Sorting Parity Encodings by Reusing Variables

Leroy Chew and Marijn J.H. Heule

Computer Science Department, Carnegie Mellon University, PA, USA

Abstract. Parity reasoning is challenging for CDCL solvers. A simple instance of two contradictory parity constraints on a modest number of variables can be difficult to refute when the variables are simply in a different order. Existing methods to solve these formulas detect the parity constraints from the formula and apply Gaussian elimination or construct a binary decision diagram. Existing implementations either lack support of proof logging or use many extension variables.
The direct encoding of the parity function requires exponentially many clauses. To get around this, Tseitin variables are introduced to realize a compact encoding. We present a technique for short clausal proofs that uses these Tseitin variables to swap and sort the encodings within the DRAT system. We show how to construct proofs of size $\mathcal{O}(n \log n)$ for two randomized and contradicting parity constraints by reusing variables.

1 Introduction

Modern SAT solving technology is based on Conflict Driven Clause Learning (CDCL) [13]. The resolution proof system [17] has a one-to-one correspondence [16] with CDCL solving, however in practice, the techniques used in modern solvers go beyond what can be succinctly represented in a resolution proof. Lower bounds [22] show that often we cannot always hope to match the calculations a solver or preprocessor performs with a matching linear or even polynomial size resolution proof. This difficulty means that when we need to present verifiable certificates of unsatisfiable instances, resolution is not always sufficient.

Extended Resolution (ER) [21] is a strong propositional proof system that can polynomially simulate CDCL and many other techniques. However, ER is not necessarily the most useful system in practice, as we also want to minimise the degree of the polynomial simulation.

The DRAT proof system [7] is polynomially equivalent to ER [9]. Yet most practitioners favour DRAT due to its ability to straightforwardly simulate known preprocessing and inprocessing techniques. DRAT works by allowing inference to go beyond preserving models and instead preserves only satisfiability.

In this paper, we demonstrate DRAT's strengths on a particular kind of unsatisfiable instances that involve parity constraints. Formulas with parity constraints have been benchmarks for SAT for decades. The `Dubois` family encodes the parity function on two sequences of literals in the same order, but with different Tseitin variables. Additionally, in one instance a literal is flipped to give a contradiction. `Urquhart` formulas [22] encode a modulo two sum of the degree of

each vertex of a graph, the unsatisfiability comes from an assertion that this sum is odd, a violation of the Handshake Lemma. The `Parity` family from Crawford and Kearns [3] takes multiple parity instances on a set of variables and combines them together. For these problems, practical solutions have been studied using Gaussian elimination, equivalence reasoning, binary decision diagrams and other approaches [23,12,14,6,20,19,10,11,5].

Extracting checkable proofs in a universal format have been another matter entirely. While it is believed that polynomial size circuitry exists to solve these problems, actually turning them into proofs could mean they may only be "short" in a theoretical polynomial-size sense rather than a practical one. Constructing a DRAT proof of parity reasoning has been investigated theoretically [15], but no implementation exists to actually produce them nor is it clear whether the size is still reasonable to be useful in practice.

There has been some investigation into looking at DRAT without the use of extension variables which is of intermediate power between resolution and ER. The power of DRAT without extension variables, known as DRAT$^-$, is somewhere in between resolution and ER. The power of DRAT does not rely on its simulation of ER as shown by several simulation results exist for DRAT$^-$ [2]. A key simulation technique was the elimination and reuse of a variable, which we use to find short DRAT$^-$ proofs of a hard parity formula.

The structure of parity constraints can be manipulated by reusing variables and we exploit the associativity and commutativity of the parity function. We demonstrate this on formulas similar to the `Dubois` family except the variables now appear in a random order in one parity constraint. We show how to obtain DRAT proofs of size $\mathcal{O}(n \log n)$ without using additional variables. Our method can also be used to produce ER proofs of similar size with new variables.

2 Preliminaries

In propositional logic a literal is a variable x or its negation $\overline{x}$, a clause is a disjunction of literals and a *Conjunctive Normal Form* (CNF) is conjunction of clauses. A unit clause is a clause containing a single literal. We denote the negation of literal l as $\bar{l}$ (or $\neg l$). The variable corresponding to literal l is $\text{var}(l)$. If C is a clause, then $\overline{C}$ is the conjunction of the negation of the literals in C each a unit clause. In this paper, we treat clauses/formulas as unordered and not containing more than one copy of each literal/clause respectively.

