On the Power of Clause-Learning SAT Solvers with Restarts*

Knot Pipatsrisawat and Adnan Darwiche

University of California, Los Angeles, USA
{thammakn,darwiche}@cs.ucla.edu

Abstract. In this work, we improve on existing work that studied the relationship between the proof system of modern SAT solvers and general resolution. Previous contributions such as those by Beame *et al* (2004), Hertel *et al* (2008), and Buss *et al* (2008) demonstrated that variations on modern clause-learning SAT solvers were as powerful as general resolution. However, the models used in these studies required either extra degrees of non-determinism or a preprocessing step that are not utilized by any state-of-the-art SAT solvers in practice. In this paper, we prove that modern SAT solvers that learn asserting clauses indeed p-simulate general resolution without the need for any additional techniques.

1 Introduction

It is well-known that modern clause-learning SAT solvers (and their original ancestor, the DPLL algorithm [7]) can be interpreted as resolution-based proof systems [4]. For each unsatisfiable formula, these solvers can be viewed as engines that produce refutation proofs. One central question in this research direction is whether the proof system implemented by modern SAT solvers still has enough freedom to generate a short resolution proof (relative to general resolution) for every unsatisfiable formula. The answer to this theoretical question could have important practical implications on the efficiency of modern SAT solvers.

There is much previous work on this subject that demonstrates the strength of variations on modern clause-learning SAT solvers. However, equivalence with respect to general resolution has yet to be proven for modern clause-learning SAT solvers as they are practiced today. In [4], Beame *et al* showed that a proof system based on a more general variation of modern SAT solvers was as powerful as general resolution. The proof presented, however, requires the solver to make decisions on variables that are already implied by unit resolution. This modification introduces an extra degree of non-determinism "that would be very hard to exploit in practice" [9]. Hertel *et al* [9] proved a slightly weaker result that clause-learning solvers *effectively p-simulate* general resolution. This approach allows them to introduce a preprocessing step to transform the CNF into one (with some new variables) that can be efficiently solved by a more practical model of solvers (the model used in [9] is based on the one developed by Van

* This is an updated version of the paper that appeared in the proceedings of CP-09.

Gelder in [16]). Buss *et al* [5] also took a similar approach by modifying the input CNF (and introduced some new variables) to show that a generalized variation of clause-learning algorithm, which allows decision making past conflicts, can effectively p-simulate general resolution.

In this work, we show that modern clause-learning SAT solvers, without any extra modifications, are indeed as powerful as general resolution. In particular, we prove that the proof system implemented by modern clause-learning solvers (which uses unit resolution, asserting clause learning, and restarting) p-simulates general resolution. We show that this result holds for any asserting-clause learning scheme. Our proof does not require any preprocessing or making decisions on implied literal or past a conflict. This result implies that modern SAT solvers in their current form are capable of producing proofs that are as "short" as any resolution proof, given appropriate branching heuristic and restart policy.

The proof of our main result is made possible by the help of two important concepts, namely *1–empowerment* [13] and *1–provability*. Together, they allow us to more accurately capture the power of modern clause-learning SAT solvers and to avoid the need to introduce any technique not present in practice.

The rest of the paper is organized as follows. In the next section, we discuss basic notations and definitions. In Section 3, we present our model of modern clause-learning SAT solvers and the proof system associated with it. Then, in Section 4, we present some interesting key results, which provide some insights on the power of modern SAT solvers and allow us to prove the main result. Next, we present our main result in Section 5. Finally, we discuss previous work in Section 6 and conclude in Section 7.

2 Preliminaries

In this section, we review some basic notations related to propositional logic and proof systems. If Δ and α are two boolean formulas and ℓ is a literal, we write $\Delta \models \alpha$ 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.

2.1 Proof Systems

A *proof system* is a language for expressing proofs that can be verified in time polynomial in the size of the proof [6]. In this work, we are concerned only with proof systems based on propositional resolution [14]. The *resolution* between clause $\alpha \vee x$ and $\beta \vee \neg x$ is the derivation of clause $\alpha \vee \beta$ (i.e., the *resolvent*). In this case, x is called the *resolved variable*. To make our analysis as related to modern SAT solvers as possible, the *weakening rule*, which allows introduction of arbitrary literals into existing clauses, is not permitted here.

Definition 1. *A resolution proof (or resolution derivation) of the clause C_k from the CNF Δ is a sequence of clauses $\Pi = C_1, C_2, ..., C_k$ where each clause C_i is either in Δ or is a resolvent of clauses preceding C_i.*

We will also treat a resolution proof as the set of clauses in it. The size of a proof is the number of clauses in it. A resolution proof of the empty clause (i.e., false) is called a *refutation proof*. The notion of p-simulation, which was introduced in [6], is used to compare the power of two proof systems. The definition presented here is obtained from [9].

Definition 2 (P-Simulation). *Proof system S p-simulates proof system T, if, for every unsatisfiable formula Δ, the shortest refutation proof of Δ in S is at most polynomially longer than the shortest refutation proof of Δ in T.*

Intuitively, if proof system S p-simulates proof system T, it means that S is unrestricted enough to express proofs that are as short as those expressible in T. As far as resolution proofs are concerned, *general resolution*, which allows any resolution operation to be performed, is the most powerful proof system. Other resolution proof systems that are known to be less powerful (i.e., do not p-simulate general resolution) include *tree-like resolution*, *linear resolution*, and *regular resolution* (see Section 2.3 of [4] for a good review).

