Certifying Bounds Propagation for Integer Multiplication Constraints

Matthew J. McIlree, Ciaran McCreesh
University of Glasgow, Scotland (UK)
m.mcilree.1@research.gla.ac.uk, ciaran.mcreesh@glasgow.ac.uk

Abstract

A constraint programming (CP) solver that implements *proof logging* will output a machine-checkable certificate of correctness alongside any result it obtains. This is useful for trusting claims of unsatisfiability or optimality, as well as for debugging and auditing solver implementations. Proofs can be constructed by having the solver log *justifications* for each inference it makes, and previous work has shown that many standard CP reasoning techniques can be efficiently justified using a *pseudo-Boolean* (PB) proof format. This paper extends PB justifications to propagators enforcing *bounds-consistency* on multiplication and division constraints. We show that even though the proof system and checker operate only on linear inequalities over 0-1 variables, non-linear reasoning over bounded domains can be efficiently expressed as a sequence of PB proof steps. Additionally, we demonstrate that bespoke proof logging for bounds-consistency algorithms offers a clear advantage over constructing justifications by brute force.

Code — https://doi.org/10.5281/zenodo.14500848

1 Introduction

Constraint satisfaction problems (CSPs) are expressed in terms of variables; possible *domain* values that can be assigned to each variable; and constraints determining which assignments are simultaneously allowed in a solution. Many important real-world challenges can be elegantly expressed in this way (Wallace 1996; Falkner et al. 2016). The discipline of *constraint programming* (CP) has over the years developed a range of sophisticated automated techniques for finding or determining non-existence of solutions to a CSP, as well as finding optimal solutions with respect to some objective function of the variables (Rossi, van Beek, and Walsh 2006; Lecoutre 2013).

If a CP solver asserts that its input problem is merely *satisfiable*, it can easily "prove" this claim by providing a witness assignment, and this can always be efficiently checked against the original constraints (Schaefer 1978). But if the claim is *unsatisfiability* of the problem, or *optimality* of a particular solution, it can be more challenging to trust it with a high degree of certainty. Combinatorial solvers, as with all complex computer software, are prone to bugs (Akgün et al. 2018; Gillard, Schaus, and Deville 2019; Paxian and Biere 2023; Vanroose et al. 2024).

Adding *proof logging* to an algorithm is a powerful way to increase trust in its answers and detect any incorrect results (McConnell et al. 2011). The idea is that while solving a problem, the program should log proof statements that justify each reasoning step, and these statements in the end form an efficiently checkable certificate of correctness for the output. This approach has seen great success in the field of Boolean satisfiability (SAT) solving (Heule 2021), and more recently has begun to be applied in CP (Veksler and Strichman 2010; Gocht, McCreesh, and Nordström 2022; Flippo et al. 2024).

A key feature differentiating CP from SAT is the use of constraint *propagators*, specialised for a variety of expressive high-level constraints. These reduce variable domains during search by eliminating values that can be inferred infeasible due to a particular constraint. Any proof logging approach for CP must consider how the inferences made by these algorithms can be justified, which is a challenge since they can enforce different levels of *consistency* on the variables and rely on a wide range of reasoning techniques.

We know already that for many important constraint types, *pseudo-Boolean* (PB) proof steps can be used to justify their reasoning (Elffers et al. 2020; McIlree and McCreesh 2023; McIlree, McCreesh, and Nordström 2024). Here, the input problem for the solver is first compiled to a representation involving only 0-1 integer linear inequalities (pseudo-Boolean constraints), and then native CP propagators can log proof steps that make use of rules based on the *cutting planes* proof system (Cook, Coullard, and Turán 1987) to derive justifications. The resulting proofs can then be checked using the *VeriPB* proof checker (Bogaerts et al. 2023b), which has been effectively used for certification in various combinatorial solving paradigms.

1.1 Our Contribution

In this work we show that PB reasoning can also be used for justifying propagation of multiplication constraints: $X \times Y = Z$ for CP variables X, Y, Z over integer domains. This fundamental arithmetic operation appears in a wide variety of problem instances, and can be used as an atomic constraint in the decomposition of arbitrary polynomial restrictions (Apt and Zoeteweij 2004), as well as division and modulo constraints. Aside from certain applications, such as indexing into a multi-dimensional array, where stronger *domain-consistent* pruning is desirable, ternary multiplication propagators generally enforce only *bounds-consistency*. In a bounds-consistent propagator, upper and lower bounds on domains of each variable are narrowed such that, for both bound values, there exist *real numbers* within the bounds of the other two variables where the multiplication relation holds. This corresponds to the *bounds($\mathcal{R}$)* consistency definition given by Choi et al. (2006), which is the standard level of consistency enforced in state-of-the-art open source solver implementations such as *Gecode* (Gecode Team 2024) and *Chuffed* (Chu et al. 2024).

At first, it might appear that a fundamentally linear PB proof system based on cutting planes would be ill-equipped to handle such reasoning. But we are able to show that this is not the case. By using a binary representation of the CP variables and some explicit proofs by contradiction, we can derive justifications for any inference made by a bounds propagator in a number of proof steps proportionate to the product of the numbers of bits required to encode the domains of X and Y. This includes handling negative domain values through the use of a case-based derivation over the signs of the variables.

These justifications are independent of any other reasoning in the proof, and so could be implemented either directly as logging statements in the solver itself (Gocht, McCreesh, and Nordström 2022), or in a later translation step from a higher-level proof format (Flippo et al. 2024).

More broadly, we demonstrate for the first time that having specialised bounds justifications for bounds-consistency propagators is worthwhile. While it is possible to auto-tabulate the solutions to any multiplication constraint and so construct justifications for any domain-consistent inference by "brute force" (Gocht, McCreesh, and Nordström 2022), we show that our method offers a significant advantage over this approach, both in terms of proof logging overhead and verification time.

The remainder of this paper will be organised as follows: after a review of essential preliminaries, we define a straightforward PB encoding of multiplication for positive domains and show how to efficiently derive bounds on the product variable based on bounds of the two multiplicands, relying only on proof rules and the correctness of the inference and *not* on propagation properties of the encoding. Next, we extend this encoding to a representation that allows multiplication of two's complement encoded variables and demonstrate some important statements we can derive from this encoding: this allows us to bring together our results into a complete justification procedure. Finally, we briefly discuss our implementation of the approach, and present some simple benchmarks comparing bounds-consistent proof logging to auto-tabulation proofs.

2 Preliminaries

We will first devote some space to the proof format and background on how PB proof logging for CP works. This is necessarily a brief overview: we refer to Buss and Nordström (2021) for more details on pseudo-Boolean reasoning; Bogaerts et al. (2023a) for the proof system underlying *VeriPB*; and Gocht, McCreesh, and Nordström (2022) for applying PB proof logging to CP.

