Certified MaxSAT Preprocessing

Downloaded from: https://research.chalmers.se, 2026-10-14 16:31 UTC

Citation for the original published paper (version of record):
Ihalainen, H., Oertel, A., Tan, Y. et al (2024). Certified MaxSAT Preprocessing. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 14739 LNAI: 396-418. http://dx.doi.org/10.1007/978-3-031-63498-7_24

N.B. When citing this work, cite the original published paper.

research.chalmers.se offers the possibility of retrieving research publications produced at Chalmers University of Technology. It covers all kind of research output: articles, dissertations, conference papers, reports etc. since 2004. research.chalmers.se is administrated and maintained by Chalmers Library

(article starts on next page)

Certified MaxSAT Preprocessing

Hannes Ihalainen[1(✉)], Andy Oertel[2,3], Yong Kiam Tan[4], Jeremias Berg[1], Matti Järvisalo[1], Magnus O. Myreen[5], and Jakob Nordström[2,3]

[1] Department of Computer Science, University of Helsinki, Helsinki, Finland
{hannes.ihalainen,jeremias.berg,matti.jarvisalo}@helsinki.fi
[2] Lund University, Lund, Sweden
andy.oertel@cs.lth.se, jn@di.ku.dk
[3] University of Copenhagen, Copenhagen, Denmark
[4] Institute for Infocomm Research (I2R), A*STAR, Singapore, Singapore
tanyk1@i2r.a-star.edu.sg
[5] Chalmers University of Technology, Gothenburg, Sweden
myreen@chalmers.se

Abstract. Building on the progress in Boolean satisfiability (SAT) solving over the last decades, maximum satisfiability (MaxSAT) has become a viable approach for solving NP-hard optimization problems. However, ensuring correctness of MaxSAT solvers has remained a considerable concern. For SAT, this is largely a solved problem thanks to the use of proof logging, meaning that solvers emit machine-verifiable proofs to certify correctness. However, for MaxSAT, proof logging solvers have started being developed only very recently. Moreover, these nascent efforts have only targeted the core solving process, ignoring the preprocessing phase where input problem instances can be substantially reformulated before being passed on to the solver proper.

In this work, we demonstrate how pseudo-Boolean proof logging can be used to certify the correctness of a wide range of modern MaxSAT preprocessing techniques. By combining and extending the VeriPB and CakePB tools, we provide formally verified end-to-end proof checking that the input and preprocessed output MaxSAT problem instances have the same optimal value. An extensive evaluation on applied MaxSAT benchmarks shows that our approach is feasible in practice.

Keywords: maximum satisfiability · preprocessing · proof logging · formally verified proof checking

1 Introduction

The development of Boolean satisfiability (SAT) solvers is arguably one of the true success stories of modern computer science—today, SAT solvers are routinely used as core engines in many types of complex automated reasoning systems. One example of this is SAT-based optimization, usually referred to as

© The Author(s) 2024
C. Benzmüller et al. (Eds.): IJCAR 2024, LNAI 14739, pp. 396–418, 2024.
https://doi.org/10.1007/978-3-031-63498-7_24

maximum satisfiability (MaxSAT) solving. The improved performance of SAT solvers, coupled with increasingly sophisticated techniques for using SAT solver calls to reason about optimization problems, have made MaxSAT solvers a powerful tool for tackling real-world NP-hard optimization problems [8].

However, Modern MaxSAT solvers are quite intricate pieces of software, and it has been shown repeatedly in the MaxSAT evaluations [51] that even the best solvers sometimes report incorrect results. This was previously a serious issue also for SAT solvers (see, e.g., [13]), but the SAT community has essentially eliminated this problem by requiring that solvers should be *certifying* [1,53], i.e., not only report whether a given formula is satisfiable or unsatisfiable but also produce a machine-verifiable proof that this conclusion is correct. Many different SAT proof formats such as RUP [33], TraceCheck [7], GRIT [17], and LRAT [16] have been proposed, with DRAT [35,36,74] established as the de facto standard; for the last ten years, proof logging has been compulsory in the (main track of the) SAT competitions [66]. It is all the more striking, then, that until recently no similar developments have been observed in MaxSAT solving.

1.1 Previous Work

A first natural question to ask—since MaxSAT solvers are based on repeated calls to SAT solvers—is why we cannot simply use SAT proof logging also for MaxSAT. The problem is that DRAT can only reason about clauses, whereas MaxSAT solvers argue about costs of solutions and values of objective functions. Translating such claims to clausal form would require an external tool to certify correctness of the translation. Also, such clausal translations incur a significant overhead and do not seem well-adapted for, e.g., counting arguments in MaxSAT.

While there have been several attempts to design proof systems specifically for MaxSAT solving [11,23,39,45,57,58,63–65], none of these have come close to providing a general proof logging solution, because they apply only for very specific algorithm implementations and/or fail to capture the full range of techniques used. Recent papers have instead proposed using pseudo-Boolean proof logging with VeriPB [9,32] to certify correctness of so-called solution-improving solvers [72] and core-guided solvers [4]. Although these works demonstrate, for the first time, practical proof logging for modern MaxSAT solving, the methods developed thus far only apply to the core solving process. This ignores the preprocessing phase, where the input formula can undergo major reformulation. State-of-the-art solvers sometimes use stand-alone preprocessor tools, or sometimes integrate preprocessing-style reasoning more tightly within the MaxSAT solver engine, to speed up the search for optimal solutions. Some of these preprocessing techniques are lifted from SAT to MaxSAT, but there are also native MaxSAT preprocessing methods that lack analogies in SAT solving.

1.2 Our Contribution

In this paper, we show, for the first time, how to use pseudo-Boolean proof logging with VeriPB to produce proofs of correctness for a wide range of prepro-

cessing techniques used in modern MaxSAT solvers. VeriPB proof logging has previously been successfully used not only for core MaxSAT search as discussed above, but also for advanced SAT solving techniques (including symmetry breaking) [9,27,32], subgraph solving [28–30], constraint programming [22,31,54,55], and 0–1 ILP presolving [37], and we add MaxSAT preprocessing to this list.

In order to do so, we extend the VeriPB proof format to include an *output section* where a reformulated output can be presented, and where the pseudo-Boolean proof establishes that this output formula and the input formula are *equioptimal*, i.e., have optimal solutions of the same value. We also enhance CakePB [10,29]—a verified proof checker for pseudo-Boolean proofs—to handle proofs of reformulation. In this way, we obtain an end-to-end formally verified toolchain for certified preprocessing of MaxSAT instances.

It is worth noting that although preprocessing is also a critical component in SAT solving, we are not aware of any tool for certifying reformulations even for the restricted case of decision problems, i.e., showing that formulas are *equisatisfiable*—the DRAT format and tools support proofs that satisfiability of an input CNF formula F implies satisfiability of an output CNF formula G but not the converse direction (except in the special case where F is a subset of G). To the best of our knowledge, our work presents the first practical tool for proving (two-way) equisatisfiability or equioptimality of reformulated problems.

We have performed computational experiments running a MaxSAT preprocessor with proof logging and proof checking on benchmarks from the MaxSAT evaluations [51]. Although there is certainly room for improvements in performance, these experiments provide empirical evidence for the feasibility of certified preprocessing for real-world MaxSAT benchmarks.

1.3 Organization of This Paper

After reviewing preliminaries in Sect. 2, we explain our pseudo-Boolean proof logging for MaxSAT preprocessing in Sect. 3, and Sect. 4 discusses verified proof checking. We present results from a computational evaluation in Sect. 5, after which we conclude with a summary and outlook for future work in Sect. 6.

2 Preliminaries

We write ℓ to denote a literal, i.e., a $\{0,1\}$-valued Boolean variable x or its negation $\overline{x} = 1 - x$. A *clause* $C = \ell_1 \vee \ldots \vee \ell_k$ is a disjunction of literals, where a *unit clause* consists of only one literal. A formula in *conjunctive normal form (CNF)* $F = C_1 \wedge \ldots \wedge C_m$ is a conjunction of clauses, where we think of clauses and formulas as sets so that there are no repetitions and order is irrelevant.

A *pseudo-Boolean (PB) constraint* is a 0–1 linear inequality $\sum_j a_j \ell_j \geq b$, where, when convenient, we can assume all literals ℓ_j to refer to distinct variables and all integers a_j and b to be positive (so-called *normalized form*). A *pseudo-Boolean formula* is a conjunction of such constraints. We identify the clause $C =$

$\ell_1 \vee \cdots \vee \ell_k$ with the pseudo-Boolean constraint $\mathrm{PB}(C) = \ell_1 + \cdots + \ell_k \geq 1$, so a CNF formula F is just a special type of PB formula $\mathrm{PB}(F) = \{\mathrm{PB}(C) \mid C \in F\}$.

A *(partial) assignment* ρ mapping variables to $\{0,1\}$, extended to literals by respecting the meaning of negation, satisfies a PB constraint $\sum_j a_j \ell_j \geq b$ if $\sum_{\ell_j : \rho(\ell_j)=1} a_j \geq b$ (assuming normalized form). A PB formula is satisfied by ρ if all constraints in it are. We also refer to total satisfying assignments ρ as *solutions*. In a *pseudo-Boolean optimization (PBO)* problem we ask for a solution minimizing a given *objective* function $O = \sum_j c_j \ell_j + W$, where c_j and W are integers and W represents a trivial lower bound on the minimum cost.

