Computing a Complete Basis for Equalities Implied by a System of LRA Constraints

Martin Bromberger[1,2] and Christoph Weidenbach[1]

[1] Max Planck Institute for Informatics, Saarbrücken, Germany
{mbromber,weidenb}@mpi-inf.mpg.de
[2] Graduate School of Computer Science, Saarbrücken, Germany

Abstract

We present three new methods that investigate the equalities implied by a system of linear arithmetic constraints. Implied equalities can be used to simplify linear arithmetic constraints and are valuable in the context of Nelson-Oppen style combinations of theories. The first method efficiently checks whether a system of linear arithmetic constraints implies an equality at all. In case the system does, the method also returns a valid equality as an explanation. The second method uses the first method to compute a basis for all implied equalities, i.e., a finite representation of all equalities implied by the linear arithmetic constraints. The third method uses the second method to check efficiently whether a system of linear arithmetic constraints implies a given equality.

1 Introduction

Polyhedra and the systems of linear arithmetic constraints $Ax \leq b$ defining them have a vast number of theoretical and real-world applications [2, 9]. It is, therefore, no surprise that the theory of linear arithmetic is also one of the most popular and best investigated theories for *satisfiability modulo theories* (SMT) solving [5, 6, 7]. Equalities are a special instance of linear arithmetic constraints. They are useful in simplifying systems of arithmetic constraints [7], and they are essential for the Nelson-Oppen style combinations of theories [4]. However, they are also an obstacle for our fast cube tests [3], which find integer solutions by exploring the interior of a polyhedron with cubes. If a system of linear arithmetic constraints $Ax \leq b$ implies an equality, then it has only a surface and no interior; so our cube tests cannot explore an interior and will certainly fail. In order to expand the applicability of our cube tests, we are interested in methods that find, isolate, and eliminate implied equalities from systems of linear arithmetic constraints. We mentioned such methods already in [3], but we will fully present three methods here for the first time.

In Section 3, we present the first method, which efficiently checks whether a system of linear arithmetic constraints $Ax \leq b$ implies an equality at all. We can detect the existence of an implied equality by searching for a cube contained in $Ax \leq b$. If the maximal edge length of such a cube is 0, there exists an implied equality (Lemma 6). This test can be further simplified. By turning all inequalities in $Ax \leq b$ into strict inequalities $Ax < b$, the interior of the original polyhedron remains while the surface disappears. If $Ax < b$ is unsatisfiable, then $Ax \leq b$ has no interior and implies an equality (Lemma 7). Based on an explanation of unsatisfiability for $Ax < b$, the method generates an implied equality as an explanation (Lemma 8).

The second method, introduced in Section 4, consists of the algorithm `EqualityBasis`($Ax \leq b$) that computes a basis for all implied equalities, i.e., a finite representation of all equalities implied by the satisfiable linear arithmetic constraints $Ax \leq b$. For this purpose, the algorithm iteratively applies the first method to find, collect, and eliminate equalities from $Ax \leq b$. When $Ax \leq b$ contains no more equalities, then the collected equalities represent an equality basis $Dx = c$, i.e., any implied equality can be obtained by a linear combination from $Dx = c$.

The equality basis has many applications. In combination with the algorithm `Subst`$(Dx = c, Ax \leq b)$, Figure 3, it eliminates all equalities implied by $Dx = c$ from $Ax \leq b$, which results in a system of inequalities with an interior and, therefore, improves the applicability of our cube tests. Together with a diophantine equation handler [7], the equality basis is useful for simplifying linear integer constraint systems. Note that simplifying $Ax \leq b$ with the equality basis is only useful if $Ax \leq b$ is satisfiable and if we want to perform subsequent computations on $Ax \leq b$, e.g., maximizing/minimizing multiple cost functions for $Ax \leq b$ or as an intermediate step towards determining an integer solution for $Ax \leq b$.

Finally, in Section 4 we introduce an efficient method testing whether a system of linear arithmetic constraints implies a given equality. The method relies on an equality basis $Dx = c$ of $Ax \leq b$ where the `Subst` algorithm efficiently explores implication of a given equality. For the Nelson-Oppen style combination of theories inside an SMT solver [4], each theory solver has to return all valid equations between variables in its theory. Linear arithmetic theory solvers sometimes guess these equations based on one satisfiable assignment. Then the equations are transferred according to the Nelson-Oppen method without verification. This leads to a backtrack of the combination procedure in case the guess was wrong and eventually led to a conflict. With the availability of our algorithm `Subst`, the guesses can be verified directly and efficiently, even before suggestion. Therefore, the method helps the theory solver in avoiding any conflicts due to wrong guesses together with the overhead of the entailed backtracking. This comes at the price of computing once and for all an equality basis, where an already found satisfying assignment has further potential for simplification, see Appendix B.

2 Preliminaries

While the difference between matrices, vectors, and their components is always clear in context, we generally use upper case letters for matrices (e.g., A), lower case letters for vectors (e.g., x), and lower case letters with an index i or j (e.g., b_i, x_j) as components of the associated vector at position i or j, respectively. The only exceptions are the row vectors $a_i^T = (a_{i1}, \ldots, a_{in})$ of a matrix $A = (a_1, \ldots, a_m)^T$, which already contain an index i that indicates the row's position inside A. In order to save space, we write vectors only implicitly as columns via the transpose $(\,)^T$ operator, which turns all rows $(b_1, \ldots, b_m)$ into columns $(b_1, \ldots, b_m)^T$ and vice versa. We will also abbreviate $(0, \ldots, 0)^T$ as $\mathbf{0}$. Likewise, we will abbreviate $(1, \ldots, 1)^T$ as $\mathbf{1}$.

In the context of SMT theory solvers, we have to deal with inequalities $a_i^T x \leq b_i$ and strict inequalities $a_i^T x < b_i$, where $a_i \in \mathbb{Q}^n$ and $b_i \in \mathbb{Q}$. We will model strict inequalities as non-strict inequalities by generalizing the field $\mathbb{Q}$ for our inequality bounds b_i to $\mathbb{Q}_\delta$.

Lemma 1 ([5])**.** *Let $a_i \in \mathbb{Q}^n$ and $b_i \in \mathbb{Q}$. Then, a set of linear arithmetic constraints S containing strict inequalities $S' = \{a_1^T x < b_1, \ldots, a_m^T x < b_m\}$ is satisfiable iff there exists a rational number $\delta > 0$ such that for all δ' with $0 < \delta' \leq \delta$, $S_{\delta'} = (S \cup S'_{\delta'}) \setminus S'$ is satisfiable, where $S'_{\delta'} = \{a_1^T x \leq b_1 - \delta', \ldots, a_m^T x \leq b_m - \delta'\}$.*

As a result of this observation, δ is expressed symbolically as an infinitesimal parameter [5]. This leads to the ordered vector field $\mathbb{Q}_\delta$ that has pairs of rationals as elements $(p, q) \in \mathbb{Q} \times \mathbb{Q}$, representing $p + q\delta$, with the following operations:

$$\begin{aligned}
(p_1, q_1) + (p_2, q_2) &\equiv (p_1 + p_2, q_1 + q_2) \\
a \cdot (p, q) &\equiv (a \cdot p, a \cdot q) \\
(p_1, q_1) \leq (p_2, q_2) &\equiv (p_1 < p_2) \vee (p_1 = p_2 \wedge q_1 \leq q_2) \\
(p_1, q_1) < (p_2, q_2) &\equiv (p_1 < p_2) \vee (p_1 = p_2 \wedge q_1 < q_2)\,,
\end{aligned}$$

where $a \in \mathbb{Q}$ [5]. Now we can represent $a_i^T x < b_i$ by $a_i^T x \leq b_i - \delta$, where $a_i \in \mathbb{Q}^n$ and $b_i \in \mathbb{Q}$.

