DRAT Proofs for XOR Reasoning

Tobias Philipp[1] and Adrián Rebola-Pardo[2]

[1] International Center for Computational Logic
Technische Universität Dresden, 01062 Dresden, Germany
tobias.philipp@tu-dresden.de
[2] TU Wien (Austria)
arebolap@forsyte.tuwien.ac.at

Abstract. Unsatisfiability proofs in the DRAT format became the *de facto* standard to increase the reliability of contemporary SAT solvers. We consider the problem of generating proofs for the XOR reasoning component in SAT solvers and propose two methods: direct translation transforms every XOR constraint addition inference into a DRAT proof, whereas T-translation avoids the exponential blow-up in direct translations by using fresh variables. T-translation produces DRAT proofs from Gaussian elimination records that are polynomial in the size of the input CNF formula. Experiments show that a combination of both approaches with a simple prediction method outperforms the BDD-based method.

1 Introduction

The satisfiability problem (SAT) is a paramount problem in computer science and artificial intelligence. Modern SAT solvers based on the DPLL algorithm [10] use many advanced techniques such as *clause learning* [27], *clause removal* [2, 12], *formula simplifications* [11, 19] and specialized reasoning procedures such as XOR reasoning [20, 21, 29, 31]. These improvements led to a spectacular performance of conflict-driven satisfiability solvers. However, even intensively-tested systems contain bugs [7, 24], and today, unsatisfiability proofs in the DRAT proof format [32] are the *de facto* standard in the SAT community. In fact, DRAT format proof generation is a requirement in the main track of the SAT competition 2016. Recently, the DRAT format received media attention because SAT solvers solved the Pythagorean Triples Problem and its 200 TB proof was expressed in this format [17].

XOR constraints frequently arise in applications such as logical cryptoanalysis [9] and pseudo-Boolean encodings [13]; 71% of instances in the application track of the SAT Competition 2014 contain XOR constraints. Gaussian elimination can be used as an efficient reasoning procedure over XOR constraints [31]. Currently, none of the state-of-the-art SAT solvers, like `Lingeling` [5], `Riss` [23] and `CryptoMiniSAT` [30], are able to produce proofs for XOR reasoning. Inability to produce unsatisfiability proofs for XOR reasoning can seriously hinder the

[2] Supported by the LogiCS doctoral program W1255-N23 of the Austrian Science Fund (FWF), and by the Vienna Science and Technology Fund (WWTF) through grant VRG11-005.

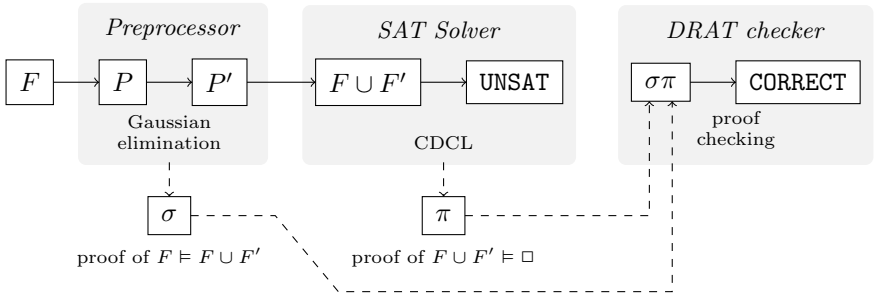

Fig. 1. Certificate-based approach for XOR reasoning: F is the input formula, P an XOR formula which is simplified to P'; F' represents the encoding as a CNF formula of P', which is then refuted. DRAT proofs π and σ are generated from Gaussian elimination preprocessing and CDCL execution. Together, they provide a full unsatisfiability certificate for F.

performance of SAT solvers when certificate generation is required, since XOR reasoning must then be disabled. This makes the solver much less efficient for some problems, e.g. in cryptography.

The problem we address here is to generate DRAT proofs in XOR reasoning, stated as an open problem in [16, 28]. As shown in Figure 1, SAT solvers with XOR reasoning modules [29, 31] detect XOR constraints P in the input formula F and apply Gaussian elimination to find small (unary and binary in `CoProcessor`), implied XOR constraints P', which are then encoded back to the formula as F'. A standard CDCL-driven SAT solver solves the new formula, producing a DRAT refutation π for $F \cup F'$. Still, the DRAT refutation does not include a witness that the XOR detection and reasoning procedure in the SAT solver was correct. To obtain a full unsatisfiability proof of F, a DRAT proof σ of $F \cup F'$ from F is needed. In this paper, we discuss how to generate such a DRAT proof.

Sinz and Biere proposed a BDD-based approach [28] which can be modified to express DRAT proofs for XOR reasoning. Heule et al. have shown that *symmetry breaking* [1] can be expressed in DRAT [15]. Although these techniques could be covered by allowing additional inference rules in the proof system, novel efficient methods for proof checking would need to be developed. Hence, verification of proof checkers would become much more costly; by generating DRAT proofs verification is avoided, since the proof itself is a certificate of correctness. Furthermore, the obtained DRAT proof is a refutation of the original clauses, so the XOR constraint detection algorithm needs not be verified.

Our contributions

1. We present *direct translations* which are based on DP-like variable elimination, and *T-translations* that avoid the exponential blow up by introducing Tseitin variables. Moreover, we describe how one can adapt the BDD-based approach [28] to handle XOR constraints.

2. We prove that T-translations are polynomial in the size of the input formula, when Gaussian elimination was used, whereas the direct translation is an exponential proof in general.
3. Experiments show that the T-translation is practical as it produces proofs of reasonable size for the problems in the SAT Competition 2014. Moreover, the direct and the T-translation outperform the BDD-based approach.