3 Modern Clause-Learning SAT Solvers as a Proof System

3.1 Modern Clause-Learning SAT Solvers

In this section, we describe a model of modern clause-learning SAT solvers. Included in our model are the following techniques: unit resolution [7], clause-learning [11, 17], restarting [8], and non-chronological backtracking [11, 3] (i.e., far-backtracking as termed by [15]). Algorithm 1 shows a pseudo code of a typical clause-learning SAT solver with restarts, which we will refer to as CLR from now on. We will first provide a high-level description of the algorithm before giving formal definitions of its different components.

This algorithm is based on making variable assignments called *decisions*. It starts with an empty decision sequence D and an empty set of learned clauses Γ (Lines 1-2). It then iterates until it either proves the satisfiability or unsatisfiability of the input. In each iteration, the conjunction of the input CNF Δ, learned clauses Γ, and decisions D are checked for inconsistency using unit resolution (Line 4). 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 6).
- If the decision sequence is not empty, a clause α is generated and a level m is computed based on α. The algorithm then erases all decisions made after level m, adds α to Γ, and moves on to the next iteration (Lines 7-10).[1]

[1] The clause α is known as an asserting clause and m as the assertion level. We will define them formally later.

Algorithm 1: CLR: Clause-learning SAT solver with restarts.

input : CNF formula Δ
output: A solution of Δ or unsat if Δ is not satisfiable

```
1  D ← ⟨⟩  // Decision literals
2  Γ ← true  // Learned clauses
3  while true do
4      if S = (Δ, Γ, D) is 1–inconsistent then
           // There is a conflict.
5          if D = ⟨⟩ then
6              return unsat
7          α ← an asserting clause of S
8          m ← the assertion level of α
9          D ← D_m // the first m decisions
10         Γ ← Γ ∧ α
11     else
           // There is no conflict.
12         if time to restart then
13             D ← ⟨⟩
14             S ← (Δ, Γ, D)
15         Choose a literal ℓ such that S ⊭ ℓ and S ⊭ ¬ℓ
16         if ℓ = null then
17             return D  // satisfiable
18         D ← D, ℓ
```

If unit resolution detects no inconsistency, the solver has an option of restarting, which amounts to resetting the decision sequence to the empty sequence (Line 13). After that, the solver makes a decision by selecting a literal ℓ whose value is not currently implied or falsified by unit resolution, and adds it to the decision sequence (Line 18). If no such literal is found, the algorithms terminates having proved satisfiability (Line 17). We will now provide the missing definitions.

- A *decision sequence* is an ordered set of literals $D = \langle \ell_1, \ldots, \ell_k \rangle$. Each literal ℓ_i is called the *decision* at *level* i. We write D_m to denote the subsequence $\langle \ell_1, \ldots, \ell_m \rangle$.
- 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).
- A CNF Δ is *1–inconsistent* iff $\Delta \vdash \mathsf{false}$. It is *1–consistent* otherwise. A SAT state (Δ, Γ, D) is *1–inconsistent* (*1–consistent*) iff $\Delta \wedge \Gamma \wedge D$ is 1–inconsistent (1–consistent). It is normal for an unsatisfiable CNF to be 1–consistent.
- A literal ℓ is *implied by state* $S = (\Delta, \Gamma, D)$ *at level* k, written $S \vdash_k \ell$, iff k is the smallest integer for which $\Delta \wedge \Gamma \wedge D_k \vdash \ell$. We say that the *implication level* of literals $\ell, \neg\ell$ is k in this case, write $S \vdash \ell$ to mean $S \vdash_i \ell$ for some i, and write $S \nvdash \ell$ to mean $S \nvdash_i \ell$ for all i.

– A state $S = (\Delta, \Gamma, \langle \ell_1, \ldots, \ell_k \rangle)$ is *normal* iff for all $1 \leq i \leq k$, S_{i-1} is 1–consistent, $S_{i-1} \not\vdash \ell_i$ and $S_{i-1} \not\vdash \neg \ell_i$.

The notion of normal states prohibits SAT solvers from (1) making a decision in the presence of a conflict and (2) making a decision on a variable that is already assigned a value. By construction, the state S on Lines 4 and 14 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 last two notions used in Algorithm 1: asserting clause and assertion level. An asserting clause is a special type of conflict clause, so we start first by defining the notion of a conflict clause. Our definition of conflict clause closely follows the graphical definition in [17].

