Cutting the Mix

Jürgen Christ[1,2](✉) and Jochen Hoenicke[1]

[1] Department of Computer Science, University of Freiburg, Freiburg im Breisgau, Germany
{christj,hoenicke}@informatik.uni-freiburg.de

[2] Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Abstract. While linear arithmetic has been studied in the context of SMT individually for reals and integers, mixed linear arithmetic allowing comparisons between integer and real variables has not received much attention. For linear integer arithmetic, the cuts from proofs algorithm has proven to have superior performance on many benchmarks. In this paper we extend this algorithm to the mixed case where real and integer variables occur in the same linear constraint. Our algorithm allows for an easy integration into existing SMT solvers. Experimental evaluation of our prototype implementation inside the SMT solver SMTInterpol shows that this algorithm is successful on benchmarks that are hard for all existing solvers.

1 Introduction

The theory of linear arithmetic is fundamental in system modelling and verification. SMT solvers supported this theory from the beginning. In recent years, lots of work has been devoted to improve the support and performance for this theory [9,12,13,17]. Usually, two theories are supported for linear arithmetic: linear arithmetic over the rational/real numbers (LRA), and linear arithmetic over integers (LIA). While the first theory can be solved by the Simplex algorithm the latter needs more techniques to ensure that the solution has integer values. While each of the two theories has many applications for itself, some applications require both theories. An example for this is the verification of timed automata or hybrid systems where continuous variables are used for physical entities and integer variables for control. Also planning and scheduling problems require mixed integer and real arithmetic. While there exists some support for this theory, there is still room for improvement.

Boosting performance on satisfiability modulo mixed linear arithmetic broadens the applicability of SMT solvers. For linear integer arithmetic, the cuts from proofs algorithm [7] greatly improved the state of the art. On SMTLIB benchmarks with many real but few or no integer solutions, the algorithm significantly

This work is supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR14 AVACS).

D. Kroening and C.S. Păsăreanu (Eds.): CAV 2015, Part II, LNCS 9207, pp. 37–52, 2015.
DOI: 10.1007/978-3-319-21668-3_3

outperforms traditional techniques like the Omega test [18] or cutting plane techniques based on Gomory cuts. Improving the performance on mixed linear arithmetic is a next logical step to increase applicability of modern SMT solvers.

In this paper, we lift the cuts from proof algorithm from linear arithmetic over the integers to mixed linear arithmetic. We give a new technique to derive cuts in mixed linear arithmetic based on a basis transformation of the constraint system. We experimentally compare our implementation with other SMT solvers that support mixed linear arithmetic. This evaluation shows that our new technique is able to solve problems that cannot be solved by existing techniques.

Related Work. Only very few publications address mixed linear arithmetic in the context of SMT. Berezin et al. [3] present an extension of the Omega test to mixed arithmetic. The decision procedure splits the variables into real-valued variables and integer-valued variables. Then, it uses Fourier-Motzkin elimination for the real-valued variables and the Omega-test to eliminate the integer variables. The Fourier-Motzkin elimination usually produces an exponential blow-up of the input problem. The technique is therefore more memory intensive than our technique.

Dutertre and de Moura [9] present a way to compute mixed Gomory cuts in the context of the state of the art Simplex-based theory solver for linear arithmetic. The derivation is based on the current assignment, a row in the Simplex tableau, and non-trivial reasoning. Since Gomory cuts are used as theory lemmas in the proof of unsatisfiability, they have to be justified. This is especially important for proof producing or interpolating solvers. The derivation of Gomory cuts is much more involved than the simple technique presented here based on extended branches.

A lot of research exists in the context of MILP solvers. These solvers use a variety of different cuts (see [14,19] for details). MILP solvers use floating-point arithmetic making them imprecise and unstable [16]. Our technique is designed for integration into SMT solvers that typically use arbitrary precision arithmetic to ensure soundness. In the evaluation section, we do not compare against MILP solvers since these can be unsound due to rounding problems.

There are several techniques for solving linear integer arithmetic [4,10,11], some of which can be extended to mixed arithmetic, although we are not aware of any publications. A variation of the algorithm from [10] is used in MathSAT and CVC4. This is similar to ours but use a diophantine equation solver to generate the branch instead of the Hermite normal form.

2 Notation and Preliminaries

The algorithms in this paper are used to solve a system of linear inequalities. We use $\boldsymbol{x}$ to denote a vector of variables $x_1, \ldots, x_n$. Given a matrix $A \in \mathbb{Q}^{m\times n}$ and a vector $\boldsymbol{b} \in \mathbb{Q}^m$ we solve the problem $A\boldsymbol{x} \leq \boldsymbol{b}$. Here $\leq$ on vectors is defined component-wise. Strict inequalities can be expressed by allowing infinitesimal numbers in $\boldsymbol{b}$. However, we ignore strict inequalities in this paper to keep the presentation simple.

The variables $x_1, \ldots, x_n$ can be (depending on the problem) real or integer valued, i. e., we are interested in integer or real solutions of the above system. It is easy to see that for every real solution there is also a rational solution, so it is enough to distinguish between solutions in $\mathbb{Q}$ and $\mathbb{Z}$. We note that $A\boldsymbol{x} \leq \boldsymbol{b}$ is equivalent to $\lambda A\boldsymbol{x} \leq \lambda\boldsymbol{b}$ for $\lambda > 0$. Therefore we can w.l.o.g. assume that A is an integer matrix.

In the case of mixed arithmetic, we distinguish between rational-valued variables $x_1, \ldots, x_{n_1}$ and integer-valued variables $x_{n_1+1}, \ldots, x_n$. The goal is to find a rational solution for the first n_1 variables and an integer solution for the second $n_2 = n - n_1$ variables. Thus, the vector $\boldsymbol{x}$ is a vector consisting of real-valued variables and integer-valued variables.

