```

Example 9 *Contrived example for* ILP *formulation:*

```
for i=N%5 to N step 2
  for j=i/2+i%4 to N+i%3
    X[j] = X[i/2+j%2];
```

In the chapter, we use "statement" and "assignment" interchangeably. However, we distinguish an assignment and its operation (or instance) which is a distinct execution of the statement. A statement can be executed multiple times, but an operation is unique in the entire execution of a program. An operation is described completely by a statement and its associated iteration vector. We denote an operation $z \in \mathcal{D}_S$ of a statement S by (S, z).

expression	Additional variable	Equivalent form	Additional Constraint
`ALExpr` div c	$\exists\alpha \in \mathbb{Z}$	α	$(0 \leq$ `ALExpr` $- c\alpha \leq c-1)$
`ALExpr` mod c	$\exists\alpha \in \mathbb{Z}$	`ALExpr` $- c\alpha$	$(0 \leq$ `ALExpr` $- c\alpha \leq c-1)$

Table 5.2: Equivalent form for `ALExpr` $\odot c$ where $\odot$ is an integer division or mod by the positive integer c.

5.4.2 Iteration Space as a Presburger Set

We will first explain how to express the iteration space of input loop nests as Presburger sets. A Presburger set can always be represented as an image of a union of higher dimensional polyhedra by the canonical projection that eliminates all existential variables. We use the higher dimensional polyhedron for our ILP formulation.

First, let us consider the simple case where the expression is a form of $\alpha = AE \odot c$ where $\odot$ is either an integer division or modular operation and c is a positive integer e.g., $\alpha = (i+j)$ div 2. By the division rule, we may replace it by $0 \leq i + j - c\alpha \leq c - 1$. For the general case, transformations are given in Table 5.2. In obtaining the equivalent form, we have introduced an existential variable and an additional constraint.

The procedure to construct a Presburger set from arbitrary loop bounds is to introduce a new existential variable for every mod or div operator until all the conditions are affine expressions of the form $a^T z + b > 0$ (with nested max and min). The maximum or minimum of affine expressions can be transformed into disjunctions and conjunctions of affine expressions. Our iteration space is then the projection of a union of polyhedra along the canonic directions corresponding to all the introduced existential variables.

We illustrate this algorithm through its application to Example 9. The iteration space can be expressed by

$$\{i, j \mid N\%5 \leq i \leq N; (i - N\%5)\%2 = 0; (i/2) + i\%4 \leq j \leq N + i\%3\}$$

Now, we replace $N\%5$ by $N - 5\alpha_1$ by introducing a new variable α_1 and the new constraint $0 \leq N - 5\alpha_1 \leq 4$. Then, we get

$$\{i, j \mid \exists\alpha_1, 0 \leq N-5\alpha_1 \leq 4; N-5\alpha_1 \leq i \leq N; (i-(N-5\alpha_1))\%2 = 0; (i/2)+i\%4 \leq j \leq N+i\%3\}$$

We introduce three more variables α_2, α_3 and α_4 to replace $i\%4$, $i\%3$ and $(i - (N - 5\alpha_1))\%2$ by $i - 4\alpha_2$, $i - 3\alpha_3$ and $(i - (N - 5\alpha_1)) - 2\alpha_4$, respectively along with the introduction of the constraints $0 \leq i - 4\alpha_2 \leq 3$, $0 \leq i - 3\alpha_3 \leq 2$ and $0 \leq (i - (N - 5\alpha_1)) - 2\alpha_4 \leq 1$ to result in the following set.

$$\{i,j \mid \exists \alpha_1,\alpha_2,\alpha_3,\alpha_4,$$
$$0 \le N - 5\alpha_1 \le 4; 0 \le i - 4\alpha_2 \le 3; 0 \le i - 3\alpha_3 \le 2; 0 \le (i - (N - 5\alpha_1)) - 2\alpha_4 \le 1;$$
$$N - 5\alpha_1 \le i \le N; 0 \le (i - (N - 5\alpha_1)) - 2\alpha_4 \le 1; (i/2) + (i - 4\alpha_2) \le j \le N + (i - 3\alpha_3)\}$$

To remove the only div (or $'/'$) operator, we introduce α_5 to replace $(i/2)$ by α_5 along with the introduction of the constraint $0 \le i - 2\alpha_5 \le 1$.

$$\{i,j \mid \exists \alpha_1,\alpha_2,\alpha_3,\alpha_4,\alpha_5,$$
$$0 \le N - 5\alpha_1 \le 4; 0 \le i - 4\alpha_2 \le 3; 0 \le i - 3\alpha_3 \le 2;$$
$$0 \le (i - (N - 5\alpha_1)) - 2\alpha_4 \le 1; 0 \le i - 2\alpha_5 \le 1;$$
$$N - 5\alpha_1 \le i \le N; 0 \le (i - (N - 5\alpha_1)) - 2\alpha_4 \le 1; \alpha_5 + (i - 4\alpha_2) \le j \le N + (i - 3\alpha_3)\}$$

Note that the constraints consists of only affine constraints. This Presburger set is equivalent to the projection of the polyhedron, $\mathcal{P}_{\text{contrived}}$:

$$\{i,j,\alpha_1,\alpha_2,\alpha_3,\alpha_4,\alpha_5 \mid$$
$$0 \le N - 5\alpha_1 \le 4; 0 \le i - 4\alpha_2 \le 3; 0 \le i - 3\alpha_3 \le 2;$$
$$0 \le (i - (N - 5\alpha_1)) - 2\alpha_4 \le 1; 0 \le i - 2\alpha_5 \le 1;$$
$$N - 5\alpha_1 \le i \le N; 0 \le (i - (N - 5\alpha_1)) - 2\alpha_4 \le 1; \alpha_5 + (i - 4\alpha_2) \le j \le N + (i - 3\alpha_3)\}$$

by the function $(i,j,\alpha_1,\alpha_2,\alpha_3,\alpha_4,\alpha_5 \to i,j)$.

Similarly, the iteration space in Example 8 can be constructed by introducing two additional variables to obtain the Presburger set, $\mathcal{D}_{\text{stencil}}$

$$\{t,i,j | \exists \alpha_1,\alpha_2, 0 \le t \le 2M; 0 \le i \le N-1;$$
$$0 \le t + i - 1 - 2\alpha_1 \le 1; (t + i - 2\alpha_1) \le j \le N - 1; j - (t + i - 2\alpha_1) - 2\alpha_2 = 0\}$$

In general, we may have guards on the execution of statements. This simply requires the addition of the constraints in the `if` condition to the iteration space of the corresponding statement. A subtle point to note is that user-defined existential variables may only occur in the `if` statements.

5.4.3 Predicate for execution order

This section focuses on the ordering of operations in input programs. In other words, it provides the condition that an operation (S, z') is executed before another operation (R, z). We denote the depth of common loops of S and R by N_{SR} and the textual order of statements S and R by the boolean predicate B_{SR} which is true when S is before R.

A operation (S, z') is executed before (R, z) if

$$z'[1 \ldots N_{SR}] \prec z[1 \ldots N_{SR}] \vee (B_{SR} \wedge z'[1 \ldots N_{SR}] = z[1 \ldots N_{SR}]) \tag{5.3}$$

When $N_{SR} = 0$, the condition will be just B_{SR}.

