A Verified Decision Procedure for Pseudo-Boolean Formulas

Tobias Philipp and **Anna Tigunova**
International Center for Computational Logic, Technische Universität Dresden, Germany

Abstract

Pseudo-Boolean formulas consist of constraints of the form $\sum_{i=1}^{n} w_i \cdot x_i \lhd k$, where x_i are propositional literals, $w_i \in \mathbb{Z}$, $k \in \mathbb{Z}$, and arise in planning, scheduling and optimization problems. We describe an *efficient and easily verifiable* decision procedure for pseudo-Boolean formulas, that is based on encoding PB formulas into the propositional satisfiability problem with the cutting-edge sequential weighted counter encoding. State-of-the-art SAT solvers that emit unsatisfiability proofs are used to solve the resulting instances. The combination of a verified translation to SAT, and certified SAT solvers leads to a verified decision procedure for PB formulas. The verification of the encoding is carried out in the Coq proof assistant.

Introduction

The satisfiability problem is one of the most prominent problems in theoretical computer science and artificial intelligence and was successfully applied in planning [16, 25], as well as scheduling [12]. In these applications pseudo-Boolean (PB) constraints often occur. PB constraints can be represented as $\sum_{i=1}^{n} w_i \cdot x_i \lhd k$, where x_i are propositional literals, $w_i \in \mathbb{Z}$, $k \in \mathbb{Z}$, and hold if and only if the weighted sum over the x_i literals is $\lhd$-related with k. For example, the *packing problem* can be formalized by means of PB formulas: we pack the given items of sizes $a_1, \ldots, a_n$ into the minimal number of containers y_i such that the volume V of the container is not exceeded. Then, we obtain:

$$\sum_{j=1}^{n} a_i x_{i,j} \leq V y_i \quad \text{for all } i \in \{1 \ldots n\}$$

Our research is based on the need for formalized and verified decision procedures that do not depend on unverified components and pen and paper proofs. Our contribution is an easy pipeline of an efficient encoding to solve PB constraints, where each step is certified, leading to *efficient and easily verifiable* decision procedure for PB formulas. Following Eén and Sörensson [11], we translate PB constraints into formulas in conjunctive normal form, and run afterwards a SAT solver that finds a solution to the original PB formula or reports unsatisfiability of the problem. This SAT-based approach is highly successful since SAT solving has significantly advanced over the last decades, and solvers are yearly evaluated in international SAT competitions. Their performance over e.g. hardware and software verification [7] has improved, enabling them to become widespread tools in the industry. Moreover, often formalizations of problems are close to the clause normal form. Therefore, the use SAT solvers is an attractive approach to tackle these problems.

Due to these reasons many translations from PB constraints into propositional formulas in conjunctive normal form have been proposed: *naive*, *nested*, *watchdog* [22], *adder-* and *sorting networks* [21, 11], *binary merge* [20], *binary decision diagrams* [11, 1], and the *sequential weighted counter encoding (SWC)* [13]. Specialized encodings also exist for cardinality constraints [2, 26] that are subsumed by PB constraints.

Among all these, we chose the SWC encoding due to the following reasons. First, it produces in the PB benchmarks 2011 and 2010 less clauses in 99% of the cases than the asymptotic best *binary merge encoding* [13]. Second, SWC satisfies two important properties: unit propagation in the encoding detects inconsistencies and maintain generalized arc consistency [13]. Third, experiments on all new instances of the PB evaluation 2012 have shown that the sequential weighted counter encoding performs better than *adder and sorting networks*, *watchdog* and *binary merge* encodings, within a timeout of 30 minutes [23]. Finally, the encoding is relatively simple to describe, which allows us to verify it with little effort using interactive theorem provers such as Coq.

