Linear Integer Arithmetic Revisited

Martin Bromberger(✉), Thomas Sturm, and Christoph Weidenbach

Max Planck Institute for Informatics, Saarbrücken, Germany
{mbromber,sturm,weidenb}@mpi-inf.mpg.de

Abstract. We consider feasibility of linear integer programs in the context of verification systems such as SMT solvers or theorem provers. Although satisfiability of linear integer programs is decidable, many state-of-the-art solvers neglect termination in favor of efficiency. It is challenging to design a solver that is both terminating and practically efficient. Recent work by Jovanović and de Moura constitutes an important step into this direction. Their algorithm CUTSAT is sound, but does not terminate, in general. In this paper we extend their CUTSAT algorithm by refined inference rules, a new type of conflicting core, and a dedicated rule application strategy. This leads to our algorithm CUTSAT++, which guarantees termination.

Keywords: Linear arithmetic · SMT · SAT · DPLL · Linear programming · Integer arithmetic

1 Introduction

Historically, feasibility of linear integer problems is a classical problem, which has been addressed and thoroughly investigated by at least two independent research lines: (i) integer and mixed real integer linear programming for optimization [9], (ii) first-order quantifier elimination and decision procedures for Presburger Arithmetic and corresponding complexity results [3,6,10–13]. We are interested in feasibility of linear integer problems, which we simply call *problems*, in the context of the combination of theories, as they occur, e.g., in the context of SMT solving or theorem proving. From this perspective, both these research lines address problems that are too general for our purposes: with the former, the optimization aspects go considerably beyond pure feasibility. The latter considers arbitrary Boolean combinations of constraints and quantifier alternation or even parametric problems.

Consequently, the SMT community has developed several interesting approaches on their own [1,4,7]. These solvers typically neglect termination and completeness in favor of efficiency. More precisely, these approaches are based on a branch-and-bound strategy, where the rational relaxation of an integer problem is used to cut off and branch on integer solutions. Together with the known a priori integer bounds [11] for a problem this yields a terminating and complete algorithm. However, these bounds are so large that for many practical problems the resulting branch-and-bound search space cannot be explored in reasonable

A.P. Felty and A. Middeldorp (Eds.): CADE-25, LNAI 9195, pp. 623–637, 2015.
DOI: 10.1007/978-3-319-21401-6_42

time. Hence, the a priori bounds are not integrated in the implementations of the approaches.

On these grounds, the recent work by Jovanović and de Moura [8], although itself not terminating, constitutes an important step towards an algorithm that is both efficient and terminating. The termination argument does no longer rely on bounds that are a priori exponentially large in the occurring parameters. Instead, it relies on structural properties of the problem, which are explored by their CUTSAT algorithm. The price for this result is an algorithm that is by far more complicated than the above-mentioned branch-and-bound approach. In particular, it has to consider divisibility constraints in addition to inequalities.

Our interest in an algorithm for integer constraints originates from a possible combination with superposition, e.g., see [5]. In the superposition context integer constraints are part of the first-order clauses. Variables in constraints are typically unguarded so that an efficient decision procedure for this case is a prerequisite for an efficient combined procedure.

Our contribution is an extension and refinement of the CUTSAT algorithm, which we call CUTSAT++. In contrast to CUTSAT, our CUTSAT++ generally terminates. The basic idea of both algorithms is to reduce a problem containing unguarded integer variables to a problem containing only guarded variables. These unguarded variables are not eliminated. Instead, one explores the unguarded variables by adding constraints on smaller variables to the problem, with respect to a strict total ordering where all unguarded variables are larger than all guarded variables. After adding sufficiently many constraints, feasibility of the problem depends only on guarded variables. Then a CDCL style algorithm tests for feasibility by employing exhaustive propagation. The most sophisticated part is to "turn" an unguarded variable into a guarded variable. Quantifier elimination techniques, such as Cooper elimination [3], do so by removing the unguarded variable. In case of Cooper elimination, the price to pay is an exponentially growing Boolean structure and exponentially growing coefficients. Since integer linear programming is NP-complete, all algorithms known today cannot prevent such a kind of behavior, in general. Since Cooper elimination does not care about the concrete structure of a given problem, the exponential behavior is almost guaranteed. The idea of both CUTSAT and CUTSAT++ is, therefore, to simulate a lazy variation of Cooper elimination. This leaves space for model assumptions and simplification rules in order for the algorithm to adapt to the specific structure of a problem and, hence, to systematically avoid certain cases of the worst-case exponential behavior observed with Cooper elimination.

The paper is organized as follows. After fixing some notation in Sect. 2, we present three examples for problems where CUTSAT diverges. The divergence of CUTSAT can be fixed by respective refinements on the original CUTSAT rules. However, in a fourth example the combination of the refinements results in a frozen state. Our conclusion is that CUTSAT lacks, in addition to the refinements, a third type of conflicting core, which we call *diophantine conflicting core*. Theorem 5, in Sect. 3, actually implies that any procedure that is based on what

we call *weak Cooper elimination* needs to consider this type of conflicting core for completeness. In Sects. 4–5 we refine the inference rules for the elimination of unguarded variables on the basis of weak Cooper elimination (Sect. 3) and show their soundness, completeness, and termination. Finally, we give conclusions and point at possible directions for future research. For detailed proofs of our Theorems and Lemmas see [2].

2 Motivation

We use *variables* x, y, z, k, possibly with indices. Furthermore, we use *integer constants* a, b, c, d, e, l, v, u, *linear polynomials* p, q, r, s, and *constraints* I, J, possibly with indices. As input *problems*, we consider finite sets of constraints C corresponding to and sometimes used as conjunction over their elements. Each constraint I is either an inequality $a_n x_n + \ldots + a_1 x_1 + c \leq 0$ or a divisibility constraint $d \mid a_n x_n + \ldots + a_1 x_1 + c$. We denote $\text{coeff}(I, x_i) = a_i \in \mathbb{Z}$. $\text{vars}(C)$ denotes the set of variables occurring in C. We sometimes write $C(x)$ in order to emphasise that $x \in \text{vars}(C)$. A problem C is satisfiable if $\exists X.C$ holds, where $X = \text{vars}(C)$. For true we denote $\top$ and for false we denote $\bot$. Since $d \mid cx + s \equiv d \mid -cx + -s$, we may assume that $c > 0$ for all $d \mid cx + s \in C$. A variable x is *guarded* in a problem C if C contains constraints of the form $x - u_x \leq 0$ and $-x + l_x \leq 0$. Otherwise, x is *unguarded* in C. Note that guarded variables are *bounded* as defined in [8] but not vice versa. A constraint is *guarded* if it contains only guarded variables. Otherwise, it is *unguarded*.

