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

Cutting to the Chase
Solving Linear Integer Arithmetic

Dejan Jovanović · Leonardo de Moura

the date of receipt and acceptance should be inserted later

Abstract We describe a new algorithm for solving linear integer programming problems. The algorithm performs a DPLL style search for a feasible assignment, while using a novel cut procedure to guide the search away from the conflicting states.

1 Introduction

One of the most impressive success stories of computer science in industrial applications was the advent of linear programming algorithms. Linear programming (LP) became feasible with the introduction of Dantzig's simplex algorithm. Although the original simplex algorithm targets problems over the rational numbers, in 1958 Gomory [16] introduced an elegant extension to the integer case (ILP). He noticed that, whenever the simplex algorithm encounters a non-integer solution, one can eliminate this solution by deriving a plane, that is implied by the original problem, but does not satisfy the current assignment. Incrementally adding these *cutting planes*, until an integer solution is found, yields an algorithm for solving linear systems over the integers. Cutting planes were immediately identified as a powerful general tool and have since been studied thoroughly both as an abstract proof system [7], and as a practical preprocessing step for hard structured problems. For such problems, one can exploit the structure by adding cuts tailored to the problem, such as the clique cuts, or the covering cuts [26], and these cuts can reduce the search space dramatically.

The main idea behind the algorithm of Gomory, i.e., to combine a model searching procedure with a conflict resolution procedure – a procedure that can derive new facts in order to eliminate a conflicting candidate solution – is in fact quite general. Somewhat later, for example, in the field of Boolean satisfiability (SAT), there was a similar development with equally impressive end results. Algorithms

Dejan Jovanović
New York University

Leonardo de Moura
Microsoft Research

and solvers for the SAT problem, although dealing with a canonical NP-complete problem, have seen a steady improvement over the years, culminating in thrilling advances in the last decade. Contrary to what one would expect of an NP-complete problem, it has become a matter of routine to use a SAT solver on problems with millions of variables and constraints. Of course, it would be naïve to attribute one single reason to this success, for there are many ingredients that contribute to the efficiency of modern SAT solvers. But, one of the most conceptually appealing techniques that these SAT solvers use is a combination of two orthogonal views on how to go about solving a satisfaction problem. One is a backtracking search for a satisfying assignment, as described in the original DPLL [10] algorithm. The other is a search for a proof that there is no solution, in this case a refutation using Boolean resolution, as described in the DP algorithm [11].

In order to combine these two approaches Silva and Sakallah [25] noticed that, although completely different, they can be used to complement each other in a surprisingly natural manner. If the search for a satisfying assignments encounters a conflicting state, i.e. one in which some clause is falsified by the current candidate assignment, one can use resolution to derive a clause, commonly called *an explanation*, that succinctly describes the conflict. As is the case with Gomory's cutting planes, this explanation clause eliminates the current assignment, so the search is forced to backtrack and consider a different one. Moreover, since this explanation is a valid deduction, it can be kept to ensure that the conflict does not occur again. These explanations can often eliminate a substantial part of the subsequent search tree. The important insight here is that the application of resolution is limited to the cases where it is needed by the search, or in other words the *search is guiding the resolution*. As is usually the case with search algorithms that attack hard problems, the search process can be greatly improved by applying heuristics at the appropriate decision points. In the case of the SAT problem, the decision of which variable to try and assign next is one of the crucial ones. With the above idea of search complemented with conflict resolution in mind, Moskewicz et al. [20] introduced the VSIDS heuristic. This heuristic prefers the variables that were involved in the resolution of recent conflicts, effectively adding the feedback in the other direction, i.e. *the resolution is guiding the search*. This approach to solving SAT problems is commonly called conflict-directed clause learning (CDCL), and is employed by most modern SAT solvers.

Unsurprisingly, the success of SAT solvers has encouraged their adoption in attacking problems from other domains, including some that were traditionally handled by the ILP solvers [3]. These ILP problems are the ones where variables are restricted to the $\{0, 1\}$ domain, and are commonly referred to as *pseudo-Boolean* (PB) problems. Although these problems still operate over Boolean variables, conflict resolution is problematic even at this level [6]. The key problem is to find an analogue conflict resolution principle for integer inequalities, since the Fourier-Moztkin resolution is imprecise for the integers, and the deduced inequalities are often *too weak* to resolve a conflict. For example, consider the inequalities

$$3x_3 + 2x_2 + x_1 \geq 4 \ , \qquad\qquad -3x_3 + x_2 + 2x_1 \geq 1 \ .$$

Assume that the search algorithm is considering a partial assignment such that $x_1 \mapsto 1$ and $x_2 \mapsto 1$. Under this assignment, the left inequality implies that $x_3 \geq 1$, and the right inequality implies that $x_3 \leq 0$. In other words, it is not possible to extend the partial assignment to x_3 and we are therefore in a conflicting state.

We can try to apply a Fourier-Motzkin resolution step to the above inequalities in order to explain the conflict. If we do so, we eliminate the variable x_3 and obtain the inequality $3x_2 + 3x_1 \geq 5$, which in the integer domain is equivalent to $x_2 + x_1 \geq 2$. This inequality is *not strong enough* to explain the conflict, as it is satisfied under the current partial assignment.

In this paper we will resolve this issue and provide an analogue of Boolean resolution, not only for PB problems, but for the general ILP case. We achieve this by introducing a technique for computing *tightly-propagating inequalities*. These inequalities are used to justify every propagation performed by our procedure, and have the property that Fourier-Moztkin resolution is precise for them. Tightly-propagating inequalities guarantee that our conflict resolution can succinctly explain each conflict.

Using the new conflict resolution procedure we then develop a CDCL-like procedure for solving arbitrary ILP problems. The procedure is inspired by recent algorithms for solving linear real arithmetic [19,18,9], and has all the important theoretical and practical ingredients that have made CDCL-based SAT solvers so successful. As in CDCL, the core of the new procedure consists of a search for an integer model that is complemented with generation of resolvents that explain the conflicts. The search process is aided with simple and efficient propagation rules that enable reduction of the search space and early detection of conflicts. The resolvents that are learned during analysis of conflicts can enable non-chronological backtracking. Additionally, all resolvents generated during the search are valid, i.e. implied by the input formula, and not conditioned by any decisions. Consequently, the resolvents can be removed when not deemed useful, allowing for flexible memory management by keeping the constraint database limited in size. Finally, *for bounded problems* all decisions (case-splits) during the search are not based on a fixed variable order, thus enabling dynamic reordering heuristics.

Existing ILP solvers can roughly be divided into two main categories: saturation solvers, and cutting-planes solvers. Saturation solvers are based on quantifier elimination procedures such as Cooper's algorithm [8] and the Omega Test [23,4]. These solvers are essentially searching for a proof, and have the same drawbacks as the DP procedure. On the other hand, the cutting-planes solvers are model search procedures, complemented with derivation of cutting planes. The main difference with our procedure is that these solvers search for a model in the rational numbers, and use the cutting-planes to eliminate non-integer solutions. Moreover, although it is a well-known fact that for every unsatisfiable ILP problem there exists a cutting-plane proof, to the best of our knowledge, there is no effective way to find this proof. Most systems based on cutting-planes thus rely on heuristics, and termination is not guaranteed. In most cases, the problem with termination is *hidden* behind the assumption that all the problem variables are bounded, which is common in traditional practical applications. In theory, this is not an invalid assumption since for any set of inequalities C, there exists an equisatisfiable set C', where every variable in C' is bounded [22]. But these theoretical bounds are of little practical value since even for very small problems (< 10 variables), unless they are of very specific structure [24], the magnitudes of the bounds obtained this way are beyond any practical algorithmic reasoning.

In contrast, our procedure *guarantees termination* directly. We describe two arguments that imply termination. First, we propose a simple heuristic for deciding when a cutting-planes based approach does not terminate, recognizing variables contributing to the divergence. Then, we show that, in such a case, one can isolate a finite number of small *conflicting cores* that are inconsistent with the corresponding current partial models. These cores consist of two inequalities and at most one divisibility constraint. Finally, we apply Cooper's quantifier elimination procedure to derive a resolvent that will *block* a particular core from ever happening again, which in turn implies termination. And, as a matter of practical importance, the resolvents do not involve disjunctions and are expressed only with valid inequalities and divisibility constraints.

2 Preliminaries

As usual, we will denote the set of integers as $\mathbb{Z}$. We assume a finite set of variables X ranging over $\mathbb{Z}$ and use x, y, z, k to denote variables, a, b, c, d to denote constants from $\mathbb{Z}$, and p, q, r and s for linear polynomials over X with coefficients in $\mathbb{Z}$. In the following, all polynomials are assumed to be in sum-of-monomials normal form

$$a_1 x_1 + \cdots + a_n x_n + c \enspace .$$

Given a polynomial $p = a_1 x_1 + \ldots + a_n x_n + c$, and a constant b, we use bp to denote the polynomial $(a_1 b)x_1 + \ldots + (a_n b)x_n + (bc)$.

The main constraints we will be working with are *linear inequalities*, which are of the form

$$a_n x_n + \cdots + a_1 x_1 + c \leq 0 \enspace ,$$

and we denote them with letters I and J. We assume the above form for all inequalities as, in the case of integers, we can rewrite $p < 0$ as $p + 1 \leq 0$, and $p = 0$ as $(p \leq 0) \wedge (-p \leq 0)$. In order to isolate the coefficient of a variable x in a linear polynomial p (inequality I), we will write $\mathsf{coeff}(p, x)$ ($\mathsf{coeff}(I, x)$), and we define $\mathsf{coeff}(p, x) = 0$ if x does not occur in the polynomial p (inequality I).

Definition 1 (Tightly-Propagating Inequality) We say that an inequality I is *tightly-propagating* for a variable x, if the coefficient with x in the inequality I is unit, i.e. if $\mathsf{coeff}(I, x) \in \{-1, 1\}$.

We call a function v that maps variables to integer values a variable assignment. A constraint $a_n x_n + \cdots + a_1 x_1 + c \leq 0$ is satisfied by a variable assignment v if all the variables $x_1, \ldots, x_n$ are assigned by v and $a_n\, v(x_n) + \cdots + a_1\, v(x_1) + c \leq 0$. A set of constraints C is satisfiable if there is an assignment that satisfies all constraints in C. Otherwise the set of constraints C is unsatisfiable. Finally, given a set of constraints C and a constraint I, we use $C \vdash_{\mathbb{Z}} I$ to denote that I is implied by C in the theory of linear integer arithmetic.

3 A Cutting-Planes Proof System

In this section, we introduce a cutting-planes proof system that will be the basis of our procedure. Each rule consists of the premises on the top and derives the conclusion at the bottom of the rule, with the necessary side-conditions presented in the box on the side.

The Combine rule derives a positive linear combination of two linear integer inequalities.