2.1 Pseudo-Boolean Proof Logging Using Cutting Planes

The pseudo-Boolean proof logging in VeriPB is based on the *cutting planes* proof system [15] with extensions as discussed briefly next. We refer the reader to [14] for and in-depth discussion of cutting planes and to [9,26,37,73] for more detailed information about the VeriPB proof system and format.

A pseudo-Boolean proof maintains two sets of *core constraints* $\mathcal{C}$ and *derived constraints* $\mathcal{D}$ under which the objective O should be minimized. At the start of the proof, $\mathcal{C}$ is initialized to the constraints in the input formula F. Any constraints derived by the rules described below are placed in $\mathcal{D}$, from where they can later be moved to $\mathcal{C}$ (but not vice versa). The proof system semantics preserves the invariant that the optimal value of any solution to $\mathcal{C}$ and to the original input problem F is the same. New constraints can be derived from $\mathcal{C} \cup \mathcal{D}$ by performing *addition* of two constraints or *multiplication* of a constraint by a positive integer, and *literal axioms* $\ell \geq 0$ can be used at any time. Additionally, we can apply *division* to $\sum_j a_j \ell_j \geq b$ by a positive integer d followed by rounding up to obtain $\sum_j \lceil a_j/d \rceil \ell_j \geq \lceil b/d \rceil$, and *saturation* to yield $\sum_j \min\{a_j, b\} \cdot \ell_j \geq b$ (where we again assume normalized form).

The negation of a constraint $C = \sum_j a_j \ell_j \geq b$ is $\neg C = \sum_j a_j \ell_j \leq b - 1$. For a (partial) assignment ρ we write $C|_\rho$ for the *restricted constraint* obtained by replacing literals in C assigned by ρ with their values and simplifying. We say that C *unit propagates* ℓ *under* ρ if $C|_\rho$ cannot be satisfied unless ℓ is assigned to 1. If repeated unit propagation on all constraints in $\mathcal{C} \cup \mathcal{D} \cup \{\neg C\}$, starting with the empty assignment $\rho = \emptyset$, leads to contradiction in the form of an unsatisfiable constraint, we say that C follows by *reverse unit propagation (RUP)* from $\mathcal{C} \cup \mathcal{D}$. Such (efficiently verifiable) RUP steps are allowed in VeriPB proofs as a convenient way to avoid writing out an explicit cutting planes derivation. We use the same notation $C|_\omega$ to denote the result of applying to C a *(partial) substitution* ω, which can map variables not only to $\{0,1\}$ but also to literals, and extend this notation to sets of constraints by taking unions.

In addition to the above rules, which derive semantically implied constraints, there is a *redundance-based strengthening rule*, or just *redundance rule* for short, that can derive non-implied constraints C as long as they do not change the feasibility or optimal value. This can be guaranteed by exhibiting a *witness substitution* ω such that for any total assignment α satisfying $\mathcal{C} \cup \mathcal{D}$ but violating C, the composition $\alpha \circ \omega$ is another total assignment that satisfies $\mathcal{C} \cup \mathcal{D} \cup \{C\}$ and

yields an objective value that is at least as good. Formally, C can be derived from $\mathcal{C} \cup \mathcal{D}$ by exhibiting ω and subproofs for

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \vdash (\mathcal{C} \cup \mathcal{D} \cup \{C\})\restriction_\omega \cup \{O \geq O\restriction_\omega\}, \tag{1}$$

using the previously discussed rules (where the notation $\mathcal{C}_1 \vdash \mathcal{C}_2$ means that the constraints $\mathcal{C}_2$ can be derived from the constraints $\mathcal{C}_1$).

During preprocessing, constraints in the input formula are often deleted or replaced by other constraints, in which case the proof should establish that these deletions maintain equioptimality. Removing constraints from the derived set $\mathcal{D}$ is unproblematic, but unrestricted deletion from the core set $\mathcal{C}$ can clearly introduce spurious better solutions. Therefore, removing C from $\mathcal{C}$ can only be done by the *checked deletion rule*, which requires a proof that the redundance rule can be used to rederive C from $\mathcal{C} \setminus \{C\}$ (see [9] for a more detailed explanation).

Finally, it turns out to be useful to allow replacing O by a new objective O' using an *objective function update rule*, as long as this does not change the optimal value of the problem. Formally, updating the objective from O to O' requires derivations of the two constraints $O \geq O'$ and $O' \geq O$ from the core set $\mathcal{C}$, which shows that any satisfying solution to $\mathcal{C}$ has the same value for both objectives. More details on this rule can be found in [37].

2.2 Maximum Satisfiability

A WCNF instance of (weighted partial) maximum satisfiability $\mathcal{F}^W = (F_H, F_S)$ is a conjunction of two CNF formulas F_H and F_S with *hard* and *soft* clauses, respectively, where soft clauses $C \in F_S$ have positive weights w^C. A solution ρ to $\mathcal{F}^W$ must satisfy F_H and has value $\textsc{cost}(F_S, \rho)$ equal to the sum of weights of all soft clauses not satisfied by ρ. The optimum $\textsc{opt}(\mathcal{F}^W)$ of $\mathcal{F}^W$ is the minimum of $\textsc{cost}(F_S, \rho)$ over all solutions ρ, or ∞ if no solution exists.

State-of-the-art MaxSAT preprocessors such as MaxPre [39,44] take a slightly different *objective-centric* view [5] of MaxSAT instances $\mathcal{F} = (F, O)$ as consisting of a CNF formula F and an objective function $O = \sum_j c_j \ell_j + W$ to be minimized under assignments ρ satisfying F. A WCNF MaxSAT instance $\mathcal{F}^W = (F_H, F_S)$ is converted into objective-centric form $\textsc{ObjMaxSAT}(\mathcal{F}^W) = (F, O)$ by letting the formula $F = F_H \cup \{C \vee b_C \mid C \in F_S, |C| > 1\}$ of $\textsc{ObjMaxSAT}(\mathcal{F}^W)$ consist of the hard clauses of $\mathcal{F}^W$ and the non-unit soft clauses in F_S, each extended with a fresh variable b_C that does not appear in any other clause. The objective $O = \sum_{(\bar{\ell}) \in F_S} w^{(\bar{\ell})} \ell + \sum w^C b_C$ contains literals ℓ for all unit soft clauses $\bar{\ell}$ in F_S as well as literals for all new variables b_C, with coefficients equal to the weights of the corresponding soft clauses. In other words, each unit soft clause $\bar{\ell} \in F_S$ of weight w is transformed into the term $w \cdot \ell$ in the objective function O, and each non-unit soft clause C is transformed into the hard clause $C \vee b_C$ paired with the unit soft clause $(\bar{b}_C)$ with same weight as C. The following observation summarizes the properties of $\textsc{ObjMaxSAT}(\mathcal{F}^W)$ that are central to our work.

Observation 1. *For any solution ρ to a WCNF MaxSAT instance $\mathcal{F}^W$ there exists a solution ρ' to $(F, O) = \textsc{ObjMaxSAT}(\mathcal{F}^W)$ with $O(\rho') = \textsc{cost}(\mathcal{F}^W, \rho)$. Conversely, if ρ' is a solution to $\textsc{ObjMaxSAT}(\mathcal{F}^W)$, then there exists a solution ρ of $\mathcal{F}^W$ for which $\textsc{cost}(\mathcal{F}^W, \rho) \leq O(\rho')$.*

For the second part of the observation, the reason $O(\rho')$ is only an upper bound on $\textsc{cost}(\mathcal{F}^W, \rho)$ is that the encoding forces b_C to be true whenever C is not satisfied by an assignment but not vice versa.

An objective-centric MaxSAT instance (F, O), in turn, clearly has the same optimum as the pseudo-Boolean optimization problem of minimizing O subject to $\text{PB}(F)$. For the end-to-end formal verification, the fact that this coincides with $\textsc{opt}(\mathcal{F}^W)$ needs to be formalized into theorems as shown in Fig. 4.

3 Proof Logging for MaxSAT Preprocessing

We now discuss how pseudo-Boolean proof logging can be used to reason about correctness of MaxSAT preprocessing steps. Our approach maintains the invariant that the current working instance in the preprocessor is synchronized with the PB constraints in the core set $\mathcal{C}$ as described in Sect. 2.2. At the end of each preprocessing step (i.e., application of a preprocessing technique) the set of derived constraints $\mathcal{D}$ is empty. All constraints derived in the proof as described in this section are moved to the core set, and constraints are always removed by checked deletion from the core set. Full details are in the online appendix [40].

3.1 Overview

All our preprocessing steps maintain *equioptimality*, which means that if preprocessing of the WCNF MaxSAT instance $\mathcal{F}^W$ yields the output instance $\mathcal{F}_P^W$, then the equality $\textsc{opt}(\mathcal{F}^W) = \textsc{opt}(\mathcal{F}_P^W)$ is guaranteed to hold. Our preprocessing is *certified*, meaning that we provide a machine-verifiable proof justifying this claimed equality. Our discussion below focuses on input instances that have solutions, but our techniques also handle the—arguably less interesting—case of $\mathcal{F}^W$ not having solutions; details are in the online appendix [40].

An overview of the workflow of our certifying MaxSAT preprocessor is shown in Fig. 1. Given a WCNF instance $\mathcal{F}^W$ as input, the preprocessor proceeds in five stages (illustrated on the left in Fig. 1), and then outputs a preprocessed MaxSAT instance $\mathcal{F}_P^W$ together with a pseudo-Boolean proof that $\textsc{opt}\big(\textsc{ObjMaxSAT}(\mathcal{F}^W)\big) = \textsc{opt}\big(\textsc{ObjMaxSAT}(\mathcal{F}_P^W)\big)$. For certified MaxSAT preprocessing, this proof can then be fed to a formally verified checker as in Sect. 4 to verify that (a) the initial core constraints in the proof correspond exactly to the clauses in $\textsc{ObjMaxSAT}(\mathcal{F}^W)$, (b) each step in the proof is valid, and (c) the final core constraints in the proof correspond exactly to the clauses in $\textsc{ObjMaxSAT}(\mathcal{F}_P^W)$. Below, we provide more details on the five stages of the preprocessing flow.