In this paper we need the notion of nonsingular and unimodular matrices and the Hermite normal form. A matrix $U \in \mathbb{Q}^{n\times n}$ is *nonsingular* if $\det(U) \neq 0$. A matrix $U \in \mathbb{Z}^{n\times n}$ is *unimodular* if $|det(U)| = 1$. We remind the reader that a matrix is nonsingular if and only if it has an inverse $U^{-1} \in \mathbb{Q}^{n\times n}$ with $UU^{-1} = U^{-1}U = Id$, where Id denote the identity matrix. Moreover, a matrix $U \in \mathbb{Z}^{n\times n}$ is unimodular if and only if it has an inverse U^{-1} that is also an integer matrix.

A matrix $H \in \mathbb{Q}^{n\times n}$ is in *Hermite normal form*[1] if (i) H is lower triangular, (ii) $h_{ii} > 0$ for $1 \leq i \leq n$, and (iii) $h_{ij} \leq 0$ and $|h_{ij}| < h_{ii}$ for $j < i \leq n$. For every nonsingular integer matrix A there is a unique unimodular matrix U and a unique matrix H in Hermite normal form with $H = AU$. This also holds for rational matrices: If a nonsingular $A \in \mathbb{Q}^{n\times n}$ is given we can multiply it with $\lambda > 0$ such that $\lambda A \in \mathbb{Z}^{n\times n}$. Then the unique H', U with $H' = \lambda AU$ results in a unique $H = (1/\lambda)H'$ with $H = AU$. Note that since U is unimodular, the matrix H is an integer matrix if and only if A is an integer matrix.

3 Solving Linear Integer Arithmetic and Mixed Linear Arithmetic

We first review the state of the art in satisfiability solving modulo the theory of linear integer arithmetic. Let $A \in \mathbb{Z}^{n\times m}$ and $\boldsymbol{b} \in \mathbb{Z}^m$. To find an integer solution of $A\boldsymbol{x} \leq \boldsymbol{b}$, SMT solvers first compute a rational solution of $A\boldsymbol{x} \leq \boldsymbol{b}$, which is called the LP relaxation. If the relaxation is unsatisfiable, the original formula is unsatisfiable, too. Otherwise let $\boldsymbol{x}_0 = (x_{01}, \ldots, x_{0n})$ be the values assigned to the variables in the solutions to the relaxation. If all x_{0i} are integral, the original formula is satisfiable. Otherwise the relaxation is refined by means of branches $x_i \leq \lfloor x_{0i} \rfloor \vee x_i \geq \lceil x_{0i} \rceil$ in a *branch-and-bound* solver or by cuts in a *branch-and-cut* solver. Both techniques remove the current non-integral solution from the relaxation. For a detailed overview and derivations of different cuts we refer to [19].

[1] We follow [7] and use column-style Hermite normal form with negative coefficients in the lower left triangle for nonsingular square matrices.

Instead of branches, modern solvers introduce extended branches. These are defined by a vector $\boldsymbol{r} \in \mathbb{Z}^n$ and branch on $\boldsymbol{r}\cdot\boldsymbol{x} \leq \lfloor \boldsymbol{r}\cdot\boldsymbol{x}_0 \rfloor \vee \boldsymbol{r}\cdot\boldsymbol{x} \geq \lceil \boldsymbol{r}\cdot\boldsymbol{x}_0 \rceil$. Thus, an extended branch does not have to be along one of the variables. Most SMT solvers come with a DPLL engine, that can be used to decide on the branch. In this case the theory solver can ask the DPLL engine to decide on the new literal and add it to the constraint system. This procedure is repeated until the LP relaxation becomes unsatisfiable or an assignment satisfies both, the LP relaxation and the integer constraints of the variables. When the LP relaxation becomes unsatisfiable a conflict is produced and the DPLL engine will explore the other branch. When all branches have been explored the original system is known to be unsatisfiable. This is the technique used in our solver SMTInterpol.

Another technique introduces cuts. These are constraints of the form $\boldsymbol{r}\cdot\boldsymbol{x} \leq c$ that are implied by the current constraint system $A\boldsymbol{x} \leq \boldsymbol{b}$ and exclude the current rational solution in the LP relaxation. An example are Gomory cuts [9]. Cuts have the advantage that they can be propagated and no backtracking is necessary. While they exclude only non-integer solutions, their negation can still be satisfiable in conjunction with the LP relaxation. Thus, cut constraints need a specialised proof rule. An interpolating SMT solver also needs a specialised procedure to interpolate these cuts. Branches on the other hand are simple case splits. A way to achieve the best of both worlds is the cuts from proofs algorithm [7]. This algorithm computes extended branches. But as we will see in the next section, one of the two cases is trivially unsatisfiable. Thus, the other case can be propagated by the theory and no backtracking is necessary. This algorithm combines the strength of cuts with the simplicity of branches.

4 Cuts from Proofs

We will now give a short overview of the cuts from proofs algorithm. We focus on the main ideas needed for our adaptation to mixed arithmetic. In this chapter we assume that $\boldsymbol{x}$ only contains integer variables.

The algorithm is based on the Simplex algorithm. The solution space forms a polyhedron in $\mathbb{Q}^n$. If the solution space is non-empty, the Simplex algorithm returns a solution of $A\boldsymbol{x} \leq b$. We further assume that the returned solution $\boldsymbol{x}_0$ is a vertex of the polyhedron, i. e., there is a nonsingular square submatrix A' and a corresponding vector b', such that $A'\boldsymbol{x_0} = \boldsymbol{b'}$. We call $A'\boldsymbol{x} \leq \boldsymbol{b}'$ the *defining constraints* of the vertex. If the returned solution is not on a vertex we introduce artificial branches on input variables into A and use these branches as defining constraints. These branches are rarely needed in practise.

