New Techniques for Linear Arithmetic: Cubes and Equalities

Martin Bromberger, Christoph Weidenbach

Noname manuscript No.
(will be inserted by the editor)

New Techniques for Linear Arithmetic: Cubes and Equalities

Martin Bromberger · Christoph Weidenbach

the date of receipt and acceptance should be inserted later

Abstract We present several new techniques for linear arithmetic constraint solving. They are all based on the linear cube transformation, a method presented here, which allows us to efficiently determine whether a system of linear arithmetic constraints contains a hypercube of a given edge length.

Our first findings based on this transformation are two sound tests that find integer solutions for linear arithmetic constraints. While many complete methods search along the problem surface for a solution, these tests use cubes to explore the interior of the problems. The tests are especially efficient for constraints with a large number of integer solutions, e.g., those with infinite lattice width. Inside the SMT-LIB benchmarks, we have found almost one thousand problem instances with infinite lattice width. Experimental results confirm that our tests are superior on these instances compared to several state-of-the-art SMT solvers.

We also discovered that the linear cube transformation can be used to investigate the equalities implied by a system of linear arithmetic constraints. For this purpose, we developed a method that computes a basis for all implied equalities, i.e., a finite representation of all equalities implied by the linear arithmetic constraints. The equality basis has several applications. For instance, it allows us to verify whether a system of linear arithmetic constraints implies a given equality. This is valuable in the context of Nelson-Oppen style combinations of theories.

Keywords Linear Arithmetic · SMT · Integer Arithmetic · Constraint Solving · Equalities · Combination of Theories

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 [5, 19]. It is, therefore,

M. Bromberger - C. Weidenbach
Max-Planck-Institut für Informatik, Campus E1 4, 66123 Saarbrücken, Germany
E-mail: {mbromber,weidenb}@mpi-inf.mpg.de

M. Bromberger
Graduate School of Computer Science, Campus E1 3, 66123 Saarbrücken, Germany

no surprise that the theory of linear arithmetic is one of the most popular and best investigated theories for *satisfiability modulo theories* (SMT) solving [14–16].

This paper serves as a collection of our results based on the linear cube transformation. On its own, the linear cube transformation allows us to efficiently determine whether a system of linear arithmetic constraints contains a hypercube of a given edge length. We were able to develop several techniques based on this transformation that allow us to investigate linear arithmetic constraints in various ways. Here, we present our previous results [8, 7] on the linear cube transformation in more detail as well as some new applications (e.g., quantifier elimination).

Finding an integer solution for a polyhedron that is defined by a system of linear inequalities $Ax \leq b$ is a well-known NP-complete problem [25]. This problem has been investigated in different research areas, e.g., in optimization via *(mixed) integer linear programming* (MILP) [19] and in constraint solving via *satisfiability modulo theories* (SMT) [4, 6, 11, 16]. For commercial MILP implementations, it is standard to integrate preprocessing techniques, heuristics, and specialized tests [19]. Although these techniques are not complete, they are much more efficient on their designated target systems of linear inequalities than a complete algorithm alone.

The SMT community is still in the process of developing a variety of specialized tests. A big challenge is to adopt the tests from the MILP community so that they still fit the requirements of SMT solving. SMT theory solvers have to solve a large number of incrementally connected, small systems of linear inequalities. Exploiting this incremental connection is key for making SMT theory solvers efficient [15]. In contrast, MILP solvers typically target one large system. The same holds for their specialized tests, which are not well suited to exploit incremental connections.

Based on the linear cube transformation, we present two tests tailored for SMT solvers: the *largest cube test* and the *unit cube test* [8]. The largest cube test finds a hypercube with maximum edge length contained in the input polyhedron, determines its rational valued center, and rounds it to a potential integer solution. The unit cube test determines if a polyhedron contains a hypercube with edge length one, which is the minimal edge length that guarantees an integer solution. Due to computational complexity, we restrict ourselves to those hypercubes that are parallel to the coordinate axes.

Most SMT linear integer arithmetic theory solvers are based on a branch-and-bound algorithm on top of the simplex algorithm. They search for a solution at the surface of a polyhedron. In contrast, our tests search in the interior of the polyhedron. This gives them an advantage on polyhedra with a large number of integer solutions, e.g., polyhedra with infinite lattice width [20].

SMT theory solvers are designed to efficiently exchange bounds [14]. This efficient exchange is the main reason why SMT theory solvers exploit the incremental connection between the different polyhedra so well. Our unit cube test also requires only an exchange of bounds. After applying the test, we can easily recover the original polyhedron by reverting to the original bounds. In doing so, the unit cube test conserves the incremental connection between the different original polyhedra. We make a similar observation about the largest cube test.

Equalities are a special instance of linear arithmetic constraints. They are useful in simplifying systems of arithmetic constraints [16], and they are essential for the Nelson-Oppen style combinations of theories [9]. However, they are also an obstacle for our fast cube tests. If a system of linear arithmetic constraints 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 have to develop methods that find, isolate, and eliminate implied equalities from systems of linear arithmetic constraints [7].

We can detect the existence of an implied equality by searching for a hypercube in our polyhedron. If the maximal edge length of such a hypercube is zero, there exists an implied equality. This test can be further simplified. By turning all inequalities into strict ones, the interior of the original polyhedron remains while the surface disappears. If the strict system is unsatisfiable, the original system has no interior and implies an equality. Based on an explanation of unsatisfiability for the strict system, the method generates an implied equality as an explanation.

We are also able to extend the above method into an algorithm that computes an equality basis, i.e., a finite representation of all equalities implied by a satisfiable system of linear arithmetic constraints. For this purpose, the algorithm repeatedly applies the above method to find, collect, and eliminate equalities from our system of constraints. When the system contains no more equalities, then the collected equalities represent an equality basis, i.e., any implied equality can be obtained by a linear combination of the equalities in the basis. The equality basis has many applications. If transformed into a substitution, it eliminates all equalities implied by our system of constraints, which results in a system of constraints with an interior and, therefore, improves the applicability of our cube tests. The equality basis also allows us to test whether a system of linear arithmetic constraints implies a given equality. We even extend this test into an efficient method that computes all pairs of equivalent variables inside a system of constraints. These pairs are necessary for the Nelson-Oppen style combination of theories.

While Hillier [17] was aware of the unit cube test, he applied it only to cones (a special class of polyhedra) as a subroutine in a new heuristic. His work never mentioned applications beyond cones, nor did he prove any structural properties connected to hypercubes. Hillier's heuristic tailored for MILP optimization lost popularity as soon as interior point methods [21] became efficient in practice. Nonetheless, our cube tests remain relevant for SMT theory solvers because there are no competitive incremental interior point methods known.

Also, Bobot et al. discuss relations between hypercubes and polyhedra [4] including infinite lattice width and positive linear combinations between inequalities. Our largest cube test can also detect these relations because it is, with some minor changes, the dual of the linear optimization problem of Bobot et al. In contrast to the linear optimization problem of Bobot et al., our tests are closer to the original polyhedron and, therefore, easier to construct. Our cube tests also produce sample points and find solutions for polyhedra with finite lattice width.

Another method that provides a sufficient condition for the existence of an integer solution is the dark shadow of the Omega Test [26]. The dark shadow is based on Fourier-Motzkin elimination and its worst case runtime is double exponential. Although not practically advantageous, formulating the unit cube test through Fourier-Motzkin elimination allows us to put the sufficient conditions of the two methods in relation. Fourier-Motzkin elimination eliminates the variable x from a problem by combining each pair of inequalities $ax \leq p$ and $q \leq bx$ (with $a, b > 0$) into a new inequality $aq - bp \leq 0$. The dark shadow creates a stronger version ($aq - bp \leq a + b - ab$) of the combined inequality to guarantee the existence of an integer solution for x. Formulating the unit cube test through Fourier-Motzkin elimination makes the combined inequality even stronger ($aq - bp \leq -ab$). This

means that the sufficient condition of the dark shadow subsumes the condition of the unit cube test. Still, our unit cube test is definable as a linear program and it is, therefore, computable in polynomial time. So the better condition of the dark shadow comes at the cost of being much harder to compute.

There also already exist several methods that find, isolate, and eliminate implied equalities [3,27,31,32]. Hentenryck and Graf [32] define unique normal forms for systems of linear constraints with non-negative variables. To compute a normal form, they first eliminate all implied equalities from the system. To this end, they determine the lower bound for each inequality by solving one linear optimization problem. Similarly, Refalo [27] describes several incremental methods that use optimization to turn a satisfiable system of linear constraints in "revised solved form" into a system without any implied equalities. Rueß and Shankar also use this optimization scheme to determine a basis of implied equalities [28]. Additionally, they present a necessary but not sufficient condition for an inequality to be part of an equality explanation. During preprocessing, all inequalities not fulfilling this condition are eliminated, thus, reducing the number of optimization problems their method has to compute. However, this preprocessing step might be in itself expensive because it relies on a non-trivial fixed-point scheme. The method presented by Telgen [31] does not require optimization. He presents criteria to detect implied equalities based on the tableau used in the simplex algorithm, but he 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 [3]. He uses Fourier Motzkin variable elimination to compute linear combinations that result in implied equalities.

Our methods that detect implied equalities do not require optimization, which is advantageous because SMT solvers are usually not fine-tuned for optimization. Moreover, we defined our methods for a rather general formulation of linear constraints, which allows us 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 Section 7), while preserving efficiency. 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. This also makes potentially expensive preprocessing techniques obsolete.

The paper is organized as follows: we start with some preliminary definitions in Section 2. Then, we define in Section 3 the linear cube transformation (Proposition 3) that allows us to efficiently compute whether a polyhedron contains a hypercube of a given edge length by solely changing the bounds of the inequalities. Based on this transformation, we develop in Section 4 two tests: the largest cube test and the unit cube test. Both tests find integer solutions for linear arithmetic constraints. For polyhedra with infinite lattice width, both tests always succeed (Lemma 4). Inside the SMT-LIB benchmarks, there are almost one thousand problem instances with infinite lattice width, and we show the advantage of our cube tests on these instances by comparing our implementation of the cube test with several state-of-the-art SMT solvers in Section 5. In Section 6, we show how to investigate equalities with the linear cube transformation. First, we introduce an efficient method for testing whether a system of linear arithmetic constraints implies a given equality (Section 6.1). Then, we extend the method so that it computes an equality basis for our system of constraints (Section 6.2). In Section 7 we

start with an implementation of our methods as an extension of Dutertre and de Moura's version of the simplex algorithm [14], which is integrated in many SMT solvers. The implementation generates justifications and preserves incrementality. The efficient computation of an equality basis can then be used in identifying equivalent variables for the Nelson-Oppen combination of theories. Section 8 concludes the paper including a further application of the linear cube transformation to quantifier elimination.

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 operator $(\ )^T$, which turns all rows $(b_1, \ldots, b_m)$ into columns $(b_1, \ldots, b_m)^T$ and vice versa. We also abbreviate the n-dimensional origin $(0, \ldots, 0)^T$ as 0^n. Likewise, we abbreviate $(1, \ldots, 1)^T$ as 1^n.

