A Simplex-Based Extension of Fourier-Motzkin for Solving Linear Integer Arithmetic

François Bobot, Sylvain Conchon, Evelyne Contejean, Mohamed Iguernelala, Assia Mahboubi, Alain Mebsout, Guillaume Melquiond

A Simplex-Based Extension of Fourier-Motzkin for Solving Linear Integer Arithmetic*

François Bobot[1], Sylvain Conchon[1], Evelyne Contejean[1], Mohamed Iguernelala[1], Assia Mahboubi[2], Alain Mebsout[1], and Guillaume Melquiond[2]

[1] LRI, Université Paris Sud, CNRS, Orsay F-91405
[2] INRIA Saclay–Île-de-France, Orsay F-91893

Abstract. This paper describes a novel decision procedure for quantifier-free linear integer arithmetic. Standard techniques usually relax the initial problem to the rational domain and then proceed either by projection (*e.g. Omega-Test*) or by branching/cutting methods (*branch-and-bound*, *branch-and-cut*, *Gomory cuts*). Our approach tries to bridge the gap between the two techniques: it interleaves an exhaustive search for a model with bounds inference. These bounds are computed provided an oracle capable of finding constant positive linear combinations of affine forms. We also show how to design an efficient oracle based on the Simplex procedure. Our algorithm is proved sound, complete, and terminating and is implemented in the ALT-ERGO theorem prover. Experimental results are promising and show that our approach is competitive with state-of-the-art SMT solvers.

1 Introduction

Linear arithmetic is ubiquitous in many domains ranging from software and hardware verification, linear programming, compiler optimization to planning and scheduling. Decision procedures for the quantifier-free linear fragment over integers (QF-LIA) are widely studied in Satisfiability Modulo Theories. Most of the procedures used by state-of-the-art SMT solvers are extensions of either the Simplex algorithm or the Fourier-Motzkin method. Both techniques first relax the initial problem to the rational domain and then proceed by branching/cutting methods or by projection.

Given a conjunction $\bigwedge_i \sum_j a_{i,j}\, x_j + b_i \leq 0$ of constraints over rationals, the Simplex algorithm [16] finds an instantiation of the variables x_j satisfying these constraints or a contradiction if they are unsatisfiable. Three well-known extensions of this method to decision procedures over integers are *branch-and-bound*, *Gomory's cutting-planes*, and *branch-and-cut* [16]. Intuitively, these extensions prune non-integer solutions from the search space until they find an integer assignment or a contradiction. The Simplex algorithm is exponential in the worst case but behaves rather well in practice. On the other hand, the complexity of

* Work financially supported by the French ANR project ANR-08-005 DeCert.

QF-LIA is NP-complete [16] and known algorithms are not as efficient in practice as the Simplex on the rational case.

By contrast, the idea behind the Fourier-Motzkin [16] algorithm is to perform successive variable eliminations in a breadth-first manner generating additional constraints with fewer variables. The original system is satisfiable in the rationals if a fixpoint is reached without deriving a trivially inconsistent inequality $c \leq 0$ where c is a positive rational. In the opposite case we conclude that the system is unsatisfiable. The *Omega-Test* [15] extends this algorithm to a decision procedure over integers by performing additional projection-based checks when the constraints are satisfiable in the rationals. These methods do not scale in practice because they introduce a (double) exponential number of inequalities, which saturates the memory.

In this paper, we present a novel decision procedure for conjunctions of quantifier-free linear integer arithmetic constraints. Our approach is not an instance of any of the above techniques. Roughly speaking, it interleaves an exhaustive search for a model with bounds inference. New bounds are computed by solving auxiliary linear optimization problems using the Simplex algorithm. Intuitively, each auxiliary problem simulates a run of the Fourier-Motzkin algorithm that would eliminate all the variables at once. In order to facilitate the reading of this article, we summarize hereafter the main ideas of our contribution.

After recalling some useful notations and mathematical background, we characterize in Section 2 when the solution set described by a conjunction of constraints can be effectively bounded along some direction. If there is no such bound, we prove that the solution set contains infinitely many integer solutions.

This characterization is based on finding constant positive linear combinations of affine forms. In Section 3, we first show that Fourier-Motzkin is a suitable algorithm to compute such combinations. Then, we explain how to cast this problem into a linear optimization problem, which can hence be solved by an efficient Simplex-based algorithm.

In Section 4, we show how to build a decision procedure for QF-LIA. The procedure uses the algorithms of Section 3 to find bounds on the solution set. If there are none, then the procedure stops since there are infinitely many integer solutions. Otherwise it performs a case-split analysis along the bounded direction and calls itself recursively to solve the simpler subproblems.

We have implemented our framework in the ALT-ERGO theorem prover [4]. In Section 5, we measure its performances on a subset of the QF-LIA benchmark and compare its performances with some state-of-the-art SMT solvers. Section 6 presents future and related works.

2 Preliminary Results

2.1 Background and notations

In all what follows, if $m, n \in \mathbb{N}$, then $[|m, n|]$ denotes the integer interval bounded by m and n. We denote matrices by upper case letters like A and column vectors

by lower case letters like x. We denote A^t the transpose of the matrix A, Ax denotes the matrix product of the matrix A by the vector x and $\ker A$ the set of vectors such that $Ax = 0$. If A is a $m \times n$ matrix, $a_{i,j}$ denotes the element of A at position (i, j), a_i denotes the i-th row vector of A (of size n), and A_j the j-th column vector of A (of size m). If x is a vector, x_i denotes its i-th coordinate. If x and y are n-vectors of the same ordered vector space, $x \geq y$ denotes the conjunction of constraints $\forall i \in [|1, n|], x_i \geq y_i$, with similar notations for $\leq$ and the associated strict orders. For instance, if x is a vector, $x > 0$ denotes the conjunction of constraints $\forall i \in [|1, n|], x_i > 0$. We equip $\mathbb{Q}^n$ with the usual scalar product associated with its canonical basis. We measure distances using the *supremum* norm $\|\cdot\|_\infty$. $B_\infty(x, r)$ denotes the closed ball centered in x and of radius r for that norm.