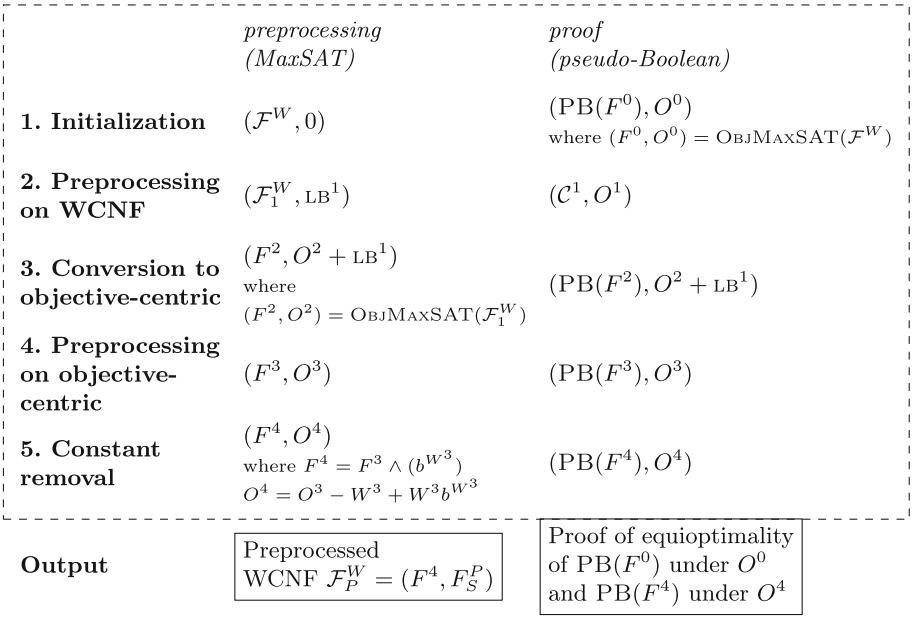

Fig. 1. Overview of the five stages of certified MaxSAT preprocessing of a WCNF instance $\mathcal{F}^W$. The middle column contains the state of the working MaxSAT instance as a WCNF instance and a lower bound on its optimum cost (Stages 1–2), or as an objective-centric instance (Stages 3–5). The right column contains a tuple $(\mathcal{C}, O)$ with the set $\mathcal{C}$ of core constraints, and objective O, respectively, of the proof after each stage.

Stage 1: Initialization. An input WCNF instance $\mathcal{F}^W$ is transformed to pseudo-Boolean format by converting it to an objective-centric representation $(F^0, O^0) = \textsc{ObjMaxSAT}(\mathcal{F}^W)$ and then representing all clauses in F^0 as pseudo-Boolean constraints as described in Sect. 2.2. The VeriPB proof starts out with core constraints $\mathrm{PB}(F^0)$ and objective O^0. The preprocessor maintains a lower bound on the optimal cost of the working instance, which is initialized to 0 for the input $\mathcal{F}^W$.

Stage 2: Preprocessing on the Initial WCNF Representation. During preprocessing on the WCNF representation, a (very limited) set of simplification techniques are applied on the working formula. At this stage the preprocessor removes duplicate, tautological, and blocked clauses [43]. Additionally, hard unit clauses are unit propagated and clauses subsumed by hard clauses are removed. Importantly, the preprocessor is performing these simplifications on a WCNF MaxSAT instance where it deals with hard and soft clauses. As the pseudo-Boolean proof has no concept of hard or soft clauses, the reformulation steps must be expressed in terms of the constraints in the proof. The next example illustrates how reasoning with different types of clauses is logged in the proof.

Example 1. Suppose the working instance has two duplicate clauses C and D. If both are hard, then the proof has two identical constraints $\text{PB}(C)$ and $\text{PB}(D)$ in the core set, and $\text{PB}(D)$ can be deleted since it follows from $\text{PB}(C)$ by reverse unit propagation (RUP). If D is instead a non-unit soft clause, the proof has the constraint $\text{PB}(D \vee b_D)$ and the term $w^D b_D$ in the objective, where b_D does not appear in any other constraint. Then in the proof we (1) remove the RUP constraint $\text{PB}(D \vee b_D)$, (2) introduce $\overline{b}_D \geq 1$ by redundance-based strengthening using the witness $\{b_D \to 0\}$, (3) remove the term $w^D b_D$ from the objective, and (4) delete $\overline{b}_D \geq 1$ with the witness $\{b_D \to 0\}$.

Stage 3: Conversion to Objective-Centric Representation. In order to apply more simplification rules in a cost-preserving way, the working instance $\mathcal{F}_1^W = (F_H^1, F_S^1)$ at the end of Stage 2 is converted into the corresponding objective-centric representation that takes the lower-bound LB inferred during Stage 1 into account. More specifically, the preprocessor next converts its working MaxSAT instance into the objective-centric instance $\mathcal{F}_2 = (F^2, O^2 + \text{LB})$ where $(F^2, O^2) = \text{ObjMaxSAT}(\mathcal{F}_1^W)$.

Here it is important to note that at the end of Stage 2, the core constraints $\mathcal{C}^1$ and objective O^1 of the proof are not necessarily $\text{PB}(F^2)$ and $O^2 + \text{LB}$, respectively. Specifically, consider a unit soft clause $(\overline{\ell})$ of $\mathcal{F}_1^W$ obtained by shrinking a non-unit soft clause $C \supseteq (\overline{\ell})$ of the input instance, with weight w^C. Then the objective function O^2 in the preprocessor will include the term $w^C \ell$ that does not appear in the objective function O^1 in the proof. Instead, O^1 contains the term $w^C b_C$ and $\mathcal{C}^1$ the constraint $\overline{\ell} + b_C \geq 1$ where b_C is the fresh variable added to C in Stage 1. In order to "sync up" the working instance and the proof we (1) introduce $\ell + \overline{b}_C \geq 1$ to the proof with the witness $\{b_C \to 0\}$, (2) update O^1 by adding $w^C \ell - w^C b_C$, (3) remove the constraint $\ell + \overline{b}_C \geq 1$ with the witness $\{b_C \to 0\}$, and (4) remove the constraint $\overline{\ell} + b_C \geq 1$ with witness $\{b_C \to 1\}$. The same steps are logged for all soft unit clauses of $\mathcal{F}_1^W$ obtained during Stage 2. In the following stages, the preprocessor will operate on an objective-centric MaxSAT instance whose clauses correspond exactly to the core constraints of the proof.

Stage 4: Preprocessing on the Objective-Centric Representation. During preprocessing on the objective-centric representation, more simplification techniques are applied to the working objective-centric instance and logged to the proof. We implemented proof logging for a wide range of preprocessing techniques. These include MaxSAT versions of rules commonly used in SAT solving like bounded variable elimination (BVE) [20,68], bounded variable addition [49], blocked clause elimination [43], subsumption elimination, self-subsuming resolution [20,60], failed literal elimination [24,46,75], and equivalent literal substitution [12,48,71]. We also cover MaxSAT-specific preprocessing rules like TrimMaxSAT [61], (group)-subsumed literal (or label) elimination (SLE) [6,44], intrinsic at-most-ones [38,39], binary core removal (BCR) [25,44], label matching [44], and hardening [2,39,56]. Here we give examples for BVE, SLE, label

matching, and BCR—the rest are detailed in the online appendix [40]. In the following descriptions, let (F, O) be the current objective-centric working instance.

Bounded Variable Elimination (BVE) [20,68]. BVE eliminates from F a variable x that does not appear in the objective by replacing all clauses in which either x or $\overline{x}$ appears with the non-tautological clauses in $\{C \vee D \mid C \vee x \in F, D \vee \overline{x} \in F\}$.

An application of BVE is logged as follows: (1) each non-tautological constraint $\mathrm{PB}(C \vee D)$ is added by summing the existing constraints $\mathrm{PB}(C \vee x)$ and $\mathrm{PB}(D \vee \overline{x})$ and saturating, after which (2) each constraint of the form $\mathrm{PB}(C \vee x)$ and $\mathrm{PB}(D \vee \overline{x})$ is deleted with the witness $x \to 1$ or $x \to 0$, respectively.

Label Matching [44]. Label matching allows merging pairs of objective variables that can be deduced to not both be set to 1 by optimal solutions. Assume that (i) F contains the clauses $C \vee b_C$ and $D \vee b_D$, (ii) b_C and b_D are objective variables with the same coefficient w in O, and (iii) $C \vee D$ is a tautology. Then label matching replaces b_C and b_D with a fresh variable b_{CD}, i.e., replaces $C \vee b_C$ and $D \vee b_D$ with $C \vee b_{CD}$ and $D \vee b_{CD}$ and adds $-wb_C - wb_D + wb_{CD}$ to O.