Our algorithm CUTSAT++ aims at deciding whether or not a given problem C is satisfiable. It either ends in the state *unsat* or in a state $\langle v, \text{sat} \rangle$, where v is a satisfiable assignment for C. In order to reach one of those two final states, the algorithm produces *lower bounds* $x \geq b$ and *upper bounds* $x \leq b$ for the variables in C. The produced bounds are stored in a sequence $M = [\![\gamma_1, \ldots, \gamma_n]\!]$, which describes a partial model. The empty sequence is denoted by $[\![]\!]$. We use $[\![M, \gamma]\!]$ and $[\![M_1, M_2]\!]$ to denote the concatenation of a bound γ at the end of M and M_2 at the end of M_1, respectively.

By $\text{lower}(x, M) = b$ and $\text{upper}(x, M) = b$ we denote the value b of the greatest lower bound $x \geq b$ and least upper bound $x \leq b$ for a variable x in M, respectively. If there is no lower (upper) bound for x in M, then $\text{lower}(x, M) = -\infty$ ($\text{upper}(x, M) = \infty$). The definitions of upper and lower are extended to polynomials as done in [8]. The partial model M is complete if all variables x are *fixed* in the sense that $\text{upper}(x, M) = \text{lower}(x, M)$. In this case we define $v[M]$ as the assignment that assigns to every variable x the value $\text{lower}(x, M)$.

A state in CUTSAT++ is of the form $S = \langle M, C \rangle$ or $S = \langle M, C \rangle \vdash I$, or one of the two *final states* $\langle v, \text{sat} \rangle$, *unsat*. The *initial-state* for a problem C is $\langle [\![]\!], C \rangle$. For a state $S = \langle M, C \rangle (\vdash I)$, an inequality $p \leq 0$ is a *conflict* if $\text{lower}(p, M) > 0$. For a state $S = \langle M, C \rangle (\vdash I)$, a divisibility constraint $d \mid ax + p$ is a *conflict* if all variables in p are fixed and $d \nmid ab + \text{lower}(p, M)$ for all b such that $\text{lower}(x, M) \leq b \leq \text{upper}(x, M)$. In a state $S = \langle M, C \rangle \vdash I$, the constraint I is always a conflict. A state is *frozen* if it is not a final state and no rule is applicable.

Via applications of the rule Decide, CUTSAT++ adds *decided bounds* $x \leq b$ or $x \geq b$ to the sequence M in state S [8]. A decided bound generally assigns a variable x to the lower or upper bound of x in M. Via applications of the propagation rules, CUTSAT++ adds *propagated bounds* $x \geq_I b$ or $x \leq_I b$ to the sequence M, where I is a generated constraint, called justification. To this end, the function $\text{bound}(J, x, M)$ computes the strictest bound b and the function $\text{tight}(J, x, M)$ computes the corresponding justification I for constraint J under the partial model M.

We are now going to discuss three examples where CUTSAT diverges. The first one shows that CUTSAT can apply Conflict and Conflict-Div infinitely often to constraints containing unguarded variables.

Example 1. Let

$$C := \{\underbrace{-x \leq 0}_{I_x}, \underbrace{-y \leq 0}_{I_y}, \underbrace{-z \leq 0}_{I_{z1}}, \underbrace{z \leq 0}_{I_{z2}}, \underbrace{z + 1 \leq 0}_{I_{z3}}, \underbrace{1 - x + y \leq 0}_{J_1}, \underbrace{x - y \leq 0}_{J_2}\}$$

be a problem. Let $S_i = \langle M_i, C \rangle$ for $i \in \mathbb{N}$ be a series of states with:

$$\begin{aligned} M_0 &:= [\![x \geq_{I_x} 0, y \geq_{I_y} 0, z \geq_{I_{z1}} 0, z \leq_{I_{z2}} 0]\!], \\ M_{i+1} &:= [\![M_i, x \geq_{J_1} i + 1, y \geq_{J_2} i + 1]\!]. \end{aligned}$$

Let the variable order be given by $z \prec y \prec x$. CUTSAT with a two-layered strategy, after propagating all constraints I_x, I_y, I_{z1}, I_{z2} , applies the rules Decide, Conflict, and Backjump to propagate arbitrarily large lower bounds for the unguarded variables x and y and, therefore, diverges. Notice that the conflicting core $\{I_{z1}, I_{z3}\}$ is guarded, which admits the application of Conflict.

A straightforward fix to Example 1 is to limit the application of the Conflict and Conflict-Div rules to guarded constraints. Our second example shows that CUTSAT can still diverge by infinitely many applications of the Solve-Div rule [8].

Example 2. Let d_i be the sequence with $d_0 := 2$ and $d_{k+1} := {d_k}^2$ for $k \in \mathbb{N}$, let $C_0 = \{4 \mid 2x + 2y, 2 \mid x + z\}$ be a problem, and let $S_0 = \langle [\![]\!], C_0 \rangle$ be the initial CUTSAT state. Let the variable order be given by $x \prec y \prec z$. Then CUTSAT has divergent runs $S_0 \Rightarrow_{CS} S_1 \Rightarrow_{CS} S_2 \Rightarrow_{CS} \ldots$. For instance, let CUTSAT apply the Solve-Div rule whenever applicable. By an inductive argument, Solve-Div is applicable in every state $S_n = \langle [\![]\!], C_n \rangle$, and the constraint set C_n has the following form:

$$C_n = \begin{cases} \{2d_n \mid d_n x + d_n y, d_n \mid \frac{d_n}{2} y - \frac{d_n}{2} z\} & \text{if } n \text{ is odd,} \\ \{2d_n \mid d_n x + d_n y, d_n \mid \frac{d_n}{2} x + \frac{d_n}{2} z\} & \text{if } n \text{ is even.} \end{cases}$$

Therefore, CUTSAT applies Solve-Div infinitely often and diverges.