2.1 Pseudo-Boolean Constraints

For our purposes, a pseudo-Boolean constraint is a linear inequality of the form $\sum_i a_i \ell_i \geq b$, where a_i and b are integer constants, and each ℓ_i is a *literal*: either a 0-1 variable x_i or its negation $\overline{x}_i := 1 - x_i$. When convenient, we will also use $x^1 = x$ and $x^{-1} = \overline{x}$, to denote literals. We can always *normalise* a PB constraint, rearranging so that $a_i \geq 0, b \geq 0$, and all literals are over distinct variables. We call b the *degree* of the constraint in this case. A PB *formula* is a set of PB constraints. The formula alone specifies a constraint satisfaction problem, or an optimisation problem if an objective $\sum_i c_i \ell_i$ to minimise is provided.

Let $C := \sum_i a_i \ell_i \geq b$ be a (normalised) PB constraint. We can obtain a native PB constraint representing its negation $\neg C$ by normalising $-\sum_i a_i \ell_i \geq -b + 1$. Similarly, for a set of literals L we have a native representation of the *reification* of C by the conjunction of literals in L:

$$(\wedge L \Rightarrow C) := \textstyle\sum_{y \in L} b\overline{y} + \sum_i a_i \ell_i \geq b. \qquad (1)$$

For a set of PB constraints we will also write $\wedge L \Rightarrow F$ for the set of reified constraints $\{\wedge L \Rightarrow C : C \in F\}$, and for single literal y we can write $y \Leftrightarrow C$ as a shorthand for the pair of constraints $y \Rightarrow C$ and $\overline{y} \Rightarrow \neg C$.

An *assignment* is a (partial) function from PB variables to $\{0, 1\}$; we extend an assignment ρ from variables to literals in the natural way, $(\rho(\overline{x}) = 1 - \rho(x))$, and for literals ℓ over variables x not in the domain of ρ, we use the convention $\rho(\ell) = \ell$. For notational convenience, we can also view ρ as the set of literals $\{\ell : \rho(\ell) = 1\}$ assigned true by ρ. Applying ρ to C yields

$$C\restriction_\rho := \textstyle\sum_{\ell_i : \rho(\ell_i) = \ell_i} a_i \ell_i \geq A - \sum_{\ell_j \in \rho(\ell_j) = 1} a_j \qquad (2)$$

substituting literals as specified by ρ. We extend this notation to applying assignments to F: $F\restriction_\rho = \bigcup_{C \in F} C\restriction_\rho$.

We say that a PB constraint C *propagates* a literal ℓ under an assignment ρ if $C\restriction_\rho$ cannot be satisfied unless ℓ is true. *Unit propagation* of a PB formula F is then the process of starting with $\rho = \emptyset$, adding any literals propagated by constraints in $F\restriction_\rho$ to ρ, and repeating until either no new literals propagate or a constraint $C\restriction_\rho$ that cannot possibly be satisfied (contradiction) is obtained.

2.2 Pseudo-Boolean Proofs

As stated, the proof system associated with the *VeriPB* proof checker can be used to construct certificates via proof logging for the results of various combinatorial solving algorithms, including CP solvers based on backtracking search and constraint propagation. It requires an input PB formula which must truthfully specify the problem being solved, and then the proof itself is a sequence of steps, where each step derives a PB constraint from the input formula and previously derived constraints via one of several efficiently checkable proof rules. Adding constraints obtained using these rules to the original formula is guaranteed to preserve at least one (optimal) solution. Hence, if a proof concludes by deriving a PB constraint such as $0 \geq 1$ which trivially has no solutions (contradiction), the original problem has been shown to be unsatisfiable. Certificates of optimality can be created by additionally providing a witness assignment that propagates to a complete solution, and then deriving contradiction from a constraint that says that the objective value must be better than the one achieved by the witness. We consider the *length* of such derivations to be the number of proof steps, or equivalently, the number of derived constraints.

We will now briefly discuss the rules required for the present work, and refer the reader to Bogaerts et al. (2023a) and (2023b) for a complete description of the proof system and the associated machine-readable proof logging format. For a given derivation where F is the set of constraints either in the input formula or previously derived, we will use:

1. "Cutting planes" rules — we can derive: $\ell \geq 0$ for any literal ℓ (literal axioms); the sum of two PB constraints in F; a positive scalar multiple of any PB constraint F; and the result of dividing a normalised PB constraint in F through by a scalar and rounding up. The result of these operations is implicitly normalised.
2. Saturation — for any normalised constraint $\sum_i a_i \ell \geq b$ in F we can derive $\sum_i \min(a_i, b)\ell_i \geq b$. This is often viewed as an additional "cutting planes" rule.
3. Reverse Unit Propagation (RUP) — we can derive a constraint C if performing unit propagation on $\neg C \cup F$ results in a contradiction.
4. Explicit Contradiction — we can derive a constraint C if we can provide a subproof deriving contradiction from $\neg C \cup F$ using any of the above rules.
5. Extension variable introduction — for a fresh PB variable x not appearing in any constraint F, and any constraint C, we can always introduce the pair of constraints $x \Rightarrow C$ and $\bar{x} \Rightarrow \neg C$, i.e. $x \Leftrightarrow C$.

Note that items 4 and 5 above are in fact special cases of *VeriPB*'s *redundance-based strengthening* rule, which we do not require in its full generality in this work. We will however, make use of some other known shortcuts for constructing derivations in certain forms.

Theorem 1. (Hooker 1992) For a set of literals L and PB variable x we can derive $\wedge L \Rightarrow 0 \geq 1$ from $\wedge L \Rightarrow x \geq 1$ and $\wedge L \Rightarrow \bar{x} \geq 1$ in $O(1)$ cutting planes steps. This is the PB equivalent of the *resolution* rule for clauses.

Theorem 2. (Demirović et al. 2024) Let F be a PB formula, ρ be a partial assignment, and suppose that from $F\!\upharpoonright_\rho$ we can derive a constraint D using a cutting planes and RUP derivation of length n. Then we can construct a derivation of length $O(n)$ from F of the constraint $\wedge\rho \Rightarrow D$.

Corollary 1. As a special case of Theorem 2, if we can derive a constraint D from constraints C_1 and C_2 in n steps, then we can derive in $O(n)$ steps the constraint $\wedge(L_1 \cup L_2) \Rightarrow D$ from $\wedge L_1 \Rightarrow C_1$ and $\wedge L_2 \Rightarrow C_2$.

2.3 Proof Logging for Constraint Programming

To implement pseudo-Boolean proof logging in a CP solver that allows arbitrary finite domain variables, the solver's input problem must first be compiled to a PB format. This encoding is only used for deriving subsequent justification constraints and does not inform the native solving process.