As $C \vee D$ is a tautology there is some literal ℓ such that $\overline{\ell} \in C$ and $\ell \in D$. Label matching is logged via the following steps: (1) introduce the constraint $\overline{b}_C + \overline{b}_D \geq 1$ with the witness $\{b_C \to \ell, b_D \to \overline{\ell}\}$, (2) introduce the constraints $b_{CD} + \overline{b}_C + \overline{b}_D \geq 2$ and $\overline{b}_{CD} + b_C + b_D \geq 1$ by redundance; these correspond to $b_{CD} = b_C + b_D$ which holds even though the variables are binary due to the constraint added in the first step, (3) update the objective by adding $-wb_C - wb_D + wb_{CD}$ to it, (4) introduce the constraints $\mathrm{PB}(C \vee b_{CD})$ and $\mathrm{PB}(D \vee b_{CD})$ which are RUP, (5) delete $\mathrm{PB}(C \vee b_C)$ and $\mathrm{PB}(D \vee b_D)$ with the witness $\{b_C \to \overline{\ell}, b_D \to \ell\}$, (6) delete the constraint $b_{CD} + \overline{b}_C + \overline{b}_D \geq 2$ with the witness $\{b_C \to 0, b_D \to 0\}$ and $\overline{b}_{CD} + b_C + b_D \geq 1$ with the witness $\{b_C \to 1, b_D \to 0\}$, (7) delete $\overline{b}_C + \overline{b}_D \geq 1$ with the witness $\{b_C \to 0\}$.

Subsumed Literal Elimination (SLE) [6,39]. Given two non-objective variables x and y such that (i) $\{C \mid C \in F, y \in C\} \subseteq \{C \mid C \in F, x \in C\}$ and (ii) $\{C \mid C \in F, \overline{x} \in C\} \subseteq \{C \mid C \in F, \overline{y} \in C\}$, subsumed literal elimination (SLE) allows fixing $x = 1$ and $y = 0$. This is proven by (1) introducing $x \geq 1$ and $\overline{y} \geq 1$, both with witness $\{x \to 1, y \to 0\}$, (2) simplifying the constraint database via propagation, and (3) deleting the constraints introduced in the first step as neither x nor y appears in any other constraints after simplification.

If x and y are objective variables, the application of SLE additionally requires that: (iii) the coefficient in the objective of x is at most as high as the coefficient of y. Then the value of x is not fixed as it would incur cost. Instead, only $y = 0$ is fixed and y removed from the objective. Intuitively, conditions (i) and (ii) establish that the values of x and y can always be flipped to 0 and 1, respectively, without falsifying any clauses. If neither of the variables is in the objective, this flip does not increase the cost of any solutions. Otherwise, condition (iii) ensures that the flip does not make the solution worse, i.e., increase its cost.

Binary Core Removal (BCR) [25,44]. Assume that the following four prerequisites hold: (i) F contains a clause $b_C \vee b_D$ for two objective variables b_C and b_D, (ii) b_C and b_D have the same coefficient w in O, (iii) the negations b_C and b_D do not appear in any clause of F, and (iv) both b_C and b_D appear in at least one other clause of F but not together in any other clause of F. Binary core removal replaces all clauses containing b_C or b_D with the non-tautological clauses in $\{C \vee D \vee b_{CD} \mid C \vee b_C \in F, D \vee b_D \in F\}$, where b_{CD} is a fresh variable, and modifies the objective function by adding $-wb_C - wb_D + wb_{CD} + w$ to it.

BCR is logged as a combination of the so-called *intrinsic at-most-ones* technique [38,39] and BVE. Applying intrinsic at most ones on the variables b_C and b_D introduces a new clause $(\bar{b}_C \vee \bar{b}_D \vee b_{CD})$ and adds $-wb_C - wb_D + wb_{CD} + w$ to the objective. Our proof for intrinsic at most ones is the same as the one presented in [4]. As this step removes b_C and b_D from the objective, both can now be eliminated via BVE.

Stage 5: Constant Removal and Output. After objective-centric preprocessing, the final objective-centric instance (F^3, O^3) is converted back to a WCNF instance. Before doing so, the constant term W_3 of O^3 is removed by introducing a fresh variable b^{W_3}, and setting $F^4 = F^3 \wedge (b^{W_3})$ and $O^4 = O^3 - W_3 + W_3 b^{W_3}$. This step is straightforward to prove.

Finally, the preprocessor outputs the WCNF instance $\mathcal{F}_P^W = (F^4, F_S^P)$ that has F^4 as hard clauses. The set F_S^P of soft clauses consists of a unit soft clause $(\bar{\ell})$ of weight c for each term $c \cdot \ell$ in O^4. The preprocessor also outputs the final proof of the fact that the minimum-cost of solutions to the pseudo-Boolean formula $\text{PB}(F^0)$ under O^0 is the same as that of $\text{PB}(F^4)$ under O^4, i.e. that $\text{OPT}(\text{OBJMAXSAT}(\mathcal{F}^W)) = \text{OPT}(\text{OBJMAXSAT}(\mathcal{F}_P^W))$.

3.2 Worked Example of Certified Preprocessing

We give a worked-out example of certified preprocessing of the instance $\mathcal{F}^W = (F_H, F_S)$ where $F_H = \{(x_1 \vee x_2), (\overline{x}_2)\}$ and three soft clauses: $(\overline{x}_1)$ with weight 1, $(x_3 \vee \overline{x}_4)$ with weight 2, and $(x_4 \vee \overline{x}_5)$ with weight 3. The proof for one possible execution of the preprocessor on this input instance is detailed in Table 1.

During Stage 1 (Steps 1–4 in Table 1), the core constraints of the proof are initialized to contain the four constraints corresponding to the hard and non-unit soft clauses of $\mathcal{F}^W$ (IDs (1)–(4) in Table 1), and the objective to $x_1 + 2b_1 + 3b_2$, where b_1 and b_2 are fresh variables added to the non-unit soft clauses of $\mathcal{F}^W$.

During Stage 2 (Steps 5–9), the preprocessor fixes $x_2 = 0$ via unit propagation by removing x_2 from the clause $(x_1 \vee x_2)$, and then removing the unit clause $(\overline{x}_2)$. The justification for fixing $x_2 = 0$ are Steps 5–7. Next the preprocessor fixes $x_1 = 1$ which (i) removes the hard clause (x_1), and (ii) increases the lower bound on the optimal cost by 1. The justification for fixing $x_1 = 1$ are Steps 8 and 9 of Table 1. At this point—at the end of Stage 2—the working instance $\mathcal{F}_1^W = (F_H^1, F_S^1)$ has $F_H^1 = \{\}$ and $F_S^1 = \{(x_3 \vee \overline{x}_4), (x_4 \vee \overline{x}_5)\}$.

Table 1. Example proof produced by a certifying preprocessor. The column (ID) refers to constraint IDs in the pseudo-Boolean proof. The column (Step) indexes all proof logging steps and is used when referring to the steps in the discussion. The letter ω is used for the witness substitution in redundance-based strengthening steps.

Step	ID	Type	Justification	Objective
1	(1)	add $x_1 + x_2 \geq 1$	input	$x_1 + 2b_1 + 3b_2$
2	(2)	add $\overline{x}_2 \geq 1$	input	$x_1 + 2b_1 + 3b_2$
3	(3)	add $x_3 + \overline{x}_4 + b_1 \geq 1$	input	$x_1 + 2b_1 + 3b_2$
4	(4)	add $x_4 + \overline{x}_5 + b_2 \geq 1$	input	$x_1 + 2b_1 + 3b_2$
Unit propagation: fix $x_2 = 0$, constraint (2)				
5	(5)	add $x_1 \geq 1$	(1) + (2)	$x_1 + 2b_1 + 3b_2$
6		delete (1)	RUP	$x_1 + 2b_1 + 3b_2$
7		delete (2)	ω: $\{x_2 \to 0\}$	$x_1 + 2b_1 + 3b_2$
Unit propagation; fix $x_1 = 1$, constraint (5)				
8		add $-x_1 + 1$ to obj	(5)	$2b_1 + 3b_2 + 1$
9		delete (5)	ω: $\{x_1 \to 1\}$	$2b_1 + 3b_2 + 1$
BVE: eliminate x_4				
10	(6)	add $x_3 + b_1 + \overline{x}_5 + b_2 \geq 1$	(3) + (4)	$2b_1 + 3b_2 + 1$
11		delete (3)	ω: $\{x_4 \to 0\}$	$2b_1 + 3b_2 + 1$
12		delete (4)	ω: $\{x_4 \to 1\}$	$2b_1 + 3b_2 + 1$
Subsumed literal elimination: $\overline{b}_2$				
13	(7)	add $\overline{b}_2 \geq 1$	ω: $\{b_2 \to 0, b_1 \to 1\}$	$2b_1 + 3b_2 + 1$
14		add $-3b_2$ to obj	(7)	$2b_1 + 1$
15	(8)	add $x_3 + b_1 + \overline{x}_5 \geq 1$	(6) + (7)	$2b_1 + 1$
16		delete (6)	RUP	$2b_1 + 1$
17		delete (7)	ω: $\{b_2 \to 0\}$	$2b_1 + 1$
Remove objective constant				
18	(9)	add $b_3 \geq 1$	ω: $\{b_3 \to 1\}$	$2b_1 + 1$
19		add $b_3 - 1$ to obj	(9)	$2b_1 + b_3$

In Stage 3, the preprocessor converts its working instance into the objective-centric representation (F, O) where $F = \{(x_3 \vee \overline{x}_4 \vee b_1), (x_4 \vee \overline{x}_5 \vee b_2)\}$ and $O = 2b_1 + 3b_2 + 1$, which exactly matches the core constraints and objective of the proof after Step 9. Thus, in this instance, the conversion does not result in any proof logging steps. Afterwards, during Stage 4 (Steps 10–17), the preprocessor applies BVE in order to eliminate x_4 (Steps 10–12) and SLE to fix b_2 to 0 (Steps 13–17). Finally, Steps 18 and 19 represent Stage 5, i.e., the removal of the constant 1 from the objective. After these steps, the preprocessor outputs the preprocessed instance $\mathcal{F}_P^W = (F_H^P, F_S^P)$, where $F_H^P = \{(x_3 \vee \overline{x}_5 \vee b_1), (b_3)\}$ and F_S^P contains two clauses: $(\overline{b}_1)$ with weight 2, and $(\overline{b}_3)$ with weight 1.

