A complete and terminating approach to linear integer solving

Martin Bromberger, Thomas Sturm, Christoph Weidenbach

▶ **To cite this version:**

Martin Bromberger, Thomas Sturm, Christoph Weidenbach. A complete and terminating approach to linear integer solving. Journal of Symbolic Computation, 2020, 100, pp.102-136. 10.1016/j.jsc.2019.07.021 . hal-02397168

HAL Id: hal-02397168
https://inria.hal.science/hal-02397168

Submitted on 23 Nov 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Author's Version:
This paper has been published in the
Journal of Symbolic Computation Volume 100 (Elsevier).
The final authenticated version is available online at
https://doi.org/10.1016/j.jsc.2019.07.021
Last update: March 28th 2019

A Complete and Terminating Approach to Linear Integer Solving

Martin Bromberger

Max Planck Institute for Informatics and Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

Thomas Sturm

CNRS, Inria, and the University of Lorraine, Nancy, France

Christoph Weidenbach

Max Planck Institute for Informatics and Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

We consider feasibility of linear integer problems in the context of verification systems such as SMT solvers or theorem provers. Although satisfiability of linear integer problems is decidable, many state-of-the-art implementations neglect termination in favor of efficiency. We present the calculus CutSat++ that is sound, terminating, complete, and leaves enough space for model assumptions and simplification rules in order to be efficient in practice. CutSat++ combines model-driven reasoning and quantifier elimination to the feasibility of linear integer problems.

Keywords: Linear arithmetic, SMT, SAT, CDCL, Linear programming, Integer arithmetic

1. Introduction

Determining feasibility of linear integer problems is a classical task, which has been addressed and thoroughly investigated by at least two independent research lines: (i) integer and mixed real integer linear programming for optimization (see Jünger et al. (2010)), (ii) first-order quantifier elimination and decision procedures for Presburger Arithmetic and corresponding complexity results (see Berman (1977, 1980); Cooper (1972); Ferrante and Rackoff (1975, 1979); Fischer and Rabin (1974); Fürer (1982); von zur Gathen and Sieveking (1978); Grädel (1987); Lasaruk and Sturm (2007); Oppen (1978); Presburger (1929); Weispfenning (1990)). We are interested in feasibility of linear integer problems, which we call simply *problems*, in the context of the combination of theories, as they occur, e.g., in verification via SMT solving (see Nieuwenhuis et al. (2006)) or theorem proving (see Bachmair et al. (1994)). The basic problem considered in this context is satisfiability of a conjunction of purely existentially quantified

Email addresses: mbromber@mpi-inf.mpg.de (Martin Bromberger), thomas@thomas-sturm.de (Thomas Sturm), weidenb@mpi-inf.mpg.de (Christoph Weidenbach)

URL: http://people.mpi-inf.mpg.de/~mbromber/ (Martin Bromberger), http://science.thomas-sturm.de (Thomas Sturm), http://people.mpi-inf.mpg.de/~weidenb/ (Christoph Weidenbach)

inequalities. From this perspective, the above-mentioned research lines, which are based on optimization and quantifier elimination, address problems that are too general for this purpose: the former considers optimization aspects that go considerably beyond feasibility; the latter considers arbitrary Boolean combinations of constraints and quantifier alternations, which are more complicated than the problems that we consider.

Consequently, the SMT and theorem proving communities have developed several interesting approaches for their specific type of linear integer problems (see Barrett et al. (2006); Dillig et al. (2009); Griggio (2012)). 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[1]. Together with the known *a priori integer bounds* (see Papadimitriou (1981)) for a problem this branch-and-bound strategy yields a terminating and complete algorithm. However, the a priori bounds grow exponentially in the number of inequalities. Even for toy examples the bounds easily exceed standard 64-Bit integer representations of today's computers. As a result, these bounds do not cause termination in a reasonable amount of time and checking the bounds is expensive as it requires big integer representations. Hence, the a priori bounds are often not integrated in implementations (see Barrett et al. (2011); Cimatti et al. (2013); Dutertre (2014); de Moura and Bjørner (2008)). Furthermore, the a priori bounds depend only on syntactic properties of a problem, such as the largest occurring constant and the number of inequalities. Therefore, it seems that a more termination promising approach needs to explore the inner structure of a problem.

The most well known calculus of this type is the CDCL (Conflict Driven Clause Learning) (see Silva and Sakallah (1996); Bayardo Jr. and Schrag (1996); Moskewicz et al. (2001); Biere et al. (2009)) calculus for propositional satisfiability (SAT). It has changed SAT solving from a purely academic exercise to a standard verification technique in industry. The calculus starts by generating a partial model assumption by deciding (guessing) the truth value of propositional variables and propagating new truth values with respect to the already set values and the problem clauses. If the model assumption evolves into an overall model, satisfiability is shown. If it falsifies a clause, this clause together with the partial model assumption can be explored to derive a new learned clause via resolution. The learned clause is a logical consequence of the problem clauses, hence it explores the inner structure of the problem. The learned clause repairs the failed partial model assumption with respect to the model generating algorithm. The learned clause is new by construction (see Weidenbach (2015)) and constitutes progress on the basis of a well-founded ordering. The latter two properties both independently guarantee termination of the CDCL calculus. For a toy example consider the clauses $P \vee Q,\ \neg P \vee \neg Q,\ \neg P \vee Q$. A run of the CDCL calculus may decide P to be true and then propagate $\neg Q$ using the clause $\neg P \vee \neg Q$. The resulting model assumption $[\![P, \neg Q]\!]$ falsifies the clause $\neg P \vee Q$. Then a resolution step between this clause and the propagating clause $\neg P \vee \neg Q$ yields the clause $\neg P$. This clause repairs the model via backtracking into $[\![\neg P]\!]$ and after one propagation into $[\![\neg P, Q]\!]$, an overall model of the three clauses.

The idea of building explicit (partial) model assumptions guiding inferences is actually older than CDCL and goes back to the superposition calculus (see Bachmair and Ganzinger (1990)). It has meanwhile been transported to a variety of logics (see Caferra et al. (2004); Piskac et al. (2010); Alagi and Weidenbach (2015)) including arithmetic theories (see Jovanović and de Moura (2013); Jovanovic and de Moura (2012)). The CutSat calculus by Jovanović and de Moura

[1] For linear arithmetic problems over the rationals, such as the rational relaxation of an integer problem, the SMT and theorem proving communities already presented efficient approaches (see Dragan et al. (2013); Dutertre and de Moura (2006)).

(2013) is closest to our work and constitutes an important step towards a calculus that is both efficient and terminating on linear integer problems. Similar to CDCL and the other above mentioned calculi, termination does no longer rely on a priori bounds, but on the generation of new inequalities that guarantee termination on the basis of a well-founded ordering. If all variables of a problem are guarded, i.e., for every variable x there are inequalities $a \leq x \leq b$ for integer constants a, b, then CutSat terminates. If there are unguarded variables, then CutSat may diverge or get stuck (see Section 3). Our main contribution is the CutSat++ calculus that provably terminates on any linear integer problem.

Our interest in an algorithm for integer constraints originates from a combination of integer constraints with fragments of first-order logic. Such combinations are standard target logics in the context of software verification. Problems in the logic are solved by, e.g., superposition (see Baumgartner and Waldmann (2013); Fietzke and Weidenbach (2012)) or SMT (see Nieuwenhuis et al. (2006)) all build on top of a linear integer arithmetic decision procedure. In this context, variables in problems are typically unguarded, which means that these variables lack an upper or a lower bound and, therefore, there exist infinitely many potential assignments for the variables. However, the problems with unguarded variables are exactly those where termination guarantees for a calculus become difficult. An efficient decision procedure for the case of unguarded variables is, therefore, a prerequisite for an efficient combined procedure.

The basic idea of CutSat++ is to reduce a problem containing unguarded integer variables to a problem containing only guarded variables. Unguarded variables are not eliminated. Instead, they are explored by learning further inequalities on smaller guarded variables to the problem. A strict total ordering on all variables, where all unguarded variables are larger than all guarded variables, provides the scale. After adding sufficiently many inequalities, feasibility of the problem depends only on guarded variables. Then a CDCL style calculus with explicit (partial) model building tests for feasibility by employing exhaustive propagation.

The most sophisticated part is to "turn" an unguarded variable into a guarded variable. The variable elimination procedure by Cooper (see Cooper (1972)), does so by replacing an unguarded variable by a case distinction via disjunctions exploring lower bounds. The lower bounds are derived from the problems inequalities. The procedure is provably terminating at the price of potentially exponentially growing coefficients and an exponentially growing Boolean problem structure. Still, the idea behind Cooper's procedure is our starting point. We will present a variation of the procedure, which is called *weak Cooper elimination*, see Section 4. Weak Cooper elimination does not create a complicated boolean structure because it considers not only the lower bounds but also the upper bounds derivable from the problems inequalities. By combining all pairs of inequalities that determine a lower and an upper bound for some variable, we get a new formula that determines whether the strictest lower bound is smaller than or equal to the strictest upper bounds for that variable without complicating the boolean structure. The only additional drawback is the introduction of new guarded variables and divisibility constraints. However, guarded variables seem to be less harmful, in practice, and divisibility constraints are needed for integer quantifier elimination anyway.

Satisfiability of linear integer problems is NP-complete. All algorithms known today need exponential time on certain classes of input problems. Since Cooper elimination and weak Cooper elimination do not care about the concrete structure of a given problem, the exponential behavior is almost guaranteed. The idea of CutSat++ is, therefore, to simulate a lazy variation of weak Cooper elimination. It is lazy because it does not apply a complete variable elimination step at once, but only partially whenever false inequalities (leading to so called *conflicting cores*) occur over unguarded variables. Since CutSat++ uses a strategy that applies the rules according to a

total variable order, which is closely related to the elimination order of a quantifier elimination procedure, it is not even necessary to actually remove the unguarded variables. Instead, the conflict is blocked by learning constraints in smaller variables according to the variable order. This leaves space for repairing model assumptions and applying simplification rules in order for the calculus 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: In Section 2, we start by informally introducing GCutSat, the calculus that is the basis for both CutSat++ and CutSat. GCutSat describes a CDCL style calculus that is on its own already a sound, complete, and terminating calculus for all problems that contain only guarded variables. In Section 3, we explain the rules and strategies used by CutSat to handle unguarded variables and show that they do not suffice to guarantee termination. Our conclusion is that CutSat lacks, in addition to some refinements, a resolution case for diophantine conflicting cores. The basis for the exploration of this conflicting core is our new variant of Cooper elimination, called *weak cooper elimination* which we present in Section 4. We actually prove, Theorem 17, in Section 4, that any procedure that is based on weak Cooper elimination needs to consider diophantine conflicting cores for completeness. In Sections 5–6 we then explore weak Cooper elimination to define the inference rules of CutSat++ for the elimination of unguarded variables. The result is the sound, complete, and terminating CutSat++ calculus, see Section 6.3, Theorem 34, Theorem 39. Finally, we give conclusions and point at possible directions for future research. A short version of this paper was published in the proceedings of CADE-25 (see Bromberger et al. (2015)).

2. The Guarded Case

In this Section, we informally present GCutSat, a CDCL style calculus by Jovanović and de Moura (2013) that is a terminating and complete calculus for all guarded problems, i.e., problems with only guarded variables.[2] On unguarded problems, GCutSat can diverge (see also Example 2). To resolve this divergence, Jovanović and de Moura (2013) extended GCutSat by strong conflict resolution (see Section 3), which is an additional set of rules and strategies based on quantifier elimination techniques. The result is the calculus CutSat, which terminates on more unguarded problems than GCutSat, but still diverges on some classes of problems (see Examples 6-10).

Our calculus CutSat++ is an alternative extension of GCutSat towards unbounded problems. It extends GCutSat by unguarded conflict resolution (see Section 5), which is again an additional set of rules and strategies based on quantifier elimination techniques. But compared to CutSat and GCutSat, CutSat++ is complete and terminating on all problems.

This Section is organized as follows: first we start by fixing some notations and expressions, in Subsection 2.1. Then we define in Subsection 2.2 the states and models traversed by GCutSat and its extensions. In Subsection 2.3, we informally explain GCutSat. As part of this explanation, we summarize GCutSat's (i) model construction via decisions and propagations, (ii) CDCL-like conflict resolution based on Fourier-Motzkin elimination, (iii) reasonable execution strategy, and (iv) slack-variable introduction that prevents stuck states. For formal details on GCutSat, we refer to the original presentation by Jovanović and de Moura (2013), Section 4.

[2] Although Jovanović and de Moura (2013) created GCutSat, they did not name it. We chose GCutSat as its name to avoid confusion between it and the whole CutSat calculus.

2.1. Notations

Before we explain the details of GCutSat, we have to fix some notations and terms: In this paper, 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. For true we denote $\top$ and for false we denote $\bot$. 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 the coefficients in those constraints with $\text{coeff}(I, x_i) = a_i \in \mathbb{Z}$. The semantic of an inequality $p \leq 0$ is very well-known, i.e., p is less than or equal to 0. The semantic of a divisibility constrain $d \mid p$ is simply p is divisible by d. This also means that $d \mid cx + s \equiv d \mid -cx - s$. We will use this fact to make the following assumption for the remainder of the paper: if we focus on only one variable x in a divisibility constraint, i.e., write it as $d \mid cx + s \in C$, then the coefficient c of x can be assumed to be positive, i.e., $c > 0$. Note that the linear polynomial s may still contain more variables and those might have negative coefficients.

The satisfiability of a problem C depends on the values its variables can take without violating its constraints. We write $\text{vars}(C)$ for the set consisting of the variables occurring in C. A problem C has a real solution if there exists an assignment $\beta : \text{vars}(C) \to \mathbb{R}$ that satisfies all constraints in C. A problem C has an integer solution if there exists an assignment $\nu : \text{vars}(C) \to \mathbb{Z}$ that satisfies all constraints in C. A problem C is satisfiable if there exists an integer solution ν for C. Note that a real assignment is not necessarily an integer assignment. Therefore, a real solution does not guarantee that C is actually satisfiable. However, if C has no real solution, then C has no integer solution and is also not satisfiable.

A variable x is *guarded* in a problem C if C contains both a constraint of the form $x - u \leq 0$ and a constraint of the form $-x + l \leq 0$. Otherwise, x is *unguarded* in C. Note that guarded variables are *bounded* as defined by Jovanović and de Moura (2013) but not vice versa. A constraint is *guarded* if it contains only guarded variables. Otherwise, it is *unguarded*.

2.2. States and Models

GCutSat decides satisfiability of a (guarded) problem C. It either ends in the state *unsat* or in a state $\langle \nu, \text{sat} \rangle$, where ν is a satisfiable assignment for C. In order to reach one of those two final states, the calculus 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 by Jovanović and de Moura (2013):

$$
\begin{aligned}
\text{lower}(c, M) &= c; \\
\text{lower}(a \cdot x, M) &= a \cdot \text{lower}(x, M), && \text{if } a > 0; \\
\text{lower}(a \cdot x, M) &= a \cdot \text{upper}(x, M), && \text{if } a < 0; \\
\text{lower}(p + q, M) &= \text{lower}(p, M) + \text{lower}(q, M); \\
\text{upper}(c, M) &= c; \\
\text{upper}(a \cdot x, M) &= a \cdot \text{upper}(x, M), && \text{if } a > 0; \\
\text{upper}(a \cdot x, M) &= a \cdot \text{lower}(x, M), && \text{if } a < 0; \\
\text{upper}(p + q, M) &= \text{upper}(p, M) + \text{upper}(q, M).
\end{aligned}
$$

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 $\nu[M] : \text{vars}(C) \to \mathbb{Z}$ as the assignment that assigns to every variable x the value $\text{lower}(x, M)$. By $\text{val}(p, M) := \text{lower}(p, M)$, we denote the value assigned to a fixed polynomial p. This also means that $\text{val}(p, M)$ is defined only if all variables occurring in p are fixed in M.

The GCutSat calculus is defined in the form of a transition system that traverses states via rules. The rules are applied in a non-deterministic way. This type of presentation supports proofs by induction on the length of rule applications, where the rules can be considered independently. A state in GCutSat (as well as CutSat and CutSat++) is either one of the two *final states* $\langle \nu, \text{sat} \rangle$, *unsat*, or of the form $S = \langle M, C, I \rangle$, where M is the current partial model, C is the current set of constraints, and I is either $\top$ or an inequality entailed by C ($C \vdash_{\mathbb{Z}} I$) that constitutes a conflict, i.e., it evaluates to false under the partial model M. In case I is just $\top$, we will often abbreviate $\langle M, C, \top \rangle$ as $S = \langle M, C \rangle$ and call it a search state. Otherwise, we call $\langle M, C, I \rangle$ a conflict state.[3] The *initial-state* for a problem C is the search state $\langle [\![]\!], C \rangle$. Therefore, the initial-states of GCutSat (as well as CutSat and CutSat++) are all search states $\langle [\![]\!], C \rangle$, where C is a constraint system.

GCutSat constructs all states $\langle M, C', I \rangle$ after the initial state $\langle [\![]\!], C \rangle$ in such a way that (i) C' is equisatisfiable to C and that (ii) any satisfiable assignment ν for the current set of constraints C' is also a satisfiable assignment for the original set of constraints C. Moreover, any partial model M generated by GCutSat (i) stays *consistent*, i.e., $\text{lower}(x, M) \leq \text{upper}(x, M)$ for all variables $x \in \mathit{vars}(C)$, and (ii) *improves*, i.e., $\text{lower}(x, M') < \text{lower}(x, M)$ if $M = [\![M', x \geq b]\!]$ and $\text{upper}(x, M') > \text{upper}(x, M)$ if $M = [\![M', x \leq b]\!]$. Finally, no state generated by GCutSat is *stuck*, i.e., a state generated by GCutSat is either a final state or a rule is applicable. All of the above properties hold also for CutSat and CutSat++.

2.3. GCutSat

GCutSat is the calculus consisting of the rules Propagate, Propagate-Div, Decide, Conflict, Conflict-Div, Resolve, Skip-Decision, Backjump, Sat, Unsat-Div, Unsat, Learn, Forget, and Slack-Intro (see Jovanović and de Moura (2013)), which are executed according to a reasonable strategy (see Definition 1). As already noted, for formal details on GCutSat we refer to the original presentation by Jovanović and de Moura (2013), Section 4. Here, we only introduce the most important mechanics in order to show non-termination on certain classes of unguarded problems and in this way motivate CutSat++.