Definition 3 (Conflict Clause). *Let $S = (\Delta, \Gamma, D)$ be a 1–inconsistent SAT state. A clause $\alpha = \ell_1 \vee \ldots \vee \ell_m$ is a conflict clause of state S iff:*

1. *$\Delta \wedge \Gamma \wedge \neg\alpha \vdash \mathsf{false}$. That is, we can show that α is implied by $\Delta \wedge \Gamma$ using just unit resolution.*
2. *For each literal ℓ_i, $S \vdash \neg\ell_i$. That is, the literals $\neg\ell_i$ are a subset of the implications (or decisions) discovered by unit resolution in state S.*

In [4], it was shown (in their Proposition 4) that every conflict clause obtained from a cut on an implication graph (or a conflict graph, to be more precise) can be derived from the current knowledge base ($\Delta \wedge \Gamma$) using what is known as *trivial resolution derivation*, which captures the kind of resolution performed by virtually all modern clause-learning SAT solvers [4]. A *trivial resolution derivation* is a resolution derivation in which:

1. Every resolution step (except the very first) is performed between the last resolvent and a clause in the knowledge base.
2. The resolved variables are all distinct.

Our definition of conflict clause is independent of the notion of implication graph and is slightly more general (for example, it encompasses unconventional clauses derived in [2]). Nevertheless, we will later show that all of the conflict clauses that we care to learn can still be "derived" using trivial resolution derivation (to be proven later in Proposition 3).

In any case, modern SAT solvers, in practice, insist on learning conflict clauses that contain *exactly* one literal falsified at the last level.

Definition 4 (Asserting Clause). *A conflict clause α of a SAT state $S = (\Delta, \Gamma, D)$ is an asserting clause iff it has exactly one literal ℓ with implication level $|D|$. The literal ℓ is called the asserted literal of α. Moreover, the assertion level of clause α is defined as the highest implication level $k < |D|$ attained by some literal in α. If α contains only one literal, the assertion level is defined to be zero.*

Given a 1–inconsistent state, there always exists an asserting clause for it [13]. This result ensures that the execution of Line 7 of Algorithm 1 will always succeed. This completes our description of CLR.

3.2 Clause-Learning Schemes

CLR can employ various *learning schemes* to derive conflict clauses (Line 7). Even though we insist on deriving asserting clauses in Algorithm 1, in general, non-asserting clauses may be used.[2] In our context, it is sufficient to view a learning scheme as a function that produces a conflict clause for every 1–inconsistent SAT state. In this work, we will focus on a certain class of learning schemes called *asserting learning schemes*, which always produces asserting clauses. We will later show (in Proposition 3) that every asserting clause can essentially be derived from a trivial resolution proof.

Given a learning scheme LS, we use CLR_{LS} to denote the SAT algorithm obtained by applying LS on Line 7 of Algorithm 1. We use CLR to denote the algorithm with any learning scheme.

3.3 Non-Determinism in CLR

Given a learning scheme LS, the only sources of non-determinism remaining in CLR_{LS} are (1) the branching heuristic, (2) the restart policy and (3) the implementation of unit resolution.[3]

In this work, we utilize the notion of *extended branching sequence* (defined by [4]) to capture (1) and (2). An extended branching sequence is simply a sequence of literals and special symbol $\mathbf{R}$ that is used to control decision making and restarting in CLR. For example, $\sigma = \langle x, \neg y, \mathbf{R}, \neg x \rangle$ indicates that the first decision should be $x = \mathsf{true}$, the second decision should be $y = \mathsf{false}$, then the solver should restart, and set $x = \mathsf{false}$ next (unit resolution and conflict analysis are applied normally between these steps). Given such a sequence σ, $\mathsf{CLR}(\Delta, \sigma)$ refers to the SAT state attained after executing CLR on the CNF Δ according to the decisions and restarting points specified in σ (i.e., the choices on Lines 12 and 18 should be made based on the next element in the sequence).[4] For simplicity, we will insist that $\mathsf{CLR}(\Delta, \sigma)$ be a 1–consistent state (unless it contains the empty clause).[5] Moreover, we shall use the notation $\mathsf{CLR}(\Delta, \sigma)$ to also refer to the knowledge base (original and learned) of the SAT state $\mathsf{CLR}(\Delta, \sigma)$.

3.4 CLR as a Proof System

Modern clause-learning SAT solvers can be viewed as a proof system that contains all proofs obtainable by executing the solver according to some decision heuristic, restart policy, and implementation of unit resolution. If we view each conflict clause as being derived from a resolution proof, we can combine these

[2] Possibly at the expense of completeness.

[3] The implementation of unit resolution may affect, for example, the order and derivations of unit implications, which, in practice, influence which conflict clauses eventually get derived.

[4] A decision in σ should be skipped if its variable is already assigned a value.

[5] This only amounts to letting Algorithm 1 deal with conflicts until it reaches a 1–consistent state.

sub-derivations into a resolution proof that is produced by the SAT solver. In particular, if a given execution of CLR on an unsatisfiable problem produces conflict clauses $C_1, ..., C_k$, we know that $\Sigma = \Delta \wedge C_1 \wedge ... \wedge C_k$ is 1–inconsistent (this is how the algorithm terminates). Let each clause C_i be derived using resolution proof π_i (from original and previously learned clauses), and τ be the unit resolution derivation of false from Σ, then $\Pi = \pi_1, ..., \pi_k, \tau$ is the refutation proof generated by this execution. For the purpose of this work, each π_i can be viewed as a trivial resolution derivation, whose size is at most linear in the number of variables. We will later justify this claim in Proposition 3. In the following definition, which defines the proof system implemented by Algorithm 1, we overload the notation CLR to refer to both the SAT algorithm and the proof system.