A straightforward fix to Example 2 is to limit the application of Solve-Div to maximal variables in the variable order $\prec$. Our third example shows that

CUTSAT can apply Conflict and Conflict-Div [8] infinitely often. The Example 3 differs from Example 1 in that the conflicting core contains also unguarded variables.

Example 3. Let

$$C := \{\underbrace{-x \leq 0}_{I_x}, \underbrace{-y \leq 0}_{I_y}, \underbrace{-z \leq 0}_{I_{z1}}, \underbrace{z \leq 0}_{I_{z2}}, \underbrace{1 - x + y + z \leq 0}_{J_1}, \underbrace{x - y - z \leq 0}_{J_2}\}$$

be a problem. Let $S_i = \langle M_i, C\rangle$ for $i \in \mathbb{N}$ be a series of states with:

$$\begin{aligned} M_0 &:= [\![x \geq_{I_x} 0, y \geq_{I_y} 0, z \geq_{I_{z1}} 0, z \leq_{I_{z2}} 0]\!], \\ M_{i+1} &:= [\![M_i, x \geq_{J_1} i+1, y \geq_{J_2} i+1]\!]. \end{aligned}$$

Let the variable order be given by $z \prec x \prec y$. CUTSAT with a two-layered strategy, after propagating all constraints I_x, I_y, I_{z1}, I_{z2}, possibly applies the rules Decide, Conflict, and Backjump to propagate arbitrarily large lower bounds for the unguarded variables x and y and, thus, diverges. Notice that the conflicting core $\{J_1, J_2\}$ is bounded in [8] after we fix x and y with Decide to their current respective lower bounds. This in turn admits the application of Conflict.

Applying the fix suggested for Examples 1–3 results in a frozen state. Here, a straightforward fix is to change the definition of conflicting cores to cover only those cores where the conflicting variable is the maximal variable.[1]

The fixes that we suggested for the above examples are restrictions to CUTSAT which have the consequence that Conflict(-Div) cannot be applied to unguarded constraints, Solve-Div is only applicable for the elimination of the maximal variable, and the conflicting variable x is the maximal variable in the associated conflicting core C'. However, our next and final example shows that these restrictions lead to frozen states.

Example 4. Let CUTSAT include restrictions to maximal variables in the definition of conflicting cores and in the Solve-Div rule as described above. Let there be additional restrictions in CUTSAT to the rules Conflict and Conflict-Div such that these rules are only applicable to conflicts that contain no unguarded variable. Let

$$C := \{\underbrace{-x \leq 0}_{I_{x1}}, \underbrace{x - 1 \leq 0}_{I_{x2}}, \underbrace{-y \leq 0}_{I_y}, \underbrace{6 \mid 4y + x}_{J}\}$$

be a problem. Let $M := [\![x \geq_{I_{x1}} 0, x \leq_{I_{x2}} 1, y \geq_{I_y} 0, x \geq 1, y \leq 0]\!]$ be a bound sequence. Let the variable order be given by $x \prec y$. CUTSAT has a run starting in state $S'_0 = \langle [\![]\!], C\rangle$ that ends in the frozen state $S = \langle M, C\rangle$. Let CUTSAT propagate I_{x1}, I_{x2}, I_y and fix x to 1 and y to 0 with two Decisions. Through these Decisions, the constraint J is a conflict. Since y is unguarded, CUTSAT cannot apply the rule Conflict-Div. Furthermore, [8] has defined conflicting cores

[1] The restrictions to maximal variables in the definition of conflicting cores and to the Solve-Div rule were both confirmed as missing but necessary in a private communication with Jovanović.

as either interval or divisibility conflicting cores. The state S contains neither an interval or a divisibility conflicting core. Therefore, CUTSAT cannot apply the rule Resolve-Cooper. The remaining rules are also not applicable because all variables are fixed and there is only one divisibility constraint. Without the before introduced restriction to the rule Conflict(-Div), CUTSAT diverges on the example. For more details see [2].

3 Weak Cooper Elimination

In order to fix the frozen state of Example 4 in the previous section, we are going to introduce in Sect. 4 a new conflicting core, which we call *diophantine conflicting core*. For understanding diophantine conflicting cores, as well as further modifications to be made, it is helpful to understand the connection between CUTSAT and a variant of Cooper's quantifier elimination procedure [3].

The original *Cooper elimination* takes a variable x, a problem $C(x)$, and produces a disjunction of problems equivalent to $\exists x.C(x)$:

$$\exists x.C(x) \equiv \bigvee_{0 \leq k < m} C_{-\infty}(k) \vee \bigvee_{-ax+p \leq 0 \in C(x)} \bigvee_{0 \leq k < a \cdot m} \left[a \mid p + k \wedge C\left(\frac{p+k}{a}\right) \right],$$

where $a > 0$ and $m = \text{lcm}\{d \in \mathbb{Z} : (d \mid ax + p) \in C(x)\}$. If there exists no constraint of the form $-ax+p \leq 0 \in C(x)$, then $C_{-\infty}(x) = \{(d \mid ax+p) \in C(x)\}$. Otherwise, $C_{-\infty}(x) = \bot$. One application of Cooper elimination results in a disjunction of quadratically many problems out of a single problem. Iteration causes an exponential increase in the coefficients due to the multiplication with a because division is not part of the language.

Our notion of *weak Cooper elimination* is a variant of Cooper elimination, which is very helpful to understand problems around CUTSAT. The idea is, instead of building a disjunction over all potential solutions for x, to add additional guarded variables and constraints without x that guarantee the existence of a solution for x. We assume here that $C(x)$ contains only one divisibility constraint for x. If not, exhaustive application of div-solve to divisibility constraints for x removes all constraints except one: $\text{div-solve}(x, d_1 \mid a_1x+p_1, d_2 \mid a_2x+p_2) = (d_1d_2 \mid dx + c_1d_2p_1 + c_2d_1p_2, d \mid -a_1p_2 + a_2p_1)$, where $d = \gcd(a_1d_2, a_2d_1)$, and c_1 and c_2 are integers such that $c_1a_1d_2 + c_2a_2d_1 = d$ [3,8]. Now weak Cooper elimination takes a variable x, a problem $C(x)$, and produces a new problem by replacing $\exists x.C(x)$ with:

$$\exists K. \left(\{I \in C(x) : \text{coeff}(x, I) = 0\} \cup \{\gcd(c, d) \mid s\} \cup \bigcup_{k \in K} R_k \right)$$

where $d \mid cx + s \in C(x)$, $k \in K$ is a newly introduced variable for every pair of constraints $-ax + p \leq 0 \in C(x)$ and $bx - q \leq 0 \in C(x)$, and

$$R_k = \{-k \leq 0, k - m \leq 0, bp - aq + bk \leq 0, a \mid k + p, ad \mid cp + as + ck\}$$

is a resolvent for the same inequalities, where $m := \text{lcm}\left(a, \frac{ad}{\gcd(ad,c)}\right) - 1$. Note

the existential quantifier $\exists K$, where all variables $k \in K$ are guarded by their respective R_k.

