Exploiting Sparsity in Polyhedral Analysis

Axel Simon and Andy King

Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK
{a.simon,a.m.king}@kent.ac.uk

Abstract. The intrinsic cost of polyhedra has lead to research on more tractable sub-classes of linear inequalities. Rather than committing to the precision of such a sub-class, this paper presents a projection algorithm that works directly on any sparse system of inequalities and which sacrifices precision only when necessary. The algorithm is based on a novel combination of the Fourier-Motzkin algorithm (for exact projection) and Simplex (for approximate projection). By reformulating the convex hull operation in terms of projection, conversion to the frame representation is avoided altogether. Experimental results conducted on logic programs demonstrate that the resulting analysis is efficient and precise.

1 Introduction

Recently there has been much interest in so-called weakly relational domains [7, 25, 29] that trade the precision of operations on systems of linear inequalities for improved tractability. These domains seek to address the scalability problems of the polyhedral domain [8] whose operations are inherently exponential, irrespective of the algorithms used to implement them: Chandru *et al.* [6] showed that eliminating variables from a system of inequalities can increase the number of inequalities exponentially; Benoy *et al.* [2] showed that polytopes (bounded polyhedra) exist whose convex hull is exponential in the number of inequalities defining the input polytopes. Exponential growth also can arise when converting into the frame representation, which is the classical approach for computing the convex hull and projection [21]. Consider, for example, the convex hull of two n-dimensional hypercubes where one is translated along one axis. The frame consists of 2^n vertices for each hypercube. However, the resulting hull can be represented by $2n$ inequalities, just as the inputs. A natural question is whether there are faster methods to calculate the convex hull that do not convert the input polyhedra into their frame representation and that over-approximate the output polyhedron in case the resulting set of inequalities has exponential size.

One answer to this question is represented by the class of weakly relational domains where inequalities are restricted in order to prevent exponential growth. The Octagon domain [25] uses inequalities of the form $\pm x_i \pm x_j \leq c_{i,j}$ where x_i and x_j are variables and $c_{i,j}$ is a constant. In this domain the convex hull reduces to calculating the element-wise maximum of two matrices. The Octagon domain was generalised into the Octahedron domain [7], allowing more than two variables with zero or unary coefficients whilst maintaining a hull operation that is polynomial in the number of variables. Finally, the two variables per inequality (TVPI) domain [29] allows arbitrary coefficients. This domain stores a planar polyhedron for each variable pair and employs a convex hull algorithm that operates on planar polyhedra [28]. All these domains employ a closure operation to propagate information between inequalities. Even incremental versions of these closure operations are quadratic, hence Blanchet *et al.* advocate a packing strategy when analysing large-scale programs [3]. They keep a set of Octagons, each describing relationships between variables occurring in a pack. Packs can overlap and are chosen by examining which variables occur in the same program statement. Packs are determined up front and hence packing variables is a commitment to a fixed degree of precision. Interestingly their program can be verified with packs that contain no more than four variables on average which suggests that useful inequalities contain relatively few variables. Halbwachs *et al.* also exploit the loose coupling of variables by partitioning the variable set into non-overlapping groups [13]. By applying the standard domain operations independently to each partition (rather than over the whole set of variables) useful speedups are obtained.

This paper shows how to exploit the fact that a given variable typically occurs in only a few inequalities. The key observation is that projection on these sparse systems can be realised efficiently by carefully applying the Fourier-Motzkin method [26]. We restrict the size of the output and the intermediate systems to be no larger than that of the input system which avoids exponential growth in the number of inequalities, thereby providing a performance guarantee. Surprisingly, even with this draconian size restriction the vast majority of variables can be eliminated. In the remaining cases we use Simplex to approximate the projection space by combining those inequalities that still contain uneliminated variables. Our method creates one inequality in the projection space for each call to Simplex. Simplex is called once for each remaining inequality which ensures that the final system is no larger than the original. This second stage over-approximates the projection (if applied at all). In terms of complexity, Fourier-Motzkin eliminates n variables in $O(nm)$ time where m is the number of inequalities. When variables remain to be eliminated, no more than m Simplex queries are performed where each query operates over m dimensions and n inequalities (note that n and m are exchanged). This method is attractive because, although Simplex is not a polynomial-time algorithm, the number of pivoting steps is about linear in the number of dimensions [27] and each pivoting step is in $O(nm)$ for the Simplex method in the tableau form. In fact, the average number of steps is polynomial [4]. To complete the set of domain operations, convex hull is recast in terms of projection [2] so that the frame representation is avoided altogether.

The remainder of the paper is structured as follows: After Section 2 introduces necessary mathematical notation, Section 3 presents techniques for using Fourier-Motzkin variable elimination for sparse inequality systems. Section 4 describes an approximation to projection. Section 5 describes how these efficient projection algorithms can be used to calculate convex hull. The paper finishes with sections on performance evaluation and related work before concluding.

2 Preliminaries

Let $Lin^{=}_X$ and $Lin^{\leq}_X$ denote the set of linear equalities and inequalities, respectively, defined over a finite set of variables X. Elements of $Lin^{=}_X$ and $Lin^{\leq}_X$ take the form of $\boldsymbol{c} \cdot \boldsymbol{y} = b$ and $\boldsymbol{c} \cdot \boldsymbol{y} \leq b$ where $|\boldsymbol{c}| = |\boldsymbol{y}|$, $b \in \mathbb{Z}$ and the elements of $\boldsymbol{c}$ and $\boldsymbol{y}$ are drawn from $\mathbb{Z}$ and X respectively. Furthermore let Con_X denote the set of all finite subsets of $Lin^{=}_X \cup Lin^{\leq}_X$ and $Ineq_X$ the set of all finite subsets of $Lin^{\leq}_X$. The set of real solutions for $\boldsymbol{c} \cdot \boldsymbol{y} \leq b$ is defined by:

$$soln^{\mathbb{R}}_{\boldsymbol{x}}(\boldsymbol{c} \cdot \boldsymbol{y} \leq b) = \left\{ \langle r_1, \ldots r_n \rangle \in \mathbb{R}^n \;\middle|\; \begin{array}{l} \boldsymbol{c} \cdot \langle r'_1, \ldots r'_m \rangle \leq b \quad \wedge \\ r_i = r'_j \text{ for all } x_i = y_j \end{array} \right\}$$