For our illustrating example, $N_{SR} = 0$ and B_{RS} is true. So, an operation of the assignment R precedes any operations of S. An operation (R, z) precedes (R, z') if $z \prec z'$. Similarly, (S, z) precedes (S, z') if $z \prec z'$.

Note that this sequencing predicate is the same as the predicate for ACLs. It is because steps in input programs are positive integers and the conditional statements do not affect the order of iteration vectors. A conditional statement only restricts the space of valid iteration vectors. For a more detail description, please refer to Feautrier's work [Fea91] on the exact dependence analysis of ACLs.

5.4.4 ILP formulation

We have statements whose iteration spaces are Presburger sets. For a consumer statement C whose memory (read) access function is M_C and a candidate producer statement P whose (write) access function is M_P.

We will assume that the iteration space of a statement given by the Presburger set $\mathcal{D}$ is such that it is the image of a *single* polyhedron $\mathcal{P}$ by the canonical projection along the directions corresponding to existential variables. This is restrictive. However, we will see that generalizations are straightforward extensions to the basic technique presented for this case.

First, we want to find all the instances of P that write to the location $M_C(z')$ before z' where $z' \in \mathcal{D}_C$. This set $\mathcal{C}$ can be described as:

$$\mathcal{C} = \{z \in \mathcal{D}_P \mid z' \in \mathcal{D}_C; M_P(z) = M_C(z'); z_c \prec z'_c\}$$

where z_c and z'_c are $z[1 \ldots N_{PC}]$ and $z'[1 \ldots N_{PC}]$ respectively. Note that depending on textual order z_c may be allowed to be equal to z'_c. Using the fact that $z \in \mathcal{D}_P$ iff $(z, \alpha) \in \mathcal{P}_P$ we get

$$\mathcal{C}' = \{(z, \alpha) \in \mathcal{P}_p \mid (z', \alpha') \in \mathcal{P}_C; M_P(z) = M_C(z'); z_c \prec z'_c\}$$

As a special case, if the memory function M_P and M_C are both affine functions, we have the same ILP formulation as that of ACLs. However, its naive formulation causes the problem that the ILP solution depends on z', as well as α'. We can avoid this by treating α' as unknown indices as well, rather than parameters, when constructing the parameterized polyhedra $\mathcal{Q}$ given as

$$\mathcal{Q} = \{(z', z, \alpha, \alpha') \mid (z', \alpha') \in \mathcal{P}_C; (z, \alpha) \in \mathcal{P}_C; M_P(z) = M_C(z'); z_c \prec z'_c\}$$

$\mathcal{Q}$ is parameterized by only z' and size parameters. In the solution of the ILP solver, (z, α, α') will be expressed as a function of z' and size parameters. The intuition behind such a formulation

is that the values of (α, α') do not matter as long as (α, α') simply exists and z is the corresponding lexicographic maximum.

Consider the stencil computation in Example 8. We want to formulate an ILP problem to find out the producer of (t', i', j') as a function of the indices t', i' and j' for the memory reference $X[i, j-1]$. Then, the set of candidate operations can be specified by

$$\begin{aligned}\mathcal{Q}_{\text{stencil}} \quad = \quad \{t', i', j', t, i, j, \alpha_1, \alpha_2, \alpha_1', \alpha_2' \mid (t', i', j') \in \mathcal{P}_{\text{stencil}}; (t, i, j) \in \mathcal{P}_{\text{stencil}}; \\ i' = i; j' - 1 = j; (t, i, j) \prec (t', i', j')\}\end{aligned}$$

This set is a polyhedra parameterized by (t', i', j'). Using the PIP solver, we find the indices of the last-write operation $(t, i, j, \alpha_1, \alpha_2, \alpha_3, \alpha_1', \alpha_2', \alpha_3')$ as a function of (t', i', j'). It is only the first three components, (t, i, j), of the last-write operation of (t', i', j') that we seek. Note that the proper values for the remaining variables exist given (t', i', j') and (t, i, j).

In fact, $\mathcal{Q}_{\text{stencil}}$ is not a single polyhedron because of lexicographic order, but a union of three polyhedra. The lexicographic order $(t, i, j) \prec (t', i', j')$ is decomposed to three cases: $t < t'$, $t = t' \wedge i < i'$ and $t = t' \wedge i = i' \wedge j < j'$. So, for each polyhedron, a different ILP problem is solved and all the individual solutions are combined to obtain the producer iteration that last wrote the value. In the presence of multiple candidate producer statements these solutions are further combined to obtain the producer statement and its iteration. For a detailed description of such steps, refer to Feautrier's work [Fea91].

With this understanding, it is straightforward to see that when the iteration space of a statement is the image of a union of polyhedra, we can solve the ILP problem once for each pair of consumer and producer polyhedron in the unions and subsequently combine the results to obtain the required producer iteration. Note that the number of calls to PIP are a function of the number of the disjunctions in the Presburger set, not that of the exponential number of elements in the union of $\mathcal{Z}$-polyhedra.

Our memory access functions are not always simple affine functions (refer to Example 9). To obtain the required Presburger set for complex memory access, we simplify the constraints $M_p(z) = M_C(z')$ using the techniques presented previously in the construction of the iteration space of statements.

With the help of Example 9, we now illustrate this approach. Consider the set that satisfies

$i'/2 + j'\%2 = j$:

$$\{(i', j', i, j) \mid i'/2 + j'\%2 = j\}$$

We introduce two existential variables β_1 and β_2 to remove the modular the integer division operations to obtain

$$\mathcal{M} = \{i', j', i, j, \beta_1, \beta_2 \mid 0 \leq i' - 2\beta_1 \leq 1; 0 \leq j' - 2\beta_2 \leq 1; \beta_1 + (j' - 2\beta_2) = j\}$$

Finally, the set of candidate operations can be specified by

$$\mathcal{Q}_{\text{contrived}} = \{(z', z, \alpha, \alpha', \beta) \mid (z', \alpha') \in \mathcal{P}_{\text{contrived}}; (z, \alpha) \in \mathcal{P}_{\text{contrived}}; (z', z, \beta) \in \mathcal{M}; z \prec z'\}$$

Once again, we treat this set as a polyhedra parameterized by z' and size parameters and find the lexicographic maximum using the PIP solver to obtain the producer for the iteration at z'.

5.4.5 Iteration Space as a Union of $\mathcal{Z}$-polyhedra

Thus far, we have expressed the iteration space of statements as Presburger sets and presented in detail how to construct these sets. However, the existential quantification in Presburger equations makes it difficult to obtain the precise set of iterations. For this, we express Presburger sets as a union of $\mathcal{Z}$-polyhedra and specialize the last-write function to each individual $\mathcal{Z}$-polyhedron. Let $\mathcal{ZP} = \{Ly + l | y \in \mathcal{P}\}$ be a $\mathcal{Z}$-polyhedron in the union and $f(z')$ is the last-write function. The last-write function in terms of the coordinates of the $\mathcal{Z}$-polyhedron is then simply $f(Ly + l)$.

5.5 Conclusions

We presented a compiler analysis for performing exact value-based dependence analysis for a richer class of loop programs. Our analysis may be viewed as (i) an initial pre-processing step to transform the input specification so that all existentially quantified formulae are replaced by polyhedral sets of higher dimensions; (ii) formulation of the precedence constraints as a parametric integer linear program that can be solved by well known tools such as PIP [Fea88]; and (iii) a final post-processing step that converts the results of this into system of affine recurrence equations defined over $\mathcal{Z}$-polyhedral domains. With these pre- and post-processing steps, we are able to simply reuse existing tools that have been developed for the limited case of affine control loops.