Let ν be a satisfiable assignment for the formula after one weak Cooper elimination step on $C(x)$. Then we compute a strictest lower bound $x \geq l_x$ and a strictest upper bound $x \leq u_x$ from $C(x)$ for the variable x under the assignment ν. We now argue that there is a value for x such that $x \geq l_x$, $x \leq u_x$, and $d \mid cx + s$ are all satisfied. Whenever $l_x \neq -\infty$ and $u_x \neq \infty$, the bounds $x \geq l_x$, $x \leq u_x$ are given by respective constraints of the form $-ax + p \leq 0 \in C(x)$ and $bx - q \leq 0 \in C(x)$ such that $l_x = \lceil \frac{\nu(p)}{a} \rceil$ and $u_x = \lfloor \frac{\nu(q)}{b} \rfloor$. In this case, the extension of ν with $\nu(x) = \frac{\nu(k+p)}{a}$ satisfies $C(x)$ because the constraint $a \mid k + p \in R_k$ guarantees that $\nu(x) \in \mathbb{Z}$, the constraint $bp - aq + bk \leq 0 \in R_k$ guarantees that $l_x \leq \nu(x) \leq u_x$, and the constraint $ad \mid cp + as + ck \in R_k$ guarantees that ν satisfies $d \mid cx + s \in C(x)$. Whenever $l_x = -\infty$ ($u_x = \infty$) we extend ν by an arbitrary small (large) value for x that satisfies $d \mid cx + s \in C(x)$. There exist arbitrarily small (large) solutions for x and $d \mid cx + \nu(s)$ because $\gcd(c, d) \mid s$ is satisfied by ν.

The advantage of weak Cooper elimination, compared to Cooper elimination, is that the output is still a conjunctive problem in contrast to a disjunction of problems. CUTSAT++ performs weak Cooper elimination not in one step but subsequently adds to the states the constraints from the R_k as well as the divisibility constraint $\gcd(c, d) \mid s$ with respect to a strict ordering on the unguarded variables.

The following Theorem, for which we have just outlined the proof, states the correctness of weak Cooper elimination.

Theorem 5.

$$\exists x.C(x) \equiv \exists K. \left(\{I \in C(x) : \text{coeff}(x, I) = 0\} \cup \{\gcd(c, d) \mid s\} \cup \bigcup_{k \in K} R_k \right)$$

The extra divisibility constraint $\gcd(c, d) \mid s$ in weak Cooper elimination is necessary whenever the problem $C(x)$ has no constraint of the form $-ax + p \leq 0 \in C(x)$ or $bx - q \leq 0 \in C(x)$. For example, let $C(x) = \{y - 1 \leq 0, -y + 1 \leq 0, 6 \mid 2x + y\}$ be a problem and x be the unguarded variable we want to eliminate. As there are no inequalities containing x, weak Cooper elimination without the extra divisibility constraint returns $C' = \{y - 1 \leq 0, -y + 1 \leq 0\}$. While C' has a satisfiable assignment $\nu(y) = 1$, $C(x)$ has not since $2x + 1$ is never divisible by 2 or 6.

For any R_k introduced by weak Cooper elimination we can also show the following Lemma:

Lemma 6. *Let k be a new variable. Let $a, b, c > 0$. Then,*

$$\begin{aligned} &(\exists x.\{-ax + p \leq 0, bx - q \leq 0, d \mid cx + s\}) \\ \equiv\ &(\exists k.\{-k \leq 0, k - m \leq 0, bp - aq + bk \leq 0, a \mid k + p,\ ad \mid cp + as + ck\}). \end{aligned}$$

That means satisfiability of the respective R_k guarantees a solution for the triple of constraints it is derived from. An analogous Lemma holds for the divisibility constraint $gcd(c, d) \mid s$ introduced by weak Cooper elimination:

Lemma 7.

$$(\exists x.d \mid cx + s) \equiv \gcd(c, d) \mid s.$$

That means satisfiability of $\gcd(c, d) \mid s$ guarantees a solution for the divisibility constraint $d \mid cx + s$. The rule Resolve-Cooper (Fig. 1) in our CUTSAT++ exploits these properties by generating the R_k and constraint $\gcd(c, d) \mid s$ in the form of strong resolvents in a lazy way. Furthermore, it is not necessary for the divisibility constraints to be a priori reduced to one, as done for weak Cooper elimination. Instead, the rules Solve-Div-Left and Solve-Div-Right (Fig. 1) perform lazy reduction.

4 Strong Conflict Resolution Revisited

Weak Cooper elimination is capable of exploring all unguarded variables to eventually create a problem where feasibility only depends on guarded variables. It is simulated in a lazy manner through an additional set of CUTSAT++ rules (Fig. 1). Instead of eliminating all unguarded variables before the application of CUTSAT++, the rules perform the same intermediate steps as weak Cooper elimination, viz., the combination of divisibility constraints via div-solve and the construction of resolvents, to resolve and block conflicts in unguarded constraints. As a result, CUTSAT++ can avoid some of the intermediate steps of weak Cooper elimination. Furthermore, CUTSAT++ is not required to apply the intermediate steps of weak Cooper elimination one variable at a time. The lazy approach of CUTSAT++ does not eliminate unguarded variables. In the worst case CUTSAT++ has to perform all of weak Cooper elimination's intermediate steps. Then the strictly-two-layered strategy (Definition 13) guarantees that CUTSAT++ recognizes that all unguarded conflicts have been produced.