4 Verified Proof Checking for Preprocessing Proofs

This section presents our new workflow for formally verified, end-to-end proof checking of MaxSAT preprocessing proofs based on pseudo-Boolean reasoning; an overview of this workflow is shown in Fig. 2. To realize this workflow, we extended the VeriPB tool and its proof format to support a new *output section* for declaring (and checking) reformulation guarantees between input and output PBO instances (Sect. 4.1); we similarly modified CakePB [29] a verified proof checker to support the updated proof format (Sect. 4.2); finally, we built a verified frontend, CakePBwcnf, which mediates between MaxSAT WCNF instances and PBO instances (Sect. 4.3). Our formalization is carried out in the HOL4 proof assistant [67] using CakeML tools [34,59,70] to obtain a verified executable implementation of CakePBwcnf.

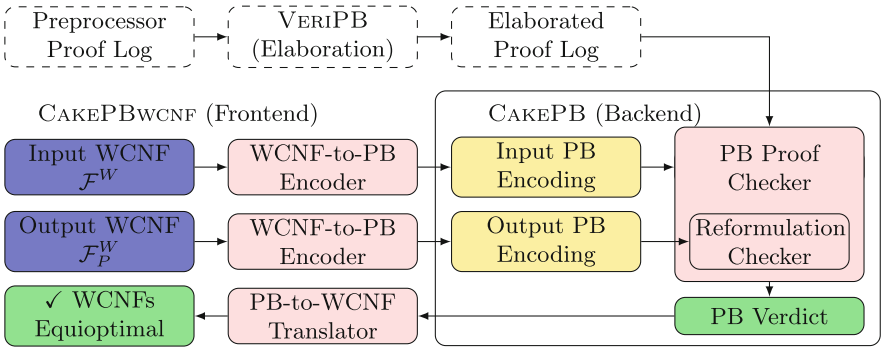

Fig. 2. Workflow for end-to-end verified MaxSAT preprocessing proof checking.

In the workflow in Fig. 2, the MaxSAT preprocessor produces a reformulated output WCNF together with a proof of equioptimality with the input WCNF. This proof is elaborated by VeriPB and then checked by CakePBwcnf, resulting in a verified *verdict*—in case of success, the input and output WCNFs are equioptimal. This workflow also supports verified checking of WCNF MaxSAT solving proofs (where the output parts of the flow are omitted).

4.1 Output Section for Pseudo-Boolean Proofs

Given an input PBO instance (F, O), the VeriPB proof system as described in Sect. 2.1 maintains the invariant that the core constraints $\mathcal{C}$ (and current objective) are equioptimal to the input instance. Utilizing this invariant, the new *output section* for VeriPB proofs allows users to optionally specify an output PBO instance (F', O') at the end of a proof. This output instance is claimed to be a reformulation of the input which is either: (i) *derivable*, i.e., satisfiability of F implies satisfiability of F', (ii) *equisatisfiable* to F, or (iii)

equioptimal to (F, O). These are increasingly stronger claims about the relationship between the input and output instances. After checking a pseudo-Boolean derivation, VeriPB runs reformulation checking which, e.g., for equioptimality, checks that $\mathcal{C} \subseteq F'$, $F' \subseteq \mathcal{C}$, and that the respective objective functions are syntactically equal after normalization; other reformulation guarantees are checked analogously.

The VeriPB tool supports an *elaboration* mode [29], where in addition to checking the proof it also converts it from *augmented format* to *kernel format*. The augmented format contains syntactic sugar rules to facilitate proof logging for solvers and preprocessors like MaxPre, while the kernel format is supported by the formally verified proof checker CakePB. The new output section is passed unchanged from augmented to kernel format during elaboration.

4.2 Verified Proof Checking for Reformulations

There are two main verification tasks involved in extending CakePB with support for the output section. The first task is to verify soundness of all cases of reformulation checking. Formally, the equioptimality of an input PBO instance *fml*, *obj* and its output counterpart *fml′*, *obj′* is specified as follows:

$$\begin{aligned} &\mathsf{sem_output}\ \mathit{fml}\ \mathit{obj}\ \mathsf{None}\ \mathit{fml}'\ \mathit{obj}'\ \mathsf{Equioptimal} \stackrel{\mathsf{def}}{=} \\ &\quad \forall\, v.\ (\exists\, w.\ \mathsf{satisfies}\ w\ \mathit{fml} \wedge \mathsf{eval_obj}\ \mathit{obj}\ w \leq v) \iff \\ &\qquad (\exists\, w'.\ \mathsf{satisfies}\ w'\ \mathit{fml}' \wedge \mathsf{eval_obj}\ \mathit{obj}'\ w' \leq v) \end{aligned}$$

This definition says that, for all values v, the input instance has a satisfying assignment with objective value less than or equal to v iff the output instance also has such an assignment; note that this implies (as a special case) that *fml* is satisfiable iff *fml'* is satisfiable. The verified correctness theorem for CakePB says that *if* CakePB successfully checks a pseudo-Boolean proof in kernel format and prints a verdict declaring equioptimality, then the input and output instances are indeed equioptimal as defined in sem_output.

The second task is to develop verified optimizations to speedup proof steps which occur frequently in preprocessing proofs; some code hotspots were also identified by profiling the proof checker against proofs generated by MaxPre. Similar (unverified) versions of these optimizations are also used in VeriPB. These optimizations turned out to be necessary in practice—they mostly target steps which, when naïvely implemented, have quadratic (or worse) time complexity in the size of the constraint database.

Optimizing Reformulation Checking. The most expensive step in reformulation checking for the output section is to ensure that the core constraints $\mathcal{C}$ are included in the output formula and vice versa (possibly with permutations and duplicity). Here, CakePB normalizes all pseudo-Boolean constraints involved to a canonical form and then copies both $\mathcal{C}$ and the output formula into respective array-backed hash tables for fast membership tests.

Optimizing Redundance and Checked Deletion Rules. A naïve implementation of these two rules would require iterating over the entire constraints database when checking all subproofs in (1) for the right-hand-side constraints $(\mathcal{C} \cup \mathcal{D} \cup \{C\})\restriction_\omega \cup \{O \geq O\restriction_\omega\}$. An important observation here is that preprocessing proofs frequently use substitutions ω that only involve a small number of variables (often a single variable, which in addition is fresh in the important special case of *reification* constraints $z \Leftrightarrow C$ encoding that z is true precisely when the constraint C is satisfied). Consequently, most of the constraints $(\mathcal{C} \cup \mathcal{D} \cup \{C\})\restriction_\omega$ can be skipped when checking redundance because they are unchanged by the substitution. Similarly, the constraint $O \geq O\restriction_\omega$ is expensive to construct when the objective O contains many terms, but this construction can be skipped if no variables being substituted occur in O. CakePB stores a lazily-updated mapping of variables to their occurrences in the constraint database and the objective, which it uses to detect these cases.

The occurrence mapping just discussed is crucial for performance due to the frequency of steps involving witnesses for preprocessing proofs, but incurs some memory overhead in the checker. More precisely, every variable occurrence in any constraint in the database corresponds to exactly one ID in the mapping. Thus, the overhead of storing the mapping is in the worst case quadratic in the number of constraints, but it is still linear in the total space usage for the constraints database.

4.3 Verified WCNF Frontend

The CakePBWcnf frontend mediates between MaxSAT WCNF problems and pseudo-Boolean optimization problems native to CakePB. Accordingly, the correctness of CakePBWcnf is stated in terms of MaxSAT semantics, i.e., the encoding, underlying pseudo-Boolean semantics, and proof system are all formally verified. In order to trust CakePBWcnf, one *only* has to carefully inspect the formal definition of MaxSAT semantics shown in Fig. 3 to make sure that it matches the informal definition in Sect. 2.2. Here, each clause C is paired with a natural number n, where $n = 0$ indicates a hard clause and when $n > 0$ it is the weight of C. The optimal cost of a weighted CNF formula *wfml* is None (representing ∞) if no satisfying assignment to the hard clauses exist; otherwise, it is the minimum cost among all satisfying assignments to the hard clauses.

$$
\begin{aligned}
&\mathsf{sat_hard}\ w\ \mathit{wfml} \stackrel{\mathsf{def}}{=} \forall C.\ \mathsf{mem}\ (0, C)\ \mathit{wfml} \Rightarrow \mathsf{sat_clause}\ w\ C \\
&\mathsf{weight_clause}\ w\ (n, C) \stackrel{\mathsf{def}}{=} \mathsf{if}\ \mathsf{sat_clause}\ w\ C\ \mathsf{then}\ 0\ \mathsf{else}\ n \\
&\mathsf{cost}\ w\ \mathit{wfml} \stackrel{\mathsf{def}}{=} \mathsf{sum}\ (\mathsf{map}\ (\mathsf{weight_clause}\ w)\ \mathit{wfml}) \\
&\mathsf{opt_cost}\ \mathit{wfml} \stackrel{\mathsf{def}}{=} \mathsf{if}\ \neg\exists\, w.\ \mathsf{sat_hard}\ w\ \mathit{wfml}\ \mathsf{then\ None} \\
&\qquad\qquad\qquad\qquad \mathsf{else\ Some}\ (\mathsf{min}_{\mathsf{set}}\ \{\ \mathsf{cost}\ w\ \mathit{wfml} \mid \mathsf{sat_hard}\ w\ \mathit{wfml}\ \}\ )
\end{aligned}
$$

Fig. 3. Formalized semantics for MaxSAT WCNF problems.