We recall that affine maps $\psi : \mathbb{Q}^n \to \mathbb{Q}^m$ are the maps of shape $\psi = \phi + t_c$, with $\phi : \mathbb{Q}^n \to \mathbb{Q}^m$ a linear map and t_c the translation of direction $c \in \mathbb{Q}^m$. An affine map $\psi : \mathbb{Q}^n \to \mathbb{Q}$ is called an affine form on $\mathbb{Q}^n$. For instance, a constant map $\psi_c : \mathbb{Q}^n \to \mathbb{Q}^m$, with value c, is an affine map since it is the sum of the zero linear map and of the translation of direction $c \in \mathbb{Q}^m$.

Definition 1 (Positive linear combination of affine forms). *Let $(\psi_i)_{i\in[|1,k|]}$ be a family of affine forms on $\mathbb{Q}^n$. An affine form ψ on $\mathbb{Q}^n$ is a* positive linear combination *of the $(\psi_i)_{i\in[|1,k|]}$ if there exists $(\lambda_i)_{i\in[|1,k|]}$ a family of nonnegative scalars such that*

$$\psi = \sum_{i=1}^{k} \lambda_i \psi_i \quad \text{and} \quad \sum_{i=1}^{k} \lambda_i > 0$$

We recall the original formulation of Farkas lemma [16, 9] on rationals:

Theorem 1 (Farkas' lemma). *Given a matrix $A \in \mathbb{Q}^{m\times n}$, and c a vector in $\mathbb{Q}^n$, then*

$$\exists x \in \mathbb{Q}^m, x \geq 0 \ \wedge \ Ax = c \quad \Leftrightarrow \quad \forall y \in \mathbb{Q}^n, y^t A \geq 0 \Rightarrow y^t c \geq 0$$

In the sequel, we use the following equivalent formulation:

Theorem 2 (Theorem of alternatives). *Let A be a matrix in $\mathbb{Q}^{m\times n}$ and b a vector in $\mathbb{Q}^m$. The system $Ax + b \leq 0$ has no solution if and only if there exists $\lambda \in \mathbb{Q}^m$ such that $\lambda \geq 0$ and $A^t\lambda = 0$ and $b^t\lambda > 0$.*

2.2 Convex polytopes with an infinite number of integer points

We consider a closed convex subset $K \subset \mathbb{Q}^n$ defined by a linear system of constraints:

$$K := \{x \in \mathbb{Q}^n \mid Ax + b \leq 0\}$$

where $A \in \mathbb{Q}^{m\times n}$, $b \in \mathbb{Q}^m$. By definition, K is the convex polytope of the (rational) solutions of the linear system $Ax + b \leq 0$. We want to determine

whether $K \cap \mathbb{Z}^n$ is empty or not, or in other words, whether the system $Ax+b \leq 0$ has integer solutions. For $i \in [|1, m|]$, we denote by L_i the following affine forms:

$$L_i : \left| \begin{array}{ccc} \mathbb{Q}^n & \longrightarrow & \mathbb{Q} \\ x & \longmapsto & (Ax+b)_i \end{array} \right.$$

Theorem 3. *If there is no constant positive linear combination of the linear forms (L_i), then for all $R \in \mathbb{Q}^+$, K contains a ball $B_\infty(w, R)$ with $w \in \mathbb{Q}^n$.*

Proof. Let $R \in \mathbb{Q}^+$. We define $\gamma \in \mathbb{Q}^m$ such that for every $i \in [|1, m|]$:

$$\gamma_i := R\|a_i\|_1 = R\sum_j |a_{i,j}|$$

and we consider the convex $K' := \{x \in \mathbb{Q}^n \mid Ax + \gamma + b \leq 0\}$. Suppose for contradiction that K' is empty. Hence by Theorem 2, there exists $\lambda \in \mathbb{Q}^m$ such that $A^t\lambda = 0$. So, $\lambda^t(Ax + b) = \sum_i \lambda_i L_i$ is constant, which contradicts the hypothesis.

Therefore K' is not empty and contains a vector w such that $Aw+b+\gamma \leq 0$. We now prove that $B_\infty(w, R) \subseteq K$. Let u be a vector such that $\|u\|_\infty \leq R$. By triangular inequality we have

$$\forall i \in [|1, \ldots, m|] \quad (Au)_i \leq |(Au)_i| \leq \|a_i\|_1\|u\|_\infty \leq R\|a_i\|_1 = \gamma_i$$

hence

$$A(w + u) + b = Aw + b + Au \leq Aw + b + \gamma \leq 0$$

which proves that $w + u$ belongs to the convex K. □

Corollary 1. *If there is no constant positive linear combination of the (L_i) then $K \cap \mathbb{Z}^n$ contains infinitely many points, for $n > 0$.*

Proof. For any $N \in \mathbb{N}$ and any $x \in \mathbb{Q}^n$, the ball $B_\infty(x, N)$ contains at least $(2N)^n$ points with integer coordinates. □

Lemma 1. *If $\sum_i \lambda_i L_i$ is a positive linear combination of the (L_i) equal to a constant c, then*

- *if c is positive, then K should be empty;*
- *otherwise for every k such that $\lambda_k \neq 0$, and for any $x \in K$, $L_k(x)$ is bounded by $\frac{c}{\lambda_k} \leq L_k(x) \leq 0$*

Proof. For any $x \in K$, we have $\lambda_i L_i(x) \leq 0$ by definition of K and non-negativeness of λ_i. Hence $\sum_i \lambda_i L_i(x) \leq 0$, which concludes the first case.

Since $\sum_i \lambda_i L_i = c$, then for k such that $\lambda_k \neq 0$ and for any $x \in K$, we have $c - \lambda_k L_k(x) = \sum_{i \neq k} \lambda_i L_i(x) \leq 0$, which concludes the second case. □

Note that, if the constant c is zero, then all the inequalities $L_k \leq 0$ associated to a nonzero λ_k are in fact equalities.