The main idea is to bring the constraint system $A'\boldsymbol{x} \leq \boldsymbol{b}'$ into a Hermite normal form H and to compute the unimodular matrix U with $A'U = H$. The Hermite normal form is uniquely defined. The constraint system $A'\boldsymbol{x} \leq \boldsymbol{b}'$ is equivalent to $H\boldsymbol{y} \leq \boldsymbol{b}'$ with $\boldsymbol{y} := U^{-1}\boldsymbol{x}$. Since the solution $\boldsymbol{x}_0$ of $A'\boldsymbol{x}_0 = \boldsymbol{b}'$ is not integral, the corresponding vector $\boldsymbol{y}_0 = U^{-1}\boldsymbol{x}_0$ is not integral, either. The cuts from proofs algorithm creates an extended branch on one of the components y_i of $\boldsymbol{y}$, i. e., $y_i \leq \lfloor y_{0i} \rfloor$ or $y_i \geq \lceil y_{0i} \rceil$.

Although the description in Dillig et al. [7] looks different, they really do the same. They introduce the notion of *proof of unsatisfiability*, which they define as an equation $d\boldsymbol{r} \cdot \boldsymbol{x} = n$ where $\boldsymbol{r}$ is an integer vector, n, d are integers, and d does not divide n. It is clear that this equation cannot have integer solutions for $\boldsymbol{x}$. In particular, they define $\boldsymbol{r}$ as the i-th row of $H^{-1}A'$ and n/d as the i-th entry of $H^{-1}b$. From this proof of unsatisfiability they generate the extended branch

$$1/g(d_i\boldsymbol{r}_i \cdot \boldsymbol{x}) \leq \lfloor n_i/g \rfloor \vee 1/g(d_i\boldsymbol{r}_i \cdot \boldsymbol{x}) \geq \lceil n_i/g \rceil$$

where g is the greatest common divisor of the components of $d_i\boldsymbol{r}_i$. However, $g = d_i$ since $\boldsymbol{r}_i$ is a row of the unimodular matrix U^{-1}. Thus, they branch on $y_i \leq \lfloor y_{0i} \rfloor$ or $y_i \geq \lceil y_{0i} \rceil$.

Due to the special shape of the Hermite normal form used here and in [7], the constraint $H\boldsymbol{y} \leq \boldsymbol{b}'$ implies that $\boldsymbol{y} \leq \boldsymbol{y}_0$. This means that the second branch can be omitted, i. e., one can introduce the cut $y_i \leq \lfloor y_{0i} \rfloor$. This is shown in the following lemma.

Lemma 1. *Let $A\boldsymbol{x}_0 = b$ and $H = AU$ the Hermite normal form of A. Let $\boldsymbol{y}_0 = U^{-1}\boldsymbol{x}_0$. Then*

$$A\boldsymbol{x} \leq \boldsymbol{b} \text{ implies that } U^{-1}\boldsymbol{x} \leq \boldsymbol{y}_0$$

Proof. We assume $A\boldsymbol{x} \leq \boldsymbol{b}$ and show for every row i that $(U^{-1}\boldsymbol{x})_i \leq y_{0i}$ by induction over i. Let $i \geq 1$. Since $(HU^{-1}\boldsymbol{x})_i = (A\boldsymbol{x})_i \leq b_i$,

$$\sum_{j=1}^{n} h_{ij}(U^{-1}\boldsymbol{x})_j = (HU^{-1}\boldsymbol{x})_i \leq b_i = (A\boldsymbol{x}_0)_i = (H\boldsymbol{y}_0)_i = \sum_{j=1}^{n} h_{ij}y_{0j}.$$

Isolating $h_{ii}(U^{-1}\boldsymbol{x})_i$ on the left hand side, we get (note that $h_{ij} = 0$ for $j > i$)

$$h_{ii}(U^{-1}\boldsymbol{x})_i \leq h_{ii}y_{0i} + \sum_{j=1}^{i-1} h_{ij}(y_{0j} - (U^{-1}\boldsymbol{x})_j).$$

From the induction hypothesis $(U^{-1}\boldsymbol{x})_j \leq y_{0j}$ and $h_{ij} \leq 0$ for $j < i$ we derive $h_{ii}(U^{-1}\boldsymbol{x})_i \leq h_{ii}y_{0i}$. Now $(U^{-1}\boldsymbol{x})_i \leq y_{0i}$ follows, since $h_{ii} > 0$. □

Example 1. The following example stems from Pugh [18]. Given the constraint system $27 \leq 11x + 13y \leq 45$ and $-10 \leq 7x - 9y \leq 4$. Figure 1 shows that these constraints form a parallelogram that does not contain any integer solution. The Simplex algorithm may choose as defining constraints the upper bounds (which are the thick lines in the diagram):

$$\begin{bmatrix} 11 & 13 \\ 7 & -9 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 45 \\ 4 \end{bmatrix}$$

This gives a non-integer solution for x and y. The algorithm then proceeds by computing the Hermite normal form as

$$H = \begin{bmatrix} 1 & 0 \\ -103 & 190 \end{bmatrix} = \begin{bmatrix} 11 & 13 \\ 7 & -9 \end{bmatrix} \begin{bmatrix} -7 & 13 \\ 6 & -11 \end{bmatrix} = AU.$$

From this it computes the cuts as

$$U^{-1}\begin{bmatrix}x\\y\end{bmatrix} \leq H^{-1}b \Leftrightarrow \begin{bmatrix}11x+13y\\6x+7y\end{bmatrix} \leq \begin{bmatrix}45\\\lfloor\frac{4639}{190}\rfloor\end{bmatrix} = \begin{bmatrix}45\\24\end{bmatrix}.$$

The second cut $6x + 7y \leq 24$ is now added to the system. It follows from the original constraints because

$$\frac{103}{190}\cdot(11x+13y \leq 45) + \frac{1}{190}(7x-9y \leq 4) \text{ gives } 6x+7y \leq \frac{4639}{190}.$$