In the context of SMT solvers, we have to deal with strict inequalities $a_i^T x < b_i$ and non-strict inequalities $a_i^T x \leq b_i$ as our constraints, where $a_i \in \mathbb{Q}^n$ and $b_i \in \mathbb{Q}$. A system of constraints is, therefore, just a set of inequalities and the rational solutions of this system are exactly those points $x \in \mathbb{Q}^n$ that satisfy all inequalities in this set. The rational solutions of this system also define a polyhedron in the $\mathbb{Q}^n$, where each rational solution is equivalent to a point in the polyhedron. For this reason, we treat *polyhedra* and their definitions through a *system of inequalities* as interchangeable. In the case that our system contains only non-strict inequalities, the polyhedron is even *closed convex*, which entails two very useful properties: firstly, the closed convex polyhedron has a surface if it is neither empty nor encompasses the whole $\mathbb{Q}^n$; secondly, any supremum $h_{\max} = \sup\{h^T x : x \in \mathbb{Q}^n \text{ satisfies } Ax \leq b\}$ over a linear objective $h \in \mathbb{Q}^n$ is either $h_{\max} = -\infty$ because there exists no point satisfying our constraints, $h_{\max} = \infty$ because the supremum is unbounded, or there exists an actual maximum, i.e., there exists an $x \in \mathbb{Q}^n$ that satisfies the constraints and its cost $h^T x$ is equivalent to our supremum $h_{\max}$.

If we also consider strict inequalities, then our polyhedron is no longer necessarily a closed convex set. This means we lose the above properties, which poses a problem when we want to adapt algorithms that originally deal only with non-strict inequalities so they can also deal with strict inequalities. For instance, the classical dual simplex algorithm [29] returns only rational solutions on the surface of the polyhedron defined by $Ax \leq b$. It is, therefore, not trivial to adapt the classical dual simplex algorithm to also handle strict inequalities.

To avoid these problems, we instead model strict inequalities as non-strict inequalities by generalizing the field $\mathbb{Q}$ for our inequality bounds b_i and variables x_i to $\mathbb{Q}_\delta$ [14].

Lemma 1 ([14]) *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* $S_{\delta'} = (S \cup S'_{\delta'}) \setminus S'$ *is satisfiable for all* δ' *with* $0 < \delta' \leq \delta$*, 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. This leads to the ordered vector space $\mathbb{Q}_\delta$ 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 $a \in \mathbb{Q}$ [14]. 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 $b_i \in \mathbb{Q}$. Since we also let the assignments for our variables range over $\mathbb{Q}_\delta$, the solutions of our inequalities describe a closed convex polyhedron in the $\mathbb{Q}_\delta^n$ and methods like the classical simplex algorithm are again complete. It is also easy to extract a purely rational solution $v' \in \mathbb{Q}^n$ from a δ-rational solution $v \in \mathbb{Q}_\delta^n$ of $Ax \leq b$. We just have to choose a small enough value $\delta' \in \mathbb{Q}$ and replace the parameter δ with this value (Lemma 1). Therefore, we call a δ-rational solution just a rational solution. Representing our strict inequalities as non-strict inequalities also allows us to use the second property listed above for closed convex polyhedra: any supremum $h_{\max} = \sup\{h^T x : x \in \mathbb{Q}_\delta^n \text{ satisfies } Ax \leq b\}$ over a linear objective $h \in \mathbb{Q}^n$ is either $h_{\max} = -\infty$, $h_{\max} = \infty$, or there exists an actual maximum and not just a limit. This property is especially useful on techniques based on optimization like the largest cube test (see Section 4.1).

With the δ-rationals, we are now able to formally define a system of constraints as $Ax \leq b$. $Ax \leq b$ is just an abbreviation for the set of inequalities $\{a_1^T x \leq b_1, \ldots, a_m^T x \leq b_m\}$. 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 $b = (b_1, \ldots, b_m)^T \in \mathbb{Q}_\delta^m$. Moreover, we assume that any constant rows $a_i = 0^n$ were eliminated from our system during an implicit preprocessing step. This is a trivial task and eliminates some unnecessarily complicated corner cases. The δ-coefficients q_i in the bounds $b_i = p_i + q_i\delta$ can take on any value in $\mathbb{Q}_\delta$. In case $q_i = 0$, the inequality $a_i^T x \leq b_i$ is equivalent to the non-strict inequality $a_i^T x \leq p_i$. In case $q_i < 0$, the inequality $a_i^T x \leq b_i$ is equivalent to the strict inequality $a_i^T x < p_i$. In case $q_i > 0$, we have no clear interpretation over the actual rationals (compare also Lemma 1). For instance, the two inequalities $x \leq \delta$ and $-x \leq -\delta$ describe a satisfiable system of constraints in $\mathbb{Q}_\delta$, but there is no clear way of interpreting $x \leq \delta$ in $\mathbb{Q}$. Beware also that our linear cube transformation can introduce positive δ-coefficients in the bounds. But since we derive the transformation via a semantically clear construction, the semantic interpretation over the rationals is still discernible if the original system has only non-positive δ-coefficients in its inequality bounds before the transformation.

For the remainder of the paper, we abbreviate with b_i^δ the strict version of a given bound $b_i \in \mathbb{Q}_\delta$. If the bound b_i is non-strict, i.e., $b_i = (p_i, 0)$, then the strict version is $b_i^\delta := (p_i, -1)$. Otherwise, the bound b_i is already strict, i.e., $b_i = (p_i, q_i)$ with $q_i < 0$, and we just standardize the δ-coefficient to -1, i.e, $b_i^\delta := (p_i, -1)$.

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

Besides cubes and polyhedra, we use two norms in this paper. The first norm we use is the *maximum norm*. It is defined by: $\|x\|_\infty = \max\{|x_1|, \ldots, |x_n|\}$, and we use it because it is closely related to our definition of cubes $C_e(z)$, i.e., the condition in the definition of $C_e(z)$ can also be expressed with the maximum norm: $\left(\|x - z\|_\infty \leq \frac{e}{2}\right) \iff \left(\forall j \in 1, \ldots, n.\ |x_j - z_j| \leq \frac{e}{2}\right)$. The second norm we use is the *1-norm*, which is defined by: $\|x\|_1 = (|x_1| + \ldots + |x_n|)$. We use it in Section 3 to define the linear cube transformation.

Using the maximum norm, we define a *closest integer* to x as a point $x' \in \mathbb{Z}^n$ with minimal distance $\left\|x - x'\right\|_\infty$. We also define the operators $\lceil x_j \rfloor$ and $\lceil x \rfloor$ such that they describe a *closest integer* to x_j and x, respectively. Formally, this means that $\lceil x \rfloor = (\lceil x_1 \rfloor, \ldots, \lceil x_n \rfloor)^T$ and

$$\lceil x_j \rfloor = \begin{cases} \lfloor x_j \rfloor & \text{if } x_j - \lfloor x_j \rfloor < 0.5\,, \\ \lceil x_j \rceil & \text{if } x_j - \lfloor x_j \rfloor \geq 0.5\,. \end{cases}$$

This definition of $\lceil x \rfloor$ is also known as *simple rounding*.

Proposition 1 *For $x \in \mathbb{Q}_\delta^n$, $\lceil x \rfloor$ is a closest integer to x, or formally:*

$$\forall x' \in \mathbb{Z}^n.\ \|x - \lceil x \rfloor\|_\infty \leq \left\|x - x'\right\|_\infty .$$

We say that a polyhedron implies an inequality $h^T x \leq g$, where $h \in \mathbb{Q}^n$, $h \neq 0^n$, and $g \in \mathbb{Q}_\delta$, if $h^T x \leq g$ holds for all $x \in P_b^A$. In the same manner, a polyhedron implies an equality $h^T x = g$, where $h \in \mathbb{Q}^n$, $h \neq 0^n$, 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 $d^T x = c$ with $d \neq 0^n$ have edge length zero.

In Section 6, we present a method that detects whether a polyhedron implies an equality at all and returns one such equality. To prove the correctness of this method, we use Farkas' Lemma [5]. But first we have to proof that Farkas' Lemma works with δ-rationals:

Lemma 2 *$Ax \leq b$ is unsatisfiable iff there exists a $y \in \mathbb{Q}^m$ with $y \geq 0^m$ and $y^T A = 0^n$ so that $y^T b < 0$, i.e., there exists a non-negative linear combination of inequalities in $Ax \leq b$ that results in an inequality $y^T Ax \leq y^T b$ that is constant and unsatisfiable.*

Proof. Let us first consider the case where $Ax \leq b$ is unsatisfiable. Dutertre and de Moura's version of the dual simplex algorithm is a complete and correct algorithm for determining the satisfiability of a linear arithmetic problem over the δ-rationals [14]. In case the problem is unsatisfiable, the algorithm returns a conflict explanation, which can be turned, together with the final simplex tableau, into the non-negative linear combination $y \in \mathbb{Q}^m$ we are looking for. Let us now

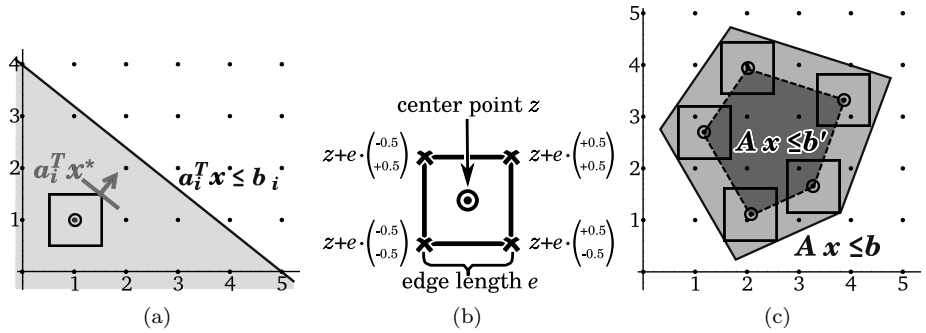

Fig. 1 **a** A square (two-dimensional cube) fitting into an inequality $a_i^T x \leq b_i$ and the cube's maximum $a_i^T x^*$ for the objective $a_i^T x$. **b** The vertices of an arbitrary square parallel to the coordinate axes (two-dimensional cube with edge length e and center z). **c** The transformed polyhedron $Ax \leq b'$ for edge length 1 together with the original polyhedron $Ax \leq b$.

consider the case where $x \in \mathbb{Q}_\delta^n$ is a solution for $Ax \leq b$. If x is a solution to the inequalities in $Ax \leq b$, then it is also a solution to any non-negative linear combination of inequalities in $Ax \leq b$. □

Our method that detects implied equalities transforms our original polyhedron $Ax \leq b$ into a second polyhedron $A'x \leq b'$ that is unsatisfiable if $Ax \leq b$ implies an equality. We also show how to extract an equality implied by $Ax \leq b$ from a *minimal* set C of unsatisfiable inequalities in $A'x \leq b'$. We call an unsatisfiable set C of inequalities *minimal* if every proper subset $C' \subset C$ is satisfiable. If 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 $Ax \leq b$ [14]. 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 Farkas' Lemma:

Lemma 3 (**[7]**) *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 $y \geq 0^n$, $y^T A = 0^n$, and $y^T b < 0$ that $y_i > 0$ for all $i \in \{1, \ldots, m\}$.*

3 Fitting Cubes into Polyhedra

We say that a cube $C_e(z)$ *fits* into a polyhedron defined by $Ax \leq b$ if all points inside the cube $C_e(z)$ are solutions of $Ax \leq b$, or formally: $C_e(z) \subseteq P_b^A$. 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 $C_e(z)$ fits into the original polyhedron. This is a simple test that requires only evaluation. We call this entire transformation the *linear cube transformation*.

We start explaining the linear cube transformation by looking at the case where the polyhedron is defined by a single inequality $a_i^T x \leq b_i$. A cube $C_e(z)$ fits into the inequality $a_i^T x \leq b_i$ if all points inside the cube $C_e(z)$ are solutions of $a_i^T x \leq b_i$, or formally: $\forall x \in C_e(z).\ a_i^T x \leq b_i$.