It is straightforward to represent any CP variable X with positive integer domain values in the range $[l \ldots u]$ as a sequence of PB bit variables $x_0, \ldots, x_{n-1}$, where n is a sufficient number of bits to represent any value in the domain as a binary string. The value of X is then given by the PB expression $\sum_{i=0}^{n-1} 2^i x_i$. If X has negative domain values, an additional bit x_n can be used, and the PB variable sequence represents a binary two's complement number, evaluated as $-2^n x_n + \sum_{i=0}^{n-1} 2^i x_i$. We can also make use of auxiliary PB variables defined by reifying inequalities on these expressions to represent the condition that X is greater than, less than, or equal to a certain value, i.e.

$$x_{\geq i} \Leftrightarrow -2^n x_n + \textstyle\sum_{i=0}^{n-1} 2^i x_i \geq i \qquad (3)$$

$$x_{=i} \Leftrightarrow x_{\geq i} + \overline{x}_{\geq i+1} \geq 2 \qquad (4)$$

These can be included in the initial PB encoding, or introduced as extension variables in the proof, as required. Since $>, <$, and $\leq$ can all be expressed in terms of $\geq$ by negating and adding 1 as necessary, we will write literals such as $x_{\leq j}$ instead of $\overline{x}_{\geq j+1}$ where appropriate.

It remains to define faithful PB encodings of all the CP constraints posted to the solver in terms of these variables. For many constraints e.g. linear inequalities this is trivial, and for other global constraints we can select the simplest "dumbest" known decompositions since we only care that the translation is correct, not that the resulting PB formula has strong propagation properties.

With the encoding in place, a backtracking solver can output a proof by logging a RUP constraint that encodes the negation of the sequence guesses made prior to each backtrack. For example, if a solver guessesd $X = 1$, $Y = 2$ and $Z = 3$ and then falsified a constraint, it would output

$$\text{RUP:} \quad \overline{x}_{=1} + \overline{y}_{=2} + \overline{z}_{=3} \geq 1, \qquad (5)$$

saying that at least one of these guesses cannot hold, and that the verifier checks this by running unit propagation under the assignment $\rho = \{x_{=1}, y_{=2}, z_{=3}\}$ on the PB encoding and previously derived constraints. This results in the proof log being a complete description of the solver's search tree, and allows an unsatisfiability or optimality conclusion to be certified.

When a CP solver also performs constraint propagation in addition to search, additional *justification* constraints may need to be derived so that inferences known to the solver are also available to the verifier on reverse unit propagation of the backtrack constraint. These resemble the *explaining* clauses generated by lazy explanation-based hybrid CP-SAT solvers, except they must be formally derived in the proof system rather than simply asserted. For example if a solver makes guesses as above and then infers that due to $X = 1$, an additional variable W must be greater than or equal to 2 (e.g. from a propagator for a constraint $W > X$) the solver should somehow derive $x_{=1} \Rightarrow w_{\geq 2} \geq 1$. The presence of this constraint ensures that the PB encoding of $W \geq 2$ is active in the verifier's constraint database after propagation of $\rho = \{x_{=1}\}$, and we call $x_{=1}$ the *reason* for the inference.

An important point to note is that although the binary encoding is not strongly propagating for individual values, it is at least guaranteed that contradictory bounds on the same variable, e.g. $x_{\geq l}$ and $x_{<u}$ for $u \geq l$, always propagate to contradiction, and we make use of this implicitly when designing justification procedures.

3 Justifying Bounds-Consistent Propagation of Multiplication Constraints

The most obvious encoding for the constraint $X \times Y = Z$ when the CP variables X, Y, and Z are encoded with PB bit variables $x_0, \ldots, x_{n-1}$; $y_0, \ldots, y_{m-1}$; $z_0, \ldots, z_{k-1}$, would be to define, for each $i \in [0 \ldots n-1]$ and $j \in [0 \ldots m-1]$

$$xy_{ij} \Leftrightarrow x_i + y_j \geq 2, \tag{6}$$

which is syntactic sugar for

$$M_{ij}^{\Rightarrow} := 2\overline{xy}_{ij} + x_i + y_j \geq 2, \text{ and} \tag{7}$$

$$M_{ij}^{\Leftarrow} := xy_{ij} + \overline{x}_i + \overline{y}_j \geq 1. \tag{8}$$

These represent the bit multiplications. We can then constrain the bits of Z with two inequalities that enforce

$$\sum_{i=0}^{k-1} 2^i z_i - \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} 2^{i+j} xy_{ij} = 0. \tag{9}$$

Note that this requires $2mn + 2$ constraints.

3.1 Deriving Simple Bounds

Proposition 1. Suppose we have a PB formula containing the encoding of a multiplication constraint $X \times Y = Z$ as above. Then for non-negative integers l_1, l_2, we can derive $\sum_{i=0}^{k-1} 2^i z_i \geq l_1 l_2$ from $\{B_1, B_2\}$ where

$$B_1 := \sum_{i=0}^{n-1} 2^i x_i \geq l_1, \quad B_2 := \sum_{i=0}^{m-1} 2^i y_i \geq l_2$$

using a cutting-planes derivation of length $O(m \cdot n)$.

Proof. For each $i \in [0 \ldots n-1]$ we can derive the constraint $D_i := \sum_{j=0}^{m-1} 2^j xy_{ij} + l_2 \overline{x}_i \geq l_2$, by first deriving $\sum_j (2^j \cdot M_{ij}^{\Leftarrow})$, then adding B_2, and saturating. If any of the coefficients of xy_{ij} were reduced by the saturation step, we can add literal axioms to maintain the required form.

We can then derive $\sum_i (2^i \cdot D_i)$, yielding

$$\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} 2^{i+j} xy_{ij} + \sum_{i=0}^{n-1} 2^i l_2 \overline{x}_i \geq (2^n - 1) l_2. \tag{10}$$

If we add $l_2 \cdot B_1$ to this we are left with

$$\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} 2^{i+j} xy_{ij} \geq l_1 l_2, \tag{11}$$

which added to the first inequality in (9) establishes the required bound. Note that this required $O(m \cdot n)$ steps. $\square$

The above is sufficient to create a justification for a lower bound on Z when non-negative lower bounds for X and Y are known. We can construct similar justifications for upper bounds, but this requires an explicit contradiction step.

Proposition 2. Under the same assumptions above, for non-negative integers u_1, u_2, we can derive $\sum_{i=0}^{k-1} -2^i z_i \geq -u_1 u_2$ from $\{U_1, U_2\}$, where

$$U_1 := \sum_{i=0}^{n-1} -2^i x_i \geq -u_1, \quad U_2 := \sum_{i=0}^{m-1} -2^i y_i \geq -u_2,$$

using a derivation of length $O(n \cdot m)$.

