Grids: A Domain for Analyzing the Distribution of Numerical Values*

Roberto Bagnara[1], Katy Dobson[2], Patricia M. Hill[2], Matthew Mundell[2], and Enea Zaffanella[1]

[1] Department of Mathematics, University of Parma, Italy,
{bagnara,zaffanella}@cs.unipr.it
[2] School of Computing, University of Leeds, UK,
{katyd,hill,mattm}@comp.leeds.ac.uk

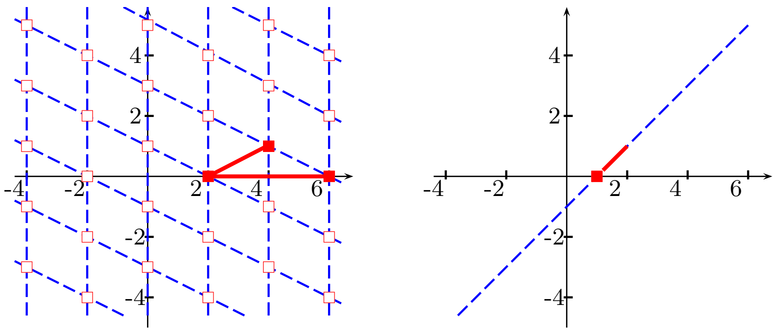
Abstract. This paper explores the abstract domain of *grids*, a domain that is able to represent sets of equally spaced points and hyperplanes over an n-dimensional vector space. Such a domain is useful for the static analysis of the patterns of distribution of the values program variables can take. We present the domain, its representation and the basic operations on grids necessary to define the abstract semantics. We show how the definition of the domain and its operations exploit well-known techniques from linear algebra as well as a dual representation that allows, among other things, for a concise and efficient implementation.

1 Introduction

We distinguish between two kinds of numerical information about the values program variables can take: outer *limits* (or bounds within which the values must lie) and the pattern of *distribution* of these values. Both kinds of information have important applications: in the field of automatic program verification, limit information is crucial to ensure that array accesses are within bounds, while distribution information is what is required to ensure that external memory accesses obey the alignment restriction imposed by the host architecture. In the field of program optimization, limit information can be used to compile out various kinds of run-time tests, whereas distribution information enables several transformations for efficient parallel execution as well as optimizations that enhance cache behavior.

Both limit and distribution information often come in a *relational* form; for instance, the outer limits or the pattern of possible values of one variable may depend on the values of one or more other variables. Domains that can capture relational information are generally much more complex than domains that do not have this capability; in exchange they usually offer significantly more precision, often important for the overall performance of the client application. Relational

* This work has been partly supported by EPSRC project EP/C520726/1 "Numerical Domains for Software Analysis," by MIUR project "AIDA — Abstract Interpretation: Design and Applications," and by a Royal Society (ESEP) award.



(a) The grid $\mathcal{L}$ (b) The grid $\mathcal{L}'$

Fig. 1. Congruence and generator systems representing two grids in $\mathbb{R}^2$

limit information can be captured, among other possibilities, by means of *polyhedral domains*, that is, domains that represent regions of some n-dimensional vector space bounded by a finite set of hyperplanes [10]. Although polyhedral domains such as the domain of convex polyhedra have been thoroughly researched and are widely used, relational domains for representing the (linear) distribution of numerical values have been less well researched. Moreover, as far as we know and at the time of writing, there is no available implementation providing all the basic operations needed by a relational abstract domain for distribution information. This is in spite of the fact that previous research has shown that a knowledge about the (discrete) distribution of numerical information, especially when combined with that of the limit information, can significantly improve the quality of the analysis results [1].

This paper closes this gap by providing a complete account of the relational domain of *grids*; a domain for capturing numerical distribution information. It includes a detailed survey of previous work in this area; gives two representations for the domain; outlines how these can be reduced and also how to convert between them; and shows how this double description directly supports methods for comparing, joining and intersecting elements of this domain. The paper also outlines affine image and preimage operations and two new widenings for grids.

Grids in a Nutshell. Figure 1 illustrates two ways of describing a grid; either by means of a finite set of congruence relations that all grid points must satisfy (given by dashed lines) or by means of a finite set of generating vectors used for constructing the grid points and lines (given by filled squares and thick lines).

The squares in Figure 1(a) illustrate a grid $\mathcal{L}$ indicating possible values of integer variables x and y resulting from executing the program fragment in Figure 2 for any value of `m`. The congruence relations $x = 0 \pmod 2$ and $x + 2y = 2 \pmod 4$ are represented by the vertical dashed lines and sloping lines, respectively. The set of congruence relations $\mathcal{C} = \bigl\{x = 0 \pmod 2, x + 2y = 2 \pmod 4\bigr\}$, called a *congruence system*, is said to *describe* $\mathcal{L}$. The filled squares mark the points $\boldsymbol{p}_1 = \left(\begin{smallmatrix}2\\0\end{smallmatrix}\right)$, $\boldsymbol{p}_2 = \left(\begin{smallmatrix}6\\0\end{smallmatrix}\right)$ and $\boldsymbol{p}_3 = \left(\begin{smallmatrix}4\\1\end{smallmatrix}\right)$ while all the squares (both filled and unfilled) mark points $\boldsymbol{v} = \pi_1\boldsymbol{p}_1 + \pi_2\boldsymbol{p}_2 + \pi_3\boldsymbol{p}_3$, where $\pi_1, \pi_2, \pi_3 \in \mathbb{Z}$

and $\pi_1 + \pi_2 + \pi_3 = 1$. The set of *points* $P = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3\}$ is said to *generate* $\mathcal{L}$. Some of these generating points can be replaced by *parameters* that give the gradient and distance between neighboring points. Specifically, by subtracting the point $\boldsymbol{p}_1$ from each of the other two generating points $\boldsymbol{p}_2$ and $\boldsymbol{p}_3$, we obtain the parameters $\boldsymbol{q}_2 = \binom{4}{0}$ and $\boldsymbol{q}_3 = \binom{2}{1}$ for $\mathcal{L}$ that are marked by the thick lines between points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ and points $\boldsymbol{p}_1$ and $\boldsymbol{p}_3$, respectively. It follows that each point $\boldsymbol{v} \in \mathcal{L}$ can be written as $\boldsymbol{v} = \boldsymbol{p}_1 + \pi_2 \boldsymbol{q}_2 + \pi_3 \boldsymbol{q}_3$ for some $\pi_2, \pi_3 \in \mathbb{Z}$.

The dashed line in Figure 1(b) illustrates the grid $\mathcal{L}'$ defining the line $x = y + 1$ and marks the vectors of values of the real variables x and y after an assignment `x := y + 1`, assuming that nothing is known about the value of y. As equalities are congruences modulo 0, the set $\mathcal{C}' = \{x - y = 1\}$ is also called a congruence system and describes $\mathcal{L}'$. Observe that the grid $\mathcal{L}'$ consists of all points that can be obtained as $\lambda \boldsymbol{\ell} + \boldsymbol{p}'$, for any $\lambda \in \mathbb{R}$, where $\boldsymbol{\ell} = \binom{1}{1}$ and $\boldsymbol{p}' = \binom{1}{0}$; the vector $\boldsymbol{\ell}$, called a *line*, defines a gradient and the vector $\boldsymbol{p}'$ is a generating point marking a position for the line (illustrated in Figure 1(b) by the thick line and the filled square, respectively).

```
x := 2; y := 0;   (P1)
for i := 1 to m   (P2)
  if ... then
    x := x + 4    (P3)
  else
    x := x + 2;
    y := y + 1    (P4)
  endif           (P5)
endfor
```

Fig. 2. Fragment based on an example in [10]

From what we have just seen, any grid can be represented both by a congruence system and by a *generator system*. The latter may consist of three components: a set of lines, a set of parameters and a set of points. For instance, the triples $\mathcal{G}_1 = \bigl(\varnothing, \varnothing, P\bigr)$ and $\mathcal{G}_2 = \bigl(\varnothing, \{\boldsymbol{q}_2, \boldsymbol{q}_3\}, \{\boldsymbol{p}_1\}\bigr)$ are both generator systems for $\mathcal{L}$ while the triple $\mathcal{G}' = \bigl(\{\boldsymbol{\ell}\}, \varnothing, \{\boldsymbol{p}'\}\bigr)$ is a generator system for $\mathcal{L}'$.

Contributions. The paper provides an account of the relational domain of *grids*, fully implemented within the Parma Polyhedra Library [2, 4]. In this section we provide the first comprehensive survey of the main research threads concerning these and similar domains. The other contributions are given below.

Minimizing representations. Assuming the grid is represented by a congruence and generator system in an n-dimensional vector space consisting of m congruences or generators, then we outline algorithms for minimizing the representation (based on the Hermite normal form algorithm [29]) that have worst-case complexity $\mathrm{O}\bigl(n^2 m\bigr)$. Note that previous proposals for minimization such as those in [14, 23] have worse complexity bounds (see below).