Unit propagation simplifies a conjunctive normal form F by building a partial assignment and applying it to F. It builds the assignment by satisfying any literal that appears in a unit clause. Doing so may negate opposite literals in other clauses and result in them effectively being removed from that clause. In this way, unit propagation can create more unit clauses and can keep on propagating until no more unit clauses remain or the empty clause is reached. We denote that the empty clause can be derived by unit propagation applied to CNF F by $F \vdash_1 \bot$. Since unit propagation is an incomplete but sound form of logical inference this is a sufficient condition to show that F is a logical contradiction.

2.1 The DRAT proof system

In this section we define the rules of the DRAT proof system. Each rule modifies a formula by either adding or removing a clause while preserving satisfiability or unsatisfiability, respectively.

Definition 1 (Asymmetric Tautology (AT)[7]). *Let F be a CNF formula. A clause C is an asymmetric tautology w.r.t. F if and only if $F \wedge \overline{C} \vdash_1 \bot$.*

Asymmetric tautologies are also known as RUP (reverse unit propagation) clauses. The rules ATA and ATE allow us to add and eliminate AT clauses. ATA steps can simulate resolution steps and weakening steps.

$$\frac{F}{F \wedge C}\ (\text{ATA: } C \text{ is AT w.r.t. } F) \qquad \frac{F \wedge C}{F}\ (\text{ATE: } C \text{ is AT w.r.t. } F)$$

Definition 2 (Resolution Asymmetric Tautology (RAT)[7]). *Let F be a CNF formula. A clause C is a resolution asymmetry tautology w.r.t. F if and only if there exists a literal $l \in C$ such that for every clause $\overline{l} \vee D \in F$ it holds that $F \wedge \overline{D} \wedge \overline{C} \vdash_1 \bot$.*

The rules RATA and RATE allow us to add and eliminate RAT clauses. RATA can be used to add new variables that neither occur in F or anywhere else. This can be used to simulate extension steps in ER.

$$\frac{F}{F \wedge C}\ (\text{RATA: } C \text{ is RAT w.r.t. } F) \quad \frac{F \wedge C}{F}\ (\text{RATE: } C \text{ is RAT w.r.t. } F)$$

3 A parity contradiction based on random orderings

In this section we will detail the main family of formulas investigated in this work. These formulas will be contradictions expressing both the parity and non-parity on a set of variables.

We define the parity of propositional literals a, b, c as follows

$$\text{xor}(a, b, c) := (\overline{a} \vee \overline{b} \vee \overline{c}) \wedge (\overline{a} \vee b \vee c) \wedge (a \vee \overline{b} \vee c) \wedge (a \vee b \vee \overline{c})$$

Let $X = \{x_1, \ldots, x_n\}$, and let σ be a bijection between literals on X, that preserves negation ($\sigma(\neg l) = \neg\sigma(l)$). Let e denote the identity permutation on the literals of X. Let $T = \{t_1, \ldots, t_{n-3}\}$. We define $\textsc{Parity}(X, T, \sigma)$ as

$$\text{xor}(\sigma(x_1), \sigma(x_2), t_1) \wedge \bigwedge_{j=1}^{n-4} \text{xor}(t_j, \sigma(x_{j+2}), t_{j+1}) \wedge \text{xor}(\overline{t}_{n-3}, \sigma(x_{n-1}), \sigma(x_n))$$

This formula is satisfiable if and only if the total parity of $\{\sigma(x_i) \mid x_i \in X\}$ is 1. The T variables act as Tseitin variables and whenever the formula is satisfied, t_{i+1} is always the sum modulo two of $\sigma(x_1), \ldots, \sigma(x_{i+2})$. The final clauses of $\text{xor}(\overline{t}_{n-3}, \sigma(x_{n-1}), \sigma(x_{n-1}))$ thus are satisfied when the sum of $t_{n-3}, \sigma(x_{n-1})$ and $\sigma(x_n)$ is 1 mod 2.