Chapter 6

$\mathcal{Z}$-Polyhedral Model: Scheduling

Value-based dependence analysis provides us with the ordering information between computations (at different loop iterations) in a specification. Scheduling involves assigning a time stamp to these computations for parallel execution.

We will first present the *reduced dependence graph* (RDG), a compact representation of all dependences in a parameterized specification, in the $\mathcal{Z}$-polyhedral model. The RDG is a crucial data structure to devise scheduling algorithms. Then, we derive precedence constraints and then formulate an ILP to obtain valid (multidimensional) schedules with minimum latency. The resultant schedule can be used to construct a space-time transformation to obtain an equivalent program in the $\mathcal{Z}$-polyhedral model.

This is joint work with DaeGon Kim.

6.1 Relevant Mathematical Background

Before proceeding to the technical contributions of this chapter, let us recall some background that will be needed for this chapter.

An Integer Polyhedra is a subset of $\mathbb{Z}^n$ the elements of which satisfy a finite number of affine inequalities with integer coefficients. A parameterized integer polyhedron is an integer polyhedron where some indices are interpreted as size parameters.

An Affine Lattices generated by a constant matrix L and the constant offset vector l, is the set of all vectors obtained by adding the constant l to integer linear combinations of the columns of L. An affine lattice is a subset of $\mathbb{Z}^n$ and can be represented as $\{Lz + l | z \in \mathbb{Z}^m\}$ where L and l are an $n \times m$ matrix and n-vector respectively. We call z the coordinates of the affine lattice.

A $\mathcal{Z}$-Polyhedron is the intersection of an integer polyhedron and an affine lattice. We use the following representation of $\mathcal{Z}$-polyhedra.

$$\{Lz + l | Qz + q \geq 0, z \in \mathbb{Z}^m\}$$

where the columns of L constitute a basis of the affine lattice $\{Lz + l | z \in \mathbb{Z}^m\}$ and valid values of z are given by the *coordinate polyhedron* $\{z | Qz + q \geq 0, z \in \mathbb{Z}^m\}$ of the $\mathcal{Z}$-polyhedron. Iteration points of the $\mathcal{Z}$-polyhedral domain are points of the affine lattice corresponding to valid coordinates.

A parameterized $\mathcal{Z}$-polyhedron is a $\mathcal{Z}$-polyhedron where some rows of its corresponding affine lattice are interpreted as size parameters. We can always express a $\mathcal{Z}$-polyhedron such that its size parameters have a one-to-one correspondence with certain indices of its coordinate polyhedron.

Input Specification The input for our scheduling analysis is a finite list of equations of the form

$$V = \begin{cases} \cdots & \cdots \\ \mathcal{D}_{V,i} : & \texttt{op}(\ldots, U.(Lz + l \to Rz + r), \ldots) \\ \cdots & \cdots \end{cases} \tag{6.1}$$

where V and U are variables declared over the $\mathcal{Z}$-polyhedral domains $\mathcal{D}_V$ and $\mathcal{D}_U$ respectively, $\mathcal{D}_{V,i}$ is the $\mathcal{Z}$-polyhedral domain of the corresponding branch of the equation (and not just the restriction domain) and `op` is an arbitrary, atomic, iteration-wise, single-valued function that takes a single time-step to evaluate. The affine lattice function $(Lz+l \to Rz+r)$ is a dependence such that the value of the (sub)expression, $U.(Lz+l \to Rz+r)$ at $Lz+l$ equals the value of U at $Rz + r$. The domains of different branches of an equation are disjoint and satisfy $\biguplus \mathcal{D}_{V,i} = \mathcal{D}_V$ to ensure that any variable is not under- or over-defined. Variables that are not defined by an equation are treated as input. These equations can be obtained from more general specifications of the equational language in Chapter 2 through the normalization transformation based on closure properties of $\mathcal{Z}$-polyhedral domains.

Parameterized equational specifications in the $\mathcal{Z}$-polyhedral model are based on parameterized $\mathcal{Z}$-polyhedra. All dependences map consumers to producers within the same program instance since every program instance in a parameterized specification is independent.

6.2 The Scheduling Problem

Here, we will present the precedence imposed by dependences and then formulate an integer linear program to obtain valid schedules. First, however, we will extend the concept of the reduced dependence graph from the polyhedral model to the $\mathcal{Z}$-polyhedral model.

6.2.1 Basic Reduced Dependency Graph

A directed multi-graph called the *reduced dependence graph*, RDG, precisely describes the dependences between iterations of variables. It is defined as follows

- For every variable in the specification, there is a vertex in the RDG labeled by its name and annotated by its domain. We will refer to vertices and variables interchangeably.
- For every dependence of the variable V on U, there is an edge from V to U. It is annotated by the corresponding dependence function. We will refer to edges and dependences interchangeably.

At a finer granularity, each branch of the case expression independently determines dependences between computations. In order to provide greater flexibility in our search for valid schedules, we will impose causality constraints independently for each branch of the case expression. Furthermore, we will express dependences individually for each $\mathcal{Z}$-polyhedra in the $\mathcal{Z}$-polyhedral domain of a branch. To enable these, we will replace every variable by a set of new variables as elaborated below.

In Equation (6.1), let $\mathcal{D}_{V,i}$ be written as a disjoint union of $\mathcal{Z}$-polyhedra given by $\biguplus_j \mathcal{Z}_j$. The variable V in the domain $\mathcal{Z}_j$ is replaced by a new variable, say X_j. Similarly, let U be replaced by new variables given as Y_k. The dependence of V in $\mathcal{D}_{V,i}$ on U is replaced by dependences from all X_j on all Y_k. An edge from X_j to Y_k may be omitted if there are no iterations in X_j that map to Y_k (i.e., if the preimage of Y_k by the dependence function does not intersect with X_j).

A construction following these rules results in the *basic reduced dependence graph*. For an example, Figure 6.1a gives the basic RDG for the following equation.

$$A = \begin{cases} \{2i | 1 \leq 2i \leq n\} & A.(2i \to i) \\ \{2i-1 | 1 \leq 2i-1 \leq n\} & 0.(i \to) \end{cases} \tag{6.2}$$

Let the two branches of the expression be denoted by X and Y respectively. Next, we will study a refinement on this RDG.

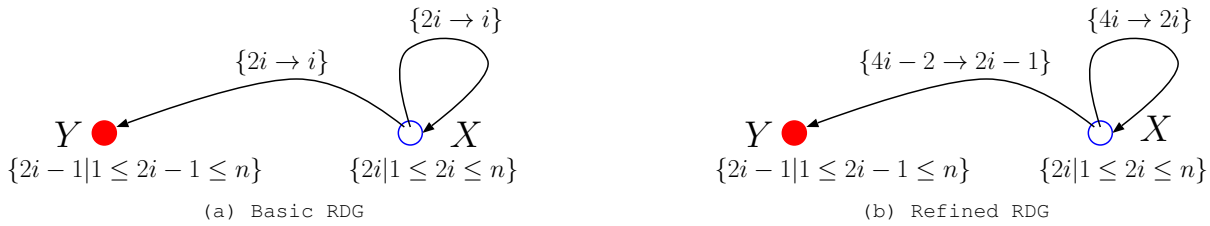


Figure 6.1: Basic and Refined Reduced Dependence Graphs for Equation 6.2.