Proof. By adding each of $\overline{x}_i \geq 0$ and $\overline{y}_j \geq 0$ to $M_{ij}^{\Rightarrow}$ and then saturating we can derive the constraints

$$W_{ij}^y := -xy_{ij} + y_j \geq 0, \quad W_{ij}^x := -xy_{ij} + x_i \geq 0. \tag{12}$$

This allows us to derive for each $i \in [0 \ldots n-1]$, the constraints $\sum_j (2^j \cdot W_{ij}^y) + U_2$ and $\sum_j (2^j \cdot W_{ij}^x)$, i.e.

$$-\sum_{j=0}^{m-1} 2^j xy_{ij} \geq -u_2, \text{ and} \tag{13}$$

$$-\sum_{j=0}^{m-1} 2^j xy_{ij} + (2^m - 1) x_i \geq 0. \tag{14}$$

Then from (13) and (14) we can derive

$$D_i := -\sum_{j=0}^{m-1} 2^j xy_{ij} + u_2 x_i \geq 0 \tag{15}$$

This is because the negation of (15) can be used in a short contradiction sub-proof: we add it to each of (13) and (14), saturate, and then add the results to obtain $0 \geq 1$. It is an interesting open question whether an efficient pure cutting-planes derivation exists for constraints in this form.

Regardless, we can derive n constraints $\{D_i\}$ in $O(m)$ steps, and then compute $\sum_i (2^i \cdot D_i) + u_2 \cdot U_1$, yielding

$$-\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} 2^{i+j} xy_{ij} \geq -u_1 u_2 \tag{16}$$

After adding this to the second inequality in (9), we have derived the required bound in $O(m \cdot n)$ steps. $\square$

3.2 Representation with Negative Values

If a CP variable X has only positive domain values then it can be represented in PB form as a sequence of bit variables, with the value given by $\sum_i 2^i x_i$. When variables' domains contain negative domain values, Gocht, McCreesh, and Nordström (2022) use a two's complement approach, which allows compact representation and straightforward proofs for linear constraints. For multiplication, however, we would prefer to use a sign-bit representation, since directly encoding multiplication of two's complement bit strings results in a more complex expression where the relationship between the bounds of X and Y and the bounds on Z is obfuscated (Baugh and Wooley 1973). We therefore propose defining sets of auxiliary bit variables to represent the absolute values of the two's complement strings, defining multiplication of the absolute value bits and sign bits separately, and then channelling between the two representations. This also allows us to neatly reuse our justifications from the positive-only case as a subroutine in the general case.

Concretely, suppose a CP variable X is encoded with a sequences of PB bit variables $x_0, \ldots, x_{n-1}$ along with an additional two's complement bit x_n. To define a multiplication constraint in our PB model, we would introduce an additional set of $n+1$ bit variables $|x|_0, \ldots, |x|_n$ to represent the absolute value of X and then define the channelling constraints

$$\overline{x}_n \Rightarrow \textstyle\sum_{i=0}^{n-1} 2^i x_i - \sum_{i=0}^{n} 2_i |x|_i = 0, \tag{17}$$

$$x_n \Rightarrow \textstyle\sum_{i=0}^{n-1} 2^i x_i + \sum_{i=0}^{n} 2^i |x|_i = 2^n. \tag{18}$$

Each equality here is represented as two inequalities with opposite signs $\gamma \in \{1, -1\}$. Recalling for any PB literal w the notation w^σ, $\sigma \in \{1, -1\}$, where $w^1 = w$ and $w^{-1} = \overline{w}$, we can then refer to any one of these four constraints using $C_x(\sigma, \gamma)$, where σ specifies the sign of the reifying term and γ the direction of the inequality, i.e. $C_x(\sigma, \gamma) :=$

$$x_n^\sigma \Rightarrow \gamma \sum_{i=0}^{n-1} 2^i x_i + \sigma\gamma \sum_{i=0}^{n} 2^i |x|_i \geq 2^n \cdot {}^{(\gamma\sigma+\gamma)}\!/_2. \tag{19}$$

If we define similar channelling constraints $C_y(\sigma, \gamma)$ and $C_z(\sigma, \gamma)$ for Y and Z, we can encode the multiplication of the magnitudes as in Section 3.1:

$$\sum_{i=0}^{k-1} 2^i |z|_i - \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} 2^{i+j} |xy|_{ij} = 0. \tag{20}$$

We can then deal with the two's complement bits separately, by first defining an auxiliary variable $s = x_n \oplus y_m$, i.e.

$$x_n \wedge y_m \Rightarrow \overline{s}; \qquad \overline{x}_n \wedge \overline{y}_m \Rightarrow \overline{s}; \tag{21}$$

$$x_n \wedge \overline{y}_m \Rightarrow s; \qquad \overline{x}_n \wedge \overline{y}_m \Rightarrow s; \tag{22}$$

and then defining

$$z_k \Leftrightarrow s + \overline{x}_{=0} + \overline{y}_{=0} \geq 3. \tag{23}$$

Note that we cannot define $z_k = x_n \oplus y_m$ directly since a negative value multiplied by 0 should result in z_k being set to 0 and not 1 (negative 0 cannot be represented in two's complement, unlike sign-bit representation).

This representation gives a faithful encoding of $X \times Y = Z$, and also allows satisfies the following useful property, allowing us to derive reified bounds on the magnitude bit sum from any bound on the two's complement sum.

Proposition 3. Suppose for bit variables $x_0, \ldots, x_n$ and $|x|_0, \ldots, |x|_n$ we have a PB formula containing constraints as defined in (17) and (18). Then for any integer b (positive or negative), and $\gamma \in \{1, -1\}$ we can derive

$$\overline{x}_n \wedge \overline{x}_{=0} \Rightarrow \gamma \cdot \textstyle\sum_{i=0}^{n} 2^i |x|_i \geq \gamma \cdot \max(\gamma b, 1) \tag{24}$$

$$x_n \wedge \overline{x}_{=0} \Rightarrow -\gamma \cdot \textstyle\sum_{i=0}^{n} 2^i |x|_i \geq \gamma \cdot \min(\gamma b, -1) \tag{25}$$

from

$$-\gamma \cdot 2^n x_n + \gamma \cdot \textstyle\sum_{i=0}^{n-1} 2^i x_i \geq b \tag{26}$$

using $O(1)$ RUP and cutting planes steps.

Proof. There are only two possibilities for the left-hand sides of (24) and (25).

First if $\gamma \cdot \max(\gamma b, 1) = b$, then $\gamma \cdot \min(\gamma b, -1) = -\gamma$. So we can add (26) to $C_x(-1, -\gamma)$ and then add sufficient literal axioms x_n, $x_{=0}$ to obtain (24); and introduce (25) by RUP (consider $\gamma = \pm 1$ in each case)

Otherwise, $\gamma \cdot \max(\gamma b, 1) = \gamma$, so $\gamma \cdot \min(\gamma b, -1) = b$, and we can add (26) to $C_x(1, -\gamma)$ and add literal axioms to obtain (25); and introduce (24) by RUP. ☐

Similarly, we can derive reified bounds on the original product variable Z from reified bounds on its magnitude.

Proposition 4. Suppose we have a PB formula containing the encoding of a multiplication constraint $X \times Y = Z$ for two's complement encoded variables as above.