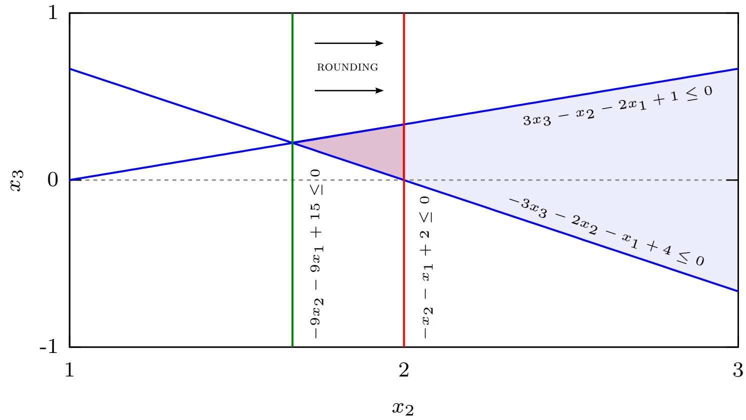

Fig. 1 Cutting-plane derivation of Example 1.

$$\textsc{Combine}\ \frac{I_1 \qquad I_2}{\lambda_1 I_1 + \lambda_2 I_2}\ \textbf{if}\ \boxed{\lambda_1, \lambda_2 > 0}$$

A special case of the above rule is the resolution step used in the Fourier-Motzkin elimination procedure that eliminates the top variable from a pair of inequalities $-ax + p \leq 0$ and $bx - q \leq 0$, generating the inequality $bp - aq \leq 0$.

The Combine rule is a valid deduction in both rational and integer arithmetic. In the context of integers arithmetic, the main deductive step, essential to any cutting-planes proof system, is based on strengthening an inequality by rounding. The Normalize rule divides an inequality with the greatest common divisor of variable coefficients, while rounding the free constant.

$$\textsc{Normalize}\ \frac{a_1 x_1 + \ldots + a_n x_n + c \leq 0}{\frac{a_1}{d} x_1 + \ldots + \frac{a_n}{d} x_n + \lceil \frac{c}{d} \rceil \leq 0}\ \textbf{if}\ \boxed{d = \mathsf{gcd}(a_1, \ldots, a_n)}$$

Example 1 Consider the two inequalities

$$-3x_3 - 2x_2 - x_1 + 4 \leq 0 \qquad\qquad 3x_3 - x_2 - 2x_1 + 1 \leq 0\ .$$

We can apply the combine rule with coefficients $\lambda_1 = \lambda_2 = 1$, simulating Fourier-Motzkin elimination, to derive an inequality where the top variable x_3 is eliminated, and then normalizing the result, obtaining the following derivation.

$$\textsc{Combine}\ \cfrac{-3x_3 - 2x_2 - x_1 + 4 \leq 0 \qquad 3x_3 - x_2 - 2x_1 + 1 \leq 0}{\textsc{normalize}\ \cfrac{-3x_2 - 3x_1 + 5 \leq 0}{-x_2 - x_1 + 2 \leq 0}}$$

This derivation is depicted in Figure 1, where it is more evident how the rounding helps eliminate the non-integer parts of the solution space. The shaded part of the figure corresponds to the real solutions of the inequalities at $x_1 = 0$, and the part of this space that does not contain any integer solutions is removed (cut off) by the derived inequality (cutting plane). Since we are interested only in integer solutions, performing rounding on an inequality corresponds to pushing the hyper-plane that defines the border of the space defined by the inequality, until it touches at least one integer point.

4 The Abstract Search Procedure

We describe our procedure as an abstract transition system in the spirit of the Abstract DPLL procedure [21]. The states of the transition system are pairs of the form $\langle M, C\rangle$, where M is a sequence of *bound refinements*, and C is a set of constraints. We use $[\![]\!]$ to denote the empty sequence. Bound refinements in M can be either *decisions* or *implied bounds*. Decided lower and upper bounds are decisions we make during the search, and we represent them in M as $x \geq b$ and $x \leq b$. On the other hand, lower and upper bounds that are implied in the current state by some inequality I, are represented as $x \geq_I b$ and $x \leq_I b$. We say that a sequence of bound refinements M is *non-redundant* if, for all variables x, the bound refinements in M are monotone, i.e. all the lower (upper) bounds are increasing (decreasing), and M does not contain the same bound for x, decided or implied.

Let $\mathsf{lower}(x, M)$ and $\mathsf{upper}(x, M)$ denote the strongest, either decided or implied, lower and upper bounds for the variable x in the sequence M, where we assume the usual values of $-\infty$ and ∞ when the corresponding bounds do not exist. We say that a sequence M is *consistent* if there is no variable x such that $\mathsf{lower}(x, M) > \mathsf{upper}(x, M)$. We lift the lower and upper bound functions to linear polynomials using identities such as: $\mathsf{lower}(p+q, M) = \mathsf{lower}(p, M) + \mathsf{lower}(q, M)$, when variables in p and q are disjoint, $\mathsf{lower}(b, M) = b$, and $\mathsf{lower}(ax, M) = a(\mathsf{lower}(x, M))$ if $a > 0$, and $\mathsf{lower}(ax, M) = a(\mathsf{upper}(x, M))$ otherwise.[1]

If in a sequence M, a variable x has both of its bounds equal, i.e. if $\mathsf{lower}(x, M) = \mathsf{upper}(x, M)$, we say that the variable x is *fixed*. Similarly a polynomial p is fixed if all of its variables are fixed. To clarify the presentation, for fixed variables and polynomials we write $\mathsf{val}(x, M)$ and $\mathsf{val}(p, M)$ as a shorthand for $\mathsf{lower}(x, M)$ and $\mathsf{lower}(p, M)$. Given a sequence M, with variables $x_1, \ldots, x_n$ fixed, we can construct a variable assignment $v[M]$ that maps each variable x_i to the value $\mathsf{val}(x_i, M)$.

Given a sequence of bound refinements M and an inequality I that contains a variable x, the inequality I implies a bound on x assuming the bounds in M. To capture this we define the function $\mathsf{bound}(I, x, M)$ representing the implied bound as

$$\mathsf{bound}(ax + p \leq 0, x, M) = \begin{cases} -\lceil \frac{\mathsf{lower}(p,M)}{a} \rceil & \text{if } a > 0 \ , \\ -\lfloor \frac{\mathsf{lower}(p,M)}{a} \rfloor & \text{if } a < 0 \ . \end{cases}$$

Above, if $a > 0$ the computed bound is an upper bound on the variable x, and if $a < 0$ it is a lower bound on x.

Example 2 Consider the inequalities $I_1 \equiv -3x_3 - 2x_2 - x_1 + 4 \leq 0$, $I_2 \equiv 3x_3 - x_2 - 2x_1 + 1 \leq 0$, and the sequence of bound refinements $M = [\![x_1 \leq 1, x_2 \leq 3]\!]$. Knowing the bounds on x_1 and x_2, the inequality I_1 implies a lower bound on x_3, and the inequality I_2 implies an upper bound on x_3. We have that

$$\mathsf{lower}(-2x_2 - x_1 + 4, M) = -2\mathsf{upper}(x_2, M) - \mathsf{upper}(x_1, M) + 4 = -3 \ ,$$
$$\mathsf{lower}(-x_2 - 2x_1 + 1, M) = -\mathsf{upper}(x_2, M) - 2\mathsf{upper}(x_1, M) + 1 = -4 \ .$$

Therefore, inequality I_1 implies the lower bound $\mathsf{bound}(I_1, x_3, M) = -\lfloor \frac{-3}{-3} \rfloor = -1$, and inequality I_2 implies the upper bound $\mathsf{bound}(I_2, x_3, M) = -\lceil \frac{-4}{3} \rceil = 1$.

[1] In general, when estimating bounds of polynomials, since two polynomials might have variables in common, for a consistent sequence M it holds that, if $\mathsf{lower}(p, M)$ and $\mathsf{lower}(q, M)$ are defined, then $\mathsf{lower}(p + q, M) \geq \mathsf{lower}(p, M) + \mathsf{lower}(q, M)$.

Definition 2 (Well-Formed Sequence) We say a sequence M is *well-formed* with respect to a set of constraints C when M is non-redundant, consistent and M is either an empty sequence or is of the form $M = [\![M', \gamma]\!]$, where the prefix M' is well-formed and the bound refinement γ is either

- $x \geq_I b$, with $I \equiv (-x + q \leq 0)$, $C \vdash_{\mathbb{Z}} I$, and $b \leq \mathsf{lower}(q, M')$; or
- $x \leq_I b$, with $I \equiv (x - q \leq 0)$, $\mathcal{C} \vdash_{\mathbb{Z}} I$, and $b \geq \mathsf{upper}(q, M')$; or
- $x \geq b$, where M' contains $x \leq_I b$; or
- $x \leq b$, where M' contains $x \geq_I b$.

Intuitively, in a well-formed sequence, every decision $x \geq b$ ($x \leq b$) amounts to *deciding* a value for x that is equal to the best upper (lower) bound so far in the sequence. Additionally all the *implied bounds are justified by tight inequalities* that are implied in $\mathbb{Z}$ by the set of constraints C. We say that a state $\langle M, C \rangle$ is well-formed if M is well-formed with respect to C.

Note that in the first two properties, when refining a bound, we allow the new bound b to *not necessarily be the most precise one* with respect to I. The reason behind this is a practical one. As we show later, given any inequality that propagates a better bound on a variable we can compute a tightly-propagating inequality that implies the same (or better) bound. But, since the procedure for computing tightly-propagating inequalities is non-trivial, in practice it is desirable to compute these tightly-propagating inequalities on-demand. During the search process we propagate new bounds using the existing (possibly non-tight) inequalities, and we compute their tight counterparts only when needed during conflict analysis. The computed tightly-propagating inequalities can often be stronger than the original inequality, implying a better bound than the original one, and the allowance of this definition enables us to compute them on-demand.

Given an implied lower (upper) bound refinement $x \geq_I b$ ($x \leq_I b$) and an inequality $ax + p \leq 0$, we define the function $\mathsf{resolve}$ that combines (if possible) the tight inequality $I \equiv \pm x + q \leq 0$ with $ax + p \leq 0$ to eliminate the variable x. If the combination is not applicable, $\mathsf{resolve}$ just returns $ax + p \leq 0$. It is defined as

$$\begin{matrix} \mathsf{resolve}(x \geq_I b, ax + p \leq 0) \\ \mathsf{resolve}(x \leq_I b, ax + p \leq 0) \end{matrix} = \begin{cases} |a|q + p \leq 0 & \text{if } a \times \mathsf{coeff}(I, x) < 0 \ , \\ ax + p \leq 0 & \text{otherwise} \ . \end{cases}$$

The $\mathsf{resolve}$ function will be used in conflict resolution due to the property that it eliminates the variable x if possible, while keeping valid deductions, and the fact that it preserves (or even improves) the bounds that can be implied. The following lemma states this property more precisely.

Lemma 1 *Given a well-formed state $\langle M, C \rangle$, with $M = [\![M', \gamma]\!]$, such that γ is an implied bound, $p \leq 0$ an inequality, and $q \leq 0 \equiv \mathsf{resolve}(\gamma, p \leq 0)$ then*

$$C \vdash_{\mathbb{Z}} (p \leq 0) \quad \textit{implies} \quad C \vdash_{\mathbb{Z}} (q \leq 0) \ , \tag{1}$$

$$\mathsf{lower}(q, M') \geq \mathsf{lower}(p, M) \ . \tag{2}$$

Proof Having that γ is an implied bound, we need only consider the following two cases:

1. γ is of the form $x \geq_I b$, where $I \equiv (-x + r \leq 0)$;
2. γ is of the form $x \leq_I b$, where $I \equiv (x - r \leq 0)$;

Let us consider only the first case, as the proof of the second case is similar. Since $\langle M, C\rangle$ is a well-formed state and $M = [\![M', \gamma]\!]$, we have that $b \leq \mathsf{lower}(r, M')$ and $C \vdash_{\mathbb{Z}} -x + r \leq 0$. We consider two cases based on the sign of the coefficient of x in p.