Note, however, that $a_i^T x < b_i$ cannot be represented by $a_i^T x \leq b_i - \delta$ if $a_i \in \mathbb{Q}^n$ and $b_i \in \mathbb{Q}_\delta$. To do so, we would have to introduce a second infinitesimal and extend $\mathbb{Q}_\delta$ to $\mathbb{Q} \times \mathbb{Q} \times \mathbb{Q}$. For the remainder of the paper, we will also abbreviate with b_i^δ that we have turned the bound $b_i = (p_i, q_i) \in \mathbb{Q}_\delta$ into a strict bound $b_i^\delta = (p_i, -1) \in \mathbb{Q}_\delta$.

In this paper, we treat *polyhedra* and their definitions through a *system of inequalities* $Ax \leq b$ as interchangeable. For such a system of inequalities, the row coefficients are given by $A = (a_1, \ldots, a_m)^T \in \mathbb{Q}^{m \times n}$, the variables are given by $x = (x_1, \ldots, x_n)^T$, and the inequality bounds are given by $b = (b_1, \ldots, b_m)^T \in \mathbb{Q}_\delta^m$. To avoid misuse of the δ infinitesimal, we also require that the δ-coefficient q_i in $b_i = p_i + q_i\delta$ is either 0 or -1. Moreover, we assume that any constant rows $a_i = \mathbf{0}$ have been eliminated from our system as a preprocessing step. This is a trivial task and eliminates some unnecessarily complicated corner cases. Note that the system of inequalities $Ax \leq b$ is just an abbreviation for the set of inequalities $\{a_1^T x \leq b_1, \ldots, a_m^T x \leq b_m\}$. Since $Ax \leq b$ and $A'x \leq b'$ are just sets, we can write their combination as $(Ax \leq b) \cup (A'x \leq b')$. A special system of inequalities is a system of equations $Dx = c$, which is equivalent to the combined system of inequalities $(Dx \leq c) \cup (-Dx \leq -c)$. For such a system of equalities, the row coefficients are given by $D = (d_1, \ldots, d_m)^T \in \mathbb{Q}^{m \times n}$, the variables are given by $x = (x_1, \ldots, x_n)^T$, and the equality bounds are given by $c = (c_1, \ldots, c_m)^T \in \mathbb{Q}^m$.

We denote by $P_b^A = \{x \in \mathbb{Q}^n : Ax \leq b\}$ the *set of rational solutions* to the system of inequalities $Ax \leq b$ and, therefore, the points inside the polyhedron. Similarly, we denote by $C_e^n(z) = \left\{x \in \mathbb{Q}^n : \forall j \in 1, \ldots, n.\ |x_j - z_j| \leq \frac{e}{2}\right\}$ the set of points contained in the *n-dimensional hypercube* that is parallel to the coordinate axes, has *edge length* $e \in \mathbb{Q}_{\geq 0}$, and has *center* $z \in \mathbb{Q}^n$. For the remainder of this paper, we will consider only hypercubes that are parallel to the coordinate axes. For simplicity, we call these restricted hypercubes *cubes*.

We say that a cube $C_e^n(z)$ *fits* into a polyhedron defined by $Ax \leq b$ if all points inside the cube $C_e^n(z)$ are solutions of $Ax \leq b$, or formally: $C_e^n(z) \subseteq P_b^A$. In order to compute this, we transform the polyhedron $Ax \leq b$ into another polyhedron $Ax \leq b'$. For this new polyhedron, we merely have to test whether the cube's center point z is a solution ($z \in P_{b'}^A$) in order to also determine whether the cube $C_e^n(z)$ fits into the original polyhedron ($C_e^n(z) \subseteq P_b^A$). This is a simple test that requires only evaluation. We presented this entire transformation in [3] as the *linear cube transformation*. To abbreviate the representation of this transformation, we will use the 1-norm [8], which is defined as $\|x\|_1 = |x_1| + \ldots + |x_n|$.

Lemma 2 ([3]). *Let $C_e^n(z)$ be a cube and $Ax \leq b$ be a polyhedron. $C_e^n(z) \subseteq P_b^A$ if and only if $Az \leq b'$, where $b'_i = b_i - \frac{e}{2}\|a_i\|_1$.*

We say that a polyhedron implies an equality $h^T x = g$, where $h \in \mathbb{Q}^n$, $h \neq \mathbf{0}$, and $g \in \mathbb{Q}$, if $h^T x = g$ holds for all $x \in P_b^A$. An equality implied by $Ax \leq b$ is *explicit* if the inequalities $h^T x \leq g$ and $-h^T x \leq -g$ appear in $Ax \leq b$. Otherwise, the equality is *implicit*. Polyhedra implying equalities have only surface points and, therefore, neither an interior nor a center. Thus, all cubes that fit into a polyhedron implying an equality $d^T x \leq c$ with $d \neq \mathbf{0}$ have edge length zero.

For the development of our first method deciding on implied equalities, we will use two formulations of Farkas' Lemma:

Lemma 3 ([2]). *$Ax \leq b$ is satisfiable if and only if every $y \in \mathbb{Q}^m$ with $y \geq \mathbf{0}$ and $y^T A = \mathbf{0}$ satisfies $y^T b \geq 0$, i.e., every non-negative linear combination of inequalities in $Ax \leq b$ does not result in a trivially unsatisfiable inequality, e.g., $0 \leq -1$.*

Lemma 4 ([2]). *$Ax \leq b$ is unsatisfiable if and only if there exists a $y \in \mathbb{Q}^m$ with $y \geq \mathbf{0}$ and $y^T A = \mathbf{0}$ so that $y^T b < 0$, i.e., there exists a non-negative linear combination of inequalities in $Ax \leq b$ that results in the trivially unsatisfiable inequality $y^T Ax \leq y^T b$.*

We call an unsatisfiable set C of inequalities *minimal* if every proper subset $C' \subset C$ is satisfiable. Whenever a polyhedron $Ax \leq b$ is unsatisfiable, there exists a minimal set C of unsatisfiable inequalities so that every inequality in C appears also in $Ax \leq b$ [5]. We call such a minimal set C an *explanation* for $Ax \leq b$'s unsatisfiability. In case we are investigating a minimal set of unsatisfiable inequalities, we can refine the second version of Farkas' Lemma:

Lemma 5. *Let $C = \{{a'}_i^T x \leq b'_i : 1 \leq i \leq m'\}$ be a minimal set of unsatisfiable constraints. Let $A' = ({a'}_1, \ldots, {a'}_{m'})^T$ and $b' = (b'_1, \ldots, b'_{m'})^T$. Then, it holds for every $y \in \mathbb{Q}^{m'}$ with $y \geq \mathbf{0}$, $y^T A' = \mathbf{0}$, and $y^T b' < 0$ that $y_i > 0$ for all $i \in \{1, \ldots, m'\}$.*

Proof. Suppose to the contrary that there exists a $y \geq \mathbf{0}$ with $y^T A' = \mathbf{0}$ and $y^T b' < 0$ such that one component of y is zero. Without loss of generality we assume that $y_{m'} = 0$. Let $C' = \{{a'}_i^T x \leq b'_i : 1 \leq i \leq m' - 1\}$, $A'' = ({a'}_1, \ldots, {a'}_{m'-1})^T$, $b'' = (b'_1, \ldots, b'_{m'-1})^T$, and $y' = (y_1, \ldots, y_{m'-1})^T$. Then, $y' \geq \mathbf{0}$, ${y'}^T A'' = \mathbf{0}$, and ${y'}^T b'' < 0$. However, by Lemma 4, this implies that $A''x \leq b''$ is unsatisfiable and, therefore, $C' \subset C$ is also unsatisfiable. Thus, C is not minimal, which contradicts our initial assumptions. □

3 Detecting Implied Equalities

In this section, we present a method that detects whether a polyhedron $Ax \leq b$ implies an equality at all and then returns one valid equality as explanation. Polyhedra implying equalities have only surface points and, therefore, no interior. Thus, all cubes that fit into a polyhedron implying an equality $h^T x = g$ with $h \neq \mathbf{0}$ have edge length zero.