Converting representations. The congruence and generator representations described informally above form the two components of a double description method for the grid domain very similar to that for convex polyhedra [20]. For a double description method, conversion algorithms between the two systems are needed; we show how conversion can be implemented using any matrix inversion algorithm, inheriting the corresponding worst-case complexity. For instance, the complexity is $\mathrm{O}(n^3)$ when adopting the standard Gaussian elimination method; since matrix inversion has the same worst-case complexity as matrix multiplication, better theoretical complexity bounds apply [5]. Previous proposals for congruence to generator conversion have complexity no better than $\mathrm{O}(n^4)$ [15].

Grid operations. For static analysis, it is useful to provide all the set-theoretic lattice operations for grids (assuming the usual subset ordering) such as comparison, join and meet. We show that these operations are straightforward given the availability of the appropriate representation(s) in minimal form; and hence show that some have complexities strictly better than that of previous proposals [14]. We also describe a grid difference operator which is new to this paper.

Affine transformation operators. Affine image and preimage operators can be used to capture the effect of assignment statements in a program when the expression is linear although, as noted by Müller-Olm and Seidl in [21], analyses that use affine spaces for approximating the semantics of procedures are not sufficiently precise to detect all valid affine relations for programs with procedures. Here we specify, for the domain of grids, the affine image and preimage operators for a *single update* where only one dimension is modified.

Widenings. It was observed by Granger [15], that, if the grid generators can be in the rationals, then the grid domain does not satisfy the ascending chain condition; so, to guarantee termination of the analysis, a widening operation is required. In [15, Proposition 10], a widening is given for non-relational grids that returns a line parallel to an axis whenever the modulus for that dimension changes. It is then proposed that a generalized form of this could be used as a widening for relational grids; however, exactly how this is to be done is unclear. In this paper, we define two possible generalizations which come with simple syntactic checks that have efficient implementations.

Related Work. In [12], Granger shows how a static analysis can usefully employ a simple *non-relational* grid domain (that is a grid described by congruences of the form $x = c \pmod{f}$ where c and f are integers) and that this domain can obtain more precise information for applications such as automatic vectorization. Larsen et al. [17] also developed a static analyzer over a non-relational grid domain specifically designed to detect when dynamic memory addresses are congruent with respect to a given modulus; they show that, this information helps in the construction of a comprehensive set of program transformations for saving energy on low-power architectures and improving performance on multimedia processors. We note that these applications should carry over to the more complex domain considered here. In addition, Miné has shown how to construct, from the non-relational congruence domain in [12], a zone-congruence domain (that is, a domain that only allows *weakly relational* congruences that have the form $x - y = a \pmod{b}$ where a and b are rationals) [19].

Concerning *fully relational* domains, note that the use of a domain of linear *equality* relations for program analysis had already been studied by Karr [16].

In [14], Granger generalized this to provide a domain of linear *congruence* relations on an integral domain, i.e., a domain generated by integral vectors in n-dimensions; and then, in [13, 15], generalizes the results to the full grid domain. In [13–15], domain elements are represented by congruence and generator systems similar to the ones defined here. Standard algorithms for solving linear equations are used in converting from generator to congruence systems; however, a more complex $\mathrm{O}(n^4)$ algorithm is provided for converting from congruence to generator systems. Assuming the number of generators is $n+1$, the algorithm for minimizing the generator system has complexity $\mathrm{O}(n^3 \log_2 n)$. Operators for comparing grids and computing the greatest lower and least upper bounds are also described. In particular, the join operation defined in [14] has complexity $\mathrm{O}(n^4 \log_2 n)$, since the generators of one grid are added, one at a time, to the generators of the other; after each addition the minimization algorithm is applied to compute a new linearly independent set. The grid meet operation which also minimizes the addition of one congruence at a time has complexity $\mathrm{O}(n^4)$.

The problem of how best to *apply* the grid domain in a program analyzer, has been studied by Müller-Olm and Seidl in [23] also building on the work of Karr [16]. Here, the prime focus is for the design of an *interprocedural* analysis for programs containing assignment statements and procedure calls. The algorithm has three stages: first, for each program point, a matrix M containing a (minimized) set of generators (i.e., vectors of values that hold at that point) is found; secondly, the determinant f of M is computed; thirdly, a congruence system with modulo f that satisfies all the vectors in M is determined. Stage one is similar to that proposed by Granger [14] for minimizing a set of generators. Stages two and three differ from the conversion in [14] in that the modulus f is computed separately and used to reduce the sizes of the coordinates. Note that the framework described in [23] subsumes previous works by the same authors.

Following an independent stream of research, Ancourt [1] considered the domain of *$\mathbb{Z}$-polyhedra*; that is a domain of *integral lattices* intersected with the domain of convex polyhedra (see also [24–26]). We are primarily interested here in the "integral lattices" component which may be seen as a subdomain of the domain of grids where the grid is full dimensional and all the grid points are integral vectors. The representation of these integral lattices is a special case of our generator representation where, for n dimensions, there must be exactly one point and n linearly independent parameters, all of which must be integral. There is no support for a congruence representation.

All the operations on $\mathbb{Z}$-polyhedra (and therefore the lattices) require canonic representations; hence Quinton et al. [25, 26] define a canonical form for these lattices with a method for its computation. We note that the algorithm for computing the canonic form has complexity $\mathrm{O}(n^4)$, where n is the number of dimensions of the vector space. Other operations provided are those of lattice intersection, affine image and affine preimage. As there is no congruence representation, the intersection of two lattices is computed directly from the generator representations [1]; a refined version of this method is provided in [25] which we note that, as for computing the canonic form, has complexity $\mathrm{O}(n^4)$. The operations of grid join and grid difference (as defined here) are not considered; instead the union operator takes two lattices $\mathcal{L}_1$ and $\mathcal{L}_2$ and returns the set $\{\mathcal{L}_1, \mathcal{L}_2\}$ unless one (say $\mathcal{L}_1$) is contained in the other, in which case they return the larger, $\mathcal{L}_2$. Similarly the difference operation returns a set of lattices representing the set difference $\mathcal{L}_1 \setminus \mathcal{L}_2$. The domain of integral lattices has been implemented in PolyLib [18] following the approach in [25, 26]. This means that only the generator representation is supported and some operations return *sets* of lattices while others manipulate and simplify these sets.

The *homogeneous form* of a representation given in Section 4, is required by the conversion algorithm. This form is not new to this paper; in fact several researchers have observed this. For instance, Granger [14] describes a map from a linear congruence system in n variables to a homogeneous one in $n+1$ variables; Nookala and Risset [24] explain that the PolyLib [18] adds a dimension to make the (generator) representation homogeneous; while Müller-Olm and Seidl [23] consider *extended states* where vectors have an extra 0'th component.

Plan of the Paper. Preliminary concepts and notation are given in Section 2. Section 3 introduces a grid together with its congruence and generator representations while Section 4 provides the main algorithms needed to support the double description. Section 5 introduces grid widening and the paper concludes in Section 6. A long version of the paper containing all proofs is available at `http://www.comp.leeds.ac.uk/hill/Papers/papers.html`.

2 Preliminaries

The *cardinality* of a set S is denoted by $\# S$. The set of integers is denoted by $\mathbb{Z}$, rationals by $\mathbb{Q}$ and reals by $\mathbb{R}$. The complexities will assume a unit cost for every arithmetic operation.

Matrices and Vectors. If H is a matrix in $\mathbb{R}^{n\times m}$, the *transposition* of H is denoted by $H^{\mathrm{T}} \in \mathbb{R}^{m\times n}$. A vector $\boldsymbol{v} = (v_1, \ldots, v_n) \in \mathbb{R}^n$ is also regarded as a matrix in $\mathbb{R}^{n\times 1}$. The *scalar product* of vectors $\boldsymbol{v}$ and $\boldsymbol{w} \in \mathbb{R}^n$, denoted by $\langle \boldsymbol{v}, \boldsymbol{w} \rangle$, is the real number $\boldsymbol{v}^{\mathrm{T}}\boldsymbol{w} = \sum_{i=1}^{n} v_i w_i$. The vector $\boldsymbol{e}_i \in \mathbb{R}^n$ has 1 in the i-th position and 0 in every other position. We let

$$\mathrm{piv}_<(\boldsymbol{v}) := \begin{cases} 0 & \text{if } \boldsymbol{v} = 0 \\ \max\{i \mid 1 \le i \le n, v_i \ne 0\} & \text{if } \boldsymbol{v} \ne \mathbf{0} \end{cases}$$

$$\mathrm{piv}_>(\boldsymbol{v}) := \begin{cases} n+1 & \text{if } \boldsymbol{v} = 0 \\ \min\{i \mid 1 \le i \le n, v_i \ne 0\} & \text{if } \boldsymbol{v} \ne \mathbf{0}. \end{cases}$$

We write $\boldsymbol{v} \Uparrow \boldsymbol{v}'$, if $\mathrm{piv}_<(\boldsymbol{v}) = \mathrm{piv}_<(\boldsymbol{v}') = k$ and either $k = 0$ or $v_k = v'_k$ and $\boldsymbol{v} \Downarrow \boldsymbol{v}'$, if $\mathrm{piv}_>(\boldsymbol{v}) = \mathrm{piv}_>(\boldsymbol{v}') = k$ and either $k = n+1$ or $v_k = v'_k$.