2 Background

2.1 Propositional Logic and XOR Constraints

We consider a totally ordered, countably infinite set of *propositional variables*. A *literal* L is either a variable A or its *negation* $\neg A$. The *complement* of a literal L is denoted by $\overline{L}$. A *clause* is a finite disjunction of literals $(L_1 \vee \cdots \vee L_n)$. *XOR constraints* are expressions of the form $[A_1, \ldots, A_n]^k$, where A_i is a variable for every $1 \leq i \leq n$, and $k \in \{0, 1\}$. A finite set of clauses (XOR constraints, resp.) F is called a *CNF formula* (*XOR formula*, resp.).

Semantics are given by *interpretations* that map formulas to truth values: the interpretation I satisfies a clause $C = (L_1 \vee \cdots \vee L_n)$, if I satisfies some literal among $L_1, \ldots, L_n$; it satisfies an XOR constraint $X = [A_1, \ldots, A_n]^k$ if the number of the variables A_i satisfied by I has the parity of k (i.e. odd if $k = 1$ and even if $k = 0$); and it satisfies a formula F if I satisfies all elements of F. We follow the usual notion of semantic equivalence.

We assume that clauses and XOR constraints are normalized: a literal may appear only once in a clause, and a variable at most once in an XOR constraint. The normal form can be obtained by removing duplicated literals in CNF clauses as well as pairs of occurrences of the same variable in XOR constraints. Observe that these operations preserve semantic equivalence, e.g. $[p, q, r, r, q]^0$ is semantically equivalent to $[p]^0$. Consider two XOR constraints

$$X = [A_1, \ldots, A_n, B_1, \ldots, B_p]^k \qquad Y = [A_1, \ldots, A_n, B'_1, \ldots, B'_q]^l$$

where the A_i, B_i and B'_i are pairwise distinct variables. The *addition* of X and Y, denoted by $X \triangle Y$, is $[B_1, \ldots, B_p, B'_1, \ldots, B'_q]^{k \oplus l}$ where $\oplus$ represents the binary XOR operation. The *resolvent of clauses C and D upon L* is the clause obtained by removing L from C, and $\overline{L}$ from D, and afterwards combining them disjunctively. A *tautology* is a clause containing a complementary pair of literals.

2.2 Gaussian Elimination-based XOR Reasoning in SAT Solvers

Contemporary SAT solvers such as `CryptoMiniSAT` detect XOR constraints in their direct encoding in the input formula [18]. The *direct encoding* [14] $\mathcal{D}(X)$ of an XOR constraint $X = [A_1, \ldots, A_n]^k$ is the CNF formula that contains all clauses of the form $(L_1 \vee \cdots \vee L_n)$, where the L_i are either A_i or $\neg A_i$, and the number of negated literals L_i is not equal to k modulo 2. The direct encoding of an XOR constraint is the unique CNF formula semantically equivalent to it.

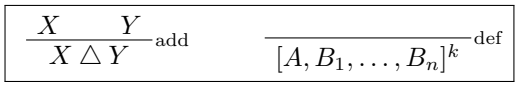

Fig. 2. XOR proof system inference rules: addition (left) and XOR definition (right), where A is a fresh variable and $k \in \{0, 1\}$

Example 1 (Direct encoding). Let $X = [p, q, r]^0$, $Y = [p, q, s]^1$ and $Z = [r, s]^1$. Their direct encodings are:

$$\mathcal{D}(X) = \{(\neg p \vee \neg q \vee \neg r),\ (\neg p \vee q \vee r),\ (p \vee \neg q \vee r),\ (p \vee q \vee \neg r)\}$$
$$\mathcal{D}(Y) = \{(p \vee q \vee s),\ (p \vee \neg q \vee \neg s),\ (\neg p \vee q \vee \neg s),\ (\neg p \vee \neg q \vee s)\}$$
$$\mathcal{D}(Z) = \{(r \vee s),\ (\neg r \vee \neg s)\}$$

Note that $\mathcal{D}([]^0)$ is the empty formula, while $\mathcal{D}([]^1)$ is the unsatisfiable singleton formula consisting of the empty clause. □

For an XOR formula P, we define $\mathcal{D}(P)$ as the union of the direct encodings of XOR constraints in P. We formalize XOR reasoning as a proof system with two inference rules, given in Fig. 2. An XOR proof of an XOR formula Q from a formula P is a proof using only additions and XOR definitions, where all premises are in P and all XOR constraints in Q are either in P or occurring along the proof. Note that the addition rule subsumes Gaussian elimination steps [29], and therefore XOR proofs subsumes Gaussian elimination procedures.

Example 2. Consider the XOR formula $P = \{[p, q, r]^0, [p, q]^1\}$. We obtain the following XOR proofs of $[r]^1$, with and without the use of a single XOR definition:

$$\dfrac{\dfrac{\dfrac{}{[x,p,q]^0}\text{def} \quad [p,q,r]^0}{[x,r]^0}\text{add} \quad \dfrac{\dfrac{}{[x,p,q]^0}\text{def} \quad [p,q]^1}{[x]^1}\text{add}}{[r]^1}\text{add} \qquad \dfrac{[p,q,r]^0 \quad [p,q]^1}{[r]^1}\text{add}$$

□

2.3 DRAT Proofs

The DRAT (Deletion Resolution Asymmetric Tautology) format [32] is based on the notion of asymmetric literal addition [19]. Given a CNF formula F and a clause C, the set $\mathrm{AL(F, C)}$ contains all literals L such that, for literals $L_1, \ldots, L_n$ occurring in C, the clause $(L_1 \vee \ldots \vee L_n \vee \overline{L})$ belongs to F. We define the *asymmetric literal addition* function ALA_F that maps a clause C to the clause $\mathrm{ALA}_F(C) = C \vee \bigvee_{L \in \mathrm{AL(F,C)}} L$. We consider the repeated application of ALA_F:

$$\mathrm{ALA}_F(C) \uparrow 0 = C \qquad \mathrm{ALA}_F(C) \uparrow n+1 = \mathrm{ALA}_F(\mathrm{ALA}_F(C) \uparrow n)$$

A clause C is an *asymmetric tautology* (AT) w.r.t. F if, for some $n \geq 0$, $\mathrm{ALA}_F(C) \uparrow n$ is a tautology. Asymmetric tautologies can also be characterized in terms of unit propagation, i.e. $(L_1 \vee \ldots \vee L_n)$ is AT w.r.t. F if and only if

unit propagation in $F \wedge \neg L_1 \wedge \ldots \wedge \neg L_n$ detects an inconsistency [3]. A clause C is a *resolution asymmetric tautology* (RAT) [19] upon L w.r.t. F if the resolvent of C and D upon L is an AT w.r.t. F for all clauses $D \in F$ with $\overline{L} \in D$.