GCutSat builds its partial model by Decide (guess/assign a value) and Propagate (propagate bounds through inequalities to obtain new bounds) rules similar to CDCL. Via applications of the rule Decide, GCutSat adds *decided bounds* $x \leq b$ or $x \geq b$, also called *decisions*, to the sequence M in state S. A decided bound assigns a variable x to the strictest lower or upper bound of x in M. This means every decisions fixes one variable to a value. Via applications of the propagation rules (Propagate & Propagate-Div), GCutSat uses the bounds in the model assumption to propagate additional bounds. For example, the inequality $2x - y \leq 0$ together with the bound $y \leq 1$ propagates $2x \leq 1$ or $x \leq \frac{1}{2}$. Since we are looking for an integer value for x the actual bound becomes $x \leq 0$, which is also the bound GCutSat would propagate. The bounds added by the propagation rules are called *propagated bounds* $x \geq_I b$ or $x \leq_I b$, where the function

[3] Jovanović and de Moura (2013) use a different notation for conflict states. The conflict state in our notation $\langle M, C, I \rangle$ is equivalent to the conflict state in their notation $\langle M, C \rangle \vdash I$ and vice versa.

bound(J, x, M) defines the strictest bound value b that can be computed for constraint J under the partial model M, and the function tight(J, x, M) or the function div-derive(J, x, M) compute a justification I, i.e., a simplified version of the propagating constraint J that is needed for conflict resolution and, therefore, annotated to the propagated bound.[4] The most important properties of the bounds added to the model are: (i) any propagated or decided bound γ *improves* the model $M = [\![M', \gamma]\!]$, i.e., lower$(x, M') <$ lower(x, M) if $M = [\![M', x \geq b]\!]$ and upper$(x, M') >$ upper(x, M) if $M = [\![M', x \leq b]\!]$, (ii) any propagated or decided bound γ keeps the model $M = [\![M', \gamma]\!]$ *consistent*, i.e., lower$(x, M) \leq$ upper(x, M) for all variables $x \in vars(C)$, and (iii) any propagated bound γ is *implied* by the current model and constraint set, i.e., $M' \cup C \vdash \gamma$.

Since decided bounds are not necessarily implied by the current model and constraint set, a decision may introduce a conflict after some further rule applications, although the current constraint set is satisfiable. In order to determine and undo the responsible decisions, GCutSat also has rules (Conflict, Conflict-Div, Resolve, Skip-Decision, Backjump, Learn, Forget) that work similar to a CDCL-like conflict resolution (see Nieuwenhuis et al. (2006)). For comparison, the original CDCL-like conflict resolution works as follows: the CDCL calculus computes from a falsified clause via resolution steps a new clause (i) that is either empty and, thereby, proves that the original clause set was unsatisfiable or (ii) that can be used to repair the partial model assumption by backjumping to the partial model before the responsible decision and by learning and propagating the clause instead.

For inequalities and divisibility constraints, we can do something very similar. The only difference is the resolution step, which is based on Fourier-Motzkin elimination instead of Boolean resolution. Let $-ax + p \leq 0$ and $bx + q \leq 0$ be two constraints over the integers with $a, b > 0$. Then they also imply the constraint $b \cdot (-ax + p) + a \cdot (bc + q) \leq 0 \equiv bp + aq \leq 0$ that does not contain the variable x. For example, we can combine $1 - x + y \leq 0$ and $x - y \leq 0$ through linear combination so the new constraint $1 \cdot (1 - x + y) + 1 \cdot (x - y) \leq 0 \equiv 1 \leq 0$ directly proves unsatisfiability of our problem. However, resolving conflicts is typically more challenging for linear integer constraints. To be more precise, the constraints used for Fourier-Motzkin based resolution must guarantee coefficients ± 1 for the propagated variables or the CDCL-like conflict resolution is not always capable of finding/undoing the responsible decision. This is also the reason why GCutSat does not annotate propagated bounds with the actual constraints J used for propagation but with the justifications I computed either by the function tight(J, x, M) or div-derive(J, x, M) (see also Jovanović and de Moura (2013)). Since these properties (summarized as tightly-propagating) are not directly relevant to the extensions of GCutSat that are the actual focus of this paper, we neglect to discuss them in more detail here but refer any interested readers to the original presentation by Jovanović and de Moura (2013).

GCutSat ends a run when it has determined whether the original constraint set has an integer solution or not. Formally, GCutSat does so with the rules Sat, Unsat-Div, and Unsat. Sat is called as soon as GCutSat has found a complete model M, i.e., a model where all variables are fixed, that also satisfies all constraints in our current constraint set C. The assignment $\nu[M]$ in the final state $\langle \upsilon[M], \text{sat} \rangle$ is then a satisfiable assignment for the current and the original constraint set.

The rules Unsat-Div and Unsat are applicable as soon as GCutSat derives a trivially unsatisfiable constraint, i.e., either a constant inequality $b \leq 0$ that is unsatisfiable or a divisibility

[4]The functions bound(J, x, M), tight(J, x, M), and div-derive(J, x, M) are also defined in Jovanović and de Moura (2013).

Slack-Intro

$$\langle M, C\rangle \quad \Longrightarrow_{\text{CS++}} \quad \langle M, C \cup C_s\rangle \qquad \textbf{if} \begin{cases} x \text{ is stuck,} \\ \text{all other unfixed variables in } \langle M, C\rangle \text{ are also stuck,} \\ x_S \text{ is the slack-variable,} \\ C_s = \{-x_S \leq 0, x - x_S \leq 0, -x - x_S \leq 0\} \end{cases}$$

Figure 1: The Slack-Intro rule of GCutSat

constraint $d \mid a_1x_1 + \ldots + a_nx_n + c$ that can never be true because $\gcd(d, a_1, \ldots, a_n) \nmid c$. The unsatisfiability of the original constraint set is then marked by the end state unsat.

If we apply the rules according to the following strategy, then GCutSat is sound, complete and terminates for *guarded problems*, i.e., those input problems C containing only guarded variables.

Definition 1 (Reasonable Strategy (see Jovanović and de Moura (2013)))**.** *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.*

If the problem is *unguarded*, i.e., the input problem C contains at least one unguarded variable, then GCutSat with a reasonable strategy can diverge or get *stuck*, i.e., reach a state that is not a final state but where no rule is applicable. To prevent the latter, GCutSat contains another rule called Slack-Intro (Fig. 1). (In contrast to the other rules, we explicitly define Slack-Intro because our version is slightly simpler than the one by Jovanović and de Moura (2013).)

GCutSat without Slack-Intro encounters stuck states because it is sometimes unable to propagate and, thereby, also unable to decide any bounds for a variable x. In this case, we call the variable x stuck. More formally, a variable x is called *stuck* in state $S = \langle M, C\rangle$ if M contains no bounds for x and we can neither use Propagate or Propagate-Div to add a bound for x. Guarded variables x are never stuck because they have by definition constraints of the form $\pm x - b \leq 0 \in C$, which GCutSat is always able to propagate. The easiest example where stuck variables result in a stuck state is the state $\langle [\![]\!], \{x - y \leq 0\}\rangle$. Here both variables are stuck, so we can never construct a model. In case all unfixed variables x are stuck, Slack-Intro allows us to add new constraints to the problem that make one of the variables unstuck. To this end, Slack-Intro exploits the following fact: an input problem C is satisfiable if it is satisfiable inside an a priori fixed finite interval for all variables, i.e., if there exists an $x_S \geq 0$ such that $C \cup \{-x_S \leq x \leq x_S\}$ is also satisfiable. Slack-Intro simulates this fact in GCutSat by adding a new variable x_S and some constraints $\{-x_S \leq 0, x - x_S \leq 0, -x - x_S \leq 0\}$ to C such that the bounds $-x_S \leq x \leq x_S$ can be propagated after x_S is propagated and decided. This also means that GCutSat expands the bounds for all previously stuck variables incrementally and symmetrically from the integer constant zero towards $\pm\infty$. Slack-Intro alone actually prevents all stuck states for GCutSat.

3. Divergence of CUTSAT

An easy example for divergence of GCutSat—already stated by Jovanović and de Moura (2013)—relies on cyclic dependencies during propagation:

Example 2. *Let $S_i = \langle M_i, C\rangle$ ($i \in \mathbb{N}$) be a series of states defined by:*

$$\begin{aligned} C &:= \{\underbrace{-x \le 0}_{I_x}, \underbrace{-y \le 0}_{I_y}, \underbrace{-z \le 0}_{I_z}, \underbrace{1 - x + y \le 0}_{J_1}, \underbrace{x - y - z \le 0}_{J_2}\} \\ M_0 &:= [\![x \ge_{I_x} 0, y \ge_{I_y} 0, z \ge_{I_z} 0, z \le 0]\!] \\ M_{i+1} &:= [\![M_i, x \ge_{J_1} i+1, y \ge_{J_2} i+1]\!] \end{aligned}$$

GCutSat *with a reasonable strategy has diverging runs starting in state $S'_0 = \langle [\![]\!], C\rangle$. Let* GCutSat *traverse the states S'_0, S_0, S_1, S_2, ... in the following fashion:* GCutSat *reaches state $S_0 := \langle M_0, C\rangle$ from state S'_0 after propagating the constraints I_x, I_y, I_z, and fixing z with a decided bound.* GCutSat *reaches state S_{i+1} from state S_i after:*

- *applying Propagate to J_1 to propagate $\gamma^x_{i+1} := x \ge_{J_1} i+1$; $M'_i := [\![M_i, \gamma^x_{i+1}]\!]$ and $S'_i := \langle M'_i, C\rangle$*
- *applying Propagate to J_2 to propagate $\gamma^y_{i+1} := y \ge_{J_2} i+1$; $M_{i+1} := [\![M'_i, \gamma^y_{i+1}]\!]$ and $S_{i+1} := \langle M_{i+1}, C\rangle$*

To summarize, this example shows how we can exploit the cyclic dependence between x and y to increase the lower bounds for x and y to an arbitrarily large value $i \in \mathbb{N}$.

To prevent this type of divergence, Jovanović and de Moura suggest to restrict the bounds propagated by the rules Propagate and Propagate-Div to only those bounds that are δ-relevant.

Definition 3 (δ-relevant (see Jovanović and de Moura (2013)))**.** *Let* $\mathrm{nb}(x, M)$ *be the number of bounds for x in M. Let $\delta > 0$ and $\mathrm{nb}_{\max} \in \mathbb{N}$ be two parameters that are fixed at the start of the* GCutSat *run. Then the new lower bound $x \ge_J b$ is δ-relevant in the state $\langle M, C\rangle$ if:*

- $\mathrm{lower}(x, M) = -\infty$, *or*
- $\mathrm{upper}(x, M) \neq \infty$, *or*
- $\mathrm{lower}(x, M) + \delta|\mathrm{lower}(x, M)| < b$ *and* $\mathrm{nb}(x, M) < \mathrm{nb}_{\max}$.

Likewise, the new upper bound $x \le_J b$ is δ-relevant in the state $\langle M, C\rangle$ if:

- $\mathrm{upper}(x, M) = \infty$, *or*
- $\mathrm{lower}(x, M) \neq -\infty$, *or*
- $\mathrm{upper}(x, M) + \delta|\mathrm{upper}(x, M)| > b$ *and* $\mathrm{nb}(x, M) < \mathrm{nb}_{\max}$.

If we now restrict the rules Propagate and Propagate-Div as suggested, then δ-relevance gives us the following guarantees: The first case of δ-relevance guarantees that GCutSat can still propagate at least one bound for every variable. The second case of δ-relevance guarantees that a bounded variable (with $x - u_x \le 0 \in C$ and $-x + l_x \le 0 \in C$) can be propagated without restrictions because its guards $l_x \le x \le u_x$ already guarantee that there are at most $|u_x - l_x + 1|$ propagations for x. The last case of δ-relevance guarantees that an unbounded variable is at most propagated $\mathrm{nb}_{\max}$ times and, thus, prevents propagation divergence. The additional conditions $\mathrm{lower}(x, M) + \delta|\mathrm{lower}(x, M)| < b$ and $\mathrm{upper}(x, M) + \delta|\mathrm{upper}(x, M)| > b$ prevent GCutSat from wasting its limited amount of propagations on those that change the bounds only slightly. We also do not introduce any stuck states with our restriction to δ-relevance because any bound $x \ge_J b$

Resolve-Cooper

$$\langle M, C\rangle \implies_{\text{CS++}} \langle M', C \cup R\rangle \quad \textbf{if} \begin{cases} (x, C') \text{ is a conflicting core,} \\ x \text{ is unguarded,} \\ x \text{ is the minimal conflicting variable,} \\ (R_k, R_c) = \text{cooper}(x, C'), \\ R = R_k \cup R_c \end{cases}$$

Solve-Div

$$\langle M, C\rangle \implies_{\text{CS++}} \langle M, C'\rangle \quad \textbf{if} \begin{cases} \text{divisibility constraints } I_1, I_2 \in C, \\ (I_1', I_2') = \text{div-solve}(x, I_1, I_2), \\ C' = C \setminus \{I_1, I_2\} \cup \{I_1', I_2'\} \end{cases}$$

div-solve$(x, d_1 \mid a_1 x + p_1, d_2 \mid a_2 x + p_2) = (d_1 d_2 \mid dx + c_1 d_2 p_1 + c_2 d_1 p_2, d \mid -a_1 p_2 + a_2 p_1)$, where $d = \gcd(a_1 d_2, a_2 d_1)$, and c_1 and c_2 are integers such that $c_1 a_1 d_2 + c_2 a_2 d_1 = d$ (see Cooper (1972); Jovanović and de Moura (2013)).

Figure 2: The strong conflict resolution rules by Jovanović and de Moura (2013)

that the third case of δ-relevance prevents from being propagated can also be introduced to the bound sequence M if we first fix x with Decide, then apply Conflict(-Div) to J, and finally add the bound $x \geq_J b$ to M with Backjump.

However, this simulation of Propagate(-Div) via Decide and conflict resolution also means that divergence is not eliminated but only shifted from propagation to conflict resolution. We can also use Example 2 for this type of divergence, however, this time we simulate propagation as described above via conflict resolution. Instead of applying Propagate between the states S_i and S_{i+1}, we fix one variable at a time with a decided bound. If we fix x to lower(x, M_i) with a decided bound, we directly detect a conflict in J_1. We enter conflict resolution by applying Conflict to J_1, which can be exited directly with the Backjump rule (see Jovanović and de Moura (2013)). This removes the decided bound on x and adds instead $x \geq_{J_1} i + 1$ on top of the bound sequence. Hence, we reach the intermediate state S_i' without applying the rule Propagate. Analogously, we do the same for y with J_2 and reach the state S_{i+1} without applying the rule Propagate.

The best way to handle this type of divergence is to forbid standard conflict resolution—i.e., the rules Conflict, Conflict-Div, Resolve, Backjump, Skip-Decision, Unsat, and Learn—from handling constraints containing unguarded variables. As an alternative to normal conflict resolution, Jovanović and de Moura (2013) suggested a secondary conflict analysis called strong conflict resolution. Strong conflict resolution is also the predecessor of the unguarded conflict resolution we are going to present in Section 5. Strong conflict resolution is based on the fact that we can transform any unguarded problem into a guarded one if we eliminate all unguarded variables with a quantifier elimination procedure. Instead of eliminating all unguarded variables before we apply GCutSat, strong conflict resolution introduces two new rules (see Figure 2) that extend the GCutSat calculus by a second type of conflict resolution, which applies quantifier elimination in-between the applications of the original GCutSat rules. We call GCutSat with this extension CutSat. The newly added rules do not apply the complete quantifier elimination algorithm, nor do they apply it at all possible instances, but only on minimal conflicts containing unguarded variables. These minimal conflicts are also called conflicting cores.

Definition 4 (Conflicting Cores (see Jovanović and de Moura (2013)))**.** *Let $S = \langle M, C\rangle$ be a*

state, $C' \subseteq C$, x a variable in C', $a, b > 0$, 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 two forms:*
(1) $C' = \{-ax + p \leq 0, bx - q \leq 0\}$ *and* $\text{bound}(-ax + p \leq 0, x, M) > \text{bound}(bx - q \leq 0, x, M)$*, i.e., the lower bound from* $-ax + p \leq 0$ *contradicts the upper bound from* $bx - q \leq 0$*; 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* $b_l = \text{bound}(-ax + p \leq 0, x, M)$*,* $b_u = \text{bound}(bx - q \leq 0, x, M)$*,* $b_l \leq b_u$*, and for all* $b_d \in [b_l, b_u]$ *we have* $d \nmid cb_d + \text{lower}(s, M)$*, i.e., there exists no value for* x *within the bounds defined by the two inequalities such that the divisibility constraint becomes satisfiable; in this case,* (x, C') *is called a* divisibility conflicting core *and its* strong resolvent *is* $(\{-k \leq 0, k - m \leq 0\}, \{bp - aq + bk \leq 0, a \mid k + p, ad \mid cp + as + ck\})$
In both cases, k *is a fresh variable and* $m = \text{lcm}\left(a, \frac{ad}{\gcd(ad,c)}\right) - 1$.

We refer to the respective strong resolvents for a conflicting core (x, C') by the function $\text{cooper}(x, C')$, which returns a strong resolvent (R_k, R_c) as defined above, Definition 4. These strong resolvents (R_k, R_c) are simply the constraints that we get after we eliminate $\exists x \in \mathbb{Z}$ from $\exists x \in \mathbb{Z}.\ C'$ with Cooper's quantifier elimination procedure (see Cooper (1972)). Note that the newly introduced variable k is guarded by the constraints in R_k. In Section 5, we will extend the definition of conflicting cores by a third type of core called a diophantine conflicting core. In the final example at the end of this section, we will also show why this core is necessary to guarantee termination.

Besides conflicting cores and strong resolvents, strong conflict resolution also relies on a total order $\prec$ over all variables such that $y \prec x$ for all guarded variables y and unguarded variables x. In relation to Cooper's quantifier elimination, the order $\prec$ 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* ($x = \text{top}(I)$). If there is a conflicting core (x, C') in some state S, then x is called a *conflicting variable*. The conflicting variable is *minimal* in the state S if there exists no conflicting variable y in S such that $y \prec x$.