Then for integers b; $\sigma_1, \sigma_2, \gamma \in \{1, -1\}$; and letting $\sigma_3 = \sigma_1 \cdot \sigma_2$, $\wedge L = x_n^{\sigma_1} \wedge y_{m-1}^{\sigma_2} \wedge \overline{x}_{=0} \wedge \overline{y}_{=0}$; we can derive

$$\wedge L \Rightarrow -\sigma_3 \gamma \cdot 2^k z_k + \sigma_3 \gamma \cdot \sum_{i=0}^{k} 2^i z_i \geq b \tag{27}$$

$$\text{from} \qquad \wedge\, L \Rightarrow \gamma \cdot \sum_{i=0}^{k-1} 2^i |z|_i \geq b \tag{28}$$

with $O(1)$ RUP or cutting planes steps.

Proof. Due to the definitions (21), (22), (23) we can introduce by RUP

$$\wedge L \Rightarrow z_k^{-\sigma_3} \geq 1, \text{ i.e. } \wedge\, L \Rightarrow -\sigma_3 z_k \geq {}^{(-\sigma_3+1)}\!/_2. \tag{29}$$

By multiplying (29) by an appropriate scalar and adding it to $C_z(-\sigma_3, \gamma\sigma_3)$ we can substitute $\wedge L$ for the reifying term of $C_z(-\sigma_3, \sigma_3\gamma)$. We can then add (28) to this, yielding

$$\wedge L \Rightarrow \sigma_3 \gamma \cdot \sum_{i=0}^{k-1} 2^i z_i \geq b - 2^k \cdot \gamma \cdot {}^{(-\sigma_3+1)}\!/_2. \tag{30}$$

We can then introduce in a single step

$$\wedge L \Rightarrow -\gamma\sigma_3 \cdot 2^k z_k \geq 2^k \cdot \gamma \cdot {}^{(-\sigma_3+1)}\!/_2 \tag{31}$$

since if $\gamma = 1$ this is just (29) multiplied by 2^k, and if $\gamma = -1$ it is a trivial constraint that can be built from literal axioms (consider $\sigma_3 = \pm 1$ in each case). Adding (31) and (30) gives (27), as required. ☐

3.3 Deriving Bounds on the Product Variable

A CP bounds propagator for the multiplication constraint $X \times Y = Z$ will typically apply pruning rules to each variable in turn, narrowing the bounds on one variable based on the bounds on the other two. We first consider computing the bounds on Z based on the bounds of X and Y. Suppose the current domains of X and Y are known to be $D_X \subseteq [l_1 \ldots u_1]$, and $D_Y \subseteq [l_2 \ldots u_2]$. Then the solver can infer $D_Z \subseteq [\inf E \ldots \sup E]$ where $E = \{l_1 l_2, l_1 u_2, u_1 l_2, u_1 u_2\}$, and prune any values outside this range.

In what follows we will use the shorthands

$$\Sigma_x := -2^n x_n + \textstyle\sum_{i=0}^{n-1} 2^i x_i, \quad \Sigma_{|x|} := \sum_{i=0}^{n} 2^i |x|_i, \tag{32}$$

with Σ_y, $\Sigma_{|y|}$, Σ_z, $\Sigma_{|z|}$ defined similarly. For any integer b we will also use $b^+ := \max(b, 1)$, $b^- := \min(b, -1)$.

Theorem 3. From a PB encoding of $X \times Y = Z$ defined as described in Section 3.2 and for conjunction of reason literals

$$\mathcal{R} := x_{\geq l_1} \wedge x_{\leq u_1} \wedge y_{\geq l_2} \wedge y_{\leq u_2} \tag{33}$$

defined as in (3) we can derive

$$G_1 := \mathcal{R} \Rightarrow \quad \Sigma_z \geq \inf E, \text{ and} \tag{34}$$
$$G_2 := \mathcal{R} \Rightarrow -\Sigma_z \geq -\sup E \tag{35}$$

in $O(n \cdot m)$ proof steps.

Proof. Let F be the PB formula consisting of the encoding of $X \times Y = Z$ and the constraints defining the reason literals $x_{\geq l_1}$, etc. We will first construct a derivation from $F\restriction_{\mathcal{R}}$ to avoid reifying everything by $\mathcal{R}$. So starting with $F\restriction_{\mathcal{R}}$, clearly we can introduce

$$\Sigma_x \geq l_1; -\Sigma_x \geq -u_1; \Sigma_y \geq l_2; -\Sigma_y \geq -u_2 \tag{36}$$

by RUP, due to the definitions of the literals in $\mathcal{R}$. Using Proposition 3, and letting $\ell_x^0 = \overline{x}_{=0}$, $\ell_y^0 = \overline{y}_{=0}$ (for compactness), we can then derive in $O(1)$:

$$\overline{x}_n \wedge \ell_x^0 \Rightarrow \Sigma_{|x|} \geq \ l_1^+, \qquad x_n \wedge \ell_x^0 \Rightarrow -\Sigma_{|x|} \geq \ l_1^-, \tag{37}$$
$$x_n \wedge \ell_x^0 \Rightarrow \Sigma_{|x|} \geq -u_1^-, \qquad \overline{x}_n \wedge \ell_x^0 \Rightarrow -\Sigma_{|x|} \geq -u_1^+, \tag{38}$$
$$\overline{y}_n \wedge \ell_y^0 \Rightarrow \Sigma_{|y|} \geq \ l_2^+, \qquad y_n \wedge \ell_y^0 \Rightarrow -\Sigma_{|y|} \geq \ l_2^-, \tag{39}$$
$$y_n \wedge \ell_y^0 \Rightarrow \Sigma_{|y|} \geq -u_2^-, \qquad \overline{y}_n \wedge \ell_y^0 \Rightarrow -\Sigma_{|y|} \geq -u_2^+. \tag{40}$$

Since the constraints in the first column are (reified) positive lower bounds, and those in the second are reified positive upper bounds, we can use Corollary 1, Proposition 1 and Proposition 2 to derive reified upper and lower bounds on the product magnitude in $O(n \cdot m)$. If we write ${}^{\sigma_1}\wedge^{\sigma_2} := x_n^{-\sigma_1} \wedge y_n^{-\sigma_2} \wedge \overline{x}_{=0} \wedge \overline{y}_{=0}$ where $\sigma_1, \sigma_2 \in \{+, -\}$ (meaning ± 1), the resulting eight constraints are

$${}^+\wedge^+ \Rightarrow \Sigma_{|z|} \geq \ l_1^+ l_2^+, \quad {}^+\wedge^- \Rightarrow -\Sigma_{|z|} \geq \ u_1^+ l_2^-, \tag{41}$$
$${}^-\wedge^- \Rightarrow \Sigma_{|z|} \geq \ u_1^- u_2^-, \quad {}^-\wedge^+ \Rightarrow -\Sigma_{|z|} \geq \ l_1^- u_2^+, \tag{42}$$
$${}^-\wedge^+ \Rightarrow \Sigma_{|z|} \geq -u_1^- l_2^+, \quad {}^+\wedge^+ \Rightarrow -\Sigma_{|z|} \geq -u_1^+ u_2^+, \tag{43}$$
$${}^+\wedge^- \Rightarrow \Sigma_{|z|} \geq -l_1^+ u_2^-, \quad {}^-\wedge^- \Rightarrow -\Sigma_{|z|} \geq \ -l_1^- l_2^-. \tag{44}$$

