A New Clause Learning Scheme for Efficient Unsatisfiability Proofs

Knot Pipatsrisawat and **Adnan Darwiche**
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095 USA
{thammakn,darwiche}@cs.ucla.edu

Abstract

We formalize in this paper a key property of *asserting* clauses (the most common type of clauses learned by SAT solvers). We show that the formalized property, which is called empowerment, is not exclusive to asserting clauses, and introduce a new class of learned clauses which can also be empowering. We show empirically that (1) the new class of clauses tends to be much shorter and induce further backtracks than asserting clauses and (2) an empowering subset of this new class of clauses significantly improves the performance of the Rsat solver on unsatisfiable problems.

Introduction

An important technique that underlies most modern SAT solvers is conflict-clause learning (Marques-Silva and Sakallah 1996). The algorithm for deriving conflict clauses in these SAT solvers is based on using implication graphs to derive a particular class of conflict clauses known as *asserting clauses* (Marques-Silva and Sakallah 1996; Moskewicz et al. 2001). In this paper, we formalize an essential property of asserting clauses, which may also be satisfied by some non-asserting clauses. We then define a new class of clauses which is a relaxation of asserting clauses and show that they tend to be shorter than asserting clauses and lead to much further backtracks. Empirical results show that selectively learning the new type of clauses significantly improves the performance of Rsat, the winner of the industrial category of the SAT'07 competition, on unsatisfiable problems.

We begin the next section by presenting a model of modern clause-learning SAT solvers, which we use as a basis for our discussions in this paper. Then, we introduce a property of conflict clauses, called *empowerment*. Next, we define a new class of conflict clauses and present an efficient algorithm for deriving them. Then, we discuss a special subset of this new class which satisfies empowerment. Finally, we present experimental results and end with conclusions. Proofs of all propositions in this paper can be found in (Pipatsrisawat and Darwiche 2008).

A Model of Modern SAT Solvers

We begin with some basic notations and definitions. If Δ and α are two Boolean formulas and ℓ is a literal, we write

```
input  : CNF formula $\Delta$
output : A solution of $\Delta$ or unsat if $\Delta$ is not satisfiable
1  $D \leftarrow \langle\rangle, \Gamma \leftarrow \{\}$
2  while true do
3      if $S = (\Delta, \Gamma, D)$ is 1–inconsistent then //Conflict
4          if $D = \langle\rangle$ then return unsat
5          $\alpha \leftarrow$ an asserting clause for $S$
6          $m \leftarrow$ the assertion level of $\alpha$
7          $D \leftarrow D_m$ // the first $m$ decisions
8          $\Gamma \leftarrow \Gamma \wedge \alpha$
9      else
10         Choose a literal $\ell$ s.t. $S \not\vdash \ell$ and $S \not\vdash \neg\ell$
11         if $\ell =$ null then return $D$ // satisfiable
12         $D \leftarrow D, \ell$
13     end
14 end
```
Algorithm 1: A pseudo-code of a SAT solver

$\Delta \models \alpha$ to mean that Δ entails α, and write $\Delta \vdash \ell$ to mean that literal ℓ can be derived from Δ using unit resolution. Furthermore, we may treat a clause as the set of literals in the clause and a CNF formula as the set of clauses it contains.

Algorithm 1 is a pseudo-code of a clause-learning SAT solver, which is based on making variable assignments, called *decisions*. It starts with an empty decision sequence D and an empty set of learned clauses Γ (Line 1). It then iterates until it either proves the satisfiability or unsatisfiability of the CNF Δ. In each iteration, the conjunction of Δ, learned clauses Γ, and decisions D are checked for inconsistency using unit resolution (Line 3). If unit resolution finds an inconsistency, the algorithm does one of two things:

- If the decision sequence is empty, the CNF Δ must be unsatisfiable and the algorithm terminates (Line 4).
- If the decision sequence is not empty, an asserting clause α is generated with an assertion level m. The algorithm then erases all decisions made after level m, adds α to Γ, and moves on to the next iteration (Lines 5-8).