2.3 Intersection with an affine subspace

In addition to K, we now consider another convex $K' \subset \mathbb{Q}^n$ defined by ℓ equations:

$$K' := \{x \in \mathbb{Q}^n \mid A'x + b' = 0\}$$

where $A' \in \mathbb{Z}^{\ell \times n}$, $b' \in \mathbb{Z}^\ell$, and study the intersection $K \cap K' \cap \mathbb{Z}^n$. We prove a sufficient condition for this intersection to contain an infinite number of points when $K \cap \mathbb{Z}^n$ is known to be infinite.

Let $(e_1, \ldots, e_n)$ be the canonical basis of $\mathbb{Q}^n$. We suppose that there exists $i_1, \ldots, i_j$ such that K is invariant by any translation of direction e_{i_k} for $k \in [|1, j|]$. Hence if we pose $E := \langle e_{i_1}, \ldots, e_{i_j} \rangle$ the vector space generated by these vectors, K is invariant by any translation of direction $e \in E$. We denote $\pi : \mathbb{Q}^n \to \mathbb{Q}^{n-j}$ the orthogonal projection along E (on the orthogonal complement $E^\perp$ of E). Note that since we consider vectors as column matrices of their coordinates on the canonical basis, computing the projection $\pi(x)$ of a vector x boils down to annihilating the coordinates $i_1, \ldots, i_j$ of x.

Theorem 4. *Assume that there are no constant positive linear combinations of the (L_i) and that $K' \cap \mathbb{Z}^n$ contains at least one point. Then if $\pi(K) \subseteq \pi(K')$, $K \cap K' \cap \mathbb{Z}^n$ contains infinitely many points.*

Proof. Let us first calculate the Smith normal form of matrix A'. This gives U, D, and V, matrices over $\mathbb{Z}$ such that $UA'V = D$, U and V are square matrices invertible over $\mathbb{Z}$, and D is diagonal (but not necessarily square). Since U and V are invertible, we have

$$\begin{aligned}\{x \in \mathbb{Q}^n \mid A'x = 0\} &= \{x \in \mathbb{Q}^n \mid UA'V(V^{-1}x) = 0\} \\ &= \{Vx \in \mathbb{Q}^n \mid Dx = 0\}\end{aligned}$$

Similarly, and since V is invertible over $\mathbb{Z}$, we have

$$\begin{aligned}\{x \in \mathbb{Z}^n \mid A'x = 0\} &= \{Vx \mid x \in \mathbb{Z}^n \wedge Dx = 0\} \\ &= \{Vx \mid x \in \ker D \cap \mathbb{Z}^n\}\end{aligned}$$

By hypothesis, there exists x_0 a point of $K' \cap \mathbb{Z}^n$. For any point x of K', we have $A'(x - x_0) = (A'x + b') - (A'x_0 + b') = 0$. Therefore,

$$\begin{aligned}K' &= \{x_0 + Vx \mid x \in \ker D\} \\ K' \cap \mathbb{Z}^n &= \{x_0 + Vx \mid x \in \ker D \cap \mathbb{Z}^n\}\end{aligned}$$

Let $R = \max(1, \max_i(\sum_j |v_{i,j}|)) = \max(1, \max_i \|v_i\|_1)$ and N an arbitrary large integer. By Theorem 3, there exists a ball $B = B_\infty(w, R(N+1))$, of diameter $2R(N+1)$, contained in K. The projection $\pi(B)$ contains at least N^{n-j} points that are at least at distance $2R$ from each other and at least at distance R from its border. Each of these points has at least one antecedent by π in K' since $\pi(B) \subseteq \pi(K) \subseteq \pi(K')$. We call $\mathcal{T} \subset K'$ this set of points. The distance between

any two points of $\mathcal{T}$ is at least $2R$ since the coordinates of the vectors in $\mathcal{T}$ on $E^\perp$ coincide with the ones of their respective projections.

For any point $y = x_0 + Vx \in K'$, there is a point of $K' \cap \mathbb{Z}^n$ at most at distance R. Indeed, since D is diagonal and $x \in \ker D$, any vector having the same non-zero coordinates as x remains in $\ker D$. Hence truncating the coordinates of x gives a vector $\lfloor x \rceil \in \ker D \cap \mathbb{Z}^n$. By definition of R, the distance between y and $x_0 + V\lfloor x \rceil$ is at most R.

Hence for each $t \in \mathcal{T}$, there exists $u \in K' \cap \mathbb{Z}^n$ such that the distance between t and u is at most R. But the distance between $\pi(t)$ and $\pi(u)$ is also at most R since the projection only annihilates some coordinates. Therefore we obtain a family $\mathcal{U}$ of N^{n-j} distinct points of $K' \cap \mathbb{Z}^n$ that have a projection inside $\pi(B)$. For each $u \in \mathcal{U}$, $u = b + e$ where $b = \pi(u) \in \pi(B)$ and $e \in E$. Now since $\pi(B) \subset \pi(K)$ there exists $e' \in E$ such that $b + e' \in B \subset K$. Hence $u = b + e' + (e - e')$ belongs to K since K is invariant by the translation of direction $e' - e \in E$. Finally the N^{n-j} points of $\mathcal{U}$ are in $K \cap K' \cap \mathbb{Z}^n$, which concludes the proof when $j < n$.

In the degenerate case where $j = n$, K is either empty or the whole space. K cannot be empty since by theorem 2 this would imply the existence of constant positive linear combination of the (L_i), hence a contradiction. Now K cannot be the whole space if there is no zero combination of the (L_i) which again contradicts the present hypothesis. □

3 Constant Positive Linear Combinations of Affine Forms

In this section, we are interested in computing constant positive linear combinations of affine forms. More precisely, we intend to build an oracle which takes as input a set of affine forms (L_i) (or equivalently, a matrix A and a vector b) and meets the following specifications:

1. if there is no constant positive linear combination of the (L_i), it says so;
2. otherwise, it returns such a combination $\sum_i \lambda_i L_i$.