The figure depicts this cut graphically. Although this cut removes only a small part of the solution space, it replaces the constraint $11x + 13y \leq 45$ by a constraint with smaller coefficients. Continuing with this constraint, the same algorithm will produce the second cut $x + y \leq \lfloor\frac{388}{103}\rfloor = 3$. This cut then replaces the previous cut in the defining constraints and the third cut is $x \leq \lfloor\frac{31}{16}\rfloor = 1$. For the fourth and last cut the Simplex algorithm chooses another constraint, e. g., $-10 \leq 7x - 9y$, since the lower right constraint is not inside the solution space anymore. This produces the cut $y \leq \lfloor\frac{17}{9}\rfloor = 1$. The solution space is now empty, which means the system is unsatisfiable. □

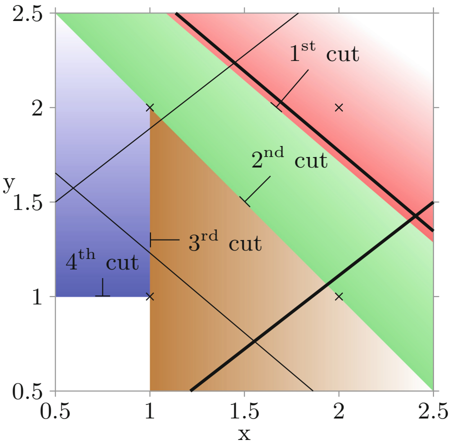


Fig. 1. Run of the cuts from proofs algorithm on Example 1. The black lines are the constraints, which form a parallelogram without integer solutions. The thick lines on the right denote the defining constraints for the first cut. The second cut is computed from the first cut and the lower right constraint and so on. The fourth cut shows that there are no integer solutions.

Why is $y_i \leq \lfloor y_{0i}\rfloor$ a good cut? It is not clear how to answer this question. One can argue that one replaces the variables $\boldsymbol{x}$ with new variables $\boldsymbol{y}$ and solves a very simple constraint system $H\boldsymbol{y} \leq \boldsymbol{b}$ by doing cuts on the $\boldsymbol{y}$ variables. Also, it can be seen that if the constraint $y_i \leq \lfloor y_{0i}\rfloor$ replaces the corresponding i-th row in the defining constraint for every cut (which is what under certain condition the Bland heuristic [19] would do), the resulting constraint system has an integer solution. But the best answer is that it empirically works.

Although the Hermite normal form is unique there is still an important way the produced cuts can be influenced. The order of the defining constraints directly determines the quality of the produced cuts. As a heuristic, the constraints that are most unlikely to change should come first. For this reason we put equality constraints first in the matrix A'. Other than this, we put the rows in the reverse order in which they would be chosen by the Bland heuristic.

Our view of the algorithm is that it transforms the basis $\boldsymbol{x}$ using a unimodular matrix U to a basis $\boldsymbol{y}$, such that the constraint system $A\boldsymbol{x} \leq \boldsymbol{b}$ has a much simpler representation $H\boldsymbol{y} \leq \boldsymbol{b}$ in the new constraint system. Then it creates cuts on the coordinates of the new basis $\boldsymbol{y}$ that are not yet integral. In the next section we want to extend this idea to mixed problems where some variables are real variables and some variables are integer variables.

5 Mixed Cuts from Proofs

As mentioned in the previous section, the basic idea of the cuts from proof algorithm can be described by a transformation of the basis $\boldsymbol{x}$ to a new basis $\boldsymbol{y}$ where the current constraint system is simpler. In this section we extend this idea to mixed real/integer arithmetic.

For mixed arithmetic the basis $\boldsymbol{x} = (x_1 \dots, x_{n_1}, x_{n_1+1}, \dots, x_n)$ is split into n_1 real variables and $n_2 = n - n_1$ integer variables. The transformed basis $\boldsymbol{y} = U\boldsymbol{x}$ should have the same number of real and integer variables. To achieve this the matrix must have the form

$$U = \begin{bmatrix} U_{(r)} & V \\ 0 & U_{(i)} \end{bmatrix}$$

where all coefficients of $U_{(i)}$ are integral.

We further require that every valid solution of $\boldsymbol{y}$ should correspond to a valid solution of $\boldsymbol{x}$. Therefore, the matrix U must be invertible and its inverse should have again this form. The inverse of U is

$$U^{-1} = \begin{bmatrix} U_{(r)}^{-1} & -U_{(r)}^{-1} V U_{(i)}^{-1} \\ 0 & U_{(i)}^{-1} \end{bmatrix}$$

We require that $U_{(i)}^{-1} \in \mathbb{Z}^{n_2 \times n_2}$ (hence $U_{(i)}$ must be unimodular) and that $U_{(r)}$ is nonsingular. We call a matrix of this form a mixed transformation matrix.

Definition 1. *Given a mixed problem with n_1 real and n_2 integer variables and $n = n_1 + n_2$. A matrix $U \in \mathbb{Q}^{n \times n}$ is a* mixed transformation matrix *if there are a nonsingular $U_{(r)} \in \mathbb{Q}^{n_1 \times n_1}$, a unimodular $U_{(i)} \in \mathbb{Z}^{n_2 \times n_2}$, and $V \in \mathbb{Q}^{n_1 \times n_2}$, such that*

$$U = \begin{bmatrix} U_{(r)} & V \\ 0 & U_{(i)} \end{bmatrix}.$$

By the above observation the mixed transformation matrices form a subgroup of $\mathbb{Q}^{n\times n}$, i. e., the inverse of a mixed transformation matrix is again a mixed transformation matrix. Thus, we have the following lemma stating that a valid solution for the original system corresponds to a valid solution of the transformed coordinate system.