Lemma 6 ([3])**.** *Let $Ax \leq b$ be a polyhedron. Then, exactly one of the following statements is true: (1) $Ax \leq b$ implies an equality $h^T x = g$ with $h \neq \mathbf{0}$, or (2) $Ax \leq b$ contains a cube with edge length $e > 0$.*

Lemma 6 states that a polyhedron contains either a cube with a positive edge length $e > 0$, or an equality. Since e is arbitrarily small, the factor $\frac{e}{2} \left\|a_i\right\|_1$ is also arbitrarily small. We can even choose our edge length so small that we can ignore the different multiples $\left\|a_i\right\|_1$ and any infinitesimals introduced by strict inequalities. We just have to turn all of our inequalities into strict inequalities.

Lemma 7. *Let $Ax \leq b$ be a polyhedron, where $a_i \neq \mathbf{0}$, $b_i = (p_i, q_i)$, $q_i \in \{-1, 0\}$, and $b_i^\delta = (p_i, -1)$ be the strict versions of the bounds b_i for all $i \in \{1, \ldots, m\}$. Then, the following statements are equivalent: (1) $Ax \leq b$ contains a cube with edge length $e > 0$, and (2) $Ax \leq b^\delta$ is satisfiable, where all bounds b_i of $Ax \leq b$ have been replaced by strict bounds b_i^δ.*

Proof. (1) ⇒ (2): If $Ax \leq b$ contains a cube of edge length $e > 0$, then $Ax \leq b - a'$ is satisfiable, where $a'_i = \frac{e}{2} \left\|a_i\right\|_1$. By Lemma 1, we know that there must exist a $\delta \in \mathbb{Q}$ such that $Ax \leq p + q\delta - a'$ holds. Now, let $\delta' = \min\{a'_i - q_i\delta : i = 1, \ldots, m\}$. Since $a'_i - q_i\delta \geq \delta'$, it holds that $Ax \leq p - \delta'\mathbf{1}$. Since $q_i \in \{-1, 0\}$ and $a'_i = \left\|a_i\right\|_1 > 0$, it also holds that $\delta' > 0$. By Lemma 1, we deduce that $Ax < p$ and, therefore, $Ax \leq b^\delta$ holds.

(2) ⇒ (1): If $Ax \leq b^\delta$ is satisfiable, then we know by Lemma 1 that there must exist a $\delta > 0$ such that $Ax \leq p - \delta\mathbf{1}$ holds. Let $a_{\max} = \max\{\left\|a_i\right\|_1 : i = 1, \ldots, m\}$, $\delta' = \frac{\delta}{2}$, and $e = \frac{\delta}{a_{\max}}$. Then, $p_i - \delta = p_i - \delta' - \frac{e}{2} a_{\max} \leq b_i - \frac{e}{2} \left\|a_i\right\|_1$. Thus, $Ax \leq b$ contains a cube with edge length $e > 0$. □

In case $Ax \leq b^\delta$ is unsatisfiable, $Ax \leq b$ contains no cube with positive edge length and, therefore by Lemma 6, an equality. Dutertre and de Moura presented a dual simplex algorithm

that can solve the systems $Ax \leq b$ and $Ax \leq b^\delta$ [5]. In case $Ax \leq b^\delta$ is unsatisfiable, the algorithm returns an explanation, i.e., a minimal set C of unsatisfiable constraints $a_i^T x \leq b_i^\delta$ from $Ax \leq b^\delta$. If $Ax \leq b$ itself is satisfiable, then we can extract equalities from this explanation: for every $a_i^T x \leq b_i^\delta \in C$, $Ax \leq b$ implies the equality $a_i^T x = b_i$.

Lemma 8. *Let $Ax \leq b$ be a satisfiable polyhedron, where $a_i \neq \mathbf{0}$, $b_i = (p_i, q_i)$, $q_i \in \{-1, 0\}$, and $b_i^\delta = (p_i, -1)$ be the strict versions of the bounds b_i for all $i \in \{1, \ldots, m\}$. Let $Ax \leq b^\delta$ be unsatisfiable. Let C be a minimal set of unsatisfiable constraints $a_i^T x \leq b_i^\delta$ from $Ax \leq b^\delta$. Then, it holds for every $a_i^T x \leq b_i^\delta \in C$ that $a_i^T x = b_i$ is an equality implied by $Ax \leq b$.*

Proof. Because of transitivity of the subset and implies relationships, we can assume that $Ax \leq b$ and $Ax \leq b^\delta$ contain only the inequalities associated with the explanation C. Therefore, $C = \{a_1^T x \leq b_1^\delta, \ldots, a_m^T x \leq b_m^\delta\}$. By Lemma 4 and $Ax \leq b^\delta$ being unsatisfiable, we know that there exists a $y \in \mathbb{Q}^m$ with $y \geq 0$, $y^T A = \mathbf{0}$, and $y^T b^\delta < 0$. By Lemma 3 and $Ax \leq b$ being satisfiable, we know that $y^T b \geq 0$ is also true. By Lemma 5, we know that $y_k > 0$ for every $k \in \{1, \ldots, m\}$.

Now, we will use $y^T b^\delta < 0$, $y^T b \geq 0$, and the definitions of $<$ and $\leq$ for $\mathbb{Q}_\delta$ to prove that $y^T b = 0$ and $b = p$. Since $y^T b^\delta < 0$, we get that $y^T p \leq 0$. Since $y^T b \geq 0$, we get that $y^T p \geq 0$. If we combine $y^T p \leq 0$ and $y^T p \geq 0$, we get that $y^T p = 0$. From $y^T p = 0$ and $y^T b \geq 0$, we get $y^T q \geq 0$. Since $y > 0$ and $q_i \in \{-1, 0\}$, we get that $y^T q = 0$ and $q_i = 0$. Since $q_i = 0$, $b = p$.

Next, we multiply $y^T A = 0$ with an $x \in P_b^A$ to get $y^T Ax = 0$. Since $y_k > 0$ for every $k \in \{1, \ldots, m\}$, we can solve $y^T Ax = 0$ for every $a_k^T x$ and get: $a_k^T x = -\sum_{i=1, i\neq k}^m \left(\frac{y_i}{y_k} a_i^T x\right)$.
Likewise, we will solve $y^T b = 0$ for every b_k to get: $b_k = -\sum_{i=1, i\neq k}^m \left(\frac{y_i}{y_k} b_i\right)$.
Since $x \in P_b^A$ satisfies all $a_i^T x \leq b_i$, we can deduce b_k as the lower bound of $a_k^T x$:

$$a_k^T x = -\sum_{i=1, i\neq k}^m \left(\frac{y_i}{y_k} a_i^T x\right) \geq -\sum_{i=1, i\neq k}^m \left(\frac{y_i}{y_k} b_i\right) = b_k \,,$$

which proves that $Ax \leq b$ implies $a_k^T x = b_k$. □

Lemma 8 justifies simplifications on $Ax \leq b^\delta$. We can eliminate all inequalities in $Ax \leq b^\delta$ that cannot appear in the explanation of unsatisfiability, i.e., all inequalities $a_i^T x \leq b_i^\delta$ that cannot form an equality $a_i^T x = b_i$ that is implied by $Ax \leq b$. For example, if we have an assignment $v \in \mathbb{Q}^n$ such that $Av \leq b$ is true, then we can eliminate every inequality $a_i^T x \leq b_i^\delta$ for which $a_i^T v = b_i$ is false. According to this argument, we can also eliminate all inequalities $a_i^T x \leq b_i^\delta$ that were already strict inequalities in $Ax \leq b$.

4 Computing an Equality Basis

An *equality basis* for a satisfiable system of inequalities $Ax \leq b$ is a system of equalities $D'x = c'$ such that all (explicit and implicit equalities) implied by $Ax \leq b$ are linear combinations of equalities from $D'x = c'$. Based on Gaussian elimination, we will represent $D'x = c'$ as an equivalent system of equalities $y - Dz = c$ such that $y = (y_1, \ldots, y_{n_y})^T$ and $z = (z_1, \ldots, z_{n_z})^T$ are a partition of the variables in x, $D \in \mathbb{Q}^{n_y \times n_z}$, and $c \in \mathbb{Q}^{n_y}$. This form represents a distinct substitution that replaces variable y_i with $c_i + d_i^T z$. With the substitution we can directly check whether an equality $h^T x = g$ is a linear combination of $y - Dz = c$ and, therefore, implied by both $Ax \leq b$ and $y - Dz = c$. We simply apply the substitution to $h^T x = g$ and see if it simplifies to $0 = 0$. Since swapping the columns in A and x^T does not influence the satisfiable assignments for x in $Ax \leq b$, we assume without loss of generality that $(x_1, \ldots, x_n) = (y_1, \ldots, y_{n_y}, z_1, \ldots, z_{n_z})$.