- If p is of the form $-ax + s$, for some $a \geq 0$ then by the definition of $\mathsf{resolve}$, we have that $\mathsf{resolve}(\gamma, p \leq 0) = p \leq 0$. Then, $q = p$, and $C \vdash_{\mathbb{Z}} (q \leq 0)$. Since $\mathsf{lower}(p, M) = -a\mathsf{upper}(x, M) + \mathsf{lower}(s, M)$, and removing the lower bound γ from M does not influence $\mathsf{upper}(x, M)$, we have that $\mathsf{lower}(p, M) = \mathsf{lower}(p, M') = \mathsf{lower}(q, M')$.
- If p is of the form $ax + s$, for some $a > 0$. By the definition of $\mathsf{resolve}$, we have that $\mathsf{resolve}(\gamma, p \leq 0) = ar + s \leq 0$. Then, $C \vdash_{\mathbb{Z}} (q \leq 0)$, since $q = ar + s$ is a positive linear combination of the inequalities $p \leq 0$ and $-x + r \leq 0$. Finally, we have that

$$\begin{aligned} \mathsf{lower}(q, M') &= \mathsf{lower}(ar + s, M') \geq a(\mathsf{lower}(r, M')) + \mathsf{lower}(s, M') \\ &\geq a(\mathsf{lower}(x, M)) + \mathsf{lower}(s, M) = \mathsf{lower}(p, M) \ . \end{aligned}$$

Since in both of the cases the statement holds, this concludes the proof.

Example 3 In the statement of Lemma 1, we get to keep (or improve) the bound $\mathsf{lower}(q, M') \geq \mathsf{lower}(p, M)$ only because all of the implied bounds were justified by tightly-propagating inequalities. If we would allow non-tight justifications, this might not hold. Consider, for example, a state $\langle M, C\rangle$ where

$$C = \{\overbrace{-x \leq 0}^{I},\ \overbrace{-3y + x + 2 \leq 0}^{J}\}\ , \qquad\qquad M = [\![x \geq_I 0,\ y \geq_J 1]\!]\ ,$$

i.e. the propagation of the bound on y is propagated by a non-tight inequality J. Now, consider the inequality $1 + 6y \leq 0$. We have that

$$\mathsf{resolve}(y \geq_J 1,\ 1 + 6y \leq 0) = 2x + 5 \leq 0\ .$$

After performing resolution on y using a non-tight inequality J, the inequality became weaker since

$$\mathsf{lower}(2x + 5, [\![x \geq_I 0]\!]) = 5\ , \qquad\qquad \mathsf{lower}(1 + 6y, M) = 7\ .$$

Finally, we define a predicate $\mathsf{improves}(I, x, M)$ as a shorthand for stating that the inequality $I \equiv ax + p \leq 0$ implies a better bound for x in M, but does not make M inconsistent. It is defined as

$$\mathsf{improves}(I, x, M) = \begin{cases} \mathsf{lower}(x, M) < \mathsf{bound}(I, x, M) \leq \mathsf{upper}(x, M), & \text{if } a < 0, \\ \mathsf{lower}(x, M) \leq \mathsf{bound}(I, x, M) < \mathsf{upper}(x, M), & \text{if } a > 0, \\ \mathsf{false}, & \text{otherwise.} \end{cases}$$

4.1 Deriving tight inequalities

Since we require that all the implied bound refinements in a well-formed sequence M are justified by tightly-propagating inequalities, and we've hinted that this is important for a concise conflict resolution procedure, we will now show how to deduce such tightly-propagating inequalities when needed in bound refinement. Given an inequality $\pm ax + p \leq 0$ such that $\mathsf{improves}(\pm ax + p \leq 0, x, M)$ holds, we show how to deduce a tightly propagating inequality that can justify the improved bound implied by $\pm ax + p \leq 0$.

The intuition behind the derivation is the following. Starting with an inequality $I \equiv ax + b_1 y_1 + \cdots + b_n y_n \leq 0$, that implies a bound on x, we will transform it using valid deduction steps into an inequality where all coefficients are divisible by a. We can do this since, in order for I to be able to imply a bound on x, the appropriate bounds for the variables $y_1, \ldots, y_n$ have to exist, and moreover these bounds are justified by tightly-propagating inequalities. For example, the bound on variable y_1 might be justified by the inequality $J \equiv y_1 + q \leq 0$. If so, we can add the inequality J to I as many times as needed to make the coefficient with y_1 divisible by a.

The deduction is described using an auxiliary transition system with the states of this system being tuples of the form

$$\left\langle M', \pm ax + as \oplus r \right\rangle \ ,$$

where $a > 0$, s and r are polynomials, M' is a prefix of the initial M, and we keep the invariant that

$$C \vdash_{\mathbb{Z}} \pm ax + as + r \leq 0, \qquad \mathsf{lower}(as + r, M) \geq \mathsf{lower}(p, M) \ .$$

The invariant above states that the derived inequality is a valid deduction that implies at least as strong of a bound on x, while the coefficients to the left of the delimiter symbol $\oplus$ are divisible by a.

The initial state for tightening of the inequality $\pm ax + p \leq 0$ is $\langle M, \pm ax \oplus p \rangle$ and the transition rules are listed below.

Consume
$\langle M, \pm ax + as \oplus aky + r \rangle \implies \langle M, \pm ax + as + aky \oplus r \rangle$
where $x \neq y$.
Resolve-Implied
$\langle [\![M, \gamma]\!], \pm ax + as \oplus p \rangle \implies \langle M, \pm ax + as \oplus q \rangle$
where γ is an implied bound and $q \leq 0 \equiv \mathsf{resolve}(\gamma, p \leq 0)$
Decided-Lower
$\langle [\![M, y \geq b]\!], \pm ax + as \oplus cy + r \rangle \implies \langle M, \pm ax + as + aky \oplus r + (ak - c)q \rangle$
where $y \leq_I b$ in M, with $I \equiv y + q \leq 0$, and $k = \lceil c/a \rceil$.
Decided-Lower-Neg
$\langle [\![M, y \geq b]\!], \pm ax + as \oplus cy + r \rangle \implies \langle M, \pm ax + as \oplus cq + r \rangle$
where $y \leq_I b$ in M, with $I \equiv y - q \leq 0$, and $c < 0$.
Decided-Upper
$\langle [\![M, y \leq b]\!], \pm ax + as \oplus cy + r \rangle \implies \langle M, \pm ax + as + aky \oplus r + (c - ak)q \rangle$
where $y \geq_I b$ in M, with $I \equiv -y + q \leq 0$, and $k = \lfloor c/a \rfloor$.
Decided-Upper-Pos

$\langle [\![M, y \leq b]\!], \pm ax + as \oplus cy + r\rangle \quad \Longrightarrow \quad \langle M, \pm ax + as \oplus cq + r\rangle$
where $y \geq_I b$ in M, with $I \equiv -y + q \leq 0$, and $c > 0$.

Round (and terminate)
$\langle M, \pm ax + as \oplus b\rangle \quad \Longrightarrow \quad \pm x + s + \lceil b/a \rceil \leq 0$

We use $\mathsf{tight}(I, x, M)$ to denote the tightly propagating inequalities derived using some strategy for applying the transition rules above.

Note that, at a particular state, there might be more than one rule that is applicable. For example, if the Decided-Lower-Neg rule is enabled, then so is the Decided-Lower rule (and similarly for the Decided-Upper-Pos and Decided-Upper rules). The Decided-Lower-Neg and Decided-Upper-Pos rules eliminate the variable y, due to the appropriate sign of the coefficient c with variable y. On the other side, the Decided-Upper and Decided-Lower rules only make the coefficient with variable y divisible by a. Additionally, any variable y that does not have a decided bound can always be eliminated completely, by choosing not to apply apply the Consume rule. This can be used if we wish to completely eliminate a variable from an inequality, which we use in Section 5.

Example 4 Given a well-formed state $\langle M, C\rangle$, where

$$C = \{\underbrace{-y \leq 0}_{I_1}, \underbrace{-x + 2 \leq 0}_{I_2}, \underbrace{-y + 7 + x \leq 0}_{I_3}, \underbrace{-3z + 2y - 5x \leq 0}_{I_4}\}$$
$$M = [\![\ y \geq_{I_1} 0,\ \ x \geq_{I_2} 2,\ \ y \geq_{I_3} 9,\ \ x \leq 2\ ]\!]$$

In this state we have that $\mathsf{bound}(I_4, z, M) = 3$, that is, I_4 is implying a lower bound of z in the current state Since I_4 is not tightly-propagating on z. we now derive a tight inequality that justifies this lower bound by applying the rules as we go backwards in the trail of bound refinements.

$\langle [\![\ y \geq_{I_1} 0,\ \ x \geq_{I_2} 2,\ \ y \geq_{I_3} 9,\ \ x \leq 2\ ]\!], -3z \oplus 2y - 5x\rangle$
$\Longrightarrow$ Decided-Upper
$x \leq 2$ is a decided bound, M contains implied bound $x \geq_{I_2} 2$.
We make the coefficient of x divisible by 3 by adding $I_2 \equiv -x + 2 \leq 0$.
$\langle [\![\ y \geq_{I_1} 0,\ \ x \geq_{I_2} 2,\ \ y \geq_{I_3} 9\ ]\!], -3z - 6x \oplus 2y + 2\rangle$
$\Longrightarrow$ Resolve-Implied
We eliminate y by adding two times $I_3 \equiv -y + 7 + x \leq 0$.
$\langle [\![\ y \geq_{I_1} 0,\ \ x \geq_{I_2} 2\ ]\!], -3z - 6x \oplus 2x + 16\rangle$
$\Longrightarrow$ Resolve-Implied
We eliminate x in $2x + 16$ by adding two times $I_2 \equiv -x + 2 \leq 0$.
$\langle [\![\ y \geq_{I_1} 0\ ]\!], -3z - 6x \oplus 20\rangle$
$\Longrightarrow$ Round
$-z - 2x + 7 \leq 0$

The derived tightly propagating inequality $-z - 2x + 7 \leq 0$ implies the same lower bound $\mathsf{bound}(-z - 2x + 7 \leq 0, z, M) = 3$ for z.

The following lemma shows that by deriving tightly propagating inequalities using the system above we do not lose precision in terms of the bounds that the inequality can imply.

Lemma 2 *Given a well-formed state $\langle M, C\rangle$ and an implied inequality I, i.e. such that $C \vdash_{\mathbb{Z}} I$, and $\mathsf{improves}(I, x, M)$ the procedure for deriving tightly-propagating inequalities terminates with a tight-inequality J such that $C \vdash_{\mathbb{Z}} J$ and*

- *if I improves the lower bound on x, then $\mathsf{bound}(I, x, M) \leq \mathsf{bound}(J, x, M)$,*
- *if I improves the upper bound on x, then $\mathsf{bound}(I, x, M) \geq \mathsf{bound}(J, x, M)$.*

Proof Note that for any inequality $I \equiv \pm ax + p \leq 0$ as in the statement of the lemma, i.e. one that improves a bound of x in M, the bounds of all variables from I, except for maybe x, are justified in M. Moreover, since we're in a well-formed state, all of the inequalities that justify these bounds are also tightly-propagating.

For the initial state $\langle M, \pm ax \oplus p\rangle$, all the variables in p have a bound in M. The transition system then keeps the following invariants for any reachable state $\langle M_k, \pm ax + q \oplus r\rangle$:

(a) all the variables in r have a bound in M_k;
(b) all the variables in q have a bound in M;
(c) all the coefficients in q are divisible by a; and
(d) $\mathsf{lower}(q + r, M) \geq \mathsf{lower}(p, M)$.