Lemma 2. *Let U be a mixed transformation matrix. Then*

$$\boldsymbol{x} \in (\mathbb{Q}^{n_1} \times \mathbb{Z}^{n_2}) \textit{ if and only if } U\boldsymbol{x} \in (\mathbb{Q}^{n_1} \times \mathbb{Z}^{n_2}).$$

Proof. Follows directly from the shape of U and U^{-1}. □

Again we bring the matrix A of defining constraints into a normal form H with $H = AU$ where U is a mixed transformation matrix. We call this the mixed normal form. A matrix H is in mixed normal form if

$$H = \begin{bmatrix} Id & 0 \\ * & H_{(i)} \end{bmatrix}$$

where $Id \in \mathbb{Q}^{n_1\times n_1}$ is the identity matrix, $*$ is an arbitrary $\mathbb{Q}^{n_2\times n_1}$ matrix, and $H_{(i)} \in \mathbb{Q}^{n_2\times n_2}$ is in Hermite normal form.

A matrix has a normal form if and only if the first n_1 constraints are linear independent on the real variables. Thus we may have to reorder the matrix A to put these rows first. Since the matrix A is nonsingular, it is always possible to reorder the rows of A such that the top-left $n_1 \times n_1$ submatrix is nonsingular.

Lemma 3. *Let $A \in \mathbb{Q}^{n\times n}$, such that the upper left $n_1 \times n_1$ submatrix is nonsingular. Then A has a unique mixed normal form $H = AU$, such that U is a mixed transformation matrix.*

Proof. Existence. We subdivide A into

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

and note that A_{11} is invertible in $\mathbb{Q}^{n_1\times n_1}$. We set $U_{(r)} = A_{11}^{-1}$. Then we transform the matrix $A_{22} - A_{21}U_{(r)}A_{12}$ into Hermite Normal Form $H_{(i)}$ with

$$H_{(i)} = (A_{22} - A_{21}U_{(r)}A_{12})U_{(i)}.$$

The mixed normal form is

$$H = \begin{bmatrix} Id & 0 \\ A_{21}U_{(r)} & H_{(i)} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} U_{(r)} & -U_{(r)}A_{12}U_{(i)} \\ 0 & U_{(i)} \end{bmatrix}.$$

Uniqueness. Assume that $H = AU$ with

$$H = \begin{bmatrix} Id & 0 \\ H_{21} & H_{22} \end{bmatrix}, A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, U = \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$$

where H_{22} is in Hermite normal form and U_{22} is unimodular. The top left corner of H gives $Id = A_{11}U_{11}$, thus $U_{11} = A_{11}^{-1}$ is unique. The top right corner of H gives $0 = A_{11}U_{12} + A_{12}U_{22}$, thus $U_{12} = -A_{11}^{-1}A_{12}U_{22}$. Inserting this into the equation for the bottom right corner of H gives

$$H_{22} = (-A_{21}A_{11}^{-1}A_{12} + A_{22})U_{22}$$

Thus, H_{22} is the unique Hermite normal form of $-A_{21}A_{11}^{-1}A_{12} + A_{22}$ and U_{22} is also unique. This shows that U is unique and therefore also $H = AU$. □

Example 2. We change the constraint system of Example 1 and make the variable x real-valued. Secondly, we require that $x - \lfloor x \rfloor \leq 0.2$. To express this we introduce an integer variable z representing $\lfloor x \rfloor$. The constraint system is

$$\begin{array}{rcl} 27 \leq & 11x + 13y & \leq 45 \\ -10 \leq & 7x - 9y & \leq 4 \\ 0 \leq & x - z & \leq 0.2 \end{array}$$

Figure 2 depicts the solution space. The integer points are denoted by crosses; x, y must lie in one of the horizontal thick lines for z, y to be integer and the last constraint to be satisfied. Also x, y must lie in the parallelogram. As can be seen from the figure, there is a solution, e. g., $x = 1.2, y = 2, z = 1$. Our algorithm uses the Simplex algorithm to find a vertex in the solution space. We assume it uses as defining constraints

$$\begin{array}{rl} 11x + 13y & \leq 45 \\ -7x + 9y & \leq 10 \\ z - x & \leq 0 \end{array}$$

with the solution $x = z = \frac{55}{38}, y = \frac{85}{38}$. The mixed normal form is

$$H = \begin{bmatrix} 1 & 0 & 0 \\ -7/11 & 100/11 & 0 \\ -1/11 & -9/11 & 1 \end{bmatrix} = \begin{bmatrix} 11 & 13 & 0 \\ -7 & 9 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/11 & -13/11 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix} = AU$$

From this it computes the cuts as

$$U^{-1} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \leq H^{-1}b \Leftrightarrow \begin{bmatrix} 11x + 13y \\ y \\ 2y + z \end{bmatrix} \leq \begin{bmatrix} 45 \\ \lfloor 85/38 \rfloor \\ \lfloor 225/38 \rfloor \end{bmatrix} = \begin{bmatrix} 45 \\ 2 \\ 5 \end{bmatrix}.$$

The figure visualises the cuts. Note that the cut $2y + z \leq 5$ is for the z variable, so x may still be in the differently shaded area right of this cut. When the two cuts $y \leq 2$ and $2y + z \leq 5$ are introduced, the Simplex algorithm will search for a new vertex, e. g., $x = \frac{8}{7}, y = 2, z = 1$ with the defining constraints

$$\begin{array}{r} -7x + 9y \leq 10, \\ y \leq 2, \\ 2y + z \leq 5. \end{array}$$

This solution has integer values for y and z and the algorithm terminates with this solution.

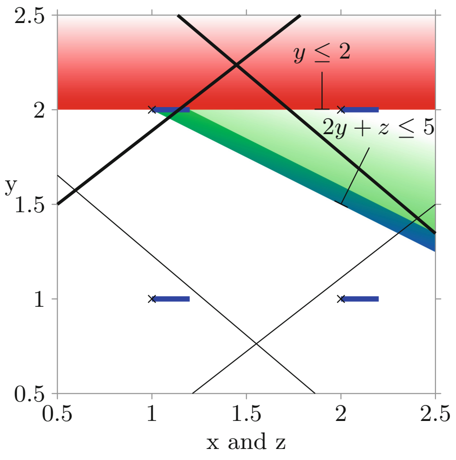


Fig. 2. Run of our algorithm on Example 2. The thick lines on the top denote the defining constraints. From the constraints two cuts are computed. These cuts meet at the vertex $y = 2, z = 1$, which can be extended to a feasible solution with $x = 8/7$.