Algorithm 1: `EqualityBasis`$(Ax \leq b)$

Input : A satisfiable system of inequalities $Ax \leq b$, where $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}_\delta^m$
Output: An equality basis $Dx = c$ for $Ax \leq b$

$z := (x_1, \ldots, x_n)^T$
$D := () \in \mathbb{Q}^{0 \times 0}, c := () \in \mathbb{Q}^0, y := () \in \mathbb{Q}^0$
Remove all rows $a_i^T z \leq b$ from $Az \leq b$ such that $a_i = \mathbf{0}$ and $b_i = 0$
while $Az \leq b^\delta$ is unsatisfiable (i.e., $Az \leq b$ contains an equality) **do**
  Let C be an explanation for $Az \leq b^\delta$ being unsatisfiable
  Select $(a_i^T z \leq b_i^\delta) \in C$; `// by Lemma 8,` $a_i^T z = b_i$ `is implied by` $Az \leq b$
  `/* Without loss of generality we can assume that` $a_{i1} \neq 0$. `*/`
  `/* Otherwise, we would just swap the columns accordingly. */`
  $(y', z', A'z' \leq b', y' - D'z' = c') := \texttt{Elim}(y, z, Az \leq b, y - Dz = c, a_i^T z = b_i)$
  $(y, z, Az \leq b, y - Dz = c) := (y', z', A'z' \leq b', y' - D'z' = c')$
end
return $y - Dz = c$

Figure 1: `EqualityBasis` computes an equality basis

Algorithm 2: `Elim`$(y, z, Az \leq b, y - Dz = c, h^T z = g)$

Input : Two variable vectors y and z, a system of inequalities $Az \leq b$, a system of equalities $y - Dz = c$, and an equality $h^T z = g$ implied by $Az \leq b$, where $h_1 \neq 0$
Output: The tuple $(y', z', A'z' \leq b', y' - D'z' = c')$, where $y' = (y_1, \ldots, y_{n_y}, z_1)^T$, $z' = (z_2, \ldots, z_{n_z})^T$, and the combined systems $(Az \leq b) \cup (y - Dz = c)$ and $(A'z' \leq b') \cup (y' - D'z' = c')$ have the same solutions

Replace z_1 in $Az \leq b$ with $z_1 := \frac{1}{h_1}(g - \sum_{j=2}^{n_z} h_j z_j)$
`/* by subtracting` $\frac{a_{i1}}{h_1} h^T z = \frac{a_{i1}}{h_1} g$ `from every inequality` $a_i^T z \leq b_i$ `*/`
Replace z_1 in $y - Dz = c$ with $z_1 := \frac{1}{h_1}(g - \sum_{j=2}^{n_z} h_j z_j)$
`/* by adding` $\frac{d_{i1}}{h_1} h^T z = \frac{d_{i1}}{h_1} g$ `to every equality` $y_i - d_i^T z = c_i$ `*/`
Add the row $z_1 - (\sum_{j=2}^{n_z} \frac{-h_j}{h_1} z_j) = \frac{g}{h_1}$ to the end of $y - Dz = c$
Remove all rows $a_i^T z \leq b_i$ from $Az \leq b$ such that $a_i = \mathbf{0}$ and $b_i = 0$
Remove the first column of $Az \leq b$, which contains only zeros due to our substitutions
$y := (y_1, \ldots, y_{n_y}, z_1)^T$, $z := (z_2, \ldots, z_{n_z})^T$
return $(y, z, Az \leq b, y - Dz = c)$

Figure 2: `Elim` removes an equality $h^T z = g$ from $Az \leq b$

The algorithm `EqualityBasis`$(Ax \leq b)$ (Figure 1) computes an equality basis $y - Dz = c$ for the set of inequalities $Ax \leq b$. To this end, `EqualityBasis` splits our system of inequalities into a system of inequalities $Az \leq b$ and a system of equalities $y - Dz = c$. While the variables z are completely defined by $Az \leq b$, $y - Dz = c$ is used to extend any assignment from the variables z to the variables y. Initially, z is just x, $y - Dz = c$ is empty, and $Az \leq b$ is just $Ax \leq b$. However, in every iteration of the while loop, `EqualityBasis` eliminates one equality $a_i^T z = b_i$ from $Az \leq b$ and adds it to $y - Dz = c$. `EqualityBasis` finds this equality based on

Algorithm 3: $\texttt{Subst}(y - Dz = c, Ax \leq b)$

Input : A system of equalities $y - Dz = c$, and a system of inequalities $Ax \leq b$ such that $(x_1, \ldots, x_n) = (y_1, \ldots, y_{n_y}, z_1, \ldots, z_{n_z})$

Output: $A'z \leq b'$, the system of inequalities that remains after we have eliminated the variables y from $Ax \leq b$ with the equations in $y - Dz = c$

1 $z' := (x_1, \ldots, x_n)^T, D' := () \in \mathbb{Q}^{0\times 0}, c' := () \in \mathbb{Q}^0, y' := () \in \mathbb{Q}^0$
2 **for** $i = 1, \ldots, n_y$ **do**
3 $\quad (y', z', Az' \leq b, y' - D'z' = c') := \texttt{Elim}(y', z', Az' \leq b, y' - D'z' = c', y_i - d_i^T z = c_i)$
4 **end**
5 **return** $Az \leq b$

Figure 3: `Subst` eliminates variables y from $Ax \leq b$ with the equations in $y - Dz = c$

the techniques we presented in the Lemmas 7 & 8. Then, `EqualityBasis` uses the algorithm `Elim` from Figure 2 to eliminate $a_i^T z = b_i$ from $Az \leq b$ and to add it to $y - Dz = c$. Note that we assume for this algorithm that $a_{i1} \neq 0$. Naturally, this is not always the case, but we could assure it by swapping columns. We refrain from doing so explicitly to simplify the algorithm.

The results of applying $\texttt{Elim}(y, z, Az \leq b, y - Dz = c, h^T z = g)$ are a new system of inequalities $A'z' \leq b'$, a new system of equalities $y' - D'z' = c'$, and a new partition of our variables $y' = (y_1, \ldots, y_{n_y}, z_1)^T$ and $z' = (z_2, \ldots, z_{n_z})^T$. Due to the new partition, the variable $y'_{n_y+1} := z_1$ is no longer defined by the inequalities $A'z' \leq b'$, but it is defined by the equalities $y' - D'z' = c'$. However, `Elim` conserves the equivalence of the split systems, i.e., the combination of the input systems $(Az \leq b) \cup (y - Dz = c)$ and the combination of the output systems $(A'z' \leq b') \cup (y' - D'z' = c')$ are equivalent.

Lemma 9. *Let $Az \leq b$ be a system of inequalities. Let $y - Dz = c$ be a system of equalities. Let $h^T z = g$ be an equality implied by $Az \leq b$. Let $(y', z', A'z' \leq b', y' - D'z' = c') := \texttt{Elim}(y, z, Az \leq b, y - Dz = c, h^T z = g)$, where $y' = (y_1, \ldots, y_{n_y}, z_1)^T$, and $z' = (z_2, \ldots, z_{n_z})^T$. Let $u \in \mathbb{Q}^{n_y}$, $v \in \mathbb{Q}^{n_z}$, $u' = (u_1, \ldots, u_{n_y}, v_1)^T$, and $v' = (v_2, \ldots, v_{n_z})^T$ Then, $(Av \leq b) \cup (u - Dv = c)$ is true if and only if $(A'v' \leq b') \cup (u' - D'v' = c')$ is true.*

Proof. Assume that $(Av \leq b) \cup (u - Dv = c)$ is true. Since $Az \leq b$ implies $h^T z = g$, $h^T v = g$ is also true. We get $(A'v' \leq b') \cup (u' - D'v' = c')$ by subtracting multiples of $h^T v = g$ from $(Av \leq b) \cup (u - Dv = c)$, which simplifies to subtracting $0 = 0$ because $h^T v = g$ is true. Therefore, we have proven that $(A'v' \leq b') \cup (u' - D'v' = c')$ is true.