If unit resolution detects no inconsistency, the algorithm tries to find a literal ℓ whose value is not currently implied or falsified by unit resolution, and adds it to the decision sequence (Line 12). If no such literal is found, the algorithms terminates having proven satisfiability (Line 11). We will now provide the missing definitions.

- A *decision sequence* is an ordered set of literals $D = \langle \ell_1, \ldots, \ell_n \rangle$. Each literal ℓ_k is called the *decision* at *level* k. We write D_m to denote the subsequence $\langle \ell_1, \ldots, \ell_m \rangle$. When appropriate, we will treat a decision sequence as the conjunction or the set of all literals in the sequence.
- A *SAT state* is a tuple (Δ, Γ, D), where Δ and Γ are CNFs such that $\Delta \models \Gamma$, and D is a decision sequence. We will write S_k to denote the state (Δ, Γ, D_k).
- (Δ, Γ, D) is 1*–inconsistent* iff $\Delta \wedge \Gamma \wedge D \vdash \mathsf{false}$.
- A literal ℓ is *implied by state* $S = (\Delta, \Gamma, D)$ *at level* n, written $S \vdash_n \ell$, iff n is the smallest integer for which $\Delta \wedge \Gamma \wedge D_n \vdash \ell$. We say that the *implication level* of literals $\ell, \neg\ell$ is n in this case, write $S \vdash \ell$ to mean $S \vdash_k \ell$ for some k, and write $S \not\vdash \ell$ to mean $S \not\vdash_k \ell$ for all k.
- $S = (\Delta, \Gamma, \langle \ell_1, \ldots, \ell_n \rangle)$ is *normal* iff for $1 \leq k \leq n$, S_{k-1} is not 1–inconsistent, $S_{k-1} \not\vdash \ell_k$ and $S_{k-1} \not\vdash \neg\ell_k$.

The notion of normal states prohibits SAT solvers from making a decision in the presence of a conflict. The state S on Line 3 of Algorithm 1 is always normal. Therefore, from now on, we will assume that every SAT state is normal.

We are now ready to define the remaining notions used in Algorithm 1. An asserting clause is a special type of conflict clause that satisfies some strong conditions, so we start first by defining the weaker notion of a conflict clause. Our definition of conflict clause closely follows the graphical definition in (Zhang et al. 2001).

Definition 1 (Conflict Clause) *Let $S = (\Delta, \Gamma, D)$ be a SAT state. A clause $\alpha = \ell_1 \vee \ldots \vee \ell_m$ is a* <u>*conflict clause*</u> *for state S iff:*

1. *$\Delta \wedge \Gamma \wedge \neg\alpha \vdash \mathsf{false}$. That is, α is implied by $\Delta \wedge \Gamma$ and this can be proven using unit resolution.*
2. *For each ℓ_i, $S \vdash \neg\ell_i$. That is, the literals $\neg\ell_i$ are a subset of the implications derived by unit resolution in state S.*

Every conflict clause must contain at least one literal that is falsified in the last level. Otherwise, the SAT state would not be normal, because the previous state must be 1–inconsistent. In practice, however, modern SAT solvers insist on learning conflict clauses that contain *exactly* one literal falsified in the last level.

Definition 2 (Asserting Clause) *A conflict clause α of a SAT state $S = (\Delta, \Gamma, D)$ is an* <u>*asserting clause*</u> *iff it has exactly one literal ℓ with implication level $|D|$. The literal ℓ is called the* <u>*asserted literal*</u> *of α. Moreover, the* <u>*assertion level*</u> *of clause α is defined as the highest implication level $k < |D|$ attained by some literal in α. If $|\alpha| = 1$, the assertion level is defined to be zero.*

An asserting clause is guaranteed to become unit once the solver backtracks to the assertion level. Moreover, an asserting clause can always be derived for a 1–inconsistent SAT state that has at least one decision.