where $\boldsymbol{x} = \langle x_1, \ldots x_n \rangle$ and $\boldsymbol{y} = \langle y_1, \ldots y_m \rangle$. The real solution set for $\boldsymbol{c} \cdot \boldsymbol{y} = b$ is defined likewise and for any linear system $E \in Con_X$ the real solution set for E is defined $soln^{\mathbb{R}}_{\boldsymbol{x}}(E) = \cap_{e \in E} soln^{\mathbb{R}}_{\boldsymbol{x}}(e)$. Two linear systems $E_1, E_2 \in Con_X$ are partially ordered by the subset relation on their solution sets, that is, $E_1 \models_{\mathbb{R}} E_2$ iff $soln^{\mathbb{R}}_{\boldsymbol{x}}(E_1) \subseteq soln^{\mathbb{R}}_{\boldsymbol{x}}(E_2)$ where $var(\boldsymbol{x}) = var(E_1) \cup var(E_2)$ and $var(o)$ denotes the set of variables in a syntactic object o. The set of integer solutions $soln^{\mathbb{Z}}_{\boldsymbol{x}}$ can be defined analogously to $soln^{\mathbb{R}}_{\boldsymbol{x}}$ to induce a different partial order $E_1 \models^{\mathbb{Z}} E_2$. The ordering $\models^{\mathbb{R}}$ over-approximates $\models^{\mathbb{Z}}$ in the sense that if $E_1 \models^{\mathbb{R}} E_2$ then $E_1 \models^{\mathbb{Z}} E_2$; this is convenient in applications that are concerned with integral entities because (domain) operations associated with the ordering $\models^{\mathbb{R}}$ are more tractable than those induced by $\models^{\mathbb{Z}}$ [27]. Thus, henceforth, $\models$ and $soln_{\boldsymbol{x}}$ will abbreviate $\models^{\mathbb{R}}$ and $soln^{\mathbb{R}}_{\boldsymbol{x}}$ respectively. The predicate $sat \subseteq Con_X$ is defined so that $sat(E)$ holds iff $soln_{\boldsymbol{x}}(E) \neq \emptyset$ where $var(\boldsymbol{x}) = var(E)$. Finally, let *false* denote a particular system $E \in Con_X$ such that $sat(\mathit{false})$ does not hold.

3 Fourier-Motzkin Projection

Eliminating a variable from two equalities by scaling and adding them is a well known principle that is attributed to Gauss. Fourier refined this elimination strategy to pairs of inequalities. The basic observation is that inequalities may only be scaled by non-negative numbers which implies that the coefficients of the variable to be eliminated must have opposing signs in the two inequalities. His method was later elaborated on by Motzkin and henceforth it will be referred to as the Fourier-Motzkin variable elimination. While this algorithm has been thoroughly studied [11, 15, 18], little practical work has been reported on combining different refinements. This section presents strategies that are useful for program analysis – each strategy is reported in a separate sub-section.

Algorithm 1 presents the basic Fourier-Motzkin algorithm to remove a variable $x_r \in X$ from a system of inequalities $E \in Ineq_X$. E is partitioned into E^+, E^r and E^-, corresponding to inequalities that have positive, zero and negative coefficient for x_r. E^r is augmented to obtain the projection by combining positive multiples of pairs of inequalities drawn from E^+ and E^-. The variable x_r is eliminated since the coefficient of x_r in each inequality added to

Algorithm 1 Fourier-Motzkin $\mathit{fourier}(x_r, E)$

Require: $x_r \in X$, $E \in \mathit{Ineq}_X$

$\langle E^+, E^r, E^- \rangle \leftarrow \langle \emptyset, \emptyset, \emptyset \rangle$
for $\boldsymbol{a} \cdot \boldsymbol{x} \leq c \in E$ **do**
  if $\pi_r(\boldsymbol{a}) = 0$ **then**
    $E^r \leftarrow E^r \cup \{\boldsymbol{a} \cdot \boldsymbol{x} \leq c\}$
  else if $\pi_r(\boldsymbol{a}) > 0$ **then**
    $E^+ \leftarrow E^+ \cup \{\boldsymbol{a} \cdot \boldsymbol{x} \leq c\}$
  else
    $E^- \leftarrow E^- \cup \{\boldsymbol{a} \cdot \boldsymbol{x} \leq c\}$
for $\boldsymbol{a}^+ \cdot \boldsymbol{x} \leq c^+ \in E^+$ **do**
  for $\boldsymbol{a}^- \cdot \boldsymbol{x} \leq c^- \in E^-$ **do**
    $\boldsymbol{a} \cdot \boldsymbol{x} \leq c \leftarrow \mathit{simplify}(\ (\pi_r(\boldsymbol{a}^+)\boldsymbol{a}^- + |\pi_r(\boldsymbol{a}^-)|\boldsymbol{a}^+) \cdot \boldsymbol{x} \leq (\pi_r(\boldsymbol{a}^+)c^- + |\pi_r(\boldsymbol{a}^-)|c^+)\ )$
    if $\boldsymbol{a} \neq \boldsymbol{0}$ **then**
      $E^r \leftarrow E^r \cup \{\boldsymbol{a} \cdot \boldsymbol{x} \leq c\}$
    else if $c < 0$ **then**
      return *false*
return E^r

E^r is $\pi_r(\boldsymbol{a}^+)\pi_r(\boldsymbol{a}^-) + |\pi_r(\boldsymbol{a}^-)|\pi_r(\boldsymbol{a}^+) = 0$ where $\boldsymbol{a}^+ \in E^+$ and $\boldsymbol{a}^- \in E^-$ and $\pi_i(\langle a_1, \ldots a_n \rangle) = a_i$. Note that a generated inequality might take the form $\boldsymbol{0} \cdot \boldsymbol{x} \leq c$. If $c < 0$, the original system E is unsatisfiable and *false* is returned as the projection, otherwise the inequality is a tautology [20] and is discarded.

3.1 Simplification

To identify equivalent inequalities, a unique representation is desirable. The *simplify* function presented as Algorithm 2 is designed to remove common factors from a newly generated inequality. The algorithm is generic in the sense that it supports equalities, so that it is also applicable in Gaussian elimination (as discussed in Section 3.5). In both cases the function is the identity if all coefficients are zero since this represents either a tautology or a contradiction. Otherwise an inequality is divided by the greatest common denominator of its coefficients. Note that dividing the constant might not result in an integral number and therefore the result is rounded down. This is sound only if integer entities are represented. In the case of equalities, the assignment $g \leftarrow c$ ensures that the *gcd* calculation also considers the constant so that the division has no remainder, thereby guaranteeing that the exact equality relationship is preserved.

3.2 Variable Selection

Whenever a set of variables $Y = \{y_1, \ldots y_n\}$ needs to be projected out, the Fourier-Motzkin algorithm can be applied iteratively by setting $E_0 = E$ and $E_i = \mathit{fourier}(y_i, E_{i-1})$. In each step, $|E_i^+| + |E_i^-|$ inequalities are removed from E_i and $|E_i^+||E_i^-|$ are added. Hence the growth in each step is in $O(|E_i|^2)$ and the number of inequalities in the final system E_n is in $O(|E|^{2^n})$ which prohibits

Algorithm 2 Simplification *simplify*($\boldsymbol{a} \cdot \boldsymbol{x} \odot c$) where $\odot \in \{\leq, =\}$

```
if a = 0 then
   return a · x ⊙ c
if ⊙ ∈ {≤} then
   g ← 0
else
   g ← c
for a_i ∈ a do
   if a_i ≠ 0 then
      if g = 0 then
         g ← a_i
      else
         g ← gcd(g, a_i)
return (a/g) · x ⊙ ⌊c/g⌋
```

Algorithm 3 Select variable *select*(Y, E)

```
Require: E ∈ Ineq_X, Y ⊆ X
⟨p_1, ... p_|X|⟩ ← 0
⟨m_1, ... m_|X|⟩ ← 0
for a · x ≤ c ∈ E do
   for i ∈ {1, ... |X|} do
      if π_i(a) > 0 then
         p_i ← p_i + 1
      else if π_i(a) < 0 then
         m_i ← m_i + 1
bestGrowth ← |E|^2
for x_i ∈ Y do
   growth ← p_i m_i − (p_i + m_i)
   if growth < bestGrowth then
      bestGrowth ← growth
      bestVar ← x_i
return ⟨bestGrowth, bestVar⟩
```

direct use of this method even for projecting out a few variables. A standard rule [11] suggests delaying the growth of the intermediate systems by always eliminating the variable that minimises $|E_i^+||E_i^-| - (|E_i^+| + |E_i^-|)$. Algorithm 3 calculates how many positive and negative coefficients each variable has in the given inequality system E. It returns the variable x_r such that applying Fourier-Motzkin elimination will result in minimal growth.