Instead of actually eliminating a conflicting variable or the conflicting core (x, C'), the rule Resolve-Cooper (see Figure 2) only adds the constraints $R_k \cup R_c$ from the strong resolvent (R_k, R_c) to the problem. By doing so, Resolve-Cooper is supposed to guarantee that the old conflicting core (x, C') is always ignored in favor of a "smaller" conflicting core. The following strategy guarantees that strong conflict resolution and the standard conflict resolution interact as little as possible. As a consequence, the strategy is also supposed to guarantee termination for the whole calculus:

Definition 5 (Two-layered Strategy)**.** *We say a strategy is* two-layered *if*

- *it is reasonable (Definition 1);*
- *the Propagate and Propagate-Div rules are limited to δ-relevant bounds;*
- *the Forget rule is never used to eliminate resolvents introduced by Resolve-Cooper;*
- *it only applies the Conflict and Conflict-Div rules if Resolve-Cooper is not applicable.*

However, we are now going to discuss four examples where CutSat diverges despite strong conflict resolution. The first one shows that CutSat can apply Conflict and Conflict-Div infinitely often to constraints containing unguarded variables.

Example 6. *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 has diverging runs starting in state $S'_0 = \langle [\![]\!], C\rangle$. Let* CutSat *traverse the states S'_0, S_0, S_1, S_2, ... in the following fashion:* CutSat *reaches state S_0 from state S'_0 after propagating the constraints I_x, I_y, I_{z1}, and I_{z2}.* CutSat *reaches state S_{i+1} from state S_i after:*

- *fixing x to i with the decided bound $\gamma^x_d := x \leq i$; $M^1_i := [\![M_i, \gamma^x_d]\!]$ and $S^1_i := \langle M^1_i, C\rangle$*
- *applying Conflict to the constraint J_1 because* $\text{lower}(1 - x + y, M^1_i) > 0$*; $M^2_i := M^1_i$ and $S^2_i := \langle M^2_i, C, J_1\rangle$*
- *undoing the decided bound γ^x_d by applying Backjump. This results in the exchange of γ^x_d with the bound $\gamma^x = x \geq_{J_1} i + 1$; $M^3_i := [\![M_i, \gamma^x]\!]$ and $S^3_i := \langle M^3_i, C\rangle$*
- *fixing y to i with the decided bound $\gamma^y_d := y \leq i$; $M^4_i := [\![M^3_i, \gamma^y_d]\!]$ and $S^4_i := \langle M^4_i, C\rangle$*
- *applying Conflict to the constraint J_2 because* $\text{lower}(x - y, M^4_i) > 0$*; $M^5_i := M^4_i$ and $S^5_i := \langle M^5_i, C, J_1\rangle$*
- *undoing the decided bound γ^y_d by applying Backjump. This results in the exchange of γ^y_d with the bound $\gamma^y = y \geq_{J_2} i + 1$; $M^6_i := [\![M^3_i, \gamma^y]\!]$ and $S_{i+1} = S^6_i := \langle M^6_i, C\rangle$*

Notice that $(z, \{I_{z1}, I_{z3}\})$ is a conflicting core in the states S_i, S^1_i, and S^4_i, and, therefore, the variable z is the minimal conflicting variable in those states. Since I_{z1} and I_{z2} bound z, the conflicting core is also guarded. Therefore, Resolve-Cooper as defined by Jovanović and de Moura (2013) is not applicable, which in turn implies that Conflict is applicable. This means the two-layered strategy was not strict enough to prevent standard conflict resolution from being applied to unguarded conflicts.

A straightforward fix to Example 6 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.

Example 7. *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 $S_0 \Longrightarrow_{CS++} S_1 \Longrightarrow_{CS++} S_2 \Longrightarrow_{CS++} \ldots$. 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. If we were to simplify the constraints in C_n, then we would even see that the constraints C_n for an even n are always just $\{4 \mid 2x + 2y, 2 \mid x + z\}$ and the constraints C_n for an odd n are always just $\{4 \mid 2x + 2y, 2 \mid y - z\}$. This means that we are cycling between the same two sets of constraints.*

A straightforward fix to Example 7 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 infinitely often. Example 8 differs from Example 6 in that the conflicting core contains also unguarded variables.

Example 8. *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 has diverging runs starting in state $S'_0 = \langle [\![]\!], C\rangle$. For instance, let* CutSat *traverse the states S'_0, S_0, S_1, S_2, ... in the following fashion:* CutSat *reaches state S_0 from state S'_0 after propagating the constraints I_x, I_y, I_{z2} and I_{z2}.* CutSat *reaches state S_{i+1} from state S_i after:*

- *fixing x to i and y to i with decided bounds $\gamma^x_d := x \leq i$ and $\gamma^y_d := y \leq i$; $M^1_i := [\![M_i, \gamma^x_d, \gamma^y_d]\!]$ and $S^1_i := \langle M^1_i, C\rangle$*
- *applying Conflict to the constraint J_1 because* $\text{lower}(1 - x + y + z, M^1_i) > 0$*; $M^2_i := M^1_i$ and $S^2_i := \langle M^2_i, C, J_1\rangle$*
- *undoing the decided bounds γ^y_d and γ^x_d by applying first Skip-Decision and then Backjump. The result is the sequence $M^3_i := [\![M_i, \gamma^x]\!]$ and the state $S^3_i := \langle M^3_i, C\rangle$, where $\gamma^x = x \geq_{J_1} i + 1$;*
- *fixing y to i and x to i + 1 with decided bounds $\gamma^y_d := y \leq i$ and $\gamma^x_d := y \leq i + 1$; $M^4_i := [\![M^3_i, \gamma^y_d, \gamma^x_d]\!]$ and $S^4_i := \langle M^4_i, C\rangle$*
- *applying Conflict to the constraint J_2 because* $\text{lower}(x - y - z, M^4_i) > 0$*; $M^5_i := M^4_i$ and $S^5_i := \langle M^5_i, C, J_1\rangle$*
- *undoing the decided bounds γ^x_d and γ^y_d by applying first Skip-Decision and then Backjump. The result is the sequence $M^6_i := [\![M^3_i, \gamma^y]\!]$ and the state $S_{i+1} = S^6_i := \langle M^6_i, C\rangle$, where $\gamma^y = y \geq_{J_2} i + 1$.*

Notice that the conflicting core $\{J_1, J_2\}$ is bounded after we fix x and y with Decide to their current respective lower bounds. This in turn admits the application of Conflict. Therefore, the two-layered strategy was again not strict enough to prevent standard conflict resolution from being applied to unguarded conflicts.

Applying the fix suggested for Example 6 to Example 8 results in a stuck 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.[5]

We could also prevent Examples 6 and 8 if we were to fix the decision order to the variable order. However, for the following counterexample, fixing the decision order is not enough:

Example 9. *Let*

$$\begin{aligned} C := \{ \quad & \underbrace{-u \le 0}_{I_{u1}}, \underbrace{u \le 0}_{I_{u2}}, \underbrace{-v \le 0}_{I_{v1}}, \underbrace{v \le 0}_{I_{v2}}, \underbrace{-w \le 0}_{I_{w}}, \underbrace{-w + x + u \le 0}_{I_{wx1}}, \underbrace{-w - x - u \le 0}_{I_{wx2}}, \\ & \underbrace{-w + y + v \le 0}_{I_{wy1}}, \underbrace{-w - y - v \le 0}_{I_{wy2}}, \underbrace{2 + 3x - 4y + 3u - 4v \le 0}_{J_1}, \\ & \underbrace{-1 + 3x - 4y + 3u - 4v \le 0}_{J_2}, \underbrace{-1 - 3x + 2y - 3u + 2v \le 0}_{J_3} \} \end{aligned}$$

be a problem. Let the variable order be given by $u \prec v \prec w \prec x \prec y$. CutSat *has a divergent run starting in state $S'_0 = \langle [\![]\!], C \rangle$ if it follows the following strategy: Let z be the smallest unfixed variable.*

- *Propagate (if possible) the strictest upper bound for z,*
- *if J_1 is a conflict, then fix all unfixed variables with the rule Decide to their current upper bound, and start the standard conflict analysis with the rule Conflict,*
- *otherwise propagate the strictest lower bound for z and fix z with a decision to the current lower bound*

Notice that we prevent the application of Resolve-Cooper by first fixing all variables and then shadowing x and y with the bounded variables u and v, respectively. If we change the definition of conflicting cores, as proposed for Example 8, then the divergent behaviour described in Example 9 is not possible. Thus, fixing the decision order to the variable order is no real alternative to the fixes proposed before.

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 applicable only 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 are too strong and therefore lead to stuck states.

Example 10. *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*

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

[5]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.

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 stuck 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, Definition 4 mentions only interval or divisibility conflicting cores and the state* S *contains neither. 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.*

4. Weak Cooper Elimination

In order to fix the stuck state of Example 10 in the previous section, we are going to introduce in Section 5 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 (see Cooper (1972)). In particular, this section explains why diophantine conflicting cores are necessary and why we need no other types of conflicting cores (besides interval, divisibility, and diophantine conflicting cores).

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

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

where $a > 0$, $m = \operatorname{lcm}\{d \in \mathbb{Z} : (d \mid a_j x_j + p_j) \in C\}$, $C_{-\infty} = \bot$ if there exists a constraint of the form $-ax + p \leq 0 \in C$, and, otherwise, $C_{-\infty} = \{(d \mid ax + p) \in C\}$. Although Cooper elimination gets rid of the variable x, the elimination also comes at a price. One application of Cooper elimination results in a disjunction of quadratically many problems out of a single problem. Moreover, we have to somehow "remove" the fractions $\frac{p+k}{a}$ used to replace x because division is not part of the linear arithmetic language. We achieve this by multiplying each constraint with a. However, this "removal" of the fractions causes a worst-case quadratic increase in the absolute size of the coefficients. The number of disjunctions and the size of coefficients have even a worst-case exponential increase if we look at several iterations of Cooper elimination.

Now that we have formally defined Cooper elimination and stated its structural properties, we will explain in an intuitive way why Cooper elimination actually works. To this end, let us look at the result of applying Cooper elimination to a small example:

Example 11. *Let*

$$C' := \{\underbrace{2 \mid x+y}_{J_1}, \underbrace{2 \mid x+z}_{J_2}, \underbrace{-3x+y \leq 0}_{J_3}, \underbrace{-2x+2y-2 \leq 0}_{J_4}, \underbrace{x-4z \leq 0}_{J_5}\}$$

be the initial problem. The result of applying Cooper elimination is:

$$[\exists x.C'] \equiv \left(\bigvee_{0\leq k<1} C_{-\infty}\right) \vee \left(\bigvee_{0\leq k<3} C_{J_3}\right) \vee \left(\bigvee_{0\leq k<2} C_{J_4}\right),$$

where

$$\begin{array}{lcl} C_{-\infty} & := & \bot \\ C_{J_3} & := & \{6 \mid k+4y, 6 \mid k+y+3z, -k \le 0, -2k+4y-6 \le 0, \\ & & \quad k+y-12z \le 0, 2 \mid k+2y-2\}, \\ C_{J_4} & := & \{4 \mid k+4y-2, 4 \mid k+2y+2z-2, -3k-4y+6 \le 0, -k \le 0, \\ & & \quad k+2y-8z-2 \le 0, 3 \mid k+y\}. \end{array}$$

A linear arithmetic problem has an integer solution if and only if the strictest lower bound for x is an integer solution for the problem. In Cooper elimination, we use this fact and do a case distinction over the strictest lower bound for x via a disjunction. First, we assume that there exists no lower bound for x. We express this case with the subformula $C_{-\infty}$. Since both J_3 and J_4 define a lower bound for x ($x \ge \lceil \frac{y}{3} \rceil$ and $x \ge \lceil \frac{2y-2}{2} \rceil$, respectively), $C_{-\infty}$ simplifies to $\bot$. Next, we assume that J_3 defines the strictest lower bound $x \ge \lceil \frac{y}{3} \rceil$. Since the ceiling function is not part of our syntax, we cannot assign x directly to $\lceil \frac{y}{3} \rceil$. However, we know that $\lceil \frac{y}{3} \rceil$ is one of the three values $\frac{y}{3}$, $\frac{y+1}{3}$, or $\frac{y+2}{3}$. Hence, we do another case distinction (via a disjunction and the factor k) and replace x with $\frac{y+k}{3}$. The result is the subformula $\bigvee_{0 \le k < 3} C_{J_3}$. Finally, we assume that J_4 defines the strictest lower bound $x \ge \lceil \frac{2y-2}{3} \rceil$. Again, we do a case distinction (via a disjunction and the factor k) and replace x with $\frac{2y+k-2}{2}$. The result is the subformula $\bigvee_{0 \le k < 2} C_{J_4}$.

Our notion of weak Cooper elimination is a variant of Cooper elimination, which is very helpful for understanding 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 contains only one divisibility constraint for x. If not, exhaustive application of div-solve to divisibility constraints for x removes all constraints except one: div-solve$(x, d_1 \mid a_1 x + p_1, d_2 \mid a_2 x + p_2) = (d_1 d_2 \mid d x + c_1 d_2 p_1 + c_2 d_1 p_2, d \mid -a_1 p_2 + a_2 p_1)$, where $d = \gcd(a_1 d_2, a_2 d_1)$, and c_1 and c_2 are integers such that $c_1 a_1 d_2 + c_2 a_2 d_1 = d$ (see Cooper (1972); Jovanović and de Moura (2013)). Now weak Cooper elimination takes a variable x, a problem C with only one divisibility constraint $d \mid cx + s \in C$ for x, and produces a new problem by replacing $\exists x.C$ with:

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

where $k \in K$ is a newly introduced variable for every pair of constraints $-ax + p \le 0 \in C$ and $bx - q \le 0 \in C$ with $a, b > 0$,

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

is a resolvent for the same inequalities, where $m := \text{lcm}\left(a, \frac{ad}{\gcd(ad,c)}\right) - 1$, and $\exists K$ abbreviates the sequence of quantified variables $\exists k_1, \ldots, k_m$ contained in $K = \{k_1, \ldots, k_m\}$. The major difference between Cooper elimination and weak Cooper elimination is that one introduces disjunctions and the other variables. For our purposes variables are more beneficial because each new variable k is guarded by the constraints in R_k. Hence, we can eliminate unguarded variables without changing the conjunctive structure of our problems.

Weak Cooper elimination works informally because the transformation determines whether there exists an assignment ν for all variables except x such that the strictest lower and upper bounds for x under ν still contain an integer solution. Let ν be a satisfiable assignment for the

formula after one weak Cooper elimination step on C. Then we compute a strictest lower bound $x \geq l_x$ and a strictest upper bound $x \leq u_x$ from C 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 $-ax + p \leq 0 \in C$ and $bx - q \leq 0 \in C$ 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 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 $d \mid cx + s \in C$. Whenever $l_x = -\infty$ ($u_x = \infty$) we extend ν by an arbitrary small (large) value for x that satisfies $d \mid cx + s \in C$. There exist arbitrarily small (large) solutions for x and $d \mid cx + \nu(s)$ because $\gcd(c, d) \mid s$ is satisfied by ν.

Now, let us look at the result of applying Cooper elimination to the problem C' from Example 11:

Example 12. *Let*

$$C' := \{\underbrace{2 \mid x + y}_{J_1}, \underbrace{2 \mid x + z}_{J_2}, \underbrace{-3x + y \leq 0}_{J_3}, \underbrace{-2x + 2y - 2 \leq 0}_{J_4}, \underbrace{x - 4z \leq 0}_{J_5}\}$$

be the initial problem. We first have to use div-solve *to simplify C' because C' contains two divisibility constraints (J_1 and J_2) for x. Applying* div-solve(x, J_1, J_2) *returns the two divisibility constraints $4 \mid 2x + 2y$ and $2 \mid y - z$, which we will use to replace J_1 and J_2 in C'. The result is the new, but equivalent problem*

$$C := \{\underbrace{4 \mid 2x + 2y}_{J_1'}, \underbrace{2 \mid y - z}_{J_2'}, \underbrace{-3x + y \leq 0}_{J_3}, \underbrace{-2x + 2y - 2 \leq 0}_{J_4}, \underbrace{x - 4z \leq 0}_{J_5}\},$$

which contains only one divisibility constraint (J_1') for x. Now we will split the application of weak Cooper elimination into three steps. First, we select all constraints $I \in C(x)$ without the variable x:

$$\{I \in C : \text{coeff}(I, x) = 0\} := \{2 \mid y - z\}.$$

Next, we take the divisibility constraint $4 \mid 2x + 2y$ and eliminate x from it :

$$(\exists x.\ 4 \mid 2x + 2y) \equiv (\gcd(2, 4) \mid 2y) \equiv (2 \mid 2y).$$

Finally, we construct resolvents for every triple of constraints $-ax + p \leq 0 \in C$, $bx - q \leq 0 \in C$, and $d \mid cx + s \in C$:

$$R_{k_1} := \{-k_1 \leq 0, k_1 - 5 \leq 0, k_1 + y - 12z \leq 0, 3 \mid k_1 + y, 12 \mid 2k_1 + 8y\}$$

for J_3,J_5, and J_1', and

$$R_{k_2} := \{-k_2 \leq 0, k_2 - 3 \leq 0, k_2 + 2y - 8z - 2 \leq 0, 2 \mid k_2 + 2y - 2, 8 \mid 2k_2 + 6y - 4\}$$

for J_4,J_5, and J_1'. The combination of these constraints is then the result of applying weak Cooper elimination:

$$[\exists x.C'] \equiv [\exists x.C] \equiv \left[\exists k_1.\exists k_2.\left(\{J_2'\} \cup \{2 \mid 2y\} \cup R_{k_1} \cup R_{k_2}\right)\right].$$

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 resolvents R_k as well as the divisibility constraint $\gcd(c, d) \mid s$ with respect to a strict ordering on the unguarded variables. The extra divisibility constraint $\gcd(c, d) \mid s$ in weak Cooper elimination is necessary whenever the problem C has no constraint of the form $-ax + p \leq 0 \in C$ or $bx - q \leq 0 \in C$. For example:

Example 13. *let $C = \{y - 1 \leq 0, -y + 1 \leq 0, 6 \mid 2x + y\}$ 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 has not since $2x + 1$ is never divisible by 6.*

The following equivalence states the correctness of weak Cooper elimination:

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

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

Lemma 14 (Divisibility Core Resolvent)**.** *Let k be a new variable. Let $a, b, c > 0$. Then*

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

Proof. See Jovanović and de Moura (2013) pp. 101-102 Lemma 4. □

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 15 (Diophantine Core Resolvent)**.** $(\exists x.d \mid cx + s) \equiv \gcd(c, d) \mid s$.