We first present a method based on the Fourier-Motzkin procedure. Then, we describe an efficient implementation based on the Simplex algorithm and prove its soundness, completeness, and termination.

3.1 The Fourier-Motzkin procedure

Let $K := \{x \in \mathbb{Q}^n \mid Ax + b \leq 0\}$ be a closed convex where $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$, and C the set of the affine forms $L_i : x \mapsto (Ax + b)_i$. Fourier-Motzkin can be seen as an algorithm that attempts to compute constant positive linear combinations $\sum \lambda_i L_i$ in order to decide whether K is empty or not. For that purpose, it eliminates iteratively all the variables from the set of affine forms. More precisely, the iteration k of the procedure consists in

1. choosing a variable x_k to eliminate,

2. partitioning the current set C_k of affine forms into a set $C^0_{x_k}$ not containing x_k and a set $C^+_{x_k}$ (resp. $C^-_{x_k}$) where x_k has positive (resp. negative) coefficients,
3. computing the set C_{k+1} of new affine forms:

$$C_{k+1} := C^0_{x_k} \ \cup \ \Pi(C^+_{x_k} \times C^-_{x_k})$$

where Π calculates a positive combination $L_{i,j} = \alpha_{i,j} L_i + \beta_{i,j} L_j$ not containing x_k for each $L_i \in C^+_{x_k}$ and $L_j \in C^-_{x_k}$.

Notice that if either $C^+_{x_k}$ or $C^-_{x_k}$ is empty, then Π returns an empty set. The iterative process terminates when all the variables are eliminated and it returns a (possibly empty) set C_f of constant affine forms. We know that K is empty if there exists $c \in C_f$ such that $c > 0$. Moreover, given a constant $c \in C_f$, it is easy to retrieve a positive linear combination $\sum_i \lambda_i L_i = c$. For that, we recursively unfold the definitional equalities $L_{i,j} = \alpha_{i,j} L_i + \beta_{i,j} L_j$ computed by Π.

Example 1. Consider the following set of affine forms:

$$C_1 : \left\{ \begin{array}{lll} L_1 = 2x + y, & L_2 = -2x + 3y - 5, & L_3 = x + z + 1, \\ L_4 = x + 5y + z, & L_5 = -x - 4y + 3, & L_6 = 3x - 2y + 2 \end{array} \right\}$$

Eliminating z from C_1 is immediate since it only appears positively:

$$C_2 : \left\{ \begin{array}{lll} L_1 = 2x + y, & L_2 = -2x + 3y - 5, & L_5 = -x - 4y + 3, \\ L_6 = 3x - 2y + 2 & & \end{array} \right\}$$

We eliminate the variable x and compute the set C_3 below using the combinations: $L_7 = L_1 + L_2$, $L_8 = L_1 + 2L_5$, $L_9 = 2L_6 + 3L_2$, $L_{10} = L_6 + 3L_5$

$$C_3 : \left\{ \begin{array}{lll} L_7 = 4y - 5, & L_8 = -7y + 6, & L_9 = 5y - 11, \\ L_{10} = -14y + 11 & & \end{array} \right\}$$

Finally, the variable y is in turn eliminated thanks to the following combinations: $L_{11} = 7L_7 + 4L_8$, $L_{12} = 7L_7 + 2L_{10}$, $L_{13} = 7L_9 + 5L_8$, $L_{14} = 14L_9 + 5L_{10}$

The iterative process terminates and returns the set

$$C_4 : \{\, L_{11} = -11, \qquad L_{12} = -13, \qquad L_{13} = -47 \qquad L_{14} = -99 \,\}$$

Moreover, unfolding the equalities introduced by Π yields

$$\left\{ \begin{array}{lllll} -11 & = & L_{11} & = & 7L_7 + 4L_8 \quad = \cdots = 11L_1 + 7L_2 + 8L_5 \\ -13 & = & L_{12} & = & 7L_7 + 2L_{10} \quad = \cdots = 7L_1 + 7L_2 + 6L_5 + 2L_6 \\ -47 & = & L_{13} & = & 7L_9 + 5L_8 \quad = \cdots = 5L_1 + 21L_2 + 10L_5 + 14L_6 \\ -99 & = & L_{14} & = & 14L_9 + 5L_{10} = \cdots = 42L_2 + 15L_5 + 33L_6 \end{array} \right.$$

A constant positive linear combination $c = \sum \lambda_i \, L_i$ can now be used in conjunction with Lemma 1 to refine the bounds on the initial set of affine forms. Since for any vector $x \in K$, and for any j, $L_j(x) \leq 0$, we have $c = \sum \lambda_i \, L_i(x) \leq \lambda_j L_j(x)$, and we obtain a lower bound $\frac{c}{\lambda_j}$ on L_j as soon as $\lambda_j \neq 0$.

Example 2. Using the linear combination $11L_1 + 7L_2 + 8L_5 = -11$, we can make the deductions $-1 \leq L_1$, $-\frac{11}{7} \leq L_2$ and $-\frac{11}{8} \leq L_3$ in the rationals. Furthermore, these deductions are refined as follows in the integers: $-1 \leq L_1$, $\lceil -\frac{11}{7} \rceil = -1 \leq L_2$ and $\lceil -\frac{11}{8} \rceil = -1 \leq L_3$.

3.2 Computing the linear combinations using a Simplex

While the Fourier-Motzkin algorithm can be used to compute all the relevant constant positive linear combinations of affine forms, it does not scale in practice. In the following, we describe an efficient Simplex-based alternative and show its soundness, completeness, and termination. As opposed to the Fourier-Motzkin algorithm, this new approach will only attempt to compute one particular constant positive linear combination.