The correctness of the PB decision procedure is then reduced to the correctness of the underlying SAT solver. Unfortunately, SAT solver can be buggy: three solvers, that participated in the SAT competition in 2009, and five solvers that participated in the SAT competition in 2007 gave incorrect results [8]. The critical case is when a formula is reported to be unsatisfiable, as unsatisfiability is hard to see. Also subtle bugs in different components of satisfiability solvers were reported in [15]. Therefore, state-of-the-art SAT solver like *Riss* or *Lingeling* [6] , emit certificates in the DRAT [29] format, which are afterwards checked by an independent program, such as *drat-trim* [29]. Still, a proof of correctness of the translation from PB constraints to CNF is necessary. Figure 1 presents our approach to the problem of justification of the whole procedure. It is a combination of a mechanically verified translation from PB constraints

into CNF and a certified SAT solver. Such a combination then forms a verified decision procedure for PB formulas.

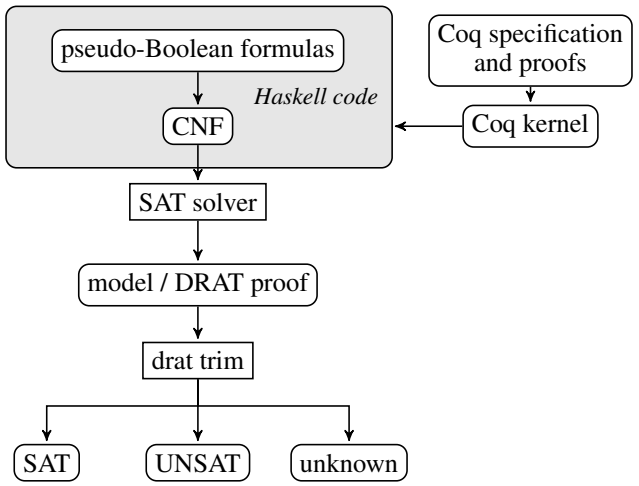

Figure 1: The certified SAT approach combined with a mechanically verified PB encoding: PB formulas can be reduced to satisfiability of the propositional satisfiability question. Then, the SAT solver emits a proof for unsatisfiability, which can be checked independently.

The rest of this paper is structured as follows: First, we give a short introduction to the proof assistant Coq, propositional logic, the DRAT format for the certification of SAT solvers, as well as PB-constraints and the SWC encoding. Then we present the formalization of the SWC encoding. After that we describe the specification and present the proof ideas. Then we focus on integration of all parts of the procedure, and evaluate our approach on selected instances from the PB competition 2016. Finally, we discuss some future improvements.

Background

The Coq Proof Assistant

We use the Coq proof assistant [5] to write our specifications and proofs. Coq is based on the calculus of inductive constructions [9] and combines higher-order logic with a typed functional programming language. In Coq we define functions in the lambda calculus. Moreover, we can express mathematical theorems and can proof them interactively. The syntax of Coq is similar to that of other typed functional programming languages. Accepted proofs can be independently checked by a small certification kernel. Finally, we can automatically extract Haskell programs from Coq theories, which give us fully verified programs. Therefore, Coq provides an essential link from the PB to the SAT encoding, because it ensures the correctness of the translation. In this way we obtain a verified Haskell program that transforms PB constraints into CNF.

Propositional Logic

In propositional logic we consider an infinite set of propositional variables $\mathcal{V}$. A literal L is either a propositional variable A or its negation $\neg A$. The complement of a literal L is denoted by $\overline{L}$, i.e. $\overline{A} = \neg A$ and $\overline{\neg A} = A$. Clauses are lists of literals and represent disjunction. Formulas are lists of clauses, representing a conjunction.

The semantics of formulas is built on interpretations. An *interpretation* I is a mapping from the set $\mathcal{V}$ of all Boolean variables to the set $\{\top, \bot\}$ of truth values, represented by the set of variables, which are true in this interpretation. The interpretation I *satisfies* the variable A, in symbols $I \models A$, if and only if $A \in I$. Similarly, I *satisfies* the clause C, in symbols $I \models C$, if and only if there is a literal $L \in C$, such that $I \models L$. For a formula F, the interpretation I *satisfies* the formula F, in symbols $I \models F$, if and only if for every clause $C \in F$, we find that the interpretation I satisfies the clause C. A *model* I of a formula F is an interpretation I, that satisfies the formula F. If such a model I of F exists, the formula F is *satisfiable*. Otherwise, the formula F is *unsatisfiable*.