ATA	RATE	RATA	ATE
$\overline{q} \vee a \vee b \vee c$	d $\overline{p} \vee a \vee b$	$\overline{p} \vee c \vee b$	d $\overline{q} \vee a \vee b \vee c$
$\overline{q} \vee \overline{a} \vee \overline{b} \vee c$	d $\overline{p} \vee \overline{a} \vee \overline{b}$	$\overline{p} \vee \overline{c} \vee \overline{b}$	d $\overline{q} \vee \overline{a} \vee \overline{b} \vee c$
$\overline{q} \vee a \vee \overline{b} \vee \overline{c}$	d $p \vee a \vee \overline{b}$	$p \vee c \vee \overline{b}$	d $\overline{q} \vee a \vee \overline{b} \vee \overline{c}$
$\overline{q} \vee \overline{a} \vee b \vee \overline{c}$	d $p \vee \overline{a} \vee b$	$p \vee \overline{c} \vee b$	d $\overline{q} \vee \overline{a} \vee b \vee \overline{c}$
$q \vee \overline{a} \vee b \vee c$	d $\overline{p} \vee q \vee c$	$\overline{p} \vee q \vee a$	d $q \vee \overline{a} \vee b \vee c$
$q \vee a \vee \overline{b} \vee c$	d $\overline{p} \vee \overline{q} \vee \overline{c}$	$\overline{p} \vee \overline{q} \vee \overline{a}$	d $q \vee a \vee \overline{b} \vee c$
$q \vee a \vee b \vee \overline{c}$	d $p \vee q \vee \overline{c}$	$p \vee q \vee \overline{a}$	d $q \vee a \vee b \vee \overline{c}$
$q \vee \overline{a} \vee \overline{b} \vee \overline{c}$	d $p \vee \overline{q} \vee c$	$p \vee \overline{q} \vee a$	d $q \vee \overline{a} \vee \overline{b} \vee \overline{c}$

Fig. 1. DRAT steps required for Lemma 1, `d` denotes a deletion step.

Suppose we pick σ so that there is some $i \in [n]$ such that $\sigma(x_j)$ is a negative literal if and only if $j = i$. Let $T' = \{t'_1, \ldots, t'_{n-3}\}$ be another set of Tseitin variables. Now the 3-CNF $\textsc{Parity}(X, T, \sigma) \wedge \textsc{Parity}(X, T', e)$ is false as it states the parity of X is true but also states it false. However the permutation σ obfuscates the similarities between the two Parity parts of the formula.

Were σ equal to e then these formulas would be equivalent to the `Dubois` formulas and a linear proof could be made by inductively deriving clauses that express $t'_j = t_j$ for $j < i - 1$ and then $t'_j \neq t_j$ for $j \geq i - 1$. This will always allow us to derive a contradiction.

When $\sigma \neq e$ we still have a contradiction due to the commutativity of the parity function. However such a straightforward DRAT proof becomes obstructed by the disarranged ordering. This permutation also makes these formulas hard for CDCL solvers (see Section 4). Our approach is to perform a sort on the X variables of within the clauses of $\textsc{Parity}(X, T', e)$. First of all we need to show that elementary swaps are feasible in DRAT.

3.1 Short proofs

Lemma 1. *Suppose we have a CNF F and two sets of xor clauses* $\text{xor}(a, b, p)$ *and* $\text{xor}(p, c, q)$*, where variable p appears nowhere in F. We can infer*

$$\frac{F \wedge \text{xor}(a, b, p) \wedge \text{xor}(p, c, q)}{F \wedge \text{xor}(b, c, p) \wedge \text{xor}(p, a, q)}$$

in a constant number of DRAT steps without adding new variables.

Proof. The idea is to eliminate variable p so that we define q directly as the parity of a, b, c using eight "ternary xor" clauses. Each of these clauses can be added directly via ATA. We can now remove (using RATE) all clauses that contain variable p. These steps are equivalent to performing Davis-Putnam (DP) resolution [4] on variable p.

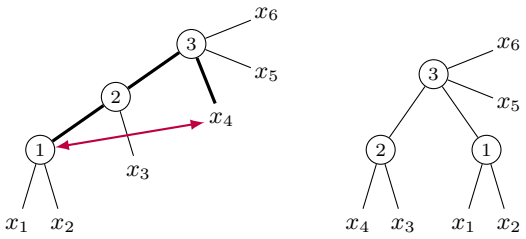

Fig. 2. Swapping the position of an internal node to balance the tree.