Example 3. Consider the formula $F = \{(p \vee q), (p \vee \neg q \vee r), (\neg q \vee \neg r)\}$ Then, the application of asymmetric literal addition for p shows that the unit clause p is an AT in F, while the unit clause q is not an AT:

$$\begin{aligned}
&\mathrm{ALA}_F(p) \uparrow 1 = (p \vee \neg q) && \mathrm{ALA}_F(q) \uparrow 1 = (q \vee \neg p)\\
&\mathrm{ALA}_F(p) \uparrow 2 = (p \vee \neg q \vee \neg r \vee r) && \mathrm{ALA}_F(q) \uparrow 2 = \mathrm{ALA}_F(q) \uparrow 1\\
&\mathrm{ALA}_F(p) \uparrow 3 = (p \vee \neg q \vee \neg r \vee r \vee q)\\
&\mathrm{ALA}_F(p) \uparrow 4 = \mathrm{ALA}_F(p) \uparrow 3
\end{aligned}$$

Moreover, the unit clause $\neg q$ is a RAT, since it can only be resolved with $(p \vee q)$, yielding p which is an AT. □

Introduction of asymmetric tautologies to a formula preserves semantic equivalence, while introduction of resolution asymmetric tautologies to a formula preserves satisfiability [19]. A *DRAT proof* in a formula F is then a sequence of clauses such that every clause is either AT or RAT with respect to the formula F together with the preceding clauses. In the following we will use the fact that resolvents of C and D are asymmetric tautologies in $\{C, D\}$ [19]. This allows to regard any resolution proof using resolution inferences of the form

$$\frac{C \vee L \qquad D \vee \overline{L}}{C \vee D}\,\mathrm{res}$$

as a DRAT proof by traversing the proof tree in a breadth-first top-down manner.

3 Variable-Elimination-Based Approach

In this section we present the *direct translation*, a method to construct DRAT proofs from XOR proofs based on the direct encoding of XOR constraints. Each inference in the XOR proof system is translated to a DRAT proof; concatenation of partial translations is the direct translation of an XOR proof into a DRAT proof. In the following, we give translations for the two inference rules in XOR reasoning, namely additions and XOR definitions. In general, the direct encoding of an addition is not a DRAT proof from the direct encoding of its premisses:

Example 4. Consider the XOR constraints $[p, q]^0$ and $[p, q]^1$. By addition we obtain $[]^1$, whose direct encoding only contains the empty clause. However, the empty clause is not a RAT in the direct encoding of the premises, given by $\mathcal{D}([p, q]^0) \cup \mathcal{D}([p, q]^1) = \{(\neg p \vee q), (p \vee \neg q), (\neg p \vee \neg q), (p \vee q)\}$ □

In fact, the problem arises when two or more variables are eliminated by addition. We propose to eliminate the variables stepwise. Consider XOR constraints X, Y and $Z = X \triangle Y$ defining a general addition inference of the form:

$$[A_1, \ldots, A_n, B_1, \ldots, B_p]^k \triangle [A_1, \ldots, A_n, B'_1, \ldots, B'_q]^l = [B_1, \ldots, B_p, B'_1, \ldots, B'_q]^{k \oplus l}$$

The proof is constructed in a bottom-up fashion: starting from each clause C in $\mathcal{D}(Z)$, a resolution proof of C from $\mathcal{D}(X) \cup \mathcal{D}(Y)$ is generated. We know that C is a clause of the form $C = (L_1 \vee \cdots \vee L_p \vee L'_1 \vee \cdots \vee L'_q)$, where literals L_i are either B_i or $\neg B_i$, and similarly for L'_i.

C can be obtained by resolving the two clauses $C \vee A_1$ and $C \vee \neg A_1$ upon A_1. These clauses contain the literals corresponding to all the B_i, B'_i as well as to A_1. In general, we can consider a clause C' of the form $C \vee K_1 \vee \cdots \vee K_j$, where the literals K_i are either A_i or $\neg A_i$. C' can be further obtained as the resolvent of $(C \vee K_1 \vee \cdots \vee K_j \vee A_{j+1})$ and $(C \vee K_1 \vee \cdots \vee K_j \vee \neg A_{j+1})$. Generating these resolution steps recursively gives a resolution proof, where clauses in level j are of the form $C \vee K_1 \vee \cdots \vee K_j$.

$$\dfrac{\dfrac{\dfrac{\ddots \quad \iddots}{C \vee A_1 \vee A_2}\text{res} \quad \dfrac{\ddots \quad \iddots}{C \vee A_1 \vee \neg A_2}\text{res}}{C \vee A_1}\text{res} \quad \dfrac{\dfrac{\ddots \quad \iddots}{C \vee \neg A_1 \vee A_2}\text{res} \quad \dfrac{\ddots \quad \iddots}{C \vee \neg A_1 \vee \neg A_2}\text{res}}{C \vee \neg A_1}\text{res}}{C}\text{res}$$

Clauses in the $(n-1)$-th level are of the form $C \vee K_1 \vee \cdots \vee K_{n-1}$. Such clauses can be guaranteed to be AT in the CNF formula $\mathcal{D}(X) \cup \mathcal{D}(Y)$. Let $\mathcal{P}(X, Y)$ be the sequence of clauses obtained from traversing the above proof tree in breadth-first, top-down manner. Then, $\mathcal{P}(X, Y)$ is a DRAT proof of the $\mathcal{D}(X \triangle Y)$ from $\mathcal{D}(X) \cup \mathcal{D}(Y)$.