Our algorithm introduces extended branches $y_i \leq \lfloor y_{0i} \rfloor \vee y_i \geq \lceil y_{0i} \rceil$. As in Lemma 1, one can see that these branches are cuts if the matrix H contains only nonpositive values in the lower left triangle. However, Lemma 3 shows that $H_{21} = A_{21}A_{11}^{-1}$ and there are examples where the coefficients of this matrix are positive. Moreover, since the mixed normal form is unique, there is no simple fix. In the above example we had to carefully choose the defining constraints to get cuts instead of branches. In our implementation of the algorithm we usually get several extended branches until the defining constraints contain enough integer constraints, which means that there are only few non-zero entries in H_{21}. Then usually cuts are generated.

6 Implementation and Evaluation

We implemented the technique presented in this paper in the SMT solver SMT-Interpol [5]. When experimenting, we discovered that most of the time spent by SMTInterpol was not in the cut engine, but in trying to find a solution to the LP relaxation and deciding on already created extended branches. In the runs we investigated further, SMTInterpol quickly creates several extended branches and cuts and then spends hours in the DPLL engine and the Simplex algorithm to solve the LP relaxation. In the end, the benchmark is solved without adding a new branch or cut. The main problem is that our implementation keeps all branches and cuts generated by the technique proposed here and even decides on them. To rectify this problem, we created a second version of SMTInterpol that removes cuts and branches that did not help to close the current branch in the decision tree of the DPLL engine. This version is still experimental and part of ongoing work.

We evaluated the technique on a number of benchmarks. We used hard conjunctive benchmarks, since we aimed at evaluating the cut engine and not the DPLL engine. Since SMTLIB [2] currently does not contain a logic for mixed linear arithmetic without arrays or quantifiers, we created the logic QF_LIRA. This logic is also supported by CVC4 [1], MathSAT 5 [6], yices 2 [8], and Z3 [15][2].

In the evaluation we include a *virtual solver* that combines yices 2 and both versions of SMTInterpol. We do not report times on this solver, but only the number of benchmarks that would be solved by a portfolio of the combined solvers. We chose yices 2 since it is the best performing solver on our benchmark set (closely followed by MathSAT).

The evaluation was performed on StarExec[3] using a timeout of 600 seconds for both CPU and wall time. Memory was limited to 8 GB. We created some benchmarks based on existing benchmarks for linear integer arithmetic, especially those that test the cut generation engine. In the following, we will discuss the benchmarks and the results.

Family Cut_Lemmas Biased. To generate this set of benchmarks, we used all benchmarks from the cut_lemmas family from QF_LIA that are flagged as unsatisfiable. We systematically switched the sort of the variables in the order of their declaration. After we switched one variable, we ran SMTInterpol for five minutes to check if the benchmark was still unsatisfiable. If it was we kept the modification. Otherwise we reverted to the last unsatisfiable modification. The goal is to create a hard unsatisfiable and a few hard satisfiable benchmarks by finding the limit where the satisfiability of the benchmarks changes. Since the modification was guided by SMTInterpol, we call this family *biased*. The chosen modifications depend on which solver we initially used to solve the benchmark. We created a total of 1575 benchmarks. Even though we could know the status of most of these benchmarks, we did not include it in the file[4]. But we checked that if two or more solvers solved the same benchmark, they agreed on the status.

The results are shown in Table 1. The difference between wall time and CPU time for SMTInterpol is caused by the Java virtual machine. SMTInterpol itself is single threaded, but, e. g., the garbage collection runs in parallel. We remark that the virtual solver (combination of yices 2 and both variants of SMTInterpol) solves all but 22 benchmarks. One of the remaining benchmarks can be solved by both CVC4 and MathSAT 5 which additionally solves three more benchmarks. The remaining 18 cannot be solved by any of the solvers used in the evaluation. Furthermore, both versions of SMTInterpol could solve benchmarks that no competing solver (CVC4, MathSAT 5, yices 2, or Z3) could solve.

Figure 3 shows scatter plots that compare yices 2 resp. MathSAT 5 and the version of SMTInterpol that forgets literals. The white area at the bottom of each

[2] Z3 actually warns the user that it does not know this logic. The solver works nevertheless and interprets the logic as expected.

[3] See https://www.starexec.org/starexec/secure/explore/spaces.jsp?id=62799 using username `public` and password `public`.

[4] This causes StarExec to report both results *sat* and *unsat* as wrong.

Table 1. Results of the evaluation on the benchmarks from the biased family. For each solver we report the number of solved benchmarks, the number of satisfiable resp. unsatisfiable benchmarks, the wall and CPU time and the number of benchmarks that could not be solved by any competing solver.

Solver	# solved	# sat	# unsat	wall time	CPU time	# only
SMTInterpol	1107/1575	447	660	64079	68397	42
SMTInterpol forget	1226/1575	465	761	94262	99743	57
CVC4	1309/1575	483	826	63338	63359	0
MathSAT 5	1404/1575	521	883	59959	60000	3
yices 2	1427/1575	509	918	29272	29289	10
Z3	1147/1575	511	636	71688	71736	0
virtual	1553/1575	539	1014			

plot is caused by the startup overhead of the Java virtual machine. Both plots show that there are several problems on which yices 2 and MathSAT 5 time out while SMTInterpol solves them in less than a second. Overall the difficulty of a problem for yices 2 and MathSAT 5 is quite unrelated to the difficulty for SMTInterpol. This undermines the thesis that these solver complement each other well. It is also reflected in the number of benchmarks solved by the virtual solver.