There and Back Again. CakePBwcnf contains a verified WCNF-to-PB encoder implementing the encoding described in Sect. 2.2. Its correctness theorems are shown in Fig. 4, where the two lemmas in the top row relate the satisfiability and cost of the WCNF to its PB optimization counterpart after running wcnf_to_pbf (and vice versa), see Observation 1. Using these lemmas, the final theorem (bottom row) shows that equioptimality for two (encoded) PB optimization problems can be *translated* back to equioptimality for the input and preprocessed WCNFs.

$$\vdash \mathsf{wfml_to_pbf}\ \mathit{wfml} = (\mathit{obj}, \mathit{pbf}) \wedge \mathsf{satisfies}\ w\ (\mathsf{set}\ \mathit{pbf}) \Rightarrow \exists w'.\ \mathsf{sat_hard}\ w'\ \mathit{wfml} \wedge \mathsf{cost}\ w'\ \mathit{wfml} \leq \mathsf{eval_obj}\ \mathit{obj}\ w$$

$$\vdash \mathsf{wfml_to_pbf}\ \mathit{wfml} = (\mathit{obj}, \mathit{pbf}) \wedge \mathsf{sat_hard}\ w\ \mathit{wfml} \Rightarrow \exists w'.\ \mathsf{satisfies}\ w'\ (\mathsf{set}\ \mathit{pbf}) \wedge \mathsf{eval_obj}\ \mathit{obj}\ w' = \mathsf{cost}\ w\ \mathit{wfml}$$

$$\vdash \mathsf{full_encode}\ \mathit{wfml} = (\mathit{obj}, \mathit{pbf}) \wedge \mathsf{full_encode}\ \mathit{wfml}' = (\mathit{obj}', \mathit{pbf}') \wedge \mathsf{sem_output}\ (\mathsf{set}\ \mathit{pbf})\ \mathit{obj}\ \mathsf{None}\ (\mathsf{set}\ \mathit{pbf}')\ \mathit{obj}'\ \mathsf{Equioptimal} \Rightarrow \mathsf{opt_cost}\ \mathit{wfml} = \mathsf{opt_cost}\ \mathit{wfml}'$$

Fig. 4. Correctness theorems for the WCNF-to-PB encoding.

Putting Everything Together. The final verification step is to specialize the end-to-end machine code correctness theorem for CakePB to the new frontend. The resulting theorem for CakePBwcnf is shown abridged in Fig. 5; a detailed explanation of similar CakeML-based theorems is available elsewhere [29,69] so we do not go into details here. Briefly, the theorem says that whenever the verdict string "`s VERIFIED OUTPUT EQUIOPTIMAL`" is printed (as a suffix) to the standard output by an execution of CakePBwcnf, then the two input files given on the command line parsed to equioptimal MaxSAT WCNF instances.

$$\begin{aligned}
&\vdash \mathsf{cake_pb_wcnf_run}\ \mathit{cl}\ \mathit{fs}\ \mathit{mc}\ \mathit{ms} \Rightarrow \\
&\quad \exists \mathit{out}\ \mathit{err}. \\
&\qquad \mathsf{extract_fs}\ \mathit{fs}\ (\mathsf{cake_pb_wcnf_io_events}\ \mathit{cl}\ \mathit{fs}) = \\
&\qquad\quad \mathsf{Some}\ (\mathsf{add_stdout}\ (\mathsf{add_stderr}\ \mathit{fs}\ \mathit{err})\ \mathit{out}) \wedge \\
&\qquad (\mathsf{length}\ \mathit{cl} = 4 \wedge \mathsf{isSuffix}\ \texttt{"s VERIFIED OUTPUT EQUIOPTIMAL\textbackslash n"}\ \mathit{out} \Rightarrow \\
&\qquad\quad \exists \mathit{wfml}\ \mathit{wfml}'. \\
&\qquad\quad\ \mathsf{get_fml}\ \mathit{fs}\ (\mathsf{el}\ 1\ \mathit{cl}) = \mathsf{Some}\ \mathit{wfml} \wedge \mathsf{get_fml}\ \mathit{fs}\ (\mathsf{el}\ 3\ \mathit{cl}) = \mathsf{Some}\ \mathit{wfml}' \wedge \\
&\qquad\quad\ \mathsf{opt_cost}\ \mathit{wfml} = \mathsf{opt_cost}\ \mathit{wfml}')
\end{aligned}$$

Fig. 5. Abridged final correctness theorem for CakePBwcnf.

5 Experiments

We updated the MaxSAT preprocessor MaxPre 2.1 [39,42,44] to MaxPre 2.2 which now produces proof logs in the VeriPB format [10]. MaxPre 2.2 is

available at the MaxPre 2 repository [50]. The generated proofs were elaborated using VeriPB [73] and then checked by the verified proof checker CakePB-wcnf. As benchmarks we used the 558 weighted and 572 unweighted MaxSAT instances from the MaxSAT Evaluation 2023 [52].

The experiments were conducted on 11th Gen Intel(R) Core(TM) i5-1145G7 @ 2.60 GHz CPUs with 16 GB of memory, a solid state drive as storage, and Rocky Linux 8.5 as operating system. Each benchmark ran exclusively on a node and the memory was limited to 14 GB. The time for MaxPre was limited to 300 s. There is an option to let MaxPre know about this time limit, but we did not use this option since MaxPre then decides which techniques to try based on how much time remains. This would have made it very hard to get reliable measurements of the overhead when proof logging is switched on in the preprocessor. The time limits for both VeriPB and CakePBwcnf were set to 6000 s to get as many instances checked as possible.

The main focus of our evaluation was the default setting of MaxPre, which does not use some of the techniques mentioned in Sect. 3 (or the online appendix [40]). We also conducted experiments with all techniques enabled to check the correctness of the proof logging implementation for all preprocessing techniques. The data and source code from our experiments can be found in [41].

The goal of the experiments was to answer the following questions:

RQ1. How much extra time is required to write the proof for the preprocessor?
RQ2. How long does proof checking take compared to proof generation?

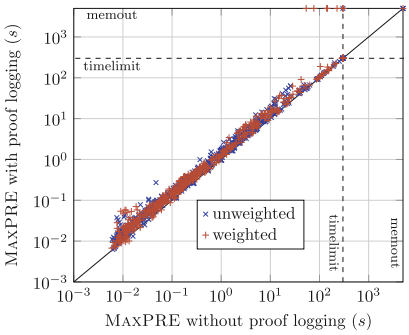

Fig. 6. Proof logging overhead for MaxPre.

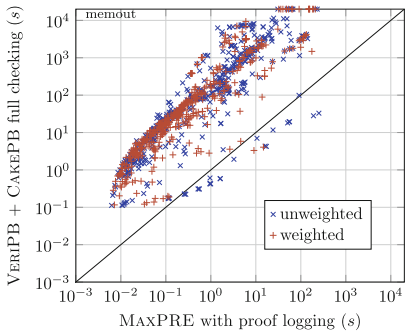

Fig. 7. MaxPre vs. combined proof checking running time.

To answer the first question, in Fig. 6 we compare MaxPre with and without proof logging. In total, 1081 instances were successfully preprocessed by MaxPre without proof logging. With proof logging enabled, 8 fewer instances were preprocessed due to either time- or memory-outs. For the successfully preprocessed instances, the geometric mean of the proof logging overhead is 46% of the

running time, and 95% of the instances were preprocessed with proof logging in at most twice the time required without proof logging.

Our comparison between proof generation and proof checking is based on the 1073 instances for which preprocessing with proof logging was successful. Out of these, 1021 instances were successfully checked and elaborated by VeriPB. For 991 instances the verdicts were confirmed by the formally verified proof checker CakePBwcnf, with the remaining instances being time- or memory-outs. This shows the practical viability of our approach, as the vast majority of preprocessing proofs were checked within the resource limits.

A scatter plot comparing the running time of MaxPre with proof logging enabled against the combined checking process is shown in Fig. 7. For the combined checking time, we only consider the instances that have been successfully checked by CakePBwcnf. In the geometric mean, the time for the combined verified checking pipeline of VeriPB elaboration followed by CakePBwcnf checking is 113× the preprocessing time of MaxPre. A general reason for this overhead is that the preprocessor has more MaxSAT application-specific context than the pseudo-Boolean checker, so the preprocessor can log proof steps without performing the actual reasoning while the checker must ensure that those steps are sound in an application-agnostic way. An example for this is reification: as the preprocessor knows its reification variables are fresh, it can easily emit redundance steps that witness on those variables; but the checker has to verify freshness against its own database. Similar behaviour has been observed in other applications of pseudo-Boolean proof logging [27,37].