6.2.2 Refined Reduced Dependence Graph

In the RDG for the generic specification given in Equation (6.1), let X be a variable derived from V and defined[1] on $\mathcal{Z}_X \in \mathcal{D}_{V,i}$ and let Y be a variable derived from U defined on $\mathcal{Z}_Y \in \mathcal{D}_U$ where $\mathcal{Z}_X$ and $\mathcal{Z}_Y$ are given as follows

$$\begin{aligned} \mathcal{Z}_X &= \{L_X z_X + l_X | z_X \in \mathcal{P}_X^c\} \\ \mathcal{Z}_Y &= \{L_Y z_Y + l_Y | z_Y \in \mathcal{P}_Y^c\} \end{aligned}$$

A dependence of the form $(Lz + l \to Rz + r)$ is directed form X to Y. X at $(Lz + l) \in \mathcal{Z}_X$ cannot be evaluated before Y at $(Rz + r) \in \mathcal{Z}_Y$. The affine lattice $\{Lz + l | z \in \mathbb{Z}^n\}$ may contain points that do not lie in the affine lattice $\{L_X z_X + l_X | z_X \in \mathbb{Z}^{n_X}\}$. Similarly, the affine lattice $\{Rz + r | z \in \mathbb{Z}^n\}$ may contain points that do not lie in the affine lattice $\{L_Y z_Y + l_Y | z_Y \in \mathbb{Z}^{n_Y}\}$. As a result, the dependence may be specified on a finer lattice than necessary and may safely be scaled to get a dependence of the form $(L'z' + l' \to R'z' + r')$ where

$$\begin{aligned} L' = L_X S, l' = L_X s + l_X \\ R' = L_Y S', r' = L_Y s' + l_Y \end{aligned} \tag{6.3}$$

where S and S' are matrices and s and s' are integer vectors. The refined RDG is a refinement of the basic RDG where every dependence has been replaced by a dependence satisfying Equation (6.3). Figure 6.1b gives the refined RDG for Equation 6.2.

6.2.3 Causality Constraints

Dependences between the different iterations of variables impose an ordering on their evaluation. A valid schedule of the evaluation of these iterations is the assignment of an execution time to

[1] $\mathcal{D}_{V,i}$ and $\mathcal{D}_U$ are interpreted as sets containing multiple $\mathcal{Z}$-polyhedra (that constitute the union of $\mathcal{Z}$-polyhedra in the $\mathcal{Z}$-polyhedral domain).

each computation so that precedence (causality) constraints are satisfied.

Let X and Y be two variables in the refined RDG defined on $\{L_X z_X + l_X | z_X \in \mathcal{P}_X^c\}$ and $\{L_Y z_Y + l_Y | z_Y \in \mathcal{P}_Y^c\}$ respectively. We seek to find schedules on X and Y of the following form

$$\begin{aligned} \lambda'_X(z_X) &= (L_X z_X + l_X \rightarrow \lambda_X(z_X)) \\ \lambda'_Y(z_Y) &= (L_Y z_Y + l_Y \rightarrow \lambda_Y(z_Y)) \end{aligned} \tag{6.4}$$

where λ_X and λ_Y are affine functions on z_X and z_Y respectively. Our motivation for such schedules is that if we seek schedules of the form $\lambda'(z')$ where λ' is an affine function and z' is an iteration in the domain of a variable, then we may potentially assign execution times to "holes" or computations that do not exist. In general, we seek multidimensional schedules in the form of affine functions on coordinate indices.

We will now formulate causality constraints using the refined RDG. Consider dependences from X to Y. All such dependences can be written as

$$(L_X(Sz + s) + l_X \rightarrow L_Y(S'z + s') + l_Y)$$

where S and S' are matrices and s and s' are vectors. The execution time for Y at $L_Y(S'z+s')+l_Y$ should precede the execution time for X at $L_X(Sz + s) + l_X$. With the nature of the schedules presented in Equation (6.4), our causality constraint becomes

$$\lambda_X(Sz + s) - \lambda_Y(S'z + s') \geq 1 \tag{6.5}$$

6.2.4 ILP Formulation

Here, we extract ILP (integer linear programming) constraints from causality constraints, as well as non-negativity of schedule. Then, we complete the ILP formulation with an objective function. Since our aim is using the PIP (Parameter Integer Programming) solver [Fea88], the ILP formulation will be in suitable form for the solver. We also show how a multidimensional schedule can be obtained in this model.

First, consider the non-negativity of schedule and therefore of the affine function λ. For each variable X, we want to impose the following condition:

$$\forall z \in \mathcal{P}_X^c, \lambda_X(z) \geq 0$$

By affine form of Farkas Lemma, this condition holds when $\lambda_X(z)$ is a non-negative affine

combination of the constraints $\mathcal{C}$ of $\mathcal{P}^c_X$, i.e.

$$\lambda_X(z) \equiv \lambda_{X,0} + \sum_{k=1}^{b_X} \lambda_{X,k}(a^T_{X,k} z + \alpha_{X,k})$$

where $\lambda_{X,i} \geq 0$ for all $i = 0, \ldots, b_X$ and $a^T_{X,k} z + \alpha_{X,k} \geq 0$ is the k-th constraint of $\mathcal{C}$. From now on, this is a prototype of affine schedule functions.

Now, consider the causality constraint presented in Equation (6.5) for the dependence from X to Y.

$$\begin{aligned} \lambda_{X,0} + \left(\sum_{k=1}^{b_X} \lambda_{X,k}(a^T_{X,k}(Sz+s) + \alpha_{X,k}) \right) - \lambda_{Y,0} \\ - \left(\sum_{k=1}^{b_Y} \lambda_{Y,k}(a^T_{Y,k}(S'z+s') + \alpha_{Y,k}) \right) - 1 \quad \geq \quad 0 \end{aligned}$$

where $Sz + s \in \mathcal{P}^c_X$ and $S'z + s' \in \mathcal{P}^c_Y$. Equivalently, we may say $z \in \mathcal{P}'_X \cap \mathcal{P}'_Y$ where $\mathcal{P}'_X$ and $\mathcal{P}'_Y$ are the preimage of $\mathcal{P}^c_X$ by $(z \to Sz + s)$ and $\mathcal{P}^c_Y$ by $(z \to S'z + s')$ respectively.

Now, first we will illustrate how to formulate the ILP constraints with the help of an example. Then, we will introduce an objective function for minimizing latency.