Proving these invariants is an easy exercise, with the interesting and important case being (d), which follows in a manner similar to Lemma 1. The cases where the transition rule eliminates a variable follow as in Lemma 1. Assume therefore that we are in the case when we don't eliminate the top variable. For example, assume a state where the decided lower bound of y in M_k (and hence in M) is at b.

$$\langle [\![M_{k-1}, y \geq b]\!], \pm ax + q \oplus r\rangle \quad ,$$

Then y must have an implied upper bound $y \leq_I b$ in M_{k-1}, and we add a positive multiple of a tightly-propagating inequality $I \equiv y + t \leq 0$. Note that in M_{k-1}, by property of implied bounds in the definition of the well-formed state, we have that $b \geq \mathsf{upper}(-t, M_{k-1}) = -\mathsf{lower}(t, M_{k-1})$. Now, for any $\lambda > 0$ we then have

$$\begin{aligned}
\mathsf{lower}(q + r + \lambda(y + t), M) &\geq \mathsf{lower}(q + r, M) + \lambda(\mathsf{lower}(y, M) + \mathsf{lower}(t, M)) && (3)\\
&\geq \mathsf{lower}(q + r, M) + \lambda(\mathsf{lower}(y, M) + \mathsf{lower}(t, M_{k-1})) && (4)\\
&\geq \mathsf{lower}(q + r, M) + \lambda(\mathsf{lower}(y, M) - b) && (5)\\
&= \mathsf{lower}(q + r, M) \ . && (6)
\end{aligned}$$

The inequality (3) holds just through computation of lower and it is not an equality as some the variables in the terms might be shared, as discussed in the definition of lower. The inequality (4) holds as lower bounds on terms can only increase in a well-formed state and M_{k-1} is a subsequence of M. The inequality (5) holds since as discussed above we have that $\mathsf{lower}(t, M_{k-1}) \geq -b$. Finally, (6) holds simply be definition of lower when a variable has a decided value in a well-formed state.

The improvement of bounds stated in the lemma then easily follows from (d). Termination follows directly, as at least one of the rules is always applicable (using (a)), and each rules either consumes a part of the sequence M or terminates. The length of the derivation is therefore bounded by the length of the sequence M.

Note that in the statement above, $\mathsf{improves}(J, x, M)$ does not necessarily hold, although the implied bound is the same or better. This is because the improves predicate requires the new bound to be consistent, and the derived inequality might in fact imply a stronger bound that can be in conflict.

4.2 Main procedure

We are now ready to define the main transition system of the decision procedure. In the following system of rules, if a rule can derive a new implied bound $x \geq_I b$ or $x \leq_I b$, the tightly propagating inequality I is written as if computed eagerly. This simplification clarifies the presentation, but we can use them as just placeholders and compute them on demand, which is what we do in our implementation. The transition rules alternate between the *search phase* with states denoted as $\langle M, C \rangle$, where new bounds are propagated and decisions on variables are made, and the *conflict resolution* phase with states denoted as $\langle M, C \rangle \vdash I$, where we try to explain the conflict encountered by the search phase. We call the inequality I in the conflict-resolution states the conflicting inequality. The conflicting inequality $I \equiv p \leq 0$ will always be implied by C and be in conflict with the current bound ($\mathsf{lower}(p, M) > 0$).

Search rules. The search phase of the transition system can either propagate a new bound on a variable using one of the Propagate rules, or decide a new bound of variable using one of the Decide rules. As mentioned before, the Propagate rule infers the new bound using a possibly non-tight inequality J, and its tight counterpart I is computed so as to enable conflict analysis. Both rules keep the state consistent while transitioning from well-formed states to well-formed states. If an inconsistency is detected, we transition into the conflict analysis phase using the Conflict rule. In addition to these basic rules, if we detect that some inequality can be inferred from other inequalities, we can remove it using the Forget rule.[2]

Decide

$$\langle M, C \rangle \implies \langle [\![M, x \geq b]\!], C \rangle \quad \textbf{if} \begin{cases} \mathsf{upper}(x, M) \neq +\infty \\ \mathsf{lower}(x, M) < b = \mathsf{upper}(x, M) \end{cases}$$

$$\langle M, C \rangle \implies \langle [\![M, x \leq b]\!], C \rangle \quad \textbf{if} \begin{cases} \mathsf{lower}(x, M) \neq -\infty \\ \mathsf{lower}(x, M) = b < \mathsf{upper}(x, M) \end{cases}$$

Propagate

$$\langle M, C \cup \{J\} \rangle \implies \langle [\![M, x \geq_I b]\!], C \cup \{J\} \rangle \quad \textbf{if} \begin{cases} \mathsf{coeff}(J, x) < 0 \\ \mathsf{improves}(J, x, M), \\ b = \mathsf{bound}(J, x, M). \\ I = \mathsf{tight}(J, x, M), \end{cases}$$

$$\langle M, C \cup \{J\} \rangle \implies \langle [\![M, x \leq_I b]\!], C \cup \{J\} \rangle \quad \textbf{if} \begin{cases} \mathsf{coeff}(J, x) > 0 \\ \mathsf{improves}(J, x, M), \\ b = \mathsf{bound}(J, x, M). \\ I = \mathsf{tight}(J, x, M), \end{cases}$$

Conflict

$$\langle M, C \rangle \implies \langle M, C \rangle \vdash p \leq 0 \quad \textbf{if } p \leq 0 \in C,\ \mathsf{lower}(p, M) > 0$$

Sat

$$\langle M, C \rangle \implies \langle v[M], \mathsf{sat} \rangle \quad \textbf{if } v[M] \text{ satisfies } C$$

Forget

$$\langle M, C \cup \{J\} \rangle \implies \langle M, C \rangle \quad \textbf{if } C \vdash_{\mathbb{Z}} J, \text{ and } J \notin C$$

[2] This rule can be used to remove the new inequalities that were learned during conflict analysis.

Conflict analysis rules. After entering the conflict resolution phase the conflict-resolution rules are used to backtrack the search and learn a reason for the detected conflict as we traverse the bound sequence backwards.

Resolve

$$\langle [\![M, \gamma]\!], C \rangle \vdash I \implies \langle M, C \rangle \vdash \mathsf{resolve}(\gamma, I) \quad \textbf{if} \quad \gamma \text{ is an implied bound.}$$

Skip-Decision

$$\langle [\![M, \gamma]\!], C \rangle \vdash p \leq 0 \implies \langle M, C \rangle \vdash p \leq 0 \qquad \textbf{if} \begin{cases} \gamma \text{ is a decided bound} \\ \mathsf{lower}(p, M) > 0 \end{cases}$$

Unsat

$$\langle [\![M, \gamma]\!], C \rangle \vdash b \leq 0 \implies \mathsf{unsat} \qquad \textbf{if} \ b > 0$$

Backjump

$$\langle [\![M, \gamma, M']\!], C \rangle \vdash J \implies \langle [\![M, x \geq_I b]\!], C \rangle \qquad \textbf{if} \begin{cases} \gamma \text{ is a decided bound} \\ \mathsf{coeff}(J, x) < 0 \\ \mathsf{improves}(J, x, M), \\ I = \mathsf{tight}(J, x, M), \\ b = \mathsf{bound}(J, x, M). \end{cases}$$

$$\langle [\![M, \gamma, M']\!], C \rangle \vdash J \implies \langle [\![M, x \leq_I b]\!], C \rangle \qquad \textbf{if} \begin{cases} \gamma \text{ is a decided bound} \\ \mathsf{coeff}(J, x) > 0 \\ \mathsf{improves}(J, x, M), \\ I = \mathsf{tight}(J, x, M), \\ b = \mathsf{bound}(J, x, M). \end{cases}$$

Learn

$$\langle M, C \rangle \vdash I \implies \langle M, C \cup I \rangle \vdash I \qquad \textbf{if} \ I \notin C$$

When applying any of the presented search rules, any newly introduced inequality is either a tight version of existing inequalities, or introduced during conflict resolution. In both cases we can see from Lemma 1 and Lemma 2 and simple inductive reasoning that these new inequalities are always implied by the original problem. This observation and standard case analysis on the rules can be used to show the soundness of the transition system.

Theorem 1 (Soundness) *For any derivation sequence $\langle [\![]\!], C_0 \rangle \implies S_1 \implies \cdots \implies S_n$, if S_n is of the form $\langle M_n, C_n \rangle$, then C_0 and C_n are equisatisfiable. If S_n is of the form $\langle M_n, C_n \rangle \vdash I$, then C_0 implies I, and C_0 and C_n are equisatisfiable. Moreover, $\langle M_n, C_n \rangle$ is well-formed.*

From the theorem above it is easy to see that if the transition system enters the unsat state, then the original constraints C_0 are unsatisfiable. The only way to enter the unsat state is by deriving a trivial false inequality $b \leq 0$, for some $b > 0$, and since this inequality is implied by C_n, it must be that C_n (and therefore C_0) is unsatisfiable.

Example 5 Consider the set of inequalities C

$$\{\underbrace{-x \leq 0}_{I_1}, \ \underbrace{6x - 3y - 2 \leq 0}_{I_2}, \ \underbrace{-6x + 3y + 1 \leq 0}_{I_3}\}$$

Now we show C to be unsatisfiable using our abstract transition system.

$\langle [\![]\!], C\rangle$
$\Longrightarrow$ $\mathsf{Propagate}$ x using $I_1 \equiv -x \leq 0$
$\langle [\![x \geq_{I_1} 0]\!], C\rangle$
$\Longrightarrow$ Decide x
$\langle [\![x \geq_{I_1} 0,\ x \leq 0]\!], C\rangle$
$\Longrightarrow$ $\mathsf{Propagate}$ y using $I_3 \equiv -6x + 3y + 1 \leq 0$
$\langle \overbrace{[\![x \geq_{I_1} 0,\ x \leq 0,\ y \leq_J -1]\!]}^{M}, C\rangle$, where $J = \mathsf{tight}(I_3, y, [\![x \geq_{I_1} 0, x \leq 0]\!])$
    $\langle [\![x \geq_{I_1} 0,\ x \leq 0]\!], 3y \oplus -6x + 1\rangle$
    $\Longrightarrow$ $\mathsf{Consume}$
    $\langle [\![x \geq_{I_1} 0,\ x \leq 0]\!], 3y - 6x \oplus 1\rangle$
    $\Longrightarrow$ Round
    $J \equiv y - 2x + 1 \leq 0$
$\Longrightarrow$ $\mathsf{Conflict}$ using $I_2 \equiv 6x - 3y - 2 \leq 0$, since $\mathsf{lower}(6x - 3y - 2, M) = 1 > 0$
$\langle [\![x \geq_{I_1} 0,\ x \leq 0,\ y \leq_J -1]\!], C\rangle \vdash 6x - 3y - 2 \leq 0$
$\Longrightarrow$ $\mathsf{Resolve}$ $\quad \mathsf{resolve}(y \leq_J -1, 6x - 3y - 2 \leq 0) = (3(-2x + 1) + 6x - 2 \leq 0)$
$\langle [\![M, x \leq 0]\!], C\rangle \vdash 1 \leq 0$
$\Longrightarrow$ Unsat
unsat

4.3 Finite Problems

We say a set of inequalities C is a *finite problem* if for every variable x in C, there are two integer constants a and b such that $\{x-a \leq 0, -x+b \leq 0\} \subseteq C$. We say a set of inequalities C is an *infinite problem* if it is not finite. That is, there is a variable x in C such that there are no values a and b such that $\{x - a \leq 0, -x + b \leq 0\} \subseteq C$. We say an inequality is *simple* if it is of the form $x - a \leq 0$ or $-x + b \leq 0$.