Assume that $(A'v' \leq b') \cup (u' - D'v' = c')$ is true. Since the last row of $y' - D'z' = c'$ is just $\frac{h^T z}{h_1} = \frac{g}{h_1}$ (see line 3 in Fig. 2), $h^T v = g$ is true. We get $(Av \leq b) \cup (u - Dv = c)$ by adding multiples of $h^T v = g$ to $(A'v' \leq b') \cup (u' - D'v' = c')$, which simplifies to adding $0 = 0$ because $h^T v = g$ is true. Therefore, we have also proven that $(Av \leq b) \cup (u - Dv = c)$ is true. □

After removing all equalities from $Ax \leq b$ the algorithm $\texttt{EqualityBasis}(Ax \leq b)$ terminates resulting in a system of inequalities $A'z \leq b'$ implying no equalities and a system of equalities $y - Dz = c$ that is an equality basis of the original system $Ax \leq b$. The algorithm is guaranteed to terminate because every call to `Elim` moves one variable from the vector z to the vector y. Note that $\texttt{EqualityBasis}(Ax \leq b)$ does not return the final system of inequalities $A'z \leq b'$. However, we can obtain it with the help of a third algorithm $\texttt{Subst}(y - Dz = c, Ax \leq b)$ (Figure 3). The algorithm $\texttt{Subst}(y - Dz = c, Ax \leq b)$ eliminates the variables y_i by substituting them with $c + d_i^T z$, which also eliminates the equalities $y - Dz = c$ from $Ax \leq b$ the same

way `EqualityBasis`($Ax \leq b$) did. Since `Subst` is based on `Elim`, `Subst` is also equivalence preserving. A fact that we will exploit to prove the correctness of `EqualityBasis`($Ax \leq b$).

Lemma 10. *Let $y - Dz = c$ be a satisfiable system of equalities. Let $Ax \leq b$ and $A^*x \leq b^*$ be two systems of inequalities, both implying the equalities in $y - Dz = c$. Let $A'z \leq b'$ be the output of* `Subst`$(y - Dz = c, Ax \leq b)$. *Let $A^{**}z \leq b^{**}$ be the output of* `Subst`$(y - Dz = c, A^*x \leq b^*)$. *Then, $A'z \leq b'$ is equivalent to $A^{**}z \leq b^{**}$ if $Ax \leq b$ is equivalent to $A^*x \leq b^*$.*

Proof. Let $Ax \leq b$ be equivalent to $A^*x \leq b^*$. Suppose to the contrary that $A'z \leq b'$ is not equivalent to $A^{**}z \leq b^{**}$. This means that there exists a $v \in \mathbb{Q}^{n_z}$ such that either $A'v \leq b'$ is true and $A^{**}v \leq b^{**}$ is false, or $A'v \leq b'$ is false and $A^{**}v \leq b^{**}$ is true. Without loss of generality we select the first case that $A'v \leq b'$ is true and $A^{**}v \leq b^{**}$ is false. We now extend this solution by $u \in \mathbb{Q}^{n_y}$, where $u_i := c_i + d_i^T v$, so $(A'v \leq b') \cup (u - Dv = c)$ is true. However, based on Lemma 9 and the transitivity of equivalence, the four systems of constraints $Ax \leq b$, $A^*x \leq b^*$, $(A'z \leq b') \cup (y - Dz = c)$, and $(A^{**}z \leq b^{**}) \cup (y - Dz = c)$ are equivalent. Therefore, $(A^{**}v \leq b^{**}) \cup (u - Dv = c)$ is true, which means that $A^{**}v \leq b^{**}$ is also true. The latter contradicts our initial assumptions. □

We also use `Subst` to determine whether a system of equalities implies (implicitly or explicitly) an equality $h^Tx = g$. Again this is based on the fact that $Dx = c$ implies an equality if and only if the equality can be obtained by a linear combination from $Dx = c$. However, with `Subst` we turn $Dx = c$ into a substitution and can directly check whether $h^Tx = g$ is a linear combination from $Dx = c$. We simply apply the substitution to $h^Tx = g$ and see if it simplifies to $0 = 0$. In this case, line 4 of `Elim` eliminates the equality.

Lemma 11. *Let $y - Dz = c$ be a satisfiable system of equalities. Let $h^Tx = g$ be an equality. Then, $h^Tx = g$ is implied by $y - Dz = c$ if and only if* `Subst`$(y - Dz = c, h^Tx = g) = \emptyset$.

Proof. First of all, let us look at the case where $h^Tx = g$ is an explicit equality $y_i - d_i^T z = c_i$ in $y - Dz = c$. Then clearly, $y_i - d_i^T z = c_i$ will be eliminated when we call `Elim` with $y_i - d_i^T z = c_i$ in line 3 of `Subst`. Therefore, the empty set is the output of both `Subst`$(y - Dz = c, y_i - d_i^T z = c_i)$ and `Subst`$(y - Dz = c, y - Dz = c)$.

Now, let us look at the case where $h^Tx = g$ is an implicit equality in $y - Dz = c$. Since both $y - Dz = c$ and $(y - Dz = c) \cup (h^Tz = g)$ imply $h^Tz = g$ and the equalities in $y - Dz = c$, the output of both `Subst`$(y - Dz = c, y - Dz = c)$ and `Subst`$(y - Dz = c, (y - Dz = c) \cup (h^Tz = g))$ must be equivalent (see Lemma 10). Therefore, the output of `Subst`$(y - Dz = c, (y - Dz = c) \cup (h^Tz = g))$ can only be the empty set. Hence, the output of `Subst`$(y - Dz = c, h^Tz = g)$ is also the empty set.

Finally, let us look at the case where $h^Tx = g$ is not an equality implied by $y - Dz = c$. Suppose to the contrary that the output of `Subst`$(y - Dz = c, h^Tz = g)$ is the empty set. We know based on Lemma 9 and transitivity of equivalence that $(y - Dz = c) \cup (h^Tz = g)$ and $(y - Dz = c) \cup \emptyset$ are equivalent. Therefore, $h^Tz = g$ is implied by $y - Dz = c$, which contradicts our initial assumption. □

In case $y - Dz = c$ is an equality basis of $Ax \leq b$, we can use the above Lemma 11 to check whether $Ax \leq b$ implies $h^Tz = g$. Naturally, $Ax \leq b$ implies $h^Tz = g$ if and only if `Subst`$(y - Dz = c, h^Tz = g) = \emptyset$.

With the help of the Lemmas 10 & 11, we are also able to prove that `EqualityBasis`($Ax \leq b$) computes an actual equality basis.