Consider Equation 6.2 and its refined RDG given in Figure 6.1b. By the non-negativity constraints, the affine functions λ_X and λ_Y will be of the form

$$\lambda_X = \lambda_{X0} + \lambda_{X1}(2i-1) + \lambda_{X2}(N-2i)$$

$$\lambda_Y = \lambda_{Y0} + \lambda_{Y1}(2i-2) + \lambda_{Y2}(N-2i+1)$$

For the edge $(4i-2 \to 2i-1)$ from X to Y, the causality constraint is

$$\begin{aligned} \lambda_{X0} + \lambda_{X1}(4i-3) + \lambda_{X2}(N-4i+2) \\ -\lambda_{Y0} - \lambda_{Y1}(2i-1) - \lambda_{Y2}(N-2i-1) \quad \geq \quad 1 \end{aligned} \tag{6.6}$$

Similarly, an ILP constraint for the edge $(4i \to 2i)$ translates into the following condition

$$\begin{aligned} \lambda_{X0} + \lambda_{X1} 2i + \lambda_{X2}(N-4i) \\ -\lambda_{X0} - \lambda_{X1} i - \lambda_{X2}(N-2i) \quad \geq \quad 1 \end{aligned} \tag{6.7}$$

where all the unknowns are non-negative. Since Y is assigned to a constant, one possibility is $\lambda_{Y0} = \lambda_{Y1} = \lambda_{Y2} = 0$. In this case, Equation (6.6) can be simplified to the following form

$$\begin{aligned} (\lambda_{X0} - \lambda_{X1} + \lambda_{X2} - 1) + (2\lambda_{X1} - \lambda_{X2})(2i-1) \\ +(\lambda_{X2})(N-2i) \quad \geq \quad 0 \end{aligned}$$

Similarly, Equation (6.7) also simplifies to

$$(\lambda_{X1} - \lambda_{X2} - 1) + (\lambda_{X1} - \lambda_{X2})(2i - 1) \geq 0$$

Now, we have the following inequalities

$$\begin{array}{rcl} \lambda_{X0} - \lambda_{X1} + \lambda_{X2} - 1 & \geq & 0 \\ 2\lambda_{X1} - \lambda_{X2} & \geq & 0 \\ \lambda_{X0} & \geq & 0 \\ \lambda_{X1} - \lambda_{X2} - 1 & \geq & 0 \\ \lambda_{X1} - \lambda_{X2} & \geq & 0 \end{array} \tag{6.8}$$

All these steps for deriving ILP constraints can be performed automatically and these inequalities can be systematically solved by standard ILP solvers.

Now, we present an objective function when all domains are bounded. In this case, there exists an affine expression $\mathcal{L}$ of the program parameters such that $\mathcal{L} - \lambda_X(z) \geq 0$ for all $z \in \mathcal{P}_X^c$ if there exists an affine function λ_X. So, it does not restrict the space of valid affine function λ_X. We want to minimize the total latency by minimizing $\mathcal{L}$. In order to use the PIP solver, for each variable X we first add $\mathcal{L} - \lambda_X(z) \geq 0$ to the constraints, and the unknown variables in $\mathcal{L}$ are placed into the outermost dimensions because PIP solver gives the lexicographic minimum of a given parameterized polyhedron.

Let us continue studying Equation 6.2. For the minimum latency affine schedule, we add the following constraints from $\mathcal{L}(= \mu_0 N + \mu_1) - \lambda_X(z)$. Here, $(\mu_0 N + \mu_1)$ is the prototype of latency and μ_0 and μ_1 are unknown constants. Solving, we get

$$\begin{array}{rcl} \mu_0 N + \mu_1 - \lambda_{X0} - \lambda_{X1}(2i - 1) - \lambda_{X1}(N - 2i) & \geq & 0 \\ \text{or } (\mu_0 - \lambda_{X0})(N - 2i) + (\mu_0 - \lambda_{X1})(2i - 1) & & \\ +(\mu_0 + \mu_1 - \lambda_{X0}) & \geq & 0 \end{array}$$

which implies the following

$$\begin{array}{rcl} \mu_0 - \lambda_{X0} & \geq & 0 \\ \mu_0 - \lambda_{X1} & \geq & 0 \\ \mu_0 + \mu_1 - \lambda_{X0} & \geq & 0 \end{array} \tag{6.9}$$

Note that for the sake of simplicity, we do not consider the variable Y that is constant on all its iterations. In fact, Equations (6.8) and (6.9) define a polyhedron. Since we put μ_0 into the first dimension index, the lexicographic minimum of this polyhedron will provide minimum latency schedules for each variable. Finally, we will get the following schedules for X and Y

$$\begin{array}{rcl} \lambda'_X & : & (2i \rightarrow i) \\ \lambda'_Y & : & (2i + 1 \rightarrow 0) \end{array}$$

Now, consider multidimensional schedules in this model. The basic idea is the same as that in Polyhedral model. Thus, rather than presenting technical details, we just provide a precise and intuitive (maybe inefficient) formulation for multidimensional schedules. The basic idea of the following ILP formulation is satisfying as many dependences as possible and the theory justifying this formulation is given by Feautrier [Fea92b].

$$\begin{gathered}\max \sum_{e \in E} x_e \\ \text{subject to } 0 \leq x_e \leq 1 \\ \lambda_X(Sz + s) - \lambda_Y(S'z + s') - 1 \geq x_e \\ \lambda_X(z) \geq 0\end{gathered}$$

A new variable x_e for each $e \in E$ is introduced. When $x_e = 1$, the dependence associated to edge e will be satisfied. The objective function maximizes the number of the edges that are satisfied. The multidimensional time can be obtained in this model by the same way in the polyhedral model.

The purpose of this section was to present the transformation of causality constraints into ILP constraints precisely and the illustration of this step with the help of an example. For technical details on Farkas scheduling algorithm, please refer to [Fea92a] by Feautrier.

6.3 Conclusions

We presented the scheduling analysis for the $\mathcal{Z}$-polyhedral model, a strict generalization of the polyhedral model, to obtain multidimensional schedules with minimum latency. An important property of our ILP formulation is that it searches for valid schedules only in the space of functions that can subsequently be used to construct a space-time transformation of the input loop program. This had been a major limitation of previous approaches. Similar to the value-based dependence analysis presented in Chapter 5, we are able to reuse existing tools that have been developed for the polyhedral model.

Chapter 7

Related Work

We will describe related work in two parts. First we will elaborate on work related to simplifying reductions. Then we will summarize related work for our contributions on foundations and transformations in the $\mathcal{Z}$-polyhedral model.

7.1 Simplifying Reductions

Our work on simplifying reductions is a generalization of a well known transformation called incrementalization. Incrementalization has been studied in many different contexts, including semi-automatic program derivation [Bir84], finite differencing in domain specific languages like SETL [Fon79, PK82] and INC [YS91], and of course strength reduction which is probably the simplest form of incrementalization [CK77], and its generalization [KMZ98].

Rajopadhye [Raj] presents some early work on detecting generalized scans in equational programs in the polyhedral model. He gives a few sufficient conditions and identifies reuse but does not give a systematic procedure for exploiting the reuse and does not consider properties of domains.

Liu et al. [LSLR05] have described how incrementalization can be used to optimize polyhedral loop computations involving aggregates in arrays *i.e.*, reductions. Their techniques sometimes improve asymptotic complexity but constitute a special case of our methods. In particular, they seek to exploit reuse only along the indices of the accumulation loops (in our notation, r_X is restricted to being a canonic vector) and would not be able to simplify an equation like $Y_{i,j} = \sum_k X_{i-j,k}$. They also require the existence of an inverse operator and do not handle the case when there is reuse of values that contribute to the same answer. Neither do they consider reduction decompositions and algebraic and expression transformations. Furthermore, they do

not have a cost model and hence cannot claim optimality. Some of their techniques such as memory/cache optimization are complementary to our work and may be useful for post processing as can techniques to solve similar problems in the polyhedral model [Fea92b, QRW00, QR00].