Definition 5. *Given a learning scheme LS, proof system* CLR_{LS} *consists of all refutation proofs that can be generated by Algorithm 1 using learning scheme LS.*

The main result of this work will show that, for all asserting learning schemes LS, CLR_{LS} p-simulates general resolution. Note that the size of τ is always at most linear in the number of variables. Hence, we leave it out from our future discussion and proofs.

4 Ingredients for the Main Result

In this section, we present three key results that allow us to prove the main result. These results, some of which are interesting in their own rights, provide insights on the power of CLR. They are made possible by two important concepts, called *1–empowerment* and *1–provability*, which allow us to formalize the ability of CLR and use it to simulate general resolution. We first give definitions of these notions before presenting the results. The first notion is called 1–empowerment, which is the ability of a clause to allow unit resolution to see a new implication. We present here a slightly modified definition of the one presented in [13].

Definition 6 (1–Empowerment [13]). *Let $\alpha \Rightarrow \ell$ be a clause where ℓ is some literal in the clause and α is a conjunction of literals. The clause is 1–empowering with respect to CNF Δ iff*

1. $\Delta \models (\alpha \Rightarrow \ell)$*: the clause is implied by* Δ*.*
2. $\Delta \wedge \alpha$ *is 1–consistent: asserting* α *does not result in a conflict that is detectable by unit resolution.*
3. $\Delta \wedge \alpha \not\vdash \ell$*: the literal* ℓ *cannot be derived from* $\Delta \wedge \alpha$ *using unit resolution.*

In this case, ℓ is called an empowering literal of the clause. On the other hand, a clause implied by Δ that is not 1–empowering is said to be absorbed by Δ.[6]

A clause implied by Δ is 1–empowering if it allows unit resolution to derive a new implication that would be impossible to derive without the clause. For

[6] This terminology, "absorbed", was introduced in [1].

example, consider $\Delta = (a \vee b \vee c) \wedge (a \vee b \vee \neg c) \wedge (a \vee \neg b \vee c) \wedge (a \vee \neg b \vee \neg c) \wedge (\neg c \vee d) \wedge (c \vee e)$. The clause $(a \vee b)$ is 1–empowering with respect to Δ because unit resolution cannot derive a from $\Delta \wedge \neg b$. On the other hand, $(d \vee e)$, which is implied by Δ, is not 1–empowering (i.e., is absorbed), because unit resolution can already derive e from $\Delta \wedge \neg d$ and derive d from $\Delta \wedge \neg e$. Note that if clause C subsumes clause C' (i.e., $C \subseteq C'$), then C absorbes C'. Moreover, adding more clauses to the knowledge base may make a 1–empowering clause become absorbed but can never make an absorbed clause become 1–empowering. Every asserting clause is 1–empowering with respect to the knowledge base at the time of its derivation with its asserted literal as an empowering literal [13].

The second key notion, called 1–provability, is related to the difficulty of deriving a clause from a CNF.

Definition 7 (1–Provability). *Given a CNF Δ, clause C is 1–provable with respect to Δ iff $\Delta \wedge \neg C \vdash$ false.*

If a clause is 1–provable with respect to a given CNF, then we can show that it is implied by the CNF using only unit resolution. For example, consider Δ defined above. The clauses $(a \vee b)$ and (a) are both implied by Δ. In this case, the clause $(a \vee b)$ is 1–provable with respect to Δ, because unit resolution is sufficient to derive a contradiction after we assert the negation of the clause on top of Δ. However, this is not the case for (a) (thus, it is not 1–provable). Notice that, according to Definition 3, every conflict clause is 1–provable with respect to the knowledge base at the time of its derivation. Moreover, a 1–provable clause still remains 1–provable after any clause is added to the knowledge base.

We are now ready to present the results, whose proofs are in the Appendix. The first key result states that, in every refutation proof of a 1–consistent CNF, there is always a clause that is both 1–empowering and 1–provable with respect to the CNF.

Proposition 1. *Let Δ be an unsatisfiable CNF that is 1–consistent and Π be a refutation proof of Δ. There exists a clause $C \in \Pi$ such that C is 1–empowering and 1–provable with respect to Δ.*

The set of clauses which are both 1–empowering and 1–provable plays an important role in our main proof. In the next result, we show that CLR with any asserting learning scheme can absorb such clauses in a quadratic number of decisions. This result essentially states that, given any 1–empowering and 1–provable clause C, we can always come up with a (short) sequence of appropriate decisions (and restarts) to force CLR with an asserting learning scheme to efficiently derive clauses that, together with the original knowledge base, allow unit resolution to see any implication that C may allow us to derive (i.e., render C useless, as far as unit resolution is concerned).

Proposition 2. *Let Δ be a CNF with n variables and C be a clause that is 1–empowering and 1–provable with respect to Δ. For any asserting learning scheme*

AS, there exists an extended branching sequence σ that allows $\mathsf{CLR}_{AS}(\Delta, \sigma)$ to absorb C while producing resolution proof of size $O(n^4)$.[7]

Because of this result, we will call any clause that is both 1–empowering and 1–provable with respect to the given CNF Δ, CLR-*derivable* with respect to Δ.