The eventual result is the complete algorithm CUTSAT++, which is a combination of the rules Resolve-Cooper, Solve-Div-Left, Solve-Div-Right (Fig. 1), a strictly-two-layered strategy (Definition 13), and the CUTSAT rules: Propagate, Propagate-Div, Decide, Conflict, Conflict-Div, Sat, Unsat-Div, Forget, Slack-Intro[2], Resolve, Skip-Decision, Backjump, Unsat, and Learn [2,8].

The lazy approach has the advantage that CUTSAT++ might find a satisfiable assignment or detect unsatisfiability without encountering and resolving a large number of unguarded conflicts. This means the number of divisibility constraint combinations and introduced resolvents might be much smaller in the lazy approach of CUTSAT++ than during the elimination with weak Cooper elimination.

In order to simulate weak Cooper elimination, CUTSAT++ uses a total order $\prec$ over all variables such that $y \prec x$ for all guarded variables y and unguarded variables x. While termination requires that the order is fixed from the beginning for all unguarded variables, the ordering among the guarded variables can be dynamically changed. In relation to weak Cooper elimination, the order $\prec$

[2] As recommended in [8], CUTSAT++ uses the same slack variable for all Slack-Intro applications.

describes the elimination order for the unguarded variables, viz., $x_i \prec x_j$ if x_j is eliminated before x_i. A variable x is called *maximal* in a constraint I if x is contained in I and all other variables in I are smaller, i.e., $y \prec x$. The maximal variable in I is also called its *top variable* ($x = \mathrm{top}(I)$).

Definition 8. *Let $S = \langle M, C\rangle$ be a state, $C' \subseteq C$, x the top variable in C', and let all other variables in C' be fixed. The pair (x, C') is a* conflicting core *if it is of one of the following three forms*

(1) $C' = \{-ax + p \leq 0, bx - q \leq 0\}$ and the lower bound from $-ax + p \leq 0$ contradicts the upper bound from $bx - q \leq 0$, i.e., $\mathrm{bound}(-ax + p \leq 0, x, M) > \mathrm{bound}(bx - q \leq 0, x, M)$*; in this case (x, C') is called an* interval conflicting core *and its strong resolvent is* $(\{-k \leq 0, k - a + 1 \leq 0\}, \{bp - aq + bk \leq 0, a \mid k + p\})$.
(2) $C' = \{-ax + p \leq 0, bx - q \leq 0, d \mid cx + s\}$ and $b_l = \mathrm{bound}(-ax + p \leq 0, x, M)$, $b_u = \mathrm{bound}(bx - q \leq 0, x, M)$, $b_l \leq b_u$, and for all $b_d \in [b_l, b_u]$ we have $d \nmid cb_d + \mathrm{lower}(s, M)$; in this case (x, C') is called a divisibility conflicting core *and its strong resolvent is* $(\{-k \leq 0, k - m \leq 0\}, \{bp - aq + bk \leq 0, a \mid k + p, ad \mid cp + as + ck\})$.
(3) $C' = \{d \mid cx + s\}$ and for all $b_d \in \mathbb{Z}$ we have $d \nmid cb_d + \mathrm{lower}(s, M)$; in this case (x, C') is called a diophantine conflicting core *and its strong resolvent is* $(\emptyset, \{gcd(c, d) \mid s\})$.
In the first two cases k is a fresh variable and $m = \mathrm{lcm}\left(a, \frac{ad}{\gcd(ad,c)}\right) - 1$.

We refer to the respective strong resolvents for a conflicting core (x, C') by the function $\mathrm{cooper}(x, C')$, which returns a pair (R_k, R_c) as defined above. Note that the newly introduced variable k is guarded by the constraints in R_k. If there is a conflicting core (x, C') in some state S, then x is called a *conflicting variable*. A *potential conflicting core* is a pair (x, C') if there exists a state S where (x, C') is a conflicting core.

Next, we define a generalization of strong resolvents. Since the strong resolvents generated out of conflicting cores will be further processed by CUTSAT++, we must guarantee that any set of constraints implying the feasibility of the conflicting core constraints prevents a second application of Resolve-Cooper to the same conflicting core. All strong resolvents of Definition 8 are also strong resolvents in the sense of the below definition (see also end of Sect. 3).

Definition 9. *A set of constraints R is a* strong resolvent *for the pair (x, C') if it holds that $R \rightarrow \exists x.C'$ and $\forall J \in R.\, \mathrm{top}(J) \prec x$.*

The rule Resolve-Cooper (Fig. 1) requires that the conflicting variable x of the conflicting core (x, C') is the top variable in the constraints of C'. This simulates a setting where all variables y with $x \prec y$ are already eliminated. We restrict Resolve-Cooper to unguarded constraints because weak Cooper elimination modifies only unguarded constraints.

Lemma 10. *Let $S = \langle M, C\rangle$ be a CUTSAT++ state. Let $C' \subseteq C$ and x be an unguarded variable. Let $R \subseteq C$ be a* strong resolvent *for (x, C'). Then Resolve-Cooper is not applicable to (x, C').*

For the resolvent R to block Resolve-Cooper from being applied to the conflicting core (x, C'), CUTSAT++ has to detect all conflicts in R. Detecting all conflicts in R is only possible if CUTSAT++ fixes all variables y with $y \prec x$ and if Resolve-Cooper is only applicable if there exists no conflict I with $\text{top}(I) \prec x$. Therefore, the remaining restrictions of Resolve-Cooper justify the above Lemma.

If we add strong resolvents again and again, then CUTSAT++ will reach a state after which every encounter of a conflicting core guarantees a conflict in a guarded constraint. From this point forward, CUTSAT++ will not apply Resolve-Cooper. The remaining guarded conflicts are resolved with the rules Conflict and Conflict-Div [8].

The rules Solve-Div-Left and Solve-Div-Right (Fig. 1) combine divisibility constraints as it is done a priori to weak Cooper elimination. In these rules, we restrict the application of div-solve(x, I_1, I_2) to constraints where x is the top variable and where all variables y in I_1 and I_2 with $y \neq x$ are fixed. The ordering restriction simulates the order of elimination, i.e., we apply div-solve(x, I_1, I_2) in a (simulated) setting where all variables y with $x \prec y$ appear to be eliminated in I_1 and I_2. Otherwise, divergence would be possible (see Example 2). Requiring smaller variables to be fixed prevents the accidental generation of a conflict for an unguarded variable x_i by div-solve(x, I_1, I_2).