Proof. We equivalently rewrite the two divisibility constraints into diophantine equations, viz., $d \mid cx + s$ and $gcd(c, d) \mid s$ into $\exists y.dy - cx = s$ and $\exists k.\, \gcd(c, d)k = s$, respectively. We choose d', $c' \in \mathbb{Z}$ such that $d' \cdot \gcd(c, d) = d$ and $c' \cdot \gcd(c, d) = c$. Assume that ν is a variable assignment such that $d\nu(y) - c\nu(x) = \nu(s)$ and, therefore, also $d \mid c\nu(x) + \nu(s)$. Hence, $\nu(s) = d\nu(y) - c\nu(x) = \gcd(c, d) \cdot (d'\nu(y) - c'\nu(x))$. After extending ν with $\nu(k) = (d'\nu(y) - c'\nu(x))$, ν satisfies $\gcd(c, d)k = s$.

Assume that ν is a variable assignment such that $\gcd(c, d)\nu(k) = \nu(s)$ holds and, therefore, also $\gcd(c, d) \mid \nu(s)$. By Beźout's Lemma, there exist $a', b' \in \mathbb{Z}$ such that $a'd - b'c = \gcd(c, d)$. Hence, $a'd\nu(k) - b'c\nu(k) = (a'd - b'c)\nu(k) = \gcd(c, d)\nu(k) = \nu(s)$. After extending ν with $\nu(y) = a'\nu(k)$ and $\nu(x) = b'\nu(k)$, the assignment ν satisfies $dy - cx = s$. □

That means satisfiability of $\gcd(c, d) \mid s$ guarantees a solution for the divisibility constraint $d \mid cx + s$. The rule Resolve-Weak-Cooper (Figure 4) in our CutSat++ exploits these properties by lazily generating the resolvents R_k and the constraint $\gcd(c, d) \mid s$ in the form of *unguarded resolvents*. 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 (Figure 4) perform lazy reduction.

Now let us return to the completeness proof of weak Cooper elimination. In this proof, we will use an assignment ν for all variables but x. Since such an assignment ν fixes all variables but x, we can determine the satisfiability of a problem C by looking at the extensions of ν for x. To this end, we will define the *solution set* $S \subseteq \mathbb{Z}$ for variable x, problem C, and assignment ν as the set of values $v \in S$ that extends ν to a satisfiable solution for C, i.e., $C\{x \mapsto v\}$ $(v \in S)$ is satisfied by ν. If we look, for instance, at the solution set S_d of a divisibility constraint, then we find the following useful property:

Lemma 16 (Divisibility Solution Sets)**.** *Let ν be an assignment for all variables except x. Let S_d be the solution set for variable x, assignment ν, and constraint $d \mid cx + s$. Then*

$$S_d = \emptyset \text{ or } S_d = \{v_0 + ev' : e \in \mathbb{Z}\} \text{ for some } v_0 \in \mathbb{Z}, v' \in \mathbb{Z} \setminus \{0\}.$$

This means that S_d is either empty or unbounded from above and below.

Proof. In case $S_d \neq \emptyset$, there exists a value $v_0 \in S_d$ such that $d \mid cv_0 + \nu(s)$. We first prove that there exists a $v' \in \mathbb{Z} \setminus \{0\}$ such that $d \mid c(v_0 + ev') + \nu(s)$ for all $e \in \mathbb{Z}$ and, therefore, $v_0 + ev' \in S_d$. We choose v' and e' such that $v' := \frac{d}{\gcd(c,d)}$ and $e' := \frac{c}{\gcd(c,d)}$. Then we deduce for any $e \in \mathbb{Z}$:

$$d \mid c(v_0 + ev') + \nu(s) \equiv d \mid cv_0 + \nu(s) + cev' \equiv$$
$$d \mid cv_0 + \nu(s) + ce\tfrac{d}{\gcd(c,d)} \equiv d \mid cv_0 + \nu(s) + dee' \equiv d \mid cv_0 + \nu(s)$$

It remains to show that for every $v_k \in S_d$ there exists an $e \in \mathbb{Z}$ such that $v_0 + ev' = v_k$. As S_d is the solution set, we know that $d \mid cv_0 + \nu(s)$ and $d \mid cv_k + \nu(s)$ are true. Hence, $d \mid c(v_0 - v_k) \equiv d \mid cv_0 + \nu(s) - (cv_k + \nu(s))$. As $d = v' \gcd(c, d)$, the term $c(v_0 - v_k)$ is only divisible by d if $v_0 - v_k$ is divisible by v'. Therefore, $\exists e \in \mathbb{Z}. v_0 - v_k = ev'$. □

In the correctness proof of weak Cooper elimination, Lemma 16 allows us to choose an arbitrary small or large solution for x that satisfies $d \mid cx + \nu(s)$. As mentioned in the outline of the proof, the ability to choose arbitrary small and large solutions for x is necessary when C contains no constraints of the form $-ax + p \leq 0$ or $bx - q \leq 0$.

Theorem 17 (Weak Coooper Correctness)**.**

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

where $d \mid cx + s$ is the only divisibility constraint in C, each $k \in K$ is a newly introduced variable for every pair of constraints $-ax + p \leq 0 \in C$ and $bx - q \leq 0 \in C$ with $a, b > 0$,

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

is a resolvent for the same inequalities (and the divisibility constraint), $m := \text{lcm}\left(a, \frac{ad}{\gcd(ad,c)}\right) - 1$, and $\exists K$ abbreviates the sequence of quantified variables $\exists k_1, \ldots, k_m$ contained in $K = \{k_1, \ldots, k_m\}$.

Proof. First, we partition the problem C as follows:

$$\begin{array}{llll} C_l & = \{-ax + p \leq 0 \in C : a > 0\}, & C_u & = \{bx - q \leq 0 \in C : b > 0\}, \\ I_d & = d \mid cx + s \in C, & C_r & = \{I \in C : \text{coeff}(I, x) = 0\}. \end{array}$$

By Lemma 14, it holds for all $-ax + p \leq 0, bx - q \leq 0 \in C$ with $a, b > 0$ that:

$$\begin{gathered}(\exists x.C) \rightarrow (\exists x.\{-ax + p \leq 0, bx - q \leq 0, d \mid cx + s\}) \\ \rightarrow (\exists k.\underbrace{\{-k \leq 0, k - m \leq 0, bp - aq + bk \leq 0, a \mid k + p,\ ad \mid cp + as + ck\}}_{R_k}).\end{gathered}$$

By Lemma 15, it holds that: $(\exists x.C) \rightarrow (\exists x.d \mid cx + s) \rightarrow \gcd(c, d) \mid s$. Since $C_r \subseteq C$, it also holds that: $(\exists x.C) \rightarrow C_r$. As all new variables $k \in K$ appear only in one resolvent R_k, the above implications prove

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

Assume, vice versa, that ν is a satisfiable assignment for the formula after one step of weak Cooper elimination. Then it is easy to deduce the following facts:

- Let S_l be the solution set for x, ν, and $I_l = -ax + p \leq 0 \in C$ with $a > 0$. Then $S_l = \left\{\lceil \frac{\nu(p)}{a} \rceil, \lceil \frac{\nu(p)}{a} \rceil + 1, \ldots\right\}$.
- Let S_u be the solution set for x, ν, and $I_u = bx - q \leq 0 \in C$ with $b > 0$. Then $S_u = \left\{\ldots, \lfloor \frac{\nu(q)}{b} \rfloor - 1, \lfloor \frac{\nu(q)}{b} \rfloor\right\}$.
- Let S_I be the solution set for x, ν, and $C_l \cup C_u$. Then $S_I = \bigcap_{I_l \in C_l} S_l \cap \bigcap_{I_u \in C_u} S_u$.
- Let the set S_I be bounded from below, i.e., $S_I = \{l, l + 1, \ldots\}$ or $S_I = \{l, \ldots, u\}$. Then $l = \max_{I \in C_l} \left\{\lceil \frac{\nu(p')}{a'} \rceil : I = -a'x + p' \leq 0\right\}$.
- Let the set S_I be bounded from above, i.e., $S_I = \{\ldots, u - 1, u\}$ or $S_I = \{l, \ldots, u\}$. Then $u = \min_{I \in C_u} \left\{\lfloor \frac{\nu(q)}{b} \rfloor : I = b'x - q' \leq 0\right\}$.
- By Lemma 15, $d \mid cx + \nu(s)$ is satisfiable because $\gcd(c, d) \mid s$ is contained in the result formula of weak Cooper elimination. By Lemma 16, the set of solutions for x, ν, and $d \mid cx + s$ has the form $S_d = \{v_0 + ev' : v' \in \mathbb{Z}\}$.
- The solution set S for x, ν, and C' is $S = S_d \cap S_I$.

Next, we do a case distinction on the structure of C:

- Let $C_l = \emptyset$, then S_I is unbounded from below. We choose a small enough $v \in S_d$, i.e., small enough $e \in \mathbb{Z}$ such that $v = v_0 + ev'$. Then the assignment $x \mapsto v$ and $y \mapsto \nu(y)$ (if $y \neq x$) satisfies C'.
- Let $C_u = \emptyset$, then S_I is unbounded from above. We choose a large enough $v \in S_d$, i.e., large enough $e \in \mathbb{Z}$ such that $v = v_0 + ev'$. Then the assignment $x \mapsto v$ and $y \mapsto \nu(y)$ for all $y \neq x$ satisfies C'.
- Let $|C_l|, |C_u| > 0$. We select $I_l = -ax + p \leq 0$ such that

$$\lceil \frac{\nu(p)}{a} \rceil = \max_{I \in C_l} \left\{\lceil \frac{\nu(p')}{a'} \rceil : I = -a'x + p' \leq 0\right\}$$

Algorithm 1: CombDivs(x, C')

Input : A variable x and a set of LIA constraints C'
Output: A set of LIA constraints C such that $C \equiv C'$ and there exists exactly one divisibility constraint $d \mid cx + s \in C$ such that $c \neq 0$

$C_d := \{d \mid cx + s \in C' : c > 0\}$
$C := C' \setminus C_d$;
if $(C_d = \emptyset)$ **then**
  return $C \cup \{1 \mid x\}$;
while $(|C_d| > 1)$ **do**
  Select $d_1 \mid a_1x + p_1, d_2 \mid a_2x + p_2 \in C_d$;
  $C_d := C_d \setminus \{d_1 \mid a_1x + p_1, d_2 \mid a_2x + p_2\}$;
  $d = \gcd(a_1d_2, a_2d_1)$;
  Choose c_1 and c_2 such that $c_1a_1d_2 + c_2a_2d_1 = d$;
  $C_d := C_d \cup \{d_1d_2 \mid dx + c_1d_2p_1 + c_2d_1p_2\}$;
  $C := C \cup \{d \mid -a_1p_2 + a_2p_1\}$;
end
return $C \cup C_d$;

Figure 3: An algorithm that combines constraints $C_d = \{d \mid cx + s \in C' : c > 0\}$ until only one divisibility constraint for x remains

and $I_u = bx - q \leq 0$ such that

$$\lfloor \frac{\nu(q)}{b} \rfloor = \min_{I \in C_u} \left\{ \lfloor \frac{\nu(q')}{b'} \rfloor : I = b'x - q' \leq 0 \right\}.$$

The resolvent for the two constraints I_l and I_u is

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

We will now show that $\frac{\nu(p+k)}{a}$ is in the set of solutions S of C. All of the remaining deductions stem from the evaluation of the resolvent under ν. Since $a \mid \nu(p+k)$, $\frac{\nu(p+k)}{a} \in \mathbb{Z}$. Furthermore, since $\frac{\nu(p+k)}{a} \in \mathbb{Z}$ and $\nu(bp - aq + bk) \leq 0$, $\frac{\nu(p+k)}{a} \in S_I = \left\{ \lceil \frac{\nu(p)}{a} \rceil, \ldots, \lfloor \frac{\nu(q)}{b} \rfloor \right\}$. Finally, since

$$ad \mid \nu(cp + as + ck) = ad \mid ac\nu(x) + a\nu(s) = d \mid c\nu(x) + \nu(s),$$

$\frac{\nu(p+k)}{a} \in S_d$. We choose the assignment ν' with $x \mapsto \frac{\nu(p+k)}{a}$ and $y \mapsto \nu(y)$ for all $y \neq x$. Hence, ν' satisfies C'.

□

We stated that weak Cooper elimination can only be applied to those problems where C contains one divisibility constraint $d \mid ax + p$ in x. To expand weak Cooper elimination to any set of constraints C', we briefly explained how to exhaustively apply div-solve to eliminate all but one constraint $d \mid ax + p$ for x. The algorithm CombDivs(x, C') (Figure 3) is a more detailed version of this procedure.

Lemma 18 (CombDivs Equivalence)**.** *Let C' be a set of LIA constraints. Let C be the set of LIA constraints we receive from* CombDivs(x, C')*. Then $C \equiv C'$.*

Proof. Follows directly from the proof of equivalence of the div-solve transformation (see Jovanović and de Moura (2013)). □

The relationship between CutSat++ and weak Cooper elimination is as follows: On the left side of the equivalence in Theorem 17 there occur two combinations of constraints: triples consisting of two inequations with a divisibility constraint and single divisibility constraints. Each triple of constraints $\{-ax + p \leq 0, bx - q \leq 0, d \mid cx + s\} \in C$ is a potential divisibility conflicting core[6], i.e., a set of constraints that can turn into a conflicting core if p, q, s are fixed and the constraints are contradictory for variable x. Moreover, the single divisibility constraint $\{d \mid cx + s\} \in C$ is a potential diophantine conflicting core, i.e., a divisibility constraint that can turn into a conflicting core if it becomes contradictory for variable x after s is fixed. On the right side of the equivalence in Theorem 17, the resolvents R_k for the respective triple of constraints and the resolvent $\gcd(c, d) \mid s$ for the divisibility constraint are equivalent to the unguarded resolvent that CutSat++ introduces for the appropriate conflicting cores. However, compared to CutSat++, weak Cooper elimination introduces all resolvents at once and, thereby, ensures that the resolvents subsume all potential conflicting cores. Therefore, we can replace all constraints containing x with the resolvents for conflicting cores over x and receive an equisatisfiable formula (see Theorem 17). This means that Theorem 17 shows that no other conflicting cores are necessary besides interval, divisibility, and diophantine conflicting cores. Moreover, Example 13 shows that we need to consider diophantine conflicting cores—and not just interval and divisibility conflicting cores—to preserve equisatisfiability.

The preprocessing step described in the CombDivs(x, C') algorithm can also be found in CutSat and CutSat++. However, CutSat and CutSat++ perform CombDivs(x, C') lazily; the former with the rule Solve-Div, the latter with the rules Solve-Div-Left and Solve-Div-Right.

5. Unguarded Conflict Resolution

Weak Cooper elimination is capable of exploring all unguarded variables to eventually create a problem where feasibility depends only on guarded variables. We simulate it in a lazy manner by extending GCutSat with three new rules, which we call the unguarded conflict resolution rules (Figure 4). The now extended rule system is our calculus CutSat++. 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. Moreover, CutSat++ is not required to apply the intermediate steps of weak Cooper elimination one variable at a time and does not eliminate unguarded variables. This has the advantage that CutSat++ might find a satisfiable assignment or detect unsatisfiability without encountering and resolving a large number of unguarded conflicts. As a result, 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. Only in the worst case, CutSat++ has to perform all of weak Cooper elimination's intermediate steps.

[6] Or a potential interval conflicting core if the divisibility constraint $d \mid cx + s$ is not part of the contradiction.

Solve-Div-Left

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

Solve-Div-Right

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

Resolve-Weak-Cooper

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

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

Figure 4: Our unguarded conflict resolution rules

Then the strictly-two-layered strategy (Definition 24) guarantees that CutSat++ recognizes that all unguarded conflicts have been produced.

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 $\prec$ describes the elimination order for the unguarded variables, viz., $x_j \succ x_i$ 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* ($x = \text{top}(I)$).

Definition 19 (Conflicting Cores)**.** *Let $S = \langle M, C\rangle$ be a state, $C' \subseteq C$, x the top variable in C', $a, b > 0$, 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) $C' = \{-ax + p \le 0, bx - q \le 0\}$ and $\text{bound}(-ax + p \le 0, x, M) > \text{bound}(bx - q \le 0, x, M)$ *, i.e., the lower bound from $-ax + p \le 0$ contradicts the upper bound from $bx - q \le 0$; in this case (x, C') is called an* interval conflicting core *and its* unguarded resolvent *is* $(\{-k \le 0, k - a + 1 \le$

$0\}, \{bp - aq + bk \leq 0, a \mid k + p\}$)
(2) $C' = \{-ax + p \leq 0, bx - q \leq 0, d \mid cx + s\}$ *and* $b_l = \text{bound}(-ax + p \leq 0, x, M)$, $b_u = \text{bound}(bx - q \leq 0, x, M)$, $b_l \leq b_u$, *and for all* $b_d \in [b_l, b_u]$ *we have* $d \nmid cb_d + \text{lower}(s, M)$, *i.e., there exists no value for* x *within the bounds defined by the two inequalities such that the divisibility constraint becomes satisfiable; in this case* (x, C') *is called a* divisibility conflicting core *and its* unguarded 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* $d \nmid cb_d + \text{lower}(s, M)$, *i.e., there exists no integer value for* x *that could satisfy* $d \mid cx + s$; *in this case* (x, C') *is called a* diophantine conflicting core *and its* unguarded resolvent *is* $(\emptyset, \{gcd(c, d) \mid s\})$.
In the first two cases, k *is a fresh variable and* $m = \text{lcm}\left(a, \frac{ad}{\text{gcd}(ad,c)}\right) - 1$.

We refer to the respective unguarded resolvents for a conflicting core (x, C') by the function w-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 pair (x, C') is a *potential conflicting core* if there exists a state S where (x, C') is a conflicting core.

Next, we define a generalization of unguarded resolvents. Since the unguarded 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-Weak-Cooper to the same conflicting core. All unguarded resolvents of Definition 19 are also unguarded resolvents in the sense of the below definition (see also end of Section 4).