The next result states that every 1–empowering conflict clause can always be derived using a trivial resolution derivation from the original and learned clauses at the time of the conflict.[8]

Proposition 3. *Let $S = (\Delta, \Gamma, D)$ be a 1–inconsistent SAT state and C be a conflict clause of S that is 1–empowering with respect to $\Delta \wedge \Gamma$. There exists a trivial resolution proof of some $C' \subseteq C$ from $\Delta \wedge \Gamma$.*

Since all resolved variables are distinct in a trivial resolution proof, the size of the proof has to be in $O(n)$, where n is the number of variables of Δ. This result is important as it shows that every empowering learning scheme (including any schemes yet to be conceived) can essentially be "implemented" with the kind of resolution derivation already employed by modern clause-learning SAT solvers. Since every asserting clause is 1–empowering, this result applies to all asserting learning schemes as well.

5 Main Result

In this section, we present our main result, which shows that the proof system implemented by modern clause-learning SAT solvers is as powerful as general resolution. We first present our main result in its most general form, then derive a corollary which is more closely related to modern SAT solvers.

Theorem 1. CLR *with any asserting learning scheme p-simulates general resolution.*

This result is applicable to a class of clause-learning algorithms that is even more general than what is used in practice (for example, it applies to any asserting learning scheme not yet proposed). By restricting the learning scheme, we obtain a more concrete result. Let 1stUIP denote the first UIP learning scheme [12, 17], which is, by far, the most popular scheme in practice.

Corollary 1. CLR_{1stUIP} *p-simulates general resolution.*

We will give an intuitive proof sketch for the main theorem before presenting the actual proof. In contrast to the proofs presented in [4], [9], and [5], we do

[7] The result given in Proposition 2 of the version that appeared in the proceedings of CP-09 contained an error on the size of the branching sequence. Here, we correct the problem and present a more direct result on the size of the proof produced instead.

[8] This result can be viewed as a variation on Proposition 4 of [4] for our definition of conflict clauses and 1–empowering clauses.

not try to simulate the derivation of *every* clause in the given resolution proof. Instead, we force CLR to go after CLR-derivable clauses only.

Suppose Δ has n variables. Let Π be a refutation proof of Δ and AS be an asserting learning scheme. If Δ is already 1–inconsistent the proof is trivial. Otherwise, we know that we can always find a CLR-derivable clause C in Π. We know that we can force CLR_{AS} to absorb C while producing a proof whose size is only polynomial in n. We can keep repeating this process until the knowledge base becomes 1–inconsistent. Since this can go on for at most $|\Pi|$ times, the combined proof is polynomial in $|\Pi|$ and n.

Proof of Theorem 1 Suppose AS is an asserting learning scheme. Given a CNF Δ with n variables and any refutation proof Π of Δ, we will construct an extended branching sequence σ that will induce CLR_{AS} to derive the empty clause and generate a proof of size $O(n^4|\Pi|)$.

In each iteration, consider $\Sigma = \Delta \wedge \Gamma$, the current knowledge base of CLR_{AS}. We may assume that Σ is 1–consistent. From Proposition 1, we can always find a clause C in Π that is CLR-derivable from Σ. Since we are using an asserting learning scheme, Proposition 2 tells us that the solver can absorb such a clause while generating a proof whose size is in $O(n^4)$. After absorbing C, we force the solver to restart. Let Σ now denote the updated knowledge base. Since Π is still a refutation proof of Σ, we can still find another CLR-derivable clause as long as Σ is still 1–consistent. We repeat this process until Σ is 1–inconsistent, at which point CLR terminates on Line 6 of Algorithm 1. In each iteration, we absorb at least one clause in Π (no absorbed clause can become 1–empowering after we add more clauses to the knowledge base). Hence, by Proposition 2, we know that each iteration produces a proof whose length is at most in $O(n^4)$. Therefore, we can use CLR_{AS} to produce a refutation proof of Δ with size $O(n^4|\Pi|)$. □

The next result shows that not only can we construct a CLR refutation proof with length polynomial in the size of any resolution proof, but the construction process can be carried out in polytime as well.

Theorem 2. *The extended branching sequence required for the simulation in Theorem 1 can be constructed in time polynomial in the sizes of the given refutation proof and of the given CNF.*

It suffices to show that finding a CLR-derivable clause in any given refutation proof (with respect to any 1–consistent CNF) can be done in polytime. For each clause C_i in the proof, checking if C_i is 1–provable only requires conditioning and closing the knowledge base under unit resolution. This can be achieved in time linear in the size of the CNF. Checking whether a literal is an empowering literal of a clause can be performed by asserting the negations of the other literals in the clause and see whether unit resolution can detect a conflict or derive the remaining literal from the knowledge base or not. This process, whose time complexity is linear in the size of the CNF, needs to be repeated for each literal in the clause. Therefore, the overall time complexity for finding a CLR-derivable clause is still polynomial in the sizes of the proof and of the CNF.