Let C and D be two clauses and L be a literal such that $L \in C$ and $\overline{L} \in D$. Then, the *resolvent of C and D upon L* is $(C \setminus \{L\}) \cup (D \setminus \{\overline{L}\})$. A *tautological* clause is a clause containing A and $\neg A$ for some variable A.

DRAT Refutations

As the second step to certify the output of our system we use *drat-trim*, which is a checker for unsatisfiability proofs in DRAT format: The Resolution Asymmetric Tautology (RAT) property is based on the notion of *asymmetric literal addition* [15], where $\mathsf{ALA}_F(C)$ is defined as

$$C \cup \{\overline{L} \mid \{L_1, \ldots, L_n, L\} \in F \text{ and } \{L_1, \ldots, L_n\} \subseteq C\}$$

We consider the recursive application of asymmetric literal addition:

$$\begin{array}{lcl} \mathsf{ALA}_F(C) \uparrow 0 & = & C \\ \mathsf{ALA}_F(C) \uparrow n+1 & = & \mathsf{ALA}_F(\mathsf{ALA}_F(C) \uparrow n) \end{array}$$

A clause C is an *asymmetric tautology (AT)* w.r.t. the formula F, if there is $n \in \mathbb{N}$ such that the clause $\mathsf{ALA}_F(C) \uparrow n$ is a tautology.

Järvisalo et al. introduced the following redundancy criteria based on an asymmetric tautologies in [15]: The clause C is a *resolution asymmetric tautology (RAT) upon L w.r.t. F*, if *(1)* the clause C is an asymmetric tautology w.r.t. the formula F, or *(2)* there is a literal $L \in C$ such that the resolvent of C and D upon L is an asymmetric tautology w.r.t. the formula F for every $D \in F$ with $\overline{L} \in D$.

Several important techniques in SAT solvers can be characterized in terms of RAT: *bounded variable elimination and addition* [27, 10, 24, 19], *blocked clause elimination* [14], *blocked clause addition* [17, 15], *probing* [18], and *extended resolution and reencoding* [19, 28].

A *DRAT refutation* for a formula is a sequence of clauses to the empty clause having the RAT property with respect to the preceding clauses, together with deletion information. If a DRAT refutation for a formula F exists, then F is unsatisfiable.

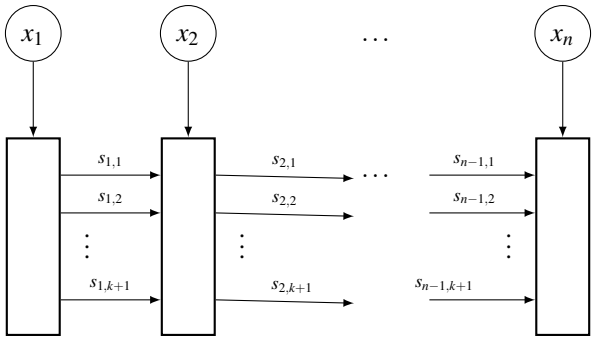

Figure 2: Illustration of the sequential weighted counter as a circuit. Each box represents a unit that adds the weight of the literal to the partial sum, if the variable x_i is true.

Pseudo-Boolean Constraints and the Sequential Weighted Counter

We consider pseudo-Boolean constraints $\sum_{i=1}^{n} w_i x_i \leq k$ in *normal form* [3] that meet the following conditions: *1.* $1 \leq k < n$, *2.* weights are between 1 and k, and *3.* no literal occurs more than once. For example, the PB constraint $2x_1 + x_2 + 2x_1 \leq 1$ is not in normal form, since x_1 occurs more than once. Notice that it is not a restriction to consider PB constraints in normal form, as all PB constraints can be transformed into a semantically equivalent PB constraint in normal form. Moreover, normalized constraints do occur in applications, such as in translations from maximum satisfiability into PB optimization.