Lemma 12. *Let $Ax \leq b$ be a satisfiable system of inequalities. Let $y - Dz = c$ be the output of* `EqualityBasis`$(Ax \leq b)$. *Then $y - Dz = c$ is an equality basis of $Ax \leq b$.*

Proof. Let $A'z \leq b'$ be the output of $\texttt{Subst}(y - Dz = c, Ax \leq b)$. Since $y - Dz = c$ is the output of $\texttt{EqualityBasis}(Ax \leq b)$, the condition in line 4 of `EqualityBasis` guarantees us that $A'z \leq b'$ implies no equalities. Now, suppose to the contrary of our initial assumptions that $Ax \leq b$ implies an equality $h^T x = g$ that $y - Dz = c$ does not imply. Since $h^T x = g$ is not implied by $y - Dz = c$, the output of $\texttt{Subst}(y - Dz = c, h^T x = g)$ is an equality ${h'}^T z = g'$, where $h' \neq \mathbf{0}$. This also implies that $(A'z \leq b') \cup ({h'}^T z = g')$ is the output of $\texttt{Subst}(y - Dz = c, (Ax \leq b) \cup (h^T x = g))$. By Lemma 10, $A'z \leq b'$ and $(A'z \leq b') \cup ({h'}^T z = g')$ are equivalent. Therefore, $A'z \leq b'$ implies the equality ${h'}^T z = g'$, which contradicts the condition in line 4 of `EqualityBasis` and, therefore, our initial assumptions. □

5 Conclusions

Through Lemmas 7 & 8, we have presented a method that efficiently checks whether a system of linear arithmetic constraints implies an equality at all. We use this method in the algorithm $\texttt{EqualityBasis}(Ax \leq b)$ to compute an equality basis $y - Dz = c$. With the algorithm $\texttt{Subst}(y - Dz = c, Ax \leq b)$, we have also presented an algorithm that simplifies a system of linear arithmetic constraints $Ax \leq b$ by eliminating all equalities $y - Dz = c$ from $Ax \leq b$ via substitution. Moreover, we explained how to use an equality basis $y - Dz = c$ of $Ax \leq b$ and $\texttt{Subst}(y - Dz = c, h^T x = g)$ to check whether $Ax \leq b$ implies a given equality $h^T x = g$.

There already exist several methods that find, isolate, and eliminate implied equalities [1, 10, 11, 12]. Hentenryck and Graf [12] define unique normal forms for systems of linear constraints with non-negative variables. To compute a normal form, they first eliminate all implied equalities $a_i^T x = b_i$ from $Ax \leq b$. To this end, they determine the lower bound for each inequality $a_i^T x \leq b_i$ in $Ax \leq b$ by solving one optimization linear program for each inequality. Similarly, Refalo [10] describes several incremental methods that turn a satisfiable system of linear constraints in "revised solved form" into a system without any implied equalities. He also uses optimization to detect and to eliminate implied equalities. The method presented by Telgen [11] does not require optimization. He presents criteria to detect implied equalities based on the tableau used in the simplex algorithm. However, Telgen was not able to formulate an algorithm that efficiently computes these criteria. In the worst case, he has to pivot the simplex tableau until he has computed all possible tableaux for the given system of constraints. Another method that detects implied equalities was presented by Bjørner [1]. He uses Fourier Motzkin variable elimination to compute linear combinations that result in implied equalities.

Our methods do not require optimization, which SMT solvers are usually not fine-tuned for. Furthermore, we have defined our methods for a rather general formulation of linear constraints, which makes it very easy to convert our results into other representations, e.g., the tableau-and-bound representation used in Dutertre and de Moura's version of the simplex algorithm (see Appendix B). Finally, our method efficiently searches for implied equalities. We neither have to check each inequality independently nor do we have to blindly pivot the simplex tableau.

Our methods can be implemented as an extension of Dutertre and de Moura's simplex algorithm [5]. The input constraints for this algorithm are represented by a so-called tableau $Ax = \mathbf{0}$ and two bounds $l_i \leq x_i \leq u_i$ for every variable x_i in the tableau. The variables are also partitioned into two sets: the non-basic variables and the basic variables. We find a satisfiable assignment by updating an intermediate assignment β. These updates are typically accompanied by a pivot of the tableau, i.e., by performing substitutions in the tableau, a non-basic variable and a basic variable are swapped.

It is also possible to transform our system of inequalities $A'x' \leq b'$ into this tableau-and-

bound representation with the help of slack variables. However, this transformation also means that other representations have to be adjusted accordingly. For example, an equality basis is no longer a system of equalities but a tableau $Ax = \mathbf{0}$ and a set of tightly bounded variables x_i such that $l_i = u_i$. This means that our goal shifts from finding equalities $a_i^T x = b_i$ to finding tightly bounded variables. It also means that turning the bounds l_i and u_i of every variable x_i into strict bounds is enough to apply Lemma 7.

The tableau-and-bound representation also grants us several advantages for the implementation of our methods. For example, we do not have to explicitly eliminate variables via substitution. As mentioned in Section 3, the specifics of the implementation will also contain justified simplifications for Lemma 7. For further details on the implementation, see the Appendix.

For future research, we plan to investigate the above methods and their performance as a preprocessing step for our cube tests [3], and as a certifying algorithm for the equalities transferred as part of the Nelson-Oppen style combination of theories [4].

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments, suggestions, and for directing us to related work.

References

[1] N. Bjørner. *Integrating Decision Procedures for Temporal Verification*. PhD thesis, Stanford, CA, USA, 1999. AAI9924398.

[2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[3] M. Bromberger and C. Weidenbach. Fast cube tests for lia constraint solving. In N. Olivetti and A. Tiwari, editors, *IJCAR 2016*, volume 9706 of *LNCS*. Springer, 2016.

[4] R. Bruttomesso, A. Cimatti, A. Franzen, A. Griggio, and R. Sebastiani. Delayed theory combination vs. nelson-oppen for satisfiability modulo theories: a comparative analysis. *AMAI*, 55(1):63–99, 2009.

[5] B. Dutertre and L. de Moura. A fast linear-arithmetic solver for dpll(t). In T. Ball and R. B. Jones, editors, *CAV*, volume 4144 of *LNCS*, pages 81–94. Springer, 2006.

[6] G. Faure, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Sat modulo the theory of linear arithmetic: Exact, inexact and commercial solvers. In H. Kleine Büning and X. Zhao, editors, *SAT 2008*, volume 4996 of *LNCS*, pages 77–90. Springer, 2008.

[7] A. Griggio. A practical approach to satisfiability modulo linear integer arithmetic. *JSAT*, 8(1/2):1–27, 2012.

[8] R. A. Horn and C. R. Johnson. Norms for vectors and matrices. In *Matrix Analysis*, pages 313–386. Cambridge University Press, second edition, 2012. Cambridge Books Online.

[9] M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors. *50 Years of Integer Programming 1958-2008*. Springer, 2010.

[10] P. Refalo. Approaches to the incremental detection of implicit equalities with the revised simplex method. In C. Palamidessi, H. Glaser, and K. Meinke, editors, *PLILP 1998*, volume 1490 of *LNCS*, pages 481–496. Springer, 1998.

[11] J. Telgen. Identifying redundant constraints and implicit equalities in systems of linear constraints. *Management Science*, 29(10):1209–1222, 1983.

[12] P. Van Hentenryck and T. Graf. Standard forms for rational linear arithmetic in constraint logic programming. *AMAI*, 5(2):303–319, 1992.

Appendix

A The Dual Simplex Algorithm

In the next section of the appendix, we present a guideline for an implementation of the above methods as an extension of the dual simplex algorithm by Dutertre and de Moura [5]. Whenever we will refer to the simplex algorithm in this appendix, we will refer to the specific version of the dual simplex algorithm presented by Dutertre and de Moura [5].

The simplex algorithm accepts only linear arithmetic systems that are defined by a system of equalities $Ax = \mathbf{0}$ and a set of bounds for the variables $l_j \leq x_j \leq u_j$ (for $j = 1, \ldots, n$). If there is no lower bound $l_j \in \mathbb{Q}_\delta$ for variable x_j, then we simply set $l_j = -\infty$. Similarly, if there is no upper bound $u_j \in \mathbb{Q}_\delta$ for variable x_j, then we simply set $u_j = \infty$.

We can easily transform a system of inequalities $Ax \leq b$ into the above format if we introduce a so-called slack variable s_i for every inequality in our system. Our system is then defined by the equalities $Ax - s = \mathbf{0}$, and the bounds $-\infty \leq x_j \leq \infty$ for every original variable x_j and the bounds $-\infty \leq s_i \leq b_i$ for every slack variable introduced for the inequality $a_i^T x \leq b_i$. We can even reduce the number of slack variables if we transform rows of the form $a_{ij} \cdot x_j \leq c_j$ directly into bounds for x_j. Moreover, we can use the same slack variable for multiple inequalities as long as the left side of the inequality is similar enough. For example, the inequalities $a_i^T x \leq b_i$ and $-a_i^T x \leq c_i$ can be transformed into the equality $a_i^T x - s_i = 0$ and the bounds $-c_i \leq s_i \leq b_i$. For more details on these simplifications we refer to [5].