Integer Combinations. The set $S = \{\boldsymbol{v}_1, \dots, \boldsymbol{v}_k\} \subseteq \mathbb{R}^n$ is *affinely independent* if, for all $\boldsymbol{\lambda} \in \mathbb{R}^k$, $\boldsymbol{\lambda} = \boldsymbol{0}$ is the only solution of $\bigl\{\sum_{i=1}^k \lambda_i \boldsymbol{v}_i = \boldsymbol{0},\ \sum_{i=1}^k \lambda_i = 0\bigr\}$. For all $\boldsymbol{\lambda} \in \mathbb{R}^k$, the vector $\boldsymbol{v} = \sum_{j=1}^k \lambda_j \boldsymbol{v}_j$ is said to be a *linear* combination of S. This combination is *affine*, if $\sum_{j=1}^k \lambda_j = 1$; and *integral*, if $\boldsymbol{\lambda} \in \mathbb{Z}^k$. The set of all linear (resp., affine, integral, integral and affine) combinations of S is denoted by linear.hull (resp., affine.hull(S), int.hull(S), int.affine.hull(S)).

Congruences and Congruence Relations. For any $a, b, f \in \mathbb{R}$, $a \equiv_f b$ denotes the *congruence* $\exists \mu \in \mathbb{Z} \mathrel{.} a - b = \mu f$. Let $\mathbb{S} \in \{\mathbb{Q}, \mathbb{R}\}$. For each vector $\boldsymbol{a} \in \mathbb{S}^n$ and scalars $b, f \in \mathbb{S}$, the notation $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \equiv_f b$ stands for the *linear congruence relation in* $\mathbb{S}^n$ defined by the set $\bigl\{\, \boldsymbol{v} \in \mathbb{R}^n \bigm| \exists \mu \in \mathbb{Z} \mathrel{.} \langle \boldsymbol{a}, \boldsymbol{v} \rangle = b + \mu f \,\bigr\}$; when $f \neq 0$, the relation is said to be *proper*; $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \equiv_0 b$ denotes the equality $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = b$. Thus, provided $\boldsymbol{a} \neq \boldsymbol{0}$, the relation $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \equiv_f b$ defines the set of affine hyperplanes $\bigl\{\, \bigl(\langle \boldsymbol{a}, \boldsymbol{x} \rangle = b + \mu f\bigr) \bigm| \mu \in \mathbb{Z} \,\bigr\}$; when $\boldsymbol{a} = \boldsymbol{0}$, we assume that $b \neq 0$; if $b \equiv_f 0$, $\langle \boldsymbol{0}, \boldsymbol{x} \rangle \equiv_f b$ defines the universe $\mathbb{R}^n$ and the empty set, otherwise.

Any vector that satisfies $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = b + \mu f$ for some $\mu \in \mathbb{Z}$ is said to *satisfy* the relation $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \equiv_f b$. Congruence relations in $\mathbb{S}^n$, such as $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \equiv_1 b$ and $\langle 2\boldsymbol{a}, \boldsymbol{x} \rangle \equiv_2 2b$, defining the same hyperplanes are considered equivalent.

The pivot notation for vectors is extended to congruences: if $\beta = \bigl(\langle \boldsymbol{a}, \boldsymbol{x} \rangle \equiv_f a_0\bigr)$ then $\mathrm{piv}_<(\beta) := \mathrm{piv}_<(\boldsymbol{a})$; if $\gamma = \bigl(\langle \boldsymbol{c}, \boldsymbol{x} \rangle \equiv_g c_0\bigr)$ and $g\boldsymbol{a} \Uparrow f\boldsymbol{c}$, then we write $\beta \Uparrow \gamma$; so that β and γ are either both equalities or both proper congruences.

3 The Grid Domain

Here we introduce grids and their representation. Note that the use of the word 'grid' here is to avoid confusion with the meaning of 'lattice' (used previously for elements similar to a grid) in its set-theoretic context (particularly relevant when working in abstract interpretation).

Grids and the Congruence Representation. A *congruence system in* $\mathbb{Q}^n$ is a finite set of congruence relations $\mathcal{C}$ in $\mathbb{Q}^n$. As we do not distinguish between syntactically different congruences defining the same set of vectors, we can assume that all proper congruences in $\mathcal{C}$ have modulus 1.

Definition 1. *Let $\mathcal{C}$ be a congruence system in $\mathbb{R}^n$. If $\mathcal{L}$ is the set of vectors in $\mathbb{R}^n$ that satisfy all the congruences in $\mathcal{C}$, we say that $\mathcal{L}$ is a* grid described by *a congruence system $\mathcal{C}$ in $\mathbb{Q}^n$. We also say that $\mathcal{C}$* is a congruence system for $\mathcal{L}$ *and write $\mathcal{L} = \mathrm{gcon}(\mathcal{C})$. If $\mathrm{gcon}(\mathcal{C}) = \varnothing$, then we say that $\mathcal{C}$ is* inconsistent.

The grid domain $\mathbb{G}_n$ *is the set of all grids in $\mathbb{R}^n$ ordered by the set inclusion relation, so that $\varnothing$ and $\mathbb{R}^n$ are the bottom and top elements of $\mathbb{G}_n$ respectively.*

The vector space $\mathbb{R}^n$ is called the *universe* grid. In set theoretical terms, $\mathbb{G}_n$ is a *lattice* under set inclusion. Many algorithms given here will require the congruence systems not only to have minimal cardinality but also such that the coefficients of (a permutation of) the congruences can form a triangular matrix.

Definition 2. *Suppose $\mathcal{C}$ is a congruence system in $\mathbb{Q}^n$. Then we say that $\mathcal{C}$ is in* minimal form *if either $\mathcal{C} = \{\langle \mathbf{0}, \boldsymbol{x} \rangle \equiv_0 1\}$ or $\mathcal{C}$ is consistent and, for each congruence $\beta = \bigl(\langle \boldsymbol{a}, \boldsymbol{x} \rangle \equiv_f b\bigr) \in \mathcal{C}$, the following hold:*

1. *if $\mathrm{piv}_<(\beta) = k$, then $k > 0$ and $a_k > 0$;*
2. *for all $\beta' \in \mathcal{C} \setminus \{\beta\}$, $\mathrm{piv}_<(\boldsymbol{\beta}') \neq \mathrm{piv}_<(\boldsymbol{\beta})$.*

Proposition 1. *Let $\mathcal{C}$ be a congruence system in $\mathbb{Q}^n$ and $m = \# \mathcal{C}$. Then there exists an algorithm for finding a congruence system $\mathcal{C}'$ in minimal form with worst-case complexity $\mathrm{O}\bigl(n^2 m\bigr)$ such that $\mathrm{gcon}(\mathcal{C}) = \mathrm{gcon}(\mathcal{C}')$.*

Note that the algorithm mentioned in Proposition 1, is based on the Hermite normal form algorithm; details about the actual algorithm are given in the proof. Note also, that when $m < n$, the complexity of this algorithm is just $\mathrm{O}\bigl(m^2 n\bigr)$.

The Generator Representation. Let $\mathcal{L}$ be a grid in $\mathbb{G}_n$. Then

- a vector $\boldsymbol{p} \in \mathcal{L}$ is called a *point* of $\mathcal{L}$;
- a vector $\boldsymbol{q} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ is called a *parameter* of $\mathcal{L}$ if $\mathcal{L} \neq \varnothing$ and $\boldsymbol{p} + \mu \boldsymbol{q} \in \mathcal{L}$, for all points $\boldsymbol{p} \in \mathcal{L}$ and all $\mu \in \mathbb{Z}$;
- a vector $\boldsymbol{\ell} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ is called a *line* of $\mathcal{L}$ if $\mathcal{L} \neq \varnothing$ and $\boldsymbol{p} + \lambda \boldsymbol{\ell} \in \mathcal{L}$, for all points $\boldsymbol{p} \in \mathcal{L}$ and all $\lambda \in \mathbb{R}$.

If L, Q and P are finite sets of vecors in $\mathbb{R}^n$ and

$$\mathcal{L} := \mathrm{linear.hull}(L) + \mathrm{int.hull}(Q) + \mathrm{int.affine.hull}(P)$$

where the symbol '+' denotes the Minkowski's sum,[3] then $\mathcal{L} \in \mathbb{G}_n$ is a grid (see [29, Section 4.4] and also Proposition 7). The 3-tuple (L, Q, P), where L, Q and P denote sets of lines, parameters and points, respectively, is said to be a *generator system* in $\mathbb{Q}^n$ for $\mathcal{L}$ and we write $\mathcal{L} = \mathrm{ggen}\bigl((L, Q, P)\bigr)$. Note that, for any grid $\mathcal{L}$ in $\mathbb{G}_n$, there is a generator system (L, Q, P) in $\mathbb{Q}^n$ for $\mathcal{L}$ (see again [29, Section 4.4] and also Proposition 6). Note also that the grid $\mathcal{L} = \mathrm{ggen}\bigl((L, Q, P)\bigr) = \varnothing$ if and only if the set of points $P = \varnothing$. If $P \neq \varnothing$, then $\mathcal{L} = \mathrm{ggen}\bigl((L, \varnothing, Q_{\boldsymbol{p}} \cup P)\bigr)$ where, for some $\boldsymbol{p} \in P$, $Q_{\boldsymbol{p}} = \{\, \boldsymbol{p} + \boldsymbol{q} \in \mathbb{R}^n \mid \boldsymbol{q} \in Q \,\}$.