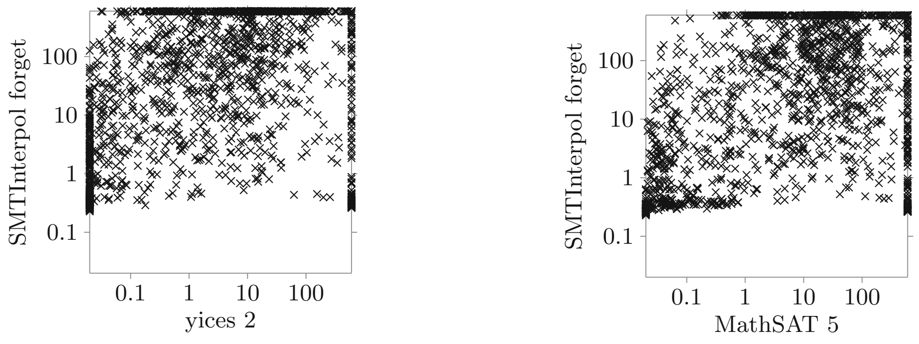


Fig. 3. Scatter plot comparing yices 2 resp. MathSAT 5 and the version of SMTInterpol that forgets literals on the biased family.

Family Cut_Lemmas Unbiased. Again, we used all benchmarks from the cut_lemmas family that are flagged as unsatisfiable. This time, we created 10 new benchmarks from each of these benchmarks by randomly changing the sort of 20 % of the variables from integer to real. We chose that percentage because it creates roughly the same number of unsat and sat benchmarks. This family does not use any solver in the creation and should thus not be biased to a specific solver. We created a total of 930 benchmarks in this division. Since we do not know the status of these benchmarks, no status information is recorded in the generated files. But, again, whenever two solvers solved the same benchmark, they agreed on the status.

Table 2. Results of the evaluation on the benchmarks from the unbiased family. For each solver we report the number of solved benchmarks, the number of satisfiable resp. unsatisfiable benchmarks, the wall and CPU time, and the number of benchmarks that could not be solved by any competing solver.

Solver	# solved	# sat	# unsat	wall time	CPU time	# only
SMTInterpol	697/930	218	479	33421	35936	27
SMTInterpol forget	751/930	236	515	43473	46340	29
CVC4	785/930	239	546	35037	35109	0
MathSAT 5	832/930	259	573	28686	28695	3
yices 2	831/930	250	581	22260	22267	7
Z3	708/930	239	469	38762	38779	0
virtual	915/930	270	645			

The results are shown in Table 2. From the 15 benchmarks that could not be solved by the virtual solver, three could only be solved by MathSAT 5. Both versions of SMTInterpol solve several benchmarks that cannot be solved by CVC4, MathSAT 5, yices 2, or Z3.

The scatter plots comparing yices 2 resp. MathSAT 5 to the forget version of SMTInterpol look almost identical to the ones shown in Fig. 3.

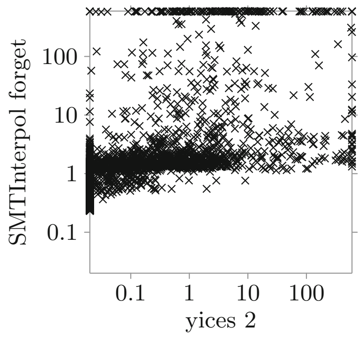
Table 3. Results of the evaluation on the benchmarks from the dillig family. For each solver we report the number of solved benchmarks, the number of satisfiable resp. unsatisfiable benchmarks, the wall and CPU time and the number of benchmarks that could not be solved by any competing solver.

Solver	# solved	# sat	# unsat	wall time	CPU time	# only
SMTInterpol	2204/2330	2187	17	10858	14710	0
SMTInterpol forget	2194/2330	2176	18	13303	17319	1
CVC4	1967/2330	1938	29	44360	44372	0
MathSAT 5	2317/2330	2288	29	17309	17333	0
yices 2	2302/2330	2273	29	17115	17132	0
Z3	2101/2330	2072	29	69255	69344	0
virtual	2330/2330	2301	29			

Family Dillig. The benchmarks from the cuts from proofs paper [7] are available in the SMTLIB in QF_LIA in the dillig family. For each of these benchmarks, we created 10 new benchmarks where we randomly changed the sort of 20 % of the variables from integer to real. This lead to a total of 2330 benchmarks. Since the benchmarks are randomly generated, we do not know the status. But the solvers agreed on the status of those benchmarks that could be solved by multiple solvers.

The results for the dillig family are shown in Table 3. The virtual solver solves *all* benchmarks in this family even though no single solver could solve all benchmarks.

The scatter plots from Fig. 4 compare yices 2 resp. MathSAT 5 and the forget version of SMTInterpol. This time there seems to be a strange line at slightly more than one second. Either SMTInterpol solves a benchmark in this time or it does not solve the benchmark at all.



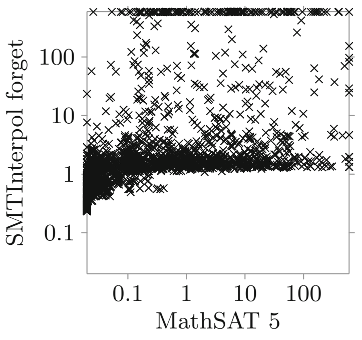


Fig. 4. Scatter plots comparing yices 2 resp. MathSAT 5 and the version of SMTInterpol that forgets literals on the dillig family.