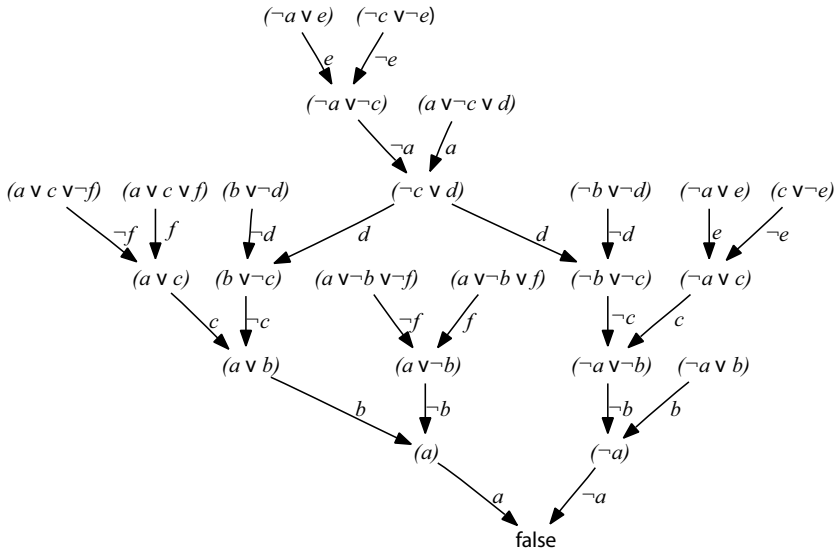

Fig. 1. A refutation proof of Δ. Each resolvent has two incoming edges from its resolved clauses (original clauses have no incoming edges). Each edge is annotated with the resolved literal of the corresponding resolved clause.

5.1 Example

We now show an example of the simulation described in the proof of Theorem 1. Consider the following unsatisfiable CNF:

$$\begin{aligned}\Delta = &(\neg a \vee e), (\neg c \vee \neg e), (a \vee \neg c \vee d), (\neg b \vee \neg d), (c \vee \neg e), (\neg a \vee b),\\ &(a \vee \neg b), (a \vee c \vee f), (a \vee c \vee \neg f), (b \vee \neg d), (a \vee \neg b \vee f), (a \vee \neg b \vee \neg f).\end{aligned}$$

Figure 1 shows a refutation proof of Δ. Alternatively, we can write this proof as

$$\begin{aligned}\Pi = &(\neg a \vee e), (\neg c \vee \neg e), (\neg a \vee \neg c), (a \vee \neg c \vee d), (\neg c \vee d),\\ &(\neg b \vee \neg d), (\neg b \vee \neg c), (c \vee \neg e), (\neg a \vee c), (\neg a \vee \neg b), (\neg a \vee b),\\ &(\neg a), (b \vee \neg d), (b \vee \neg c), (a \vee c \vee f), (a \vee c \vee \neg f), (a \vee c), (b \vee d),\\ &(a \vee \neg b \vee f), (a \vee \neg b \vee \neg f), (a \vee \neg b), (a), \mathsf{false}.\end{aligned}$$

Initially, one of the CLR-derivable clauses in Π is $(\neg b \vee \neg c)$ and $\neg c$ is the empowering literal. If the solver assigns $b = \mathsf{true}$ and then $c = \mathsf{true}$, unit resolution will detect a conflict. In this case, $(\neg b \vee \neg c)$ and $(\neg c \vee d)$ are both asserting clauses. Suppose $(\neg b \vee \neg c)$ is learned. After adding $(\neg b \vee \neg c)$ to the knowledge base, $(\neg c \vee d)$ and $(\neg b \vee \neg c)$, for example, become absorbed. Next, we force the solver to restart. Suppose we choose $(\neg a)$ as the next CLR-derivable clause to absorb. We must now force the solver to set $a = \mathsf{true}$, which will immediately

cause a conflict. In this case, $(\neg a)$ is derived. Once again, we force the solver to restart. Suppose we select $(a \vee \neg b)$ as the next CLR-derivable clause. We must now force the solver to set $a = \mathsf{false}, b = \mathsf{true}$. Since $\neg a$ is already implied by the last learned clause, the solver can skip the decision on a and only needs to assert $b = \mathsf{true}$ to cause a conflict. From this conflict, assume that the asserting clause $(a \vee \neg b)$ is learned. Adding this clause into the knowledge base will actually cause it to become 1–inconsistent. Hence, the solver can now terminate, as unit resolution can derive false from the set of original and learned clauses. The whole extended branching sequence used in this process is $\langle b, c, \mathbf{R}, a, \mathbf{R}, \neg a, b \rangle$.

6 Related Work

Early work in this direction was published by Beame *et al* in [4]. In that work, the authors showed that a slight variation of modern SAT solvers can simulate general resolution. However, one key modification required by the proof is to allow the solvers to make decisions on variables that are already assigned. This requirement essentially introduces another degree of freedom, which makes it harder to come up with a good decision heuristic and to actually implement in practice. It is interesting to note that the proof in [4] also requires the solver to restart at every conflict.

Van Gelder proposed a different proof system called POOL for studying modern SAT solvers as resolution engines [16]. In that work, the author focused on understanding the strength of POOL and using it to model modern SAT solvers. The author did not directly compare modern SAT solvers against general resolution.