Let $K := \{x \in \mathbb{Q}^n \mid Ax + b \leq 0\}$ be a closed convex where $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$, and C the set of the affine forms $L_i : x \mapsto (Ax + b)_i$ of the form $\sum_{j=1}^{n} a_{i,j}\, x_j + b_i$. Consider the combination $\sum \lambda_i\, L_i$ of the affine forms. This sum unfolds as follows:

$$\lambda_1\, (\sum_{j=1}^{n} a_{1,j}\, x_j + b_1) \quad + \quad \cdots \quad + \quad \lambda_m\, (\sum_{j=1}^{n} a_{m,j}\, x_j + b_m)$$

and factorizing the x_i gives:

$$x_1\, (\sum_{i=1}^{m} a_{i,1}\, \lambda_i) \quad + \quad \cdots \quad + \quad x_n\, (\sum_{i=1}^{m} a_{i,n}\, \lambda_i) \quad + \quad \sum_{i=1}^{m} b_i\, \lambda_i$$

Since we are only interested in computing constant positive linear combinations, we require that for every k, $\sum_{i=1}^{m} a_{i,k}\, \lambda_i = 0$, which eliminates the variable x_k. Moreover, we look for the combinations that maximize the value of $\sum_{i=1}^{m} b_i\, \lambda_i$, since this will improve efficiency, as described in Section 4.2. More precisely, we compute such a constant positive linear combination by solving the following problem in the rationals:

$$\begin{array}{ll} \mathit{maximize} & \sum_{i=1}^{m} b_i\, \lambda_i \\ \mathit{subject\ to} & A^t\lambda = 0 \quad \wedge \quad \sum_{i=1}^{m} \lambda_i > 0 \quad \wedge \quad \bigwedge_{i=1}^{m} \lambda_i \geq 0 \end{array}$$

This problem reminds of the dual Simplex input, but here we have equalities $A^t\lambda = 0$ instead of the usual inequalities and an extra constraint $\sum \lambda_i > 0$.

In order to solve the above problem, we first introduce a slack variable s and a positive parameter ε to transform the strict inequality $\sum_{i=1}^{m} \lambda_i > 0$ into $\sum_{i=1}^{m} \lambda_i - \varepsilon = s \ \wedge \ s \geq 0$, following Lemma 1 of [8]. Then we solve the system of equalities in $\mathbb{Q}$ modulo the constraints $\bigwedge_{i=1}^{m} \lambda_i \geq 0 \wedge s \geq 0$. This returns unsat if this system is inconsistent in $\mathbb{Q}$ modulo the non-negativeness constraints, or a

matrix of the form

$$\begin{pmatrix} & & \lambda_1 & \lambda_2 & \cdots & s \\ \lambda_{b_1} & \mapsto & c_{1.1} & c_{1.2} & \cdots & c_{1.m+1} \\ \lambda_{b_2} & \mapsto & c_{2.1} & c_{2.2} & \cdots & c_{2.m+1} \\ \vdots & & & & & \\ \lambda_{b_{n+1}} & \mapsto & c_{n+1.1} & c_{n+1.2} & \cdots & c_{n+1.m+1} \end{pmatrix}$$

Finally, we initialize the Simplex algorithm with this matrix and try to maximize the objective function. The Simplex returns either unsat if the given system has no solution, or unbound if the objective function has no upper bound, or a maximum m and a valuation ν for the vector λ.

If the Simplex algorithm returns unsat, then the oracle answers that there is no constant positive linear combination. If it returns unbound, the oracle just returns a positive constant. Otherwise, the Simplex algorithm necessarily returns a solution with a non-positive maximum for the objective function. Indeed, if the maximum were to be positive, one could multiply coordinate-wise any solution λ by a constant larger than 1 and obtain another solution with a larger objective value. The oracle then returns the corresponding linear combination. Note that as soon as the Simplex exploration discovers a positive value for the objective function, the answer will eventually be unbound so it can exit immediately.

3.3 Soundness, completeness, and termination

On top of the Simplex algorithm we only add some substitutions, so the termination of this oracle follows directly from the one of the Simplex algorithm.

Let us justify that the introduction of the parameter ε does affect neither the soundness nor the completeness of the oracle. Let us denote by $P_{>0}$ the original problem and by $P_{\geq\varepsilon}$ the problem we actually send to the Simplex algorithm. First, remember that for both problems, the answer is either unsat or unbound or a solution with a non-positive evaluation of the objective function: indeed, if ν is a solution, so is $\alpha\nu$ for any scalar α with the constraint $\alpha > 0$ for $P_{>0}$, and $\alpha > 1$ for $P_{\geq\varepsilon}$, and the value of the objective function is multiplied accordingly. Moreover, any solution of $P_{\geq\varepsilon}$ is obviously a solution of $P_{>0}$. Let us now proceed with the proof by case analysis.

1. If $P_{>0}$ is unsat, so is $P_{\geq\varepsilon}$ by inclusion of solutions.
2. If $P_{\geq\varepsilon}$ is unsat, let us assume by contradiction that $P_{>0}$ is not unsat, hence has a solution ν. Then $\frac{\varepsilon}{\sum \nu_i}\nu$ is a solution of $P_{\geq\varepsilon}$, a contradiction.
3. If $P_{\geq\varepsilon}$ is unbound, so is $P_{>0}$ by inclusion of solutions.
4. If $P_{>0}$ is unbound, we show that $P_{\geq\varepsilon}$ is also unbound. Let M be an arbitrary large value. By hypothesis on $P_{>0}$, there is a solution ν of $P_{>0}$ with an evaluation of the objective function greater than M. Then, if $\sum \nu_i \geq \varepsilon$, ν is a solution of $P_{\geq\varepsilon}$ with an evaluation of the objective function greater than M.

Otherwise, $\frac{\varepsilon}{\sum \nu_i}\nu$ is a solution of $P_{\geq\varepsilon}$, with an evaluation of the objective function greater than $\frac{\varepsilon}{\sum \nu_i}M$, hence greater that M since $\varepsilon > \sum \nu_i$.

5. If $P_{>0}$ (resp. $P_{\geq\varepsilon}$) has a solution with a non-positive evaluation of the objective function, so has $P_{\geq\varepsilon}$ (resp. $P_{>0}$), since the other cases are impossible, as shown above.

4 The Decision Procedure

Let us now build a decision procedure for QF-LIA based upon an oracle that follows the interface described at the beginning of Section 3 and the theorems presented in Section 2.