Let Propagate-Simple be a rule such as Propagate, but with an extra condition requiring J to be a simple inequality. We say a strategy for applying the rules is *reasonable* if a rule R different from Propagate-Simple is applied only if Propagate-Simple is not applicable. Informally, a *reasonable* strategy prevents the generation of derivations where simple inequalities are ignored and C is essentially treated as an infinite problem.

Theorem 2 (Termination) *Given a finite problem C, there is no infinite derivation sequence starting from $\langle [\![]\!], C\rangle$ that uses a reasonable strategy.*

Proof The proof of the statement is an adaptation of the proof used to show termination of the Abstract DPLL [21]. We say that a state $\langle M_i, C_i\rangle$ is *reachable* if there is a derivation sequence

$$\langle [\![]\!], C\rangle \Longrightarrow \cdots \Longrightarrow \langle M_i, C_i\rangle \ .$$

A sequence M is *bounded* if there is no variable x in C such that $\mathsf{lower}(x, M) = -\infty$ or $\mathsf{upper}(x, M) = \infty$. Given a derivation T starting at $\langle [\![]\!], C\rangle$, let $\langle M_0, C_0\rangle$ be the first state in T where Propagate-Simple is not applicable. Then, M_0 is bounded because C is a finite problem. We say $\langle M_0, C_0\rangle$ is the *actual* initial state of T.

The *level* of a state $\langle M_i, C_i\rangle$ is the number of decided bounds in M_i. The level of any reachable state $\langle M_i, C_i\rangle$ is $\leq n$, where n is the number of variables in C.

Let $\mathsf{subseq}_j(M)$ denote the maximal prefix subsequence of M of level $\le j$. Let V denote the set of variables used in C.

First, we define an auxiliary function $w(M)$ as

$$w(M) = \begin{cases} \infty & \text{if } M \text{ is unbounded,} \\ \sum_{x \in V} (\mathsf{upper}(x, M) - \mathsf{lower}(x, M)) & \text{otherwise.} \end{cases}$$

Now, we define a function weight that maps a sequence M into a $(n+1)$-tuple, where n is the number of variables in C. It is defined as

$$\mathsf{weight}(M) = \langle w(\mathsf{subseq}_0(M)), w(\mathsf{subseq}_1(M)), \ldots, w(\mathsf{subseq}_n(M)) \rangle \ .$$

Given two bounded sequences M and M', we say $M \lessdot M'$ if $\mathsf{weight}(M) <_{\mathsf{lex}} \mathsf{weight}(M')$, where $<_{\mathsf{lex}}$ is the lexicographical extension of the order $<$ on natural numbers.

For any transition $\langle M_i, C_i \rangle \Longrightarrow \langle M_{i+1}, C_{i+1} \rangle$ performed by Decide, Propagate or Propagate-Simple, as these rules only improve the variable bounds, if M_i is bounded, then M_{i+1} is also bounded and $M_{i+1} \lessdot M_i$.

Now let's consider the conflict resolution rules. The conflict resolution process starts from a state $\langle M, C \rangle \vdash I$ and then traverses over the elements of the trail backwards. Since the size of the sequence M is finite, conflict resolution is always a finite sequence of steps. For each conflict resolution step of the transition system

$$\langle M_k, C_k \rangle \vdash p \le 0 \Longrightarrow \langle M_{k+1}, C_{k+1} \rangle \vdash q \le 0 \ ,$$

the sequence M_{k+1} is a subsequence of M_k, and the rules keep as invariant the fact that $q \le 0$ is a valid deduction and $\mathsf{lower}(q, M) > 0$. For applications of the Resolve rule this follows from Lemma 1 and Lemma 2, and for applications of the Skip-Decision rule this follows from the preconditions of the rule itself.

Let's show that we can not get stuck in conflict analysis, i.e. that a transition using the conflict resolution rules is always possible. Assume that no rule other than possibly Backjump is applicable, i.e. that we are in a state $\langle M_k, C_k \rangle \vdash p \le 0$ where $M_k = [\![M'_k, \gamma]\!]$ such that

- γ is a decided bound (Resolve is not applicable); and
- $\mathsf{lower}(p, M'_k) \le 0$ (Skip-Decision is not applicable).

If $\gamma = x \le b$, then we know that $\mathsf{lower}(x, M'_k) = b$ and, additionally, that $p = -ax + q$ for some $a > 0$ as otherwise we would have that $\mathsf{lower}(p, M_k) = \mathsf{lower}(p, M'_k) > 0$. We can now compute

$$0 < \mathsf{lower}(-ax + q, M_k) = -a\mathsf{upper}(x, M_k) + \mathsf{lower}(q, M_k) = -ab + \mathsf{lower}(q, M'_k) \ ,$$

and therefore $\mathsf{lower}(q, M'_k) > ab$. From here we see that $-ax + q \le 0$ implies a lower bound $\mathsf{bound}(p \le 0, x, M'_k) > b$ on x, improving on the current $\mathsf{lower}(x, M'_k) = b$. For the Backjump rule to be applicable we must also show that the new bound does not exceed any existing upper bounds in M'_k. Assume the opposite, i.e. that

$$\mathsf{upper}(x, M'_k) < \mathsf{bound}(p \le 0, x, M'_k) = \left\lceil \frac{\mathsf{lower}(q, M'_k)}{a} \right\rceil \ .$$

But then we can conclude that

$$\begin{aligned}\mathsf{lower}(-ax+q, M'_k) &= -a\mathsf{upper}(x, M'_k) + \mathsf{lower}(q, M'_k)\\ &> -a\mathsf{upper}(x, M'_k) + a\mathsf{upper}(x, M'_k) = 0 \ .\end{aligned}$$

This contradicts our assumption and therefore the new bound does not exceed the existing bound and the Backjump rule is applicable. Similarly, if $\gamma = x \geq b$ we can conclude that the Backjump rule is applicable.

The only way to exit the conflict analysis state and get back into the search mode, is by an application of the Backjump rule. But, for any transition $\langle M_i, C_i\rangle \vdash p \leq 0 \implies \langle M_{i+1}, C_{i+1}\rangle$ performed by the Backjump rule, since this transition can backtrack at most up to the first decision and will therefore not eliminate the bounded initial state M_0, if $\mathsf{subseq}_0(M_i)$ is bounded, then M_{i+1} is also bounded and $M_{i+1} \lll M_i$. Tough Backjump may eliminate several bounds from M_i, it improves the bound of a variable in some lower level. Since, for finite problems, the *actual* initial state $\langle M_0, C_0\rangle$ is bounded and $\lll$ is well-founded for bounded states, we have that any derivation will eventually terminate.

Theorem 3 (Completeness) *Given a finite problem C, any derivation starting from the initial state $\langle[\![]\!], C\rangle$, that uses a reasonable strategy, terminates either in the $\langle v, \mathsf{sat}\rangle$ state, or in the* unsat *state. In the former case C is satisfiable, and in the latter case C is unsatisfiable.*

Proof If we terminate a derivation in the $\langle v, \mathsf{sat}\rangle$ state we know by the precondition of the Sat rule and Theorem 1 that C is satisfiable with v being a witness variable assignment. If we terminate a derivation in the state unsat, again by Theorem 1, we know that C is unsatisfiable. Since we know from the previous theorem that any reasonable derivation will terminate, lets show that these are the only two possible terminal states.

The conflict analysis rules either enter the unsat state, or eventually return to the search phase. Therefore, we need only consider the search phase states as possible other terminal states. Assume that the derivation terminates in a search state $\langle M, C'\rangle$. By Theorem 1 we know that M is a well-formed sequence. If there is a variable x from C' that is not fixed, since M is a well-formed sequence, it must be that $\mathsf{lower}(x, M) < \mathsf{upper}(x, M)$. Since we're using a reasonable strategy, then both of these bounds must be finite and therefore both Decide rules are enabled for a transition. Assume therefore that all variables x from C' are fixed in M. If so, then either the variable assignment $v[M]$ satisfies all the constraints from C' and we can transition into the sat state, or there is a constraint $p \leq 0 \in C'$ such that $v[M](p) = \mathsf{lower}(p, M) > 0$ and we can transition into a conflict analysis state. Since in both cases there is a transition available, no search state can be a terminal state.

4.4 Infinite Problems.

As mentioned in the introduction, for infinite problems a termination argument can be constructed using the fact that for any set of inequalities C, there is an equisatisfiable C' where every variable in C' is bounded, but it has little practical value. In Section 5, we describe an extra set of rules that guarantee termination even for infinite problems. Here we discuss some possible heuristics remedies that are sufficient to solve many infinite problems encountered in practice.

Slack Introduction. One of the obvious drawbacks of the presented system is that it is easy to find a system of constraints where the transition system can not make any progress. For example, if the original constraints don't contain any explicit variable bounds, then the Propagate rule is not applicable as no inequality can infer new bounds, and the Decide rule is not applicable since it requires a variable that is already bounded from one side. Therefore for such sets of constraints, no search rule is applicable and the system can not make progress. This issue can be resolved by introducing fresh variables that create artificial bounds that can start-up the computation.

Given a state $S = \langle M, C \rangle$, we say that a variable x is *unbounded* at S if there is no bound on x is M, i.e. when $\mathsf{lower}(x, M) = -\infty$ and $\mathsf{upper}(x, M) = \infty$. We also say that x is *stuck* at state S if it is unbounded and the Propagate rule cannot be used to deduce a lower or upper bound for x. A state S is *stuck* if all undecided variables in S are stuck, and no inequality in C is false in M. That is, there is no possible transition for a stuck state S.

We avoid stuck states, by observing that for every finite set of inequalities C, there is an equisatisfiable set C' such that for every variable x in C', $(-x \leq 0) \in C'$. The idea is to replace every occurrence of x in C with $(x^+ - x^-)$, and add the inequalities $-x^+ \leq 0$ and $-x^- \leq 0$. Using this transformation we can avoid the problem of stuck states since having a lower bound for each variable implies that Decide rule is always applicable. Instead of using this eager preprocessing step, we use a lazy approach, where *slack variables* are dynamically introduced. When in a stuck state $\langle M, C \rangle$, we simply select an unbounded variable x, add a fresh *slack* variable $x_s \geq 0$, and add new inequalities to C that "bound" x in the interval $[-x_s, x_s]$. This idea is captured by the following rule:

Slack-Intro

$$\langle M, C \rangle \Longrightarrow \langle M, C \cup \{x - x_s \leq 0, -x - x_s \leq 0, -x_s \leq 0\} \rangle \ \textbf{if} \begin{cases} \langle M, C \rangle \text{ is stuck} \\ x_s \text{ is fresh} \end{cases}$$

Note that it is sound to reuse a slack variable x_s used for "bounding" x, to bound some other variable y, and this is what we do in our implementation.

4.5 Relevant propagations

Although we can avoid the stuck states using the slack variables as described above, this still does not guarantee termination for infinite problems. Unlike in SAT and Pseudo-Boolean solvers, the Propagate rules cannot be applied to exhaustion for infinite problems, since the Propagate rules may remain applicable indefinitely.