A normalized PB constraint can then be transformed to an equivalent propositional formula. For instance, the expression $3x_1 + 2x_2 + 4x_3 \leq 5$ is a normalized constraint, that is semantically equivalent to the following formula:

$$\neg(x_1 \wedge x_3) \wedge \neg(x_2 \wedge x_3)$$

The *sequential weighted counter (SWC)* [13] is an encoding of normalized PB constraints into formulas in conjunctive normal form. It generalizes the *sequential counter encoding* [26] for at-most-k constraints (expressions of the form $\sum x_i \leq k$, where only k variables can be mapped to $\top$). SWC can also be seen as a variant of a *BDD-based encodings for monotone predicates* [1].

The SWC encoding works as follows: given a sequence of variables $x_1, \ldots, x_n$ and associated weights $w_1, \ldots, w_n$, the sequential weighted counter uses the auxiliary propositional variables $s_{i,j}$, where $1 \leq i \leq n$ and $1 \leq j \leq k$. The variable $s_{i,j}$ expresses that the sum up to the ith variable is at least j.

Example 1. *For instance, consider the PB-constraint* $3x_1 + 2x_2 + 4x_3 < 5$ *and an interpretation* I *where* $I(x_1) = \top$, $I(x_2) = \bot$ *and* $I(x_3) = \top$. *Then, the following variables are all mapped to* $\top$ *under* I*:*

$$s_{1,1}, s_{1,2}, s_{1,3}, \quad s_{2,1}, s_{2,2}, s_{2,3}, \quad s_{3,1}, s_{3,2}, s_{3,3},$$

$$s_{4,1}, s_{4,2}, s_{4,3}, s_{4,4}, s_{4,5}$$

To get more insight into the encoding process, refer to Fig. 2, which illustrates the structure as a circuit.

Formalization

We carried out the formalization of the syntax and semantics of propositional logic, PB constraints, and the SWC encoding in the Coq interactive theorem prover. As most of the formalization is straightforward, we present only the details of SWC encoding.

We express the sequential weighted counter encoding in Coq as follows: let wf be a function $\mathsf{wf} : \mathbb{N} \mapsto \mathbb{N}$, that assigns weights to literals.

Definition 1 (Formalization of SWC in Coq)**.** *The sequential weighted counter for a normalized PB constraint* $\sum_{i=1}^{n} w_i x_i \leq k$*, denoted by* $\mathsf{SWC}(n, k, wf)$*, is a formula in conjunctive normal form consisting of the following parts:*

1. *The first formula states the monotonicity of the sum, as only positive non-zero weights are allowed. If the sum up to the variable* x_i *is at least* j*, then the sum to the next variable* x_{i+1} *is at least* j*. Formally,*

$$(\neg s_{i-1,j} \vee s_{i,j}) \quad \text{for all } 2 \leq i < n \text{ and } 1 \leq j \leq k \qquad (F_1)$$

2. *The second formula states that the sum up to the variable* x_i *must be at least* w_i*, if* x_i *is satisfied by the considered interpretation:*

$$(\neg x_i \vee s_{i,j}) \quad \text{for all } 1 \leq i < n, \text{ and } 1 \leq j \leq w_i + 1 \qquad (F_2)$$

3. *The third formula states that the sum up to the variable* x_i *increases by* w_i*:*

$$(\neg s_{i-1,j} \vee \neg x_i \vee s_{i,j+w_i}) \text{ for all } 2 \leq i < n, 1 \leq j < k - w_i + 1 \qquad (F_3)$$

4. *The fourth formula expresses when an interpretation cannot be a model of the PB-constraint, i.e. when the weighted sum exceeds* k*. Formally:*

$$(\neg s_{i-1,k+1-w_i} \vee \neg x_i) \quad \text{for all } 1 \leq 2 < n - 1 \qquad (F_4)$$