We can think of $a_i^T x$ as an objective function that we want to maximize and see b_i as a guard for the maximum objective of any solution in the cube. Thus, we can express the universal quantifier in the above equation as an optimization problem (see Figure 1a): $\max\{a_i^T x : x \in C_e(z)\} \leq b_i$. This also means that all points in $x \in C_e(z)$ satisfy the inequality $a_i^T x \leq b_i$ if a point $x^* \in C_e(z)$ with maximum value $a_i^T x^* = \max\{a_i^T x : x \in C_e(z)\}$ for the objective function $a_i^T x$ satisfies the inequality $a_i^T x^* \leq b_i$. Since every cube is a bounded polyhedron, one of the points with maximum objective value is a vertex $x^v \in C_e(z)$. A vertex x^v of the cube $C_e(z)$ is one of the points with maximum distance to the center z (see Figure 1b), or formally: $x^v = \left(z_1 \pm \frac{e}{2}, \ldots, z_n \pm \frac{e}{2}\right)^T$. If we insert the above equation into the objective function $a_i^T x$, we get: $a_i^T \left(z_1 \pm \frac{e}{2}, \ldots, z_n \pm \frac{e}{2}\right)^T = a_i^T z + \frac{e}{2} \sum_{j=1}^{n} \pm a_{ij}$, which in turn is maximal if we choose x^v such that $\pm a_{ij}$ is always positive:

$$a_i^T x^v = a_i^T z + \tfrac{e}{2} \textstyle\sum_{j=1}^{n} |a_{ij}| = a_i^T z + \tfrac{e}{2} \left\|a_i\right\|_1 .$$

Hence, we transform the inequality $a_i^T x \leq b_i$ into $a_i^T x \leq b_i - \frac{e}{2} \left\|a_i\right\|_1$, and $C_e(z)$ fits into $a_i^T x \leq b_i$ if $a_i^T z \leq b_i - \frac{e}{2} \left\|a_i\right\|_1$.

Proposition 2 *Let $C_e(z)$ be a cube and $a_i^T x \leq b_i$ be an inequality. All $x \in C_e(z)$ fulfill $a_i^T x \leq b_i$ if and only if $a_i^T z \leq b_i - \frac{e}{2} \left\|a_i\right\|_1$.*

Next, we look at the case where multiple inequalities $a_i^T x \leq b_i$ (for $i = 1, \ldots, m$) define the polyhedron $Ax \leq b$. Since P_b^A is the intersection of all $P_{b_i}^{a_i}$, the cube fits into $Ax \leq b$ if and only if it fits into all inequalities $a_i^T x \leq b_i$:

$$\forall i \in \{1, \ldots, m\}.\ \forall x \in C_e(z).\ a_i^T x \leq b_i .$$

We can express this by m optimization problems:

$$\forall i \in \{1, \ldots, m\}.\ \max\{a_i^T x : x \in C_e(z)\} \leq b_i$$

and, after applying Proposition 2, by the following m inequalities:

$$\forall i \in \{1, \ldots, m\}.\ a_i^T z \leq b_i - \tfrac{e}{2} \left\|a_i\right\|_1 .$$

Hence, the linear cube transformation transforms the polyhedron $Ax \leq b$ into the polyhedron $Ax \leq b'$, where $b'_i = b_i - \frac{e}{2} \left\|a_i\right\|_1$, and $C_e(z)$ fits into $Ax \leq b$ if $Az \leq b'$.

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

Until now, we have discussed how to use the linear cube transformation to determine if one cube $C_e(z)$ with fixed center point z fits into a polyhedron. A generalization of this problem determines whether a polyhedron $Ax \leq b$ contains a cube of edge length e at all. Actually, a closer look at the transformed polyhedron $Ax \leq b'$ reveals that the linear cube transformation ($b'_i = b_i - \frac{e}{2} \left\|a_i\right\|_1$) is dependent only on the edge length e of the cube. Therefore, the solutions $P_{b'}^A$ of the transformed polyhedron $Ax \leq b'$ are exactly all center points of cubes with edge length e that fit into the original polyhedron $Ax \leq b$ (see Figure 1c). By determining the satisfiability of the transformed polyhedron $Ax \leq b'$, we can now also determine whether a polyhedron $Ax \leq b$ contains a cube of edge length e at all. If we choose a suitable algorithm, e.g., the simplex algorithm, then we even get the center point z of a cube $C_e(z)$ that fits into $Ax \leq b$. This observation is the foundation for the cube tests that we present in Section 4.

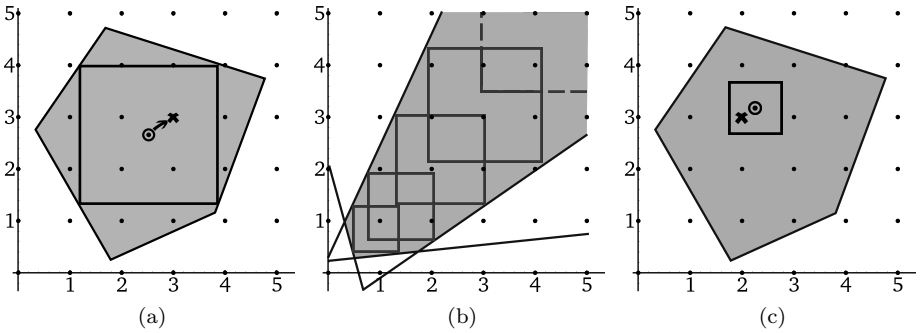

Fig. 2 **a** The largest cube inside a polyhedron, its center point, and a closest integer point to the center. **b** An infinite lattice width polyhedron, containing cubes for every edge length $e > 0$. **c** A unit cube inside a polyhedron, its center point, and a closest integer point to the center.

4 Fast Cube Tests

A polyhedron $Ax \leq b$ has an integer solution if and only if $P_b^A \cap \mathbb{Z}^n \neq \emptyset$, i.e., if the set of rational solutions contains an integer point. In this section, we show how to use the linear cube transformation to find such an integer solution. In contrast to arbitrary polyhedra, determining whether a cube $C_e(z)$ contains an integer point is easy. Because of the cubes symmetry, it is enough to test whether it contains a closest integer point $\lceil z \rfloor$ to the center z.

Proposition 4 *A cube $C_e(z)$ contains an integer point if and only if it contains a closest integer point $\lceil z \rfloor$ to the center z.*

Note that every point $z \in \mathbb{Q}_\delta^n$ is also a cube $C_0(z)$ of edge length 0. In order to be efficient, our tests look only at cubes with special properties. In the case of the largest cube test, we check for an integer solution in one of the largest cubes fitting into the polyhedron $Ax \leq b$. In the case of the unit cube test, we look for a cube of edge length one, which always guarantees an integer solution. Due to these restrictions, both tests are not complete but very fast to compute.

4.1 Largest Cube Test

A well-known test, implemented in most ILP solvers, is *simple rounding*. For simple rounding, the ILP solver computes a rational solution x for a set of inequalities, *rounds* it to a closest integer $\lceil x \rfloor$, and determines whether this point is an integer solution. Not all types of rational solutions are good candidates for this test to be successful. Especially *surface points*, such as *vertices*, the usual output of the simplex algorithm, are not good candidates for rounding. For many polyhedra, *center and interior points* z are a better choice because all integer points adjacent to z are solutions, including a closest integer point $\lceil z \rfloor$.

We now use the linear cube transformation (Section 3) to calculate a rational center point with the simplex algorithm. The center point we calculate is the center point of a largest cube that fits into the polyhedron $Ax \leq b$ (see Figure 2a). We

determine the center z of this largest cube and the associated edge length e with the following linear program (LP):

$$\begin{array}{ll} \text{maximize} & x_e \\ \text{subject to} & Ax + a' \frac{x_e}{2} \leq b, \text{ where } a'_i = \|a_i\|_1 \\ & x_e \geq 0\,. \end{array}$$

This linear program employs the linear cube transformation from Section 3. The only generalization is a variable x_e for the edge length instead of a constant value e. Additionally, this linear program maximizes the edge length as an optimization goal. If the resulting maximum edge length is unbounded, the original polyhedron contains cubes of arbitrary edge length (see Figure 2b) and, thus, infinitely many integer solutions. Since the linear program contains all solutions of the original polyhedron (see $x_e = 0$), the original polyhedron is empty if and only if the above linear program is infeasible. If the maximum edge length is a finite value e, we use the resulting assignment z for the variables x as a center point and $C_e(z)$ is a largest cube that fits into the polyhedron. From the center point, we round to a closest integer point $\lceil z \rfloor$ and determine if it fits into the original polyhedron. If this is the case, we are done because we have found an integer solution for $Ax \leq b$. Otherwise, the largest cube test does not know whether or not $Ax \leq b$ has an integer solution. An example for the latter case, are the following inequalities: $3x_1 - x_2 \leq 0$, $-2x_1 - x_2 \leq -2$, and $-2x_1 + x_2 \leq 1$. These inequalities have exactly one integer solution $(1, 3)^T$, but the largest cube contained by the inequalities has edge length $e = \frac{3}{17}$ and center point $(\frac{3}{17}, \frac{3}{2})^T$, which rounds to $(0, 2)^T$.

The largest cube test also upholds the incremental advantages of Dutertre and de Moura's version of the dual simplex algorithm [14]. The only difference is the extra column $a' \frac{x_e}{2}$, which the theory solver can internally create while it is notified of all potential arithmetic literals. Adding this column from the start does not influence the correctness of the solution because $x_e \geq 0$ guarantees that the largest cube test is satisfiable exactly when the original inequalities $Ax \leq b$ are satisfiable. Even for explanations of unsatisfiability, it suffices to remove the bound $x_e \geq 0$ to obtain an explanation for the original inequalities $Ax \leq b$. The only disadvantage is the additional variable x_e, which only shrinks the search space when it is increased. Therefore, increasing x_e can never resolve any conflicts during the satisfiability search. The simplex solver recognizes this with at least one additional pivot that sets x_e to 0. Hence, adding the extra column $a' \frac{x_e}{2}$ from the beginning has only constant influence on the theory solver's run-time, and is therefore negligible.

4.2 Unit Cube Test

Most SMT solvers implement a simplex algorithm that is specialized towards feasibility and not towards optimization [1, 14, 16, 24]. Therefore, a test based on optimization, such as the largest cube test, does not fit well with existing implementations. As an alternative, we have developed a second test based on cubes that does not need optimization.

We avoid optimization by fixing the edge length e to the value 1 for all the cubes $C_e(z)$ we consider (see Figure 2c). We do so because cubes $C_1(z)$ of edge length 1 are the smallest cubes to always guarantee an integer solution, independent of the center point z. A cube with edge length 1 is also called a

unit cube. To prove this guarantee, we first fix $e = 1$ in the definition of cubes, $C_1(z) = \left\{x \in \mathbb{Q}_\delta^n : \forall j \in 1, \ldots, n.\ |x_j - z_j| \leq \frac{1}{2}\right\}$, and look at the following property for the rounding operator $\lceil . \rfloor$: $\forall z_j \in \mathbb{Q}_\delta . |\lceil z_j \rfloor - z_j| \leq \frac{1}{2}$. We see that any unit cube contains a closest integer $\lceil z \rfloor$ to its center point z. Furthermore, 1 is the smallest edge length that guarantees an integer solution for a cube with center point $z = (\ldots, \frac{1}{2}, \ldots)^T$. Thus, 1 is the smallest value that we can fix as an edge length to guarantee an integer solution for all cubes $C_1(z)$.