Proposition 1 *A SAT state $S = (\Delta, \Gamma, D)$ with $|D| > 0$ is 1–inconsistent iff it has an asserting clause.*

The completeness of Algorithm 1 can be shown by realizing that asserting clauses learned by the algorithm never repeat (otherwise, the conflict would have been prevented). Since every conflict comes with an asserting clause (Proposition 1) and there are only finitely many clauses, the algorithm can only experience a finite number of conflicts. It cannot keep making decisions indefinitely either and, therefore, has to terminate. This proof will be referred to again when we introduce a new class of conflict clauses.

Empowerment

In this section, we introduce a new property, which is satisfied by all asserting clauses.

Definition 3 (1–Empowerment) *Let $\alpha \Rightarrow \ell$ be a clause where ℓ is a literal and α is a conjunction of literals. The clause is* <u>*1–empowering*</u> *with respect to CNF Δ via ℓ iff*

1. *$\Delta \models (\alpha \Rightarrow \ell)$: the clause is implied by Δ.*
2. *$\Delta \wedge \alpha \not\vdash \ell$: the literal ℓ cannot be derived from $\Delta \wedge \alpha$ using unit resolution.*

We will also say in this case that ℓ is the empowering literal of clause $\alpha \Rightarrow \ell$. Moreover, we will say that a clause is 1–empowering with respect to state (Δ, Γ, D) iff it is 1–empowering with respect to $\Delta \wedge \Gamma$. Intuitively, a clause is 1–empowering with respect to a CNF Δ iff (1) it is logically implied by the CNF and (2) adding it to the CNF allows unit resolution to derive a *new implication* that could not be derived without the clause. Consider the CNF $\Delta = (a \vee b) \wedge (\neg a \vee c) \wedge (b \vee \neg c \vee d)$ for an example, and the clause $(b \vee d)$ which is implied by the CNF. Adding $\neg d$ to Δ does not allow unit resolution to derive b even though b is implied by $\Delta \wedge \neg d$. Yet, this derivation becomes possible once we add the clause $(b \vee d)$ to Δ. Hence, the clause is 1–empowering with respect to Δ via literal b. Note, however, that the clause $(b \vee c)$ is not 1–empowering with respect to the same CNF, because $\Delta \wedge \neg b \vdash c$ and $\Delta \wedge \neg c \vdash b$.

This means that adding a conflict clause that is not 1–empowering to the knowledge base will not lead to any new implication. Note that if a clause is not 1–empowering with respect to a knowledge base, adding more clauses to the knowledge base will not make it so.

Proposition 2 *All asserting clauses of a SAT state S are 1–empowering with respect to S and the asserted literals are their empowering literals.*

Given this result, the learning component of Algorithm 1 can be viewed as one that tries to *empower* unit resolution so that it becomes more complete. In particular, each time unit resolution discovers a contradiction, it does so under a particular decision sequence. When this sequence is not empty, it means that unit resolution has missed an implication due to its incompleteness. By adding the empowering clause, we are effectively allowing unit resolution to discover this implication at an earlier stage, using only part of the current decision sequence.[1] Therefore, the shorter the empowering clause is, the better it is in making unit resolution more complete. Moreover, an empowering clause with a smaller assertion level will allow unit resolution to derive the missed

[1]In (del Val 1994), a similar analysis on the completeness of unit resolution is given in the context of knowledge compilation.

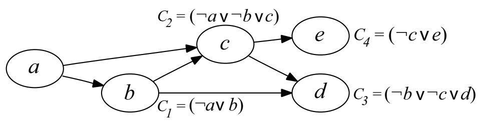
$C_2 = (\neg a \vee \neg b \vee c)$
$C_4 = (\neg c \vee e)$
$C_1 = (\neg a \vee b)$
$C_3 = (\neg b \vee \neg c \vee d)$
a
b
c
d
e

Figure 1: An implication graph. Each node is labeled with an implication and its reason.

implication under a smaller subset of the current decision sequence.