3.3 Complete Redundancy Removal

Each Fourier-Motzkin step may introduce redundant inequalities. Algorithm 4 uses the Simplex method to check every inequality for redundancy. The function *simplex*$(\boldsymbol{a}, \boldsymbol{x}, E)$ calculates a vector $\boldsymbol{m}$ that maximises $\boldsymbol{a} \cdot \boldsymbol{x}$ subject to the linear inequalities in E. Running *compress* after each Fourier-Motzkin elimination step

Algorithm 4 Complete Redundancy Removal *compress*(E)

Require: $E \in \mathit{Ineq}_X$
  if $\neg sat(E)$ **then**
    return *false*
  for $\boldsymbol{a} \cdot \boldsymbol{x} \leq c \in E$ **do**
    $\boldsymbol{m} \leftarrow simplex(\boldsymbol{a}, \boldsymbol{x}, E \setminus \{\boldsymbol{a} \cdot \boldsymbol{x} \leq c\})$
    if $\boldsymbol{m} \cdot \boldsymbol{a} \leq c$ **then**
      $E \leftarrow E \setminus \{\boldsymbol{a} \cdot \boldsymbol{x} \leq c\}$
  return E

Algorithm 5 Quasi-Syntactic Redundancy Removal *quasi*(E)

Require: $E \in \mathit{Ineq}_X$
  while $\{\boldsymbol{a_1} \cdot \boldsymbol{x} \leq c_1, \boldsymbol{a_2} \cdot \boldsymbol{x} \leq c_2\} \subseteq E \wedge \boldsymbol{a_1} = \boldsymbol{a_2}$ **do**
    if $c_1 > c_2$ **then**
      $E \leftarrow E \setminus \{\boldsymbol{a_1} \cdot \boldsymbol{x} \leq c_1\}$
    else
      $E \leftarrow E \setminus \{\boldsymbol{a_2} \cdot \boldsymbol{x} \leq c_2\}$
  return E

is prohibitively expensive and therefore it is desirable to only apply *compress* when more lightweight redundancy removal algorithms fail to constrain growth.

3.4 Quasi-Syntactic Redundancy Removal

Lassez *et al.* identify several classes of redundant inequalities that can be detected by purely syntactic means [20]. For instance, inequalities with identical coefficients are called syntactically redundant (if the constant is equal) and quasi-syntactically redundant (if the constants differ). Given a pair of quasi-syntactic redundant inequalities, only the one with the smaller constant needs to be retained. Algorithm 5 removes both classes of redundancy by examining pairs of inequalities. In practise, inequalities can be sorted lexicographically by their coefficients which allows the algorithm to run in $O(|E| \log |E|)$.

3.5 Equality Removal

Rather than modelling an equality as two opposing inequalities, it is more prudent to retain equalities that arise during the analysis and precede the Fourier-Motzkin elimination with a Gaussian elimination phase. Algorithm 6 takes as input the system of equalities and inequalities E and the set of variables Y that are to be eliminated. It returns as output a triple consisting of a set of variables that remain to be eliminated, a set of equalities P in the projection space, and a system of inequalities that still retain variables to be eliminated. The algorithm iterates as long as there remains an equality $\boldsymbol{a} \cdot \boldsymbol{x} = c \in E$. If there exists a coefficient $\pi_i(\boldsymbol{a}) \neq 0$ and $\pi_i(\boldsymbol{x}) \in Y$ then Gaussian elimination is performed on all inequalities and remaining equalities that contain a non-zero coefficient for

Algorithm 6 Equality Removal $\mathit{gauss}(Y, E)$

Require: $E \in \mathit{Con}_X$, $Y \subseteq X$
  $P \leftarrow \emptyset$
  while $\boldsymbol{a} \cdot \boldsymbol{x} = c \in E$ **do**
    $E \leftarrow E \setminus \{\boldsymbol{a} \cdot \boldsymbol{x} = c\}$
    $s \leftarrow -1$
    for $x_i \in Y$ **do**
      if $\pi_i(\boldsymbol{a}) \neq 0$ **then**
        $s \leftarrow i$
    if $s = -1$ **then**
      $P \leftarrow P \cup \{\boldsymbol{a} \cdot \boldsymbol{x} = c\}$
      $s \leftarrow i$ **such that** $\pi_i(\boldsymbol{a}) \neq 0$
    $Y \leftarrow Y \setminus \{x_s\}$
    if $\pi_s(\boldsymbol{a}) < 0$ **then**
      $\langle \boldsymbol{a}, c \rangle \leftarrow \langle -\boldsymbol{a}, -c \rangle$
    $E' \leftarrow \emptyset$
    for $\boldsymbol{b} \cdot \boldsymbol{x} \odot d \in E$ where $\odot \in \{\leq, =\}$ **do**
      if $\pi_s(\boldsymbol{b}) = 0$ **then**
        $E' \leftarrow E' \cup \{\boldsymbol{b} \cdot \boldsymbol{x} \odot d\}$
      else
        $\boldsymbol{e} \cdot \boldsymbol{x} \odot f \leftarrow \mathit{simplify}(\ (a_s \boldsymbol{b} - b_s \boldsymbol{a}) \cdot \boldsymbol{x} \odot (a_s d - b_s c)\ )$
        if $\boldsymbol{e} = \boldsymbol{0}$ **then**
          if $(\odot \in \{\leq\} \wedge f < 0) \vee (\odot \in \{=\} \wedge f \neq 0)$ **then**
            return $\langle Y, \mathit{false}, P \rangle$
        else
          $E' \leftarrow E' \cup \{\boldsymbol{e} \cdot \boldsymbol{x} \odot f\}$
    $E \leftarrow E'$
  return $\langle Y, E, P \rangle$

$\pi_i(\boldsymbol{x})$. Since $\pi_i(\boldsymbol{x})$ is to be eliminated, the equality is then discarded. Alternatively, if there is no variable $\pi_i(\boldsymbol{x}) \in Y$ with $\pi_i(\boldsymbol{a}) \neq 0$ then the equality is part of the projection space P. Observe that each iteration of the while loop makes progress in the sense that it reduces the set of variables that appear in E.