As for congruence systems, for many procedures in the implementation, it is useful if the generator systems have a minimal number of elements.

Definition 3. *Suppose $\mathcal{G} = (L, Q, P)$ is a generator system in $\mathbb{Q}^n$. Then we say that $\mathcal{G}$ is in* minimal form *if either $L = Q = P = \varnothing$ or $\# P = 1$ and, for each generator $\boldsymbol{v} \in L \cup Q$, the following hold:*

1. *if $\mathrm{piv}_>(\boldsymbol{v}) = k$, then $v_k > 0$;*
2. *for all $\boldsymbol{v}' \in (L \cup Q) \setminus \{\boldsymbol{v}\}$, $\mathrm{piv}_>(\boldsymbol{v}') \neq \mathrm{piv}_>(\boldsymbol{v})$.*

Proposition 2. *Let $\mathcal{G} = (L, Q, P)$ be a generator system in $\mathbb{Q}^n$ and $m = \# L + \# Q + \# P$. Then there exists an algorithm for finding a generator system $\mathcal{G}'$ in minimal form with worst-case complexity $\mathrm{O}\bigl(n^2 m\bigr)$ such that $\mathrm{ggen}(\mathcal{G}') = \mathrm{ggen}(\mathcal{G})$.*

As for Proposition 1, the algorithm mentioned in Proposition 2 is based on the Hermite normal form algorithm. Note also that, when $m < n$, the complexity of this algorithm is again just $\mathrm{O}\bigl(m^2 n\bigr)$.

[3] This is defined, for each $S, T \subseteq \mathbb{R}^n$, by $S + T := \{\, \boldsymbol{s} + \boldsymbol{t} \in \mathbb{R}^n \mid \boldsymbol{s} \in S, \boldsymbol{t} \in T \,\}$.

Double Description. We have shown that any grid $\mathcal{L}$ can be described by using a congruence system $\mathcal{C}$ and also generated by a generator system $\mathcal{G}$. For the same reasons as for the polyhedral domain, it is useful to represent the grid $\mathcal{L}$ by the *double description* $(\mathcal{C}, \mathcal{G})$. Just as for the double description method for convex polyhedra, in order to maintain and exploit such a view of a grid, an implementation must include algorithms for converting a representation of one kind into a representation of the other kind and for minimizing both representations. Note that having easy access to both representations is assumed in the implementation of many grid operators including those described here.

Suppose we have a double description $\bigl(\mathcal{C}, \mathcal{G}\bigr)$ of a grid $\mathcal{L} \in \mathbb{G}_n$, where both $\mathcal{C}$ and $\mathcal{G}$ are in minimal form. Then, it follows from the definition of minimal form that $\# \mathcal{C} \leq n + 1$ and $\# L + \# Q \leq n$. In fact, we have a stronger result.

Proposition 3. *Let $(\mathcal{C}, \mathcal{G})$ be a double description where both $\mathcal{C}$ and $\mathcal{G}$ are in minimal form. Letting $\mathcal{C} = \mathcal{E} \cup \mathcal{F}$, where $\mathcal{E}$ and $\mathcal{F}$ are sets of equalities and proper congruences, respectively, and $\mathcal{G} = (L, Q, P)$, then $\# \mathcal{F} = \# Q = n - \# L - \# \mathcal{E}$.*

Example 1. Consider the grids $\mathcal{L}$ and $\mathcal{L}'$ in Figure 1. The congruence systems $\mathcal{C}$ and $\mathcal{C}'$ are in minimal form and the generator systems $\mathcal{G}_2$ and $\mathcal{G}'$ are also in minimal form; however, $\mathcal{G}_1$ is not in minimal form as it contains more than one point. Furthermore, for $i = 1$, 2, the pairs $(\mathcal{C}, \mathcal{G}_i)$ are double descriptions for $\mathcal{L}$ while $(\mathcal{C}', \mathcal{G}')$ is a double description for $\mathcal{L}'$.

Comparing Grids. For any pair of grids $\mathcal{L}_1 = \operatorname{ggen}\bigl((L, Q, P)\bigr)$, $\mathcal{L}_2 = \operatorname{gcon}(\mathcal{C})$ in $\mathbb{G}_n$, we can decide whether $\mathcal{L}_1 \subseteq \mathcal{L}_2$ by checking if every generator in (L, Q, P) satisfies every congruence in $\mathcal{C}$. Note that a parameter or line $\boldsymbol{v}$ *satisfies* a congruence $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \equiv_f b$ if $\langle \boldsymbol{a}, \boldsymbol{v} \rangle \equiv_f 0$. Therefore, assuming the systems $\mathcal{C}$ and $\mathcal{G}$ are already in minimal form, the complexity of comparison is $\mathrm{O}\bigl(n^3\bigr)$.

Given that it is known that one grid is a subset of another, there are quicker tests for checking equality - the following definition is used in their specification.

Definition 4. *Let $\mathcal{C}_1, \mathcal{C}_2$ be congruence systems in minimal form. Then $\mathcal{C}_1, \mathcal{C}_2$ are said to be* pivot equivalent *if, for each $i, j \in \{1, 2\}$ where $i \neq j$, for each $\beta \in \mathcal{C}_i$, there exists $\gamma \in \mathcal{C}_j$ such that $\beta \Uparrow \gamma$.*

Let $\mathcal{G}_1 = \bigl(L_1, Q_1, \{\boldsymbol{p}_1\}\bigr)$ and $\mathcal{G}_2 = \bigl(L_2, Q_2, \{\boldsymbol{p}_2\}\bigr)$ be generator systems in minimal form. Then $\mathcal{G}_1, \mathcal{G}_2$ are said to be pivot equivalent *if, for each $i, j \in \{1, 2\}$ where $i \neq j$: for each $\boldsymbol{q}_i \in Q_i$, there exists $\boldsymbol{q}_j \in Q_j$ such that $\boldsymbol{q}_i \Downarrow \boldsymbol{q}_j$; and, for each $\boldsymbol{\ell}_i \in L_i$, there exists $\boldsymbol{\ell}_j \in L_j$ such that $\operatorname{piv}_>(\boldsymbol{\ell}_i) = \operatorname{piv}_>(\boldsymbol{\ell}_j)$.*

Proposition 4. *Let $\mathcal{L}_1 = \operatorname{gcon}(\mathcal{C}_1) = \operatorname{ggen}(\mathcal{G}_1)$ and $\mathcal{L}_2 = \operatorname{gcon}(\mathcal{C}_2) = \operatorname{ggen}(\mathcal{G}_2)$ be non-empty grids in $\mathbb{G}_n$ such that $\mathcal{L}_1 \subseteq \mathcal{L}_2$. If $\mathcal{C}_1$ and $\mathcal{C}_2$ are pivot equivalent congruence systems in minimal form or $\mathcal{G}_1$ and $\mathcal{G}_2$ are pivot equivalent generator systems in minimal form, then $\mathcal{L}_1 = \mathcal{L}_2$.*

It follows from Proposition 4, that provided $\mathcal{L}_1 \subseteq \mathcal{L}_2$ and $\mathcal{L}_1$ and $\mathcal{L}_2$ have both their generator or congruence systems already in minimal form, then the complexity of checking if $\mathcal{L}_1 = \mathcal{L}_2$ is just $\mathrm{O}\bigl(n\bigr)$. Moreover, if it is found that one pair of corresponding pivot elements of the congruence or generator systems differ, then we can immediately deduce that the grids they describe also differ.

Intersection and Grid Join. For grids $\mathcal{L}_1, \mathcal{L}_2 \in \mathbb{G}_n$, the *intersection* of $\mathcal{L}_1$ and $\mathcal{L}_2$, defined as the set intersection $\mathcal{L}_1 \cap \mathcal{L}_2$, is the largest grid included in both $\mathcal{L}_1$ and $\mathcal{L}_2$; similarly, the *grid join* of $\mathcal{L}_1$ and $\mathcal{L}_2$, denoted by $\mathcal{L}_1 \oplus \mathcal{L}_2$, is the smallest grid that includes both $\mathcal{L}_1$ and $\mathcal{L}_2$. In theoretical terms, the intersection and grid join operators are the binary *meet* and *join* operators on the lattice $\mathbb{G}_n$. They can easily be computed; if $\mathcal{L}_1 = \mathrm{gcon}(\mathcal{C}_1) = \mathrm{ggen}(\mathcal{G}_1)$ and $\mathcal{L}_2 = \mathrm{gcon}(\mathcal{C}_2) = \mathrm{ggen}(\mathcal{G}_2)$, then $\mathcal{L}_1 \cap \mathcal{L}_2 = \mathrm{gcon}(\mathcal{C}_1 \cup \mathcal{C}_2)$ and $\mathcal{L}_1 \oplus \mathcal{L}_2 = \mathrm{ggen}(\mathcal{G}_1 \cup \mathcal{G}_2)$.