Example 2 (SWC Encoding)**.** *Consider the PB-constraint* $3x_1 + 2x_2 + 4x_3 \leq 5$*, where* wf *is a function giving the corresponding weights. Then* $\mathsf{SWC}(3, 5, \mathsf{wf}) = F_1 \wedge F_2 \wedge F_3 \wedge F_4$*, where*

$$\begin{aligned}
F_1 = \;& (\neg s_{1,1} \vee s_{2,1}) \wedge (\neg s_{1,2} \vee s_{2,2}) \wedge (\neg s_{1,3} \vee s_{2,3}) \wedge \\
& (\neg s_{1,4} \vee s_{2,4}) \wedge (\neg s_{1,5} \vee s_{2,5}) \\
F_2 = \;& (\neg x_1 \vee s_{1,1}) \wedge (\neg x_1 \vee s_{1,2}) \wedge (\neg x_1 \vee s_{1,3}) \wedge \\
& (\neg x_2 \vee s_{2,1}) \wedge (\neg x_2 \vee s_{2,2}) \\
F_3 = \;& (\neg s_{1,1} \vee \neg x_2 \vee s_{2,3}) \wedge (\neg s_{1,2} \vee \neg x_2 \vee s_{2,4}) \wedge \\
& (\neg s_{1,3} \vee \neg x_2 \vee s_{2,5}) \\
F_4 = \;& (\neg s_{1,4} \vee \neg x_2) \wedge (\neg s_{2,2} \vee \neg x_3)
\end{aligned}$$

Specification

Given a normalized PB constraint, its SWC encoding is semantically equivalent only with respect to the original variables, because we use auxiliary variables. Therefore, an interpretation satisfying a PB-constraint must be extended to a model over the SWC encoding while preserving the truth value for the original PB variables. Formally, an interpretation J is an *extension* of the interpretation I, if it agrees with the truth values for all literals of the form x_i for every $i \in \mathbb{N}$ (in contrast to variables of the form $s_{i,j}$).

The specification of encodings consists of two parts: completeness and soundness. Completeness means that whenever an interpretation satisfies the PB-constraint, it can be extended to an interpretation that satisfies the formula. Soundness means that whenever an interpretation does not satisfy the PB-constraint, it falsifies the formula:

Definition 2 (Encoding Specification)**.** *The formula F encodes a PB constraint $\sum w_i x_i \leq k$ iff for every interpretation I, we find the following:*

- *If I satisfies $\sum w_i x_i \leq k$, then there is an extension J of I such that J is a model of F.*
- *If I does not satisfy $\sum w_i x_i \leq k$, then I is not a model of F.*

Theorem 1 (Main Theorem)**.** *The formula $\mathsf{SWC}(n,k,\mathsf{wf})$ encodes the $\sum w_i x_i \leq k$, if the weight function wf gives the corresponding weights.*

Proof of the Main Theorem

Our proof of the main theorem is based on the informal proofs given in [13] and [26], and splits in two essential components:

1. *Soundness*

   The proof is done by contradiction. It uses the following intuition: suppose J is the model of *SWC* encoding. Therefore, its restriction I to the variables x_i must also be a model of the PB-constraint. Otherwise, it would violate one of the invariants, springing from the formulas $F_1 \ldots F_4$, which essentially construct the SWC encoding.

2. *Completeness*

   To prove completeness we need to show that the model I of PB formula can be extended a model J of the SWC specification.

   For that we need to take the PB model I and show that we can construct an interpretation, that assigns the variables $s_{i,j}$ to truth values, such that they satisfy the constraints $F_1, \ldots, F_4$. This can be done using following two properties: *1.* the weighted sum is monotone, as long as he weights are positive, and *2.* the extended model J does not change the initial variables x_i.

For further details see [13].

Integration

Our decision procedure works as follows: It accepts a variant of the input format used in the PB competitions and evaluations, where a PB formula has to be given in normal form. We then compute the SWC encoding, which is done by a program that was automatically extracted from the Coq theory. This is done for each occurring PB constraint. The resulting formulas are then syntactically transformed into the DIMACS CNF format such that it can be given to the SAT solver.