The value of applying Gaussian elimination is fourfold: (1) it avoids reformulating each equality as two inequalities; (2) it reduces the number of inequalities that Fourier-Motzkin is applied to; (3) it reduces the number of variables that remain to be eliminated and perhaps most subtly (4) it increases the number of inequalities that can be identified as quasi-syntactically redundant. The last point stems from the observation that substituting an equality into a system often reformulates one inequality to the extent that it becomes quasi-syntactically redundant with respect to another [20]. This motivates the substitution of all equalities, even those that do not contain variables to be eliminated.

3.6 Combining All Strategies

This section composes the strategies previously presented so as to ensure tractability even in those pathological cases when the size of the projection is exponential [2]. The overall projection method is presented as Algorithm 7 and takes a linear system E and a set of variables Y that are to be eliminated. The algorithm applies Gaussian elimination to produce a system of inequalities E no larger than the initial input. Fourier-Motzkin elimination is then performed, which is interleaved with quasi-syntactic redundancy removal, until no more variables can be eliminated without exceeding the preset limit. Due to sparsity, a variable will often only appear once with a certain polarity (say with a positive coefficient). In this case the number of inequalities removed will be $|E^+|+|E^-| = 1+n$ and the number of newly created inequalities is at most $|E^+||E^-| = n$ which makes the system shrink. Another frequently occurring case is that of $|E^+| = |E^-| = 2$. If the limit is exceeded, complete redundancy removal is activated in an attempt to remove enough inequalities to resume Fourier-Motzkin. At this stage, the limit is further reduced to $|E|$. This is good practise since E is usually reduced considerably. Finally, if Fourier-Motzkin cannot be reapplied and variables remain to be eliminated, the system E is partitioned into those inequalities E' that contain variables in Y and into those in P that do not. The projection of the set E' is approximated by the extreme point projection which is presented next.

4 Extreme Point Projection

While the Fourier-Motzkin method works well on sparse systems, Huynh *et al.* [14] propose using the extreme point method of Lassez [19] for dense systems. This method can find inequalities in the projection space incrementally, thereby enabling the projection to be approximated with a limited number of inequalities. To illustrate the method, consider eliminating the variables Y from a linear system $E = \{\boldsymbol{a}_1 \cdot \boldsymbol{x} \leq c_1, \ldots \boldsymbol{a}_n \cdot \boldsymbol{x} \leq c_n\}$. W.l.o.g., let $Y = var(\boldsymbol{y})$ and

$$\begin{pmatrix} \boldsymbol{a}_1 \\ \hline \vdots \\ \hline \boldsymbol{a}_n \end{pmatrix} = (A|B) \qquad \begin{pmatrix} x_1 \\ \hline \vdots \\ \hline x_m \end{pmatrix} = \begin{pmatrix} \boldsymbol{y} \\ \hline \boldsymbol{z} \end{pmatrix} \qquad \boldsymbol{c} = \begin{pmatrix} c_1 \\ \hline \vdots \\ \hline c_n \end{pmatrix}$$

where $\boldsymbol{y}$ and $\boldsymbol{z}$ are column vectors and $\boldsymbol{x} = \langle x_1, \ldots x_m\rangle$. Then $A\boldsymbol{y} + B\boldsymbol{z} \leq \boldsymbol{c}$ is equivalent to E and the problem of calculating an inequality in the projection space reduces to finding non-negative linear combinations $\boldsymbol{\lambda} \in \mathbb{R}^n$ of rows of A such that $\boldsymbol{\lambda}A = \boldsymbol{0}$. Then $\boldsymbol{\lambda}(A\boldsymbol{y} + B\boldsymbol{z}) \leq \boldsymbol{\lambda}\boldsymbol{c}$ and $\boldsymbol{\lambda}(A\boldsymbol{y} + B\boldsymbol{z}) = \boldsymbol{\lambda}B\boldsymbol{z}$ hence $(\boldsymbol{\lambda}B)\boldsymbol{z} \leq \boldsymbol{\lambda}\boldsymbol{c}$ which yields an inequality in the projection space. The vector $\boldsymbol{\lambda} = \boldsymbol{0}$ is the trivial solution to the system $\boldsymbol{\lambda}A = \boldsymbol{0}$ yielding a tautology. Observe that if $\boldsymbol{\lambda} \in \mathbb{R}^n$ is a solution to $\boldsymbol{\lambda}A = \boldsymbol{0}$ and $\{\lambda_i \geq 0 \mid \lambda_i \in \boldsymbol{\lambda}\}$, then so is $s\boldsymbol{\lambda}$ where $s \in \mathbb{R}$ is any non-negative scalar. Hence, w.l.o.g., we can enforce the constraint $\lambda_1 + \ldots + \lambda_n = 1$. The set of all extreme points of the bounded space, given by $\boldsymbol{\lambda}A = \boldsymbol{0}$, $\{\lambda_i \geq 0 \mid \lambda_i \in \boldsymbol{\lambda}\}$ and $\lambda_1 + \ldots + \lambda_n = 1$, corresponds to the exact projection which potentially contains an exponential number of inequalities. To give a performance guarantee, we only enumerate $|E|$ extreme points thereby ensuring that the number of inequalities does not grow beyond the set limit.

Since extreme point enumeration does not consider the B matrix, two extreme points $\boldsymbol{\lambda}_a \neq \boldsymbol{\lambda}_b$ might produce the same coefficient vector $\boldsymbol{\lambda}_a B = \boldsymbol{\lambda}_b B$

Algorithm 7 Projection $project(Y, E)$

Require: $E \in Con_X$, $Y \subseteq X$
  $\langle Y, E, P\rangle \leftarrow gauss(Y, E)$
  if $\neg sat(E)$ **then**
    return $false$
  $limit \leftarrow |E|$
  $\langle g, x_i\rangle \leftarrow select(Y, E)$
  while $Y \neq \emptyset \wedge |E| + g \leq limit$ **do**
    $E \leftarrow fourier(x_i, E)$
    if $E = false$ **then**
      return $false$
    $E \leftarrow quasi(E)$
    $Y \leftarrow Y \setminus \{x_i\}$
    $\langle g, x_i\rangle \leftarrow select(Y, E)$
    if $|E| + g > limit$ **then**
      $E \leftarrow compress(E)$
      if $E = false$ **then**
        return $false$
      $limit \leftarrow |E|$
      $\langle g, x_i\rangle \leftarrow select(Y, E)$
  if $Y = \emptyset$ **then**
    return $compress(E \cup P)$
  $E' \leftarrow \emptyset$
  for $\boldsymbol{a} \cdot \boldsymbol{x} \leq c \in E$ **do**
    if $\exists x_i \in Y . \pi_i(\boldsymbol{a}) \neq 0$ **then**
      $E' \leftarrow E' \cup \{\boldsymbol{a} \cdot \boldsymbol{x} \leq c\}$
    else
      $P \leftarrow P \cup \{\boldsymbol{a} \cdot \boldsymbol{x} \leq c\}$
  return $compress(extreme(Y, E') \cup P)$

such that one of the resulting inequalities will be quasi-syntactically redundant. Kohler [18] observed that if the set of indices containing zero coefficients in $\boldsymbol{\lambda}_a$ is a strict superset of those of $\boldsymbol{\lambda}_b$, then the latter leads to a redundant inequality. This observation can be exploited by maximising the number of zero coefficients in each $\boldsymbol{\lambda}$ which is the indirect result of running a linear program that maximises a specific $\lambda_i \in \boldsymbol{\lambda}$. Algorithm 8 formalises this heuristic. As a final comment, note that Fourier-Motzkin elimination can be seen as a special case of the extreme point method where A only contains one column (and hence one variable to eliminate). The extreme points are those solutions that combine exactly one positive row with one negative row in A.