We can then apply Proposition 4 to transform these into conditional bounds on the original bit variables for Z, noting that the product σ_1, σ_2 of the signs in the condition ${}^{\sigma_1}\wedge^{\sigma_2}$ determines whether the inequality should be flipped.

$${}^+\wedge^+ \Rightarrow \ \Sigma_z \geq \ l_1^+ l_2^+, \qquad {}^+\wedge^- \Rightarrow \ \Sigma_z \geq \ u_1^+ l_2^-, \tag{45}$$
$${}^-\wedge^- \Rightarrow \ \Sigma_z \geq \ u_1^- u_2^-, \qquad {}^-\wedge^+ \Rightarrow \ \Sigma_z \geq \ l_1^- u_2^+, \tag{46}$$
$${}^-\wedge^+ \Rightarrow -\Sigma_z \geq -u_1^- l_2^+, \qquad {}^+\wedge^+ \Rightarrow -\Sigma_z \geq -u_1^+ u_2^+, \tag{47}$$
$${}^+\wedge^- \Rightarrow -\Sigma_z \geq -l_1^+ u_2^-, \qquad {}^-\wedge^- \Rightarrow -\Sigma_z \geq \ -l_1^- l_2^-. \tag{48}$$

Separately, we can deal with the cases where X and Y are 0, introducing by RUP

$$x_{=0} \Rightarrow \ \Sigma_z \geq 0, \qquad y_{=0} \Rightarrow \ \Sigma_z \geq 0, \tag{49}$$
$$x_{=0} \Rightarrow -\Sigma_z \geq 0, \qquad y_{=0} \Rightarrow -\Sigma_z \geq 0. \tag{50}$$

Since we have shown we can derive these twelve constraints from $F\restriction_{\mathcal{R}}$ in $O(n \cdot m)$ steps, Theorem 2 says we can derive versions of them additionally reified by $\mathcal{R}$ in $O(n \cdot m)$ from F.

Suppose now that we have the negation of $G_1\restriction_{\mathcal{R}}$, i.e. $-\Sigma_z \geq -\inf E + 1$, and let H_l be the set of lower bound constraints on Σ_z: (45), (46), and (49). For each constraint in $C \in H_l$, an exhaustive case analysis on the possible signs for l_1, l_2, u_1, u_2 (nine cases) tells us that either $(\inf E - 1)$ is strictly less than the right-hand side of C, or one of the reifying conditions on the left-hand side of C must be violated. This means we can derive by RUP the six constraints

$$\overline{x}_n \wedge \overline{y}_n \wedge \overline{x}_{=0} \wedge \overline{y}_{=0} \Rightarrow 0 \geq 1, \tag{51}$$
$$\overline{x}_n \wedge y_n \wedge \overline{x}_{=0} \wedge \overline{y}_{=0} \Rightarrow 0 \geq 1, \tag{52}$$
$$x_n \wedge y_n \wedge \overline{x}_{=0} \wedge \overline{y}_{=0} \Rightarrow 0 \geq 1, \tag{53}$$
$$x_n \wedge \overline{y}_n \wedge \overline{x}_{=0} \wedge \overline{y}_{=0} \Rightarrow 0 \geq 1, \tag{54}$$
$$x_{=0} \Rightarrow 0 \geq 1, \quad y_{=0} \Rightarrow 0 \geq 1, \tag{55}$$

which immediately allow us to derive contradiction $(0 \geq 1)$ in $O(1)$ resolution steps (Theorem 1).

Then by Theorem 2 this means we can derive $\mathcal{R} \Rightarrow 0 \geq 1$ from $\mathcal{R} \Rightarrow H_l$ and $\neg G_1$ in $O(1)$. Hence we can complete our derivation of G_1 from F by following the above procedure to first derive $\mathcal{R} \Rightarrow H_l$ in $O(m \cdot n)$ and then introducing G_1 using an explicit contradiction sub-proof that derives from $\mathcal{R} \Rightarrow 0 \geq 1$ as above in $O(1)$ and then introduces $0 \geq 1$ by RUP (the negation of G_1 propagates all the literals in $\mathcal{R}$ leading to contradiction).

An analogous derivation allows us to derive G_2 by redundance from reified versions of the upper bounds: (47), (48), and (50). ☐

3.4 Deriving Bounds on the Multiplicands

The bounds propagator for $X \times Y = Z$ will also compute bounds on X based on bounds of Y and Z. If we show how to justify this, we have likewise shown by symmetry how to justify bounds on Y based on X and Z, and thus completed the description of our method for adding proof logging to a bounds-consistent multiplication propagator. Unlike bounds on the product variable, these inferences cannot be compactly expressed as a simple min/max operation, since the solver must deal with different cases depending on whether 0 is within the bounds of Z.

Suppose that the current domains of Y and Z are known to be $D_Y \subseteq [l_2 \dots u_2]$, and $D_z \subseteq [l_3 \dots u_3]$. First if $l_2 \leq 0 \leq u_2$ and $l_3 \leq 0 \leq u_3$ then we cannot infer anything about the bounds on X, since any value is possible. Next, if $l_2 = u_2 = 0$ and $0 \notin [l_3 \dots u_3]$ then we can immediately infer contradiction, and the justification follows by RUP. In any other case, we can compute bounds on X which may reduce the domain, and a complete set of cases for this is given by Schulte and Stuckey (2001) and also Apt and Zoeteweij (2004).

Fortunately, we do not need to take each case in turn for constructing justifications, instead we can rely solely on the fact the propagator maintains bounds consistency.

Theorem 4. Suppose for a multiplication constraint $X \times Y = Z$ a bounds-consistent propagator can infer bounds $l \leq X \leq u$ based on $l_2 \leq Y \leq u_2$ and $l_3 \leq Z \leq u_3$.

Then for a PB encoding of $X \times Y = Z$ defined as described in Section 3.2 and for a conjunction of reason literals

$$\mathcal{R} := y_{\geq l_2} \wedge y_{\leq u_2} \wedge z_{\geq l_3} \wedge z_{\leq u_3} \tag{56}$$

defined as in (3), we can derive

$$\mathcal{R} \Rightarrow \Sigma_x \geq l, \quad \text{and} \quad \mathcal{R} \Rightarrow -\Sigma_x \geq -u \tag{57}$$

in $O(n \cdot m)$ proof steps.

Proof. Let F be the PB formula containing the encoding of $X \times Y = Z$ and definitions of the reason literals. We can introduce $x_{>u}$, $x_{<l}$ as extension variables in the proof if necessary, so assume F contains the constraints defining these too. Now let $\mathcal{R}_1 = \mathcal{R} \cup \{x_{>u}\}$, $\mathcal{R}_2 = \mathcal{R} \cup \{x_{<l}\}$.