In practice, the cost of computing the grid intersection and join depends on a number of factors: if generator systems $\mathcal{G}_1$ and $\mathcal{G}_2$ for $\mathcal{L}_1$ and $\mathcal{L}_2$ are known, then the complexity of computing $\mathcal{L}_1 \oplus \mathcal{L}_2$ is linear in either $\#\,\mathcal{G}_1$ or $\#\,\mathcal{G}_2$; if, however, only congruence systems $\mathcal{C}_1$ and $\mathcal{C}_2$ for $\mathcal{L}_1$ and $\mathcal{L}_2$ (not necessarily in minimal form) are known, then the complexity is that of minimizing and converting them which is, at worst, $\mathrm{O}\bigl(n^2 \max(\#\,\mathcal{C}_1, \#\,\mathcal{C}_2, n)\bigr)$. A similar argument applies to the complexities of the meet operation. However, the above operations are not directly comparable with the meet and join operations given in [14]. For such a comparison, for instance for the join operation, we assume that generator systems for $\mathcal{L}_1$ and $\mathcal{L}_2$ in minimal form are available (i.e., each with at most $n+1$ generators) and the operation returns a generator system in minimal form for $\mathcal{L}_1 \oplus \mathcal{L}_2$. Then the complexity is $\mathrm{O}\bigl(n^3\bigr)$, the complexity of minimizing a generator system with at most $2n+2$ generators, which is strictly better than $\mathrm{O}\bigl(n^4 \log_2 n\bigr)$, the complexity of the equivalent operation in [14].

Example 2. Consider the grids $\mathcal{L}_1 = \mathrm{gcon}(\mathcal{C}_1)$ and $\mathcal{L}_2 = \mathrm{gcon}(\mathcal{C}_2)$ in $\mathbb{G}_2$ where $\mathcal{C}_1 := \{x \equiv_2 0,\ -x + y \equiv_3 0\}$ and $\mathcal{C}_2 := \{x \equiv_4 0,\ -x + 2y \equiv_6 0\}$. Then the grid intersection is $\mathcal{L}_1 \cap \mathcal{L}_2 = \mathrm{gcon}(\mathcal{C}_1 \cup \mathcal{C}_2)$; thus, as $\mathcal{C} = \{x \equiv_{12} 0,\ y \equiv_3 0\}$ is a reduced form of $\mathcal{C}_1 \cup \mathcal{C}_2$, we have $\mathcal{L}_1 \cap \mathcal{L}_2 = \mathrm{gcon}(\mathcal{C})$.

Consider $\mathcal{L}_1 = \mathrm{ggen}\bigl((\varnothing, \varnothing, P_1)\bigr)$ and $\mathcal{L}_2 = \mathrm{ggen}\bigl((\varnothing, \varnothing, P_2)\bigr)$ in $\mathbb{G}_2$, where $P_1 := \left(\begin{smallmatrix} 2 & 0 & 0 \\ 2 & 3 & 0 \end{smallmatrix}\right)$ and $P_2 := \left(\begin{smallmatrix} 4 & 0 & 0 \\ 2 & 3 & 0 \end{smallmatrix}\right)$. Then the grid join $\mathcal{L}_1 \oplus \mathcal{L}_2$ is generated by $(\varnothing, \varnothing, P_1 \cup P_2)$; thus, the generator system $\mathcal{G} := \bigl(\varnothing, \left(\begin{smallmatrix} 2 & 0 \\ 0 & 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right)\bigr)$ is a minimal form of $(\varnothing, \varnothing, P_1 \cup P_2)$ and $\mathcal{L}_1 \oplus \mathcal{L}_2 = \mathrm{ggen}(\mathcal{G})$. Note that here $\mathcal{L}_1 \oplus \mathcal{L}_2 \neq \mathcal{L}_1 \cup \mathcal{L}_2$.

Grid Difference. For grids $\mathcal{L}_1, \mathcal{L}_2 \in \mathbb{G}_n$, the *grid difference* $\mathcal{L}_1 \ominus \mathcal{L}_2$ of $\mathcal{L}_1$ and $\mathcal{L}_2$ is the smallest grid containing the set-theoretic difference of $\mathcal{L}_1$ and $\mathcal{L}_2$.

Proposition 5. *The grid $\mathcal{L}_1 \ominus \mathcal{L}_2$ is returned by the algorithm in Figure 3.*

Assuming $\mathcal{C}_1$ and $\mathcal{C}_2$ are available and in minimal form, it follows from the complexities of minimization, conversion and comparison operations that the grid difference algorithm in Figure 3 has worst-case complexity $\mathrm{O}\bigl(n^4\bigr)$.

Affine Images and Preimages. Affine transformations for the vector space $\mathbb{R}^n$ will map hyperplanes to hyperplanes and preserve intersection properties between hyperplanes; such transformations can be represented by matrices in $\mathbb{R}^{n \times n}$. It follows that the set $\mathbb{G}_n$ is closed under the set of all affine transformations for $\mathbb{R}^n$. Simple and useful linear affine transformations for numerical

Input: Nonempty grids $\mathcal{L}_1 = \mathrm{gcon}(\mathcal{C}_1)$ and $\mathcal{L}_2 = \mathrm{gcon}(\mathcal{C}_2)$ in $\mathbb{G}_n$.
Output: A grid in $\mathbb{G}_n$.

$\mathcal{L}' := \varnothing$
while $\exists \beta = (e \equiv_f 0) \in \mathcal{C}_2$
  $\mathcal{C}_2 := \mathcal{C}_2 \setminus \{\beta\}$
  if $\mathcal{L}_1 \nsubseteq \mathrm{gcon}\bigl(\{\beta\}\bigr)$
    if $\mathcal{L}_1 \subseteq \mathrm{gcon}\bigl(\{2e \equiv_f 0\}\bigr)$
      $\mathcal{L}_\beta := \mathrm{gcon}\bigl(\mathcal{C}_1 \cup \{2e - f \equiv_{2f} 0\}\bigr)$
      $\mathcal{L}' := \mathcal{L}' \oplus \mathcal{L}_\beta$
    else
      return $\mathcal{L}_1$
return $\mathcal{L}'$

Fig. 3. The grid difference algorithm

domains, including the grids, are provided by the 'single update' affine image and affine preimage operators.

Given a grid $\mathcal{L} \in \mathbb{G}_n$, a variable x_k and linear expression $e = \langle \boldsymbol{a}, \boldsymbol{x} \rangle + b$ with coefficients in $\mathbb{Q}$, the *affine image operator* $\phi(\mathcal{L}, x_k, e)$ maps the grid $\mathcal{L}$ to

$$\Bigl\{ \bigl(p_1, \ldots, p_{k-1}, \langle \boldsymbol{a}, \boldsymbol{p} \rangle + b, p_{k+1}, \ldots, p_n\bigr)^{\mathrm{T}} \in \mathbb{R}^n \Bigm| \boldsymbol{p} \in \mathcal{L} \Bigr\}.$$

Conversely, the *affine preimage operator* $\phi^{-1}(\mathcal{L}, x_k, e)$ maps the grid $\mathcal{L}$ to

$$\Bigl\{ \boldsymbol{p} \in \mathbb{R}^n \Bigm| \bigl(p_1, \ldots, p_{k-1}, \langle \boldsymbol{a}, \boldsymbol{p} \rangle + b, p_{k+1}, \ldots, p_n\bigr)^{\mathrm{T}} \in \mathcal{L} \Bigr\}.$$

Observe that the affine image $\phi(\mathcal{L}, x_k, e)$ and preimage $\phi^{-1}(\mathcal{L}, x_k, e)$ are invertible if and only if the coefficient a_k in the vector $\boldsymbol{a}$ is non-zero.

Program Analysis Using Grids. We show how the grid domain can be used to find properties of the program variables not found using the polyhedra domain [10], constraint-based analysis [28] or polynomial invariants [27].

Example 3. The program fragment in Figure 2 is annotated with program points $\mathtt{P}j$, for $j = 1, \ldots, 5$. Let $\mathcal{L}_j^i \in \mathbb{G}_2$ denote the grid computed at the i-th iteration executed by the point $\mathtt{P}j$. Initially, $\mathcal{L}_j^0 = \varnothing = \mathrm{gcon}\bigl(\{1 = 0\}\bigr)$, for $j = 1, \ldots, 5$. After one and two iterations of the loop we have:

$$\begin{aligned}
\mathcal{L}_1^1 &= \mathrm{gcon}\bigl(\{x = 2,\ y = 0\}\bigr), \quad \mathcal{L}_2^1 = \mathrm{gcon}\bigl(\{x = 2,\ y = 0\}\bigr),\\
\mathcal{L}_3^1 &= \mathrm{gcon}\bigl(\{x = 6,\ y = 0\}\bigr), \quad \mathcal{L}_4^1 = \mathrm{gcon}\bigl(\{x = 4,\ y = 1\}\bigr),\\
\mathcal{L}_5^1 &= \mathrm{gcon}\bigl(\{x = 4,\ y = 1\}\bigr) \oplus \mathrm{gcon}\bigl(\{x = 6,\ y = 0\}\bigr)\\
&= \mathrm{gcon}\bigl(\{x + 2y = 6,\ x \equiv_2 0\}\bigr),\\
\mathcal{L}_2^2 &= \mathrm{gcon}\bigl(\{x = 2, y = 0\}\bigr) \oplus \mathrm{gcon}\bigl(\{x + 2y = 6, x \equiv_2 0\}\bigr)\\
&= \mathrm{gcon}\bigl(\{x + 2y \equiv_4 2, x \equiv_2 0\}\bigr).
\end{aligned}$$

Subsequent computation steps show that an invariant for P2 has already been computed since $\mathcal{L}_3^2 = \mathcal{L}_3^1$, $\mathcal{L}_4^2 = \mathcal{L}_4^1$, $\mathcal{L}_5^2 = \mathcal{L}_5^1$ so that $\mathcal{L}_2^3 = \mathcal{L}_2^2$. Thus at the end of the program, the congruences $x + 2y \equiv_4 2$ and $x \equiv_2 0$ hold.