Our second test tries to find a unit cube that fits into the polyhedron $Ax \leq b$ and, thereby, a guarantee for an integer solution for $Ax \leq b$. Again, we employ the linear cube transformation from Section 3 and obtain the linear program:

$$Az \leq b', \text{ where } b'_i = b_i - \tfrac{1}{2} \left\|a_i\right\|_1 .$$

In addition to being a linear program without an optimization objective, we only have to change the row bounds b'_i of the original inequalities. In Dutertre and de Moura's version of the dual simplex algorithm [14], which is implemented in many SMT solvers [1,14,16,24], such a change of bounds is already part of the framework so that integrating the unit cube test into theory solvers is possible with only minor adjustments to the existing implementation. Since our unit cube test requires only an exchange of bounds, we can easily return to the original polyhedron by reverting the bounds. In doing so, the unit cube test upholds the incremental connection between the different original polyhedra.

4.3 Mixed Linear Integer and Rational Arithmetic

We can also extend our cube tests to the theory of mixed linear integer and rational arithmetic. In this theory, we partition our variables $x = (x_1, \ldots, x_n)^T$ into two vectors: the integer variables $x^\mathbb{Z} = (x_1^\mathbb{Z}, \ldots, x_k^\mathbb{Z})^T$ and the rational variables $x^\mathbb{Q} = (x_1^\mathbb{Q}, \ldots, x_l^\mathbb{Q})^T$. Based on this partitioning, we also split the coefficient matrix A into two matrices $A = (S, R)$, where $S = (s_1, \ldots, s_m)^T \in \mathbb{Q}^{m \times k}$ defines the coefficients for the integer variables and $R = (r_1, \ldots, r_m)^T \in \mathbb{Q}^{m \times l}$ defines the coefficients for the rational variables. The system has a solution if there exists an integer assignment for the variables $x^\mathbb{Z}$ and a rational assignment for the variables $x^\mathbb{Q}$ that satisfies our system of inequalities $s_i^T x^\mathbb{Z} + r_i^T x^\mathbb{Q} \leq b_i$ (for $i = 1, \ldots, m$).

Because only integer variables need to be assigned to integer values, tests like simple rounding should be restricted to integer variables. For instance, if z is a rational solution for the overall polyhedron, then simple rounding applies $\lceil . \rfloor$ only to the components $z^\mathbb{Z}$ of z that correspond to integer variables. The same holds for our fast cube tests. Instead of looking for hypercubes of the same dimension n as the number of total variables, we are looking for hypercubes of dimension k that expand in the directions that correspond to integer variables, but are flat in the directions that correspond to rational variables. Such a hypercube of dimension k with center point z is defined as the set:

$$F_e(z) = \left\{x \in \mathbb{Q}_\delta^k : \forall j \in 1, \ldots, k.\ |x_j^\mathbb{Z} - z_j^\mathbb{Z}| \leq \tfrac{e}{2}\right\} \times \left\{z^\mathbb{Q}\right\} .$$

We can also modify the linear cube transformation so that we can compute whether a polyhedron $Sx^\mathbb{Z} + Rx^\mathbb{Q} \leq b$ contains a hypercube $F_e(z)$ that is less than full dimensional:

Proposition 5 *Let $F_e(z)$ be a flat cube of dimension k and $Sx^\mathbb{Z} + Rx^\mathbb{Q} \leq b$ be a polyhedron. $F_e(z) \subseteq P_b^A$ if and only if $Sz^\mathbb{Z} + Rz^\mathbb{Q} \leq b'$, where $b'_i = b_i - \frac{e}{2} \left\|s_i\right\|_1$.*

Since the hypercube $F_e(z)$ only expands in the directions that correspond to integer variables, the inequality bounds b' of the modified linear cube transformation are only influenced by the coefficients of the integer variables. Using Proposition 5, we can now modify our fast cube tests so that they work for mixed linear integer and rational arithmetic. For the largest cube test, we compute the center point of a largest cube $F_e(z)$ that is flat in the directions that correspond to rational variables and fits into the polyhedron $Sx^{\mathbb{Z}} + Rx^{\mathbb{Q}} \leq b$. We determine the center z of this largest cube and the associated edge length e with the following LP:

$$\begin{array}{ll} \text{maximize} & x_e \\ \text{subject to} & Sx^{\mathbb{Z}} + Rx^{\mathbb{Q}} + s' \frac{x_e}{2} \leq b, \text{ where } s'_i = \|s_i\|_1 \\ & x_e \geq 0\,. \end{array}$$

From the resulting center point z we receive a candidate mixed integer rational solution by applying the rounding operator $\lceil . \rfloor$ to the components $z^{\mathbb{Z}}$ of z that correspond to integer variables. For the unit cube test, we search for a cube $F_1(z)$ that is flat in the directions that correspond to rational variables, has edge length 1, and fits into the polyhedron $Sx^{\mathbb{Z}} + Rx^{\mathbb{Q}} \leq b$. A linear program that accomplishes this task is: $Sx^{\mathbb{Z}} + Rx^{\mathbb{Q}} \leq b'$, where $b'_i = b_i - \frac{1}{2}\|s_i\|_1$.

Again, 1 is the smallest value that we can fix as an edge length to guarantee a mixed rational integer solution for all cubes $F_1(z)$.

5 Experiments

While our tests are useful for many types of polyhedra, the motivation for our tests stems from a special type of polyhedron, a so-called *infinite lattice width* polyhedron [20]. A polyhedron $Ax \leq b$ has *infinite lattice width* if for every objective $c \in \mathbb{Q}^n \setminus \{0^n\}$, either its maximum or minimum objective value is unbounded:

$$\forall c \in \mathbb{Q}^n \setminus \{0^n\}.\ \sup\left\{c^T x : x \in P_b^A\right\} = \infty \text{ or } \inf\left\{c^T x : x \in P_b^A\right\} = -\infty\,.$$

Polyhedra with infinite lattice width seem trivial at first glance because their interior expands arbitrarily far in all directions (see Figure 2b). Therefore, a polyhedron with infinite lattice width contains an infinite number of integer solutions [20]. Nonetheless, many SMT solvers are inefficient on those polyhedra because they use a branch-and-bound approach with an underlying simplex solver [14]. Although such an approach terminates inside finite a priori bounds [25], it does not explore the infinite interior, but rather directs the search along the solutions suggested by the simplex solver: the vertices of the polyhedron. Thus, the SMT solvers concentrate their search on a bounded part of the polyhedron. This bounded part contains only a finite number of integer solutions, whereas the complete interior contains infinitely many integer solutions. The advantage of our cube tests is that they actually exploit the infinite interior because polyhedra with infinite lattice width contain cubes for every edge length (see Figure 2b). Our tests are, therefore, always successful on polyhedra with infinite lattice width and usually need only a small number of pivoting steps before finding a solution.

Lemma 4 ([8]) *Let $Ax \leq b$ be a polyhedron. Let $a' \in \mathbb{Q}^m$ be a vector such that its components are $a'_i = \|a_i\|_1$. Then $Ax \leq b$ contains a cube $C_e(z)$ for every non-negative $e \in \mathbb{Q}_\delta$ if and only if $Ax \leq b$ has infinite lattice width.*

We have found instances of polyhedra with the infinite lattice width property in some classes of the SMT-LIB benchmarks. These instances are 229 of the 233 *dillig*

Benchmark Name	CAV-2009		DILLIG		PRIME-CONE		SLACKS		ROTATE	
#Instances	503		229		19		229		229	
Solvers:	**solved**	**time**	**solved**	**time**	**solved**	**time**	**solved**	**time**	**solved**	**time**
SPASS-IQ-0.1+uc	**503**	22	**229**	9	**19**	0.4	**229**	**26**	**229**	**9**
SPASS-IQ-0.1	**503**	713	**229**	218	**19**	0.4	197	95	**229**	214
ctrl-ergo	**503**	**12**	**229**	**5**	**19**	0.4	**229**	46	24	6760
cvc4-1.4	467	12903	206	4146	18	3	152	4061	208	6964
mathsat5-3.13+uc	**503**	42.37	**229**	18	**19**	0.4	**229**	39	**229**	21
mathsat5-3.13	500	4601	225	2315	**19**	3.5	181	4573	**229**	1507
yices-2.5.1	469	11403	213	2553	**19**	**0.1**	147	5725	180	10073
z3-4.4.1	466	682	213	475	**19**	**0.1**	158	371	213	473

Fig. 3 Experimental Results

benchmarks designed by Dillig et al. [11], 503 of the 591 *CAV-2009* benchmarks also by Dillig et al. [11], 229 of the 233 *slacks* benchmarks which are the dillig benchmarks extended with slack variables [18], and 19 of the 37 *prime-cone* benchmarks, that is, "a group of crafted benchmarks encoding a tight n-dimensional cone around the point whose coordinates are the first n prime numbers" [18]. The remaining problems (4 from dillig, 88 from CAV-2009, 4 from slacks, and 18 from prime-cone) do not have infinite lattice width because they are either tightly bounded or unsatisfiable. For our experiments, we look only at the instances of those benchmark classes that actually fulfill the infinite lattice width property.

Using these benchmark instances, we have confirmed our theoretical assumptions (Lemma 4) in practice. We integrated the unit cube test into our own branch-and-bound solver *SPASS-IQ* (http://www.spass-prover.org/spass-iq) and ran it on the infinite lattice width instances; once with the unit cube test turned on (*SPASS-IQ-0.1+uc*) and once with the test turned off (*SPASS-IQ-0.1*). For every problem, SPASS-IQ-0.1+uc applies the unit cube test exactly once. This application happens before we start the branch-and-bound approach. We also compared our solver with state-of-the-art SMT solvers for linear integer arithmetic: *cvc4-1.4* [1], *mathsat5-3.13* [10], *yices2.5.1* [13], and *z3-4.4.1* [24]. All these solvers employ a branch-and-bound approach with an underlying dual simplex solver [14]. The only exception is mathsat5, which, subsequent to our first publication on the unit cube test [8], now also performs the unit cube test in advance. That is why we also test mathsat5 once with the unit cube test turned on (mathsat5-3.13+uc) and once with the test turned off (mathsat5-3.13).

The solvers had to solve each problem in under 10 minutes. For the experiments, we used a Debian Linux server with 32 Intel Xeon E5-4640 (2.4 GHz) processors and 512 GB RAM. Figure 3 lists the results of the different solvers (column one) on the different benchmark classes (row one). Row two lists the number of benchmark instances we considered for our experiments. For each combination of benchmark class and solver, we have listed the number of instances the solver could solve in the given time as well as the total time (in seconds) of the instances solved (columns labelled with "solved" and "time", respectively).

Our solver that employs the unit cube test solves all instances with the application of the unit cube test and is 25 times faster than our solver without the test. The SMT theory solvers in their standard setting were not able to solve all instances within the allotted time. Moreover, our unit cube test was over 100 times faster than any state-of-the-art SMT solver without the unit cube test. The results for mathsat5 further support the superiority of the test.