Thanks to an eager top-level propagating strategy, as defined below, any unguarded conflict in CUTSAT++ is either resolved with Solve-Div-Right (Fig. 1) or CUTSAT++ constructs a conflicting core that is resolved with Resolve-Cooper. Both cases may require multiple applications of the Solve-Div-Left rule (Fig. 1). We define the following further restrictions on the CUTSAT++ rules, which will eventually generate the above described behavior.

Definition 11. *Let $\bowtie \in \{\leq, \geq\}$. We call a strategy for CUTSAT++* eager top-level propagating *if we restrict propagations and decisions for every state $\langle M, C\rangle$ in the following way:*

1. *Let x be an unguarded variable. Then we only allow to propagate bounds $x \bowtie \text{bound}(I, x, M)$ if x is the top variable in I. Furthermore, if I is a divisibility constraint $d \mid ax + p$, then we only propagate $d \mid ax + p$ if:*
   (a) *either* $\text{lower}(x, M) \neq -\infty$ *and* $\text{upper}(x, M) \neq \infty$ *or*
   (b) $\gcd(a, d) \mid \text{lower}(p, M)$ *holds and $d \mid ax + p$ is the only divisibility constraint in C with x as top variable.*
2. *Let x be an unguarded variable. Then we only allow decisions $\gamma = x \bowtie b$ if:*
   (a) *for every constraint $I \in C$ with $x = \text{top}(I)$ all occurring variables $y \neq x$ are fixed*
   (b) *there exists no $I \in C$ where $x = \text{top}(I)$ and I is a conflict in $[\![M, \gamma]\!]$*
   (c) *either* $\text{lower}(x, M) \neq -\infty$ *and* $\text{upper}(x, M) \neq \infty$ *or there exists at most one divisibility constraint in C with x as top variable.*

An eager top-level propagating strategy has two advantages. First, the strategy dictates an order of influence over the variables, i.e., a bound for unguarded variable x is influenced only by previously propagated bounds for variables y with

Solve-Div-Left

$$\langle M, C\rangle \Rightarrow_{\text{CS}} \langle M, C'\rangle \quad \textbf{if} \quad \begin{cases} \text{divisibility constraints } I_1, I_2 \in C, \\ x \text{ is unguarded and top in } I_1 \text{ and } I_2, \\ \text{all other vars. in } I_1, I_2 \text{ are fixed}, \\ (I_1', I_2') = \text{div-solve}(x, I_1, I_2), \\ C' = (C \setminus \{I_1, I_2\}) \cup \{I_1', I_2'\}, \\ I_2' \text{ is not a conflict} \end{cases}$$

Solve-Div-Right

$$\langle M, C\rangle \Rightarrow_{\text{CS}} \langle M', C'\rangle \quad \textbf{if} \quad \begin{cases} \text{divisibility constraints } I_1, I_2 \in C, \\ x \text{ is unguarded and top in } I_1 \text{ and } I_2, \\ \text{all other vars. in } I_1, I_2 \text{ are fixed}, \\ (I_1', I_2') = \text{div-solve}(x, I_1, I_2), \\ C' = (C \setminus \{I_1, I_2\}) \cup \{I_1', I_2'\}, \\ I_2' \text{ is a conflict}, \\ y = \text{top}(I_2'), \\ M' = \text{prefix}(M, y) \end{cases}$$

Resolve-Cooper

$$\langle M, C\rangle \Rightarrow_{\text{CS}} \langle M', C \cup R_k \cup R_c\rangle \quad \textbf{if} \quad \begin{cases} (x, C') \text{ is a conflicting core}, \\ x \text{ is unguarded}, \\ \text{all } z \prec x \text{ are fixed and } C' \subseteq C, \\ \text{if } J \in C \text{ is a conflict, then } \text{top}(J) \not\prec x, \\ \text{cooper}(x, C') = (R_k, R_c), \\ y = \min_{I \in R_c}\{\text{top}(I)\}, \\ M' = \text{prefix}(M, y) \end{cases}$$

In the above rules, $M' = \text{prefix}(M, y)$ defines the largest prefix of M that contains only decided bounds for variables x with $x \prec y$.

Fig. 1. Our strong conflict resolution rules

$y \prec x$. Furthermore, the strategy makes only decisions for unguarded variable x when all constraints with $x = \text{top}(I)$ are fixed and satisfied by the decision. This means, any conflict $I \in C$ with $x = \text{top}(I)$ is impossible as long as the decision for x remains on the bound sequence. For the same purpose, i.e., avoiding conflicts I where $x = \text{top}(I)$ is fixed by a decision, CUTSAT++ backjumps in the rules Resolve-Cooper and Solve-Div-Right to a state where this is not the case.

Definition 12. *A strategy is* reasonable *if Propagate applied to constraints of the form* $\pm x - b \leq 0$ *has the highest priority over all rules and the Forget Rule is applied only finitely often [8].*

Definition 13. *A strategy is* strictly-two-layered *if:*
(1) it is reasonable, (2) it is eager top-level propagating, (3) the Forget, Conflict, Conflict-Div rules only apply to guarded constraints, (4) Forget cannot be applied to a divisibility constraint or a constraint contained in a strong resolvent, and (5) only guarded constraints are used to propagate guarded variables.