A New Class of Conflict Clauses

In this section, we introduce a new class of clauses, which is a relaxation of asserting clauses. This class of clauses may satisfy the empowerment property as well. Later, we argue and show empirically that an empowering subset of this new class could improve the performance of SAT solvers.

Definition 4 (Bi-Asserting Clause) *A conflict clause β of a SAT state (Δ, Γ, D) is a* $\underline{\textit{bi-asserting clause}}$ *if it has exactly two literals with implication level $|D|$. The assertion level of β is defined as the second highest implication level of its literal. If $|\beta| = 2$, its assertion level is defined to be zero.*

We will show in a later section that a class of bi-asserting clauses tends to be much shorter and have smaller assertion levels than asserting clauses for the same conflict. Deriving a bi-asserting clause is similar to deriving a normal asserting clause. The standard algorithm performs a series of resolutions to derive an asserting clause (see (Eén and Sörensson 2003; Ryan 2004)). We will refer to it as Algorithm 2 in our discussion. We describe this algorithm through the following example. Let Δ be the conjunction of $C_1 = (\neg a \vee b), C_2 = (\neg a \vee \neg b \vee c), C_3 = (\neg b \vee \neg c \vee d), C_4 = (\neg c \vee e), C_5 = (\neg d \vee \neg e)$. Now, consider the state $S = (\Delta, \mathsf{true}, \langle a \rangle)$. From this state, unit resolution will derive some implications before discovering a conflict. The derivations of implications are visualized in Figure 1. An edge $X \rightarrow Y$ in this graph indicates that X's literal directly contributes to the implication of Y's. A reason of an implication is the clause in which the literal becomes unit. In this case, assume that the implications b, c, d, e are discovered in this order and that C_5 is found to be empty (all of its literals have been falsified). The following trace shows how Algorithm 2 derives an asserting clause from this example.

Step	Clause	Explanation
1	$(\neg d \vee \neg e)$	C_5: the empty clause
2	$(\neg c \vee e)$	C_4: the reason of e
3	$(\neg c \vee \neg d)$	resolving 1 & 2 on e
4	$(\neg b \vee \neg c \vee d)$	C_3: the reason of d
5	$(\neg b \vee \neg c)$	resolving 3 & 4 on d
6	$(\neg a \vee \neg b \vee c)$	C_2: the reason of c
7	$(\neg a \vee \neg b)$	resolving 5 & 6 on c
8	$(\neg a \vee b)$	C_1: the reason of b
9	$(\neg a)$	resolving 7 & 8 on b

The algorithm starts by initializing the conflict clause to be the empty clause (Step 1). Then, it picks a literal in the clause whose negation was implied *last* (e in this case). The algorithm resolves the reason of this literal (Step 2) with the conflict clause and replaces the conflict clause with the resolvent, $(\neg c \vee \neg d)$ (Step 3). Next, since, in the current conflict clause, the negation of $\neg d$ was implied last, the algorithm resolves its reason (Step 4) with the conflict clause and produces $(\neg b \vee \neg c)$ (Step 5). This process is repeated until the conflict clause becomes asserting (Step 9). We refer to the clauses in Steps 3,5,7 as *intermediate conflict clauses*.

This algorithm can be modified to derive bi-asserting clauses without any added complexity. We only need to detect when the intermediate conflict clause contains two (instead of one) literals falsified at the conflict level. Note, however, that not every bi-asserting clause is 1–empowering and the algorithm described above, does not guarantee to produce 1–empowering bi-asserting clauses. For example, the bi-asserting clause $(\neg c \vee \neg d)$ is not 1–empowering for the above example, while $(\neg b \vee \neg c)$ is. This suggests that bi-asserting clauses should be learned with care to avoid adding clauses that will not contribute any new implication.