Observe that, using convex polyhedra, a similar analysis will find instead that the inequalities $x - 2y \geq 2$, $x + 2y \geq 6$ and $y \geq 0$ hold [10].

4 Implementation

In this section, we describe convenient internal representations of the congruence and generator systems in terms of arrays (i.e., matrices) and show how matrix inversion provides a basis for converting between these representations.

Homogeneous Representations. A congruence system $\mathcal{C}$ is *homogeneous* if, for all $\bigl(\langle \boldsymbol{a}, \boldsymbol{x} \rangle \equiv_f b\bigr) \in \mathcal{C}$, we have $b = 0$. Similarly, a generator system (L, Q, P) is *homogeneous* if $\mathbf{0} \in P$. For the implementation, it is convenient to work with a homogeneous system. Thus we first convert any congruence or generator system in $\mathbb{Q}^n$ to a homogeneous system in $\mathbb{Q}^{n+1}$. The extra dimension is denoted with a 0 subscript; the vector $\hat{\boldsymbol{x}} = (x_0, \ldots, x_n)^{\mathrm{T}}$; and $\boldsymbol{e}_0$ denotes the vector $(1, \mathbf{0}^{\mathrm{T}})^{\mathrm{T}}$.

Consider the congruence system $\mathcal{C} = \mathcal{E} \cup \mathcal{F}$ in $\mathbb{Q}^n$, where $\mathcal{E}$ is a set of equalities and $\mathcal{F}$ is a set of proper congruences. Then the *homogeneous form* for $\mathcal{C}$ is the congruence system $\hat{\mathcal{C}} = \hat{\mathcal{E}} \cup \hat{\mathcal{F}}$ in $\mathbb{Q}^{n+1}$ defined by:

$$\begin{aligned}
\hat{\mathcal{E}} &:= \Bigl\{ \bigl\langle (-b, \boldsymbol{a}^{\mathrm{T}})^{\mathrm{T}}, \hat{\boldsymbol{x}} \bigr\rangle = 0 \Bigm| \bigl(\langle \boldsymbol{a}, \boldsymbol{x} \rangle = b\bigr) \in \mathcal{E} \Bigr\}, \\
\hat{\mathcal{F}} &:= \Bigl\{ \bigl\langle f^{-1}(-b, \boldsymbol{a}^{\mathrm{T}})^{\mathrm{T}}, \hat{\boldsymbol{x}} \bigr\rangle \equiv_1 0 \Bigm| \bigl(\langle \boldsymbol{a}, \boldsymbol{x} \rangle \equiv_f b\bigr) \in \mathcal{F} \Bigr\} \cup \bigl\{ \langle \boldsymbol{e}_0, \hat{\boldsymbol{x}} \rangle \equiv_1 0 \bigr\}.
\end{aligned}$$

The congruence $\langle \boldsymbol{e}_0, \hat{\boldsymbol{x}} \rangle \equiv_1 0$ expresses the fact that $1 \equiv_1 0$. By writing $\hat{\mathcal{E}} = (E^{\mathrm{T}} \boldsymbol{x} = \mathbf{0})$ and $\hat{\mathcal{F}} = (F^{\mathrm{T}} \boldsymbol{x} \equiv_1 \mathbf{0})$, where $E, F \subseteq \mathbb{Q}^{n+1}$, it can be seen that the pair (F, E), called the *matrix form* of $\hat{\mathcal{C}}$, is sufficient to determine $\mathcal{C}$.

Consider next a generator system $\mathcal{G} = (L, Q, P)$ in $\mathbb{Q}^n$. Then the *homogeneous form* for $\mathcal{G}$ is the generator system $\hat{\mathcal{G}} := \bigl(\hat{L}, \hat{Q} \cup \hat{P}, \{\mathbf{0}\}\bigr)$ in $\mathbb{Q}^{n+1}$ where

$$\hat{L} := \bigl\{ (0, \boldsymbol{\ell}^{\mathrm{T}})^{\mathrm{T}} \bigm| \boldsymbol{\ell} \in L \bigr\},\ \hat{Q} := \bigl\{ (0, \boldsymbol{q}^{\mathrm{T}})^{\mathrm{T}} \bigm| \boldsymbol{q} \in Q \bigr\},\ \hat{P} := \bigl\{ (1, \boldsymbol{p}^{\mathrm{T}})^{\mathrm{T}} \bigm| \boldsymbol{p} \in P \bigr\}.$$

The original grid $\mathcal{L} = \mathrm{gcon}(\mathcal{C})$ (resp., $\mathcal{L} = \mathrm{ggen}(\mathcal{G})$) can be recovered from the grid $\hat{\mathcal{L}} = \mathrm{gcon}(\hat{\mathcal{C}})$ (resp., $\hat{\mathcal{L}} = \mathrm{ggen}(\hat{\mathcal{G}})$) since $\mathcal{L} = \bigl\{ \boldsymbol{v} \in \mathbb{R}^n \bigm| (1, \boldsymbol{v}^{\mathrm{T}})^{\mathrm{T}} \in \hat{\mathcal{L}}) \bigr\}$. Note that, if $(\mathcal{C}, \mathcal{G})$ is a double description for a grid and $\hat{\mathcal{C}}$ and $\hat{\mathcal{G}}$ are homogeneous forms for $\mathcal{C}$ and $\mathcal{G}$, then $(\hat{\mathcal{C}}, \hat{\mathcal{G}})$ is also a double description.

Converting Representations. By considering the matrix forms of the (homogeneous forms of the) representations, we can build the conversion algorithms on top of those for matrix inversion. For an informal explanation why this is appropriate, suppose that the generator system $\mathcal{G} = \bigl(\varnothing, Q, \{\mathbf{0}\}\bigr)$ in $\mathbb{Q}^n$ is in minimal form and Q is a non-singular square matrix. Letting $\mathcal{L} = \mathrm{ggen}(\mathcal{G}) = \{ Q\boldsymbol{\pi} \mid$

$\boldsymbol{\pi} \in \mathbb{Z}^n \}$, then we also have $\mathcal{L} = \{ \boldsymbol{v} \in \mathbb{R}^n \mid Q^{-1}\boldsymbol{v} \equiv_1 0 \}$, so that $(Q^{-1}, \varnothing)$ is the matrix form of a congruence system for the same grid $\mathcal{L}$. Similarly we can use matrix inversion to convert the matrix form of a homogeneous congruence system in minimal form consisting of n proper congruences for a grid $\mathcal{L}$ to a generator system for $\mathcal{L}$. When the matrices to be inverted have less than n linearly independent columns, the algorithms first add vectors $\boldsymbol{e}_i$ where $1 \leq i \leq n$, as necessary, so as to make the matrices non-singular and hence invertible.

Proposition 6. *Let $\mathcal{C}$ be a congruence system in $\mathbb{Q}^n$ in minimal form; (F, E) the matrix form of the homogeneous form for $\mathcal{C}$; N a matrix in $\mathbb{Z}^{n+1}$ whose vectors are of the form $\boldsymbol{e}_i$, $i \in \{0, \ldots, n\}$, and such that $(N, \hat{F}, \hat{E})$ is square and nonsingular; and $(\hat{L}, \hat{Q}, M) := \bigl((N, \hat{F}, \hat{E})^{-1}\bigr)^{\mathrm{T}}$ where $\# \hat{L} = \# N$, $\# \hat{Q} = \# \hat{F}$ and $\# M = \# \hat{E}$. Then $\hat{\mathcal{G}} = \bigl(\hat{L}, \hat{Q}, \{\mathbf{0}\}\bigr)$ is the homogeneous form for a generator system $\mathcal{G}$ in minimal form and $\mathrm{ggen}(\mathcal{G}) = \mathrm{gcon}(\mathcal{C})$.*

Proposition 7. *Let $\mathcal{G}$ be a generator system in $\mathbb{Q}^n$ in minimal form; $\hat{\mathcal{G}} = \bigl(\hat{L}, \hat{Q}, \{\mathbf{0}\}\bigr)$ the homogeneous form for $\mathcal{G}$; M a matrix in $\mathbb{Z}^{n+1}$ whose vectors are of the form $\boldsymbol{e}_i$, $i \in \{0, \ldots, n\}$, and such that $(\hat{L}, \hat{Q}, M)$ is square and non-singular; and $(N, \hat{F}, \hat{E}) := \bigl((\hat{L}, \hat{Q}, M)^{-1}\bigr)^{\mathrm{T}}$ where $\# N = \# \hat{L}$, $\# \hat{F} = \# \hat{Q}$ and $\# \hat{E} = \# M$. Then $(\hat{F}, \hat{E})$ is the matrix form of the homogeneous form for a congruence system $\mathcal{C}$ in minimal form and $\mathrm{gcon}(\mathcal{C}) = \mathrm{ggen}(\mathcal{G})$.*

Both algorithms just perform matrix inversion; so their complexity depends on the inversion algorithm adopted in the implementation. As far as we know, the current best theoretical worst-case complexity is $\mathrm{O}\bigl(n^{2.376}\bigr)$ [5]. Note that, in the current implementation in the PPL, the conversion algorithm is based on the Gaussian elimination method, which has complexity $\mathrm{O}\bigl(n^3\bigr)$.