We also compared our test with the *ctrl-ergo* solver, which includes a subroutine that is essentially the dual to our largest cube test [4]. As expected, both approaches are comparable for infinite lattice width polyhedra. In order to also compare the two approaches on benchmarks without infinite lattice width, we created the *rotate* benchmarks by adding the same four inequalities to all infinite width instances of the dillig benchmarks. These four inequalities essentially describe a square bounding the variables x_0 and x_1 in an interval $[-u, u]$. For a large enough choice of u (e.g., $u = 2^{10}$), the square is so large that the benchmarks are still satisfiable and not absolutely trivial for branch-and-bound solvers. To add a challenge, we rotated the square by a small factor $1/r$, which resulted in the following four inequalities:

$$-b \cdot r \cdot r + r \leq b \cdot r \cdot x_0 - x_1 \leq b \cdot r \cdot r - r\,, \text{ and}$$
$$-b \cdot r \cdot r + r \leq x_0 + b \cdot r \cdot x_1 \leq b \cdot r \cdot r - r\,.$$

These changes have nearly no influence on SPASS-IQ, and two SMT solvers even benefit from the proposed changes. For ctrl-ergo the *rotate* benchmarks are very hard because its subroutine detects only infinite lattice width. Without infinite lattice width, ctrl-ergo starts its search from the boundaries of the polyhedron instead of looking at the polyhedron's interior. We can even control the number of iterations (r^2) ctrl-ergo spends on the parts of the boundary without any integer solutions if we choose r accordingly (e.g., $r = 2^{10}$). In contrast, we use our cube tests to also extract interior points for rounding. This difference makes our tests much more stable under small changes to the polyhedron.

There exist alternative methods for solving linear integer constraints that do not rely on a branch-and-bound approach [6,18]. These have not yet matured enough to be competitive with our tests or state-of-the-art SMT theory solvers.

Most problems in the linear integer arithmetic SMT-LIB benchmarks with finite lattice width can be solved without using any actual integer arithmetic techniques. A standard simplex solver for the rationals typically finds a rational solution for such a problem that is also an integer solution. Applying the unit cube test on these trivial problem classes is a waste of time. In the worst case, it doubles the eventual solution time. For these examples it is beneficial to first compute a general rational solution and to check it for integer satisfiability before applying the unit cube test. This has the additional benefit that rational unsatisfiable problems are filtered out before applying the unit cube test. The unit cube test is also guaranteed to fail on problems containing boolean variables, i.e., variables that are either 0 or 1, unless they are absolutely trivial and describe a unit cube themselves. Whenever the problem contains a boolean variable, it is beneficial to skip the unit cube test. This is also the reason why we provide no experimental results for the theory of mixed linear integer and rational arithmetic, i.e., the few mixed benchmarks available in the SMT-LIB all contain boolean variables.

6 From Cubes to Equalities

If a polyhedron implies an equality, then it has only surface points and neither an interior nor a center. There is no way such a polyhedron contains a unit cube and a largest cube has edge length zero and is just a point in the original polyhedron. Equalities are, therefore, a challenge for the applicability of our cube tests.

There even exist systems of inequalities that imply infinitely many equalities. For instance, the system consisting of the inequalities $-2x_1+x_2 \leq -2$, $x_1+3x_2 \leq 8$, and $x_1 - 2x_2 \leq -2$ has only one rational solution: the point $(x_1, x_2) = (2, 2)$. Therefore, it implies the equalities $-2x_1 + x_2 = -2$ and $x_1 + 3x_2 = 8$, and all linear combinations of those two equalities, i.e., $\lambda_1 \cdot (-2x_1 + x_2) + \lambda_2 \cdot (x_1 + 3x_2) = \lambda_1 \cdot (-2) + \lambda_2 \cdot 8$ for all $\lambda_1, \lambda_2 \in \mathbb{Q}$. The above example also points us to another fact about equalities: there exists a finite representation of all equalities implied by a system of inequalities—even if the system implies infinitely many equalities.

One such finite representation is the *equality basis* for a satisfiable system of inequalities $Ax \leq b$. An equality basis 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 $D'x = c'$. We prefer to represent each equality basis $D'x = c'$ 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 $c \in \mathbb{Q}^{n_y}$. The existence of such an equivalent system of equalities is guaranteed by Gaussian elimination. Moreover, each variable y_i appears exactly once in the system $y - Dz = c$, that is to say, y_i appears only in the row $y_i - d_i^T z_i = c_i$. We choose to represent our equality bases in this manner because this form also correlates to a distinct substitution $\sigma_{y,z}^{D,c}$ that replaces variable y_i with $c_i + d_i^T z$:

$$\sigma_{y,z}^{D,c} := \{y_i \mapsto c_i + d_i^T z : i \in \{1, \ldots, n_y\}\}.$$

The substitution $\sigma_{y,z}^{D,c}$ is important because it allows us to eliminate all equalities from $Ax \leq b$. We simply apply the substitution $\sigma_{y,z}^{D,c}$ to $Ax \leq b$ and receive a new system $A'z \leq b'$ that neither contains the variables y nor implies any equalities.[1] And the substitution $\sigma_{y,z}^{D,c}$ for the equality basis $y - Dz = c$ has even further applications. For instance, 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 $\sigma_{y,z}^{D,c}$ to $h^T x = g$ and see if it simplifies to $0 = 0$. We even use $\sigma_{y,z}^{D,c}$ for the *Nelson-Oppen* style combination of theories (see Section 7).

6.1 Finding Equalities

The first step in computing an equality basis for a polyhedron $Ax \leq b$ is to detect whether the system contains any equalities. We have already stated a criterion that detects this:

Lemma 5 ([8]) *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 $h \neq 0^n$, or (2) $Ax \leq b$ contains a cube with edge length $e > 0$.*

A cube with positive edge length is enough to prove that there exists no implied equality. The actual edge length e of this cube is not relevant. Therefore, we can assume that the edge length e is arbitrarily small. We can even assume that our edge length is so small that we can ignore the different multiples $\|a_i\|_1$ and any infinitesimals introduced by strict inequalities. We just have to turn all of our inequalities into strict inequalities.

[1] If we combine the equality basis with a *diophantine equation handler* [16], then we even receive a substitution σ' that eliminates the equalities in such a way that we can reconstruct an integer solution from them. The result is a new system of inequalities that implies no equalities and has an integer solution if and only if $Ax \leq b$ has one.

Lemma 6 *Let $Ax \leq b$ be a polyhedron, where $a_i \neq 0^n$, $b_i = (p_i, q_i)$, $q_i \leq 0$, 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.*

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

(2) $\Rightarrow$ (1): If $Ax \leq b^\delta$ is satisfiable, then we know by Lemma 1 that there must exist a $\delta > 0$ such that $Ax \leq p - \delta 1^m$ holds. Let $a_{\max} = \max\{\|a_i\|_1 : i = 1, \ldots, m\}$, $\delta' = \frac{\delta}{2}$, and $e = \frac{\delta}{a_{\max}}$. Then $p_i - \delta = p_i - \delta' - \frac{e}{2} a_{\max} \leq b_i - \frac{e}{2} \|a_i\|_1$. 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 5, an equality. 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, 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 7 *Let $Ax \leq b$ be a satisfiable polyhedron, where $a_i \neq 0^n$, $b_i = (p_i, q_i)$, $q_i \leq 0$, and $b_i^\delta = (p_i, -1)$ 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 2 and $Ax \leq b^\delta$ being unsatisfiable, we know that there exists a $y \in \mathbb{Q}^m$ with $y \geq 0$, $y^T A = 0^n$, and $y^T b^\delta < 0$. By Lemma 2 and $Ax \leq b$ being satisfiable, we know that $y^T b \geq 0$ is also true. By Lemma 3, we know that $y_k > 0$ for every $k \in \{1, \ldots, m\}$.

Now, we 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 $q_i \leq 0$, we get that $y^T q = 0$ and $q_i = 0$. Since $q_i = 0$, $b = p$.

Next, we multiply $y^T A = 0^n$ 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 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 7 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

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

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 $z - Dy = c$ for $Ax \leq b$

1 $l := 1$, $n_z := n$, $(z_1, \ldots, z_{n_z}) := (x_1, \ldots, x_n)$, $y := ()^T$, and $(y - Dz = c) := \emptyset$
2 Remove all rows $a_i^T z \leq b_i$ from $Az \leq b$ with $a_i = 0^n$ and $b_i = 0$
3 **while** $Az \leq b^\delta$ is unsatisfiable (i.e., $Az \leq b$ contains an equality) **do**
4 Let C be an explanation for $Az \leq b^\delta$ being unsatisfiable
5 Select $(a_i^T z \leq b_i^\delta) \in C$; `// by Lemma 7,` $a_i^T z = b_i$ `is implied by` $Az \leq b$
6 Select a variable z_k such that $a_{ik} \neq 0$.
7 $\sigma' := \{z_k \mapsto \frac{b_i}{a_{ik}} - \sum_{j=1, j \neq k}^{n} \frac{a_{ij}}{a_{ik}} z_j\}$
8 $z' := (z_1, \ldots, z_{k-1}, z_{k+1}, \ldots, z_n)^T$, $y' := (y_1, \ldots, y_l, z_k)^T$, $l := l + 1$
9 $(A'z' \leq b') := (Az \leq b)\sigma'$
10 $(y' - D'z' = c') := (y - Dz = c)\sigma' \cup \{z_k + \sum_{j=1, j \neq k}^{n} \frac{a_{ij}}{a_{ik}} z_j = \frac{b_i}{a_{ik}}\}$
11 $z := z'$, $y := y'$, $(Az \leq b) := (A'z' \leq b')$, $(y - Dz = c) := (y' - D'z' = c')$
12 Remove all rows $a_i^T z \leq b_i$ from $Az \leq b$ with $a_i = 0^n$ and $b_i = 0$
13 **end**
14 **return** $y - Dz = c$

Fig. 4 `EqBasis` computes an equality basis

$Ax \leq b$. For example, if we have an assignment $v \in \mathbb{Q}_\delta^n$ 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 $Ax \leq b$.

6.2 Computing an Equality Basis

We now present the algorithm `EqBasis`$(A'x \leq b')$ (Figure 4) that computes an equality basis for a polyhedron $A'x \leq b'$. In a nutshell, `EqBasis` iteratively detects and removes equalities from our system of inequalities and collects them in a system of equalities until it has a complete equality basis. To this end, `EqBasis` computes in each iteration one system of inequalities $Az \leq b$ and one system of equalities $y - Dz = c$ such that $A'x \leq b'$ is equivalent to $(y - Dz = c) \cup (Az \leq b)$. While the variables z are completely defined by the inequalities $Az \leq b$, the equalities $y - Dz = c$ 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 $A'x \leq b'$.

In every iteration l of the while loop, `EqBasis` eliminates one equality $a_i^T z = b_i$ from $Az \leq b$ and adds it to $y - Dz = c$. `EqBasis` finds this equality based on the techniques we presented in the Lemmas 6 & 7 (line 3). If the current system of inequalities $Az \leq b$ implies no equality, then `EqBasis` is done and returns the current system of equalities $y - Dz = c$. Otherwise, `EqBasis` turns the found equality $a_i^T z = b_i$ into a substitution $\sigma' := \{z_k \mapsto \frac{b_i}{a_{ik}} - \sum_{j=1, j \neq k}^{n} \frac{a_{ij}}{a_{ik}} z_j\}$ (line 7) and applies it to $Az \leq b$ (line 9). This has the following effects: (1) the new system of inequalities $A'z' \leq b'$ implies no longer the equality $a_i^T z = b_i$; and (2) it no longer contains the variable z_k. Next, we apply σ' to our system of equalities (line 10) and concatenate the equality $z_k + \sum_{j=1, j \neq k}^{n} \frac{a_{ij}}{a_{ik}} z_j = \frac{b_i}{a_{ik}}$ to the end of $(y - Dz = c)\sigma'$. This has the following effects: (1) the new system of equalities $y' - D'z' = c'$ implies $a_i^T z = b_i$; and (2) the variable z_k appears exactly once in

$y' - D'z' = c'$. This means that we can now re-partition our variables so that $z := (z_1, \ldots, z_{k-1}, z_{k+1}, \ldots, z_n)^T$ and $y_l := z_k$ to get two new systems $Az \leq b$ and $y - Dz = c$ that are equivalent to our original polyhedron (line 11). Finally, we remove all rows $0 \leq 0$ from $Az \leq b$ because those rows are trivially satisfied but would obstruct the detection of equalities with Lemma 6.