Definition 20 (Unguarded Resolvents)**.** *A set of constraints* R *is an* unguarded resolvent *for the pair* (x, C') *if it holds that* $R \rightarrow \exists x.C'$ *and* $\forall J \in R. \text{top}(J) \prec x$, *i.e., the satisfiability of the unguarded resolvent* R *implies the satisfiability of* C', *while using only variables that are smaller than* x.

Lemma 21 (Unguarded Resolvent Soundness)**.** *Let* $C' \subseteq C$. *Let* w-cooper$(x, C') = (R_k, R_c)$. *Let* $R = R_k \cup R_c$. *Then* $\exists k : C \cup R \equiv C$. *Furthermore,* R *is a unguarded resolvent for* (x, C').

Proof. Follows directly from the Lemmas 14 and 15. The interval conflicting core is the only new case; however, since w-cooper$(x, \{-ax + p \leq 0, bx - q \leq 0\})$ is equivalent to w-cooper$(x, \{-ax + p \leq 0, bx - q \leq 0, 1 \mid x\})$ (after simplifications), we have to consider only divisibility and diophantine conflicting cores. Therefore, by Lemmas 14 and 15, $R \rightarrow \exists x : C'$. Finally, $J \in R : \text{top}(J) \prec x$ holds because k is the minimal element of $\prec$ and all other variables in R appear in C' where x is maximal. □

We restrict the rule Resolve-Weak-Cooper (Figure 4) to unguarded constraints because we could otherwise generate infinitely many guarded variables. This poses no problem for efficiency because the standard conflict resolution is already very efficient on guarded constraints. Moreover, Resolve-Weak-Cooper 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. The restrictions also prevent Resolve-Weak-Cooper from being applied to the conflicting core (x, C') whenever our set of constraints already contains an unguarded resolvent R for this core. Since Resolve-Weak-Cooper adds an unguarded resolvent for every conflicting core it is applied to, this also means that Resolve-Weak-Cooper is never applied twice to the same conflicting core.

Lemma 22 (Blocking Resolve-Weak-Cooper)**.** *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 an* unguarded resolvent *for (x, C'). Then Resolve-Weak-Cooper is not applicable to (x, C').*

Proof. Assume for contradiction that $D = (x, C')$ is a conflicting core, $R \in C$ is an unguarded resolvent for D in state S, and Resolve-Weak-Cooper is applicable to D in state S. Resolve-Weak-Cooper requires that all variables $y \prec x$ are fixed (Figure 4). This holds especially for all variables in R (Definition 20). Due to the restriction that every conflict $J \in C$ has $\text{top}(J) \not\prec \text{top}(I)$ in Resolve-Weak-Cooper, there is no conflict in R. Furthermore, since all variables $y \prec x$ are fixed, R is satisfied by the partial assignment defined by M. By Definition 19, all conflicting cores have no satisfiable solution for x under partial model M. However, by Definition 20, R satisfiable implies that there exists an x such that C' is satisfiable under M. This contradicts the assumption that (x, C') is a conflicting core! □

If we add unguarded 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-Weak-Cooper. The remaining guarded conflicts are resolved with the rules Conflict and Conflict-Div.

The rules Solve-Div-Left and Solve-Div-Right (Figure 4) combine divisibility constraints as 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 is possible (see Example 7). 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, CutSat++ constructs only unguarded conflicts that are either blocked by a resolvent or resolvable by either Solve-Div-Right or Resolve-Weak-Cooper (Figure 4). Note, however, that, due to the rules 1(b) and 2(c) of the strategy, multiple applications of the Solve-Div-Left rule might be necessary before a resolution is possible.

Definition 23 (Eager Top-Level Propagating Strategy)**.** *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. Moreover, if I is a divisibility constraint $d \mid ax + p$, then we only propagate $d \mid ax + p$ if:*
   (a) $\text{lower}(x, M) \neq -\infty$ *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.*

$$
\begin{aligned}
&\langle [\![]\!], C\rangle \\
&\Longrightarrow^{\text{Propagate}}_{\text{CS++}} && \langle [\![x \geq_{I_x} 0]\!], C\rangle \\
&\Longrightarrow^{\text{Propagate}}_{\text{CS++}} && \langle [\![x \geq_{I_x} 0, y \geq_{I_y} 0]\!], C\rangle \\
&\Longrightarrow^{\text{Propagate}}_{\text{CS++}} && \langle [\![x \geq_{I_x} 0, y \geq_{I_y} 0, z \leq_{I_{z1}} 0]\!], C\rangle \\
&\Longrightarrow^{\text{Propagate}}_{\text{CS++}} && \langle [\![x \geq_{I_x} 0, y \geq_{I_y} 0, z \leq_{I_{z1}} 0, z \geq_{I_{z2}} 0]\!], C\rangle \\
&\Longrightarrow^{\text{Conflict}}_{\text{CS++}} && \langle [\![x \geq_{I_x} 0, y \geq_{I_y} 0, z \leq_{I_{z1}} 0, z \geq_{I_{z2}} 0]\!], C, z + 1 \leq 0\rangle \\
&\Longrightarrow^{\text{Resolve}}_{\text{CS++}} && \langle [\![x \geq_{I_x} 0, y \geq_{I_y} 0, z \leq_{I_{z1}} 0]\!], C, 1 \leq 0\rangle \\
&\Longrightarrow^{\text{Unsat}}_{\text{CS++}} && \mathit{unsat}
\end{aligned}
$$

Figure 5: A CutSat++ run for Example 25 depicted as a transition of states

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 $y \prec x$. This in itself already prevents divergence due to bound propagation. Moreover, the strategy makes decisions for unguarded variable x only 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. We need this restriction because Resolve-Weak-Cooper can only resolve conflicts between constraints (see Definition 19) and, therefore, cannot resolve conflicts based on top variable decisions. 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-Weak-Cooper and Solve-Div-Right to a state where this is not the case. To avoid stuck states resulting from the eager top-level propagating strategy, the slack variable x_S has to be the smallest unguarded variable in $\prec$. Otherwise, the constraints $x - x_S \leq 0$, $-x - x_S \leq 0$ introduced by Slack-Intro cannot be used to propagate bounds for variable x and x would remain stuck.

Definition 24 (Strictly-Two-Layered Strategy)**.** *A strategy is* strictly-two-layered *if:*
(1) it is reasonable (Definition 1), (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 an unguarded 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 24-(3) it partitions conflict resolution into two layers: While every unguarded conflict is handled with the rules Resolve-Weak-Cooper, Solve-Div-Left, and Solve-Div-Right (Figure 4), every guarded conflict is handled with the rules Conflict(-Div). The conditions 24-(1) and 24-(5) make the guarded variables independent from the unguarded variables. The conditions 24-(2) and 24-(4) give a guarantee that the rules Resolve-Weak-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.

To better show how CutSat++ works, let us look again at the examples from Section 2. In contrast to CutSat, CutSat++ terminates with a correct solution on these examples:

$$
\begin{aligned}
&\langle [\![]\!], C_0\rangle \\
&\Longrightarrow^{\text{Slack-Intro}}_{\text{CS++}} \quad \langle [\![]\!], C_1\rangle \\
&\qquad\qquad \text{where } C_1 := C_0 \cup \{\underbrace{-x_s \le 0}_{I_{x_s}}, \underbrace{x - x_s \le 0}_{I_{\text{x1}}}, \underbrace{-x - x_s \le 0}_{I_{\text{x2}}}\} \\
&\Longrightarrow^{\text{Propagate}}_{\text{CS++}} \quad \langle [\![x_s \ge_{I_{x_s}} 0]\!], C_1\rangle \\
&\Longrightarrow^{\text{Decide}}_{\text{CS++}} \quad \langle [\![x_s \ge_{I_{x_s}} 0, x_s \le 0]\!], C_1\rangle \\
&\Longrightarrow^{\text{Propagate}}_{\text{CS++}} \quad \langle [\![x_s \ge_{I_{x_s}} 0, x_s \le 0, x \le_{I_{\text{x1}}} 0]\!], C_1\rangle \\
&\Longrightarrow^{\text{Propagate}}_{\text{CS++}} \quad \langle [\![x_s \ge_{I_{x_s}} 0, x_s \le 0, x \le_{I_{\text{x1}}} 0, x \ge_{I_{\text{x2}}} 0]\!], C_1\rangle \\
&\Longrightarrow^{\text{Slack-Intro}}_{\text{CS++}} \quad \langle [\![x_s \ge_{I_{x_s}} 0, x_s \le 0, x \le_{I_{\text{x1}}} 0, x \ge_{I_{\text{x2}}} 0]\!], C_2\rangle \\
&\qquad\qquad \text{where } C_2 := C_1 \cup \{\underbrace{y - x_s \le 0}_{I_{\text{y1}}}, \underbrace{-y - x_s \le 0}_{I_{\text{y2}}}\} \\
&\Longrightarrow^{\text{Propagate}}_{\text{CS++}} \quad \langle [\![x_s \ge_{I_{x_s}} 0, x_s \le 0, x \le_{I_{\text{x1}}} 0, x \ge_{I_{\text{x2}}} 0, y \le_{I_{\text{y1}}} 0]\!], C_2\rangle \\
&\Longrightarrow^{\text{Propagate}}_{\text{CS++}} \quad \langle [\![x_s \ge_{I_{x_s}} 0, x_s \le 0, x \le_{I_{\text{x1}}} 0, x \ge_{I_{\text{x2}}} 0, y \le_{I_{\text{y1}}} 0, y \ge_{I_{\text{y2}}} 0]\!], C_2\rangle \\
&\Longrightarrow^{\text{Slack-Intro}}_{\text{CS++}} \quad \langle [\![x_s \ge_{I_{x_s}} 0, x_s \le 0, x \le_{I_{\text{x1}}} 0, x \ge_{I_{\text{x2}}} 0, y \le_{I_{\text{y1}}} 0, y \ge_{I_{\text{y2}}} 0]\!], C_3\rangle \\
&\qquad\qquad \text{where } C_3 := C_2 \cup \{\underbrace{z - x_s \le 0}_{I_{\text{z1}}}, \underbrace{-z - x_s \le 0}_{I_{\text{z2}}}\} \\
&\Longrightarrow^{\text{Propagate}}_{\text{CS++}} \quad \langle [\![x_s \ge_{I_{x_s}} 0, x_s \le 0, x \le_{I_{\text{x1}}} 0, x \ge_{I_{\text{x2}}} 0, y \le_{I_{\text{y1}}} 0, y \ge_{I_{\text{y2}}} 0, \\
&\qquad\qquad z \le_{I_{\text{z1}}} 0]\!], C_3\rangle \\
&\Longrightarrow^{\text{Propagate}}_{\text{CS++}} \quad \langle [\![x_s \ge_{I_{x_s}} 0, x_s \le 0, x \le_{I_{\text{x1}}} 0, x \ge_{I_{\text{x2}}} 0, y \le_{I_{\text{y1}}} 0, y \ge_{I_{\text{y2}}} 0, \\
&\qquad\qquad z \le_{I_{\text{z1}}} 0, z \ge_{I_{\text{z2}}} 0]\!], C_3\rangle \\
&\Longrightarrow^{\text{Sat}}_{\text{CS++}} \quad \langle \nu[[\![x_s \ge_{I_{x_s}} 0, x_s \le 0, x \le_{I_{\text{x1}}} 0, x \ge_{I_{\text{x2}}} 0, y \le_{I_{\text{y1}}} 0, y \ge_{I_{\text{y2}}} 0, \\
&\qquad\qquad z \le_{I_{\text{z1}}} 0, z \ge_{I_{\text{z2}}} 0]\!]], C_3\rangle
\end{aligned}
$$

Figure 6: A CutSat++ run for Example 26 depicted as a transition of states

Example 25. *Let*

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

be a problem. Let the variable order be given by $z \prec y \prec x$. Then CutSat++ *would solve the problem as depicted in Figure 5. First of all,* CutSat++ *has to propagate I_x, I_y, I_{z1}, and I_{z2}, i.e., all constraints containing only one variable. However, this already turns I_{z3} into a conflicting constraint. Although we could add decided bounds and propagated bounds until $(x, \{J_1, J_2\})$ turns into a conflicting core, we could never apply Resolve-Weak-Cooper to $(x, \{J_1, J_2\})$ because* $\text{top}(I_{\text{z3}}) \prec x$ *and I_{z3} is conflicting. Therefore,* CutSat++ *has to apply at some point Conflict to I_{z3}. After applying Resolve to $z + 1 \le 0$ with the bound $z \ge_{I_{\text{z2}}} 0$,* CutSat++ *has derived the trivially unsatisfiable constraint $1 \le 0$. Finally,* CutSat++ *uses $1 \le 0$ to apply Unsat and, thereby, return* unsat.

$\langle [\![]\!], C_0 \rangle$

$\Longrightarrow^{\text{Propagate}}_{\text{CS++}}$ $\langle [\![z \geq_{I_{z1}} 0]\!], C_0 \rangle$

$\Longrightarrow^{\text{Propagate}}_{\text{CS++}}$ $\langle [\![z \geq_{I_{z1}} 0, z \leq_{I_{z2}} 0]\!], C_0 \rangle$

$\Longrightarrow^{\text{Propagate}}_{\text{CS++}}$ $\langle [\![z \geq_{I_{z1}} 0, z \leq_{I_{z2}} 0, x \geq_{I_x} 0]\!], C_0 \rangle$

$\Longrightarrow^{\text{Propagate}}_{\text{CS++}}$ $\langle [\![z \geq_{I_{z1}} 0, z \leq_{I_{z2}} 0, x \geq_{I_x} 0, y \geq_{I_y} 0]\!], C_0 \rangle$

$\Longrightarrow^{\text{Decide}}_{\text{CS++}}$ $\langle [\![z \geq_{I_{z1}} 0, z \leq_{I_{z2}} 0, x \geq_{I_x} 0, y \geq_{I_y} 0, x \leq 0]\!], C_0 \rangle$

$\Longrightarrow^{\text{Resolve-Weak-Cooper}}_{\text{CS++}}$ $\langle [\![z \geq_{I_{z1}} 0, z \leq_{I_{z2}} 0, x \geq_{I_x} 0, y \geq_{I_y} 0]\!], C_1 \rangle$

where the new order is $z \prec k \prec x \prec y$, and $C_1 := C_0 \cup R_k \cup R_c$

$\Longrightarrow^{\text{Propagate}}_{\text{CS++}}$ $\langle [\![z \geq_{I_{z1}} 0, z \leq_{I_{z2}} 0, x \geq_{I_x} 0, y \geq_{I_y} 0, k \leq_{K_2} 0]\!], C_1 \rangle$

$\Longrightarrow^{\text{Propagate}}_{\text{CS++}}$ $\langle [\![z \geq_{I_{z1}} 0, z \leq_{I_{z2}} 0, x \geq_{I_x} 0, y \geq_{I_y} 0, k \leq_{K_2} 0, k \geq_{K_1} 0]\!], C_1 \rangle$

$\Longrightarrow^{\text{Conflict}}_{\text{CS++}}$ $\langle [\![z \geq_{I_{z1}} 0, z \leq_{I_{z2}} 0, x \geq_{I_x} 0, y \geq_{I_y} 0, k \leq_{K_2} 0, k \geq_{K_1} 0]\!], C_1, k + 1 \leq 0 \rangle$

$\Longrightarrow^{\text{Resolve}}_{\text{CS++}}$ $\langle [\![z \geq_{I_{z1}} 0, z \leq_{I_{z2}} 0, x \geq_{I_x} 0, y \geq_{I_y} 0, k \leq_{K_2} 0]\!], C_1, 1 \leq 0 \rangle$

$\Longrightarrow^{\text{Unsat}}_{\text{CS++}}$ *unsat*

Figure 7: A CutSat++ run for Example 27

Example 26. *Let $C_0 = \{4 \mid 2x + 2y, 2 \mid x + z\}$ be a problem. Let the variable order be given by $x \prec y \prec z$. Then* CutSat++ *would solve the problem as depicted in Figure 6. Since C_0 has no constraints containing only one variable,* CutSat++ *cannot propagate any bounds and all variables are stuck in state S_0. To resolve the stuck state,* CutSat++ *applies Slack-Intro to x. Now,* CutSat++ *is able to propagate the bound $x_s \geq_{I_{x_s}} 0$ for the slack variable x_s and even fix it with the decided bound $x_s \leq 0$. Since x_s is fixed,* CutSat++ *propagates the bounds $x \leq_{I_{x1}} 0$ and $x \geq_{I_{x2}} 0$. However, the variables y and z are again stuck. We resolve this by repeating what we did for x, first to the variable y and then to the variable z. Finally,* CutSat++ *returns the satisfiable assignment with an application of the Sat rule.*

Note that CutSat++ *could never apply the rules Solve-Div-Left or Solve-Div-Right because the top variables of the divisibility constraints are different.*

Example 27. *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 (R_k, R_c) be the output of w-cooper$(y, \{J_1, J_2\})$ *so that*

$$(R_k, R_c) := (\{\underbrace{-k \leq 0}_{K_1}, \underbrace{k \leq 0}_{K_2}\}), \{\underbrace{k + 1 \leq 0}_{K_3}, \underbrace{1 \mid k + y + z + 1}_{K_4}\}).$$