More precisely, at this step it is necessary to create unique variables $s_{i,j}$ for each PB constraint and preserve the original names of x_k variables. Notice that this syntactical transformation was not verified using Coq. After running the certifying SAT solver *Lingeling*, we check unsatisfiability results with *drat-trim*.

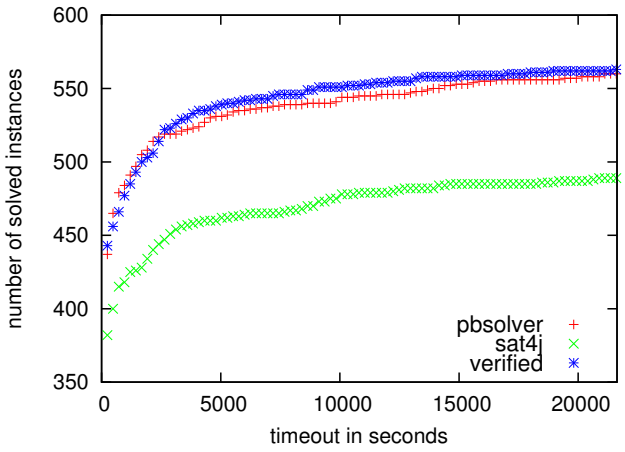

Figure 3: Comparison of the two PB solvers *pbsolver* and *sat4j* with our verified checker on selected instances of the PB evaluation 2016.

Experimental Evaluation

We compare the verified PB solver with the two existing PB solvers *sat4j* and *pbsolver*. *pbsolver* is a *MiniSAT*-based PB solver that combines several different encodings and chooses the best among them [23]. *sat4j* is a Java library for solving Boolean satisfaction and optimization problems and can also handle PB formulas [4]. For the evaluation, we considered formulas from the PB competition 2016 that were satisfaction problems containing only linear constraints and small integers (in total 777 formulas). The experiment was performed on a cluster of Intel Xeon E5-2670 CPUs with 8 cores and 20 MB level 3 cache that is shared by all cores. Each program was run on a single core, with a memory limit of 2.5GB and a time limit of 6 hours.

A dedicated normalization procedure was applied before running the verified Haskell encoder. If we do not consider instances that timed out, the time needed for the dedicated normalization procedure and our verified transformation is in average 140 seconds, whereas the average run time of the SAT solver and the DRAT checker is 624 seconds. The solver *sat4j* solved 489 instances, and *pbsolver* as well as the verified PB program solved 563 instances, and the resulting run times are visualized in Fig. 3. The performance gap of *pbsolver* over our verified approach can be explained that we use a better SAT solver. From the data we conclude that our verified PB solver is efficient enough to be useful in practice.

Discussion & Conclusion

The contribution of this work is the introduction of a routine for solving PB formulas with the SAT approach, i.e. we encode PB constraints using a mechanically verified program and afterwards using a SAT solver that emits unsatisfiability proofs that can be independently checked. For the transformation of PB constraints to conjunctive normal form the SWC encoding was used as it is highly-efficient in practice and easy to verify. The translation was formalized in the

Coq proof assistant. We claim that we have developed a trustworthy decision procedure for checking satisfiability of PB formulas, assuming that the Coq proof assistant as well as the proof checker are correct.

An alternative approach for developing a verified decision procedure for the PB problem is to develop specialized reasoning procedures, and show them to be correct, or to use a certifying-based approach: both methods seem to be more difficult than the presented approach. However, a major disadvantage of our approach is that some PB problems need specialized reasoning procedures, such as cardinality reasoning such as the Fourier Motzkin procedure, and there are no available certifying SAT solvers that apply these methods.