7.2 $\mathcal{Z}$-Polyhedral Model

Building on the work on $\mathcal{Z}$-polyhedra by Le Verge (elaborated in section 4.1), Quinton et. al. [QRR96, QRR97] were the first to propose the extension to a language based on unions of $\mathcal{Z}$-polyhedra . However, as a consequence of their representation and interpretation, they did not have a unique canonic representation. Also they could not establish the equivalence between identical $\mathcal{Z}$-polyhedra nor did they provide the difference of two $\mathcal{Z}$-polyhedra. Other consequences included complex semantics for change of basis and function composition. In many ways, our work is a logical completion of their efforts initiated a decade ago.

Ramanujam [Ram95] describes algorithms to generate code, both sequential and parallel, after applying non-unimodular transformations to nested loop programs. His work is restricted to a single, perfectly nested loop nest, and the same transformation is applied to all the statements in the loop body. The code generation problem thus reduced to scanning the image, by a non-unimodular function, of a single polyhedron. Other work [LP94, Xue94] has also been restricted to a single loop nest with the same non-unimodular, non-singular transformation applied to all statements.

Rajopadhye and Lenders [LR94] propose a technique for designing multi-rate VLSI arrays, which are regular arrays of processing elements, but where different registers are clocked at different rates. This leads to very efficient hardware structures. The mathematical formalism is based on using systems of recurrence equations (i.e., equational programs) defined over $\mathcal{Z}$-polyhedral domains, which are viewed as the images of polyhedra by non-singular affine transformations. Although the focus of the paper is on synthesis methods, notably scheduling and localization, the authors discuss the "legality" of the proposed specification, in terms of checking whether a variable is actually defined at all points in the domain where it is declared. This requires determining whether the values of other variables specified on the right hand side of the equation are defined at precisely those points, which requires the closure properties we describe here. Rajopadhye and Lenders provide sufficient conditions, not a complete solution.

Recently, Seghir et al. [SL06] showed independently that the affine image of an integer polyhedron can be written as a union of $\mathcal{Z}$-polyhedra. They used this result to count the number

of solutions to a Presburger formula in order to find the memory footprint of affine computations. However, they do not study the language- and model- theoretic aspects of this result.

7.2.1 Value Based Dependence Analysis in the $\mathcal{Z}$-Polyhedral Model

There has been extensive research on the dependence analysis of loop programs [Ban88, Pug92, DRV00, Fea91]. Our work on value based dependence analysis extends these techniques to more general programs in the $\mathcal{Z}$-polyhedral model. On the other hand, the decidable theory of Presburger formulae has been known for many decades and has also been incorporated in the Omega tool [Pug92]. Additionally, the Omega tool has been used extensively in optimizing compilers. However, in spite of the generality of the underlying machinery, the Omega test is still restricted to ACLs, and thus, similar in scope to previous techniques.

7.2.2 Scheduling in the $\mathcal{Z}$-Polyhedral Model

The scheduling problem is one of the most widely studied problems in the polyhedral model. For a detailed exposition of the various scheduling techniques, we refer the interested reader to the text by Darte et al. [DRV00].

The scheduling problem on recurrence equations with uniform (constant-sized) dependences was originally presented by Karp et. al. [KMW67]. A similar problem was posed by Lamport [Lam74] for programs with uniform dependences. Quinton showed how these techniques can be used for systolic synthesis [Qui84, Qui87]. Shang and Fortes [SF91] and Lisper [Lis90] presented optimal linear schedules for uniform dependence algorithms. Rao [Rao85] first presented affine by variable schedules for uniform dependences and also proposed an improvement of the technique by Karp et al. [KMW67] for multidimensional schedules. The first result of scheduling programs with affine dependences was solved by Rajopadhye and Fujimoto [RPF86], and independently by Quinton and Van Dongen [QV89]. These results were generalized to variable dependent schedules by Mauras et. al. [MQRS90]. Feautrier presented the optimal solution to the affine scheduling problem (by variable) [Fea92a]. Feautrier also provided the extension to multidimensional time [Fea92b].

Darte and Robert [DR95] (as well as Feautrier [Fea92a] and Quinton [Qui87]) formulate the schedule as the floor of a rational affine function, thus enabling LP rather than ILP to be used for efficient solutions. However, given a rational schedule, it is not known how to produce a space-time mapping and subsequently generate code. The schedules that we construct from the

techniques presented in Chapter 6 can be used to perform appropriate program transformations to obtain an equivalent specification that can be input directly to the final code generation step, which has already been solved by Bastoul [Bas04]. More importantly, we have extended the scheduling analysis to a class of programs that is strictly more general than ACLs.

Chapter 8

Conclusions

As in our discussion of related work, we will present the conclusions of our advances in the polyhedral model in two parts—first on the simplification transformation optimizing the asymptotic computational complexity of reductions in the polyhedral model. Then we will present conclusions on our work on generalization of the polyhedral formalism and techniques to specifications in the $\mathcal{Z}$-polyhedral model.

8.1 Simplifying Reductions

Using properties of polyhedra, in Chapter 3, we provided a precise characterization of the asymptotic computational complexity of high level equational programs in the polyhedral model. We presented automatic program transformation techniques that achieve simplification of the computational complexity of programs that have reductions and whose values exhibit reuse. Then, we described the mathematical foundations of the simplification and provided an algorithm that performs it optimally using complexity as the cost function.

8.2 Z-Polyhedral Model

In Chapter 4, we presented a novel representation and interpretation of $\mathcal{Z}$-polyhedra and prove the various closure properties of the family of unions of $\mathcal{Z}$-polyhedra required to extend the polyhedral model. In addition, we prove closure of domains in the $\mathcal{Z}$-polyhedral model under images by arbitrary affine functions which had been a major limitation of the polyhedral model. As a corollary, we prove that Presburger sets and unions of LBLs, widely assumed to be richer classes, are equivalent to unions of $\mathcal{Z}$-polyhedra.

In addition to our technical results, we believe that our interpretation of a $\mathcal{Z}$-polyhedron as

a set of coordinates on a lattice and functions defined between lattices and their coordinates, is the correct intuition for reasoning about $\mathcal{Z}$-polyhedral sets. Some closure results for unions of $\mathcal{Z}$-polyhedra were previously known [NR00], however, the generalizations of polyhedral techniques to the $\mathcal{Z}$-polyhedral model have so far been unsolved (with the exception of the final code generation step that does not require closure, solved by Bastoul [Bas04]). To demonstrate the benefits of our interpretation, we have also presented value based dependence analysis and scheduling analysis [GKR07] for specifications in the $\mathcal{Z}$-polyhedral model. These are direct extensions of the corresponding analyses of specifications in the polyhedral model. We must note that, our abstraction allows the reuse of previously developed tools in the polyhedral model with straightforward pre- and post-processing.

The language-theoretic implications of the $\mathcal{Z}$-polyhedral model are also very interesting. Our equational language is purely functional, and through its incorporation into a general purpose functional language, one may make decades of research in the high-performance and automatic parallelization communities available to modern functional languages.

8.2.1 Value Based Dependence Analysis in the $\mathcal{Z}$-Polyhedral Model

We presented a compiler analysis for performing exact value-based dependence analysis for a richer class of "static control loop programs." This class is significantly more general than the largest previously known class, namely *affine control loops* for which such analysis was possible [Fea91, Pug92, MAL93]. It includes loop programs whose iteration spaces are arbitrary Presburger sets and whose access functions are affine functions extended with `div` and `mod` operators. Such a class is not only more general, but it is also important, since many common applications such as red-black SOR, and Canon's algorithm for matrix multiplication, can be concisely and naturally written using the extended syntax that we provide.