To prove the correctness of the algorithm, we first need to prove that moving the equality from our system of inequalities to our system of equalities preserves equivalence, i.e, the systems $(Az \leq b) \cup (y - Dz = c)$ and $(A'z' \leq b') \cup (y' - D'z' = c')$ are equivalent in line 10.

Lemma 8 *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 $Az \leq b$ with $h_k \neq 0$. Let $\sigma' := \{z_k \mapsto \frac{g}{h_k} - \sum_{j=1, j\neq k}^{n} \frac{h_j}{h_k} z_j\}$ be a substitution based on this equality. Let $y' := (y_1, \ldots, y_l, z_k)^T$ and $z' := (z_1, \ldots, z_{k-1}, z_{k+1}, \ldots, z_n)^T$. Let $(A'z' \leq b') := (Az \leq b)\sigma'$. Let $(y' - D'z' = c') := (y - Dz = c)\sigma' \cup \{z_k + \sum_{j=1, j\neq k}^{n} \frac{h_j}{h_k} z_j = \frac{g}{h_k}\}$. Let $u \in \mathbb{Q}_\delta^{n_y}$, $v \in \mathbb{Q}_\delta^{n_z}$, $u' = (u_1, \ldots, u_{n_y}, v_k)^T$, and $v' = (v_1, \ldots, v_{k-1}, v_{k+1}, \ldots, v_{n_z})^T$. 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. First, we create a new substitution $\sigma_v := \{z_k \mapsto \frac{g}{h_k} - \sum_{j=1, j\neq k}^{n} \frac{h_j}{h_k} v_j\}$ that is equivalent to σ' except that it directly assigns the variables z_i to their values v_i. Let us now assume that either $(Av \leq b) \cup (u - Dv = c)$ or $(A'v' \leq b') \cup (u' - D'v' = c')$ is true. This means that $h^T v = g$ is also true, either by definition of $(Av \leq b)$ or $(u' - D'v' = c')$. But $h^T v = g$ is true also implies that $v_k = \frac{g}{h_k} - \sum_{j=1, j\neq k}^{n} \frac{h_j}{h_k} v_j$ is true. Therefore, σ_v simplifies to the assignment $z_k \mapsto v_k$. So $(Av \leq b) \cup (u - Dv = c)$ and $(A'v' \leq b') \cup (u' - D'v' = c')$ simplify to the same expressions and if one combined system is true, so is the other. □

The algorithm `EqBasis`$(A'x \leq b')$ decomposes the original system of inequalities $A'x \leq b'$ into a reduced system $Az \leq b$ that implies no equalities, and an equality basis $y - Dz = c$. The algorithm is guaranteed to terminate because the variable vector z decreases by one variable in each iteration. Note that `EqBasis`$(A'x \leq b')$ constructs $y - Dz = c$ in such a way that the substitution $\sigma_{y,z}^{D,c}$ is the concatenation of all substitutions σ' from every previous iteration. Therefore, we also know that $\sigma_{y,z}^{D,c}$ applied to $A'x \leq b'$ results in the system of inequalities $Az \leq b$ that implies no equalities. We exploit this fact to prove the correctness of `EqBasis`$(A'x \leq b')$, but first we need two more auxiliary lemmas.

Lemma 9 *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 $A'z \leq b' := (Ax \leq b)\sigma_{y,z}^{D,c}$ and $A^{**}z \leq b^{**} := (A^*x \leq b^*)\sigma_{y,z}^{D,c}$. 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 $v \in \mathbb{Q}_\delta^{n_z}$ 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 $u \in \mathbb{Q}_\delta^{n_y}$, where $u_i := c_i + d_i^T v$, so $(A'v \leq b') \cup (u - Dv = c)$ is true. Based on the definition of $\sigma_{y,z}^{D,c}$ and n_y recursive applications of Lemma 8, 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. □

Now we can also prove what we have already explained at the beginning of this section. The equality $h^Tx = g$ is implied by $Ax \leq b$ if and only if $y - Dz = c$ is an equality basis and $(h^Tx = g)\sigma^{D,c}_{y,z}$ simplifies to $0 = 0$. An equality basis is already defined as a set of equalities $y - Dz = c$ that implies exactly those equalities implied by $Ax \leq b$. So we only need to prove that $h^Tx = g$ is implied by $y - Dz = c$ if $(h^Tx = g)\sigma^{D,c}_{y,z}$ simplifies to $0 = 0$.

Lemma 10 *Let $y - Dz = c$ be a satisfiable system of equalities. Let $h^Tx = g$ be an equality. Then $y - Dz = c$ implies $h^Tx = g$ iff $(h^Tx = g)\sigma^{D,c}_{y,z}$ simplifies to $0 = 0$.*

Proof. First, let us look at the case where $h^Tx = g$ is an explicit equality $y_i - d_i^Tz = c_i$ in $y - Dz = c$. Then $(y_i - d_i^Tz = c_i)\sigma^{D,c}_{y,z}$ simplifies to $0 = 0$ because $\sigma^{D,c}_{y,z}$ maps y_i to $d_i^Tz + c_i$ and the variables z_j are not affected by $\sigma^{D,c}_{y,z}$.

Next, 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 $y - Dz = c$, both $(y - Dz = c)\sigma^{D,c}_{y,z}$ and $((y - Dz = c) \cup (h^Tz = g))\sigma^{D,c}_{y,z}$ must be equivalent (see Lemma 9). As we stated at the beginning of this proof, $(y_i - d_i^Tz = c_i)\sigma^{D,c}_{y,z}$ simplifies to $0 = 0$. An equality $h'^Tc = g'$ that simplifies to $0 = 0$ is true for all $v \in \mathbb{Q}_\delta^{n_z}$. Moreover, only equalities that simplify to $0 = 0$ are true for all $v \in \mathbb{Q}_\delta^{n_z}$. This means $(y - Dz = c)\sigma^{D,c}_{y,z}$ is satisfiable for all assignments and, therefore, $(h^Tz = g)\sigma^{D,c}_{y,z}$ must simplify to $0 = 0$.

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 $((y - Dz = c) \cup (h^Tz = g))\sigma^{D,c}_{y,z}$ is satisfiable for all assignments. We know based on Lemma 8 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. □

With Lemma 10, we have now all auxiliary lemmas needed to prove that the algorithm `EqBasis` is correct:

Lemma 11 *Let $A'x \leq b'$ be a satisfiable system of inequalities. Let $y - Dz = c$ be the output of* `EqBasis`$(A'x \leq b')$. *Then $y - Dz = c$ is an equality basis of $A'x \leq b'$.*

Proof. Let $Az \leq b$ be the result of applying $\sigma^{D,c}_{y,z}$ to $A'x \leq b'$. Since $y - Dz = c$ is the output of `EqBasis`$(A'x \leq b')$, the condition in line 3 of `EqBasis` guarantees us that $Az \leq b$ implies no equalities. Let us now suppose to the contrary of our initial assumptions that $A'x \leq b'$ implies an equality $h'^Tx = g'$ that $y - Dz = c$ does not imply. Since $h'^Tx = g'$ is not implied by $y - Dz = c$, the output of $(h'^Tx = g')\sigma^{D,c}_{y,z}$ is an equality $h^Tz = g$, where $h \neq 0^{n_z}$. This also implies that $(Az \leq b) \cup (h^Tz = g)$ is the output of $((A'x \leq b') \cup (h'^Tx = g'))\sigma^{D,c}_{y,z}$. By Lemma 9, $Az \leq b$ and $(Az \leq b) \cup (h^Tz = g)$ are equivalent. Therefore, $Az \leq b$ implies the equality $h^Tz = g$, which contradicts the condition in line 3 of `EqBasis` and, therefore, our initial assumptions. □

7 Implementation and Application

It is not straight forward how to efficiently integrate our method that finds an equality basis into an SMT solver. Therefore, we now explain how to implement our method as an extension of Dutertre and de Moura's version [14] of the dual simplex algorithm [2, 22]. We choose to specialize this version of the dual simplex algorithm because it is implemented in most SMT solvers and has all properties necessary for an efficient theory solver: it produces minimal conflict explanations, handles backtracking efficiently, and is highly incremental. Whenever we refer to the simplex algorithm in this section, we refer to the specific version of the dual simplex algorithm presented by Dutertre and de Moura [14].

We defined the theory for the equality basis by representing our input constraints through inequalities $Ax \leq b$ because inequalities represent the set of solutions more intuitively. In the simplex algorithm, the input constraints are represented instead by a so-called tableau $Ax = 0^m$ and two bounds $l_i \leq x_i \leq u_i$ for every variable x_i in the tableau. Therefore, it might seem difficult to efficiently integrate our method in the simplex algorithm. The truth, however, is that the tableau-and-bound representation grants us several advantages for the implementation of our equality basis method. For example, we do not have to explicitly eliminate variables via substitution, but we do so automatically via pivoting.

Later in this Section, we also explain how the integration of our methods in the simplex algorithm can be used for the combination of theories with the Nelson-Oppen Method. For the Nelson-Oppen style combination of theories inside an SMT solver [9], 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 satisfying 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 an equality basis, the guesses can be verified directly and efficiently. Therefore, the method helps the theory solver in avoiding any conflicts due to wrong guesses together with the overhead of backtracking. This comes at the price of computing the equality basis, which should be negligible because the integration we propose is incremental and includes justified simplifications.

7.1 The Dual Simplex Algorithm

The input of the simplex algorithm (Figure 5) is a set of equalities $Ax = 0^m$ 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 $Ax - s = 0^m$, 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

Algorithm 2: `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** $a_{jk} := -\frac{a_{ik}}{a_{ij}}$;
$a_{ji} := \frac{1}{a_{ij}}$
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 3: `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 4: `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 5: `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

Fig. 5 The functions of the dual simplex algorithm by Dutertre and de Moura [14]

$a_i^T x \leq b_i$ and $-a_i^T x \leq c_i$ can be transformed into the equality $a_i^T x - s_i = 0$ and the bounds $-c_i \leq s_i \leq b_i$. SMT solvers typically assign the slack variables during a preprocessing step with a normalization procedure based on a variable ordering. After the normalization, all terms are represented in one directed acyclic graph (DAG) so that all equivalent terms are assigned to the same node and, thereby, to the same slack variable. For more details on these simplifications we refer to [14].

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 represents 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. 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 $Ax = 0^m$.

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., $A(\beta(x)) = 0^m$ and $l_i \leq \beta(x_i) \leq u_i$ for every variable x_i. The algorithm starts with an assignment β that fulfills $A(\beta(x)) = 0^m$ 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 continues to fulfill $A(\beta(x)) = 0^m$ 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. Since all non-basic variables fulfill their bounds and the tableau guarantees that $Ax = 0^m$, `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 a satisfiable assignment and `Check`() returns 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 fulfilling the conditions in lines 6 or 12 of `Check`(). If it finds a non-basic variable x_j fulfilling the conditions, then we pivot x_i with x_j and update our β assignment so $\beta(x_i)$ is set to the previously violated bound value, which satisfies our invariant once more. If it finds no non-basic variable fulfilling the conditions, 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. The algorithm terminates 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.