Let the variable order be given by $z \prec x \prec y$. Then CutSat++ *would solve the problem as depicted in Figure 7. First of all,* CutSat++ *has to propagate I_{z1}, I_{z2}, I_x, and I_y, i.e., all constraints containing only one variable. Then,* CutSat++ *fixes x with a decided bound $x \geq_{I_x} 0$, which turns $(y, \{J_1, J_2\})$ into a conflicting core. Since only J_1 is a conflicting constraint and $y = \text{top}(J_1)$, the*

$$
\begin{aligned}
&\langle [\![]\!], C_0\rangle \\
&\Longrightarrow^{\text{Propagate}}_{\text{CS++}} && \langle [\![x \geq_{I_{x1}} 0]\!], C_0\rangle \\
&\Longrightarrow^{\text{Propagate}}_{\text{CS++}} && \langle [\![x \geq_{I_{x1}} 0, x \leq_{I_{x2}} 1]\!], C_0\rangle \\
&\Longrightarrow^{\text{Propagate}}_{\text{CS++}} && \langle [\![x \geq_{I_{x1}} 0, x \leq_{I_{x2}} 1, y \geq_{I_y} 0]\!], C_0\rangle \\
&\Longrightarrow^{\text{Decide}}_{\text{CS++}} && \langle [\![x \geq_{I_{x1}} 0, x \leq_{I_{x2}} 1, y \geq_{I_y} 0, x \geq 1]\!], C_0\rangle \\
&\Longrightarrow^{\text{Resolve-Weak-Cooper}}_{\text{CS++}} && \langle [\![x \geq_{I_{x1}} 0, x \leq_{I_{x2}} 1, y \geq_{I_y} 0]\!], C_1\rangle \\
& && \quad \text{where } C_1 := C_0 \cup J_2 \\
&\Longrightarrow^{\text{Propagate-Div}}_{\text{CS++}} && \langle [\![x \geq_{I_{x1}} 0, x \leq_{I_{x2}} 1, y \geq_{I_y} 0, x \leq_{J_2'} 0]\!], C_1\rangle \\
& \quad \text{where } J_2' := \text{div-derive}(J_2, x, [\![x \geq_{I_{x1}} 0, x \leq_{I_{x2}} 1, y \geq_{I_y} 0]\!]) = x \leq 0 \\
&\Longrightarrow^{\text{Decide}}_{\text{CS++}} && \langle [\![x \geq_{I_{x1}} 0, x \leq_{I_{x2}} 1, y \geq_{I_y} 0, x \leq_{J_2'} 0, y \leq 0]\!], C_1\rangle \\
&\Longrightarrow^{\text{Sat}}_{\text{CS++}} && \langle \nu[[\![x \geq_{I_{x1}} 0, x \leq_{I_{x2}} 1, y \geq_{I_y} 0, x \leq_{J_2'} 0, y \leq 0]\!]], C_1\rangle
\end{aligned}
$$

Figure 8: A CutSat++ run for Example 28

rule Resolve-Weak-Cooper is applicable and CutSat++ *will use it to resolve the conflicting core* $(y, \{J_1, J_2\})$*. The resulting resolvent is* (R_k, R_c) *for which* CutSat++ *introduces the fresh variable* k *and updates the variable order to* $z \prec k \prec x \prec y$*. Resolve-Weak-Cooper also truncates the bound sequence to remove all decided bounds for variables greater than or equal to* k*. Next,* CutSat++ *has to propagate the newly introduced constraints* K_1 *and* K_2 *because* CutSat++ *follows a reasonable strategy and* K_1 *and* K_2 *are of the form* $\pm k + b \leq 0$*. However, this turns the constraint* K_3 *into a guarded conflict constraint. After applying Conflict to* K_3 *and resolving* K_3 *with* $x \geq_{K_1} 0$*,* CutSat++ *has derived the trivially unsatisfiable constraint* $1 \leq 0$*. Finally,* CutSat++ *uses* $1 \leq 0$ *to apply Unsat and, thereby, return* unsat*.*

Example 28. *Let*

$$C_0 := \{\underbrace{-x \leq 0}_{I_{x1}}, \underbrace{x - 1 \leq 0}_{I_{x2}}, \underbrace{-y \leq 0}_{I_y}, \underbrace{6 \mid 4y + x}_{J}\}$$

be a problem. Let $(R_k, R_c) := (\emptyset, \{J_2\})$ *be the output of* w-cooper$(y, \{J\})$ *where* $J_2 = 2 \mid x$*. Let the variable order be given by* $x \prec y$*. Then* CutSat++ *would solve the problem as depicted in Figure 8. First of all,* CutSat++ *has to propagate* I_{x1}*,* I_{x2}*, and* I_y*, i.e., all constraints containing only one variable. Then* CutSat++ *will fix* x *to the bound* $x \leq_{I_{x1}} 1$ *with the decided bound* $x \geq 1$*, which turns* $(y, \{J\})$ *into a conflicting core. Note that* CutSat++ *cannot use* $J_1 = 6 \mid 4y + x$ *to propagate a bound for* y *because* CutSat++ *follows an eager top-level propagating strategy and* $\gcd(4, 6) = 2$ *does not divide* 1 *the value* x *is fixed to. For the same reason* CutSat++ *cannot fix* y *with a decided bound. Since only* J *is a conflicting constraint and* $y = \text{top}(J)$*, the rule Resolve-Weak-Cooper is applicable and* CutSat++ *will use it to resolve the conflicting core* $(y, \{J\})$*. The resulting resolvent is* $(\emptyset, \{J_2\})$*, where* $J_2 = 2 \mid x$*. Resolve-Weak-Cooper also truncates the bound sequence so that all decided bounds for variables greater than or equal to* x *are removed. Now* CutSat++ *propagates the constraint* J_2 *to fix* x *to* 0 *instead of* 1*. Since* $6 \mid 4y + x$ *is satisfied by the bounds for* x *and* y*,* CutSat++ *is also able to fix* y *with a decided bound. Finally,* CutSat++ *returns the satisfiable assignment with an application of the Sat rule.*

6. 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 from GCutSat (see Jovanović and de Moura (2013)), as well as Resolve-Weak-Cooper, Solve-Div-Left, and Solve-Div-Right (Fig. 4). Before we prove termination and completeness for CutSat++, we have to prove another property over unguarded resolvents. We have proven in Section 5 that Resolve-Weak-Cooper applied to the conflicting core (x, C') adds an unguarded resolvent R that blocks another application of Resolve-Weak-Cooper to (x, C'). However, CutSat++ is able to remove constraints from R with the rules Solve-Div-Left and Solve-Div-Right. This removes the original conflicting core R from our state. Nonetheless, CutSat++ is still unable to apply Resolve-Weak-Cooper to conflicting core (x, C') because the rules Solve-Div-Left and Solve-Div-Right guarantee that a new unguarded resolvent R' for the conflicting core (x, C') is introduced:

Lemma 29 (Resolvent Stability)**.** *Let $S = \langle M, C\rangle$ be a state reachable by* CutSat++ *from the initial state $\langle [\![]\!], C_0\rangle$ and let $S' = \langle M', C'\rangle$ be a state reachable by* CutSat++ *from S. Let C contain an unguarded resolvent R for (x, C''). Then C' contains also an unguarded resolvent R' for (x, C'').*

Proof. Assume for a contradiction that C contains an unguarded resolvent R for (x, C'') and C' contains no unguarded resolvent R' for (x, C''). W.l.o.g., we assume that S' is the first state after S where $R \not\subseteq C'$. By Definition 24-(4), CutSat++ with a strictly-two-layered strategy cannot remove constraints from an unguarded resolvent R except with the rules Solve-Div-Right and Solve-Div-Left. Through the equivalence proven for div-solve(x, I_1, I_2) (see Jovanović and de Moura (2013)), we know that there exist $I'_1, I'_2 \in C'$ such that $\{I'_1, I'_2\} \equiv \{I_1, I_2\}$ and $R \subseteq (C' \setminus \{I'_1, I'_2\}) \cup \{I_1, I_2\}$. We use this equivalence to construct $R' = (R \setminus \{I_1, I_2\}) \cup \{I'_1, I'_2\}$ such that $R' \to R$. Since $R' \to R$ and $R \to \exists x : C''$, R' is also an unguarded resolvent of (x, C'') such that $R' \to \exists x : C''$. Furthermore, R' is a subset of C', which contradicts our initial assumption! □

Together with Lemma 22, this property implies that Resolve-Weak-Cooper is applied at most once to every conflicting core. This is essential for our termination proof.

6.1. Termination

For the termination proof of CutSat++, we consider a (possibly infinite) sequence of rule applications $\langle [\![]\!], C_0\rangle = S_0 \Longrightarrow_{\text{CS++}} S_1 \Longrightarrow_{\text{CS++}} \dots$ 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 $S_0 \Longrightarrow_{\text{CS++}} \dots \Longrightarrow_{\text{CS++}} S_s$, such that there is no further application of the rules Slack-Intro and Forget after state S_s:

Lemma 30 (Slacking-Phase Termination)**.** *Let $\langle [\![]\!], C_0\rangle = S_0 \Longrightarrow_{CS++} S_1 \Longrightarrow_{CS++} \dots$ be a sequence of rule applications applied to a problem C_0 following the strictly-two-layered strategy. Then the sequence reaches a state S_s ($s \in \mathbb{N}_0^+$), after at most finitely many rule applications $S_0 \Longrightarrow_{CS++} \dots \Longrightarrow_{CS++} S_s$, such that there is no further application of the rules Slack-Intro and Forget after state S_s.*

Proof. 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 at most finitely often applicable. □

Next, the sequence reaches a state S_w ($w \geq s$), after a finite derivation of rule applications $S_s \Longrightarrow_{\text{CS++}} \ldots \Longrightarrow_{\text{CS++}} S_w$, such that there is no further application of the rules Resolve-Weak-Cooper, Solve-Div-Left, and Solve-Div-Right after state S_w: The rules Resolve-Weak-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 $S_s \Longrightarrow_{\text{CS++}} \ldots \Longrightarrow_{\text{CS++}} S_w$ contains no application of the rule Slack-Intro. By Lemma 22, an application of Resolve-Weak-Cooper to the conflicting core (x, C') prevents any further applications of Resolve-Weak-Cooper to the same core. By Definition 19, the constraints learned through an application of Resolve-Weak-Cooper contain only variables y such that $y \prec x$. Therefore, an application of Resolve-Weak-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-Weak-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-Weak-Cooper, Solve-Div-Left, and Solve-Div-Right are applicable at most finitely often.

Lemma 31 (Unguarded-Resolution-Phase Termination)**.** *Let $\langle [\![]\!], C_0\rangle = S_0 \Longrightarrow_{CS++} S_1 \Longrightarrow_{CS++} \ldots$ be a sequence of rule applications applied to a problem C_0 following the strictly-two-layered strategy. Then the sequence reaches a state S_w, after finitely many rule applications $S_0 \Longrightarrow_{CS++} \ldots \Longrightarrow_{CS++} S_w$, such that there is no further application of the rules Resolve-Weak-Cooper, Solve-Div-Left, and Solve-Div-Right after state S_w.*

Proof. By Lemma 30, we assume, w.l.o.g., that the sequence continues from a state S_s such that S_s is reached by the sequence after at most finitely many rule applications $S_0 \Longrightarrow_{\text{CS++}} \ldots \Longrightarrow_{\text{CS++}} S_s$ and that there is no further application of the rules Slack-Intro and Forget after state S_s. Let $x_1 \prec \ldots \prec x_n$ be the order of variables for all unguarded variables x_i. We consider a weight vector that strictly decreases after every call to Resolve-Weak-Cooper, Solve-Div-Left, and Solve-Div-Right. For this weight vector, we define $\text{cores}(x_i, C)$ as the set of potential conflicting cores in the problem C with conflicting variable x_i. We also define a subset $\text{woSR}(x_i, C)$ of $\text{cores}(x_i, C)$ so it contains all potential conflicting cores within $\text{cores}(x_i, C)$ that do not have an unguarded resolvent $R \subseteq C$. It is easy to see that $|\text{cores}(x_i, C)| \leq 2^{|C|}$ and, therefore, both functions define finite sets. Now we define the weight vector $\text{weight}_c(S)$ for every state $S = \langle M, C, I\rangle$:

$$\text{weight}_c(S) = (|\text{cores}(x_n, C)|, |\text{woSR}(x_n, C)|, \ldots, |\text{cores}(x_1, C)|, |\text{woSR}(x_1, C)|)$$

We order the two weight_c vectors of two subsequent search-states with the well-founded lexicographic order $>_{\text{lex}}$ based on the well-founded order $>$.

By Definition 24, Conflict(-Div) is only applicable to guarded constraints and guarded variables are only propagated through guarded constraints. Therefore, the conflict I in a conflict state

$\langle M, C, I\rangle$ stays always guarded—even after an application of the Resolve rule—and Learn is only applicable to guarded constraints. Therefore, Resolve-Weak-Cooper, Solve-Div-Left, and Solve-Div-Right are the only rules potentially learning unguarded constraints and, thereby, the only rules that can increase $|\operatorname{cores}(x_i, C)|$ and $|\operatorname{woSR}(x_i, C)|$ between two subsequent states $S_i \Longrightarrow_{\text{CS++}} S_{i+1}$. After all other transitions $S_i \Longrightarrow_{\text{CS++}} S_{i+1}$, it holds that $\text{weight}_c(S_i) \geq_{\text{lex}} \text{weight}_c(S_{i+1})$. Whenever CutSat++ applies Solve-Div-Left, Solve-Div-Right, or Resolve-Weak-Cooper, then the weight vector strictly decreases, i.e., $\text{weight}_c(S') >_{\text{lex}} \text{weight}_c(S)$:

1. By Lemma 22, an application of Resolve-Weak-Cooper to conflicting core (x_i, C^*) implies that there is no unguarded resolvent $R' \subseteq C'$ for (x_i, C^*). By Lemma 21, the new problem $C = C' \cup R$ contains an unguarded resolvent R for (x_i, C^*). Therefore, $|\operatorname{woSR}(x_i, C')| > |\operatorname{woSR}(x_i, C)|$. By Definition 20, it holds for all $y \in \operatorname{vars}(R)$ that $y \prec x$. Hence, Resolve-Weak-Cooper has not introduced new potential conflicting cores (x_j, C^{**}) with $j \geq i$ and $|\operatorname{cores}(x_j, C')| \geq |\operatorname{cores}(x_j, C)|$ for all $j \geq i$. By Lemma 29, $|\operatorname{woSR}(x_j, C')| \geq |\operatorname{woSR}(x_j, C)|$ for all $j > i$. Therefore, the weight decreases after an application of Resolve-Weak-Cooper, i.e., $\text{weight}_c(S') >_{\text{lex}} \text{weight}_c(S)$.

2. Let Solve-Div-Left (Solve-Div-Right) be applied to the pair of divisibility constraints (I_1, I_2) such that $\operatorname{top}(I_1) = x_i$ and $\text{div-solve}(x_i, I_1, I_2) = (I_1', I_2')$. The new constraint set is $C = C' \setminus \{I_1, I_2\} \cup \{I_1', I_2'\}$. The number of potential conflicting cores is independent of the actual divisibility constraints in C'. Only the number of divisibility constraints with $top(I) = x$ is important. Therefore, removing I_1 and replacing it with I_1' doesn't increase the number of cores, i.e., $|\operatorname{cores}(x_i, C' \setminus \{I_1\} \cup \{I_1'\})| = |\operatorname{cores}(x_i, C')|$. We will, however, decrease the number of conflicting cores in x_i, i.e., $|\operatorname{cores}(x_i, C')| > |\operatorname{cores}(x_i, C)|$, because we replace I_2 with I_2' where $\operatorname{top}(I_2') \prec x_i$. Since all variables y in I_1' and I_2' are smaller than or equal to x_i, we do not introduce any new conflicting cores for x_j with $j > i$; thus, $|\operatorname{cores}(x_j, C')| = |\operatorname{cores}(x_j, C)|$ for $j > i$. Finally, Lemma 29 implies that $|\operatorname{woSR}(x_j, C')| \geq |\operatorname{woSR}(x_j, C)|$ for $j > i$. Therefore, $\text{weight}_c(C') >_{\text{lex}} \text{weight}_c(C)$.

This proves already that the weight_c vector monotonically decreases with respect to the order $>_{\text{lex}}$ if we continue from the before mentioned state S_s. Moreover, the set of weight vectors has a minimum $(0, \ldots, 0)$ because no set can contain less than zero elements. As $>_{\text{lex}}$ is well-founded, there exists no way to decrease the weight $\text{weight}_c(C_s)$ without reaching the minimum $(0, \ldots, 0)$ after finitely many applications of the rules Solve-Div-Left, Solve-Div-Right, or Resolve-Weak-Cooper. Finally, the weight_c vector cannot decrease below $(0, \ldots, 0)$; hence, CutSat++ is not able to apply Solve-Div-Left, Solve-Div-Right, or Resolve-Weak-Cooper after we reach a state S with $\text{weight}_c(S) = (0, \ldots, 0)$. We conclude that the rules Solve-Div-Left, Solve-Div-Right, and Resolve-Weak-Cooper are at most finitely often applicable. □

Next, the sequence reaches a state S_b $(b \geq w)$, after a finite derivation of rule applications $S_w \Longrightarrow_{\text{CS++}} \ldots \Longrightarrow_{\text{CS++}} S_b$, such that the bounds remain invariant for every guarded variable x, i.e., $\operatorname{lower}(x, M_b) = \operatorname{lower}(x, M_j)$ and $\operatorname{upper}(x, M_b) = \operatorname{upper}(x, M_j)$ for every state $S_j = \langle M_j, C_j, I_j\rangle$ after $S_b = \langle M_b, C_b, I_b\rangle$ $(j \geq b)$. CUISAT++ reaches such a state because the strictly-two-layered strategy guarantees that unguarded variables do not influence guarded variables. Due to this independence, it is easy to extend the proof for the completely guarded case (see Jovanović and de Moura (2013)) so it also proves that CutSat++ has to reach a state S_b. By definition of S_b, we now also know that the sequence after S_b contains no further propagations, decisions, or conflict resolutions for the guarded variables.

Lemma 32 (Guarded-Search-Phase Termination)**.** *Let* $\langle [\![]\!], C_0\rangle = S_0 \Longrightarrow_{CS++} S_1 \Longrightarrow_{CS++} \ldots$ *be a sequence of rule applications applied to a problem* C_0 *following the strictly-two-layered*

strategy. Then the sequence reaches a state S_b, after finitely many rule applications $S_0 \Longrightarrow_{CS++} \ldots \Longrightarrow_{CS++} S_b$, such that the bounds remain invariant for every guarded variable x.