To analyse further the causes of proof checking overhead, we also compared VeriPB to CakePBwcnf. The checking of the elaborated kernel proof with CakePBwcnf is 6.7× faster than checking and elaborating the augmented proof with VeriPB. This suggests that the bottleneck for proof checking is VeriPB; VeriPB *without* elaboration is about 5.3× slower than CakePBwcnf. As elaboration is a necessary step before running CakePBwcnf, improving the performance of VeriPB would benefit the performance of the pipeline as a whole. One specific feature that seems desirable would be to augment RUP rule applications with LRAT-style hints [16], so that VeriPB would not need to perform unit propagation to elaborate RUP steps to cutting planes derivations. Though these types of engineering challenges are important to address, they are beyond the scope of the current paper and we have to leave them as future work.

6 Conclusion

In this work, we show how to use pseudo-Boolean proof logging to certify correctness of the MaxSAT preprocessing phase, extending previous work for the main solving phase in unweighted model-improving solvers [72] and general core-guided solvers [4]. As a further strengthening of previous work, we present a fully formally verified toolchain which provides end-to-end verification of correctness.

In contrast to SAT solving, there is a rich variety of techniques in maximum satisfiability solving, and it still remains to design pseudo-Boolean proof logging

methods for general, weighted, model-improving MaxSAT solvers [21,47,62] and *implicit hitting set (IHS)* MaxSAT solvers [18,19] with *abstract cores* [3]. Nevertheless, our work adds further weight to the conclusion that pseudo-Boolean reasoning seems like a very promising foundation for MaxSAT proof logging. We are optimistic that this work is another step on the path towards general adoption of proof logging in the context of SAT-based optimization.

Acknowledgments. This work has been financially supported by the University of Helsinki Doctoral Programme in Computer Science DoCS, the Research Council of Finland under grants 342145 and 346056, the Swedish Research Council grants 2016-00782 and 2021-05165, the Independent Research Fund Denmark grant 9040-00389B, the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, and by A*STAR, Singapore. Part of this work was carried out while some of the authors participated in the extended reunion of the semester program *Satisfiability: Theory, Practice, and Beyond* in the spring of 2023 at the Simons Institute for the Theory of Computing at UC Berkeley. We also acknowledge useful discussions at the Dagstuhl workshops 22411 *Theory and Practice of SAT and Combinatorial Solving* and 23261 *SAT Encodings and Beyond*. The computational experiments were enabled by resources provided by LUNARC at Lund University.

References

1. Alkassar, E., Böhme, S., Mehlhorn, K., Rizkallah, C., Schweitzer, P.: An introduction to certifying algorithms. IT - Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik **53**(6), 287–293 (2011)
2. Ansótegui, C., Bonet, M.L., Gabàs, J., Levy, J.: Improving SAT-based weighted MaxSAT solvers. In: Milano, M. (eds.) Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP 2012). LNCS, vol. 7514, pp. 86–101. Springer, Cham (2012). https://doi.org/10.1007/978-3-642-33558-7_9
3. Berg, J., Bacchus, F., Poole, A.: Abstract cores in implicit hitting set MaxSat solving. In: Pulina, L., Seidl, M. (eds.) Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT 2020). LNCS, vol. 12178, pp. 277–294. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51825-7_20
4. Berg, J., Bogaerts, B., Nordström, J., Oertel, A., Vandesande, D.: Certified core-guided MaxSAT solving. In: Pientka, B., Tinelli, C. (eds.) Proceedings of the 29th International Conference on Automated Deduction (CADE-29). LNCS, vol. 14132, pp. 1–22. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-38499-8_1
5. Berg, J., Järvisalo, M.: Unifying reasoning and core-guided search for maximum satisfiability. In: Calimeri, F., Leone, N., Manna, M. (eds.) Proceedings of the 16th European Conference on Logics in Artificial Intelligence (JELIA 2019). LNCS, vol. 11468, pp. 287–303. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-19570-0_19
6. Berg, J., Saikko, P., Järvisalo, M.: Subsumed label elimination for maximum satisfiability. In: Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016). FAIA, vol. 285, pp. 630–638. IOS Press (2016)
7. Biere, A.: Tracecheck (2006). http://fmv.jku.at/tracecheck/

8. Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, 2nd edn., vol. 336. IOS Press, February 2021
9. Bogaerts, B., Gocht, S., McCreesh, C., Nordström, J.: Certified dominance and symmetry breaking for combinatorial optimisation. J. Artif. Intell. Res. **77**, 1539–1589 (2023). Preliminary version in AAAI 2022
10. Bogaerts, B., McCreesh, C., Myreen, M.O., Nordström, J., Oertel, A., Tan, Y.K.: Documentation of VeriPB and CakePB for the SAT competition 2023, March 2023. Available at https://satcompetition.github.io/2023/checkers.html
11. Bonet, M.L., Levy, J., Manyà, F.: Resolution for Max-SAT. Artif. Intell. **171**(8–9), 606–618 (2007)
12. Brafman, R.I.: A simplifier for propositional formulas with many binary clauses. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **34**(1), 52–59 (2004)
13. Brummayer, R., Lonsing, F., Biere, A.: Automated testing and debugging of SAT and QBF solvers. In: Strichman, O., Szeider, S. (eds.) Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT 2010). LNCS, vol. 6175, pp. 44–57. Springer, Cham (2010). https://doi.org/10.1007/978-3-642-14186-7_6
14. Buss, S.R., Nordström, J.: Proof complexity and SAT solving. In: Biere et al. [8], Chap. 7, pp. 233–350
15. Cook, W., Coullard, C.R., Turán, G.: On the complexity of cutting-plane proofs. Discret. Appl. Math. **18**(1), 25–38 (1987)
16. Cruz-Filipe, L., Heule, M.J.H., Hunt Jr., W.A., Kaufmann, M., Schneider-Kamp, P.: Efficient certified RAT verification. In: de Moura, L. (eds.) Proceedings of the 26th International Conference on Automated Deduction (CADE-26). LNCS, vol. 10395, pp. 220–236. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63046-5_14
17. Cruz-Filipe, L., Marques-Silva, J.P., Schneider-Kamp, P.: Efficient certified resolution proof checking. In: Legay, A., Margaria, T. (eds.) Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2017). LNCS, vol. 10205, pp. 118–135. Springer, Cham (2017). https://doi.org/10.1007/978-3-662-54577-5_7
18. Davies, J., Bacchus, F.: Solving MAXSAT by solving a sequence of simpler SAT instances. In: Lee, J. (eds.) Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP 2011). LNCS, vol. 6876, pp. 225–239. Springer, Cham (2011). https://doi.org/10.1007/978-3-642-23786-7_19
19. Davies, J., Bacchus, F.: Exploiting the power of MIP solvers in MAXSAT. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 166–181. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39071-5_13
20. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT 2005). LNCS, vol. 3569, pp. 61–75. Springer, Cham (2005). https://doi.org/10.1007/11499107_5
21. Eén, N., Sörensson, N.: Translating pseudo-Boolean constraints into SAT. J. Satisfiability Boolean Model. Comput. **2**(1–4), 1–26 (2006)
22. Elffers, J., Gocht, S., McCreesh, C., Nordström, J.: Justifying all differences using pseudo-Boolean reasoning. In: Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI 2020), pp. 1486–1494, February 2020

23. Filmus, Y., Mahajan, M., Sood, G., Vinyals, M.: MaxSAT resolution and subcube sums. In: Pulina, L., Seidl, M. (eds.) Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT 2020). LNCS, vol. 12178, pp. 295–311. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51825-7_21
24. Freeman, J.W.: Improvements to propositional satisfiability search algorithms. Ph.D. thesis, University of Pennsylvania (1995)
25. Gimpel, J.F.: A reduction technique for prime implicant tables. In: Proceedings of the 5th Annual Symposium on Switching Circuit Theory and Logical Design, (SWCT 1964), pp. 183–191. IEEE Computer Society (1964)
26. Gocht, S.: Certifying correctness for combinatorial algorithms by using pseudo-Boolean reasoning, Ph.D. thesis, Lund University, June 2022. https://portal.research.lu.se/en/publications/certifying-correctness-for-combinatorial-algorithms-by-using-pseu
27. Gocht, S., Martins, R., Nordström, J., Oertel, A.: Certified CNF translations for pseudo-Boolean solving. In: Proceedings of the 25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022). Leibniz International Proceedings in Informatics (LIPIcs), vol. 236, pp. 16:1–16:25, August 2022
28. Gocht, S., McBride, R., McCreesh, C., Nordström, J., Prosser, P., Trimble, J.: Certifying solvers for clique and maximum common (connected) subgraph problems. In: Simonis, H. (eds.) Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP 2020). LNCS, vol. 12333, pp. 338–357. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58475-7_20
29. Gocht, S., McCreesh, C., Myreen, M.O., Nordström, J., Oertel, A., Tan, Y.K.: End-to-end verification for subgraph solving. In: Proceedings of the 368h AAAI Conference on Artificial Intelligence (AAAI 2024), pp. 8038–8047, February 2024
30. Gocht, S., McCreesh, C., Nordström, J.: Subgraph isomorphism meets cutting planes: solving with certified solutions. In: Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020), pp. 1134–1140, July 2020
31. Gocht, S., McCreesh, C., Nordström, J.: An auditable constraint programming solver. In: Proceedings of the 28th International Conference on Principles and Practice of Constraint Programming (CP 2022). Leibniz International Proceedings in Informatics (LIPIcs), vol. 235, pp. 25:1–25:18, August 2022
32. Gocht, S., Nordström, J.: Certifying parity reasoning efficiently using pseudo-Boolean proofs. In: Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI 2021), pp. 3768–3777, February 2021
33. Goldberg, E., Novikov, Y.: Verification of proofs of unsatisfiability for CNF formulas. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2003), pp. 886–891, March 2003
34. Guéneau, A., Myreen, M.O., Kumar, R., Norrish, M.: Verified characteristic formulae for CakeML. In: Yang, H. (eds.) Proceedings of the 26th European Symposium on Programming (ESOP 2017). LNCS, vol. 10201, pp. 584–610. Springer, Cham (2017). https://doi.org/10.1007/978-3-662-54434-1_22
35. Heule, M.J.H., Hunt Jr., W.A., Wetzler, N.: Trimming while checking clausal proofs. In: Proceedings of the 13th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2013), pp. 181–188, October 2013
36. Heule, M.J.H., Hunt Jr., W.A., Wetzler, N.: Verifying refutations with extended resolution. In: Bonacina, M.P. (eds.) Proceedings of the 24th International Conference on Automated Deduction (CADE-24). LNCS, vol. 7898, pp. 345–359. Springer, Cham (2013). https://doi.org/10.1007/978-3-642-38574-2_24