5 Grid Widening

A simple and general characterization of a widening for enforcing and accelerating convergence of an upward iteration sequence is given in [6–9]. We assume here a minor variation of this classical definition (see footnote 6 in [9, p. 275]).

Definition 5. (Widening.) *Let $\langle D, \vdash, \mathbf{0}, \oplus \rangle$ be a join-semilattice. The partial operator $\nabla \colon D \times D \rightarrowtail D$ is a* widening *if*

1. *for each $d_1, d_2 \in D$, $d_1 \vdash d_2$ implies that $d_1 \nabla d_2$ is defined and $d_2 \vdash d_1 \nabla d_2$;*
2. *for each increasing chain $d_0 \vdash d_1 \vdash \cdots$, the increasing chain defined by $d'_0 := d_0$ and $d'_{i+1} := d'_i \nabla (d'_i \oplus d_{i+1})$, for $i \in \mathbb{N}$, is not strictly increasing.*

In addition to the formal requirements in Definition 5, it is also important to have a widening that has an efficient implementation, preferably, one that depends on a simple syntactic mapping of the representations. At the same time, so that the widening is well-defined, the result of this operation should be independent of the actual representation used. For this reason, the two widenings we propose assume specific minimal forms for the congruence and generator systems.

Definition 6. *A congruence system* $\mathcal{C}$ *is in* strong minimal form *if, for each pair of distinct proper congruences,* $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \equiv_1 b$ *and* $\langle \boldsymbol{c}, \boldsymbol{x} \rangle \equiv_1 d$ *in* $\mathcal{C}$*, if* $\operatorname{piv}_<(\boldsymbol{c}) = k > 0$*, then* $-c_k < 2a_k \leq c_k$*. A generator system* $\mathcal{G} = \bigl((L, Q, P)\bigr)$ *in* $\mathbb{Q}^n$ *is in* strong minimal form *if* $\mathcal{G}$ *is in minimal form and, for each pair of distinct parameters* $\boldsymbol{u}, \boldsymbol{v} \in Q$*, if* $\operatorname{piv}_>(\boldsymbol{v}) = k \leq n$*, then* $-v_k < 2u_k \leq v_k$*.*

Proposition 8. *There exists an algorithm with complexity* $\mathrm{O}\bigl(n^3\bigr)$ *for converting a congruence system* $\mathcal{C}$ *(resp., generator system* $\mathcal{G}$*) in minimal form to a congruence system* $\mathcal{C}'$ *(resp., generator system* $\mathcal{G}'$*) in strong minimal form such that* $\operatorname{gcon}(\mathcal{C}) = \operatorname{gcon}(\mathcal{C}')$ *(resp.,* $\operatorname{ggen}(\mathcal{G}) = \operatorname{ggen}(\mathcal{G}')$*).*

The widenings defined below use either the congruence or the generator systems.

Definition 7. *Let* $\mathcal{L}_1 = \operatorname{gcon}(\mathcal{C}_1)$ *and* $\mathcal{L}_2 = \operatorname{gcon}(\mathcal{C}_2)$ *be two grids in* $\mathbb{G}_n$ *such that* $\mathcal{L}_1 \subseteq \mathcal{L}_2$*,* $\mathcal{C}_1$ *is in minimal form and* $\mathcal{C}_2$ *is in strong minimal form. Then the* grid widening $\mathcal{L}_1 \nabla_{\mathcal{C}} \mathcal{L}_2$ *is defined by*

$$\mathcal{L}_1 \nabla_{\mathcal{C}} \mathcal{L}_2 := \begin{cases} \mathcal{L}_2, & \text{if } \mathcal{L}_1 = \varnothing \text{ or } \dim(\mathcal{L}_1) < \dim(\mathcal{L}_2), \\ \operatorname{gcon}(\mathcal{C}_S), & \text{otherwise,} \end{cases}$$

where $\mathcal{C}_S := \{\, \gamma \in \mathcal{C}_2 \mid \exists \beta \in \mathcal{C}_1 \,.\, \beta \Uparrow \gamma \,\}$*.*

Definition 8. *Let* $\mathcal{L}_1 = \operatorname{ggen}(\mathcal{G}_1)$ *and* $\mathcal{L}_2 = \operatorname{ggen}(\mathcal{G}_2)$ *be two grids in* $\mathbb{G}_n$ *such that* $\mathcal{L}_1 \subseteq \mathcal{L}_2$*,* $\mathcal{G}_1 = (L_1, Q_1, P_1)$ *is in minimal form and* $\mathcal{G}_2 = (L_2, Q_2, P_2)$ *is in strong minimal form. Then the* grid widening $\mathcal{L}_1 \nabla_{\mathcal{G}} \mathcal{L}_2$ *is defined by*

$$\mathcal{L}_1 \nabla_{\mathcal{G}} \mathcal{L}_2 := \begin{cases} \mathcal{L}_2, & \text{if } \mathcal{L}_1 = \varnothing \text{ or } \dim(\mathcal{L}_1) < \dim(\mathcal{L}_2); \\ \operatorname{ggen}(\mathcal{G}_S), & \text{otherwise,} \end{cases}$$

where $\mathcal{G}_S := \bigl(L_2 \cup (Q_2 \setminus Q_S), Q_S, P_2\bigr)$ *and* $Q_S := \{\, \boldsymbol{v} \in Q_2 \mid \exists \boldsymbol{u} \in Q_1 \,.\, \boldsymbol{u} \Downarrow \boldsymbol{v} \,\}$*.*

Proposition 9. *The operators '*$\nabla_{\mathcal{C}}$*' and '*$\nabla_{\mathcal{G}}$*' are both widenings on* $\mathbb{G}_n$*.*

In Definition 7, it is required that $\mathcal{C}_2$ is in strong minimal form. The following example shows that this is necessary for the operator '$\nabla_{\mathcal{C}}$' to be well-defined.

Example 4. Let $\mathcal{L}_1 := \operatorname{gcon}(\mathcal{C}_1)$, $\mathcal{L}_2 := \operatorname{gcon}(\mathcal{C}_2)$ and $\mathcal{L}'_2 := \operatorname{gcon}(\mathcal{C}'_2)$ where $\mathcal{C}_1 = \{x \equiv_2 0,\ y \equiv_2 0\}$, $\mathcal{C}_2 = \{x \equiv_1 0,\ x + y \equiv_2 0\}$, $\mathcal{C}'_2 = \{x \equiv_1 0,\ 3x + y \equiv_2 0\}$; then $\mathcal{L}_2 = \mathcal{L}'_2$. Note that only $\mathcal{C}_1$ and $\mathcal{C}_2$ are in strong minimal form. Therefore, assuming $\mathcal{C}_S$ (resp., ${\mathcal{C}_S}'$) is defined as in Definition 7 using $\mathcal{C}_1$ and $\mathcal{C}_2$ (resp., $\mathcal{C}_1$ and $\mathcal{C}'_2$), we have $\mathcal{C}_S = \{x + y \equiv_2 0\}$ and ${\mathcal{C}_S}' = \{3x + y \equiv_2 0\}$. Thus $\mathcal{L}_1 \nabla_{\mathcal{C}} \mathcal{L}_2 = \operatorname{gcon}(\mathcal{C}_S) \neq \operatorname{gcon}({\mathcal{C}_S}')$.

Example 5. To see that the widenings depend on the variable ordering, consider the grids $\mathcal{L}_1 = \operatorname{gcon}(\mathcal{C}_1) = \operatorname{gcon}(\mathcal{C}'_1)$ and $\mathcal{L}_2 = \operatorname{gcon}(\mathcal{C}_2) = \operatorname{gcon}(\mathcal{C}'_2)$ in $\mathbb{G}_2$, where

$$\mathcal{C}_1 := \{5x + y \equiv_1 0,\ 22x \equiv_1 0\}, \quad \mathcal{C}_2 := \{5x + y \equiv_1 0,\ 44x \equiv_1 0\},$$
$$\mathcal{C}'_1 := \{9y + x \equiv_1 0,\ 22y \equiv_1 0\}, \quad \mathcal{C}'_2 := \{9y + x \equiv_1 0,\ 44y \equiv_1 0\}.$$

Assume for $\mathcal{C}_1$ and $\mathcal{C}_2$ that the variables are ordered so that x precedes y, as in the vector $(x, y)^{\mathrm{T}}$; then, $\mathcal{C}_1$ and $\mathcal{C}_2$ are in strong minimal form and, according to Definition 7, we obtain $\mathcal{L}_1 \nabla_{\mathrm{c}} \mathcal{L}_2 = \mathrm{gcon}\bigl(\{5x + y \equiv_1 0\}\bigr)$. On the other hand, $\mathcal{C}'_1$ and $\mathcal{C}'_2$ are in strong minimal form when taking the variable order where y precedes x. In this case, by Definition 7, $\mathcal{L}_1 \nabla_{\mathrm{c}} \mathcal{L}_2 = \mathrm{gcon}\bigl(\{9y + x \equiv_1 0\}\bigr)$.

