

Extended Resolution in Modern SAT Solving

Norbert Manthey

Knowledge Representation and Reasoning Group
Technische Universität Dresden
`norbert.manthey@tu-dresden.de`

Abstract: Modern SAT solvers are applied in many academic and industrial fields. The CDCL algorithm, which is employed by most SAT solvers, is based on resolution, but solvers also use stronger reasoning techniques in preprocessing steps. Only few attempts have been made to include stronger reasoning techniques in the search itself. The current work revisits the embedding of extended resolution into SAT solvers, and improves existing work. Experiments indicate that the approach is very promising.

1 Introduction

SAT solvers are used in many academic and industrial fields, either directly to solve a given formula, or SAT solving technology is embedded into other reasoners, for example bounded model checker, higher order logic provers, ASP or SMT solvers [4]. The huge application field is due to the recent development starting from the well known DPLL procedure [6]: modern solvers use *clause learning* [13] as deduction, employ *restarts* [7] to avoid searching at the wrong place, and have sophisticated decision heuristics to drive their search.

From a proof complexity point of view, this combination is known to be as powerful as general resolution [12], whereas the DPLL algorithm is only as strong as tree-like resolution. Hence, a CDCL solver can produce an unsatisfiability proof for a formula with a length that is polynomial compared to the shortest resolution proof. However, for propositional formulas, proof theory provides stronger deduction techniques, as for example *cutting planes* [5], or *extended resolution* [14]. There exist formulas where these three systems are known to be able to produce exponentially shorter proofs than resolution. Most modern SAT solvers do not exploit this reasoning power, or just exploit it in a limited way: During a formula simplification phase, LINGELING [3] uses a form of cutting planes, and extended resolution is used in *bounded variable addition* (BVA) [11]. SAT4J [9] can use cutting planes during search, but reports a slowdown of multiple orders of magnitude. Finally, Huang [8] and Audemard et al. [1] presented how to use extended resolution in the CDCL algorithm: Huang replaces a disjunction of two literals with a *fresh variable*, and Audemard et al. replace a conjunction of two literals with a fresh variable. However, both approaches are not used in any modern SAT solver. Even worse, *restricted extended resolution* (RER) has been dropped from the solver GLUCOSE again, due to its high code complexity. The presented work reimplements and analyzes RER. With the current state of the work, the performance of the SAT solver RISS can be improved slightly.¹ The implementation is a starting point to

integrate extended resolution into CDCL SAT solvers.

2 Extended Resolution

The SAT problem is to find a *model* for a propositional formula [4]. Formulas are sets of *clauses*, which are sets of *literals*. A literal x is a Boolean variable v , or a negated variable $\bar{v}$. Given two clauses $C = (l \vee E)$ and $D = (\bar{l} \vee F)$, then the resolvent R is $R = C \otimes D = (E \vee F)$. A resolution proof for a formula F is a sequence of resolvents R_i , where each resolvent is produced by resolving two clauses that are present either in the proof with a lower index, or in the formula F . Given a formula F , a variable v is *fresh*, if F does not contain v .

Let l_1 and l_2 be two literals that are present in the formula F . Extended resolution introduces a fresh variable v , and adds three clauses $(v \vee \bar{l}_1)$, $(v \vee \bar{l}_2)$, and $(\bar{v} \vee l_1 \vee l_2)$, to represent $v \leftrightarrow (l_1 \vee l_2)$ [14]. This functional dependency of v to l_1 and l_2 can be changed to any other Boolean function, and even the number of *input literals* can be modified, as in the simplification technique BVA [11].

2.1 Restricted Extended Resolution

Audemard et al. [1] introduce a fresh variable, if two consecutive learned clauses C_i and C_{i+j} share all except one literal: $C_i = (\bar{l}_1 \vee D)$ and $C_{i+j} = (\bar{l}_2 \vee D)$. The different literals l_1 and l_2 can occur only at the very first position. In the *local extension*, the fresh variable v represents the disjunction of l_1 and l_2 : $v \leftrightarrow (l_1 \vee l_2)$. The corresponding clauses are added to the formula F . The clauses C_i and C_{i+j} are replaced by $C' = (\bar{v} \vee D)$, because C_i and C_{i+j} can be obtained by resolution from C' with the two binary extension clauses. Hence, by introducing a fresh variable, the number of clauses is reduced. The more the variable v is used further in the unsatisfiability proof, the higher is the potential to reduce the size of the proof.

Once an extension is made, all clauses C in the formula that contain l_1 and l_2 are rewritten to $C := (C \setminus \{l_1, l_2\}) \cup \{v\}$. Similarly to the two learned clauses, the clauses that have been rewritten can be obtained by resolution again. Furthermore, whenever a new clause is learned, its literals

¹The source code of the RER extension in RISS is available at <http://tools.computational-logic.org>.

are checked to contain the pair l_1 and l_2 , so that this pair can be replaced by v . This check is done for all extensions that have been introduced. When an introduced variable becomes irrelevant, such a variable is removed again, and the original clauses are recovered, to decrease the cost of the check [1]. On average, Audemard et al. report an extension every 1000 conflicts.