Example 6 Consider the following set of constraints

$$C = \{\overbrace{-x \leq 0}^{I_x}, \overbrace{-y \leq 0}^{I_y}, \overbrace{-z \leq 0}^{I_z}, \overbrace{-x + y + 1 \leq 0}^{I}, \overbrace{x - y - z \leq 0}^{J}\} \ .$$

The constraints are satisfiable by the assignment $x = 1$, $y = 0$, $z = 1$. Starting from the initial state $\langle [\![]\!], C\rangle$ we can first obtain lower bounds for the variables and then decide the value of the variable z as follows

$$\langle [\![]\!], C\rangle \Longrightarrow^* \langle \underbrace{[\![x \geq_{I_x} 0, y \geq_{I_y} 0, z \geq_{I_x} 0, z \leq 0]\!]}_{M}, C\rangle \ .$$

In this branch of the search, where z is fixed to the value 0, the constraints are unsatisfiable. We can now generate the following infinite sequence of states by only applying the $\mathsf{Propagate}$ rule.

$$\begin{aligned}\langle M, C\rangle &\Longrightarrow \langle [\![M, x \geq_I 1]\!], C\rangle \\ &\Longrightarrow \langle [\![M, x \geq_I 1, y \geq_J 1]\!], C\rangle \Longrightarrow \\ &\langle [\![M, x \geq_I 1, y \geq_J 1, x \geq_I 2]\!], C\rangle \Longrightarrow \ldots\end{aligned}$$

To try and avoid the infinite loops we adopt a simple heuristic that limits the applications of the propagate rule. Using this heuristic we cut the possible propagation loops. Let $\mathsf{nb}(x, M)$ denote the number of lower and upper bounds for a variable x in the sequence M. Given a state $S = \langle M, C\rangle$, some $\delta > 0$, and a bound on a number of propagations Max, we say a new lower bound $x \geq_I b$ is *δ-relevant* at S if

1. $\mathsf{upper}(x, M) \neq +\infty$, or
2. $\mathsf{lower}(x, M) = -\infty$, or
3. $\mathsf{lower}(x, M) + \delta|\mathsf{lower}(x, M)| < b$ and $\mathsf{nb}(x, M) < \mathsf{Max}$.

The intuition for the above definition of relevancy is as follows. If x has a upper bound, then any lower bound is δ-relevant because x becomes bounded, and termination is not an issue for bounded variables. If x does not already have a lower bound, then any new lower bound $x \geq_I b$ is relevant. Finally, the third case states that the magnitude of the improvement must be significant and the number of bound improvements for x in M must be smaller than Max. In theory, to prevent non-termination during bound propagation we only need the cutoff Max. The condition $\mathsf{lower}(x, M) + \delta|\mathsf{lower}(x, M)| < b$ is pragmatic, and is inspired by an approach used in [1]. The idea is to block any bound improvement for x that is *insignificant* with respect to the already known bound for x.

Even when only δ-relevant propagations are performed, it is still possible to generate an infinite sequence of transitions. The key observation is that $\mathsf{Backjump}$ is essentially a propagation rule, that is, it backtracks M, but it also adds a new improved bound for some variable x. It is easy to construct non-terminating examples, where $\mathsf{Backjump}$ is used to generate an infinite sequence of non δ-relevant bounds.

5 Strong Conflict Resolution

In this section, we extend our procedure to be able to handle divisibility constraints, by adding propagation, solving and consistency checking rules specific to divisibility constraints into our system. Then we show how to ensure that our procedure terminates even in cases when some variables are unbounded. A *linear divisibility constraints* is of the form

$$d \mid a_1x_1 + \cdots + a_nx_n + c \ ,$$

where d is a non-zero integer constant. We denote divisibility constraints with the (possibly subscripted) letter D. A variables assignment υ satisfies the divisibility constraint D, if υ assigns the variables $x_1, \ldots, x_n$ and $d \mid a_1\,\upsilon(x_1) + \cdots + a_n\,\upsilon(x_n) + c$. Throughout this section we allow divisibility constraints as part of the input problem.

Solving divisibility constraints. We will add one proof rule to the proof system, in order to help us keep the divisibility constraints in a normal form. As Cooper originally noticed in [8], given two divisibility constraints, we can always eliminate a variable from one of them, obtaining equivalent constraints.

$$\textsc{div-solve}\ \frac{d_1 \mid a_1x + p_1, d_2 \mid a_2x + p_2}{\begin{array}{c} d_1d_2 \mid dx + \alpha(d_2p_1) + \beta(d_1p_2) \\ d \mid a_2p_1 - a_1p_2 \end{array}}\ \textbf{if}\ \boxed{\begin{array}{c} d = \mathsf{gcd}(a_1d_2, a_2d_1) \\ \alpha(a_1d_2) + \beta(a_2d_1) = d \end{array}}$$

Since we could not find the proof of correctness of the above rule in the literature, we provide the following simple one.

Lemma 3 *Consider the following two divisibility constraints*

$$d_1 \mid a_1x + p_1 \ , \tag{7}$$
$$d_2 \mid a_2x + p_2 \ . \tag{8}$$

These are equivalent to the divisibility constraints

$$d_1d_2 \mid dx + \alpha(d_2p_1) + \beta(d_1p_2) \ , \tag{9}$$
$$d \mid a_2p_1 - a_1p_2 \ , \tag{10}$$

where $d = \mathsf{gcd}(a_1d_2, a_2d_1)$ *and* $\alpha(a_1d_2) + \beta(a_2d_1) = d$.

Proof ($\Rightarrow$) Assume (7) and (8). Multiplying them with d_2 and d_1, receptively, we also have that $d_1d_2 \mid d_2a_1x + d_2p_1$ and $d_1d_2 \mid d_1a_2x + d_1p_2$. We can add these two together, multiplied with α and β to obtain (9).

On the other hand, multiplying (7) and (8) with a_2 and a_1, respectively, we get that $d_1a_2 \mid a_1a_2x + a_2p_1$ and $d_2a_1 \mid a_1a_2x + a_1p_2$. Now, since $d = \mathsf{gcd}(a_1d_2, a_2d_1)$ we also know that $d \mid a_1a_2x + a_2p_1$ and $d \mid a_1a_2x + a_1p_2$. Subtracting these two we get (10).

($\Leftarrow$) Assume (9) and (10). Using the assumption that $d = \alpha(a_1d_2) + \beta(a_2d_1)$, we can be rewrite them as

$$d_1d_2 \mid \alpha d_2(a_1x + p_1) + \beta d_1(a_2x + p_2) \ , \tag{11}$$
$$d \mid a_2(a_1x + p_1) - a_1(a_2x + p_2) \ . \tag{12}$$

Using the first direction applied to above (taking $a_1x + p_1$ as the variable) we get that

$$dd_2 \mid \mathsf{gcd}(d_1d_2a_2, \alpha dd_2) \mid a_2\beta d_1(a_2x + p_2) + \alpha d_2a_1(a_2x + p_2) \ ,$$
$$dd_2 \mid d(a_2x + p_2) \ ,$$

form which we get (8). Similarly, assuming $a_2x + p_2$ is the variable, we get (7).

We use the above proof rule to enable normalization of divisibility constraints into a triangular form, and basic consistency checking, by adding transition rules Solve-Div and Unsat-Div to our transition system.

Solve-Div

$$\langle M, C\rangle \quad \Longrightarrow \langle M, C'\rangle \qquad \textbf{if} \begin{cases} D_1, D_2 \in C, \\ (D_1', D_2') = \textsc{div-solve}(D_1, D_2), \\ C' = C \setminus \{D_1, D_2\} \cup \{D_1', D_2'\}. \end{cases}$$

Unsat-Div

$$\langle M, C \cup \{(d \mid a_1x_1 + \cdots + a_nx_n + c)\}\rangle \quad \Longrightarrow \quad \mathsf{unsat} \quad \textbf{if } gcd(d, a_1, \ldots, a_n) \nmid c$$

Propagation. With divisibility constraints as part of our problem, we can now achieve even more powerful bound propagation. We allow propagation on divisibility constraint D if all but one variable in D are fixed.

Let $\langle M, C\rangle$ be a well-formed state, let $D \equiv (d \mid ax + p) \in C$ be a divisibility constraint with $a > 0$, $d > 0$. Assume that the variable x has a lower bound $\mathsf{lower}(x, M) = b$, with $x \geq_I b \in M$ and $I \equiv (-x + q)$. Assume, additionally, that p is fixed, i.e. assume that $\mathsf{val}(p, M) = k$. If the bound b does not satisfy the divisibility constraint, i.e. if $d \nmid ab + k = \mathsf{lower}(ax + p, M)$, we can deduce a better lower bound on x. This bound is obtained by skipping over the integer values that do not satisfy the divisibility constraint. Let c be the first such value, i.e. the smallest $c > b$ with $d \mid ac + k$.

Since $\langle M, C\rangle$ is a well formed state we know that $b \leq \mathsf{lower}(q, M')$ for some prefix M' of M, and therefore $b \leq \mathsf{lower}(q, M)$. Now, in order to satisfy the divisibility constraint we must have an integer z such that $dz = ax + p$, and therefore $I_1 \equiv -dz + ax + p \leq 0$. Note that b, the lower bound of x, does not satisfy the divisibility constraint, and therefore this inequality implies a bound on z that requires rounding. Since p is fixed, and we have a lower bound on x, we can now use our system for deriving tight inequalities to deduce a tightly propagating inequality $I_2 \equiv -z + r \leq 0$ that, in the state, bounds z from below. Moreover, by using a strategy that never uses the Consume rule on the variable x, we can ensure that r does not include x. From Lemma 2 we can now conclude that

$$\mathsf{lower}(r, M) \geq \left\lceil \frac{\mathsf{lower}(ax + p, M)}{d} \right\rceil = \left\lceil \frac{ab + k}{d} \right\rceil = \frac{ac + k}{d} \in \mathbb{Z} \ ,$$

with the last inference resulting by choice of c. Now, we use the new inequality I_2 to derive an inequality I_3 that provides a new bound on x.

$$\textsc{Combine}\ \frac{\dfrac{\overbrace{-z + r \leq 0}^{I_2}}{-dz + dr \leq 0} \qquad \dfrac{\overbrace{dz = ax + p}^{D}}{dz - ax - p \leq 0}}{\underbrace{-ax + dr - p \leq 0}_{I_3}}$$

Since we know that r and p don't include x (and therefore x did not get eliminated) we can compute the bound that this inequality infers on x in the current model

$$\mathsf{bound}(I_3, x, M) = \left\lceil \frac{\mathsf{lower}(dr - p, M)}{a} \right\rceil \geq \left\lceil \frac{d\mathsf{lower}(r, M) - k)}{a} \right\rceil \geq \left\lceil \frac{d\frac{ac+k}{d} - k}{a} \right\rceil = c \ .$$

We can also use our procedure to convert this new constraint into a tightly propagating inequality J. Similar reasoning can be applied for the upper bound inequalities. We denote, as a shorthand, the result of this whole derivation with $J = \mathsf{div\text{-}derive}(D, x, M)$ and the constant c with $\mathsf{bound}(D, x, M)$.

We can now use the derivation above to empower propagation and inconsistency detection driven by divisibility constraints, as summarized below.