Learning 1–Empowering Bi-Asserting Clauses

In this section, we address the issue of learning 1–empowering bi-asserting clauses. In general, checking whether the clause $\alpha \Rightarrow \ell$ is empowering with respect to CNF Δ via literal ℓ can be done by checking if $\Delta \wedge \neg\alpha \vdash \ell$. The time complexity of this test is linear in the size of the knowledge base. In practice, however, this test would incur too much overhead. Hence, we will next present an efficient algorithm that detects the empowerment of the derived bi-asserting clause, *but only with respect to the clauses used in its derivation.* Even though the derived bi-asserting clauses are not guaranteed to be 1–empowering with respect to other clauses in the knowledge base, we will show in the next section that they tend to in the majority of cases. Our approach for generating empowering bi-asserting clauses is based on the notions of merge resolutions. Defined in (Andrews 1968), a resolution between clauses C_1 and C_2 is a *merge resolution* iff C_1 and C_2 share a literal ($C_1 \cap C_2 \neq \emptyset$).

Proposition 3 *An intermediate conflict clause of Algorithm 2 is 1–empowering with respect to the clauses used in its derivation if at least a step in the derivation is a merge resolution.*

Based on this result, all we need to ensure that the conflict clause is 1–empowering is the existence of a single merge resolution step in the resolution derivation of the clause. This additional check incurs very little overhead.

We close this section by noting that integrating bi-asserting clauses into Algorithm 1 still maintains the completeness of the algorithm given that a standard unit propagation algorithm is used (Ryan 2004). This is due to the result in (Ryan 2004), which states that every intermediate conflict clause in the derivation must not already appear in the knowledge base at the time of the derivation. As a result, our completeness proof presented earlier still holds.

Experimental Results

We modified Rsat (Pipatsrisawat and Darwiche 2007), the winner of the SAT'07 competition (industrial category), to detect any occurrence of 1–empowering bi-asserting clauses (with respect to the clauses in the derivation) during conflict clause derivation. If found, the bi-asserting clause is

Family	Total	# solved SAT			# solved UNSAT		
		Rsat	Rsat+	Rsat++	Rsat	Rsat+	Rsat++
difp	29	29	28	28	0	0	0
dlx iq unsat 1	32	1	0	2	10	16	16
grieu 2005	32	20	18	19	0	0	0
engine	10	0	0	0	7	7	7
fpga	21	11	11	11	3	1	4
fvp sat 3	20	20	20	20	0	0	0
fvp unsat 1,2	26	1	1	1	25	25	25
IBM	274	107	104	105	164	165	165
liveness sat 1	10	5	7	6	0	0	0
liveness unsat 1,2	21	0	0	0	7	7	7
narain 2005	10	6	6	6	2	2	2
npe	6	3	3	3	1	1	1
pipe ooo	29	0	0	0	12	14	16
pipe sat 1	10	9	10	10	0	0	0
pipe unsat 1.0,1.1	27	0	0	0	14	20	19
SAT-Race	200	68	64	66	101	110	110
SAT Comp. 07	173	49	48	50	56	59	60
vliw sat 2	9	9	9	9	0	0	0
vliw unsat 2,3,4	10	0	0	0	0	6	6
Total	949	338	329	336	402	433	438

Table 1: Performance of Rsat with different learning schemes.

learned instead of the asserting clause otherwise learned. We considered Rsat with the following learning schemes: (i) learn only asserting clauses (normal Rsat) (ii) learn 1–empowering bi-asserting clause when possible (Rsat+) (iii) learns 1–empowering bi-asserting clause only when its assertion level is at least 2 levels smaller than that of the asserting clause (Rsat++).

Although empowerment is only with respect to the clauses used in the derivation, in practice, it usually results in empowerment with respect to the whole formula. We found that, on 95% of the problems, over 80% of the bi-asserting clauses derived are 1–empowering with respect to the whole formula at the time of learning.