Our formalization of the problem contain specifications and proofs for the SWC and the *sequential counter encoding*, which is optimized for cardinality constraints. For the SWC encoding, we splitted the proof into 46 lemmas, for the sequential counter encoding we used 94 lemmas. In total, the project consists 3800 lines of code. In the process of formalization and proofs, we did not observe flaws in the published proofs. The system is available at:
`https://iccl.inf.tu-dresden.de/web/VPB`
We believe that the presented approach can easily be adapted for several other applications, such as weighted maximum satisfiability, planning, and answer-set programming. These are widely used to model a large scope of problems, therefore, the system that we developed might become useful to produce verified proofs in multiple domains. We are also working on the question how we can express cardinality reasoning in the DRAT format to improve the performance of the presented approach.

Acknowledgements The authors thank the ZIH of TU Dresden for providing the computational resources to produce the experimental data for the empirical evaluation.

References

[1] Ignasi Abío et al. "BDDs for pseudo-Boolean constraints: revisited". In: *SAT 2011*. Ed. by Karem A. Sakallah and Laurent Simon. Vol. 6695. LNCS. Heidelberg: Springer, 2011, pp. 61–75.

[2] Roberto Asín et al. "Cardinality Networks and Their Applications". In: *SAT 2009*. Ed. by Oliver Kullmann. Heidelberg: Springer, 2009, pp. 167–180.

[3] Peter Barth. *A Davis-Putnam Based Enumeration Algorithm for Linear Pseudo-Boolean Optimization*. Tech. rep. Max Plank Institute for Computer Science, 1995.

[4] Daniel Le Berre and Anne Parrain. "The Sat4j library, release 2.2". In: *JSAT* 7.2-3 (2010), pp. 59–6. URL: `http://jsat.ewi.tudelft.nl/content/volume7/JSAT7_4_LeBerre.pdf`.

[5] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development, Coq'Art: The Calculus of Inductive Constructions*. Springer, 2010.

[6] A. Biere. "Yet another Local Search Solver and Lingeling and Friends Entering the SAT Competition 2014". In: ed. by Anton Belov et al. Vol. B-2014-2. Department of Computer Science Series of Publications B. University of Helsinki, Helsinki, Finland, 2014, pp. 39–40.

[7] Armin Biere et al. "Symbolic Model Checking Using SAT Procedures instead of BDDs". In: *DAC 1999*. 1999, pp. 317–320.

[8] Robert Brummayer, Florian Lonsing, and Armin Biere. "Automated Testing and Debugging of SAT and QBF Solvers". English. In: *SAT 2010*. Ed. by Ofer Strichman and Stefan Szeider. Vol. 6175. LNCS. Springer Berlin Heidelberg, 2010, pp. 44–57.

[9] Thierry Coquand and Gérard Huet. "The calculus of constructions". In: *Information and Computation* 76.2–3 (1988), pp. 95 –120.

[10] Niklas Eén and Armin Biere. "Effective Preprocessing in SAT Through Variable and Clause Elimination". In: *SAT 2005*. Ed. by Fahiem Bacchus and Toby Walsh. Vol. 3569. LNCS. Heidelberg: Springer, 2005, pp. 61–75.

[11] Niklas Eén and Niklas Sörensson. "Translating pseudo-Boolean constraints into SAT". In: *JSAT* 2 (2006). Ed. by Daniel Le Berre and Laurent Simon, pp. 1–26.

[12] Peter Großmann et al. "Solving Periodic Event Scheduling Problems with SAT". In: *IEA/AIE 2012*. Ed. by He Jiang et al. Vol. 7345. LNCS. Springer, 2012, pp. 166–175. ISBN: 978-3-642-31086-7.

[13] S. Hölldobler, N. Manthey, and P. Steinke. "A Compact Encoding of Pseudo-Boolean Constraints into SAT". In: *Proc. 35th German Conf. on A.I. (KI 2012)*. Ed. by B. Glimm and A. Krüger. Vol. 7526. LNCS. Heidelberg: Springer, 2012, pp. 107–118.

[14] Matti Järvisalo, Armin Biere, and Marijn Heule. "Blocked Clause Elimination". In: *TACAS 2010*. Ed. by Javier Esparza and Rupak Majumdar. Vol. 6015. LNCS. Heidelberg: Springer, 2010, pp. 129–144.