Nevertheless, POOL later became a basis of the work by Hertel *et al* [9], which proved that modern SAT solvers can *effectively* p-simulate general resolution. In other words, the authors showed that, with an additional preprocessing step, modern solvers can become as strong as general resolution. While the preprocessing is deterministic and independent of the proof being simulated, it can be regarded as an extra component not utilized by any solver in practice.

Buss *et al* [5] also presented a similar argument. The authors showed that with a preprocessing step (different from the one in [9]), a generalized version of clause-learning algorithm can p-simulate general resolution. Apart from requiring an extra preprocessing step, the proof also needed the solver to make decisions on assigned variables.

7 Conclusions and Discussion

In this paper, we proved that modern clause-learning SAT solvers that utilize restarts correspond to a proof system that is as powerful as general resolution. Our work improves on previous results by avoiding the needs for additional degrees of non-determinism and preprocessing. Our proof is made possible by the notions of 1–empowerment and 1–provability, which allow us to capture the power of modern SAT solvers in a more direct and natural way, and to avoid

the need for any special technique. The result presented in this paper essentially shows that modern SAT solvers, as used in practice, are capable of simulating any resolution proof (given the right branching and restarting heuristics, of course).

Note that the our proof requires the solver to restart at every conflict. While no actual solver utilizes this particular restart policy, the proof suggests that a frequent restart policy might be a key to the efficiency of modern solvers. In our proof, restarting gives the solver the freedom to go after any clause necessary for a short refutation. Interestingly, in recent years, there has been a clear trend towards more and more frequent restarts for modern SAT solvers (e.g., [10], [**?**]).

In spite of our result, more theoretical work still remains to be done in this research direction. The construction of our proof requires the solver to backtrack to the top level upon each conflict (i.e., restart). While it is easy to implement such a strategy, in practice, state-of-the-art solvers only backtrack to the assertion level at each conflict (this type of backtracking is termed far-backtracking in [15]). It still remains an open question whether far-backtracking (or even chronological backtracking) is sufficient to achieve the presented result.

A Proofs

In this appendix, we present proofs of Propositions 1, 2, 3. The proof of Proposition 1 is accomplished with the help of two lemmas.

Lemma 1. *Let $C_1 = \alpha \vee \ell$, $C_2 = \beta \vee \neg\ell$. Suppose C_1 and C_2 are not 1–empowering with respect to Δ. Then, $\alpha \vee \beta$ is 1–provable with respect to Δ.*

Proof Let $C = \alpha \vee \beta$. Since both C_1, C_2 are not 1–empowering, we know that $\Delta \wedge \neg\alpha \vdash \ell$ and $\Delta \wedge \neg\beta \vdash \neg\ell$.[9] Therefore, unit resolution must be able to derive both ℓ and $\neg\ell$ from $\Delta \wedge \neg C = \Delta \wedge \neg\alpha \wedge \neg\beta$. Therefore, C is 1–provable with respect to Δ. □

Lemma 2. *Let C be a clause that is not 1–provable with respect to Δ and Π be a resolution proof of C from Δ. Then, there exists $C' \in \Pi$, such that C' is* CLR*-derivable from Δ.*

Proof Let $\Pi = C_1, ..., C_n$ and C_i be the first clause in Π that is not 1–provable with respect to Δ (i may be equal to n). Clearly, C_i must be the resolvent of two 1–provable clauses C_j, C_k for some $j, k < i$. Assume for the sake of contradiction that C_j, C_k are both not 1–empowering. Lemma 1 implies that C_i must be 1–provable, which is a contradiction. Hence, either C_j or C_k must be both 1–provable and 1–empowering. □

Proof of Proposition 1 Given a 1–consistent CNF Δ, it is easy to see that the empty clause (false) is not 1–provable with respect to Δ (otherwise, Δ would be 1–inconsistent by definition). Since every refutation proof contains the empty

[9] It is also possible that asserting $\neg\alpha$ or $\neg\beta$ may result in a 1–inconsistent CNF. We omit this case as C is trivially 1–provable.

clause, Lemma 2 implies that the proof Π must contain a clause that is both 1–empowering and 1–provable. □

Next, we present the proof of Proposition 2.

Proof of Proposition 2 Let $C = (\alpha \vee \ell)$ be the clause under consideration and ℓ be an empowering literal. Moreover, let δ be an extended branching sequence consisting of the literals in $\neg\alpha$ in any order. Since C is 1–empowering, the SAT state right after asserting δ must be 1–consistent ($\star$). Moreover, because C is 1–empowering and 1–provable, at this point, neither ℓ nor $\neg\ell$ can be implied by unit resolution.[10] Hence, CLR can select $\neg\ell$ as the next decision. The 1–provability of C ensures that asserting δ together with $\neg\ell$ will result in a 1–inconsistent state S ($\star\star$).