7.2 Implementation Details

In case of the tableau-and-bound representation, an equality basis simplifies to the tableau $Ax = 0^m$ and a set of *tightly* bounded variables, i.e., a set of variables x_j such that $\beta(x_j) := l_j$ 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.

Algorithm 6: `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**
  $l'_k := l_k; u'_k := u_k$; `// Remember the original bounds`
  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 7: `FixEqs`(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** $u_j := u'_j$; $l_j := u'_j$; `update`(x_j, u_j);
    if $\beta(x_i) < l_i$ and $a_{ij} < 0$ **then** $l_j := l'_j$; $u_j := l'_j$; `update`(x_j, l_j);
    if $\beta(x_i) > u_i$ and $a_{ij} > 0$ **then** $l_j := l'_j$; $u_j := l'_j$; `update`(x_j, l_j);
    if $\beta(x_i) > u_i$ and $a_{ij} < 0$ **then** $u_j := u'_j$; $l_j := u'_j$; `update`(x_j, u_j);
  end
end
if $\beta(x_i) > u_i$ **then** $u_i := u'_i$; $l_i := u'_i$;
if $\beta(x_i) < l_i$ **then** $l_i := l'_i$; $u_i := l'_i$;

Algorithm 8: `FindTBnds`()

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**
  `FixEqs`(x_i)
end
`/* For all variables` x_k `with` $l_k < u_k$ `recover their original bounds` l'_k, u'_k `*/`
for $x_k \in \mathcal{B} \cup \mathcal{N}$ **do**
  if $l_k < u_k$ **then** $l_k := l'_k; u_k := u'_k$;
end
return *true*

Fig. 6 The functions used to turn our original tableau into a basis of equalities

To find all tightly bounded variables, we present a new extension of the simplex algorithm called `FindTBnds`() (Figure 6). This extension uses our Lemmas 6 & 7 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, `FindTBnds`() determines if our constraint system is actually satisfiable with a call of `Check`(). If the system is unsatisfiable, then it has no solutions and implies all equalities. In this case, `FindTBnds`() returns *false*.

Otherwise, we get a satisfiable assignment β from `Check`() and we use this assignment in `Initialize`() (Figure 6) 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 7). 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 $l_j = u_j$. 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 never produces 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 `FindTBnds`() checks whether the system contains another tightly bounded variable (see also Lemma 6). 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 7). `FindTBnds`() uses `FixEqs`(x_i) (Figure 6) to turn these tightly bounded variables x_j into explicit equalities by setting $l_j = u_j$. After `FixEqs`(x_i) is done, we go back to the beginning of the loop in `FindTBnds`() and do another call to `Check`.

If `Check` returns *true*, then the original system of inequalities implies no further tightly bounded variables (Lemma 6). 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 because $l_i = u_i$. Moreover, the tableau $Ax = 0^m$ and the non-basic variables that are tightly bounded represent an equality basis for our original constraint system. The simplex algorithm even represents the current tableau and the tightly bounded non-basic variables in such a way that they also describe a substitution σ for the elimination of equalities: the rows of the tableau map each basic variable x_i to their row definition $\sum_{x_j \in \mathcal{N}} a_{ij} x_j$ and the tightly bounded non-basic variables x_j, i.e., all variables x_j with $j \in \mathcal{N}$ and $l_j = u_j$, are mapped to their tight bound l_j.

After applying `FindTBnds`(), we can efficiently find all valid equations between variables as needed for the Nelson-Oppen style combination of theories. For each variable x_i, we use the substitution σ that we get from the tableau and the tightly

bounded variables to get a normalized term that represents each variable. If the variable x_i is non-basic and tightly bounded (i.e., $l_i = u_i$), then the normalized term is the constant value l_i. If the variable x_i is non-basic and not tightly bounded (i.e., $l_i \neq u_i$), then the normalized term is the variable x_i itself. If the variable x_i is basic, then the normalized term is $\left(\sum_{x_j \in \mathcal{N}, l_j \neq u_j} a_{ij} x_j\right) + \left(\sum_{x_j \in \mathcal{N}, l_j = u_j} a_{ij} l_j\right)$, where all basic mathematical operations between constant values are replaced by the results of those operations.

We know from Lemma 10 that $x_i\sigma = x_k\sigma$ simplifies to $0 = 0$ if σ is the substitution we get from an equality basis and $x_i = x_k$ is implied by our constraints. Therefore, both $x_i\sigma$ and $x_k\sigma$ must be represented by the same normalized term if x_i and x_k are equivalent. So the equality basis together with a normalization procedure has turned semantic equivalence into syntactic equivalence. It is very easy to find variables x_i represented by the same normalized term if we store these terms in a DAG, which most SMT solvers already provide for assigning slack variables.

7.3 Incrementality, Explanations, and Justifications

Note that asserting additional bounds to our system can increase the number of tightly bounded variables. In this case, we have to apply `FindTBnds`() again to find all tightly bounded variables and to complete the new equality basis. We already mentioned that `Check`() never pivots a non-basic variable x_j into a basic one if $l_j = u_j$ because of the conditions in the lines 6 & 12 of `Check`(). So even if the SMT solver asserts additional bounds for the variables and applies `Check`() again, the tightly bounded non-basic variables we have computed in the last call to `FindTBnds`() stay non-basic. Hence, our next application of `FindTBnds`() does not perform any computations for the tightly bounded variables that were detected by earlier applications of `FindTBnds`(). This means that our algorithm to compute the equality basis is highly incremental.

Another important feature of an efficient SMT theory solver is that it produces good—maybe even minimal—conflict explanations. In a typical SMT solver, a SAT solver based on CDCL (conflict-driven clause learning) selects and asserts a set of theory literals that satisfy the boolean model. Then the theory solvers verify that the asserted literals that belong to their theory are consistently satisfiable. If the theory solver finds a conflict between the asserted literals, then it returns a conflict explanation. The SAT solver uses the conflict explanation to start a conflict analysis that determines a good point for back jumping so it can select a new set of theory literals. Naturally, a good conflict explanation greatly enhances the conflict analysis and, therefore, the remaining search.

The literals asserted in our simplex based theory solver are bounds for our variables.[2] Our algorithm `FindTBnds`() asserts bounds independently of the SAT solver. This leads to problems in the conflict analysis because the conflict explanation is no longer comprehensible for the SAT solver. Hence, we have to extend

[2] Actually, the literals we assert are full inequalities $a_i^T x \leq b_i$. Due to slacking, the left side of those constraints is abstracted to a slack variable s such that $s = a_i^T x$. The definition of the slack variable $s = a_i^T x$ is directly stored in the simplex solver and only a bound $s \leq b_i$ remains as the literal for the SMT solver.

`FindTBnds`() so it produces justifications (for the bounds it asserts in `FixEqs`(x_i)) that the SAT solver can comprehend and reproduce.

We only need to justify bounds asserted by `FixEqs`(x_i) because all other bounds asserted by `FindTBnds`() are reverted to their original bounds $x_k \geq l'_k$ and $x_k \leq u'_k$. And even in `FixEqs`(x_i), we only have to justify the bounds $x_k \leq l'_k$ ($x_k \geq u'_k$) that make the tight bounds $x_k \geq l'_k$ ($x_k \leq u'_k$) explicit. We also see that the bounds asserted by `FixEqs`(x_i) are just linear combinations of existing bounds if we look again at the proof of Lemma 7. The proof also shows that we can derive this linear combination from the conflict explanation C of the strict system. For instance, if the call to `Check`() from line 3 of `FindTBnds`() exits in line 7 with (false, x_i), then the conflict explanation is
$C = \{x_i > l'_i\} \cup \{x_j < u'_j : j \in \mathcal{N} \text{ and } a_{ij} > 0\} \cup \{x_j > l'_j : j \in \mathcal{N} \text{ and } a_{ij} < 0\}$. [14]
If the call to `Check`() exits instead in line 13 with (false, x_i), then the conflict explanation is
$C = \{x_i < u'_i\} \cup \{x_j > l'_j : j \in \mathcal{N} \text{ and } a_{ij} > 0\} \cup \{x_j < u'_j : j \in \mathcal{N} \text{ and } a_{ij} < 0\}$. [14]

We receive the set of tightly propagating bounds that we found with the last call to `Check`() if we turn all bounds in C into non-strict bounds:

$$C' = \{x_k \leq u'_k : (x_k < u'_k) \in C\} \cup \{x_k \geq l'_k : (x_k > l'_k) \in C\}.$$

`FixEqs`(x_i) asserts now for every bound $(x_k \geq l'_k) \in C'$ the bound $x_k \leq l'_k$. From the proof of Lemma 7, we see that the bound $x_k \leq l'_k$ is a linear combination of the bounds $C' \setminus \{x_k \geq l'_k\}$. Hence, $x_k \leq l'_k$ is implied by the bounds $C' \setminus \{x_k \geq l'_k\}$ and, therefore, the clause

$$\left(\bigvee_{(x_i \geq l'_i) \in C', x_i \neq x_k} x_i < l'_i\right) \vee \left(\bigvee_{(x_i \leq u'_i) \in C'} x_i > u'_i\right) \vee \left(x_k \leq l'_k\right)$$

justifies the asserted bound $x_k \leq l'_k$. Together with the slack variable definitions stored in the simplex tableau, this clause is a tautology and the SAT solver can learn it without restrictions. Moreover, all literals in this clause except for $x_k \leq l'_k$ are asserted as unsatisfiable in the current model of our SAT solver. Therefore, the SAT solver can assert the literal $x_k \leq l'_k$ on its own through unit propagation.

Symmetrically, `FixEqs`(x_i) asserts for every bound $(x_k \leq u'_k) \in C'$ the bound $x_k \geq u'_k$ and the justification for this bound is the clause:

$$\left(\bigvee_{(x_i \geq l'_i) \in C'} x_i < l'_i\right) \vee \left(\bigvee_{(x_i \leq u'_i) \in C', x_i \neq x_k} x_i > u'_i\right) \vee \left(x_k \geq u'_k\right).$$

But `FindTBnds`() is not our only method that asserts literals independently of the SAT solver. If we use the equality basis computed by `FindTBnds`() for a Nelson-Oppen style combination of theories, then we also assert equalities $x_i = x_k$ for all pairs of equivalent variables x_i, x_k. Hence, we also have to justify these assertions to the SAT solver.

We get these justifications by looking at the normalized representations of the variables x_i and x_k that are equivalent. The current set of non-basic variables defines a basis and, therefore, already on its own a normalized representation for all variables. Since this normalized representation only depends on the current tableau $Ax = 0^m$, it is also independent of any of the asserted bounds. The normalized representation we use for the Nelson-Oppen style combination is only an extension of this representation by the tight bounds $x_j = c_j$ of all tightly bounded non-basic variables. Therefore, the equality $x_i = x_k$ is implied by those tight bounds $x_j = c_j$ that were actively used to compute this representation.