[15] Matti Järvisalo, Marijn J. H. Heule, and Armin Biere. "Inprocessing rules". In: *IJCAR 2012*. Ed. by Bernhard Gramlich, Dale Miller, and Uli Sattler. Vol. 7364. LNCS. Heidelberg: Springer, 2012, pp. 355–370.

[16] Henry A. Kautz and Bart Selman. "Planning as Satisfiability". In: *ECAI 1992*. Ed. by Bernd Neumann. 1992, pp. 359–363.

[17] Oliver Kullmann. "On a Generalization of Extended Resolution". In: *Discrete Applied Mathematics* 96-97 (1999), pp. 149–176.

[18] Inês Lynce and João P. Marques-Silva. "Probing-Based Preprocessing Techniques for Propositional Satisfiability". In: *ICTAI 2003*. Sacramento, California, USA: IEEE Computer Society, 2003, pp. 105–110. ISBN: 0–7695–2038–3.

[19] Norbert Manthey, Marijn J. H. Heule, and Armin Biere. “Automated Reencoding of Boolean Formulas”. In: *HVC 2012*. Ed. by Armin Biere, Amir Nahir, and Tanja Vos. Vol. 7857. LNCS. Heidelberg: Springer, 2013, pp. 102–117.

[20] Norbert Manthey, Tobias Philipp, and Peter Steinke. “A More Compact Translation of Pseudo-Boolean Constraints into CNF Such That Generalized Arc Consistency Is Maintained”. In: *Annual German Conference on AI (KI 2014)*. Ed. by Carsten Lutz and Michael Thielscher. Vol. 8736. LNCS. Springer, 2014, pp. 123–134.

[21] David E. Muller and Franco P. Preparata. “Bounds to Complexities of Networks for Sorting and for Switching”. In: *Journal of the ACM* 22.2 (1975), pp. 195–201.

[22] Yacine Boufkhad Olivier Bailleux and Olivier Roussel. “New Encodings of Pseudo-Boolean Constraints into CNF”. In: *SAT 2009*. Ed. by Oliver Kullmann. Heidelberg: Springer, 2009, pp. 181–194.

[23] Tobias Philipp and Peter Steinke. “PBLib – A C++ Toolkit for Encoding Pseudo-Boolean Constraints into CNF”. In: *SAT 2015*. 2015.

[24] Jussi Rintanen. “Compact Representation of Sets of Binary Constraints”. In: *ECAI 2006*. Ed. by Gerhard Brewka et al. Vol. 141. Frontiers in Artificial Intelligence and Applications. IOS Press, 2006, pp. 143–147.

[25] Jussi Rintanen. “Engineering Efficient Planners with SAT”. In: *ECAI 2012*. Ed. by Luc De Raedt et al. Vol. 242. Frontiers in Artificial Intelligence and Applications. IOS Press, 2012, pp. 684–689.

[26] Carsten Sinz. “Towards an Optimal CNF Encoding of Boolean Cardinality Constraints”. In: ed. by Peter van Beek. Vol. 3709. LNCS. Springer, 2005, pp. 827–831.

[27] Sathiamoorthy Subbarayan and Dhiraj K. Pradhan. “NiVER: Non-increasing Variable Elimination Resolution for Preprocessing SAT Instances”. In: *SAT 2004*. Ed. by Holger H. Hoos and David G. Mitchell. Vol. 3542. LNCS. Heidelberg: Springer, 2005, pp. 276–291.

[28] G. S. Tseitin. “On the complexity of derivations in the propositional calculus”. In: *Studies in Mathematics and Mathematical Logic* Part II (1968), pp. 115–125.

[29] Nathan Wetzler, Marijn J.H. Heule, and Warren A. Hunt. “DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs”. In: *SAT 2014*. Ed. by Carsten Sinz and Uwe Egly. Vol. 8561. LNCS. Springer, 2014, pp. 422–429.