5 Convex Hull via Projection

The convex hull operation takes as input two inequality sets $E_1, E_2 \in Con_X$ and produces as output an $E \in Con_X$ such that $soln_{\boldsymbol{x}}(E_i) \subseteq soln_{\boldsymbol{x}}(E)$, $soln_{\boldsymbol{x}}(E)$ is minimal and $var(\boldsymbol{x}) = var(E_1 \cup E_2)$. For purpose of exposition, let E_1 and E_2

Algorithm 8 Extreme-Point Projection $extreme(Y, E)$

Require: $E \in Ineq_X$, $Y \subseteq X$

$(A|B) \leftarrow \begin{pmatrix} \boldsymbol{a}_1 \\ \vdots \\ \boldsymbol{a}_n \end{pmatrix}$ **where** $(\boldsymbol{a}_i \cdot \boldsymbol{x} \leq c_i) \in E, var(\boldsymbol{y}) = Y$ **and** $A\boldsymbol{y} + B\boldsymbol{z} \leq \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix}$

$\Lambda \leftarrow \boldsymbol{\lambda}A = 0 \cup \{\lambda_i \geq 0 \mid \lambda_i \in \boldsymbol{\lambda}\} \cup \{\sum \lambda_i = 1\}$

$E' \leftarrow \emptyset$

for $\boldsymbol{f} \in \langle 1, 0, \ldots 0\rangle, \langle 0, 1, \ldots 0\rangle \ldots \langle 0, 0, \ldots 1\rangle$ **do**

  $\boldsymbol{m} \leftarrow simplex(\boldsymbol{f}, \boldsymbol{\lambda}, \Lambda)$

  $e \leftarrow simplify(\boldsymbol{m}B \leq \boldsymbol{m} \cdot \langle c_1, \ldots c_n\rangle)$

  $E' \leftarrow E' \cup \{e\}$

return E'

be represented in matrix form as $A_i\boldsymbol{x} \leq \boldsymbol{c}_i$, $i = 1, 2$ (with the equalities in E_i expressed as two rows in A_i). The smallest convex set of points P that includes $soln_{\boldsymbol{x}}(E_1) \cup soln_{\boldsymbol{x}}(E_2)$ is given by

$$P = \left\{ \boldsymbol{x} \,\middle|\, \begin{array}{l} \boldsymbol{x} = \sigma_1\boldsymbol{x}_1 + \sigma_2\boldsymbol{x}_2 \wedge \sigma_1 + \sigma_2 = 1 \wedge \sigma_1 \geq 0 \wedge \\ A_1\boldsymbol{x}_1 \leq \boldsymbol{c}_1 \qquad\quad \wedge A_2\boldsymbol{x}_2 \leq \boldsymbol{c}_2 \ \ \wedge \sigma_2 \geq 0 \end{array} \right\}.$$

To avoid the non-linearity $\boldsymbol{x} = \sigma_1\boldsymbol{x}_1 + \sigma_2\boldsymbol{x}_2$, the system can be relaxed by setting $\boldsymbol{y}_1 = \sigma_1\boldsymbol{x}_1$ and $\boldsymbol{y}_2 = \sigma_2\boldsymbol{x}_2$ so that $\boldsymbol{x} = \boldsymbol{y}_1 + \boldsymbol{y}_2$ and $A_i\boldsymbol{y}_i \leq \sigma_i\boldsymbol{c}_i$ to define:

$$P' = \left\{ \boldsymbol{x} \,\middle|\, \begin{array}{l} \boldsymbol{x} = \boldsymbol{y}_1 + \boldsymbol{y}_2 \ \wedge \sigma_1 + \sigma_2 = 1 \ \ \wedge \sigma_1 \geq 0 \wedge \\ A_1\boldsymbol{y}_1 \leq \sigma_1\boldsymbol{c}_1 \wedge A_2\boldsymbol{y}_2 \leq \sigma_2\boldsymbol{c}_2 \wedge \sigma_2 \geq 0 \end{array} \right\}.$$

Note that, although $P \subseteq P'$, in general $P \neq P'$ since P might not be topologically closed whereas P' is represented by a set of (non-strict) inequalities and therefore is closed. In fact projecting out σ_i and the variables in $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ yields a system E representing the closure of the convex hull of the two input polyhedra E_1 and E_2 as is formally proved in [2]. Henceforth let $hull(E_1, E_2) = E$ encapsulate this computational tactic for calculating the convex hull. Since entailment can be realised straightforwardly with Simplex, this section completes the suite of polyhedral domain operations without recourse to the frame representation.

6 Performance Evaluation

In order to assess the precision and efficiency of the domain operations reported thus far, the algorithms have been integrated into an argument size analyser [10]. The analysis is key to termination checking [10], termination inference [24], control generation [17] and determinacy inference [23]. The last application uses argument size relationships that are synthesised for each clause in the program to infer a determinacy condition for each predicate that, if satisfied by a call, guarantees that there is at most one computed answer for that call and that the answer is produced only once if ever. The value of this analysis quickly degrades unless three variable inequalities can be inferred (see [23, Section 2.2]) which precludes the use of octagons [25] or the TVPI domain [29].

6.1 Argument-Size Analysis of Logic Programs

This section summarises the essential details of an argument-size analysis. The analysis abstracts the standard T_P [22] operator of a logic program P. In this presentation, T_P is defined for clauses of the form $p(\boldsymbol{x}) \leftarrow H, p_1(\boldsymbol{x}_1), \ldots p_n(\boldsymbol{x}_n)$ where $\boldsymbol{x}$ and $\boldsymbol{x}_i$ are vectors of variables, H is a finite (possibly empty) set of Herbrand equations $\{s_1 = t_1, \ldots s_n = t_n\}$ and s_i and t_i are arbitrary terms. The set of unifiers of H is denoted by $\mathit{unify}(H)$. For a given clause c, the operator $T_c(I)$ maps one set of ground atoms I to another in the following manner:

$$T_c(I) = I \cup \left\{ \theta(p(\boldsymbol{x})) \;\middle|\; \begin{array}{c} c = p(\boldsymbol{x}) \leftarrow H, p_1(\boldsymbol{x}_1), \ldots p_n(\boldsymbol{x}_n) \wedge \\ \mathit{var}(\theta(c)) = \emptyset \hfill \wedge \\ \theta \in \mathit{unify}(H) \;\;\wedge\;\; \theta(p_i(\boldsymbol{x}_i)) \in I \end{array} \right\}$$

The condition $\mathit{var}(\theta(c)) = \emptyset$ ensures that the substitution θ grounds c, hence the atom $\theta(p(\boldsymbol{x}))$ is variable-free. The operator lifts to a program $P = \{c_1, \ldots c_n\}$ by defining $T_P(I) = I_n$ where $I_0 = I$ and $I_i = T_{c_i}(I_{i-1})$. Since T_P is monotonic and the computation domain of sets of ground atoms constitutes a complete lattice under the subset ordering, then $\mathrm{lfp}(T_P)$ exists which provides a convenient fixpoint formulation of the semantics of P [22].