Proof. This proof is based on the termination proof for CutSat on finite problems, i.e., problems without unguarded variables (see Jovanović and de Moura (2013)). The proof uses a weight function that strictly decreases whenever CutSat++ changes a bound for a guarded variable and otherwise stays the same. By Lemmas 30 and 31, we assume, w.l.o.g., that the sequence continues from a state S_w such that S_w is reached by the sequence after at most finitely many rule applications $S_0 \Longrightarrow_{\text{CS++}} \ldots \Longrightarrow_{\text{CS++}} S_w$, and there is no further application of the rules Slack-Intro, Forget, Resolve-Weak-Cooper, Solve-Div-Left, and Solve-Div-Right after state S_w. The level_B of a state $S = \langle M, C\rangle$ is the number of decisions for guarded variables in M. The maximal prefix of M containing only j decisions for guarded variables is denoted by $\text{B-subseq}_j(M) = M_j$. Since CutSat++ follows a reasonable strategy, it prefers to propagate constraints of the form $\pm x - b \leq 0$. This allows us to assume, w.l.o.g., that M_0 contains both a lower and upper bound for all guarded variables x. The *guarded weight* of the j-th level_B is defined by the function $w_B(M_j)$:

$$w_B(M_j) = \sum_{x \text{ is guarded}} \Big(\text{upper}(x, M_j) - \text{lower}(x, M_j)\Big).$$

The *guarded weight* of a state is the vector:

$$\text{weight}_B(\langle M, C\rangle) = \langle w_B(\text{B-subseq}_0(M)), \cdots, w_B(\text{B-subseq}_n(M))\rangle,$$

where n is the number of guarded variables. We order the two weight_B-vectors of two subsequent search-states with the well-founded lexicographic order $>_{\text{lex}}$ based on the well-founded order $>$. It is easy to see that the minimum of weight_B is $(0, \ldots, 0)$ and that any change to a bound of a guarded variable changes weight_B. Furthermore, by the definition of the strictly-two-layered strategy, we see that we only propagate guarded variables with guarded constraints. Thus, the strategy also implies that the conflict rules—Conflict, Conflict-Div, Backjump, Resolve, Skip-Decision, Unsat, and Learn—only handle guarded constraints. Given the proof for Theorem 2 by Jovanović and de Moura (2013), we see that every application of Propagate, Propagate-Div, and Decide applied to a guarded variable decreases weight_B strictly. We see in the same proof that weight_B strictly decreases between one application of Conflict(-Div) and Backjump as long as the conflict rules handle only guarded constraints—as is the case for CutSat++. Since the bound sequence M is finite, the conflict rules are at most $|M|$ times applicable between one application of Conflict(-Div), and Backjump or Unsat. The remaining rules—Propagate, Propagate-Div, and Decide—applied to unguarded variables, have no influence on weight_B or the bounds of guarded variables. Since weight_B cannot decrease below $(0, \ldots, 0)$, we conclude that CutSat++ is not able to change the bounds for guarded variables infinitely often. □

Next, the sequence reaches a state S_u ($u \geq b$), after a finite derivation of rule applications $S_b \Longrightarrow_{\text{CS++}} \ldots \Longrightarrow_{\text{CS++}} S_u$, such that the bounds also remain invariant for every unguarded variable x, 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 $S_j = \langle M_j, C_j, I_j\rangle$ after $S_u = \langle M_u, C_b, I_u\rangle$ ($j \geq u$). After S_b, CutSat++ propagates and decides only unguarded variables or ends with an application of Sat or Unsat(-Div). These claims are facts because CutSat++ employs a 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.

Lemma 33 (Unguarded-Extension-Phase Termination)**.** *Let $\langle [\![]\!], C_0\rangle = S_0 \Longrightarrow_{CS++} S_1 \Longrightarrow_{CS++} \dots$ be a sequence of rule applications applied to a problem C_0 following the strictly-two-layered strategy. Then the sequence reaches a state S_u, after finitely many rule applications $S_0 \Longrightarrow_{CS++} \dots \Longrightarrow_{CS++} S_u$, such that the bounds remain invariant for every unguarded variable x.*

Proof. By Lemmas 30, 31, and 32, we assume, w.l.o.g., that the sequence continues from a state $S_b = \langle M_b, C_b, I_b\rangle$ such that S_b is reached by the sequence after at most finitely many rule applications $S_0 \Longrightarrow_{\text{CS++}} \dots \Longrightarrow_{\text{CS++}} S_b$ and only the rules Sat, Unsat-Div, Propagate, Propagate-Div, and Decide are applied after S_b, but only for unguarded variables. Assume for a contradiction that there exists an infinite CutSat++ run starting in S_b. Since there is only a finite number of unguarded variables and no rule to undo a decision, the Decide rule is applied at most finitely often. Furthermore, any application of Sat or Unsat-Div ends the sequence making it finite. Thus, we assume, w.l.o.g., that there is no application to the rules Sat, Unsat-Div, and Decide in the infinite run starting in the state S_b.

Since there are at most finitely many variables in state S_b and no rule to introduce further variables after S_b, there exists a smallest unguarded variable x that is propagated infinitely often. We assume, w.l.o.g., that the run starting in S_b propagates only variables y bigger than or equal to x. Therefore, the bounds of all variables z smaller than x remain invariant in all subsequent states $S_j = \langle M_j, C_j, I_j\rangle$ of S_b, i.e., $\text{lower}(z, M_j) = \text{lower}(z, M_b)$ and $\text{upper}(z, M_j) = \text{upper}(z, M_b)$. Since there exists no applicable rule that changes the constraint set, we notice that the constraint set C_b also remains invariant for all states after S_b. Thus, we find all constraints C^* that will be used to propagate x in the set C_b. Since CutSat++ is eager top-level propagating, any constraint $I \in C^*$ has x as their top variable. This leads us to the deduction that $\text{bound}(I, x, M_j) = \text{bound}(I, x, M_b)$ for all subsequent states $S_j = \langle M_j, C_j\rangle$ and inequalities $I \in C^*$. Since the bounds defined by the inequalities in C^* remain invariant after state S_b, CutSat++ propagates x at most finitely often with inequalities.

Therefore, there exists an infinite CutSat++ run only if x is propagated infinitely often with Propagate-Div. We assume, w.l.o.g., that the run starting in S_b propagates x only with Propagate-Div. Next, we deduce that variable x stays unbounded in the remaining states of the derivation sequence. Otherwise, there exists a finite set $x \in \{l_x, \dots, u_x\}$ bounding x and, therefore, allowing only finitely many propagations. In the case that x stays unbounded, the definition of the eager top-level propagating strategy states that Propagate-Div is only applicable to x if $I_d = d \mid ax+p \in C^*$ is the only divisibility constraint in C_b with x as their top variable. Furthermore, we know because of Definition 23 and Lemma 15 that there must exist $v \in \mathbb{Z}$ such that $d \mid av+\text{lower}(p, M_k)$ is satisfied. This—together with the fact that all propagated bounds are implied by the partial model and improve the partial model—proves that Propagate-Div can only be applied finitely often. More specifically, if the lower bound of x is $\text{lower}(x, M_b) = l_x \neq -\infty$ then Propagate-Div propagates for x at most $v - l_x$ lower bounds. If the upper bound of x is $\text{upper}(x, M_b) = u_x \neq \infty$ then Propagate-Div propagates for x at most $u_x - v$ upper bounds. This contradicts the assumption that x is the smallest variable propagated infinitely often, which in turn contradicts our initial assumption! □

After state S_u, only the rules Sat, Unsat, and Unsat-Div are applicable, which lead to a final state. Hence, the sequence $S_0 \Longrightarrow_{\text{CS++}} S_1 \Longrightarrow_{\text{CS++}} \dots$ is finite. We conclude that CutSat++

always terminates:

Theorem 34 (CutSat++ Termination)**.** *If* CutSat++ *starts from an initial state* $\langle [\![]\!], C_0 \rangle$*, then there is no infinite derivation sequence.*

Proof. By Lemmas 30, 31, 32, and 33, CutSat++ reaches a state S_u after which only the rules Sat, Unsat, and Unsat-Div are applicable, which lead to a final state. Therefore, CutSat++ does not diverge. □

6.2. Stuck States

Our CutSat++ calculus never reaches a stuck state. Let x be the smallest unfixed variable with respect to $\prec$. We can propagate a constraint $\pm x - b \leq 0 \in C$ and then fix x by introducing a decision whenever x is guarded. 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 unguarded conflict resolution rules. Therefore, CutSat++ has always a rule applicable unless a final state is reached.

The proof outlined above works because all unguarded conflicts encountered by CutSat++ are either the result of multiple contradicting divisibility constraints resolvable by Solve-Div-Left and Solve-Div-Right, or the conflict is expressible via a conflicting core. Since conflicting cores are only defined over constraints and propagated bounds, we have to guarantee that CutSat++ never encounters an unguarded conflict I where $x = \text{top}(I)$ is fixed with a decided bound. We express this property with the following invariant fulfilled by every state visited by CutSat++:

Definition 35 (Eager Top-Level Propagated States)**.** *A state* $S = \langle M, C, I \rangle$ *is called* eager top-level propagated *if it holds for all unguarded variables* x*, all decided bounds* $\gamma = x \bowtie b$ *(*$\bowtie \in \{\leq, \geq\}$*) in* $M = [\![M', \gamma, M'']\!]$*, and all constraints* $J \in C$ *with* $\text{top}(J) = x$ *that: (1) all other variables contained in* J *are fixed in* M' *and (2)* J *is no conflict in* S*.*

Lemma 36 (Eager Top-Level Stability)**.** *If* S' *is an eager top-level propagated state (Definition 35), then any successor state* $S = \langle M, C, I \rangle$ *reachable by* CutSat++ *is eager top-level propagated.*

Proof. Let S' be an eager top-level propagated state and S its successor, i.e., $S' \Longrightarrow_{\text{CS++}} S$. We prove this Lemma with a case distinction on the rule leading to the above transition:

1. Let the applied rule be Propagate(-Div). Then $S' = \langle M', C' \rangle$ and $S = \langle [\![M', x \bowtie_J b]\!], C' \rangle$, where $\bowtie \in \{\leq, \geq\}$. Let $J' \in C'$ be the constraint used for propagation. Then J' fulfills the properties improves(J', x, M'), bound$(J', x, M') = b$ and $J = \text{tight}(J', x, M')$ (or $J = \text{div-derive}(J', x, M')$). Let the unguarded variable y be fixed by a decided bound γ in $M' = [\![M'', \gamma, M''']\!]$. Let $I \in C'$ be a constraint with $\text{top}(I) = y$. Since S' is eager top-level propagated, all variables in I are fixed in M' and M''. The variable x is not fixed in M' because the predicate improves(J', x, M') must be true for Propagate(-Div) to be applicable. Therefore, x is not contained in I and I is still no conflict in S. Furthermore, all variables in I are still fixed in $[\![M', x \bowtie_J b]\!]$. We conclude that S is eager top-level propagated.

2. Let the applied rule be Decide. Then $S' = \langle M', C' \rangle$ and $S = \langle [\![M', x \bowtie b]\!], C' \rangle$, where $\bowtie \in \{\leq, \geq\}$. We will use the eager top-level propagating strategy (Definition 23) to prove that S is an eager top-level propagated successor state. We consider all unguarded variables y decided in S' by a decided bound γ. Let $I \in C'$ be a constraint with $\text{top}(I) = y$. The bound γ is part of

M', i.e., $M' = [\![M'', \gamma, M''']\!]$. As S' is eager top-level propagated, all other variables contained in I are fixed in M' and M''. Since $\text{lower}(x, M') < \text{upper}(x, M')$ is a condition of the Decide rule, the variable x is not fixed in M'. Therefore, x is not contained in I ($\text{top}(I) = y \prec x$), and I is still no conflict in S. Furthermore, all variables in I are still fixed in $[\![M', x \bowtie_J b]\!]$. Next, we prove that S is eager top-level propagated although variable x is newly decided. Considering Definition 23-(2a) we see that Definition 35-(1) is fulfilled. Similarly, Definition 23-(2b) enforces Definition 35-(2). We conclude that S is eager top-level propagated.

3. Let the applied rule be Unsat(-Div) or Sat. Then the successor state S is neither a search- or conflict-state. The Lemma is thereby trivially fulfilled.

4. Let the applied rule be Forget. Then $S' = \langle M', C' \cup \{J\}\rangle$ and $S = \langle M', C'\rangle$. Therefore, any conflict $I \in C'$ and any decided bound in S is also contained in S'. We conclude that S is eager top-level propagated.

5. Let the applied rule be Slack-Intro. Then $S' = \langle M', C'\rangle$, x is a stuck variable in S' and $S = \langle M', C' \cup \{-x_S \leq 0, x - x_S \leq 0, -x - x_S \leq 0\}\rangle$. Either Slack-Intro was applied before and $-x_S \leq 0 \in C'$ or x_S has $\text{upper}(x_S, M) = \infty$ and $-x_S \leq 0$ is not a conflict in S. Since x was stuck in S', it is not fixed and the top variable in the new constraints $\{x - x_S \leq 0, -x - x_S \leq 0\}$. We conclude that S is eager top-level propagated.

6. Let the applied rule be Resolve-Weak-Cooper. Then $S' = \langle M', C'\rangle$ and $S = \langle M, C' \cup R_c \cup R_k\rangle$. Notice that $M = \text{prefix}(M', y)$ with $y = \min_{I \in R_c}\{\text{top}(I)\}$. Therefore, M is the prefix of M' without decided bounds in variables greater or equal to y. Since $y \preceq x$ for all $I \in R$ and $x = \text{top}(I)$, we deduce that any $I \in R$ that is a conflict has no decided bound for its top variable x in S. Since M is a prefix of M', every conflict $I \in C'$ appearing in state S also appears in state S'. Now it is easy to see that S is eager top-level propagated because S' was eager top-level propagated.

7. Let the applied rule be Solve-Div-Right. Then $S' = \langle M', C' \cup \{I_1, I_2\}\rangle$ and $S = \langle M, C' \cup \{I'_1, I'_2\}\rangle$. We notice that $M = \text{prefix}(M', y)$ with $y = \text{top}(I'_2)$. Therefore, M is the prefix of M' without decided bounds in variables greater or equal to y, which includes especially the variable $x = \text{top}(I_1)$. Thus, neither the top variable of I'_1 nor the top variable I'_2 is fixed by a decision. Since M is a prefix of M', every conflict $I \in C'$ appearing in state S also appears in state S'. Now it is easy to see that S is eager top-level propagated because S' was eager top-level propagated.

8. Let the applied rule be Solve-Div-Left. Then $S' = \langle M', C' \cup \{I_1, I_2\}\rangle$ and $S = \langle M', C' \cup \{I'_1, I'_2\}\rangle$. Since the bound sequence is the same in both states, every conflict $I \in C'$ appearing in state S also appears in state S'. By the definition of the Solve-Div-Left rule, I'_2 is no conflict in state S. Note that div-solve is an equivalence preserving transformation. Thus, if I'_1 were a conflict in S and $\text{top}(I'_1) = x$ fixed by a Decision, then I_1 or I_2 is a conflict in S'. Therefore, I'_1 is no conflict or $\text{top}(I'_1) = x$ has no decided bound. Now it is easy to see that S is eager top-level propagated because S' was eager top-level propagated.

9. Let the applied rule be Conflict or Conflict-Div. Then $S' = \langle M', C'\rangle$ and $S = \langle M', C', I\rangle$, where I is a conflict. It is easy to see that S is eager top-level propagated because S' is eager top-level propagated.

10. Let the applied rule be Resolve or Skip-Decision. Then $S' = \langle [\![M, \gamma]\!], C', J'\rangle$ and $S = \langle M, C', J\rangle$, where J' and J are conflicts in S' and S, respectively. Since M is a prefix of M', every conflict $I \in C'$ appearing in state S also appears in state S'. Now it is easy to see that S is eager top-level propagated because S' was eager top-level propagated.

11. Let the applied rule be Learn. Then $S' = \langle [\![M', \gamma]\!], C', I\rangle$ and $S = \langle M', C' \cup I, I\rangle$, where I is a conflict. Since CutSat++ uses a two-layered strategy (Definition 24), I is a guarded constraint. Now it is easy to see that S is eager top-level propagated because S' was eager

top-level propagated.

12. Let the applied rule be Backjump. Then $S' = \langle [\![M', \gamma, M'']\!], C', I \rangle$ and $S = \langle [\![M', \gamma']\!], C' \rangle$, where I is a conflict in S'. Since CutSat++ uses a two-layered strategy (Definition 24), I is a guarded constraint. Now it is easy to see that S is eager top-level propagated because S' was eager top-level propagated. □

Since the initial state $\langle [\![]\!], C_0 \rangle$ trivially fulfils the eager top-level propagated properties, it is clear that CutSat++ produces only eager top-level states; except for the final states. The eager top-level propagated property is so important because we will use it to show that CutSat++ resolves any conflict it encounters. In case the conflict is a guarded constraint, this is done with the CDCL based conflict rules. Otherwise, the conflict I is an unguarded constraint and CutSat++ simulates weak Cooper elimination with the unguarded conflict resolution rules. First, we use Solve-Div-Left to simulate the algorithm in Figure 3. This either ends with a call to Solve-Div-Right resolving the conflict or CutSat++ finds a conflicting core. Then the conflicting core is resolved with the rule Resolve-Weak-Cooper.

Lemma 37 (Conflicts Progress)**.** *Let $S = \langle M, C \rangle$ be a state reachable by* CutSat++*. Let $I \in C$ be a conflict in state S. Then state S is not stuck.*

Proof. Assume for a contradiction that state S is stuck. W.l.o.g., we assume that $x = \text{top}(I)$ is the smallest variable in our order that is the top variable in a conflicting constraint $I' \in C$. If x is a guarded variable, then Conflict or Conflict-Div is applicable, which contradicts our initial assumption! Therefore, x is an unguarded variable. Furthermore, all variables y smaller than x are fixed. Otherwise, we deduce for the smallest unfixed variable y that either

- y is stuck and Slack-Intro is applicable
- Propagate is applicable to a constraint I' where $\text{top}(I') = y$
- C contains at least two divisibility constraints I_1, I_2 that have y as their top variable and Solve-Div-Left or Solve-Div-Right is applicable
- S contains a diophantine conflicting core (y, I_d) and Resolve-Weak-Cooper is applicable
- Decide is applicable to y because all conditions in Definition 23-(2) are fulfilled

Since S is eager top-level propagated and I is a conflict with top variable x, we know that state S contains no decided bound for x (Definition 35 and Lemma 36). W.l.o.g., we assume that C contains at most one divisibility constraint I_d with x as its top variable. Otherwise, Solve-Div-Left or Solve-Div-Right are applicable, which contradicts our initial assumption! Let $x \geq b_l$ be the strictest lower bound $b_l = \text{bound}(x, I_l, M)$ for an inequality $I_l \in C$ with top variable x or $b_l = -\infty$ if there is no inequality propagating a lower bound. Let $x \leq b_u$ be the strictest upper bound $b_u = \text{bound}(x, I_u, M)$ for an inequality $I_u \in C$ with top variable x or $b_u = \infty$ if there is no inequality propagating an upper bound. Since the strictly-two-layered strategy forbids the application of Forget to unguarded constraints, CutSat++ never removes an unguarded inequality. Furthermore, any bound $x \bowtie b$ ($\bowtie \in \{\leq, \geq\}$) propagated from a divisibility constraint requires another bound $x \bowtie b'$ propagated from an inequality. We deduce that $b_u \neq \infty$ if $\text{upper}(x, M) \neq \infty$ and $b_l \neq -\infty$ if $\text{lower}(x, M) \neq -\infty$. Next, we do a case distinction on whether the bounds b_u and b_l are finite:

1. Let $b_u = \infty$ and $b_l = -\infty$. Then it holds for all inequalities $ax + p \leq 0$ that lower($ax + p$) $= -\infty$. Thus, the conflict I is no inequality. A divisibility constraint is a conflict only if lower$(x, M) \neq -\infty$ and upper$(x, M) \neq \infty$. This contradicts the assumption that I is a conflict.