From $F\restriction_{\mathcal{R}_1}$ and letting $l_1 = u + 1$ we can derive

$$^{+}\wedge^{+} \Rightarrow \ \Sigma_z \geq \ l_1^{+} l_2^{+}, \qquad ^{-}\wedge^{+} \Rightarrow \ \Sigma_z \geq \ l_1^{-} u_2^{+}, \tag{58}$$

$$^{+}\wedge^{-} \Rightarrow -\Sigma_z \geq -l_1^{+} u_2^{-}, \qquad ^{-}\wedge^{-} \Rightarrow -\Sigma_z \geq -l_1^{-} l_2^{-}. \tag{59}$$

$$x_{=0} \Rightarrow \ \Sigma_z \geq 0, \qquad x_{=0} \Rightarrow -\Sigma_z \geq 0, \tag{60}$$

$$y_{=0} \Rightarrow \ \Sigma_z \geq 0, \qquad y_{=0} \Rightarrow -\Sigma_z \geq 0. \tag{61}$$

in exactly the same $O(n \cdot m)$ steps as in the proof of Theorem 3. But from $F\restriction_{\mathcal{R}_1}$ we can also introduce by RUP

$$\Sigma_z \geq l_3 \quad \text{and} \quad -\Sigma_z \geq -u_3. \tag{62}$$

We also know that due to bounds consistency, there cannot be any real numbers $a \in [l_2 \ldots u_2]$, $b \in [l_3 \ldots u_3]$ such that $(u + 1) \times a = b$. We can use this to ascertain by case analysis on the possible signs for $(u + 1)$, l_2 and u_2, (six cases) that for all the constraints in (58) and (59); and at least one of the constraints in each of (60) and (61); either the bound on Σ_z contradicts one of the bounds in (62), or else one of the reifying conditions is violated. Hence we can introduce again the same clauses (51) to (55) and so derive contradiction from $F\restriction_{\mathcal{R}_1}$.

An analogous argument will show that we can derive contradiction from $F\restriction_{\mathcal{R}_2}$ in $O(n \cdot m)$ steps. Thus by Theorem 2 we can derive $\wedge\mathcal{R}_1 \Rightarrow 0 \geq 1$ and $\wedge\mathcal{R}_2 \Rightarrow 0 \geq 1$ from F in $O(n \cdot m)$ steps. Recalling the definitions of $\mathcal{R}_1$, $\mathcal{R}_2$, and reified PB constraints, these are the same as

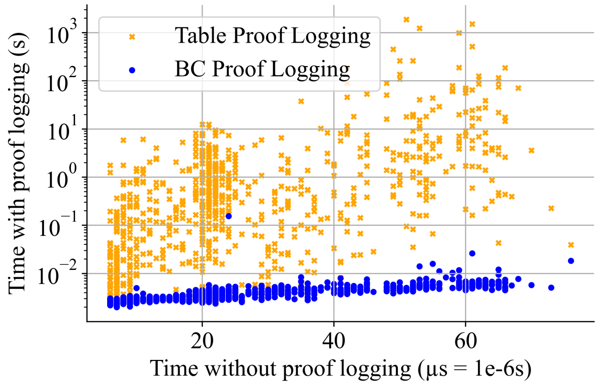
$$\mathcal{R} \Rightarrow \overline{x}_{>u} \geq 1 \quad \text{and} \quad \mathcal{R} \Rightarrow \overline{x}_{<l} \geq 1, \tag{63}$$

and then our desired constraints (57) follow by RUP. $\square$

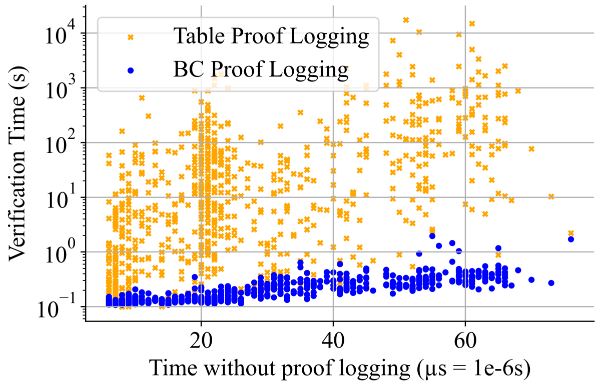
4 Implementation and Experiments

The constructive proofs of Theorem 3 and Theorem 4 together define a procedure for constructing justifications that can be implemented in a proof logging CP solver. We omit implementation details, noting that all that is strictly required is to have the solver write out each rule application we have outlined at the point of bounds propagation using little more than template-based print statements. To illustrate this, we have extended the Glasgow Constraint Solver (GCS) project of Gocht, McCreesh, and Nordström (2022) with a bounds consistent multiplication propagator[1]. Since GCS already has support for auto-tabulation with proof logging, we also created an alternative implementation that uses generated tables for justifications. This involves precomputing all possible solutions to a constraint, and then defining a new constraint that says at least one of these solutions must hold. Once these reified solutions are logged in the proof, any bounds-consistent propagation justification follows by RUP. For our justification procedures to be worthwhile, they need to be at least better (faster to produce and verify) than this brute force method.

[1]https://doi.org/10.5281/zenodo.14500848



(a) Solve time (log scale) with proof logging vs. solve time without proof logging for both approaches.



(b) *VeriPB* verification time (log scale) vs. solve time without proof logging for both approaches.

Figure 1: Comparing our approach using bounds-consistent justifications to auto-tabulation proofs.

To test this empirically, we ran the solver on some minimal CSP instances, with and without both kinds of proof logging and verified the produced proof files with the *VeriPB* tool. For further validation, we tested the solver on one thousand synthetic instances, each consisting of a single ternary multiplication constraint with domains that are randomly chosen (increasingly large) sub-intervals of $[-200 \ldots 200]$. We used hardware with dual AMD EPYC 7643 CPUs, 2TBytes RAM, running Ubuntu 22.04. The results of this are shown in Figure 1. Clearly BC proof logging is much faster than the auto-tabulation baseline here. Then for an additional example, we modelled the “n-fractions” problem from CSPLib[2] for $n = 2$, which uses several multiplication constraints with domains of up to size $10,000$. Using GCS augmented with our multiplication constraint, we were able to generate a 428MB proof of the uniqueness of the solution (up to symmetry) in 8.8s, and verified this using VeriPB in 4864.5s. We stopped running the same problem using tabulation proofs when the proof file size exceeded 2TB.

We leave a more comprehensive analysis of the behaviour of a proof logging solver on more diverse benchmarks to future work. Precise overheads can be highly implementation-dependent as the solver continuously writes to disk. But our indicative results demonstrate that our approach is practical, and far better than any known alternative.

5 Conclusion

While many constraint propagators are almost trivial to add pseudo-Boolean proof logging, bounds-consistent multiplication justifications do require some more intricate construction. Nevertheless, we have shown that there is a clear return on investing this effort, and that once again *VeriPB* proofs are surprisingly amenable to certifying reasoning in constraint programming.