Let D be the asserting clause derived by AS from S. If $\Delta \wedge D$ absorbs C, we are done. Otherwise, we add D to the knowledge base and let the solver take care of any conflict until the SAT state on Line 4 of Algorithm 1 becomes 1–consistent. There are at most n conflicts that the solver may need to go through here, because the solver has to undo at least one decision every conflict and we are not making any new decision. Hence, by Proposition 3, the total size of resolution proof produced is in $O(n^2)$. Once the solver is in a 1–consistent state, we restart, and repeat this whole process. We will now argue that this can only be repeated $O(n)$ times.

Every asserting clause learned in the process must generate at least one new implication (i.e., its asserted literal) under a subset of δ. In every iteration, at least one more literal that was implied at the conflict level will now be implied by the time that δ is asserted. Because of ($\star$) and ($\star\star$), which hold in every iteration, a conflict only happens after $\neg\ell$ is asserted (thus, after δ is asserted). This implies that, in our proof, once a literal becomes an asserted literal of some asserting clause, it will never be assigned at any future conflict level again. Thus, each literal can only become an asserted literal (of any clause) only once in this whole process.

Since there are only n variables, this can be repeated at most $O(n)$ times before the empowering literal (ℓ) itself is implied by the assertion of some asserting clause. Whenever that happens, it means that ℓ no longer is an empowering literal of C. At this point, the proof produced so far has size in $O(n^3)$. In order to completely absorb C, we need to repeat this whole process for each empowering literal of C. Since there are $\leq n$ literals in any clause, the total size of the proof is clearly in $O(n^4)$. □

Next we prove Proposition 3.

Proof of Proposition 3 Let $C = (\alpha \vee \ell)$ and ℓ be its empowering literal. Let σ be a branching sequence consisting of the literals of $\neg\alpha$ (in any order) followed by $\neg\ell$. Let DEC be the *decision learning scheme* (as defined based on implication

[10] That ℓ is not implied is straightforward. If $\neg\ell$ was implied, the current state would be 1–inconsistent, because C is 1–provable. This contradicts ($\star$).

graph in Section 2 of [17]). In this scheme, conflict clauses contain only literals of decision variables.

Asserting σ will result in a conflict and DEC will derive an asserting clause C' which consists entirely of the negations of decision literals. Since the decisions in δ are all negations of literals in C, we have $C' \subseteq C$. Since DEC is a learning scheme based on implication graph, Proposition 4 of [4] implies that every conflict clause produced by it, C' in particular, can be derived using a trivial resolution proof. □

References

1. Atserias, A., Fichte, J. K., and Thurley, M. Clause-learning algorithms with many restarts and bounded-width resolution. In *Proceedings of the Twelfth International Conference on Theory and Applications of Satisfiability Testing (SAT'09)* (2009), pp. 114–127.
2. Audemard, G., Bordeaux, L., Hamadi, Y., Jabbour, S., and Sais, L. A generalized framework for conflict analysis. In *SAT'08* (2008), pp. 21–27.
3. Bayardo, R. J. J., and Schrag, R. C. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of AAAI'97* (1997), pp. 203–208.
4. Beame, P., Kautz, H., and Sabharwal, A. Towards understanding and harnessing the potential of clause learning. *JAIR 22* (2004), 319–351.
5. Buss, S. R., Hoffmann, J., and Johannsen, J. Resolution trees with lemmas: Resolution refinements that characterize DLL algorithms with clause learning. *Logical Methods in Computer Science 4*, 4 (2008).
6. Cook, S. A., and Reckhow, R. A. The relative efficiency of propositional proof systems. *J. Symb. Log. 44*, 1 (1979), 36–50.
7. Davis, M., Logemann, G., and Loveland, D. A machine program for theorem-proving. *Commun. ACM 5*, 7 (1962), 394–397.
8. Gomes, C. P., Selman, B., and Crato, N. Heavy-tailed distributions in combinatorial search. In *Proceedings of CP-97* (1997), pp. 121–135.
9. Hertel, P., Bacchus, F., Pitassi, T., and Van Gelder, A. Clause learning can effectively p-simulate general propositional resolution. In *Proc. of AAAI-08* (2008), pp. 283–290.
10. Huang, J. The effect of restarts on the efficiency of clause learning. In *Proc. of IJCAI-07* (2007), pp. 2318–2323.
11. Marques-Silva, J. P., and Sakallah, K. A. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of ICCAD'96* (1996), pp. 220–227.
12. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., and Malik, S. Chaff: Engineering an efficient sat solver. In *Proceedings of DAC'01, June 2001.* (2001).
13. Pipatsrisawat, K., and Darwiche, A. A new clause learning scheme for efficient unsatisfiability proofs. In *Proceedings of AAAI-08* (2008), pp. 1481–1484.
14. Robinson, J. A. A machine-oriented logic based on the resolution principle. *J. ACM 12*, 1 (1965), 23–41.
15. Sang, T., Beame, P., and Kautz, H. Heuristics for fast exact model counting. In *SAT'05* (2005), pp. 226–240.
16. Van Gelder, A. Pool resolution and its relation to regular resolution and dpll with clause learning. In *LPAR'05* (2005), pp. 580–594.
17. Zhang, L., Madigan, C. F., Moskewicz, M. W., and Malik, S. Efficient conflict driven learning in boolean satisfiability solver. In *ICCAD'01* (2001), pp. 279–285.