4.1 The algorithm

Let $A \in \mathbb{Q}^{m\times n}$ and $b \in \mathbb{Q}^m$. The procedure shall decide whether the system $Ax + b \leq 0$ has a solution $x \in \mathbb{Z}^n$. Let $L = (L_i)_{i\in[|1,m|]}$ be the associated family of affine forms. Figure 1 sketches the algorithm. It takes as input both the system L of inequalities and an additional argument *Eq* representing affine relations between variables, *e.g.* a set of equalities, or a substitution, or an echelon matrix, etc. This last argument is initially empty. The result of the decision procedure is stored in the *sols* variable. It is a finite set of integer solutions, possibly empty, or an indeterminate infinite set of integer solutions.

The *check* functions at lines 6 and 10 compute the integer solutions of a system of equations, but the special shape of the systems they deal with allows important optimizations that will be detailed below.

The algorithm is recursive. Recursive calls are performed on smaller and smaller systems L until complete resolution. Branching is caused by the loop on line 15. The results are merged along the various branches at lines 6 and 10. One can also consider that there are implicit statements $sols \leftarrow sols \cup \emptyset$ at lines 4, 13, and 17. Notice that the algorithm performs computations only when going from the root to the leaves of the call tree. For the sake of clarity, we have described a simple version of the algorithm. An actual implementation would likely be more complicated. For instance, it would exit as soon as a branch finds an infinity of solutions or even a single solution if one is interested only in satisfiability. It could also use *splitting on demand* [3] at line 15.

From lines 3 to 7, the algorithm deals with degenerate systems that contain no inequalities or only trivial inequalities. It then calls the oracle on L. If it answers that no suitable combination of the affine forms exists, Corollary 1 can be applied. There are infinitely many solutions with integer coordinates, assuming *Eq* imposes no restriction (hence the call to $check_\infty$). The decision procedure is done in this branch.

Otherwise, the oracle returns a constant positive linear combination $\sum_i \lambda_i L_i$ equal to c. In that case, Lemma 1 can be applied. If c is positive, the procedure is done too: the system has no solution.

```
1  global sols ← ∅
2  procedure lia(L = (L_i), Eq)
3      remove trivial inequalities c ≤ 0 with c constant from L
4      if some c was positive then return
5      if L = ∅ then
6          sols ← sols ∪ check_1(Eq)
7          return
8      call oracle(L)
9      if there is no constant positive linear combination then
10         sols ← sols ∪ check_∞(Eq)
11         return
12     let ∑ λ_i L_i = c the constant positive linear combination found by the oracle
13     if c > 0 then return
14     choose k such that λ_k ≠ 0, and μ > 0 such that μL_k has integer coefficients only
15     for all v from ⌈μ c/λ_k⌉ to 0 do
16         create a substitution σ from μL_k(x_1, ..., x_n) = v
17         if there is no possible substitution then continue to next iteration
18         remove L_k from L
19         apply σ to L
20         call lia(L, Eq ∪ {σ})
21     return
```

Fig. 1. Algorithm for the decision procedure

Otherwise, we have $\frac{c}{\lambda_k} \leq L_k \leq 0$ for all k such that $\lambda_k \neq 0$. The decision procedure chooses a value for k. Since the coefficients of μL_k are in $\mathbb{Z}$, for any point $x \in K \cap \mathbb{Z}^n$, $\mu L_k(x)$ is an integer between $\mu\frac{c}{\lambda_k}$ and 0. For each integer $v \in [\mu\frac{c}{\lambda_k}, 0]$, the decision procedure considers the equality $\mu L_k = v$ from which it infers a substitution if possible, applies the result to all the other affine forms $(L_i)_{i\neq k}$ and removes L_k from the system while updating Eq with the substitution. The decision procedure is then called recursively. If no solution is found after a complete exploration of all the possible integer values in $[\mu\frac{c}{\lambda_k}, 0]$, then the procedure returns at line 21 without updating *sols*. This is what happens if L has rational solutions but no integer solutions.

Note that, if the constant c is zero, then several equalities might appear at once. An optimized procedure should therefore compute a substitution taking all of them into account, rather than one after the other, as is done in Figure 1.

We now give more details on the computations performed at the leaves of the call tree by the *check* functions. The choice of these functions depends on the substitution scheme at line 16. We describe here two possible scenarios.

Integer substitution with slack variables. Let us first consider the case where the substitution introduces slack variables. The variables $x_1, \ldots, x_n$ of L_k are expressed as affine combinations of new variables $x_{n+1}, \ldots, x_{n+\ell}$, such that the integer solutions of $L_k(x_1, \ldots, x_n) = v$ are completely parameterized by these new variables. Removing L_k from L and applying the substitution produces a

system equisatisfiable to L and solutions to the original system can be trivially computed thanks to Eq.

A way to obtain the substitution is the Generalized GCD test [2] with the approach given by Pugh in the Omega-Test [15]. Note that the substitution may have only one solution, *e.g.* $2x = 6$. The substitution may also not exist, *e.g.* $5x = 2$, in which case, the exit case described line 17 applies.

In that case, functions *check* are implemented as follows. For $check_1$, the solutions are constrained purely by Eq. More precisely, the set of integer solutions is parameterized by the set of variables that are never the target of substitution in Eq. Moreover, the Eq system has been built only from adding successively (cf. line 20) new substitutions not featuring the variables previously substituted (cf. line 19), so it is never inconsistent. Therefore, only two situations are possible when Eq is passed to $check_1$: either Eq involves all the variables of the system, in which case there is exactly one solution, or it does not (some variables have not been substituted) and there are infinitely many integer solutions. In this implementation Eq is always a system with integer coefficients, and function $check_\infty$ is trivial for integer substitutions. Indeed, it just returns an indeterminate infinite set, whatever the value of Eq, since some variables have to be unsubstituted at that point.