The above *strictly-two-layered* strategy is the final restriction to CUTSAT++. With the condition 13-(3) it partitions conflict resolution into two layers: While every unguarded conflict is handled with the rules Resolve-Cooper,

Forget

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

Slack-Intro

$$\langle M, C\rangle \Rightarrow_{\text{CS}} \langle M, C \cup C_s\rangle \quad \textbf{if } \begin{cases} \langle M, C\rangle \text{ is stuck,} \\ x \text{ is stuck,} \\ x_S \text{ is the slack-variable,} \\ C_s = \{-x_S \leq 0, x - x_S \leq 0, \\ \qquad -x - x_S \leq 0\} \end{cases}$$

Sat

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

Unsat

$$\langle M, C\rangle \vdash b \leq 0 \Rightarrow_{\text{CS}} \text{unsat} \quad \textbf{if } b > 0$$

Unsat-Div

$$\langle M, C\rangle \Rightarrow_{\text{CS}} \text{unsat} \quad \textbf{if } \begin{cases} d \mid a_1 x_1 + \ldots + a_n x_n + c \in C, \\ \gcd(d, a_1, \ldots, a_n) \nmid c \end{cases}$$

A variable x is called *stuck* in state $S = \langle M, C\rangle$ if M contains no bounds for x and there is no inequality $I = ax + p \leq 0 \in C$ that propagates a bound for x [8]. Variables x with a constraint of the form $\pm x - b \leq 0 \in C$ are never stuck as CUTSAT++ is able to propagate at least one bound for x, i.e., either $x \geq -b$ or $x \leq b$. A state S is a *stuck state* if all unfixed variables x are stuck and if the rules Sat, Unsat-Div, Conflict, and Conflict-Div are not applicable [8]. The slack variable x_S is the smallest unguarded variable in $\prec$. As long as Slack-Intro is never applied, we treat x_S as non existent.

Fig. 2. The Forget, Slack-Intro, Sat, Unsat, and Unsat-Div rules

Solve-Div-Left, and Solve-Div-Right (Fig. 1), every guarded conflict is handled with the rules Conflict(-Div) [2]. The conditions 13-(1) and 13-(5) make the guarded variables independent from the unguarded variables. The conditions 13-(2) and 13-(4) give a guarantee that the rules Resolve-Cooper, Solve-Div-Left, and Solve-Div-Right are applied at most finitely often. We assume for the remainder of the paper that all runs of CUTSAT++ follow a strictly-two-layered strategy.

5 Termination and Completeness

The CUTSAT++ rules are Propagate, Propagate-Div, Decide, Conflict, Conflict-Div, Sat, Unsat-Div, Forget, Slack-Intro, Resolve, Skip-Decision, Backjump, Unsat, and Learn [2,8], as well as Resolve-Cooper, Solve-Div-Left, and Solve-Div-Right (Fig. 1). For the termination proof of CUTSAT++, we consider a (possibly infinite) sequence of rule applications $\langle [\![], C_0\rangle = S_0 \Rightarrow_{\mathsf{CS}} S_1 \Rightarrow_{\mathsf{CS}} \ldots$ on a problem C_0, following the strictly-two-layered strategy.

First, this sequence reaches a state S_s ($s \in \mathbb{N}_0^+$) after a finite derivation of rule applications $S_0 \Rightarrow_{\mathsf{CS}} \ldots \Rightarrow_{\mathsf{CS}} S_s$ such that there is no further application of the rules Slack-Intro and Forget (Fig. 2) after state S_s: Such a state S_s exists for two reasons: Firstly, the strictly-two-layered strategy employed by CUTSAT++ is

also reasonable. The reasonable strategy explicitly forbids infinite applications of the rule Forget. Secondly, the Slack-Intro rule is applicable only to stuck variables and only once to each stuck variable. Only the initial set of variables can be stuck because all variables x introduced during the considered derivation are introduced with at least one constraint $x - b \leq 0$ that allows at least one propagation for the variable. Therefore, the rules Slack-Intro and Forget are applicable at most finitely often.

Next, the sequence reaches a state S_w $(w \geq s)$ after a finite derivation of rule applications $S_s \Rightarrow_{\mathsf{CS}} \ldots \Rightarrow_{\mathsf{CS}} S_w$ such that there is no further application of the rules Resolve-Cooper, Solve-Div-Left, and Solve-Div-Right after state S_w: The rules Resolve-Cooper, Solve-Div-Left, Solve-Div-Right, and Slack-Intro are applicable only to unguarded constraints. Through the strictly-two-layered strategy, they are also the only rules producing unguarded constraints. Therefore, they form a closed loop with respect to unguarded constraints, which we use in our termination proof. We have shown in the previous paragraph that $S_s \Rightarrow_{\mathsf{CS}} \ldots \Rightarrow_{\mathsf{CS}} S_w$ contains no application of the rule Slack-Intro. By Lemma 10, an application of Resolve-Cooper to the conflicting core (x, C') prevents any further applications of Resolve-Cooper to the same core. By Definition 8, the constraints learned through an application of Resolve-Cooper contain only variables y such that $y \prec x$. Therefore, an application of Resolve-Cooper blocks a conflicting core (x, C') and introduces potential conflicting cores only for smaller variables than x. This strict decrease in the conflicting variables guarantees that we encounter only finitely many conflicting cores in unguarded variables. Therefore, Resolve-Cooper is applicable at most finitely often. An analogous argument applies to the rules Solve-Div-Left and Solve-Div-Right. Thus the rules Resolve-Cooper, Solve-Div-Left, and Solve-Div-Right are applicable at most finitely often.

Next, the sequence reaches a state S_b $(b \geq w)$ after a finite derivation of rule applications $S_w \Rightarrow_{\mathsf{CS}} \ldots \Rightarrow_{\mathsf{CS}} S_b$ such that for every guarded variable x the bounds remain invariant, i.e., $\text{lower}(x, M_b) = \text{lower}(x, M_j)$ and $\text{upper}(x, M_b) = \text{upper}(x, M_j)$ for every state $S_j = \langle M_j, C_j \rangle (\vdash I_j)$ after $S_b = \langle M_b, C_b \rangle (\vdash I_b)$ $(j \geq b)$: The strictly-two-layered strategy guarantees that only bounds of guarded variables influence the propagation of further bounds for guarded variables. Any rule application involving unguarded variables does not influence the bounds for guarded variables. A proof for the termination of the solely guarded case was already provided in [8]. We now know that the sequence after S_b contains no further propagations, decisions, or conflict resolutions for the guarded variables.