Our analysis may be viewed as (i) an initial pre-processing step to transform the input specification so that all existentially quantified formulae are replaced by polyhedral sets of higher dimensions; (ii) formulation of the precedence constraints as a parametric integer linear program that can be solved by well known tools such as PIP [Fea88]; and (iii) a final post-processing step that converts the results of this into system of affine recurrence equations defined over $\mathcal{Z}$-polyhedral domains. As mentioned above, with these precisely defined pre- and post-processing steps, we are able to simply reuse existing machinery and tools that have been previously developed for the limited case of ACLS.

The output of the analysis can exploit the above representation of $\mathcal{Z}$-polyhedra [GR07] and can be analyzed for parallelism using sophisticated and more precise scheduling techniques, that allow us to detect more parallelism than is possible under the polyhedral model alone [GKR07].

8.2.2 Scheduling in the $\mathcal{Z}$-Polyhedral Model

The polyhedral model has been widely studied for the automatic parallelization of loop programs. However, it is limited in expressivity. In Chapter 6, we presented the scheduling analysis for the $\mathcal{Z}$-polyhedral model, a strict generalization of the polyhedral model, to obtain multidimensional schedules with minimum latency. Our schedules are of the form $\lambda'(z) = (Lz + l \rightarrow \lambda(z))$ where $\{Lz + l | z \in \mathbb{Z}^n\}$ is the lattice corresponding to the $\mathcal{Z}$-polyhedron and $\lambda(z)$ is the affine function (comprising of integer scalars) on the coordinates of this lattice. As a result, an important property of our ILP formulation is that it searches only in the space of functions that can subsequently be used to construct a space-time transformation of the program.

Our schedules express unbounded parallelism and thus cannot directly be realized in any parallel hardware. However, such parallelism detection is the crucial precursor to any resource constrained analysis.

REFERENCES

[ABC+06] Krste Asanovic, Ras Bodik, Bryan C. Catanzaro, Joseph J. Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William L. Plishker, John Shalf, Samuel W. Williams, and Katherine A. Yelick. The landscape of parallel computing research: a view from berkeley. Technical Report UCB/EECS-2006-183, Electrical Engineering and Computer Sciences, University of California at Berkeley, December 2006.

[Ban88] Utpal Banerjee. *Dependence Analysis for Supercomputing.* Kluwer Academic Publishers, 1988.

[Bas02] C. Bastoul. Generating loops for scanning polyhedra. Technical Report 2002/23, PRiSM, Versailles University, 2002.

[Bas04] Cédric Bastoul. Code generation in the polyhedral model is easier than you think. In *IEEE PACT*, pages 7–16, 2004.

[BCG+03] C. Bastoul, A. Cohen, A. Girbal, S. Sharma, and O. Temam. Putting polyhedral loop transformations to work. In *Languages and Compilers for Parallel Computers*, pages 209–225, Oct 2003.

[Bir84] R. S. Bird. The promotion and accumulation strategies in transformational programming. *ACM Trans. Program. Lang. Syst.*, 6(4):487–504, 1984.

[CK77] John Cocke and Ken Kennedy. An algorithm for reduction of operator strength. *Commun. ACM*, 20(11):850–856, 1977.

[CLW97] P. Clauss, V. Loechner, and D. Wilde. Deriving formulae to count solutions to parameterized linear systems using ehrhart polynomials: Applications to the analysis of nested-loop programs, 1997.

[DCD97] E. De Greef, F. Catthoor, and H. De Man. Memory size reduction through storage order optimization for embedded parallel multimedia applications. In *Parallel Processing and Multimedia*, Geneva, Switzerland, July 1997.

[DR95] Alain Darte and Yves Robert. Affine-by-statement scheduling of uniform and affine loop nests over parametric domains. *J. Parallel Distrib. Comput.*, 29(1):43–59, 1995.

[DRV00] A. Darte, Y. Robert, and F. Vivien. *Scheduling and Automatic Parallelization.* Birkhäuser, 2000.

[DSV05] A. Darte, R. Schreiber, and G. Villard. Lattice-based memory allocation. *IEEE Transactions on Computers*, 54(10):1242–1257, October 2005.

[Dup97] F. Dupont de Dinechin. *Systèmes structurés d'équations récurrentes : mise en œuvre dans le langage Alpha et applications.* Thèse de doctorat, université de Rennes I, January 1997.

[Ehr67] E. Ehrhart. Sur une problème de géométrie diophantine linéaire. *J. reine angew. Math.*, 227:1–29, 1967.

[Fea88] P. Feautrier. Parametric integer programming. *RAIRO Recherche Opérationelle*, 22(3):243–268, Sep 1988.

[Fea91] Paul Feautrier. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming*, 20(1):23–53, 1991.

[Fea92a] Paul Feautrier. Some efficient solutions to the affine scheduling problem. part I. one-dimensional time. *International Journal of Parallel Programming*, 21(5):313–348, October 1992.

[Fea92b] Paul Feautrier. Some efficient solutions to the affine scheduling problem. part II. multidimensional time. *International Journal of Parallel Programming*, 21(6):389–420, December 1992.

[FLVG95] Agustin Fernández, José M. Llabería, and Miguel Valero-García. Loop transformation using nonunimodular matrices. *IEEE Trans. Parallel Distrib. Syst.*, 6(8):832–840, 1995.

[Fon79] Amelia C. Fong. Inductively computable constructs in very high level languages. In *POPL '79: Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 21–28, New York, NY, USA, 1979. ACM Press.

[GKR07] Gautam, DaeGon Kim, and S. Rajopadhye. Scheduling in the $\mathcal{Z}$-polyhedral model. In *IPDPS'07: International Parallel and Distributed Processing Symposium*, 2007.

[GR06] Gautam and S. Rajopadhye. Simplifying reductions. In *POPL '06: Symposium on Principles of programming languages*, pages 30–41, New York, NY, USA, 2006. ACM Press.

[GR07] Gautam and S. Rajopadhye. The $\mathcal{Z}$-polyhedral model. In *PPoPP'07: Symposium on Principles and Practice of Parallel Programming*, 2007.

[GRQ02] Gautam, S. Rajopadhye, and P. Quinton. Scheduling reductions on realistic machines. In *SPAA '02: Symposium on Parallel algorithms and architectures*, pages 117–126, 2002.

[Her51] C. Hermite. Sur l'introduction des variables continues dans la theorie des nombres. *J. Reine Angew. Math.*, 41:191–216, 1851.

[HPF99] Paul Hudak, John Peterson, and Joseph Fasel. A gentle introduction to Haskell 98, 1999.

[KMW67] R. M. Karp, R. E. Miller, and S. V. Winograd. The organization of computations for uniform recurrence equations. *JACM*, 14(3):563–590, July 1967.

[KMZ98] V. Kislenkov, V. Mitrofanov, and E. Zima. Multidimensional chains of recurrences. In *ISSAC '98: Proceedings of the 1998 international symposium on Symbolic and algebraic computation*, pages 199–206, New York, NY, USA, 1998. ACM Press.

[Lam74] Leslie Lamport. The parallel execution of DO loops. *Communications of the ACM*, pages 83–93, February 1974.