On the other hand, translation of XOR definitions is straightforward. If the XOR constraint X contains a variable that does not occur in F, and $\mathcal{D}(X) = \{C_1, \ldots, C_n\}$, then $(C_1, \ldots, C_n)$ is a DRAT proof of $\mathcal{D}(X)$ from F.

The direct translation of an XOR proof is then given by the concatenation of such partial translations of the addition and XOR definition inferences along the XOR proof.

4 T-Translation of XOR Proofs

In this section we introduce *T-translations* that avoid the exponential blow-up in the proof length by expressing single XOR constraints as conjunction of several XOR constraints of fixed size. We assume from now on that variables in XOR constraints are sorted. The *natural splitting* [14] of $X = [A_1, \ldots, A_n]^k$, denoted by $\mathcal{S}(X)$, is $\{X\}$ if $|X| \leq 3$, and otherwise the set containing the following XOR constraints:

$$\overbrace{[A_1, A_2, s_0]^0\ [s_0, A_3, s_1]^0\ \ldots\ [s_{n-4}, A_{n-2}, s_{n-3}]^0}^{\text{splitting matrix}}\ \overbrace{[s_{n-3}, A_{n-1}, A_n]^k}^{\text{independent constraint}} \tag{1}$$

where the s_i are fresh variables. The set of XOR constraints in the left is called the *splitting matrix* of X, denoted by $\hat{\mathcal{S}}(X)$. The rightmost XOR constraint is called the *independent constraint* of X, denoted by $\mathcal{I}_X$; in the case when $|X| \leq 3$ we define $\hat{\mathcal{S}}(X) = \emptyset$ and $\mathcal{I}_X = X$.

Example 5. We show three XOR constraints with their respective splittings, where the x_i, z_i are fresh variables. Each independent constraint is underlined.

$$\begin{aligned} X &= [p_1, p_2, p_3, p_4, p_5]^1 & \mathcal{S}(X) &= \{[p_1, p_2, x_0]^0, [x_0, p_3, x_1]^0, \underline{[x_1, p_4, p_5]^1}\} \\ Y &= [p_4, p_5, p_6]^0 & \mathcal{S}(Y) &= \{\underline{[p_4, p_5, p_6]^0}\} \\ Z &= [p_1, p_2, p_3, p_6]^1 & \mathcal{S}(Y) &= \{[p_1, p_2, z_0]^0, \underline{[z_0, p_3, p_6]^1}\} \end{aligned}$$

□

The *linear encoding* of X, is $\mathcal{L}(X) = \mathcal{D}(\mathcal{S}(X))$, i.e. the direct encoding of the splitting. Notice that the linear encoding is equivalent w.r.t. satisfiability to the direct encoding of the XOR constraint itself, and has polynomial size. Given an XOR proof of an XOR formula Q from an XOR formula P, its *T-translation* is a DRAT proof of $\mathcal{D}(Q)$ from $\mathcal{D}(P)$ constructed as follows:

1. Obtain a *splitter* XOR proof of $\mathcal{S}(P)$ from P; its direct translation, called the *prefix proof*, is a DRAT proof of $\mathcal{L}(P)$ from $\mathcal{D}(P)$.
2. Generate an *intermediate* XOR proof of $\mathcal{S}(Q)$ from $\mathcal{S}(P)$; its direct translation, called the *lift proof*, is a DRAT proof of $\mathcal{L}(Q)$ from $\mathcal{L}(P)$.
3. Derive $\mathcal{D}(Q)$ from $\mathcal{L}(Q)$ through the *suffix proof*; the concatenation of prefix, lift and suffix is a proof of $\mathcal{D}(Q)$ from $\mathcal{D}(P)$.

4.1 Prefix Proof – Towards the Splitted Representation

In general, given the direct encoding of the splitted XOR constraint $\mathcal{L}(X) = \mathcal{D}(\mathcal{S}(X)) = \{C_1, \ldots, C_n\}$, the sequence $(C_1, \ldots, C_n)$ is not a DRAT proof from $\mathcal{D}(X)$. We therefore propose to generate a *splitter* XOR proof of $\mathcal{S}(X)$ from an XOR constraint X; applying the direct translation to this splitter yields a DRAT proof of $\mathcal{L}(X)$ from $\mathcal{D}(X)$ as follows.

Consider $X = [A_1, \ldots, A_n]^k$. Observe that $X = \triangle_{Y \in \mathcal{S}(X)} Y$. Hence, we can conclude that $\mathcal{I}_X = X \triangle (\triangle_{Y \in \hat{\mathcal{S}}(X)} Y)$. The procedure to construct the splitter of X consists on, firstly, introducing all XOR constraints in $\hat{\mathcal{S}}(X)$ as XOR definitions, which is possible as long as this is done in the order shown in (1). Secondly, the missing constraint $\mathcal{I}_X$ can be derived by, starting with X, iteratively applying addition inferences with all constraints from $\hat{\mathcal{S}}(X)$. Furthermore, provided this operation is performed in the order from (1), we are able to guarantee that this process never involves an XOR constraint larger than X, which is essential to bound the length of the obtained DRAT proof.

Example 6. Consider again the constraints in Example 5. Since the splitting of Y is $\{Y\}$, its splitter is the empty proof. The splitter of X is given by:

$$\dfrac{\dfrac{[p_1, p_2, p_3, p_4, p_5]^1 \qquad \dfrac{}{[p_1, p_2, x_0]^0}\,\text{def}}{[x_0, p_3, p_4, p_5]^1}\,\text{add} \qquad \dfrac{}{[x_0, p_3, x_1]^0}\,\text{def}}{[x_1, p_4, p_5]^1}\,\text{add}$$

□

Applying direct translation results in a DRAT proof of $\mathcal{L}(X)$ from $\mathcal{D}(X)$, which we refer to as the *prefix* proof. In the case the aforementioned orders are used, the obtained proof is polynomial in the size of the input CNF formula.