Additionally, this work is an exemplification of how PB proofs under restrictions and proofs by contradiction can be employed when designing a justification procedure, and we expect these will be useful when implementing proofs for further constraints, particularly those that behave differently when negative integer values are involved.

Acknowledgements

The authors would like to thank Jakob Nordström, Arthur Gontier, and the participants of the 1st International Workshop on Highlights in Organizing and Optimizing Proof-logging Systems (WHOOPS '24) for helpful discussions regarding the derivations in this work.

Ciaran McCreesh was supported by a Royal Academy of Engineering research fellowship, and by the Engineering and Physical Sciences Research Council [grant number EP/X030032/1].

References

Akgün, Ö.; Gent, I. P.; Jefferson, C.; Miguel, I.; and Nightingale, P. 2018. Metamorphic Testing of Constraint Solvers. In Hooker, J., ed., *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, 727–736. Cham: Springer International Publishing.

Apt, K. R.; and Zoeteweij, P. 2004. A Comparative Study of Arithmetic Constraints on Integer Intervals. In *Recent Advances in Constraints*, volume 3010, 1–24. Berlin, Heidelberg: Springer Berlin Heidelberg.

Baugh, C.; and Wooley, B. 1973. A Two's Complement Parallel Array Multiplication Algorithm. *IEEE Transactions on Computers*, C-22(12): 1045–1047.

Bogaerts, B.; Gocht, S.; McCreesh, C.; and Nordström, J. 2023a. Certified Dominance and Symmetry Breaking for Combinatorial Optimisation. *Journal of Artificial Intelligence Research*, 77: 1539–1589.

Bogaerts, B.; McCreesh, C.; Oertel, A.; Myreen, M. O.; Tan, Y. K.; and Nordstrom, J. 2023b. Documentation of VeriPB and CakePB for the SAT Competition 2023. Technical report.

Buss, S.; and Nordström, J. 2021. Chapter 7. Proof Complexity and SAT Solving. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 233–350. IOS Press.

Choi, C. W.; Harvey, W.; Lee, J. H. M.; and Stuckey, P. J. 2006. Finite Domain Bounds Consistency Revisited. In Sattar, A.; and Kang, B.-h., eds., *AI 2006: Advances in Artificial Intelligence*, Lecture Notes in Computer Science, 49–58. Berlin, Heidelberg: Springer.

Chu, G.; Stuckey, P. J.; Schutt, A.; Ehlers, T.; Gange, G.; and Francis, K. 2024. Chuffed, a lazy clause generation solver. https://github.com/chuffed/chuffed. Accessed 2024-12-13.

Cook, W.; Coullard, C. R.; and Turán, Gy. 1987. On the Complexity of Cutting-Plane Proofs. *Discrete Applied Mathematics*, 18(1): 25–38.

Demirović, E.; McCreesh, C.; McIlree, M.; Nordström, J.; Oertel, A.; and Sidorov, K. 2024. Pseudo-Boolean Reasoning About States and Transitions to Certify Dynamic Programming and Decision Diagram Algorithms. Forthcoming.

Elffers, J.; Gocht, S.; McCreesh, C.; and Nordström, J. 2020. Justifying All Differences Using Pseudo-Boolean Reasoning. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, 1486–1494. AAAI Press.

Falkner, A.; Friedrich, G.; Haselböck, A.; Schenner, G.; and Schreiner, H. 2016. Twenty-Five Years of Successful Application of Constraint Technologies at Siemens. *AI Magazine*, 37(4): 67–80.

Flippo, M.; Sidorov, K.; Marijnissen, I.; Smits, J.; and Demirović, E. 2024. A Multi-Stage Proof Logging Framework to Certify the Correctness of CP Solvers. In Shaw, P., ed., *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*, volume 307 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 11:1–11:20. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-336-2.

Gecode Team. 2024. Gecode: Generic Constraint Development Environment. http://www.gecode.org. Accessed: 2024-12-13.

Gillard, X.; Schaus, P.; and Deville, Y. 2019. SolverCheck: Declarative Testing of Constraints. In Schiex, T.; and de Givry, S., eds., *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, 565–582. Cham: Springer International Publishing.

Gocht, S.; McCreesh, C.; and Nordström, J. 2022. An Auditable Constraint Programming Solver. In Solnon, C., ed.,

[2]https://www.csplib.org/Problems/prob041/

Proceeding of the 28th International Conference on Principles and Practice of Constraint Programming, volume 235 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 25:1–25:18. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Heule, M. J. 2021. Chapter 15. Proofs of Unsatisfiability. In Biere, A.; Heule, M.; Van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 635–668. IOS Press.

Hooker, J. N. 1992. Generalized Resolution for 0–1 Linear Inequalities. *Annals of Mathematics and Artificial Intelligence*, 6(1): 271–286.

Lecoutre, C. 2013. *Constraint Networks: Targeting Simplicity for Techniques and Algorithms*. John Wiley & Sons.

McConnell, R. M.; Mehlhorn, K.; Näher, S.; and Schweitzer, P. 2011. Certifying Algorithms. *Computer Science Review*, 5(2): 119–161.

McIlree, M. J.; and McCreesh, C. 2023. Proof Logging for Smart Extensional Constraints. In Yap, R. H. C., ed., *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 26:1–26:17. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-300-3.

McIlree, M. J.; McCreesh, C.; and Nordström, J. 2024. Proof Logging for the Circuit Constraint. In Dilkina, B., ed., *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 38–55. Cham: Springer Nature Switzerland. ISBN 978-3-031-60599-4.

Paxian, T.; and Biere, A. 2023. Uncovering and Classifying Bugs in MaxSAT Solvers through Fuzzing and Delta Debugging. In Järvisalo, M.; and Berre, D. L., eds., *Proceedings of the 14th International Workshop on Pragmatics of SAT Co-Located with the 26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023), Alghero, Italy, July 4, 2023*, volume 3545 of *CEUR Workshop Proceedings*, 59–71. CEUR-WS.org.

Rossi, F.; van Beek, P.; and Walsh, T., eds. 2006. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier. ISBN 978-0-444-52726-4.

Schaefer, T. J. 1978. The Complexity of Satisfiability Problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, 216–226. Association for Computing Machinery.

Schulte, C.; and Stuckey, P. J. 2001. When Do Bounds and Domain Propagation Lead to the Same Search Space. In *Proceedings of the 3rd ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, PPDP '01, 115–126. New York, NY, USA: Association for Computing Machinery. ISBN 978-1-58113-388-2.

Vanroose, W.; Devriendt, J.; Tsourous, D.; Verhaeghe, H.; and Guns, T. 2024. Mutational Fuzz Testing for Constraint Modeling Systems. Forthcoming.

Veksler, M.; and Strichman, O. 2010. A Proof-Producing CSP Solver. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, 204–209.

Wallace, M. 1996. Practical Applications of Constraint Programming. *Constraints*, 1(1): 139–168.