The simplex algorithm also partitions the variables into two sets: the set of *non-basic* variables $\mathcal{N}$ and the set of *basic* variables $\mathcal{B}$. Initially, our original variables are the non-basic variables and the slack variables are the basic variables. The non-basic variables $\mathcal{N}$ define the basic variables over a *tableau* derived from our system of equalities. Each row in this tableau represent one basic variable $x_i \in \mathcal{B}$: $x_i = \sum_{x_j \in \mathcal{N}} a_{ij} x_j$. The simplex algorithm exchanges variables from $x_i \in \mathcal{B}$ and $x_j \in \mathcal{N}$ with the `pivot` algorithm (Figure 4). To do so, we also have to change the tableau via substitution. All tableaux constructed in this way are equivalent to the original system of equalities $Ax = \mathbf{0}$.

The goal of the simplex algorithm is to find an assignment β that maps every variable x_i to a value $\beta(x_i) \in \mathbb{Q}_\delta$ that satisfies our constraint system, i.e., $A(\beta(x)) = \mathbf{0}$ and $l_i \leq \beta(x_i) \leq u_i$ for every variable x_i. The algorithm starts with an assignment β that fulfils $A(\beta(x)) = \mathbf{0}$ and $l_j \leq \beta(x_j) \leq u_j$ for every non-basic variable $x_j \in \mathcal{N}$. Initially, we get such an assignment through our tableau. We simply choose a value $l_j \leq \beta(x_j) \leq u_j$ for every non-basic variable $x_j \in \mathcal{N}$ and define the value of every basic variable $x_i \in \mathcal{B}$ over the tableau: $\beta(x_i) := \sum_{x_j \in \mathcal{N}} a_{ij} \beta(x_j)$. As an invariant, the simplex algorithm will continue to fulfil $A(\beta(x)) = \mathbf{0}$ and $l_j \leq \beta(x_j) \leq u_j$ for every non-basic variable $x_j \in \mathcal{N}$ and every intermediate assignment β.

The simplex algorithm finds a satisfiable assignment or an explanation of unsatisfiability through the `Check`() algorithm (Figure 4). Since all non-basic variables fulfil their bounds and the tableau guarantees that $Ax = \mathbf{0}$, `Check`() only looks for a basic variable that violates one of its bounds. If all basic variables x_i satisfy their bounds, then β is already a satisfiable assignment and `Check`() can return true. If `Check`() finds a basic variable x_i that violates one of its bounds, then it looks for a non-basic variable x_j it can pivot with. We fulfil our invariant again after an update of our β assignment that sets $\beta(x_i)$ to the previously violated bound value. However, not all non-basic variables can be used for pivoting. If every non-basic variable violates the conditions in lines 6 & 12 of `Check`(), then the row of x_i and all non-basic variables x_j with $a_{ij} \neq 0$ build an unresolvable conflict. Hence, `Check`() has found a row that explains the conflict and it can return unsatisfiable.

Algorithm 4: `pivot`(x_i, x_j)

Input : A basic variable x_i and a non-basic variable x_j so that a_{ij} is non-zero
Effect : Transforms the tableau so x_i becomes non-basic and x_j basic

Let $x_i = \sum_{k \in \mathcal{N}} a_{ik} x_k$ be the row defining the basic variable x_i
We rewrite this row as $x_j = \frac{1}{a_{ij}} x_i - \sum_{k \in \mathcal{N} \setminus \{x_j\}} \frac{a_{ik}}{a_{ij}} x_k$ so it defines x_j instead
foreach $x_k \in \mathcal{N} \setminus \{x_j\}$ **do** $a_{jk} := -\frac{a_{ik}}{a_{ij}}$;
$a_{ji} := \frac{1}{a_{ij}}$
Then, we substitute x_j in all other rows with $\frac{1}{a_{ij}} x_i - \sum_{k \in \mathcal{N} \setminus \{x_j\}} \frac{a_{ik}}{a_{ij}} x_k$
for $x_l \in \mathcal{B}$ **do**
    foreach $x_k \in \mathcal{N} \setminus \{x_j\}$ **do** $a_{lk} := a_{lk} + a_{lj} a_{jk}$;
    $a_{li} := a_{lj} a_{ji}$; $a_{lj} := 0$
end
$\mathcal{N} = (\mathcal{N} \cup \{x_i\}) \setminus \{x_j\}$; $\mathcal{B} = (\mathcal{B} \cup \{x_j\}) \setminus \{x_i\}$

Algorithm 5: `update`(x_j, v)

Input : A non-basic variable x_j and a value $v \in \mathbb{Q}_\delta$
Effect : Sets the value $\beta(x_j)$ of x_j to v and updates the values of all basic variables

foreach $x_i \in \mathcal{B}$ **do** $\beta(x_i) := \beta(x_i) + a_{ij}(v - \beta(x_j))$;
$\beta(x_j) := v$

Algorithm 6: `pivotAndUpdate`(x_i, x_j, v)

Input : A basic variable x_i, a non-basic variable x_j, and a value $v \in \mathbb{Q}_\delta$
Effect : Pivots variables x_i and x_j and updates the value $\beta(x_i)$ of x_i to v

$\theta := \frac{v - \beta(x_i)}{a_{ij}}$
$\beta(x_i) := v$; $\beta(x_j) := \beta(x_j) + \theta$
foreach $x_k \in \mathcal{B} \setminus \{x_i\}$ **do** $\beta(x_k) := \beta(x_k) + a_{kj}\theta$;
`pivot`(x_i, x_j)

Algorithm 7: `Check`()

Output : Returns *true* iff there exists a satisfiable assignment for the tableau and the bounds u and l; otherwise, it returns (*false*,x_i), where x_i is the conflicting basic variable

while *true* **do**
    select the smallest basic variable x_i such that $\beta(x_i) < l_i$ or $\beta(x_i) > u_i$
    if there is no such x_i **then return** *true*;
    if $\beta(x_i) < l_i$ **then**
        select the smallest non-basic variable x_j such that
        ($a_{ij} > 0$ and $\beta(x_j) < u_j$) or ($a_{ij} < 0$ and $\beta(x_j) > l_j$)
        if there is no such x_j **then return** (*false*,x_i);
        `pivotAndUpdate`(x_i, x_j, l_i)
    end
    if $\beta(x_i) > u_i$ **then**
        select the smallest non-basic variable x_j such that
        ($a_{ij} < 0$ and $\beta(x_j) < u_j$) or ($a_{ij} > 0$ and $\beta(x_j) > l_j$)
        if there is no such x_j **then return** (*false*,x_i);
        `pivotAndUpdate`(x_i, x_j, u_i)
    end
end

Figure 4: The functions of the dual simplex algorithm by Dutertre and de Moura [5]

The algorithm is guaranteed to terminate due to a variable selection strategy called Bland's rule. Bland's rule is based on a predetermined variable order and always selects the smallest variables fulfilling the conditions for pivoting.

B Specifics for an Implementation

An *equality basis* for a satisfiable system of inequalities $Ax \leq b$ is a system of equalities $Dx = c$ such that all (explicit and implicit equalities) implied by $Ax \leq b$ are linear combinations of equalities from $Dx = c$. In case of the tableau-and-bound representation of the simplex algorithm, an equality basis simplifies to a tableau and a set of *tightly* bounded variables, i.e., $\beta(x_j) := l_j$ or $\beta(x_j) := u_j$ for all satisfiable assignments β. Therefore, one way of determining an equality basis is to find all tightly bounded variables.