4.2 Lifted Proof

We generate now an *intermediate* XOR proof of $\mathcal{S}(Q)$ from $\mathcal{S}(P)$; its direct translation will be the *lift* DRAT proof. It suffices to give proofs of $\mathcal{S}(Z)$ from $\mathcal{S}(X) \cup \mathcal{S}(Y)$ for every addition inference $Z = X \triangle Y$; the intermediate XOR proof is the concatenation of such proofs for every addition inference along the original XOR proof. Assume that the XOR constraints X, Y, Z contain exactly the variables $A_1 < \cdots < A_n$.

Similarly to the prefix proof, XOR constraints in the matrix $\hat{\mathcal{S}}(Z)$ can be introduced in the same order as in (1) as XOR definitions. The rest of the proof is directed towards deriving the independent XOR constraint $\mathcal{I}_Z$ by addition. It is possible to show that:

$$\mathcal{I}_Z = (\triangle_{X' \in \mathcal{S}(X)} X') \triangle (\triangle_{Y' \in \mathcal{S}(Y)} Y') \triangle (\triangle_{Z' \in \hat{\mathcal{S}}(Z)} Z')$$

As before, the result holds regardless of the order on which addition inferences are applied. However, it is possible to choose an order which produces intermediate XOR constraints of size bounded by 5, which is needed to avoid an exponential blow-up. This is attained by first adding the XOR constraints containing the literal A_1, afterwards adding those containing A_2, and so on.

Example 7. Consider X, Y and $Z = X \triangle Y$ as in Example 5. Then, one can derive $\mathcal{S}(Z)$ from $\mathcal{S}(X) \cup \mathcal{S}(Y)$ as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[p_1, p_2, x_0]^0 \quad \overline{[p_1, p_2, z_0]^0}^{\text{def}}}{[x_0, z_0]^0}^{\text{add}} \quad [x_0, p_3, x_1]^0}{[x_1, z_0, p_3]^0}^{\text{add}} \quad [x_1, p_4, p_5]^1}{[z_0, p_3, p_4, p_5]^1}^{\text{add}} \quad [p_4, p_5, p_6]^0}{[z_0, p_3, p_6]^1}^{\text{add}}$$

The only XOR constraint in the matrix of Z has been introduced by XOR definition on z_0. To derive the independent constraint $[z_0, p_3, p_6]^1$, we have first used up the XOR constraints containing p_1, then the remaining ones containing p_2 (in this case, none), then the remaining ones containing p_3 and so forth. □

The *lift* proof is a DRAT proof of $\mathcal{L}(Q)$ from $\mathcal{L}(P)$ obtained by applying the direct translation to the intermediate translation described above for every addition inference along the original XOR proof.

4.3 Suffix Proof – Towards the Direct Encoding

Suffix proofs are generated by listing all clauses in the direct encoding of X, since every such clause is an AT w.r.t. $\mathcal{L}(X)$. Once the three parts of the proof have been generated, the T-translation consists of their concatenation.

Theorem 8 (Main Theorem). *Let P, Q be XOR formulae, and π be an XOR proof of Q from P. Consider the prefix π_p, lift π_l and suffix π_s obtained from P, π and Q respectively. Then, $\pi_p \, \pi_l \, \pi_s$ is a DRAT proof of $\mathcal{D}(Q)$ from $\mathcal{D}(P)$.*

Proof. See [25, Corollary 7.30]. □

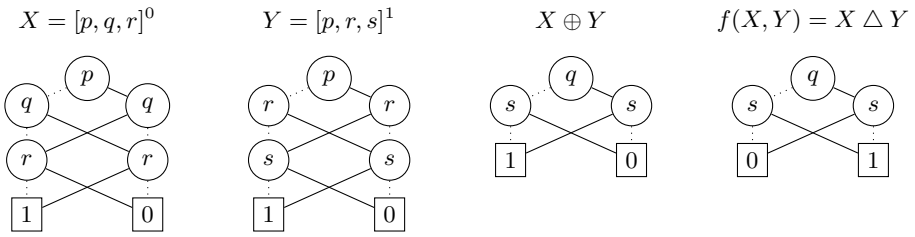

Fig. 3. BDD representation of two XOR constraints X, Y, as well as of $X \oplus Y$ and $f(X, Y)$. A dotted (solid) line from variable A indicates the BDD after assigning A to false (true, resp.). The correct BDD for the XOR constraint $X \triangle Y$ is that of $f(X, Y)$.

5 Proof generation using BDDs

An alternative to the proposed method consists in expressing XOR constraint addition as an operation over binary decision diagrams (BDDs) [8]. A DRAT proof can then be generated using a method proposed by Sinz and Biere [28].

Let us consider two XOR constraints X and Y, and assume we have computed their BDDs B_X and B_Y. The binary Boolean function f is defined by $f(x, y) = 1$ if and only if $x = y$. Then, as shown by Figure 3, the BDD of the XOR constraint $X \triangle Y$ can be computed by applying the binary Boolean function f to B_X and B_Y using a well-known algorithm to apply Boolean functions to BDDs [8].

Sinz and Biere [28] propose a proof method for BDD operations as extended resolution proofs. In particular, for the encoding of a BDD B as a CNF formula $\mathcal{E}(B)$ described in [28], a method to derive $\mathcal{E}(B_1 \wedge B_2)$ from $\mathcal{E}(B_1) \cup \mathcal{E}(B_2)$ by an extended resolution proof is proposed. Due to space constraints, we do not discuss this method in detail; it can however be adapted to our problem:

- By performing minor changes in the case where the operated BDDs are leaves, it is possible to extend the method so that BDDs are operated with the aforementioned Boolean function f instead of $\wedge$. This outlines another method to lift an XOR proof into an extended resolution proof.
- Extended resolution proofs can be easily transformed into DRAT proofs [19].
- Given a clause C and its BDD encoding B_C, clauses in $\mathcal{E}(B_C)$ can be derived by simply enumerating them. The encoding $\mathcal{E}(B_X)$ of the BDD of an XOR constraint X is derived by conjoining all the clauses in $\mathcal{D}(X)$ as BDDs, and then lifting this operation into a DRAT proof as above.
- In an analogous way to the T-translation suffix, clauses in the direct encoding of an XOR constraint X can be derived as asymmetric tautologies in $\mathcal{E}(B_X)$.