What we are left with is that two levels of parity have been replaced with one level of ternary parity. We can reverse the above steps to get us two levels of parity yet again, but we can swap a and c (since they appear symmetrically in our "ternary xor" clauses). We re-use the eliminated p to now mean the xor of b and c using RATA. Finally, we remove the "ternary xor" clauses using ATE. □

Note that here elimination is required only because we want to re-use the variable p. We can also show a similar step in ER without the elimination steps, introducing the "ternary xor" clauses immediately with resolution. We can introduce the four $\text{xor}(p', b, c)$ extension clauses for p', and by resolving them with the ternary clauses on b we get eight intermediate clauses which can resolve with each other on c to get the remaining four $\text{xor}(p', a, q)$ clauses. This process involves 50% more addition steps, but since it contains no deletion steps we have 25% fewer steps in total. This may be useful in different settings involving formulas that include parity where p is shared among other clauses outside of the parity and the global constraints of RAT prevent p being re-used.

Sorting the input literals. We can switch the two parity inputs using Lemma 1 in a constant number of proof steps. Furthermore the technique in DRAT does not require any additional extension variables and since the number of addition and deletion steps in Lemma 1 is the same, the working CNF does not change in size. Sorting using adjacent variables requires $\Theta(n^2)$ swaps.

Let us ignore the variables x_{n-1} and x_n and the clauses that include them as special cases. We can take $\text{xor}(x_1, x_2, t_1) \wedge \bigwedge_{i=1}^{n-4} \text{xor}(t_i, x_{i+2}, t_{i+1})$ as the definition of t_{n-3} in circuit form, using the X variables as input gates and the t_i variables as xor ($\oplus$) gates. This circuit is a tree with linear depth. the distance between two input nodes is linear in the worst-case, which is why we get $\Omega(n^2)$ many swaps. However Lemma 1 allows us even more flexibility, we can not only rearrange the input variables but the Tseitin variables.

For example if we have $\text{xor}(t_i, x_{i+2}, t_{i+1})$ and $\text{xor}(t_{i-1}, x_{i+1}, t_i)$ clauses we can eliminate t_i so that t_{i+1} is defined as the parity of x_{i+1}, x_{i+2}, and t_{i-1}. However we can now redefine t_i as the xor of x_{i+1}, x_{i+2} (using $\text{xor}(t_i, x_{i+1}, x_{i+2})$) and t_{i+1} as the xor of t_i and t_{i-1} (using $\text{xor}(t_{i+1}, t_i, t_{i-1})$). See Figure 2 for an example and notice how we change the topology of the tree.

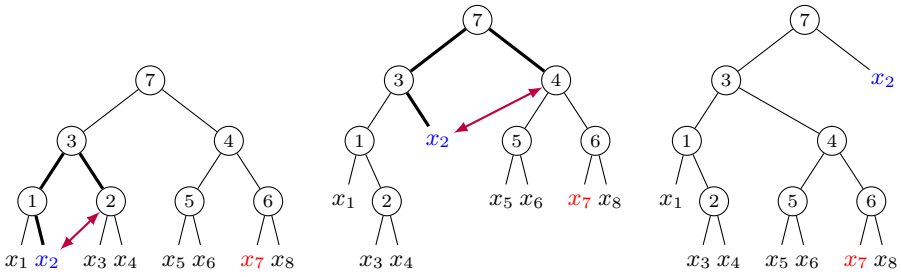

Fig. 3. Moving x_2 up to the source of the tree

In $\lfloor \frac{n}{2} \rfloor$ many swaps we can change our linear depth tree into a tree that consists of a two linear branches of depth at most $\lceil \frac{n}{2} \rceil$ joined at the top by an xor. This means that using a divide and conquer approach, we can turn this tree in a balanced binary tree of $\lceil \log_2 n \rceil$ depth in $\mathcal{O}(n \log n)$ many steps.

The purpose of a log depth tree structure is to allow leaf-to-leaf swapping from both ends of the the tree without having to do a linear number of swaps, in fact we can do arbitrary leaf swaps in $\mathcal{O}(\lceil \log n \rceil)$ many individual steps. This is done by pushing a variable up its branch to the source node of the tree and pushing it back down another branch to its destination as in Figures 3 and 4. Then we can reverse the steps with the variable being swapped out. The resulting tree even retains the position of all other nodes.