To find all tightly bounded variables, we present a new extension of the simplex algorithm called `FindTightBounds`() (Figure 5). This extension uses our Lemmas 7 & 8 to iteratively find all bounds $l_j \leq x_j$ ($x_j \leq u_j$) that hold tightly for all satisfiable assignments β, and then turns them into explicit equalities by setting $u_j := l_j$ ($l_j := u_j$). But first of all, `FindTightBounds`() determines if our constraint system is actually satisfiable with a call of `Check`(). If the system is unsatisfiable, then it has no solutions and, therefore, implies all equalities. In this case, `FindTightBounds`() stops an returns *false*.

Otherwise, we get a satisfiable assignment β from `Check`() and we use this assignment in `Initialize`() (Figure 5) to eliminate all bounds that do not hold tightly under β (i.e., $\beta(x_i) > l_i$ or $\beta(x_i) < u_i$). We know that we can eliminate these bounds without losing any tightly bounded variables because we only need the bounds that can be part of an equality explanation, i.e., only bounds that hold tightly for all satisfiable assignments (see Lemma 8). For the same reason, `Initialize`() eliminates all originally strict bounds, i.e., bounds with a non-zero delta part.

Next, `Initialize`() tries to turn as many variables x_i with $l_i = u_i$ into non-basic variables. We do so because x_i is guaranteed to stay a non-basic variable if $l_i = u_i$ (see lines 6 & 12 of `Check`). Pivoting like this essentially eliminates the tightly bounded non-basic variable x_i and replaces it with the constant value l_i. There only exists one case when `Initialize`() cannot turn the variable x_i with $l_i = u_i$ into a non-basic variable. This case occurs whenever all non-basic variables x_j with non-zero coefficient a_{ij} also have tight bounds $l_j = u_j$. However, in this case the complete row $x_i = \sum_{x_j \in \mathcal{N}} a_{ij} x_j$ simplifies to $x_i = l_i$, so it will never produce a conflict and we can also ignore this row.

As its last action, `Initialize`() turns the bounds of all variables x_j with $l_j < u_j$ into strict bounds. Since `Initialize`() transformed these bounds into strict bounds, the condition of the while loop in line 3 of `FindTightBounds`() checks whether the system contains another tightly bounded variable (see also Lemma 7). If `Check` returns (*false*, x_i), then the row x_i represents an equality explanation and all variables x_j with a non-zero coefficient in the row hold tightly (see Lemma 8). `FindTightBounds`() uses `FixEqualities`(x_i) (Figure 5) to turn these tightly bounded variables x_j into explicit equalities. `FixEqualities`(x_i) simply sets $l_j = u_j$ and, thereby, the variable x_j is (essentially) replaced with the constant value l_j. After `FixEqualities`(x_i) is done fixing the tightly bounded variables, we go back to the beginning of the loop in `FindTightBounds`() and do another call to `Check`.

If `Check` returns *true*, then the original system of inequalities implies no further tightly bounded variables (Lemma 7). We exit the loop and revert the bounds of the remaining variables x_j with $l_j < u_j$ to their original values. As a result, we have also reverted to a linear system equivalent to our original constraint system. The only difference is that now all tightly bounded variables x_i are explicit equalities, i.e., $l_i = u_i$, and represent an equality basis for our

Algorithm 8: `Initialize`()

Effect : Removes all bounds l_k and u_k that cannot produce equalities; turns as many basic variables x_i with $l_i = u_i$ into non-basic variables as is possible; the bounds for all variables x_k are turned into strict bounds if $l_k < u_k$

for $x_k \in \mathcal{B} \cup \mathcal{N}$ **do**
  if $\beta(x_k) > l_k$ **then** $l_k := -\infty$;
  if $\beta(x_k) < u_k$ **then** $u_k := +\infty$;
  if $\beta(x_k) = p_k + q_k\delta$ such that $q_k \neq 0$ **then** $l_k := -\infty; u_k := \infty$;
end
for $x_i \in \mathcal{B}$ **do**
  if $l_i = u_i$ **then**
    select the smallest non-basic variable x_j such that a_{ij} is non-zero and $l_j < u_j$
    if there is such an x_j **then** `pivot`(x_i, x_j);
  end
end
for $x_k \in \mathcal{B} \cup \mathcal{N}$ **do**
  if $l_k < u_k$ **then**
    if $l_k \neq -\infty$ **then** $l_k := l_k + \delta$;
    if $u_k \neq +\infty$ **then** $u_k := u_k - \delta$;
    if $l_k \neq -\infty$ and $x_k \in \mathcal{N}$ **then** `update`(x_k, l_k);
    if $u_k \neq +\infty$ and $x_k \in \mathcal{N}$ **then** `update`(x_k, u_k);
  end
end

Algorithm 9: `FixEqualities`(x_i)

Input : A basic variable x_i that explains the conflict
Effect : Turns the bounds of all variables responsible for the conflict into equalities

for $x_j \in \mathcal{N}$ **do**
  if $l_j < u_j$ **then**
    if $\beta(x_i) < l_i$ and $a_{ij} > 0$ **then** $u_j := u_j + \delta$; $l_j := u_j$; `update`(x_j, u_j);
    if $\beta(x_i) < l_i$ and $a_{ij} < 0$ **then** $l_j := l_j - \delta$; $u_j := l_j$; `update`(x_j, l_j);
    if $\beta(x_i) > u_i$ and $a_{ij} > 0$ **then** $l_j := l_j - \delta$; $u_j := l_j$; `update`(x_j, l_j);
    if $\beta(x_i) > u_i$ and $a_{ij} < 0$ **then** $u_j := u_j + \delta$; $l_j := u_j$; `update`(x_j, u_j);
  end
end
if $\beta(x_i) > u_i$ **then** $u_i := u_i + \delta$; $l_i := u_i$;
if $\beta(x_i) < l_i$ **then** $l_i := l_i - \delta$; $u_i := l_i$;

Algorithm 10: `FindTightBounds`()

Effect : Finds as many tightly bounded variables as possible
Output : *false* iff the system of linear arithmetic constraints is unsatisfiable

if `Check`() returns (*false*,x_i) **then return** *false*;
`Initialize`()
while `Check`() returns (*false*,x_i) **do**
  `FixEqualities`(x_i)
end
For all variables x_i with $l_i < u_i$ recover their old bounds
return *true*

Figure 5: The functions used to turn our original tableau into a basis of equalities

Algorithm 11: `IsImplied`($h^T x = g$)

Input : An equality $h^T x = g$
Output : Returns *true* iff $h^T x = g$ is implied by our system of constraints

```
1  if FindTightBounds() returns false then return true;
2  for i := 1, ..., n do
3  |  if x_i ∈ 𝒩 and l_i = u_i then g := g − h_i · l_i;   h_i := 0;
4  |  if x_i ∈ ℬ then
5  |  |  for x_j ∈ 𝒩 do
6  |  |  |  if l_j = u_j then g := g − a_ij · h_i · l_j;
7  |  |  |  if l_j ≠ u_j then h_j := h_j + a_ij · h_i;
8  |  |  end
9  |  end
10 end
11 return (h = 0 and g = 0)
```

Figure 6: `IsImplied` checks whether $h^T x = g$ is implied by our system of inequalities

original constraint system. Hence, we can continue with other computations on the tableau without losing our equality basis.

The algorithm `FindTightBounds`() terminates because every call to `FixEqualities`(x_i) simplifies the row $x_i = \sum_{x_j \in \mathcal{N}} a_{ij} x_j$ to $x_i = l_i$. Hence, the row will never again be an equality explanation and the number of iterations we spent in the while loop of `FindTightBounds`() is bounded by the number of variables.

Finally, we present `IsImplied`($h^T x = g$), an algorithm that uses the equality basis to detect whether our original system of constraints implies the equality $h^T x = g$. The algorithm substitutes all basic variables x_i in $h^T x = g$ with their row definition $\sum_{x_j \in \mathcal{N}} a_{ij} x_j$. The algorithm also replaces every tightly bounded non-basic variable x_j with the constant bound value c_j it is tightly bounded to. As a result of these substitutions, $h^T x = g$ simplifies to $\mathbf{0}^T x = 0$ if and only if $h^T x = g$ is implied by our system of constraints. Note that the call to `FindTightBounds`() in line 1 of `IsImplied` is not necessary if we made the call once before.