Example 9. Consider XOR constraints X and Y with direct encodings:

$$\mathcal{D}(X) = \{C_1, C_2\} \quad \mathcal{D}(Y) = \{C_3, C_4\} \quad \mathcal{D}(X \triangle Y) = \{D_1, D_2, D_3, D_4\}$$

The encodings of the BDDs corresponding to clauses C_i can be derived by enumerating the clauses in $\mathcal{E}(C_i)$. Now, since X is semantically equivalent to $C_1 \wedge C_2$

Translation	*Length*
Direct addition $Z = X \triangle Y$	$2^{u(X,Y)-1} - 2^{d(X,Y)-1}$ if $d(X,Y) > 0$, $2^{u(X,Y)-1}$ if $d(X,Y) = 0$
Direct XOR definition X	$2^{l(X)-1}$ if $l(X) > 0$, and 1 if $l(X) = 0$
Prefix proof of XOR constraint X	$4(l(X) - 4) + 3 \cdot 2^{l(X)-1}$
Lifted proof of addition $Z = X \triangle Y$	$36u(X,Y)$
Suffix proof of XOR constraint X	$5 \cdot 2^{l(X)-1}$

Table 1. Above, (exact) proof lengths of direct translations for each inference in the input XOR proof. Below, proof length bounds for each T-translation part. We use the following measures: $l(X) = |X^\star|$, $u(X,Y) = |X^\star \cup Y^\star|$ and $d(X,Y) = |(X \triangle Y)^\star|$ where $X^\star = \{A_1, \ldots, A_n\}$ for an XOR constraint $X = [A_1, \ldots, A_n]^k$.

and ROBDDs are canonical [8], we have that $B_X = B_{C_1} \wedge B_{C_2}$. Applying the method from [28] yields a DRAT proof of $\mathcal{E}(B_X)$ from $\mathcal{E}(B_{C_1}) \cup \mathcal{E}(B_{C_2})$; and analogously for Y. Our variation in this method replacing $\wedge$ by f can then provide a DRAT proof of $\mathcal{E}(B_{X \triangle Y})$ from $\mathcal{E}(B_X) \cup \mathcal{E}(B_Y)$. The direct encoding of $X \triangle Y$ can be derived from $\mathcal{E}(B_{X \triangle Y})$ by introducing every clause as an AT.

$$\left.\begin{array}{l} C_1 \rightsquigarrow \mathcal{E}(B_{C_1}) \\ C_2 \rightsquigarrow \mathcal{E}(B_{C_2}) \end{array}\right\} \wedge \; \mathcal{E}(B_{C_1 \wedge C_2}) = \mathcal{E}(B_X)$$

$$\left.\begin{array}{l} C_3 \rightsquigarrow \mathcal{E}(B_{C_2}) \\ C_4 \rightsquigarrow \mathcal{E}(B_{C_2}) \end{array}\right\} \wedge \; \mathcal{E}(B_{C_3 \wedge C_4}) = \mathcal{E}(B_Y)$$

$$\left.\begin{array}{l} \mathcal{E}(B_X) \\ \mathcal{E}(B_Y) \end{array}\right\} f \; \mathcal{E}(B_{X \triangle Y}) \rightsquigarrow \{D_1, \ldots, D_4\} = \mathcal{D}(X \triangle Y)$$

□

6 Length Analysis of the Constructed DRAT Proofs

Table 1 presents the exact length measures of direct translations. Translation of addition inferences is exponential on the u measure, which is the motivation behind T-translations: by bounding the maximum size of intermediate XOR constraints, we are able to asymptotically reduce the size of the lift translation.

In order to relate the number of variables occurring in the input formula to the size of the input formula, we assume that the XOR proof was obtained by Gaussian elimination, which is a safe assumption in all practical cases. In particular, we consider an XOR proof of an XOR formula Q from an XOR formula P of size n^2, where n is the number of variables in P.

Theorem 10. *If Gaussian elimination was used to obtain an XOR proof π of Q from P, then the length of the DRAT proof obtained from π by T-translation is bounded by $O((|\mathcal{D}(P)| + |\mathcal{D}(Q)|)^3)$.*

Proof. See [25, Theorem 8.2] for a proof for a regularized version of the T-translation. The proof can easily be adapted to the setting explained here. Observe this is a very loose bound, since the cubic exponent only applies to the involved XOR constraints in every inference. □

Note that the size of the suffix subproof can be ignored if simplified XOR constraints are introduced back in the SAT solver with the linear encoding. T-translations are then polynomial in the size of the input CNF formula. This bound does not contradict the exponential bounds for prefix and suffix proofs: the size of XOR constraints is logarithmic in the usual measure of a proof generation method, which is the size of the input CNF formula, in our case $\mathcal{D}(P)$. This bound allows us to show a complexity gap between direct and T-translations. While the T-translation has polynomial length in the size of the input formula, this is not true in general for the direct translation. The following example shows a family of XOR proofs whose direct translation is of exponential length on the size of the input CNF formula.