Next, the sequence reaches a state S_u $(u \geq b)$ after a finite derivation of rule applications $S_b \Rightarrow_{\mathsf{CS}} \ldots \Rightarrow_{\mathsf{CS}} S_u$ such that also for every unguarded variable x the bounds remain invariant, i.e., $\text{lower}(x, M_b) = \text{lower}(x, M_j)$ and $\text{upper}(x, M_b) = \text{upper}(x, M_j)$ for every state $S_j = \langle M_j, C_j \rangle (\vdash I_j)$ after $S_u = \langle M_u, C_b \rangle (\vdash I_u)$ $(j \geq u)$. After S_b, CUTSAT++ propagates and decides only unguarded variables or ends with an application of Sat or Unsat(-Div). CUTSAT++ employs the strictly-two-layered strategy, which is also an eager top-level propagating strategy. Through the top variable restriction for propagating constraints, the

eager top-level propagating strategy induces a strict order of propagation over the unguarded variables. Therefore, any bound for an unguarded variable x is influenced only by bounds for variables $y \prec x$. This strict variable order guarantees that unguarded variables are propagated and decided only finitely often.

After state S_u, only the rules Sat, Unsat, and Unsat-Div are applicable, which lead all to a final state. Hence, the sequence $S_0 \Rightarrow_{\mathsf{CS}} S_1 \Rightarrow_{\mathsf{CS}} \ldots$ is finite. We conclude that CUTSAT++ always terminates:

Theorem 14. *If CUTSAT++ starts from an initial state* $\langle [\![], C_0 \rangle$*, then there is no infinite derivation sequence.*

All CUTSAT++ rules are sound, i.e., if $\langle M_i, C_i \rangle (\vdash I_i) \Rightarrow_{\mathsf{CS}} \langle M_j, C_j \rangle (\vdash I_j)$, then any satisfiable assignment υ for C_j is a satisfiable assignment also for C_i. The rule Resolve-Cooper is sound because of the Lemmas 6 and 7. The soundness of Solve-Div-Left and Solve-Div-Right follows from the fact that div-solve is an equivalence preserving transformation. The soundness proofs for all other rules are either trivial or given in [8].

Furthermore, CUTSAT++ never reaches a frozen state. Let x be the smallest unfixed variable with respect to $\prec$. Whenever x is guarded we can propagate a constraint $\pm x - b \leq 0 \in C$ and then fix x by introducing a decision. If we cannot propagate any bound for x, then x is unguarded and stuck and, therefore, Slack-Intro is applicable. If we cannot fix x by introducing a decision, then x is unguarded and there is a conflict. Guarded conflicts are resolved via the Conflict(-Div) rules. Unguarded conflicts are resolved via the strong conflict resolution rules. Unless a final state is reached, CUTSAT has always a rule applicable.

Summarizing, CUTSAT++ is terminating, sound, and never reaches a frozen state. In combination with the fact that Sat is applicable only if a satisfiable solution $\upsilon[M]$ is found and that Unsat and Unsat-Div detect trivially unsatisfiable constraints, these facts imply completeness:

Theorem 15. *If CUTSAT++ starts from an initial state* $\langle [\![], C_0 \rangle$*, then it either terminates in the* unsat *state and* C_0 *is unsatisfiable, or it terminates with* $\langle \upsilon, \mathrm{sat} \rangle$ *where* υ *is a satisfiable assignment for* C_0*.*

6 Conclusion and Future Work

The starting point of our work was an implementation of CUTSAT [8] as a theory solver for hierarchic superposition [5]. In that course, we observed divergence for some of our problems. The analysis of those divergences led to the development of the CUTSAT++ algorithm presented in this paper, which is a substantial extension of CUTSAT by means of the weak Cooper elimination described in Sect. 3.

As a next step, we plan to develop a prototypical implementation of CUTSAT++, to test its efficiency on benchmark problems. Depending on the outcome, we consider integrating CUTSAT++ as a theory solver for hierarchic superposition modulo linear integer arithmetic [5].

Finally, we point at some possible improvements of CUTSAT++. We see great potential in the development of constraint reduction techniques from (weak) Cooper elimination [3]. For practical applicability such reduction techniques might be crucial. The choice of the variable order $\prec$ has considerable impact on the efficiency of CUTSAT++. It might be possible to derive suitable orders via the analysis of the problem structure. We might benefit from results and experiences of research in quantifier elimination with variable elimination orders.

Acknowledgments. This research was supported in part by the German Transregional Collaborative Research Center SFB/TR 14 AVACS and by the ANR/DFG project STU 483/2-1 SMArT.

References

1. Barrett, C.W., Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Splitting on demand in SAT modulo theories. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 512–526. Springer, Heidelberg (2006)
2. Bromberger, M., Sturm, T., Weidenbach, C.: Linear integer arithmetic revisited. ArXiv e-prints, abs/1503.02948 (2015)
3. Cooper, D.C.: Theorem proving in arithmetic without multiplication. In: Meltzer, B., Michie, D. (eds.) 1971 Proceedings of the Seventh Annual Machine Intelligence Workshop, Edinburgh. Machine Intelligence, vol. 7, pp. 91–99. Edinburgh University Press (1972)
4. Dillig, I., Dillig, T., Aiken, A.: Cuts from proofs: a complete and practical technique for solving linear inequalities over integers. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 233–247. Springer, Heidelberg (2009)
5. Fietzke, A., Weidenbach, C.: Superposition as a decision procedure for timed automata. Math. Comput. Sci. **6**(4), 409–425 (2012)
6. Fischer, M.J., Rabin, M.: Super-exponential complexity of Presburger arithmetic. SIAM-AMS Proc. **7**, 27–41 (1974)
7. Griggio, A.: A practical approach to satisability modulo linear integer arithmetic. JSAT **8**(1/2), 1–27 (2012)
8. Jovanović, D., de Moura, L.: Cutting to the chase. J. Autom. Reasoning **51**(1), 79–108 (2013)
9. 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. Springer, Heidelberg (2010)
10. Lasaruk, A., Sturm, T.: Weak quantifier elimination for the full linear theory of the integers. A uniform generalization of Presburger arithmetic. Appl. Algebra Eng. Commun. Comput. **18**(6), 545–574 (2007)
11. Papadimitriou, C.H.: On the complexity of integer programming. J. ACM **28**(4), 765–768 (1981)
12. Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen. welchem die Addition als einzige Operation hervortritt. In: Comptes Rendus du premier congres de Mathematiciens des Pays Slaves, pp. 92–101. Warsaw, Poland (1929)
13. Weispfenning, V.: The complexity of almost linear diophantine problems. J. Symb. Comput. **10**(5), 395–403 (1990)