In some context, *e.g.* an SMT solver, one might need more information than what this indeterminate set seems to carry. If one needs some explicit witnesses, the proof of Theorem 3 explains how to effectively compute them from L. Witnesses for the original system can then be deduced from Eq. If one needs to know which equalities are implied by the system, then Eq describes them entirely. Indeed, the decision procedure will not exit at line 11 if some constant positive linear combination still exists in L.

Gaussian elimination Let us now consider the case of a substitution performed by a simple Gaussian elimination on rational numbers; it does not introduce any slack variables but the coefficients involved in the substitutions are rationals, possibly non integers. Function $check_1$ now has to test whether the set of equalities Eq admits some solution and to return them. In this scenario, there can be either zero solution, or one, or an infinite number of them. The implementation of function $check_\infty$ can in that case take benefit of Theorem 4. The hypotheses of the theorem are actually verified thanks to the Gaussian elimination, and the vector space E of Theorem 4 is generated by the vectors of the canonical basis associated with the variables already substituted. Since these variables have been eliminated from L, the set of solutions of L is obviously invariant by translation along these coordinates. By construction of Eq the hypothesis of inclusion of the respective projections also holds. Therefore if the system of equalities Eq admits at least one integer solution, then there are an infinite number of solutions for the problem considered in the current branch. Otherwise there is no solution for this branch.

Note that, whichever of these substitution schemes is used, the solver for linear integer arithmetic embedded in the *check* functions has to deal with equations only and is therefore simple.

Producing explanations. An important feature when developing a decision procedure for SMT is to provide the most precise explanations that improve the backtrack level when branching. In our setting, the explaination of each inconsistency or lower bound $\frac{c}{\lambda_k}$ inferred by the oracle is the explanations of the inequalities $L_i \leq 0$ such that $\lambda_i \neq 0$. The explanations of the inequalities that have not participated in the inference process are thus discarded.

4.2 Soundness, completeness, and termination

Termination is obvious, assuming the oracle is itself terminating. Indeed, at each recursive call, one affine form at least is removed from the system. Note that it has to be effectively removed from the system; otherwise the oracle may just return the linear combination that bounds this form again, hence causing the procedure to enter an infinite recursion.

Soundness depends on the completeness of the oracle: if the oracle does not find any constant positive linear combination, there should be none. Theorems of Section 2.2 then cover all the possible cases. Completeness of the decision procedure comes from termination and soundness.

While the oracle can return any constant positive linear combination, for efficiency reasons, it should strive to find a positive constant if possible, and zero if not. Indeed, this ensures that the algorithm will not branch too early.

5 Experimental Results

We implemented the decision procedure with the Simplex-based oracle in a modified version[3] of ALT-ERGO [4]. Equalities are handled using a rewriting system that relies on substitutions with integer slack variables [14]. Inequalities are added to a dictionary associating affine forms with integer interval domains. The case-split analysis is implemented as a recursive function with non-chronological backtracking and uses a heuristic that privileges affine forms with smaller intervals. In the current implementation, we do not use a traditional Simplex to cut down the search space.

In this section, we benchmark our implementation and compare its performances with some leading state-of-the-art SMT solvers including MATHSAT5 v5.1.3 [11], Z3 v3.2 [5] and YICES2 v2.0-prototype [6]. We could not include the MISTRAL solver of [7] because it was not possible to obtain it. The test suite contains 1070 instances taken from the QF-LIA category of SMT-LIB[4]. This includes the following families:

- CAV-2009: randomly-generated instances used in [7]. Most of them are satisfiable. They are reported very hard for modern SMT solvers,
- SLACKS: reformulation of CAV-2009 instances used in [12] that introduces slack variables to bound all variables,

[3] A prototype is available at `http://alt-ergo.lri.fr/ijcar2012/`
[4] The SMT-LIB library: `http://www.smtlib.org`

– CUT-LEMMAS: crafted instances encoding the validity of cutting planes in $\mathbb{Z}$,
– PRIME-CONE: crafted instances used in [12] that encode a tight n-dimensional cone around the point whose coordinates are the first n prime numbers.
– PIDGEONS (*sic*): crafted instances encoding the pigeonhole principle. They are reported hard for any solver not using cutting planes [12],
– PB2010: industrial instances coming from the PB competition (2010),
– MIPLIB2003: instances generated from some optimization problems in [1].

We have selected these families because they are known to be well-suited for stressing the integer-reasoning part of solvers. Moreover, contrarily to some other families, they do not require solvers to be especially efficient for other tasks: preprocessing, *if-then-else* handling, SAT solving, theory propagation. In fact, a large part of the QF-LIA benchmark, *e.g.* NEC-SMT, does not even require fast LIA solvers [7].

All measures were obtained on a 64-bit machine with a quad-core Intel Xeon processor at 3.2 GHz and 24 GB of memory. Provers were given a time limit of 600 seconds and a memory limit of 2 GB for each test. The results of our experiments are reported in Figure 2. The first two columns show the families we considered and the number of their instances. For each prover, we report both the number of solved instances within the required time for every family and the time needed for solving them (not counting timeouts). The last rows summarize the total number of solved instances and the accumulated time for each prover.

SMT SOLVERS		ALT-ERGO		MATHSAT5		MATHSAT5+CFP		YICES 2		Z3	
families	**#inst.**	**solved**	**time**	**solved**	**time**	**solved**	**time**	**solved**	**time**	**solved**	**time**
CAV-2009	591	590	**253**	588	4857	589	4544	386	11664	590	5195
SLACKS	233	233	**67**	166	3551	155	6545	142	6102	187	9897
CUT-LEMMAS	93	93	**216**	62	3424	59	2775	92	1892	67	3247
PRIME-CONE	37	37	**0.4**	37	1	37	2.2	37	2.3	37	14
PIDGEONS	19	19	2	19	0.16	19	0.16	19	**0.01**	19	0.28
PB2010	81	23	390	38	743	34	1540	25	8.3	64	1831
MIPLIB2003	16	2	34.7	12	432	12	501	11	145.4	12	**241**
total	1070	997	**963.1**	922	13008	905	15907	712	19814	976	20425
total QF-LIA[5]	5882	4410	68003	5597	**47635**	5524	50481	3220	71324	5597	54503

Fig. 2. Experimental results. Underlined values are for tools that have proved the most instances. Bolded results are for tools that have proved both the most instances and the fastest.