Reducing the Procedural Overhead To find a matching pattern, Audemard et al. report a size based filter: if two consecutive learned clauses $C_i = (\overline{l_1} \vee D)$ and $C_{i+j} = (\overline{l_2} \vee E)$ do not have the same size, they are no candidates for an extension. Additionally, since D has to match E , a *Bloom filter* is introduced which checks the sums of the literals in D and E . The two sets D and E can only match, if $\sum_{l \in D} = \sum_{l \in E}$.

As presented in [11], increasing the number of variables in a formula to decrease the number of clauses in CDCL solvers does not influence the performance. Hence, the new RER implementation does not delete introduced variables. When an extension $v \leftrightarrow (l_1 \vee l_2)$ is added, for the smaller literal l_1 the literals l_2 and v are stored. When l_1 is used as the smaller literal in another extension with l_2 and w , then the old extension is not used for new clauses any more. Thus, the check for new clauses becomes less costly, and the cost for keeping all introduced variables is very low.

3 Evaluation and Future Work

The modified version of RER is implemented into the SAT solver RISS [10], and evaluated on the instances of the SAT competition with a 3600s timeout. The results for satisfiable ($\top$) and unsatisfiable formulas ($\perp$) are presented in Table 1. Surprisingly, RER helps to solve more satisfiable instances. Compared to the results of [1], on the whole benchmarks an extension is performed at most every 1000 conflicts. Most of the time, less than 10 extensions are made. Furthermore, on crafted formulas more extensions are made than on application formulas.

RISS is a state-of-the-art SAT solver, and with the given implementation, a starting point for further research is given. With the current implementation, heuristics for deciding when to introduce a fresh variable with extended resolution can be developed, and tested. Furthermore, new functional dependencies can be tested. As a first step,

and to stay as close as possible to RER, any functional dependency for a fresh variable v that can be hidden in two consecutive learned clauses should be considered. Candidates for such extensions are XOR gates, $v \leftrightarrow (a \oplus b)$, or IF-THEN-ELSE(ITE) gates, $v \leftrightarrow \text{ITE}(s, t, f)$. Both gates appear frequently in propositional formulas from cryptographic instances and scheduling, because ITE gates are used to encode binary decision diagrams into SAT. A first attempt to use ITE gates similarly to the conjunction of two literals in RER did not lead to any improvements. We believe that good heuristics and useful functional dependencies can be discovered by automatic configuration tools. Then, the addition of extended resolution to the CDCL algorithm might result in the next exponential boost of SAT solvers.

Acknowledgements

The author thanks the ZIH of TU Dresden for providing the computational resources for the empirical evaluation.

References

- [1] Gilles Audemard, George Katsirelos, and Laurent Simon. A restriction of extended resolution for clause learning sat solvers. AAAI Press, 2010.
- [2] A. Balint, A. Belov, M. J.H. Heule, and M. Järvisalo, editors. *Proceedings of SAT Challenge 2013*, volume B-2013-1 of *Department of Computer Science Series of Publications B*. University of Helsinki, Helsinki, Finland, 2013.
- [3] A. Biere. Lingeling, Plingeling and Treengeling entering the SAT competition 2013. In Balint et al. [2], pages 51–52.
- [4] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*. IOS Press, Amsterdam, 2009.
- [5] W. Cook, C.R. Coullard, and Gy. Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25 – 38, 1987.
- [6] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.
- [7] Jinbo Huang. The effect of restarts on the efficiency of clause learning. In *IJCAI*, pages 2318–2323, 2007.
- [8] Jinbo Huang. Extended clause learning. *Artif. Intell.*, 174(15):1277–1284, October 2010.
- [9] Daniel Le Berre and Anne Parrain. The SAT4J library, release 2.2, system description. *JSAT*, 7:59–64, 2010.
- [10] N. Manthey. The SAT solver RISS3G at SC 2013. In Balint et al. [2], pages 72–73.
- [11] Norbert Manthey, Marijn J. H. Heule, and Armin Biere. Automated reencoding of Boolean formulas. In *HVC 2012*, 2012.
- [12] Knot Pipatsrisawat and Adnan Darwiche. On modern clause-learning satisfiability solvers. *J. Autom. Reasoning*, 44(3):277–301, 2010.
- [13] João P. Marques Silva and Karem A. Sakallah. GRASP - a new search algorithm for satisfiability. In *ICCAD 1996*, pages 220–227, Washington, 1996. IEEE Computer Society.
- [14] G. Tseitin. On the complexity of proofs in propositional logics. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, volume 2. Springer-Verlag, 1983.

Table 1: Evaluation on recent competition benchmarks.

Category	Year	RISS		RISS+RER	
		$\top$	$\perp$	$\top$	$\perp$
Crafted	2011	79	42	81	42
	2012	229	145	228	146
	2013	104	84	106	86
Application	2011	86	117	89	116
	2012	237	296	243	296
	2013	95	86	98	86