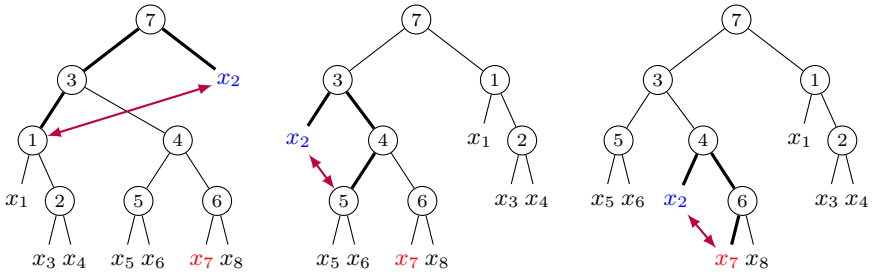

Fig. 4. Moving x_2 down to swap with x_7

Note that we also have the variables x_n and x_{n-1} that only appear in the clauses of $\text{xor}(\bar{t}_{n-3}, x_{n-1}, x_n)$. Suppose the two children of t_{n-3} in its definition circuit are a and b, in other words $\text{xor}(t_{n-3}, a, b)$ are the clauses currently defining t_{n-3}. Without loss of generality suppose we want to swap x_{n-1} with a.

The clauses of $\text{xor}(\bar{t}_{n-3}, x_{n-1}, x_n)$ are exactly the same as the clauses of $\text{xor}(t_{n-3}, x_{n-1}, \overline{x}_n)$. Using Lemma 1 we can eliminate t_{n-3} and gain eight clauses that represent that $\overline{x}_n$ is the ternary xor of a, b and x_{n-1}. Then we can reverse the steps but instead swap the positions of x_{n-1} and a.

In this way we can introduce x_{n-1} or x_n into the tree and swap it with any leaf. Once again we only require $\mathcal{O}(\log n)$ many applications of Lemma 1 to completely swap the position of x_{n-1} or x_n with any leaf.

Arriving at the empty clause. The total number of leaf-to-leaf swaps we are required to perform is bounded above linearly so we stay within $\mathcal{O}(n \log n)$ many steps. We can now undo the balanced tree into a linear tree in (we reverse what we did to balance it) keeping within an $\mathcal{O}(n \log n)$ upper bound.

Recall that we performed a sort on the variables in $\textsc{Parity}(X, T', e)$ thereby transforming it into $\textsc{Parity}(X, T', \sigma')$ with $\text{var}(\sigma'(x)) = \text{var}(\sigma(x))$, resulting in the formula $\textsc{Parity}(X, T, \sigma) \wedge \textsc{Parity}(X, T', \sigma')$. Thus the final part of the proof now involves refuting a formula equivalent to one of the `Dubois` formulas.

We create a proof that inductively shows equivalence or non-equivalence between variables $t_j \in T$ and the $t'_j \in T'$ starting from $j = 1$ to $j = n - 3$. If there is an even number of instances i, $1 \leq i \leq j + 1$ where $\sigma'(x_i) \neq \sigma(x_i)$ we derive $(t'_j \vee \bar{t}_j)$ and $(\overline{t'}_j \vee t_j)$. If there are an odd number of instances i, $1 \leq i \leq j + 1$ where $\sigma'(x_i) \neq \sigma(x_i)$ we instead derive $(\overline{t'}_j \vee \bar{t}_j)$ and $(t'_j \vee t_j)$.

Whichever case, we can increase j with the addition (ATA) of six clauses. We can think of this as working via DP resolution in a careful order: $\sigma(x_{j+1})$, t_{j-1}, t'_j from $j = 1$ to $n - 3$ in increasing j (and treat $\sigma(x_1)$ as t_0).

Finally, when $j = n - 3$, we have either already exceeded the single value i such that $\sigma'(x_i) \neq \sigma(x_i)$, or it appears in $n - 1$ or n. Either way, we can add the four clauses $(\sigma(x_{n-1}) \vee \sigma(x_n))$,$(\neg\sigma(x_{n-1}) \vee \sigma(x_n))$, $(\sigma(x_{n-1}) \vee \neg\sigma(x_n))$, $(\neg\sigma(x_{n-1}) \vee \neg\sigma(x_n))$ then the two unit clauses $(\sigma(x_n))$ and $(\neg\sigma(x_n))$ and finally the empty clause. This final part of the refutation uses $\mathcal{O}(n)$ many ATA steps.