Although the first two families were reported very hard for modern SMT solvers, our approach only requires 320 seconds to solve almost all the instances. Thus, it significantly outperforms the other solvers' approaches. This observation

[5] The time limit is 180 seconds for the tests of the complete benchmark.

also applies for the third and the fourth families. From the results of the sixth and the seventh families, we notice that our technique does not perform well on large difference-logic-like problems compared to MATHSAT5 and Z3's. We think this is partly due to our naive implementation of the Simplex algorithm which computes on dense matrices while sparse matrices would be better suited for these problems. We plan to implement advanced techniques in the very near future such as the revised Simplex method [16] to overcome this issue.

The last row of Table 2 shows the results for the whole QF-LIA benchmark. There are two reasons for the poor results. First, ALT-ERGO has yet to be tuned for parts other than the LIA solver. Second, some families, *e.g.* BOFILL, contain large intervals that need splitting, and the decision procedure does not deal efficiently with them. This will possibly require a combination of our approach with other established techniques for integers.

6 Conclusion and Future Works

We have presented a new decision procedure for quantifier-free linear integer arithmetic that combines a model search mechanism with bounds inference. These bounds are discovered thanks to a Simplex-based oracle that computes constant positive linear combinations of affine forms. We proved the soundness, the completeness and the termination of our method and implemented it in the ALT-ERGO SMT solver.

Designing efficient decision procedures for QF-LIA has been an active research topic in the SMT community over the last decade. An efficient integration of the Simplex algorithm in the DPLL(T) framework has been proposed in [8]. This integration rests on a preprocessing step that enables fast backtracking and efficient theory propagation. The contribution of [7] is seen as a generalization of *branch-and-bound*. Using the notion of *the defining constraints* of a vertex, it derives additional inequalities that prune higher dimensional subspaces not containing integer solutions. In our setting, the Simplex algorithm is instead used on auxiliary problems to refine the search space by bounds inference.

The approach described in [10] focuses on combining several existing techniques using heuristics and *layering* to take advantage of each of them. We believe that the ideas we described in this paper can naturally be used to enhance this combination approach. Yet another different contribution described in [12] consists in extending the inference rules of the CDCL procedure with linear arithmetic reasoning. This tight integration naturally takes advantage of the good CDCL properties: model search, dynamic variable reordering, propagation, conflicts explanation, and backjumping. The extension of our framework with an efficient conflict learning mechanism as done in [12] or in [13] for the rationals would greatly improve our decision procedure.

As reflected by our contribution, the Simplex we use does not directly work on the initial problem nor on its dual. Therefore, fast incrementality and backtracking techniques developed for Simplex-based approaches are not suitable for our setting. To alleviate this issue, we have used memoization techniques to reuse

previously computed results at the expense of a larger memory footprint. In the near future, we plan to better integrate with DPLL(T) by extending our method with a conflict resolution technique, a cleverer case-split analysis, and an efficient theory propagation. We also believe that a combination, *à la* MATHSAT, of state-of-the-art techniques with ours would be beneficial. Furthermore, the use of advanced data-structures and algorithms such as sparse matrices and the revised Simplex would greatly enhance our implementation.

References

1. T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):361–372, 2006.
2. U. Banerjee. *Dependence Analysis for Supercomputing.* Kluwer Academic Publishers, Norwell, MA, USA, 1988.
3. C. Barrett, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Splitting on demand in SAT modulo theories. In *LPAR '06*, volume 4246 of *LNCS*, pages 512–526. Springer-Verlag, November 2006. Phnom Penh, Cambodia.
4. F. Bobot, S. Conchon, E. Contejean, M. Iguernelala, S. Lescuyer, and A. Mebsout. The Alt-Ergo Automated Theorem Prover. `http://alt-ergo.lri.fr`.
5. L. de Moura and N. Bjørner. Z3, an efficient SMT solver. `http://research.microsoft.com/projects/z3`.
6. L. de Moura and B. Dutertre. Yices: An SMT Solver. `http://yices.csl.sri.com`.
7. I. Dillig, T. Dillig, and A. Aiken. Cuts from proofs: A complete and practical technique for solving linear inequalities over integers. In *CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, volume 5643 of *LNCS*, pages 233–247. Springer, 2009.
8. B. Dutertre and L. de Moura. A fast linear-arithmetic solver for DPLL(T). In *CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *LNCS*, pages 81–94. Springer, 2006.
9. G. Farkas. Über die theorie der einfachen ungleichungen. *Journal für die Reine und Angewandte Mathematik*, 124:1–27, 1902.
10. A. Griggio. A practical approach to satisability modulo linear integer arithmetic. *Journal on Satisfiability, Boolean Modeling and Computation*, 8:1–27, 2012.
11. A. Griggio, B. Schaafsma, A. Cimatti, and R. Sebastiani. MathSAT 5: An SMT Solver for Formal Verification. `http://mathsat.fbk.eu//`.
12. D. Jovanovic and L. de Moura. Cutting to the chase solving linear integer arithmetic. In *CADE-23, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, volume 6803 of *LNCS*, pages 338–353. Springer, 2011.
13. K. Korovin and A. Voronkov. Solving systems of linear inequalities by bound propagation. In *CADE-23, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, volume 6803 of *LNCS*, pages 369–383. Springer, 2011.
14. D. Kroening and O. Strichman. *Decision Procedures: An Algorithmic Point of View.* Springer Publishing Company, Incorporated, 1 edition, 2008.
15. W. Pugh. The Omega test: a fast and practical integer programming algorithm for dependence analysis. In *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, Supercomputing '91, pages 4–13, New York, NY, USA, 1991. ACM.
16. A. Schrijver. *Theory of linear and integer programming.* Wiley-Interscience series in discrete mathematics and optimization. John Wiley & sons, 1998.