Propagate-Div

$$\langle M, C\rangle \implies \langle [\![M, x \geq_I c]\!], C\rangle \quad \textbf{if} \begin{cases} D \equiv (d \mid ax + p) \in C,\ \mathsf{val}(p, M) = k, \\ b = \mathsf{lower}(x, M),\ d \nmid ab + k, \\ c = \mathsf{bound}(D, x, M),\ c \leq \mathsf{upper}(x, M) \\ I = \mathsf{div\text{-}derive}(D, x, M) \end{cases}$$

$$\langle M, C\rangle \implies \langle [\![M, x \leq_I c]\!], C\rangle \quad \textbf{if} \begin{cases} D \equiv (d \mid ax + p) \in C,\ \mathsf{val}(p, M) = k, \\ b = \mathsf{upper}(x, M),\ d \nmid ab + k, \\ c = \mathsf{bound}(D, x, M),\ c \geq \mathsf{lower}(x, M) \\ I = \mathsf{div\text{-}derive}(J, D, x, M) \end{cases}$$

Conflict-Div

$$\langle M, C\rangle \implies \langle M, C\rangle \vdash I \quad \textbf{if} \begin{cases} D \equiv (d \mid ax + p) \in C,\ \mathsf{val}(p, M) = k, \\ b = \mathsf{lower}(x, M),\ d \nmid ab + k, \\ \mathsf{bound}(D, x, M) > \mathsf{upper}(x, M) \\ I = \mathsf{div\text{-}derive}(D, x, M) \end{cases}$$

$$\langle M, C\rangle \implies \langle M, C\rangle \vdash I \quad \textbf{if} \begin{cases} D \equiv (d \mid ax + p) \in C,\ \mathsf{val}(p, M) = k \\ b = \mathsf{upper}(x, M),\ d \nmid ab + k, \\ \mathsf{bound}(D, x, M) < \mathsf{lower}(x, M) \\ I = \mathsf{div\text{-}derive}(J, D, x, M) \end{cases}$$

Note that, as in the case of propagation with inequalities, we do not need to derive the explanation inequality eagerly, but instead only record the new bound and do the derivation on demand, if needed for conflict analysis.

5.1 Eliminating Conflicting Cores.

As we have seen in the previous section, for sets of inequality constraints containing unbounded variables, there is no guarantee that the procedure described in the previous section will terminate. In this section, we describe an extension of the transition system, based on Cooper's quantifier elimination procedure, that guarantees termination and can additionally handle divisibility constraints.

Let U be a subset of the variables in X. We will select the set U to contain the unbounded variables from from the initial set of constraint C, and refer to U as the set of *unbounded* variables. Let $\prec$ be a total order over the variables in X such that for all variables $x \in X \setminus U$ and $y \in U$, $x \prec y$. We say a variable x is *maximal* in a constraint C containing x if $y \prec x$ for all variables y in C different from x. For now, we assume that $\prec$ is fixed, but we describe later how to change dynamically U and $\prec$ without compromising termination.

Let $S = \langle M, C \rangle$ be a well-formed state and consider two inequalities from C and the bounds that they imply

$$I_1 \equiv bx - q \leq 0 \ , \qquad b_1 = \mathsf{bound}(I_1, x, M) \ ,$$
$$I_2 \equiv -ax + p \leq 0 \ , \qquad b_2 = \mathsf{bound}(I_2, x, M) \ .$$

If the polynomials p and q are fixed at S and the implied bounds are in conflict, i.e. $b_1 > b_2$, we call the set $\{I_1, I_2\}$ an *interval conflicting core*. If the bounds are not in conflict but there is a divisibility constraint $D \equiv (d \mid cx + s) \in C$, with s fixed, such that for all values $k \in [b_1, b_2]$, the divisibility constraint does not hold i.e. $d \nmid ck + \mathsf{val}(s, M)$, we call the set $\{I_1, I_2, D\}$ a *divisibility conflicting core*. We do not consider cores containing more than one divisibility constraint because we can always use the Solve-Div rule to eliminate all but one of them. From hereafter, we assume a core is always of the form $\{I_1, I_2, D\}$, since we can include the redundant divisibility constraint $(1 \mid x)$ in any interval conflicting core.

We say x is a *conflicting variable* at state S if there is an interval or divisibility conflicting core for x. The variable x is the *minimal conflicting variable* at S if there is no $y \prec x$ such that y is also a conflicting variable at S. Let x be a minimal conflicting variable at state $S = \langle M, C \rangle$ and

$$D = \{-ax + p \leq 0, \ bx - q \leq 0, \ (d \mid cx + r)\}$$

be a conflicting core for x, We call a *strong resolvent* for D a set R of inequality and divisibility constraints equivalent to

$$\exists x. - ax + p \leq 0 \land bx - q \leq 0 \land (d \mid cx + r) \ .$$

The key property of the strong resolvent R is that in any state $\langle M', C' \rangle$ with $R \subset C'$, x is not the minimal conflicting variable or D is not a conflicting core.

We compute the resolvent R using Cooper's left quantifier elimination procedure. It can be summarized by the rule

$$\textsc{Cooper-Left} \ \frac{(d \mid cx + s), \ \ -ax + p \leq 0, \ \ bx - q \leq 0}{\begin{array}{c} 0 \leq k \leq m, \ \ bp - aq + bk \leq 0, \\ a \mid k + p, \ \ ad \mid ck + cp + as \end{array}} \ \textbf{if} \ \boxed{a, b, c > 0}$$

where k is a fresh variable and $m = \mathsf{lcm}(a, \frac{ad}{\mathsf{gcd}(ad, c)}) - 1$. The fresh variable k is bounded so it does not need to be included in U. We extend the total order $\prec$ to k by making k the minimal variable. For the special case, where $(d \mid cx + s)$ is $(1 \mid x)$, we get that $m = a - 1$ and the rule above simplifies to

$$\frac{-ax + p \leq 0, \ \ bx - q \leq 0}{0 \leq k < a, \ bp - aq + bk \leq 0, \ a \mid p + k}$$

Lemma 4 *The* Cooper-Left *rule is sound and produces a strong resolvent.*

Proof Multiplying the premises with appropriate coefficients we can obtain new, equivalent constraints that have abc as coefficient with x

$$(ab)d \mid (abc)x + (ab)s \ , \tag{13}$$

$$(bc)p \leq (abc)x \ , \quad (abc)x \leq (ac)q \ . \tag{14}$$

In order for an integer solution to the inequalities above to exist, from the left inequality we can conclude that there must exist a $k \geq 0$ such that $(abc)x = (bc)p + (bc)k$, and therefore

$$a \mid p + k \ .$$

Additionally, there must be enough room for this solution so, it must be that $(ac)q - (bc)p \geq (bc)k$, i.e

$$bp - aq + bk \leq 0 \ .$$

Now, substituting $(abc)x$ into the divisibility constraint we get that $(ab)d \mid (bc)k + (bc)p + (ab)s$, or equivalently that

$$ad \mid ck + cp + as \ .$$

In order to bound k from above, we note that a sufficient (and necessary) condition for a divisibility constraint $a \mid bx + c$ to have a solution, is to have a solution with $0 \leq x < \frac{a}{\mathsf{gcd}(a,b)}$. We use this and deduce that in our case, since we have two divisibility constraints, it must be that

$$0 \leq k < \mathsf{lcm}\left(a, \frac{ad}{\mathsf{gcd}(ad, c)}\right) \ .$$

The rule Cooper-Left is biased to lower bounds. We may also define the Cooper-Right rule that is based on Cooper's right quantifier elimination procedure and is biased to upper bounds. We use $\mathsf{cooper}(D)$ to denote a procedure that computes the strong resolvent R for a conflicting core D. Now, we extend our procedure with a new rule for introducing resolvents for minimal conflicting variables.

Resolve-Cooper

$$\langle M, C\rangle \implies \langle M, C \cup \mathsf{cooper}(D)\rangle \quad \textbf{if} \begin{cases} x \in U, \\ x \text{ is the minimal conflicting variable}, \\ D \text{ is a conflicting core for } x. \end{cases}$$

Note that in addition to fresh variables, the Resolve-Cooper rule also introduces new constraints without resorting to the Learn rule. We will show that this cannot happen indefinitely, as the rule can only be applied a finite number of times.

Lemma 5 *For any initial state $\langle [\![]\!], C\rangle$, the* Resolve-Cooper *rule can be applied only a finite number of times, if*

- *it is never applied to cores containing inequalities introduced by the* Learn *rule, and;*
- *the* Forget *rule is never used to eliminate resolvents introduced by* Resolve-Cooper*.*

Proof First notice that, although the Cooper-Left and Cooper-Right rules introduce fresh variables k, these variables are initially bounded, and are therefore never included in the set U. Consequently, these variables are never considered by Resolve-Cooper and, therefore Resolve-Cooper will only apply to the variables from the initial set of constraints C.

Now, consider a conflicting core

$$D = \{-ax + p \leq 0,\ bx - q \leq 0,\ (d \mid cx + r)\} \enspace ,$$

and a derivation sequence T satisfying the conditions above. In such a derivation sequence, the Resolve-Cooper rule can only be applied once. This is true because the resolvent $R = \mathsf{cooper}(D)$ is equivalent to $\exists x.D$. Although the resolvent introduces a fresh variable, it is a finite one and therefore smaller than all the variables in U. Therefore, for any state where we could try and apply the strong resolution again, i.e. $\langle M', C'\rangle$ such that $R \subseteq C'$, x is not the minimal conflicting variable or D is not a conflicting core. The rule Resolve-Cooper will therefore not be applicable to the same core, at any state that already contains the resolvent R. Additionally, since we do not eliminate resolvents introduced by Resolve-Cooper using the Forget rule, a resolvent for a core D will be generated at most once.

Now, let U be the set of unbounded variables $\{y_1, \ldots, y_m\}$, such that $y_m \prec \ldots \prec y_1$. Since Resolve-Cooper considers these variables in an ordered fashion, all possible resolvents can be defined by saturation, using the following sequence

$$S_0 = C \qquad S_{i+1} = S_i \cup \{R \mid R \text{ is a resolvent for a core } D \subseteq S_i \text{ for variable } y_{i+1}\}$$

The final set of all possible resolvents will be $\mathsf{saturated}(C) = S_{m+1}$. Since Resolve-Cooper can be applied at most once for a core D, and there are a finite number of cores D in each S_i, it follows that the Resolve-Cooper rule can be applied only a finite number of times.

Now we are ready to present and prove a simple and flexible strategy that will guarantee termination of our procedure even in the unbounded case.

Definition 3 (Two-layered strategy) We say a strategy is two-layered for an initial state $\langle [\![]\!], C_0\rangle$ if

1. it is reasonable (i.e., gives preference to the Propagate-Simple rules);
2. the Propagate and Propagate-Div rules are limited to δ-relevant bound refinements;
3. the Forget rule is never used to eliminate resolvents introduced by Resolvent-Cooper;
4. it only applies the Conflict and Conflict-Div rules if Resolve-Cooper is not applicable.

Theorem 4 (Termination) *Given a set of constraints C, there is no infinite derivation sequence starting from $S_0 = \langle [\![]\!], C\rangle$ that uses a two-layered strategy when U contains all unbounded variables in C.*

Proof First we note that, if the Conflict or the Conflict-Div rule applies to a non-U-constraint, it must be that Resolve-Cooper is not applicable. Since the strategy prefers Resolve-Cooper this, in effect, splits the procedure into two layers, one dealing with bounded variables, and the other one dealing with the unbounded variables using the strong resolution. And, since the Learn rule will therefore only be able to learn constraint over bounded variables, we will never apply Resolve-Cooper to cores involving those constraints.

The strategy also dictates that we don't remove the strong resolvents introduced by Resolve-Cooper so we know, by Lemma 5, that in any derivation sequence

$$T = \langle [\![]\!], C_0 \rangle \Longrightarrow \langle M_1, C_1 \rangle \Longrightarrow \cdots \Longrightarrow \langle M_n, C_n \rangle \Longrightarrow \cdots$$

produced by a two-layered strategy, the Resolve-Cooper rule can only be applied a finite number of times. Consequently, the number of fresh variables introduced in T is bounded.

Then, there must be a state $S_n = \langle M_n, C_n \rangle$ in T such that the Resolve-Cooper rule is not applicable to any state that is reachable from S_n. Therefore no additional fresh variable is created after S_n.