4 Experiments

The formulas we ran experiments on are labelled $\texttt{rpar}(n, g)$. Which represent $\textsc{Parity}(X, T, \sigma_{(n,g)}) \wedge \textsc{Parity}(X, T', e)$ using the DIMACS format. The parameter n is the number of input variables and a random number generator g. The CNF uses variables $X = \{1, \ldots, n\}$, $T = \{n + 1, \ldots, 2n - 3\}$, $T' = \{2n - 2, \ldots, 3n - 6\}$, e is the identity permutation, and $\sigma_{(n,g)}$ is a random permutation based on g, where one random literal $i_{n,g}$ is flipped by σ.

We ran a program rParSort that generated an instance $\texttt{rpar}(n, \mathsf{rnd}_s)$ based on a seed s and also generated a DRAT proof based on the work in Section 3.1. We compare the size of our proofs by ones produced by the state-of-the-art SAT solver CaDiCaL [1] (version 1.2.1) and the tool EBDDRES [18] (version 1.1). The latter solves the instance using binary decision diagrams and turns the construction into an ER proof. These ER proofs can easily be transformed into the DRAT format as DRAT generalizes ER. Proof sizes (in the number of DRAT steps, i.e. lines in the proof) are presented and compared in Figure 5.

rParSort proofs remained feasible for values as large as $n = 4000$ with proofs only being 150MB due to the $\mathcal{O}(n \log n)$ upper bound in proof lines. We believe

n	vars	clauses	lines	size(KB)
10	24	64	1 681	25
20	54	144	7 469	115
50	144	384	30 657	481
101	297	792	77 971	1 426
250	744	1 984	253 777	4 810
500	1 494	3 984	583 885	11 176
1 000	2 994	7 984	1 344 837	29 278
2 000	5 994	15 984	3 023 541	67 405
3 000	8 994	23 984	4 778 373	107 276
4 000	11 994	31 984	6 668 629	150 181

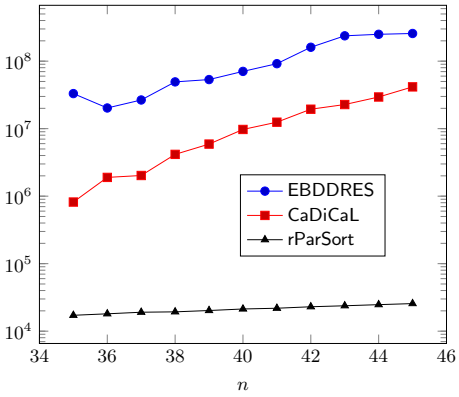

Fig. 5. rParSort proof sizes for $\texttt{rpar}(n, \mathsf{rnd}_{53})$ formulas (left). Comparisons of average (of 10) proof sizes on $n \in \{35, \ldots, 45\}$ (right).

leading coefficient is also kept small by number of factors such as the proof lines being width 4 and only 16 being needed per swap step.

CaDiCaL showed difficulty for modest values of n. While proofs with less than 10^6 lines are common for $n = 35$, the size and running time grows exponentially and by $n = 41$ proofs are larger than 10^7 lines. CaDiCaL times out using a 5000 seconds limit on some instances with $n = 46$ and on most instances with $n \geq 50$.

The size of proofs produced by EBDDRES appears to grow slower compared to CDCL, which is not surprising as BDDs can solve the formulas in polynomial time. However, as can be observed in Figure 5, the ER proofs are actually bigger for small n. The extracted DRAT proofs (converted from the ER proofs) are large: the average proof with $n \geq 35$ had more than 10^7 lines. This means that this BDD-based approach is not practical to express parity reasoning in DRAT.

Conclusion

We have shown that through manipulating existing encoding variables DRAT can take advantage of the commutativity of xor definitions via Lemma 1. Our proof generator is capable of producing reasonable-sized proofs for instances with tens of thousands of variables, while state-of-the-art SAT solvers without xor detection and Gaussian elimination, such as CaDiCaL, can only solve instances up to about 60 variables. Although these formulas are also doable for BDD-based approaches, the resulting proofs are too big for practical purposes.

The DRAT proofs are in the fragment of DRAT$^-$, where the number of variables stays fixed, which is of potential benefit to the checker. If we are not concerned with the introduction of new variables, our DRAT proofs can easily be made into ER proofs with only a 50% increase in addition steps (and the introduction of new variables). This is an alternative approach that may prove useful in other settings where elimination of a variable is not so easy.