Example 11. Consider XOR constraints $X_k = [p_{k-1}, p_{k+1}, q_{k+1}]^0$ and $Y_k = [p_k, p_{k+1}, q_1, \ldots, q_{k+1}]^0$ for $k \geq 0$, and the XOR formula $P_k = \{Y_0, X_1, \ldots, X_k\}$. A family of XOR proofs is given by $\varphi_k = Y_1, \ldots, Y_k$, where the i-th XOR constraint is obtained by the addition $Y_i = Y_{i-1} \triangle X_i$; these correspond to records of Gaussian elimination over P_k. Note that all premises in P_k are of length 3, so the size of $\mathcal{D}(P_k)$ is $4(k+1)$. However, the i-th addition has measures $u(Y_{i-1}, X_i) = i + 4$ and $d(Y_{i-1}, X_i) = i + 3$. Thus, the direct translation of the i-th addition is of length 2^{i+2}, totaling to translation length $2^{O(k)}$ for φ_k. □

7 Experimental Evaluation

We implemented the three approaches for proof generation in the Scala programming language. Our algorithms for BDD manipulation are described in [6], and we based our implementation in the one released by J.C. Filliâtre[3]. We ran experiments over the instances of the application track of the SAT Competition 2014 and obtained XOR proofs from the preprocessor `CoProcessor` [22]. 210 out of 300 instances yielded nonempty XOR proofs, which constituted our benchmarks. The average length of these benchmarks was 36,000 XOR constraints, with some instances up to 10 times longer; benchmarks contain XOR constraints averaging 3.47 in size. For each benchmark we computed DRAT proofs using the direct translation, the T-translation, and the BDD-based approach. The experiments were run on an 2-core 3.5 GHz AMD Opteron machine with 192 GB RAM. A 5 minute timeout was set, and proofs were generated in memory but not stored in disk. Figures 4a, 4b and 4c compare these lengths.

Our results show that the BDD-based approach performs consistently worse than both direct and T-translations. In particular, it times out on 15% of the benchmarks, compared to 13% for direct translations and none for T-translations;

[3] `https://github.com/zhihan/bdd-scala`

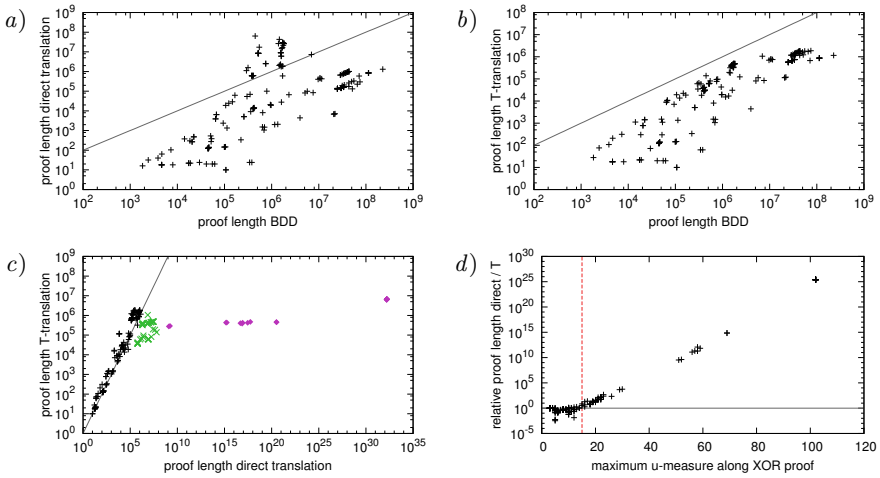

Fig. 4. Graphs (a), (b), (c) compare the three different forms of proof construction: the direct translation, the T-translation, and the BDD-based approach. The gray line indicates equal length. In Graph (c), green points correspond to the instances where BDD-based generation yields shorter proofs than direct translation; black points are the instances where the converse holds; and purple points are those where direct translation times out, where length was computed using Table 1. Graph (d) compares relative length of direct translations w.r.t. T-translations to the maximum value of the u-measure along the input XOR proof. Gray line indicates equal length of direct and T-translations; the proposed threshold value of 15 is indicated by a red dashed line.

all terminating instances for the BDD-based approach terminate for direct and T-translation, as shown by Figures 4a and 4b.

Comparison between direct and T-translation is more complex, partly due to timeouts for direct translations. Given the sheer size of some direct translations, they would have been impossible to generate within any reasonable time. A length comparison is nevertheless possible, since direct translation length can be predicted by using the results from Table 1. Figure 4c shows these results, where predicted data is provided where the direct translation times out. Direct translations are strictly shorter than T-translations on 46% of the instances. Moreover, whereas in some cases direct translation yields up to 300 times shorter proofs, in some other instances direct translation produces proofs 25 orders of magnitude larger than T-translation, which is consistent with our theoretical analysis in Section 6. Furthermore, Figure 4c shows that, whenever T-translation is outperformed by direct translation, then so is the BDD-based approach. This suggests that the latter should not be considered for proof generation.

Further data are presented in Figure 4d, showing a tight relation between the maximum u measure (defined in Table 1) along the input XOR proof and a length comparison of direct and T-translations. In particular, we find that T-translation outperforms direct translation in proof size whenever the former measure is

Size of XOR constraint	3	5	7	9	11	13
Minimum length of BDD prefix	209	1704	15436	151855	1633456	19384645
Maximum length of BDD prefix	216	1879	16507	156914	1668015	19523431
Length of T-translation prefix	0	44	196	780	3092	12316

Table 2. Sizes of generated prefixes with the BDD-based method and with T-translation. BDD prefixes were generated by conjoining BDDs in random orders, for a sample of size 30; minimum and maximum recorded prefix lengths are shown.

larger than 15. The obtained data suggest an approach for proof generation by computing the maximum u measure in the input, and then comparing it to a threshold value of 15 to decide for the direct encoding or the linear encoding.

The particular order on which clauses in $\mathcal{D}(X)$ are conjoined to construct the BDD of the XOR constraint X does not have a significant influence on the length of the prefix, as shown in Table 2. In particular, data suggests an average difference of around 4% between the minimum and the maximum BDD-based prefix length, and in all cases a worse behaviour than the T-translation prefix.