We experimented with 949 SAT problems from previous SAT/SAT-Race competitions and contemporary libraries.[2] Each solver is given 1,800 seconds per problem on a 3.8GHz machine with 3GB of RAM. Table 1 reports the number of problems solved by different versions. According to this table, Rsat+ and Rsat++ perform considerably better than Rsat. For satisfiable problems, Rsat+ tends to be worse than the others, whereas Rsat and Rsat++ have comparable performances. For unsatisfiable problems, however, Rsat++ and Rsat+ solve 36 and 31 more problems than Rsat, respectively. Moreover, the new learning scheme decreases the running time on unsatisfiable problems by 34% (71,246 s for Rsat and 47,400 s for Rsat++). Overall, Rsat++ used only 99,209 seconds, while Rsat used 114,600 seconds.[3]

We have also evaluated the version of Rsat that learns bi-asserting clauses regardless of their empowerment. This version of Rsat solved only 647 problems (normal Rsat solved 740 problems). This is because these bi-asserting clauses are not necessarily capable of generating any new implications. So, in the long run, they only contribute to the overhead of unit propagation no matter how short they are.

We also measured the sizes of different conflict clauses derived by Rsat++ and found that, on average, the size of a bi-asserting clause is slightly less than half the size of an asserting clause. Moreover, on average, the backtrack induced by a bi-asserting clause is about 5-6 times longer than the one induced by an asserting clause.

[2]http://www.satcompetition.org/2007,http://fmv.jku.at/sat-race-2006/, http://www.research.ibm.com/haifa/projects/verification/RB_Homepage/fvbenchmarks.html, http://www.miroslav-velev.com/sat_benchmarks.html

[3]Only problems solved by both Rsat and Rsat++ are considered.

Related Work and Conclusions

Learning non-asserting clauses have been previously proposed by (Ryan 2004) and (Dershowitz, Hanna, and Nadel 2007). In both work, non-asserting clauses are learned *in addition* to asserting clauses (i.e. the solvers may add two clauses per conflict). Moreover, additional clauses learned in both cases may not be empowering nor bi-asserting.

In conclusion, we formalized a key property of asserting clauses called empowerment and presented a new class of conflict clauses called bi-asserting clauses. Our experiments showed that, by selectively learning empowering bi-asserting clauses, Rsat produces shorter conflict clauses, performs longer backtracks, and performs better on unsatisfiable problems.

Acknowledgments

This work has been partially supported by Air Force grant #FA9550-05-1-0075 and by NSF grant #IIS-0713166.

References

Andrews, P. B. 1968. Resolution with merging. *J. ACM* 15(3):367–381.

del Val, A. 1994. Tractable databases: How to make propositional unit resolution complete through compilation. In *Proceedings of KR-94, pp. 551-561.*

Dershowitz, N.; Hanna, Z.; and Nadel, A. 2007. Towards a better understanding of the functionality of a conflict-driven sat solver. In *SAT*, 287–293.

Eén, N., and Sörensson, N. 2003. An extensible sat-solver. In *Proceedings of SAT 2003.*

Marques-Silva, J. P., and Sakallah, K. A. 1996. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of IEEE/ACM Intl. Conf. on CAD*, 220–227.

Moskewicz, M.; Madigan, C.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient sat solver. In *Proceedings of DAC'01, June 2001.*

Pipatsrisawat, K., and Darwiche, A. 2007. Rsat 2.0: Sat solver description. Technical Report D–153, Automated Reasoning Group, Comp. Sci. Department, UCLA.

Pipatsrisawat, K., and Darwiche, A. 2008. A new clause learning scheme for efficient unsatisfiability proofs. Technical Report D–156, Automated Reasoning Group, Comp. Sci. Department, UCLA.

Ryan, L. 2004. Efficient Algorithms for Clause-Learning SAT Solvers. Master's thesis, Simon Fraser University.

Zhang, L.; Madigan, C. F.; Moskewicz, M. W.; and Malik, S. 2001. Efficient conflict driven learning in boolean satisfiability solver. In *ICCAD-01*, 279–285.