References

1. Biere, A.: Cadical at the sat race 2019 (2019)
2. Buss, S., Thapen, N.: Drat proofs, propagation redundancy, and extended resolution. In: International Conference on Theory and Applications of Satisfiability Testing. pp. 71–89. Springer (2019)
3. Crawford, J.M., Kearns, M.J., Schapire, R.E.: The minimal disagreement parity problem as a hard satisfiability problem (1994)
4. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM **7**, 210–215 (1960)
5. Han, C.S., Jiang, J.H.R.: When boolean satisfiability meets gaussian elimination in a simplex way. In: Madhusudan, P., Seshia, S.A. (eds.) Computer Aided Verification. pp. 410–426. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
6. Heule, M., van Maaren, H.: Aligning cnf- and equivalence-reasoning. In: Hoos, H.H., Mitchell, D.G. (eds.) Theory and Applications of Satisfiability Testing. pp. 145–156. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
7. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Gramlich, B., Miller, D., Sattler, U. (eds.) Automated Reasoning. pp. 355–370. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
8. Jeřábek, E.: Dual weak pigeonhole principle, Boolean complexity, and derandomization. Annals of Pure and Applied Logic **129**, 1–37 (2004)
9. Kiesl, B., Rebola-Pardo, A., Heule, M.J.H.: Extended resolution simulates DRAT. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10900, pp. 516–531. Springer (2018)
10. Laitinen, T., Junttila, T., Niemelä, I.: Extending clause learning dpll with parity reasoning. In: Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence. p. 21–26. IOS Press, NLD (2010)
11. Laitinen, T., Junttila, T., Niemela, I.: Equivalence class based parity reasoning with dpll(xor). In: Proceedings of the 2011 IEEE 23rd International Conference on Tools with Artificial Intelligence. p. 649–658. ICTAI '11, IEEE Computer Society, USA (2011)
12. Li, C.M.: Equivalent literal propagation in the dll procedure. Discrete Applied Mathematics **130**(2), 251 – 276 (2003), the Renesse Issue on Satisfiability
13. Marques Silva, J.P., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Handbook of Satisfiability. IOS Press (2009)
14. Ostrowski, R., Grégoire, É., Mazure, B., Saïs, L.: Recovering and exploiting structural knowledge from cnf formulas. In: Van Hentenryck, P. (ed.) Principles and Practice of Constraint Programming - CP 2002. pp. 185–199. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
15. Philipp, T., Rebola-Pardo, A.: Drat proofs for xor reasoning. In: Michael, L., Kakas, A. (eds.) Logics in Artificial Intelligence. pp. 415–429. Springer International Publishing, Cham (2016)
16. Pipatsrisawat, K., Darwiche, A.: On the power of clause-learning sat solvers as resolution engines. Artificial Intelligence **175**(2), 512 – 525 (2011)
17. Robinson, J.A.: Theorem-proving on the computer. Journal of the ACM **10**(2), 163–174 (1963)
18. Sinz, C., Biere, A.: Extended resolution proofs for conjoining bdds. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) Computer Science – Theory and Applications. pp. 600–611. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)

19. Soos, M.: Enhanced gaussian elimination in dpll-based sat solvers. In: Berre, D.L. (ed.) POS-10. Pragmatics of SAT. EPiC Series in Computing, vol. 8, pp. 2–14. EasyChair (2012)
20. Soos, M., Nohl, K., Castelluccia, C.: Extending sat solvers to cryptographic problems. In: Kullmann, O. (ed.) Theory and Applications of Satisfiability Testing - SAT 2009. pp. 244–257. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
21. Tseitin, G.C.: On the complexity of derivations in propositional calculus. In: Slisenko, A.O. (ed.) Studies in Mathematics and Mathematical Logic, Part II, pp. 115–125 (1968)
22. Urquhart, A.: Hard examples for resolution. Journal of the ACM **34**(1), 209–219 (1987)
23. Warners, J.P., van Maaren, H.: A two-phase algorithm for solving a class of hard satisfiability problemsfn1fn1supported by the dutch organization for scientific research (nwo) under grant sion 612-33-001. Operations Research Letters **23**(3), 81 – 88 (1998)