Argument-size analysis aspires to find size invariants for each p that describe a tuple of terms $\boldsymbol{t}$ whenever $p(\boldsymbol{t}) \in \mathrm{lfp}(T_P)$. Size is quantified in terms of a norm that maps a ground term to a non-negative size. In our experiments we use the term-size norm $|.|_{\mathit{term\text{-}size}}$ [9] which is defined as follows:

$$|t|_{\mathit{term\text{-}size}} = \begin{cases} 1 + \sum_{i=1}^{n} |t_i|_{\mathit{term\text{-}size}} & \text{if } t = f(t_1, \ldots t_n) \wedge n > 0 \\ 0 & \text{otherwise} \end{cases}$$

The established approach to finding such invariants involves describing Herbrand (syntactic) equations with linear equations. Formally, a linear equation $\boldsymbol{c} \cdot \boldsymbol{x} = b$ describes $s = t$ with respect to $|.|$, denoted by $(\boldsymbol{c} \cdot \boldsymbol{x} = b) \propto_{|.|} (s = t)$, iff $|\theta(\boldsymbol{x})| \in \mathit{soln}_{\boldsymbol{x}}(\boldsymbol{c} \cdot \boldsymbol{x} = b)$ whenever θ is a grounding substitution for $s = t$ such that $\theta \in \mathit{unify}(\{s = t\})$ where $|\langle t_1, \ldots t_n \rangle| = \langle |t_1|, \ldots |t_n| \rangle$. Since $\propto_{|.|}$ is a relation, a natural question is whether there is a best description of a given Herbrand equation $s = t$. In fact, this is given by $e = \alpha_{|.|}(s = t)$ where $\alpha_{|.|}$ is defined such that e is the best abstraction with $e \propto_{|.|} (s = t)$. For the term-size norm, and more generally the class of semi-linear norms [5], the function $\alpha_{|.|}$ is well-defined. The mapping $\alpha_{|.|}$ extends to sets of Herbrand equations by $\alpha_{|.|}(H) = \{\alpha_{|.|}(s_i = t_i) \mid (s_i = t_i) \in H\}$.

Example 1. To illustrate, consider the equation $\mathtt{C} = \mathtt{succ(N)} * \mathtt{pow(X,N)}$ where $*$ is an infix functor. The linear equation $\mathtt{C} = 3 + \mathtt{X} + 2 * \mathtt{N}$ describes the Herbrand equation with respect to $|.|_{\mathit{term\text{-}size}}$. To see this, let θ be a grounding unifier of the Herbrand equation. Then $|\theta(\mathtt{C})|_{\mathit{term\text{-}size}} = |\mathtt{succ}(\theta(\mathtt{N})) * \mathtt{pow}(\theta(\mathtt{X}), \theta(\mathtt{N}))|_{\mathit{term\text{-}size}}$, hence: $|\theta(\mathtt{C})|_{\mathit{term\text{-}size}} = 1 + (1 + |\theta(\mathtt{N})|_{\mathit{term\text{-}size}}) + (1 + |\theta(\mathtt{X})|_{\mathit{term\text{-}size}} + |\theta(\mathtt{N})|_{\mathit{term\text{-}size}})$ Observe that the linear equation expresses the relative sizes of any ground instance of the variables $\mathtt{C}$, $\mathtt{X}$ and $\mathtt{N}$ that satisfies the syntactic equation.

To capture linear invariants between the arguments of predicates, it is necessary to lift the $\models$ ordering on linear systems to atoms paired with linear systems as follows: $\langle p(\boldsymbol{x}_1), E_1\rangle \models \langle p(\boldsymbol{x}_2), E_2\rangle$ iff $soln_{\boldsymbol{x}_1}(E_1) \subseteq soln_{\boldsymbol{x}_2}(E_2)$. Observe that two pairs $\langle p(\boldsymbol{x}_1), E_1\rangle$ and $\langle p(\boldsymbol{x}_2), E_2\rangle$ that differ syntactically may express the same invariants, that is, $\langle p(\boldsymbol{x}_1), E_1\rangle \models \langle p(\boldsymbol{x}_2), E_2\rangle \models \langle p(\boldsymbol{x}_1), E_1\rangle$ yet $E_1 \neq E_2$. To express invariants between argument positions it is thus necessary to construct sets of syntactically different but equivalence pairs. (This is more than an aesthetic predilection since this construction simplifies the way formal arguments are matched against actual arguments.) Formally, equivalence is defined by $\langle p(\boldsymbol{x}_1), E_1\rangle \equiv \langle p(\boldsymbol{x}_2), E_2\rangle$ iff $\langle p(\boldsymbol{x}_1), E_1\rangle \models \langle p(\boldsymbol{x}_2), E_2\rangle \models \langle p(\boldsymbol{x}_1), E_1\rangle$ which, in turn, induces a notion of equivalence class. To simultaneously record the invariants that hold on different predicates, the ordering is further extended to sets of equivalence classes to obtain a preorder. Specifically, given two sets of equivalence classes I_1 and I_2, the preorder $\models$ is defined $I_1 \models I_2$ iff for all $[\langle p(\boldsymbol{x}), E_1\rangle]_\equiv \in I_1$ there exists $[\langle p(\boldsymbol{x}), E_2\rangle]_\equiv \in I_2$ such that $\langle p(\boldsymbol{x}), E_1\rangle \models \langle p(\boldsymbol{x}), E_2\rangle$. Sets of equivalence classes provide a computation domain for the following operator that simulates T_P in such as fashion so as to discover argument-size relationships. The operator is denoted by T_c^{CLP} since it operates in the domain of linear constraints. Like before, it is defined in a clause-wise fashion:

$$T_c^{\text{CLP}}(I) = I \cup \left\{ [\langle p(\boldsymbol{x}), hull(F, F')\rangle]_\equiv \;\middle|\; \begin{array}{ll} c = p(\boldsymbol{x}) \leftarrow H, p_1(\boldsymbol{x}_1), \ldots p_n(\boldsymbol{x}_n) & \wedge \\ [\langle p_i(\boldsymbol{x}_i), E_i\rangle]_\equiv \in I \wedge [\langle p(\boldsymbol{x}), F\rangle]_\equiv \in I & \wedge \\ E = \alpha_{|.|}(H) \cup (\cup_{i=1}^n E_i) & \wedge \\ F' = project(var(c) \setminus var(\boldsymbol{x}), E) & \end{array} \right\}$$

This operator can be lifted to the level of a program $P = \{c_1, \ldots c_n\}$ by defining $T_P^{\text{CLP}}(I) = I_n$ where $I_0 = I$ and $I_i = T_{c_i}^{\text{CLP}}(I_{i-1})$. The computational domain is neither a complete lattice nor admits finite ascending chains. However, by adding a widening operator [8] a post-fixpoint can be finitely computed, that is, a set of equivalence classes I such that $T_P^{\text{CLP}}(I) \models I$. Such a post-fixpoint faithfully describes the lfp of the original program in the following sense: if $T_P^{\text{CLP}}(I) \models I$ and $p(\boldsymbol{t}) \in \text{lfp}(T_P)$ then there exists $[\langle p(\boldsymbol{x}), E\rangle]_\equiv \in I$ such that $|\boldsymbol{t}| \in soln_{\boldsymbol{x}}(E)$. The proof is not given since it can be constructed straightforwardly by adapting proofs that have been reported elsewhere [12]. $T_{c_i}^{\text{CLP}}$ provides a way to calculate a post-fixpoint in a bottom-up fashion by iterating and stabilising each strongly connected component (SCC) of the static call graph in turn. SCCs that contain a single, non-recursive clause can be evaluated exactly without a stability check.