37. Hoen, A., Oertel, A., Gleixner, A., Nordström, J.: Certifying MIP-based presolve reductions for 0–1 integer linear programs. In: Proceedings of the 21st International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2024), May 2024, to appear
38. Ignatiev, A., Morgado, A., Marques-Silva, J.: RC2: an efficient MaxSAT solver. J. Satisfiability Boolean Model. Comput. **11**(1), 53–64 (2019)
39. Ihalainen, H., Berg, J., Järvisalo, M.: Clause redundancy and preprocessing in maximum satisfiability. In: Blanchette, J., Kovács, L., Pattinson, D. (eds.) Proceedings of the 11th International Joint Conference on Automated Reasoning (IJCAR 2022). LNCS, vol. 13385, pp. 75–94. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-10769-6_6
40. Ihalainen, H., et al.: Certified MaxSAT preprocessing (2024). https://arxiv.org/abs/2404.17316. Full-length version
41. Ihalainen, H., et al.: Experimental Repository for "Certified MaxSAT Preprocessing", February 2024. https://doi.org/10.5281/zenodo.10630852
42. Jabs, C., Berg, J., Ihalainen, H., Järvisalo, M.: Preprocessing in SAT-based multi-objective combinatorial optimization. In: Proceedings of the 29th International Conference on Principles and Practice of Constraint Programming (CP 2023). Leibniz International Proceedings in Informatics (LIPIcs), vol. 280, pp. 18:1–18:20 (2023)
43. Järvisalo, M., Biere, A., Heule, M.: Blocked clause elimination. In: Esparza, J., Majumdar, R. (eds.) Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2010). LNCS, vol. 6015, pp. 129–144. Springer, Cham (2010). https://doi.org/10.1007/978-3-642-12002-2_10
44. Korhonen, T., Berg, J., Saikko, P., Järvisalo, M.: MaxPre: an extended MaxSAT preprocessor. In: Gaspers, S., Walsh, T. (eds.) Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT 2017). LNCS, vol. 10491, pp. 449–456. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66263-3_28
45. Larrosa, J., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: A framework for certified Boolean branch-and-bound optimization. J. Autom. Reason. **46**(1), 81–102 (2011)
46. Le Berre, D.: Exploiting the real power of unit propagation lookahead. Electron. Notes Discrete Math. **9**, 59–80 (2001)
47. Le Berre, D., Parrain, A.: The Sat4j library, release 2.2. J. Satisfiability Boolean Model. Comput. **7**, 59–64 (2010)
48. Li, C.M.: Integrating equivalency reasoning into Davis-Putnam procedure. In: Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence, pp. 291–296. AAAI Press/The MIT Press (2000)
49. Manthey, N., Heule, M.J.H., Biere, A.: Automated reencoding of Boolean formulas. In: Biere, A., Nahir, A., Vos, T. (eds.) 8th International Haifa Verification Conference (HVC 2012), Revised Selected Papers. LNCS, vol. 7857, pp. 102–117. Springer, Cham (2013). https://doi.org/10.1007/978-3-642-39611-3_14
50. MaxPre 2: MaxSAT preprocessor. https://bitbucket.org/coreo-group/maxpre2
51. MaxSAT evaluations: Evaluating the state of the art in maximum satisfiability solver technology. https://maxsat-evaluations.github.io/
52. MaxSAT evaluation 2023, July 2023. https://maxsat-evaluations.github.io/2023
53. McConnell, R.M., Mehlhorn, K., Näher, S., Schweitzer, P.: Certifying algorithms. Comput. Sci. Rev. **5**(2), 119–161 (2011)

54. McIlree, M., McCreesh, C.: Proof logging for smart extensional constraints. In: Proceedings of the 29th International Conference on Principles and Practice of Constraint Programming (CP 2023). Leibniz International Proceedings in Informatics (LIPIcs), vol. 280, pp. 26:1–26:17, August 2023
55. McIlree, M., McCreesh, C., Nordström, J.: Proof logging for the circuit constraint. In: Proceedings of the 21st International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2024), May 2024, to appear
56. Morgado, A., Heras, F., Marques-Silva, J.: Improvements to core-guided binary search for MaxSAT. In: Cimatti, A., Sebastiani, R. (eds.) Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT 2012). LNCS, vol. 7317, pp. 284–297. Springer, Cham (2012). https://doi.org/10.1007/978-3-642-31612-8_22
57. Morgado, A., Ignatiev, A., Bonet, M.L., Marques-Silva, J.P., Buss, S.R.: DRMaxSAT with MaxHS: first contact. In: Janota, M., Lynce, I. (eds.) Proceedings of the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT 2019). LNCS, vol. 11628, pp. 239–249. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24258-9_17
58. Morgado, A., Marques-Silva, J.: On validating Boolean optimizers. In: Proceedings of the 23rd IEEE International Conference on Tools with Artificial Intelligence, (ICTAI 2011), pp. 924–926 (2011)
59. Myreen, M.O., Owens, S.: Proof-producing translation of higher-order logic into pure and stateful ML. J. Funct. Program. **24**(2–3), 284–315 (2014)
60. Ostrowski, R., Grégoire, É., Mazure, B., Saïs, L.: Recovering and exploiting structural knowledge from CNF formulas. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 185–199. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46135-3_13
61. Paxian, T., Raiola, P., Becker, B.: On preprocessing for weighted MaxSAT. In: Henglein, F., Shoham, S., Vizel, Y. (eds.) VMCAI 2021. LNCS, vol. 12597, pp. 556–577. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67067-2_25
62. Paxian, T., Reimer, S., Becker, B.: Dynamic polynomial watchdog encoding for solving weighted MaxSAT. In: Beyersdorff, O., Wintersteiger, C. (eds.) Proceedings of the 21st International Conference on Theory and Applications of Satisfiability Testing (SAT 2018). LNCS, vol. 10929, pp. 37–53. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94144-8_3
63. Py, M., Cherif, M.S., Habet, D.: Towards bridging the gap between SAT and MaxSAT refutations. In: Proceedings of the 32nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2020), pp. 137–144, November 2020
64. Py, M., Cherif, M.S., Habet, D.: A proof builder for Max-SAT. In: Li, CM., Manyá, F. (eds.) Proceedings of the 24th International Conference on Theory and Applications of Satisfiability Testing (SAT 2021). LNCS, vol. 12831, pp. 488–498. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-80223-3_33
65. Py, M., Cherif, M.S., Habet, D.: Proofs and certificates for Max-SAT. J. Artif. Intell. Res. **75**, 1373–1400 (2022)
66. The International SAT Competitions web page. http://www.satcompetition.org
67. Slind, K., Norrish, M.: A brief overview of HOL4. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) TPHOLs 2008. LNCS, vol. 5170, pp. 28–32. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71067-7_6
68. Subbarayan, S., Pradhan, D.K.: NiVER: non-increasing variable elimination resolution for preprocessing SAT instances. In: Hoos, H.H., Mitchell, D.G. (eds.) SAT

2004. LNCS, vol. 3542, pp. 276–291. Springer, Heidelberg (2005). https://doi.org/10.1007/11527695_22
69. Tan, Y.K., Heule, M.J.H., Myreen, M.O.: Verified propagation redundancy and compositional UNSAT checking in CakeML. Int. J. Softw. Tools Technol. Transf. **25**, 167–184 (2023). Preliminary version in TACAS 2021
70. Tan, Y.K., Myreen, M.O., Kumar, R., Fox, A.C.J., Owens, S., Norrish, M.: The verified CakeML compiler backend. J. Funct. Program. **29**, e2:1–e2:57 (2019)
71. Van Gelder, A.: Toward leaner binary-clause reasoning in a satisfiability solver. Ann. Math. Artif. Intell. **43**(1), 239–253 (2005)
72. Vandesande, D., De Wulf, W., Bogaerts, B.: QMaxSATpb: a certified MaxSAT solver. In: Gottlob, G., Inclezan, D., Maratea, M. (eds.) Proceedings of the 16th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR 2022). LNCS, vol. 13416, pp. 429–442. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-15707-3_33
73. VeriPB: Verifier for pseudo-Boolean proofs. https://gitlab.com/MIAOresearch/software/VeriPB
74. Wetzler, N., Heule, M.J.H., Hunt Jr., W.A.: DRAT-trim: efficient checking and trimming using expressive clausal proofs. In: Sinz, C., Egly, U. (eds.) Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT 2014). LNCS, vol. 8561, pp. 422–429. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09284-3_31
75. Zabih, R., McAllester, D.A.: A rearrangement search strategy for determining propositional satisfiability. In: Proceedings of the 7th National Conference on Artificial Intelligence (AAAI 1988), pp. 155–160. AAAI Press/The MIT Press (1988)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.