Family Tightrhombus. We created 44 benchmarks specifically designed to test the cut engine. These benchmarks are inspired by the tightrhombus benchmarks used to test the cut engine for QF_LIA. They encode a tight rhombus in the following way. Choose coefficients $c_x > 0$, $c_y > 0$ and scale $s > 0$. The rhombus for QF_LIA is created as

$$0 \leq (c_x \cdot s)x - (c_y \cdot s + 1)y \leq s - 1 \ \wedge\ 1 \leq (c_x \cdot s + 1)x - (c_y \cdot s)y \leq s$$

for integer variables x and y. For mixed arithmetic, we use a real-valued variable y and bound the distance between y and the nearest integer point. The bound can be computed for each rhombus. Since yices 2 does not support the `to_int` construct from SMTLIB, we encode it using a fresh integer variable z. We created benchmarks for $c_x = 273, c_y = 245$ resp. $c_x = 283, c_y = 245$ for scales $s = 10^{i+1}$, $0 \leq i \leq 10$. We carefully chose the bound on the distance between y and z such that the benchmark is barely satisfiable. To create unsatisfiable benchmarks, we subtracted a small value from the bound. We chose 10^{-13} to not create trivially unsatisfiable benchmarks since the minimal distance between y and z for scale 10^{11} is very small.

The results for this family are shown in Table 4. We omit the virtual solvers since both version of SMTInterpol solve all the benchmarks. While CVC4 only solves benchmarks with scale up to 4, MathSAT solves all satisfiable benchmarks with $c_x = 273$, but has problems on the other set. Similarly, Z3 solves almost all satisfiable benchmarks with $c_x = 283$ (except for scale $s = 10^{11}$) but has

Table 4. Results of the evaluation on the benchmarks from the tightrhombus family. For each solver we report the number of solved benchmarks, the number of satisfiable resp. unsatisfiable benchmarks, the wall and CPU time and the number of benchmarks that could not be solved by any competing solver.

Solver	# solved	# sat	# unsat	wall time	CPU time	# only
SMTInterpol	44/44	22	22	15	20	12
SMTInterpol forget	44/44	22	22	15	20	12
CVC4	20/44	10	10	585	585	0
MathSAT 5	21/44	15	6	326	326	0
yices 2	18/44	10	8	514	514	0
Z3	21/44	15	6	364	365	0

problems on the other set. Also note that 12 benchmarks were solved only by either of the variants of SMTInterpol, but not by the other solvers.

This evaluation shows that the clear winner is a combination of the (to our knowledge unpublished) technique used by yices 2 with the technique presented in this paper. Such a combination advances the state of the art in mixed linear arithmetic solving in the SMT context. Furthermore, the technique presented in this paper is able to solve some benchmarks that no other technique can solve. The comparison between the performances of the different versions of SMTInterpol shows that removal of literals that do not contribute to closing of a decision branch sometimes is beneficial.

7 Conclusion and Future Work

We presented a novel method to compute cuts in mixed linear arithmetic solving in the context of SMT. The method is inspired by the cuts from proofs algorithm used to solve integer linear arithmetic. It transforms the original constraint system into a simpler one. This is achieved by transforming the basis of the original constraint system into a new one. Cuts and branches are then created for the new basis. We showed in some experiments that this new technique is able to solve benchmarks that cannot be solved by state-of-the-art solvers.

The evaluation showed that the cut engine is not the bottleneck. Instead SMTInterpol spends most time in the Simplex algorithm and the DPLL engine deciding on the extended branches. An investigation when and which branches and cuts should be removed from the solver is part of future work.

References

1. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer, Heidelberg (2011)

2. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB Standard: 2.0. In: SMT (2010)
3. Berezin, S., Ganesh, V., Dill, D.L.: An online proof-producing decision procedure for mixed-integer linear arithmetic. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 521–536. Springer, Heidelberg (2003)
4. Bobot, F., Conchon, S., Contejean, E., Iguernelala, M., Mahboubi, A., Mebsout, A., Melquiond, G.: A simplex-based extension of fourier-motzkin for solving linear integer arithmetic. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 67–81. Springer, Heidelberg (2012)
5. Christ, J., Hoenicke, J., Nutz, A.: SMTInterpol: an interpolating smt solver. In: Donaldson, A., Parker, D. (eds.) SPIN 2012. LNCS, vol. 7385, pp. 248–254. Springer, Heidelberg (2012)
6. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT solver. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013 (ETAPS 2013). LNCS, vol. 7795, pp. 93–107. Springer, Heidelberg (2013)
7. Dillig, I., Dillig, T., Aiken, A.: Cuts from proofs: a complete and practical technique for solving linear inequalities over integers. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 233–247. Springer, Heidelberg (2009)
8. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 737–744. Springer, Heidelberg (2014)
9. Dutertre, B., de Moura, L.: Integrating simplex with DPLL(T). Technical report, CSL, SRI INTERNATIONAL (2006)
10. Griggio, A.: A practical approach to satisfiability modulo linear integer arithmetic. JSAT **8**(1/2), 1–27 (2012)
11. Jovanović, D., de Moura, L.: Cutting to the chase solving linear integer arithmetic. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS, vol. 6803, pp. 338–353. Springer, Heidelberg (2011)
12. King, T., Barrett, C., Dutertre, B.: Simplex with sum of infeasibilities for SMT. In: FMCAD, pp. 189–196. IEEE (2013)
13. King, T., Barrett, C.W., Tinelli, C.: Leveraging linear and mixed integer programming for SMT. In: FMCAD, pp. 139–146. IEEE (2014)
14. Martin, A.: General mixed integer programming: computational issues for branch-and-cut algorithms. In: Jünger, M., Naddef, D. (eds.) Computational Combinatorial Optimization. LNCS, vol. 2241, p. 1. Springer, Heidelberg (2001)
15. de Moura, L., Bjørner, N.S.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
16. Neumaier, A., Shcherbina, O.: Safe bounds in linear and mixed-integer linear programming. Math. Program. **92**(2), 283–296 (2004)
17. de Oliveira, D.C.B., Monniaux, D.: Experiments on the feasibility of using a floating-point simplex in an SMT solver. In: PAAR-2012, pp. 19–28. EasyChair (2012)
18. Pugh, W.: The omega test: a fast and practical integer programming algorithm for dependence analysis. Commun. ACM **8**, 4–13 (1992)
19. Schrijver, A.: Theory of Linear and Integer Programming. John Wiley & Sons, Chichester (1986)