2. Let $b_u = \infty$ and $b_l \in \mathbb{Z}$. Then it holds for all inequalities $ax + p \leq 0$ with $a < 0$ that lower$(ax + p) = -\infty$. Thus, the conflict I is no inequality. A divisibility constraint is a conflict only if lower$(x, M) \neq -\infty$ and upper$(x, M) \neq \infty$. This contradicts the assumption that I is a conflict.

3. Let $b_l = -\infty$ and $b_u \in \mathbb{Z}$. Then it holds for all inequalities $ax + p \leq 0$ with $a > 0$ that lower$(ax + p) = -\infty$. Thus, the conflict I is no inequality. A divisibility constraint is a conflict only if lower$(x, M) \neq -\infty$ and upper$(x, M) \neq \infty$. This contradicts the assumption that I is a conflict.

4. Let $b_u < b_l$. Then $(x, \{I_l, I_u\})$ is a conflicting core and Resolve-Weak-Cooper is applicable. This contradicts the assumption that no rule is applicable.

5. Let $\{b_l, \ldots, b_u\} \neq \emptyset$. Then I is not an inequality. If $(x, \{I_l, I_u, I_d\})$ is a conflicting core, then Resolve-Weak-Cooper is applicable contradicting our initial assumption. Therefore, there exists a solution $b_d \in \{b_l, \ldots, b_u\}$ for x satisfying I_d. Let D be the set of divisibility constraints used to propagate a bound for x in M. All constraints $D' \subseteq D$ not contained in C, i.e., $D' = D \setminus C = D \setminus \{I_d\}$, were eliminated with div-solve. It is easy to see that there exists a set of constraints $D^* = D^{**} \cup \{I_d\}$ contained in C that implies satisfiability of D:

$$D^* = D^{**} \cup \{I_d\} \rightarrow D,$$

and D^{**} contains only variables y smaller than x (Proof the same as for Lemma 29). In state S, the set of divisibility constraints D^* is fixed and satisfied under the partial assignment of M. Otherwise, S would contain a conflict $I' \in D^* \subseteq C$ with top$(I') \prec x$, which contradicts our initial assumption! Thus, setting x to the solution $b_d \in \{b_l, \ldots, b_u\}$ satisfies I_d in M and also $D \cup \{I_d\}$. Furthermore, all propagated constraints are satisfied if x is set to b_d:

$$\text{lower}(x, M) \leq b_d \leq \text{upper}(x, M).$$

This contradicts the assumption that there exists a conflict I with top$(I) = x$. □

The remainder of the proof follows directly the proof outline from above:

Theorem 38 (CUTSAT++ Progress)**.** *Let $S = \langle M, C\rangle$ be a state reachable by* CutSat++*. Then S is not stuck.*

Proof. Assume for a contradiction that $S = \langle M, C\rangle$ is a stuck state. CutSat++ can propagate at least two bounds for every guarded variable and afterwards use decided bounds to fix them. Therefore, we assume that all guarded variables are fixed. By Lemma 37, there is no conflict in state S. Since there is no conflict, at least one variable is unfixed or rule Sat would be applicable. Therefore, there must exist a smallest unfixed and unguarded variable x. With the Slack-Intro rules CutSat++ introduces for all variables at least one lower or upper bound. Therefore, there exists a violation to the conditions in Definition 23-(2) or Decide would be applicable to x. Since x is the smallest unfixed variable, the condition in Definition 23-(2a) holds. Definition 23-(2c) is also easy to satisfy by applications of Solve-Div-Left and Solve-Div-Right. Therefore, Definition 23-(2b) is violated. Thus, there exists a constraint $I \in C$ that is a conflict in $S' = \langle [\![M, \gamma]\!], C\rangle$, where γ is a decided bound in x and $x = \text{top}(I)$. By Lemma 37, it is not possible that $I \in C$ is also a conflict in S or S would not be stuck. Finally, I is a conflict only in S'

and not S if Propagate(-Div) is applicable to I. With Solve-Div-Left and Solve-Div-Right it is relatively easy to fulfil the conditions for Definition 23-(1) and, therefore, Propagate(-Div) is applicable. We conclude that CutSat++ has always one applicable rule, which is a contradiction to our assumption! □

6.3. Completeness

All CutSat++ rules are sound, i.e., if $\langle M_i, C_i, I_i\rangle \Longrightarrow_{\text{CS++}} \langle M_j, C_j, I_j\rangle$, then any satisfiable assignment ν for C_j is also a satisfiable assignment for C_i. The rule Resolve-Weak-Cooper is sound because of the Lemmas 14 and 15. 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 by Jovanović and de Moura (2013).

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

Theorem 39 (CUTSAT++ Completeness)**.** *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* $\langle \nu, \text{sat}\rangle$ *where* ν *is a satisfiable assignment for* C_0*.*

Proof. By Theorem 34, CutSat++ is terminating. By Jovanović and de Moura (2013) and the Lemmas 14, and 15, CutSat++ is sound. By Theorem 38, CutSat++ never reaches a stuck state. Since CutSat++ is terminating and never reaches a stuck state, every application of CutSat++ ends via the rules Sat, Unsat, or Unsat-Div in one of the final states. The rule Sat is only applicable in a state $\langle M, C\rangle$ where $\nu[M]$ satisfies C and because of soundness also C_0. The rules Unsat and Unsat-Div are only applicable to states $\langle M, C, I\rangle$ where the constraint set C contains a trivially unsatisfiable constraint. When CutSat++ encounters a trivially unsatisfiable constraint, then the soundness of the CutSat++ rules guarantees that C_0 is unsatisfiable. □

7. Conclusion and Future Work

The starting point of our work was an implementation of the CutSat (see Jovanović and de Moura (2013)) calculus as a theory solver for hierarchic superposition (see Fietzke and Weidenbach (2012)). In that course, we observed divergence for some of our problems. The analysis of those divergences led to the development of the CutSat++ calculus presented in this paper, which is, as far as we know, the first sound, complete, and terminating calculus for linear integer problems based on the model assumption, conflict, learning approach motivated by CDCL style SAT solving.

There is a reasonable gap between the CutSat++ calculus and an efficient implementation. For example, the efficiency of CDCL-based SAT relies on a tight connection between the heuristics for picking propositional decision variables and the learned clauses. For CutSat++ this corresponds to the connection between picking a variable for a decision or slack introduction, see Section 2.3, and the newly added inequalities. Although some basic heuristics have been considered in the CutSat (see Jovanović and de Moura (2013)) implementation, they are not yet as mature as the respective CDCL heuristics. In addition, when CutSat++ selects a variable for slack introduction, e.g., see Figure 6, the afterwards propagated bound is always with respect to the integer constant 0. The rule slack intro can be modified to yield an afterwards propagated bound for any integer constant. From the branch-and-bound based approaches to solving linear

integer problems it is well-known that always choosing bound 0 for branching is not a good heuristic, in general. The experiments from (see Jovanović and de Moura (2013)) show that the efficiency of CutSat is in particular superior to branch-and-bound based approaches if the computation of the rational relaxation becomes expensive. So using a rational relaxation for determining the bound propagated after slack intro, may not be the best solution. We need a better understanding of heuristics fitting CutSat++. For example, heuristics motivated by quantifier elimination procedures may yield to further insights.

Another dimension is the variable order used in CutSat++. The ordering is crucial for termination. However, it is an a priori fixed ordering. If for some problem a best single ordering exists and if it is known, then this is the perfect choice. For example, in superposition-based first-order theorem proving, also based on model assumptions and an a priori ordering, see Section 1, for many decidable fragments of first-order logic a priori orderings are known that guarantee termination of the superposition calculus on the fragment. However, it seems that the lazy, dynamic and problem driven way CDCL-based SAT solving develops the variable order (see Weidenbach (2015)) is more efficient, in general. It is an open problem to further develop CutSat++ in a way where the ordering becomes dynamic and the calculus still guarantees termination.

Finally, we see great potential in the development of constraint reduction techniques from (weak) Cooper elimination (see Cooper (1972)) but also from quantifier elimination procedures, in general.

Acknowledgment. *This research was supported in part by the German Transregional Collaborative Research Center SFB/TR 14 AVACS, by the ANR/DFG Programme Blanc project STU 482/2-1 SMArT, and by the European Union's Horizon 2020 research and innovation programme under grant agreement No H2020-FETOPEN-2015-CSA 712689.*

References

Alagi, G., Weidenbach, C., 2015. NRCL - A model building approach to the bernays-schönfinkel fragment. In: Lutz, C., Ranise, S. (Eds.), Frontiers of Combining Systems - 10th International Symposium, FroCoS 2015, Wroclaw, Poland, September 21-24, 2015. Proceedings. Vol. 9322 of Lecture Notes in Computer Science. Springer, pp. 69–84.

Bachmair, L., Ganzinger, H., 1990. On restrictions of ordered paramodulation with simplification. In: 10th International Conference on Automated Deduction, CADE-10. Vol. 449 of LNCS. Springer, pp. 427–441.

Bachmair, L., Ganzinger, H., Waldmann, U., 1994. Refutational theorem proving for hierarchic first-order theories. Applicable Algebra in Engineering, Communication and Computing, AAECC 5 (3/4), 193–212.

Barrett, C., Conway, C., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C., 2011. CVC4. In: Gopalakrishnan, G., Qadeer, S. (Eds.), CAV. Vol. 6806 of LNCS. Springer, pp. 171–177.
URL http://dx.doi.org/10.1007/978-3-642-22110-1_14

Barrett, C., Nieuwenhuis, R., Oliveras, A., Tinelli, C., 2006. Splitting on demand in sat modulo theories. In: Hermann, M., Voronkov, A. (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning. Vol. 4246 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 512–526.
URL http://dx.doi.org/10.1007/11916277_35

Baumgartner, P., Waldmann, U., 2013. Hierarchic Superposition with Weak Abstraction. In: Bonacina, M. P. (Ed.), Automated Deduction - CADE-24. Vol. 7898 of Lecture Notes in Artificial Intelligence. Springer, Lake Placid, NY, USA, pp. 39–57.

Bayardo Jr., R. J., Schrag, R., 1996. Using csp look-back techniques to solve exceptionally hard sat instances. In: Freuder, E. C. (Ed.), Proceedings of the Second International Conference on Principles and Practice of Constraint Programming, Cambridge, Massachusetts, USA, August 19-22, 1996. Vol. 1118 of LNCS. Springer, pp. 46–60.

Berman, L., 1977. Precise bounds for Presburger arithmetic and the reals with addition. In: 18th Annual Symposium on Foundations of Computer Science (FOCS 1977), 31 October–2 November, Providence, RI, USA. IEEE, pp. 95–99.

Berman, L., 1980. The complexity of logical theories. Theoretical Computer Science 11 (1), 71 – 77.

Biere, A., Heule, M., van Maaren, H., Walsh, T. (Eds.), 2009. Handbook of Satisfiability. Vol. 185 of Frontiers in Artificial Intelligence and Applications. IOS Press.

Bromberger, M., Sturm, T., Weidenbach, C., 2015. Linear integer arithmetic revisited. In: Felty, A. P., Middeldorp, A. (Eds.), Automated Deduction - CADE-25. Vol. 9195 of Lecture Notes in Computer Science. Springer International Publishing, pp. 623–637.
URL http://dx.doi.org/10.1007/978-3-319-21401-6_42

Caferra, R., Leitsch, A., Peltier, N., 2004. Automated Model Building. Vol. 31 of Applied Logic Series. Kluwer.

Cimatti, A., Griggio, A., Schaafsma, B., Sebastiani, R., 2013. The MathSAT5 SMT Solver. In: Piterman, N., Smolka, S. (Eds.), Proceedings of TACAS. Vol. 7795 of LNCS. Springer.

Cooper, D. C., 1972. Theorem proving in arithmetic without multiplication. In: Meltzer, B., Michie, D. (Eds.), Proceedings of the Seventh Annual Machine Intelligence Workshop, Edinburgh, 1971. Vol. 7 of Machine Intelligence. Edinburgh University Press, pp. 91–99.

de Moura, L., Bjørner, N., 2008. Z3: An efficient SMT solver. In: Ramakrishnan, C., Rehof, J. (Eds.), Tools and Algorithms for the Construction and Analysis of Systems. Vol. 4963 of LNCS. Springer, pp. 337–340.
URL http://dx.doi.org/10.1007/978-3-540-78800-3_24

Dillig, I., Dillig, T., Aiken, A., 2009. Cuts from proofs: A complete and practical technique for solving linear inequalities over integers. In: Bouajjani, A., Maler, O. (Eds.), Computer Aided Verification. Vol. 5643 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 233–247.
URL http://dx.doi.org/10.1007/978-3-642-02658-4_20

Dragan, I., Korovin, K., Kovács, L., Voronkov, A., 2013. Bound propagation for arithmetic reasoning in vampire. In: Bjørner, N., Negru, V., Ida, T., Jebelean, T., Petcu, D., Watt, S. M., Zaharie, D. (Eds.), 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2013, Timisoara, Romania, September 23-26, 2013. IEEE Computer Society, pp. 169–176.
URL http://dx.doi.org/10.1109/SYNASC.2013.30

Dutertre, B., 2014. Yices 2.2. In: Biere, A., Bloem, R. (Eds.), Computer-Aided Verification (CAV'2014). Vol. 8559 of LNCS. Springer.

Dutertre, B., de Moura, L., 2006. A fast linear-arithmetic solver for dpll(t). In: Ball, T., Jones, R. B. (Eds.), Computer Aided Verification. Vol. 4144 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 81–94.
URL http://dx.doi.org/10.1007/11817963_11

Ferrante, J., Rackoff, C. W., March 1975. A decision procedure for the first order theory of real addition with order. SIAM Journal on Computing 4 (1), 69–76.

Ferrante, J., Rackoff, C. W., 1979. The Computational Complexity of Logical Theories. Vol. 718 of LNM. Springer.

Fietzke, A., Weidenbach, C., 2012. Superposition as a decision procedure for timed automata. Mathematics in Computer Science 6 (4), 409–425.

Fischer, M. J., Rabin, M., 1974. Super-exponential complexity of Presburger arithmetic. SIAM-AMS Proceedings 7, 27–41.

Fürer, M., 1982. The complexity of presburger arithmetic with bounded quantifier alternation depth. Theoretical Computer Science 18 (1), 105 – 111.

Grädel, E., 1987. The complexity of subclasses of logical theories. Ph.D. thesis, Universität Basel.

Griggio, A., 2012. A practical approach to satisability modulo linear integer arithmetic. JSAT 8 (1/2), 1–27.

Jovanović, D., de Moura, L., 2013. Cutting to the chase. Journal of Automated Reasoning 51 (1), 79–108.
URL http://dx.doi.org/10.1007/s10817-013-9281-x

Jovanovic, D., de Moura, L. M., 2012. Solving non-linear arithmetic. In: Gramlich, B., Miller, D., Sattler, U. (Eds.), Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings. Vol. 7364 of Lecture Notes in Computer Science. Springer, pp. 339–354.

Jünger, M., Liebling, T. M., Naddef, D., Nemhauser, G. L., Pulleyblank, W. R., Reinelt, G., Rinaldi, G., Wolsey, L. A. (Eds.), 2010. 50 Years of Integer Programming 1958-2008. Springer.

Lasaruk, A., Sturm, T., Dec. 2007. Weak quantifier elimination for the full linear theory of the integers. A uniform generalization of Presburger arithmetic. Applicable Algebra in Engineering, Communication and Computing 18 (6), 545–574.

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., Malik, S., 2001. Chaff: Engineering an efficient sat solver. In: Design Automation Conference, 2001. Proceedings. ACM, pp. 530–535.

Nieuwenhuis, R., Oliveras, A., Tinelli, C., November 2006. Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(t). Journal of the ACM 53, 937–977.

Oppen, D. C., 1978. A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic. J. Comput. Syst. Sci. 16 (3), 323–332.

Papadimitriou, C. H., Oct. 1981. On the complexity of integer programming. J. ACM 28 (4), 765–768.
URL http://doi.acm.org/10.1145/322276.322287

Piskac, R., de Moura, L. M., Bjørner, N., 2010. Deciding effectively propositional logic using DPLL and substitution sets. Journal of Automated Reasoning 44 (4), 401–424.

Presburger, M., 1929. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In: Comptes Rendus du premier congres de Mathematiciens des Pays Slaves. Warsaw, Poland, pp. 92–101.

Silva, J. P. M., Sakallah, K. A., 1996. Grasp - a new search algorithm for satisfiability. In: International Conference on Computer Aided Design, ICCAD. IEEE Computer Society Press, pp. 220–227.

von zur Gathen, J., Sieveking, M., 1978. A bound on solutions of linear integer equalities and inequalities. Proceedings of the AMS 72, 155–158.

Weidenbach, C., 2015. Automated Reasoning Building Blocks. In: Meyer, R., Platzer, A. (Eds.), Correct System Design. Vol. 9360 of Lecture Notes in Computer Science. Springer, Oldenburg, Germany, pp. 172–188, author: Wehrheim, Heike.

Weispfenning, V., Nov. 1990. The complexity of almost linear diophantine problems. Journal of Symbolic Computation 10 (5), 395–403.