6 Conclusion

We have defined a domain of *grids* and shown that any element may be represented either by a congruence system which is a finite set of congruences (either equalities or proper congruences); or a generator system which is a triple of finite sets of vectors (denoting sets of lines, parameters and points). Assuming such a system in $\mathbb{Q}^n$ has m congruences or generators, then the minimization algorithms have worst-case complexity $\mathrm{O}\bigl(n^2 m\bigr)$. It is shown that any matrix inversion algorithms such as Gaussian elimination which has complexity $\mathrm{O}\bigl(n^3\bigr)$, can be used for converting between generator and congruence systems in minimal form. Thus, the complexity of converting any system with m elements is no worse than $\mathrm{O}\bigl(n^2 m\bigr)$ if $m > n$ and $\mathrm{O}\bigl(n^3\bigr)$, otherwise.

The minimization and conversion algorithms, form the basis for a double description method for grids so that any generator or congruence systems, possibly in minimal form, can be provided on demand; the complexity of such a provision being as stated above. Assuming this method, we have shown that operations for comparison, intersection and grid join are straightforward. The complexity of comparing two grids is $\mathrm{O}\bigl(n^3\bigr)$ but, for just checking equality when it is already known that one of the grids is a subset of the other, we have described simpler procedures with complexity $\mathrm{O}\bigl(n\bigr)$. The intersection and grid join just take the union of the congruence or generator systems, respectively, so that, from a theoretical perspective, these have complexity $\mathrm{O}\bigl(n\bigr)$. However, in the implementation, we assume a common divisor for all the coordinates or coefficients in the system; hence, combining the systems requires changing the denominators of both components to their least common multiple with a consequential need to scale all the numerators in the representation; giving a worst-case complexity of $\mathrm{O}\bigl(n^2\bigr)$. We have also described an algorithm for computing the grid difference with complexity $\mathrm{O}\bigl(n^4\bigr)$. Observe that this operator is useful in the specification of the certificate-based widening for the grid powerset domain [3].

The grid domain is implemented in the PPL [2, 4] following the approach described in this paper. Among the tests available in the PPL are the examples in this paper and implementations of the running examples in [22, 23]. The PPL provides full support for lifting any domain to the powerset of that domain, so that a user of the PPL can experiment with powersets of grids and the extra precision this provides. An interesting line of research is the combination of the grids domain with the polyhedral domains provided by the PPL: not only the $\mathbb{Z}$-polyhedra domain, but also many variations such as the grid-polyhedra, grid-octagon, grid-bounded-difference, grid-interval domains and their powersets.

References

1. C. Ancourt. *Génération Automatique de Codes de Transfert pour Multiprocesseurs à Mémoires Locales*. PhD thesis, Université de Paris VI, March 1991.
2. R. Bagnara, P. M. Hill, and E. Zaffanella. *The Parma Polyhedra Library User's Manual*. Department of Mathematics, University of Parma, Parma, Italy, release 0.9 edition, March 2006. Available at `http://www.cs.unipr.it/ppl/`.
3. R. Bagnara, P. M. Hill, and E. Zaffanella. Widening operators for powerset domains. *Software Tools for Technology Transfer*, 8(4/5):449–466, 2006.
4. R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In M. V. Hermenegildo and G. Puebla, editors, *Static Analysis: Proceedings of the 9th International Symposium*, volume 2477 of *Lecture Notes in Computer Science*, pages 213–229, Madrid, Spain, 2002. Springer-Verlag, Berlin.
5. D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.
6. P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In B. Robinet, editor, *Proceedings of the Second International Symposium on Programming*, pages 106–130, Paris, France, 1976. Dunod, Paris, France.
7. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the Fourth Annual ACM Symposium on Principles of Programming Languages*, pages 238–252, New York, 1977. ACM Press.
8. P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.
9. P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In M. Bruynooghe and M. Wirsing, editors, *Proceedings of the 4th International Symposium on Programming Language Implementation and Logic Programming*, volume 631 of *Lecture Notes in Computer Science*, pages 269–295, Leuven, Belgium, 1992. Springer-Verlag, Berlin.
10. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages*, pages 84–96, Tucson, Arizona, 1978. ACM Press.
11. R. Giacobazzi, editor. *Static Analysis: Proceedings of the 11th International Symposium*, volume 3148 of *Lecture Notes in Computer Science*, Verona, Italy, 2004. Springer-Verlag, Berlin.
12. P. Granger. Static analysis of arithmetical congruences. *International Journal of Computer Mathematics*, 30:165–190, 1989.
13. P. Granger. *Analyses Sémantiques de Congruence*. PhD thesis, École Polytechnique, 921128 Palaiseau, France, July 1991.
14. P. Granger. Static analysis of linear congruence equalities among variables of a program. In Samson Abramsky and T. S. E. Maibaum, editors, *TAPSOFT'91: Proceedings of the International Joint Conference on Theory and Practice of Software Development, Volume 1: Colloquium on Trees in Algebra and Programming (CAAP'91)*, volume 493 of *Lecture Notes in Computer Science*, pages 169–192, Brighton, UK, 1991. Springer-Verlag, Berlin.
15. P. Granger. Static analyses of congruence properties on rational numbers (extended abstract). In P. Van Hentenryck, editor, *Static Analysis: Proceedings of the*

4th International Symposium, volume 1302 of *Lecture Notes in Computer Science*, pages 278–292, Paris, France, 1997. Springer-Verlag, Berlin.
16. M. Karr. Affine relationships among variables of a program. *Acta Informatica*, 6:133–151, 1976.
17. S. Larsen, E. Witchel, and S. P. Amarasinghe. Increasing and detecting memory address congruence. In *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques (PACT'02)*, pages 18–29, Charlottesville, VA, USA, 2002. IEEE Computer Society Press.
18. V. Loechner. *PolyLib*: A library for manipulating parameterized polyhedra. Available at `http://icps.u-strasbg.fr/~loechner/polylib/`, March 1999. Declares itself to be a continuation of [30].
19. A. Miné. A few graph-based relational numerical abstract domains. In M. V. Hermenegildo and G. Puebla, editors, *Static Analysis: Proceedings of the 9th International Symposium*, volume 2477 of *Lecture Notes in Computer Science*, pages 117–132, Madrid, Spain, 2002. Springer-Verlag, Berlin.
20. T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall. The double description method. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games – Volume II*, number 28 in Annals of Mathematics Studies, pages 51–73. Princeton University Press, Princeton, New Jersey, 1953.
21. M. Müller-Olm and H. Seidl. Precise interprocedural analysis through linear algebra. In N. D. Jones and X. Leroy, editors, *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2004)*, pages 330–341, Venice, Italy, 2004. ACM Press.
22. M. Müller-Olm and H. Seidl. Analysis of modular arithmetic. In M. Sagiv, editor, *Programming Languages and Systems, Proceedings of the 14th European Symposium on Programming*, volume 3444 of *Lecture Notes in Computer Science*, pages 46–60, Edinburgh, UK, 2005. Springer-Verlag, Berlin.
23. M. Müller-Olm and H. Seidl. A generic framework for interprocedural analysis of numerical properties. In C. Hankin and I. Siveroni, editors, *Static Analysis: Proceedings of the 12th International Symposium*, volume 3672 of *Lecture Notes in Computer Science*, pages 235–250, London, UK, 2005. Springer-Verlag, Berlin.
24. S. P. K. Nookala and T. Risset. A library for Z-polyhedral operations. *Publication interne* 1330, IRISA, Campus de Beaulieu, Rennes, France, 2000.
25. P. Quinton, S. Rajopadhye, and T. Risset. On manipulating Z-polyhedra. Technical Report 1016, IRISA, Campus Universitaire de Bealieu, Rennes, France, July 1996.
26. P. Quinton, S. Rajopadhye, and T. Risset. On manipulating Z-polyhedra using a canonic representation. *Parallel Processing Letters*, 7(2):181–194, 1997.
27. E. Rodríguez-Carbonell and D. Kapur. An abstract interpretation approach for automatic generation of polynomial invariants. In Giacobazzi [11], pages 280–295.
28. S. Sankaranarayanan, H. Sipma, and Z. Manna. Constraint-based linear-relations analysis. In Giacobazzi [11], pages 53–68.
29. A. Schrijver. *Theory of Linear and Integer Programming*. Wiley Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, 1999.
30. D. K. Wilde. A library for doing polyhedral operations. Master's thesis, Oregon State University, Corvallis, Oregon, December 1993. Also published as IRISA *Publication interne* 785, Rennes, France, 1993.