6.2 Experimental results

For simplicity, an argument-size analyser was implemented in SICStus Prolog 3.8.5 which comes equipped with a built-in Simplex solver. The analyser was applied to a range of standard Prolog benchmarks varying in size between 100 and 10000+ LOC. Figure 1 presents the analysis times in seconds when the analyser is run on a 2.40GHz PC with 512 MB of RAM running Windows XP with all modules compiled to so-called compactcode (interpreted bytecode). The

benchmark	LOC	vars approx'ed ratio	vars approx'ed %	proj approx'ed ratio	proj approx'ed %	sparsity size	sparsity system	sparsity vars	time
gabriel	114	0/186	0.0	0/60	0.0	5.6	10.4	1.4	0.06
browse	137	0/294	0.0	0/79	0.0	6.5	11.9	1.4	0.06
ime_v2-2-1	181	21/888	2.3	8/132	6.0	11.9	21.3	1.6	0.70
kalah	284	0/533	0.0	0/133	0.0	7.3	12.4	1.4	0.14
mastermind	311	0/352	0.0	0/89	0.0	6.3	12.2	1.4	0.11
sdda	331	4/432	0.9	2/137	1.4	6.2	11.0	1.4	0.11
press	349	14/802	1.7	7/215	3.2	6.5	11.9	1.5	0.31
trs	368	7/1651	0.4	5/209	2.3	12.2	21.4	1.6	0.37
peep	371	11/665	1.6	6/163	3.6	7.7	12.5	1.6	0.34
qplan	424	0/380	0.0	0/104	0.0	7.9	14.7	1.4	0.09
ga	437	0/479	0.0	0/87	0.0	10.7	20.0	1.4	0.17
read	442	4/844	0.4	2/213	0.9	7.7	15.4	1.3	0.23
simple_analyzer	488	5/1183	0.4	3/287	1.0	8.6	15.0	1.4	0.44
ann	503	9/1089	0.8	3/268	1.1	7.7	12.9	1.5	0.39
nbody	562	0/684	0.0	0/147	0.0	9.2	15.9	1.3	0.13
ili	582	6/1789	0.3	3/504	0.5	7.8	13.6	1.3	0.64
asm	594	1/761	0.1	1/217	0.4	7.2	12.3	1.3	0.24
nand	603	29/1356	2.1	6/240	2.5	11.1	19.8	1.4	1.53
bryant	670	22/1381	1.5	4/252	1.5	14.1	26.3	1.3	1.38
sim_v5-2	986	14/2923	0.4	8/840	0.9	6.0	11.2	1.4	0.88
peval	993	36/2709	1.3	18/719	2.5	9.7	17.3	1.3	1.79
sim	1071	0/2412	0.0	0/394	0.0	12.0	20.1	1.3	0.61
rubik	1229	0/1062	0.0	0/276	0.0	5.7	9.4	1.5	0.20
chat	4698	105/7917	1.3	50/1581	3.1	9.7	19.1	1.5	4.58
pl2wam	4775	96/4078	2.3	34/1020	3.3	8.0	13.4	1.5	3.20
lptp	7419	213/12525	1.7	81/3624	2.2	8.2	15.2	1.4	9.97
aqua_c	15026	493/32340	1.5	188/6292	2.9	10.3	19.5	1.5	27.59

Fig. 1. Timing and precision results

leftmost column records the time to actually calculate the size invariants and write the results to an output file (little variance was observed between different runs of the analyser). This excludes the time to read, parse and normalise the input program and compute the SCCs (which is a small and varying fraction of the analysis time). These experiments were conducted using the classic widening [8] but delaying its application within an SCC until 2 complete iterations had been computed. Performance figures for an argument size analysis have been reported for the cTI termination inference tool [24]. cTI realises its argument size analysis with the Parma Polyhedra Library (PPL version 0.5 [1]) and timings of 0.26s, 0.17s, 3.89s, 3.99s and 2.12s are reported for read, ann, chat, lptp and pl2wam – the largest five benchmarks that we have in common and which were publicly available. These experiments were also performed on a 2.4GHz PC with 512 MB of RAM, albeit running Linux, with widening activated after one SCC iteration. Repeating our experiments with this widening tactic gives times of 0.11s, 0.27s, 3.98s, 6.12s, 1.94s for the same benchmarks. These timing results suggest that the domain operations reported in this paper are not as grossly inefficient as one might expect.

In order to assess to precision of the analysis, columns 3–6 of Figure 1 present statistics on the frequency with which extreme point elimination is required. Columns 5 and 5 give the ratio and percentages of the number of times the projection algorithm actually applies extreme point elimination and therefore (possibly) loses precision. The percentages are low (with the notable exception of ime_v2-2-1). How much precision is lost has been assessed in columns 4 and 5 which show how many variables remain to be projected out when the extreme point method takes over. Note that even at this stage, inequalities, that do not mention the variables that remain to be eliminated, are already in the projection space and are therefore exact. The ratio between the number of variables and projections approximated, indicates that typically 3 or less variables remain to be eliminated when the extreme point elimination is applied. Columns 7, 8 and 9 respectively report statistics on the way $\mathit{project}(Y, E)$ is called, namely, average $|var(E)|$, $|E|$ and $(\sum_{e \in E} |e|)/|E|$ where $|e|$ denotes the number variables of e with non-zero coefficients. These figures suggest that sparsity is the norm in argument-size analysis, which helps to explain the low number of calls to the extreme point algorithm. (Note that although the mean number of variables is low, one projection operation in aqua_c eliminates 60 out of 90 variables.) Interestingly, widening after one rather than two SCC iterations almost always reduces ratio of approximated projections and variables.

Finally, approximately 1% of the inequalities generated by Fourier-Motzkin projection contain very large, relatively prime coefficients (only observed for aqua_c). These inequalities often arise alongside a low coefficient inequality that almost exactly describes the same half-space. These large coefficient inequalities obfuscate the presentation of the results and slowdown the analysis with costly arbitrary-precision arithmetic. In the spirit of the weakly relational domains that use inequalities with coefficients of -1, 0 or 1 [7, 25], we discard any inequality which contains a coefficient whose absolute value exceeds a preset bound. The large coefficient issue has only been observed on very large benchmarks and understanding the conditions in which it arises will be a topic for future work.