8 Conclusion

Contemporary SAT solvers employ XOR reasoning techniques to efficiently solve the propositional satisfiability problem. It was an open problem [16, 28] to efficiently express XOR reasoning in terms of the DRAT format. We have adapted a known BDD-based approach [28] to generate such proofs, although this method is resource-intensive and generated proofs are very long. We propose two alternatives: direct translation transforms every XOR step into a DRAT proof, whereas T-translation avoids the exponential blow-up of the direct translation by first generating a new XOR proof using Tseitin variables, and afterwards applying the direct translation. For XOR proofs produced by Gaussian elimination, T-translations are polynomial in the size of the input CNF formula. Experiments have shown that direct and T-translations outperform the BDD-based approach. The direct encoding sometimes generates proofs of enormous size; however, it is possible to predict instances where this happens, so that T-translation is applied instead. Our approach allows efficient XOR reasoning when certificates of correctness are needed, producing unsatisfiability proofs of adequate size.

In the future, we plan to implement both translations in `CoProcessor` and apply similar ideas to obtain proofs for cardinality resolution [26], as suggested in [16]. Another interesting problem is adapting the presented approaches to SAT solvers where XOR reasoning takes place within the CDCL procedure[20, 21].

Acknowledgements We would like to thank an anonymous reviewer who pointed out that the BDD-based approach could be used as a baseline.

Bibliography

[1] Aloul, F.A., Ramani, A., Markov, I.L., Sakallah, K.A.: Solving difficult SAT instances in the presence of symmetry. In: DAC 2002. pp. 731–736. ACM (2002)

[2] Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Boutilier, C. (ed.) IJCAI 2009. pp. 399–404. Morgan Kaufmann Publishers Inc., Pasadena (2009)

[3] Beame, P., Kautz, H., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. Journal of Artificial Intelligene Research 22(1), 319–351 (2004)

[4] Belov, A., Diepold, D., Heule, M.J., Järvisalo, M. (eds.): Proceedings of SAT Competition 2014, Department of Computer Science Series of Publications B, vol. B-2014-2. University of Helsinki, Helsinki, Finland (2014)

[5] Biere, A.: Yet another local search solver and lingeling and friends entering the SAT competition 2014. In: Belov et al. [4], pp. 39–40

[6] Brace, K.S., Rudell, R.L., Bryant, R.E.: Efficient implementation of a BDD package. In: DAC. pp. 40–45 (1990)

[7] Brummayer, R., Biere, A.: Fuzzing and delta-debugging SMT solvers. In: Workshop SMT 2010. pp. 1–5. ACM (2009)

[8] Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers 35(8), 677–691 (1986)

[9] Courtois, N., Bard, G.V.: Algebraic cryptanalysis of the data encryption standard. In: Galbraith, S.D. (ed.) IMA 2007. LNCS, vol. 4887, pp. 152–169. Springer (2007)

[10] Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM 5(7), 394–397 (1962)

[11] Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 61–75. Springer, Heidelberg (2005)

[12] Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)

[13] Eén, N., Sörensson, N.: Translating pseudo-Boolean constraints into SAT. Journal on Satisfiability, Boolean Modeling and Computation 2, 1–26 (2006)

[14] Gwynne, M., Kullmann, O.: On SAT representations of XOR constraints. In: Dediu, A.H., Martín-Vide, C., Sierra-Rodríguez, J.L., Truthe, B. (eds.) LATA 2014. LNCS, vol. 8370, pp. 409–420. Springer (2014)

[15] Heule, M., Hunt, Jr, W.A., Wetzler, N.: Expressing symmetry breaking in DRAT proofs. In: CADE 2015. pp. 591–606 (2015)

[16] Heule, M.J.H., Biere, A.: Proofs for satisfiability problems. In: All about Proofs, Proofs for all (2015)

[17] Heule, M.J.H., Kullmann, O., Marek, V.W.: Solving and verifying the boolean Pythagorean Triples Problem via cube-and-conquer. CoRR abs/1605.00723 (2016)
[18] Heule, M.: march. Towards a lookahead SAT solver for general purposes. Master's thesis
[19] Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 355–370. Springer, Heidelberg (2012)
[20] Laitinen, T.: Extending SAT Solver with Parity Reasoning. Ph.D. thesis
[21] Laitinen, T., Junttila, T.A., Niemelä, I.: Classifying and propagating parity constraints. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 357–372. Springer (2012)
[22] Manthey, N.: Coprocessor 2.0 – a flexible CNF simplifier. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 436–441. Springer, Heidelberg (2012)
[23] Manthey, N.: Riss 4.27. In: Belov et al. [4], pp. 65–67
[24] Manthey, N., Lindauer, M.: SpyBug: Automated bug detection in the configuration space of SAT solvers. In: SAT 2016. pp. 554–561 (2016)
[25] Rebola-Pardo, A.: Unsatisfiability Proofs in SAT Solving with Parity Reasoning. Master thesis, Technische Universität Dresden, Informatik Fakultät (2015)
[26] Roussel, O., Manquinho, V.M.: Pseudo-Boolean and cardinality constraints. In: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 695–733. IOS Press (2009)
[27] Silva, J.P.M., Sakallah, K.A.: GRASP - a new search algorithm for satisfiability. In: ICCAD 1996. pp. 220–227. IEEE Computer Society, Washington (1996)
[28] Sinz, C., Biere, A.: Extended resolution proofs for conjoining BDDs. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) CSR 2006. LNCS, vol. 3967, pp. 600–611. Springer (2006)
[29] Soos, M.: Enhanced Gaussian elimination in DPLL-based SAT solvers. In: POS 2010 (2010)
[30] Soos, M.: Cryptominisat v4. In: Belov et al. [4], pp. 23–34
[31] Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Kullmann, O. (ed.) SAT 2009. pp. 244–257. Springer, Heidelberg (2009)
[32] Wetzler, N., Heule, M.J., Hunt, Jr, W.A.: DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 422–429. Springer (2014)