For instance, if x_i and x_k are both non-basic, both variables must be tightly bounded so that $x_i = x_k = v$. Otherwise, they cannot have the same normalized

representation. Therefore, $x_i = v$ and $x_k = v$ imply $x_i = x_k$, or as a clause:

$$(x_i < v \vee x_i > v) \vee (x_k < v \vee x_k > v) \vee (x_i = x_k)\,.$$

Next, we look at the case where two basic variables x_i and x_k are equivalent. But before we give the complete formal justification, let us look at an example. Let the variables x_1, x_2, x_3, x_4, x_5 be non-basic and the variables x_6 and x_7 be basic. In this example, the basic variables are defined by the non-basic variables as follows: $x_6 = 2x_1 - x_2 + 3x_4$ and $x_7 = 2x_1 - x_2 + x_5$. Moreover, let the variables x_2, x_3, x_4, and x_5 be tightly bounded such that $x_2 = 1$, $x_3 = 0$, $x_4 = 1$, and $x_5 = 3$. If we now replace the tightly non-basic variables, in the definitions of x_6 and x_7 we get that both of their normalized representations are $2x_1$ and we have actively used the tight bounds $x_2 = 1, x_4 = 1, x_5 = 3$ to compute this normalization. Hence, $x_6 = x_7$ is implied by the tight bounds $x_2 = 1$, $x_4 = 1$, and $x_5 = 3$. The variables x_6 and x_7 are also equivalent if we had not asserted that $x_2 = 1$ because the normalized representation of both variables without $x_2 = 1$ is $2x_1 - x_2$. Hence, $x_2 = 1$ is redundant for the justification and $x_6 = x_7$ is also implied by just the tight bounds $x_4 = 1$ and $x_5 = 3$.

To find which tightly bounded variables are redundant, we can just look at the coefficients. If a_{ij} and a_{kj} are the same, then any tight bound $x_j = c_j$ is redundant in the justification. This gives us the following clause as a general justification:

$$\Big(\bigvee_{j \in \mathcal{N}, l_j = u_j, a_{ij} \neq a_{kj}, (a_{kj}, a_{ij}) \neq (0,0)} x_j < l_j \vee x_j > u_j\Big) \vee x_i = x_k \tag{1}$$

From this clause, we also get the justification for the mixed case, i.e., the case where x_i is basic and x_k non-basic. We simply treat x_k as if it were defined as a basic variable by itself ($x_k = 1 \cdot x_k$), so $a_{kk} = 1$ and all other $a_{kj} = 0$. If we simplify these restrictions in the clause justification (1) for the case with two basic variables, then we receive the following general justification for the mixed case:

$$\Big(\bigvee_{j \in \mathcal{N}, l_j = u_j, a_{ij} \neq 0, (j, a_{ij}) \neq (k,1)} x_j < l_j \vee x_j > u_j\Big) \vee x_i = x_k$$

All literals in the above clauses except for $x_i = x_k$ are asserted as unsatisfiable in the current model of our SAT solver. This holds because these literals contain only tightly bounded variables. Hence, the SAT solver can assert the literal $x_i = x_k$ on its own through unit propagation. Note also that all justifications we defined are in some sense minimal: each of the above clauses is a tautology and, if we remove one literal from the clause, then it is no longer a tautology. This fact is another property that enhances any potential conflict analysis.

8 Conclusions

We have presented the linear cube transformation (Proposition 3), which allows us to efficiently determine whether a polyhedron contains a cube of a given edge length. Based on this transformation we have created two tests for linear integer arithmetic: the largest cube test and the unit cube test. Our tests can be integrated into SMT theory solvers without sacrificing the advantages that SMT solvers gain from the incremental structure of subsequent subproblems. Furthermore, our experiments have shown that these tests increase efficiency on certain polyhedra such that previously hard sets of constraints become trivial.

One obstacle for our cube tests are equalities. Resolving these obstacles led to an additional application of the linear cube transformation: investigating equalities. Through Lemmas 6 & 7, 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 $\texttt{EqBasis}(Ax \leq b)$ to compute an equality basis $y - Dz = c$, which is a finite representation of all equalities implied by the inequalities $Ax \leq b$.

We also presented various applications for the equality basis $y - Dz = c$. (1) We can use the equality basis to eliminate all equalities from $Ax \leq b$. It is, therefore, useful as a preprocessing step for our cube tests. (2) We can use the equality basis to directly check whether an equality $h^T x = g$ is implied by $Ax \leq b$. (3) In Section 7, we also use the equality basis to efficiently compute all pairs of equivalent variables in $Ax \leq b$. These pairs are necessary for a backjump-free *Nelson-Oppen* style combination of theories.

The results presented in this paper have further applications. For instance, our methods for detecting implied equalities are also useful for quantifier elimination. In general, a *quantifier elimination* (QE) procedure takes a formula $\exists y.\phi(y)$, where $\phi(y)$ itself is quantifier-free but may contain extra variables x called *parameters*, and returns an equivalent formula ϕ' that is quantifier-free. *Linear virtual substitution* is a complete QE procedure for the theory of linear rational arithmetic [23]. It eliminates the variable y by creating a case distinction exploiting the following fact: a linear real arithmetic formula $\phi(y)$ is satisfiable if and only if $\phi(l)$ is satisfiable, where l is the strictest lower bound (or upper bound) of y, i.e., the smallest value for y in any solution to the problem. This value is either represented by one of the inequalities in $\phi(y)$ containing y or $-\infty$ ($+\infty$). There are only finitely many inequalities in $\phi(y)$, so by a case distinction over all inequalities containing y satisfiability can be preserved:

$$\exists y.\phi(y) \equiv \phi(-\infty) \vee \bigvee_{a_{iy}y + a_i^T x \leq b_i \text{ in } \phi(y) \text{ with } a_{iy} < 0} \phi\left(-\frac{b_i}{a_{iy}} + \frac{a_i^T x}{a_{iy}}\right).$$

This case distinction is the source of the worst-case doubly exponential complexity of the procedure in case of quantifier alternations. At the same time, there are also instances that we can resolve without case distinctions. For instance, if the formula $\phi(y)$ implies an equality $h_y \cdot y + h^T x = g$ where $h_y \neq 0$, then we already know one guaranteed definition for the strictest lower bound of y:

$$\frac{g}{h_y} - \frac{h^T x}{h_y}.$$

A quantifier-free formula that is equivalent to the original one is simply:

$$\exists y.\phi(y) \equiv \phi\left(\frac{g}{h_y} - \frac{h^T x}{h_y}\right).$$

This technique is well-known and integrated in many QE implementations [12,23, 30]. Even so, we are unaware of any implementation that makes use of non-explicit equalities for this purpose. This is where our methods that find implicit equalities come into play. Our methods are applicable because QE procedures typically keep ϕ in a disjunctive form and the respective disjuncts contain often only conjuncts of inequalities. This allows us to efficiently search for an equality.

For future research, we plan to implement the methods around the equality basis and investigate their performance for the above mentioned applications. Moreover, we want to work out even more applications for the linear cube transformation.

Acknowledgments

The authors would like to thank the anonymous reviewers of FMSD, IJCAR 2016, and SMT 2016 for their valuable comments, suggestions, and for directing us to related work. Special thanks are also due to Bruno Dutertre, Tim King, and Andrew Reynolds for drawing our attention to additional applications.

References

1. Barrett, C., Conway, C., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: CAV, *LNCS*, vol. 6806, pp. 171–177 (2011)
2. Beale, E.M.L.: An alternative method for linear programming **50**(4), 513–523 (1954)
3. Bjørner, N.: Integrating decision procedures for temporal verification. Ph.D. thesis, Stanford, CA, USA (1999)
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: IJCAR 2012, *LNCS*, vol. 7364, pp. 67–81 (2012)
5. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press (2004)
6. Bromberger, M., Sturm, T., Weidenbach, C.: Linear integer arithmetic revisited. In: CADE-25, *LNCS*, vol. 9195, pp. 623–637 (2015)
7. Bromberger, M., Weidenbach, C.: Computing a complete basis for equalities implied by a system of LRA constraints. In: SMT 2016, pp. 15–30 (2016)
8. Bromberger, M., Weidenbach, C.: Fast cube tests for lia constraint solving. In: IJCAR 2016, *LNCS*, vol. 9706 (2016)
9. Bruttomesso, R., Cimatti, A., Franzen, A., Griggio, A., Sebastiani, R.: Delayed theory combination vs. Nelson-Oppen for satisfiability modulo theories: a comparative analysis. AMAI **55**(1), 63–99 (2009)
10. Cimatti, A., Griggio, A., Schaafsma, B., Sebastiani, R.: The MathSAT5 SMT Solver. In: TACAS, *LNCS*, vol. 7795 (2013)
11. Dillig, I., Dillig, T., Aiken, A.: Cuts from proofs: A complete and practical technique for solving linear inequalities over integers. In: CAV, *LNCS*, vol. 5643, pp. 233–247 (2009)
12. Dolzmann, A., Sturm, T., Weispfenning, V.: Real quantifier elimination in practice. In: Algorithmic Algebra and Number Theory, pp. 221–247. Springer (1999)
13. Dutertre, B.: Yices 2.2. In: CAV 2014, *LNCS*, vol. 8559 (2014)
14. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: CAV, *LNCS*, vol. 4144, pp. 81–94 (2006). Extended version: Integrating simplex with DPLL(T). Tech. rep., CSL, SRI INTERNATIONAL (2006)
15. Faure, G., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Sat modulo the theory of linear arithmetic: Exact, inexact and commercial solvers. In: SAT 2008, *LNCS*, vol. 4996, pp. 77–90 (2008)
16. Griggio, A.: A practical approach to satisfiability modulo linear integer arithmetic. JSAT **8**(1/2), 1–27 (2012)
17. Hillier, F.S.: Efficient heuristic procedures for integer linear programming with an interior. Operations Research **17**(4), 600–637 (1969)
18. Jovanović, D., de Moura, L.: Cutting to the chase. JAR **51**(1), 79–108 (2013)
19. Jünger, M., Liebling, T.M., Naddef, D., Nemhauser, G.L., Pulleyblank, W.R., Reinelt, G., Rinaldi, G., Wolsey, L.A. (eds.): 50 Years of Integer Programming 1958-2008 (2010)
20. Kannan, R., Lovász, L.: Covering minima and lattice point free convex bodies. In: FSTTCS, *LNCS*, vol. 241, pp. 193–213 (1986)
21. Karmarkar, N.: A new polynomial-time algorithm for linear programming. Combinatorica **4**(4), 373–396 (1984)
22. Lemke, C.E.: The dual method of solving the linear programming problem. Naval Research Logistics Quarterly **1**(1), 36–47 (1954)
23. Loos, R., Weispfenning, V.: Applying linear quantifier elimination. The Computer Journal **36**(5), 450–462 (1993)
24. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems, *LNCS*, vol. 4963, pp. 337–340 (2008)

25. Papadimitriou, C.H.: On the complexity of integer programming. J. ACM **28**(4), 765–768 (1981)
26. Pugh, W.: The omega test: A fast and practical integer programming algorithm for dependence analysis. In: Supercomputing 1991, Supercomputing '91, pp. 4–13. ACM (1991)
27. Refalo, P.: Approaches to the incremental detection of implicit equalities with the revised simplex method. In: PLILP 1998, *LNCS*, vol. 1490, pp. 481–496 (1998)
28. Rueß, H., Shankar, N.: Solving linear arithmetic constraints. Tech. rep., SRI International, Computer Science Laboratory (2004)
29. Schrijver, A.: Theory of Linear and Integer Programming. John Wiley & Sons, Inc., New York, NY, USA (1986)
30. Sturm, T.: Real quadratic quantifier elimination in risa/asir. Tech. rep., ISIS-RM-5E, Fujitsu Laboratories Ltd. (1996)
31. Telgen, J.: Identifying redundant constraints and implicit equalities in systems of linear constraints. Management Science **29**(10), 1209–1222 (1983)
32. Van Hentenryck, P., Graf, T.: Standard forms for rational linear arithmetic in constraint logic programming. AMAI **5**(2), 303–319 (1992)