7 Related Work

Huynh, Lassez and Lassez [14] observed that sparsity is a key issue in variable elimination and suggest applying Fourier-Motzkin on (small) sparse systems and their extreme point method for dense systems. However, the context of their work was originally output in constraint logic programming [16] where over-approximation is usually unacceptable. They therefore systematically enumerate all extreme points in a breadth-first manner. Curiously, they do not consider switching between different projection strategies depending on the density of the system which, as this paper shows, is a good strategy.

Lassez, Huynh and McAloon [20] catalogue different types of redundant inequalities which include so-called syntactic and quasi-syntactic redundancies (as discussed in Section 3.3). They identify five other classes of redundancies that reduce to syntactic and quasi-syntactic redundancies if all equalities are removed from the inequality system. For example, pairs of opposing inequalities such as $x - y \leq 5$ and $-x + y \leq -5$ can be merged into $x - y = 5$ and all occurrences of x in the remaining inequalities can be replaced by $y + 5$. Note that merging inequalities with opposing coefficients and constants does not find implicit equalities whose opposing inequalities are linear combinations of two or more inequalities. Implicit equalities can be readily detected with a Simplex solver and future work will assess whether the benefit of removing all such equalities justifies the cost of their detection.

Our implementation of Fourier-Motzkin can potentially be further refined by applying Kohler's rule [18]. Kohler distilled his observations on extreme vectors (mentioned in Section 4) into a cheap strategy to avoid generating redundant inequalities during Fourier-Motzkin elimination. The idea is to count the number of inequalities in the original system that feed into an inequality e produced in the n-th elimination step. The observation is that if the count of e exceeds $n+1$ then e is redundant. Kohler's rule has not been applied in our implementation because its correctness can, in general, be compromised when it is combined with other redundancy removal techniques [14].

8 Conclusion

This paper presented algorithms to approximate the projection and convex hull operations on the abstract domain of polyhedra, thereby providing an alternative to the classic approach based on the (potentially exponential) frame representation. Experimental results show that the sparsity of inequalities generated by program analyses allows most operations to be carried out exactly. Being able to approximate only when the size of the result becomes unmanageable is a distinct advantage over weakly relational domains which sacrifice precision up front.

Acknowledgements We thank Jacob Howe and Peter Linnington for discussions on polyhedra and Lunjin Lu and Jonathan Martin whose work [17, 23] motivated this study. This work was partly supported by EPSRC project EP/C015517.

References

1. R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Possibly Not Closed Convex Polyhedra and the Parma Polyhedra Library. In *Static Analysis Symposium*, volume 2477 of *LNCS*, pages 213–229. Springer-Verlag, 2002.
2. F. Benoy, A. King, and F. Mesnard. Computing Convex Hulls with a Linear Solver. *Theory and Practice of Logic Programming*, 5(1&2):259–271, 2005.
3. B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A Static Analyzer for Large Safety-Critical Software. In *Programming Language Design and Implementation*, pages 196–207. ACM Press, 2003.
4. K.-H. Borgwardt. The average number of pivot steps required by the simplex method is polynomial. *Zeitschrift für Operations Research*, 26:157–177, 1982.

5. A. Bossi, N. Cocco, and M. Fabris. Norms on Terms and their Use in Proving Universal Termination of a Logic Program. *TCS*, 124:297–328, 1994.
6. V. Chandru, C. Lassez, and J.-L. Lassez. Qualitative Theorem Proving in Linear Constraints. *Annals of Math. and Artificial Intelligence*, To appear.
7. R. Claris and J. Cortadella. The Octahedron Abstract Domain. In R. Giacobazzi, editor, *Static Analysis Symposium*, volume 3148 of *LNCS*, pages 312–327, 2004.
8. P. Cousot and N. Halbwachs. Automatic Discovery of Linear Constraints among Variables of a Program. In *POPL*, pages 84–97. ACM Press, 1978.
9. D. De Schreye and S. Decorte. Termination of Logic Programs: The Never-Ending Story. *The Journal of Logic Programming*, 19&20:199–260, 1994.
10. D. De Schreye and K. Verschaetse. Deriving Linear Size Relations for Logic Programs by Abstract Interpretation. *New Generat. Comput.*, 13(2):117–154, 1995.
11. R. J. Duffin. On Fourier's Analysis of Linear Inequality Systems. *Mathematical Programming Study*, 1:71–95, 1974.
12. R. Giacobazzi, S. K. Debray, and G. Levi. Generalized Semantics and Abstract Interpretation for Constraint Logic Programs. *J. Logic Program.*, 3(25), 1995.
13. N. Halbwachs, D. Merchat, and C. Parent-Vigouroux. Cartesian Factoring of Polyhedra in Linear Relation Analysis. In *Static Analysis Symposium*, volume 2694 of *LNCS*. Springer Verlag, June 2003.
14. T. Huynh, C. Lassez, and J.-L. Lassez. Practical Issues on the Projection of Polyhedral Sets. *Annals of Math. and Artificial Intelligence*, 6(4):295–315, 1992.
15. J.-L. Imbert. Fourier's Elimination: Which to Choose? In *First Workshop on Principles and Practice of Constraint Programming*, pages 117–129, 1993.
16. J. Jaffar, M. J. Maher, P. J. Stuckey, and R. H. C. Yap. Output in CLP(R). In *Fifth Generation Computer Systems*, volume 2, pages 987–995, Tokyo, 1992.
17. A. King and J. C. Martin. Control Generation by Program Transformation. *Fundamenta Informaticae*. To appear.
18. D. A. Kohler. Projections of Convex Polyhedral Sets. Operations Research Centre Report ORC 67-29, University of California, Berkeley, 1967.
19. J.-L. Lassez. Querying Constraints. In *Symposium on Principles of Database Systems*, pages 288–298. ACM Press, 1990.
20. J.-L. Lassez, T. Huynh, and K. McAloon. Simplification and Elimination of Redundant Linear Arithmetic Constraints. In F. Benhamou and A. Colmerauer, editors, *Constraint Logic Programming*, pages 73–87. The MIT Press, 1993.
21. H. Le Verge. A Note on Chernikova's algorithm. Technical Report 1662, Institut de Recherche en Informatique, Campus Universitaire de Beaulieu, France, 1992.
22. J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.
23. L. Lu and A. King. Determinacy Inference for Logic Programs. In *European Symposium on Programming*, volume 3444 of *LNCS*, pages 108–123. Springer, 2005.
24. F. Mesnard and R. Bagnara. cTI: a Constraint-Based Termination Inference Tool for ISO-Prolog. *Theory and Practice of Logic Programming*, 5(1&2):243–257, 2005.
25. A. Miné. The Octagon Abstract Domain. In *Eighth Working Conference on Reverse Engineering*, pages 310–319. IEEE Computer Society, 2001.
26. T. S. Motzkin. *Beiträge zur Theorie der Linearen Ungleichungen*. PhD thesis, Universität Zürich, 1936.
27. A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.
28. A. Simon and A. King. Convex Hull of Planar H-Polyhedra. *International Journal of Computer Mathematics*, 81(4):259–271, March 2004.
29. A. Simon, A. King, and J. M. Howe. Two Variables per Linear Inequality as an Abstract Domain. In *Proceedings of Logic-Based Program Development and Transformation*, volume 2664 of *LNCS*, pages 71–89. Springer-Verlag, 2002.