Now, assume that the derivation sequence T is infinite. Then, since the propagation step is limited to the δ-relevant ones, it must be that, after S_n, the Conflict rule is being applied infinitely often. Moreover, since Resolve-Cooper does not apply after the state S_n, it must be that the Conflict rule is applied to only non-U-constraints. But we know, by Theorem 2, that if all variables are bounded, this can not happen.

As an improvement, we note that we do not need to fix the ordering $\prec$ at the beginning. It *can* be modified but, in this case, termination is only guaranteed if we eventually stop modifying it. Moreover, we can start applying the strategy with $U = \emptyset$. Then, for any non-δ-relevant bound refinement $\gamma(x)$, produced by the Backjump rules, we add x to the set U. Moreover, a variable x can be removed from U whenever a lower and upper bound for x can be deduced, and they do not depend on any decided bounds (variable becomes bounded).

6 Experimental Evaluation

We implemented the procedure described in a new solver cutsat. The implementation uses only the basic rules, with addition of slack introduction, and does not include strong conflict resolution. The strategy of applying the rules resembles those found in SAT solvers, and includes heuristics from the SAT community such as dynamic ordering based on conflict activity and Luby restarts. When a variable is to be decided, and we have an option to choose between the upper and lower bound, we choose the value that could satisfy most constraints. We propagate bounds on a variable x only when the variable x is to be decided next, and the propagation only includes inequalities where all variables but x are already assigned. The solver source code, binaries used in the experiments, and all the accompanying materials are available at the authors website[3].

In order to evaluate our procedure we took a variety of already available integer problems from the literature, but we also crafted some additional ones. We include the problems used in [14] to evaluate their new simplex-based procedure

[3] http://cs.nyu.edu/~dejan/cutsat/

Table 1 Experimental results.

problems	miplib2003 (16)		pb2010 (81)		dillig (250)		slacks (250)		pigeons (19)		primes (37)	
cutsat	722.78	12	1322.61	46	4012.65	223	2722.19	152	0.15	19	5.08	37
smt solvers	time(s)	solved	time(s)	solved	time(s)	solved	time(s)	solved	time(s)	solved	time(s)	solved
mathsat5+cfp	575.20	11	2295.60	33	**2357.18**	**250**	160.67	98	0.23	19	1.26	37
mathsat5	**89.49**	**11**	1224.91	38	3053.19	245	**3243.77**	**177**	0.30	19	**1.03**	**37**
yices	226.23	8	57.12	37	5707.46	159	7125.60	134	**0.07**	**19**	0.64	32
z3	532.09	9	**168.04**	**38**	885.66	171	589.30	115	0.27	19	11.19	23
pb solvers												
sat4j	**22.34**	**10**	**798.38**	**67**	0.00	0	0.00	0	110.81	8	0.00	0
sat4j+cp	28.56	10	349.15	60	0.00	0	0.00	0	**4.85**	**19**	0.00	0
mip solvers												
glpk	242.67	12	1866.52	46	4.50	248	0.08	10	**0.09**	**19**	**0.44**	**37**
cplex	53.86	15	1512.36	58	8.65	250	8.76	248	0.51	19	3.47	37
gurobi	**28.96**	**15**	**1332.53**	**58**	**5.48**	**250**	**8.12**	**248**	0.21	19	0.80	37

that incorporates a new way of generating cuts to eliminate rational solutions. These problems are generated randomly, with all variables unbounded. This set of problems, which we denote with dillig, was reported hard for modern SMT solvers. We also include a reformulation of these problems, so that all the variables are bounded, by introducing slack variables, which we denote as slack. Next, we include the pure integer problems from the MIPLIB 2003 library [2], and we denote this problem set as miplib2003. The original problems are all very hard optimization instances, but, since we are dealing with the decision problem only, we have removed the optimization constraints and turned them into feasibility problems.[4] We include PB problems from the 2010 pseudo-Boolean competition that were submitted and selected in 2010, marked as pb2010, and problems encoding the pigeonhole principle using cardinality constraints, denoted as pigeons. The pigeonhole problems are known to have no polynomial Boolean resolution proofs, and will therefore be hard for any resolution solver that does not use cutting planes. And finally, we include a group of crafted benchmarks encoding a tight n-dimensional cone around the point whose coordinates are the first n prime numbers, denoted as primes. In these benchmarks all the variables are bounded from below by 0. We include the satisfiable versions, and the unsatisfiable versions which exclude points smaller than the prime solution.

In order to compare to the state-of-the art we compare to three different types of solvers. We compare to the top SMT solvers that support integer reasoning, i.e yices 1.0.29 [15], z3 2.15 [12], mathsat5 [17] and mathsat5+cfp that simulates the algorithm from [14]. On all 0-1 problems in our benchmark suite, we also compare to the sat4j [5] PB solver, one of the top solvers from the PB competition, and a version sat4j+cp that is based on cutting planes. And, as last, we compare with the two top commercial MIP solvers, namely, gurobi 4.0.1 and cplex 12.2, and the open source MIP solver glpk 4.38. The MIP solvers have largely been ignored in the theorem-proving community, as it is claimed that, due to the use of floating point arithmetic, they are not sound.

[4] All of the problems have a significant Boolean part, and 13 (out of 16) problems are pure PB problems

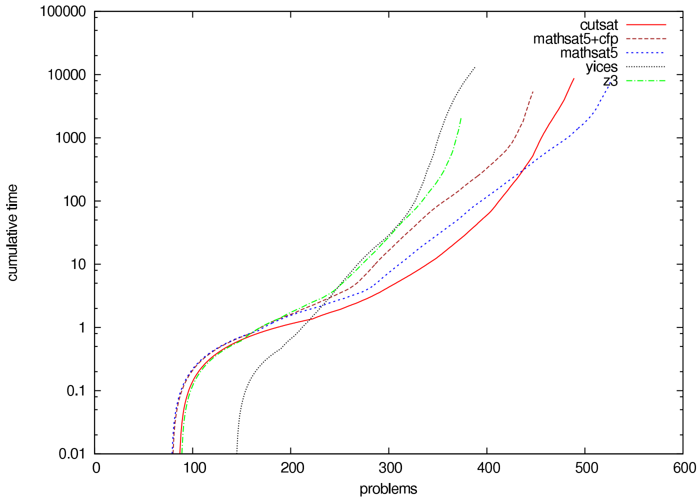

Fig. 2 Comparison of cutsat with other SMT solvers. The plot presents the number of problems solved against the cumulative time (logarithmic time scale).

All tests were conducted on an Intel Pentium E2220 2.4 GHz processor, with individual runs limited to 2GB of memory and 600 seconds. The results of our experimental evaluation are presented in Table 1. The rows are associated with the individual solvers, and columns separate the problem sets. For each problem set we write the number of problems that the solver managed to solve within 600 seconds, and the cumulative time for the solved problems. We mark with bold the results that are best in a group of solvers, and we underline the results that are best among all solvers. For the better understanding of the comparison of cutsat with individual SMT solvers we present cumulative solving times in Figure 2.

Compared to the SMT solvers, cutsat performs surprisingly strong, particularly being a prototype implementation. On all problem sets it outperforms, or is the same as, all smt solvers except mathsat5. Most importantly, it outperforms even mathsat5 on the real-world miplib2003 and pb2010 problem sets. The random dillig problems seem to be attainable by the solvers that implement the procedure from [14], but the same solvers surprisingly fail to solve the same problems with the slack reformulation (slacks). The commercial MIP solvers outperform all the SMT solvers and cutsat by a big margin.

7 Conclusion

We proposed a new approach for solving ILP problems. It has all key ingredients that made CDCL-based SAT solvers successful. Our solver justifies propagation steps using tightly-propagating inequalities that guarantee that any conflict detected by the search procedure can be resolved using only inequalities. We presented an approach to integrate Cooper's quantifier elimination algorithm in a model guided search procedure. Our first prototype is already producing encouraging results.

We see many possible improvements and extensions to our procedure. A solver for mixed integer-real problems is the first natural extension. One basic idea would be to make the real variables bigger than the integer variables in the variable order $\prec$, and use Fourier-Moztkin resolution (instead of Cooper's procedure) to explain conflicts on rational variables. We plan to integrate the the cutsat solver into an SMT solver using a proposed mcSAT calculus [13] that allows the integration of model-based procedures with the standard DPLL(T) framework. This would also allow the promising possibility of our solver to be complemented with a Simplex-based procedure. The idea is to use Simplex to check whether the current state or the search is feasible in the rational numbers.

Acknowledgements. We would like to thank Ken McMillan for reading an early draft and providing useful feedback, and Alberto Griggio for providing us with a custom version of mathsat5.

References

1. Tobias Achterberg. *SCIP: Solving constraint integer programs.* PhD thesis, TU Berlin, 2007.
2. Tobias Achterberg, Thorsten Koch, and Alexander Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):361–372, 2006.
3. Peter Barth. A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization. Research Report MPI-I-95-2-003, Saarbrücken, 1995.
4. Sergey Berezin, Vijay Ganesh, and David L. Dill. An online proof-producing decision procedure for mixed-integer linear arithmetic. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2619 of *LNCS*, pages 521–536. Springer, 2003.
5. Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2 system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.
6. Donald Chai and Andreas Kuehlmann. A fast pseudo-boolean constraint solver. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(3), 2005.
7. Vašek Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4(4):305–337, April 1973.
8. D.C. Cooper. Theorem proving in arithmetic without multiplication. *Machine Intelligence*, 7(91-99):300, 1972.
9. Scott Cotton. Natural domain SMT: A preliminary assessment. In *FORMATS*, 2010.
10. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):397, 1962.
11. Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960.
12. Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *TACAS 2008, Budapest, Hungary*, volume 4963 of *LNCS*, page 337. Springer, 2008.
13. Leonardo de Moura and Dejan Jovanović. A model-constructing satisfiability calculus. In *Verification, Model Checking, and Abstract Interpretation*, volume 7737 of *LNCS*, pages 1–12. Springer, 2013.
14. Isili Dillig, Thomas Dillig, and Alex Aiken. Cuts from proofs: A complete and practical technique for solving linear inequalities over integers. In *CAV*, 2009.
15. Bruno Dutertre and Leonardo de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *CAV*, LNCS, pages 81–94, 2006.
16. Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.
17. Alberto Griggio. A practical approach to SMT(LA(Z)). *SMT workshop*, 2010.
18. Konstantin Korovin, Nestan Tsiskaridze, and Andrei Voronkov. Conflict resolution. In *Principles and Practice of Constraint Programming*, 2009.
19. Kenneth L. McMillan, Andreas Kuehlmann, and Mooly Sagiv. Generalizing DPLL to richer logics. In *CAV*, 2009.

20. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient SAT solver. In *DAC*, 2001.
21. Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories: From an abstract DPLL procedure to DPLL(T). *J. ACM*, 53(6):937–977, 2006.
22. Christos H. Papadimitriou. On the complexity of integer programming. *Journal of the ACM*, 28(4):765–768, 1981.
23. William Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. In *ACM/IEEE conference on Supercomputing*, 1991.
24. Sanjit A. Seshia and Randal E. Bryant. Deciding quantifier-free Presburger formulas using parameterized solution bounds. In *Logic in Computer Science*, pages 100–109. IEEE, 2004.
25. João P. Marques Silva and Karem A. Sakallah. GRASP – a new search algorithm for satisfiability. In *ICCAD*, 1997.
26. Laurence A. Wolsey and George L. Nemhauser. *Integer and combinatorial optimization*. Wiley New York, 1999.