[Le 92] H. Le Verge. *Un environnement de transformations de programmmes pour la synthèse d'architectures régulières*. PhD thesis, L'Université de Rennes I, IRISA, Campus de Beaulieu, Rennes, France, Oct 1992.

[Le 94] H. Le Verge. Recurrences on lattice polyhedra and their applications. April 1995 (Based on a manuscript written by H. Le Verge just before his untimely death in 1994).

[LF97] V. Lefebvre and P. Feautrier. Optimizing storage size for static control programs in automatic parallelizers. In Lengauer, Griebl, and Gorlatch, editors, *Euro-Par'97*, volume 1300 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[Lis90] B. Lisper. Linear programming methods for minimizing execution time of indexed computations. In *Int. Workshop on Compilers for Parallel Computers*, 1990.

[LLL01] Amy W. Lim, Shih-Wei Liao, and Monica S. Lam. Blocking and array contraction across arbitrarily nested loops using affine partitioning. In *PPoPP '01: Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming*, pages 103–112, New York, NY, USA, 2001. ACM Press.

[LP94] Wei Li and Keshav Pingali. A singular loop transformation framework based on non-singular matrices. *Int. J. Parallel Program.*, 22(2):183–205, 1994.

[LR94] P. Lenders and S. V. Rajopadhye. Multirate VLSI arrays and their synthesis. Technical Report 94-70-01, Oregon State University, Computer Science Dept, Corvallis OR 97331, December 1994.

[LSLR05] Yanhong A. Liu, Scott D. Stoller, Ning Li, and Tom Rothamel. Optimizing aggregate array computations in loops. *ACM Trans. Program. Lang. Syst.*, 27(1):91–125, 2005.

[LZP99] Rune B. Lyngsø, Michael Zuker, and Christian N. S. Pedersen. Fast evaluation of internal loops in rna secondary structure prediction. *Bioinformatics*, 15(6):440–445, 1999.

[Mac99] Josh MacDonald. Program analysis with presburger integer formulae, 1999.

[MAL93] D. Maydan, S. P. Amarsinghe, and M. Lam. Array data flow analysis and its use in array privatization. In *Principles of Programming Languages*, pages 2–15. ACM, January 1993.

[Mau89] C. Mauras. *ALPHA: un langage équationnel pour la conception et la programmation d'architectures parallèles synchrones*. PhD thesis, L'Université de Rennes I, Rennes, France, December 1989.

[MQRS90] C. Mauras, P. Quinton, S. Rajopadhye, and Y. Saouter. Scheduling affine parametrized recurrences by means of variable dependent timing functions. In S. Y. Kung and E. Swartzlander, editors, *International Conference on Application Specific Array Processing*, pages 100–110, Princeton, New Jersey, 1990. IEEE Computer Society.

[NR00] S.P.K Nookala and T. Risset. A library for z-polyhedral operations. Technical Report PI 1330, IRISA, Rennes, France, 2000.

[PK82] Robert Paige and Shaye Koenig. Finite differencing of computable expressions. *ACM Trans. Program. Lang. Syst.*, 4(3):402–454, 1982.

[Pug92] W. Pugh. A practical algorithm for exact array dependence analysis. *Commun. ACM*, 35(8):102–114, 1992.

[QR00] Fabien Quilleré and Sanjay Rajopadhye. Optimizing memory usage in the polyhedral model. *ACM Trans. Program. Lang. Syst.*, 22(5):773–815, 2000.

[QRR96] P. Quinton, S. Rajopadhye, and T. Risset. Extension of the alpha language to recurrences on sparse periodic domains. In *ASAP '96: International Conference on Application-Specific Systems, Architectures, and Processors*, page 391, 1996.

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

[QRW00] Fabien Quilleré, Sanjay Rajopadhye, and Doran Wilde. Generation of efficient nested loops from polyhedra. *Int. J. Parallel Program.*, 28(5):469–498, 2000.

[Qui84] Patrice Quinton. Automatic synthesis of systolic arrays from uniform recurrent equations. In *ISCA '84: Proceedings of the 11th annual international symposium on Computer architecture*, pages 208–214, 1984.

[Qui87] P. Quinton. The systematic design of systolic arrays. In F. Fogelman Soulie, Y. Robert, and M. Tchuente, editors, *Automata Networks in Computer Science*, chapter 9, pages 229–260. Princeton University Press, 1987. Preliminary versions appear as IRISA Tech Reports 193 and 216, 1983, and in the proceedings of the IEEE Symposium on Computer Architecture, 1984.

[QV89] Patrice Quinton and Vincent Van Dongen. The mapping of linear recurrence equations on regular arrays. *Journal of VLSI Signal Processing*, 1(2):95–113, 1989.

[Raj] Sanjay Rajopadhye. LACS: a language for affine communication structures. Technical Report RR-2093.

[Ram95] J. Ramanujam. Beyond unimodular transformations. *J. Supercomput.*, 9(4):365–389, 1995.

[Rao85] S. K. Rao. *Regular Iterative Algorithms and their Implementations on Processor Arrays*. PhD thesis, Stanford University, Information Systems Lab., Stanford, Ca, October 1985.

[RF93] Xavier Redon and Paul Feautrier. Detection of recurrences in sequential programs with loops. In *PARLE '93: Parallel Architectures and Languages Europe*, pages 132–145, 1993.

[RPF86] S. V. Rajopadhye, S. Purushothaman, and R. M. Fujimoto. On synthesizing systolic arrays from recurrence equations with linear dependencies. In *Proceedings, Sixth Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 488–503, New Delhi, India, December 1986. Springer Verlag, LNCS 241. Later appeared in Parallel Computing, June 1990.

[SF91] W. Shang and J. Fortes. Time optimal linear schedules for algorithms with uniform dependencies. *IEEE Transactions on Computers*, 40(6):723–742, June 1991. preliminary results reported in ASAP '88.

[SL06] Rachid Seghir and Vincent Loechner. Memory optimization by counting points in integer transformations of parametric polytopes. In *CASES'06: International Conference on Compilers, Architectures, and Synthesis for Embedded Systems*, pages 74–82, 2006.

[Smi61] H. J. S. Smith. On systems of linear indeterminate equations and congruences. *Phil. Trans. Roy. Soc. London*, 151:293–326, 1861.

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

[TVSA01a] W. Thies, F. Vivien, J. Sheldon, and S. Amarasinghe. A unified framework for schedule and storage optimization. In *Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation*, pages 232–242. ACM Press, 2001.

[TVSA01b] William Thies, Frédéric Vivien, Jeffrey Sheldon, and Saman P. Amarasinghe. A unified framework for schedule and storage optimization. In *PLDI*, pages 232–242, 2001.

[WA99] Barry Wilkinson and Michael Allen. *Parallel programming: techniques and applications using networked workstations and parallel computers*. Prentice-Hall, Inc., 1999.

[Wil93] D. Wilde. A library for doing polyhedral operations. Technical Report PI 785, IRISA, Rennes, France, Dec 1993. an extended version of the author's MS Thesis, Computer Science Dept, Oregon State University, Corvallis, OR. Dec 1993.

[Xue94] Jingling Xue. Automating non-unimodular loop transformations for massive parallelism. *Parallel Computing*, 20(5):711–728, 1994.

[YS91] Daniel M. Yellin and Robert E. Strom. Inc: a language for incremental computations. *ACM Trans. Program. Lang. Syst.*, 13(2):211–236, 1991.