On the Power of Clause-Learning SAT Solvers as Resolution Engines[☆]

Knot Pipatsrisawat[*,1], Adnan Darwiche

Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095 USA

Abstract

In this work, we improve on existing results on the relationship between proof systems obtained from conflict-driven clause-learning 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 conflict-driven clause-learning SAT solvers corresponded to proof systems as powerful as general resolution. However, the models used in these studies required either an extra degree of non-determinism or a preprocessing step that is not utilized by state-of-the-art SAT solvers in practice. In this paper, we prove that conflict-driven clause-learning SAT solvers yield proof systems that indeed p-simulate general resolution without the need for any additional techniques. Moreover, we show that our result can be generalized to certain other practical variations of the solvers, which are based on different learning schemes and restart policies.

Key words: Boolean satisfiability, clause-learning SAT solvers, DPLL, proof complexity, resolution proof

[☆]This work extends our previous work in [25] with generalized results (Section 6) and additional discussion about related work (Section 7).

[*]Corresponding author

Email addresses: thammakn@cs.ucla.edu (Knot Pipatsrisawat), darwiche@cs.ucla.edu (Adnan Darwiche)

[1]Univesrity of California, Los Angeles, 4801 Boelter Hall, Los Angeles, CA 90095 USA. Tel. 1-310-794-4343, Fax. 1-310-794-5057.

1. Introduction

It is well-known that conflict-driven clause-learning (CDCL) SAT solvers (e.g., [27, 21, 11]) and their original ancestor, the DPLL algorithm [9], can be interpreted as resolution-based proof systems [5]. For each unsatisfiable formula, these solvers can be viewed as engines that produce refutation proofs. One central question in this research direction is whether proof systems implemented by CDCL SAT solvers have enough freedom to generate a proof that is at most polynomially longer than the shortest general resolution proof for every unsatisfiable formula. The answer to this theoretical question could have important practical implications on the efficiency of CDCL SAT solvers.

There is much previous work on this subject that demonstrates the strength of variations on CDCL SAT solvers. However, equivalence with respect to general resolution has yet to be proven for CDCL SAT solvers as they are practiced today. In [5], Beame *et al* showed that a proof system based on a more general variation of CDCL SAT solvers was as powerful as general resolution. The proof presented, however, requires the solver to make decisions to negate the values of some 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" [15]. Hertel *et al* [15] 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 [15] is based on the one developed by van Gelder in [29]). Buss *et al* [7] also took a similar approach by modifying the input CNF and introducing some new variables to show that a generalized variation of the clause-learning algorithm, which allows decision making past conflicts, can effectively p-simulate general resolution.

In this work, we show that CDCL SAT solvers, without any extra modifications, can be as powerful as general resolution, given the right heuristics. In particular, we prove that proof systems implemented by CDCL SAT solvers, which utilize unit resolution, clause learning, and restarting, p-simulate general resolution. First, we show that this result holds for any solvers utilizing an asserting clause-learning scheme. Then, we generalize our main result to show that certain variations of CDCL SAT solvers also correspond to proof systems that are as powerful as general resolution. In particular, we identify a class of non-asserting clause-learning schemes and a class of restart policies that still allow the solvers to achieve their full potential. Our proofs do not utilize a preprocessing step or require the solvers to make decisions on an implied literal or past a conflict. This result implies that CDCL SAT solvers in their current form and some of its variations 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* [24] and *1–provability*, which we will define later in this paper. Together, they allow us to more accurately capture the power of CDCL SAT solvers. In particular, we use these two notions to characterize the set of clauses that CDCL SAT solvers can learn. Contrary to previous attempts in [5, 15, 7], we do not force the solver to simulate the derivation of every single clause in the given refutation proof (which may not be possible without some modifications to the solvers). Instead, we guide the solver to go after clauses that satisfy these properties, thus avoiding the need to introduce such modifications.

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 CDCL SAT solvers and a proof system associated with it. Then, in Section 4, we present some results, which provide insights into the power of CDCL SAT solvers and allow us to prove the main result. Next, in Section 5, we present our main result, which applies to any clause-learning SAT solvers that learn asserting clauses. In Section 6, we generalize the main result to cover certain non-asserting learning schemes and restart policies that have been studied in the past. Finally, we discuss related work in Section 7 and conclude in Section 8.

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 $\Delta \vdash_1 \ell$ 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 [8]. In this work, we are concerned only with proof systems based on propositional resolution [26]. 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 practical SAT solvers as possible, the *weakening rule* (see usage in [6], for example), 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*. For the purposes of this work, a proof system can be viewed as the set containing all refutation proofs allowed by the system. The notion of p-simulation, which was introduced in [8], is used to compare the power of two proof systems. The definition presented here is obtained from [15].

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 in other work (such as [5] and [15]), we will relax the notion of p-simulation to allow the size of the simulating proof to depend on the size of the input formula. This relaxation is necessary if one wishes to relate any theoretical result to practical CDCL SAT Solvers, because every such solver must read the input formula and, therefore, cannot solve any problem in time less than linear in the size of the input. Note that if we only consider CNF formulas whose shortest refutation proof is at least linear in the size of the formula, then notion of p-simulation used here essentially becomes identical to the original definition of p-simulation [8].

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 [5] for a review).

3. Conflict-Driven Clause-Learning SAT Solvers as Proof Systems

3.1. Conflict-Driven Clause-Learning SAT Solvers

In this section, we describe a model of CDCL SAT solvers. Included in our model are the following techniques: unit resolution [9], clause-learning [20, 30], restarting [14], and non-chronological backtracking [20, 4] (i.e., far-backtracking as termed by [28]). Algorithm 1 shows a pseudo code of a typical conflict-driven clause-learning SAT solver with restarts, which we will refer to as CLR (for Clause Learning with Restarts) 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 CNF. 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 8-11).[2]

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 14). 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 21). If no such literal is found, the algorithms terminates having proved satisfiability (Line 19). 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 $\langle \ell_1, \ldots, \ell_m \rangle$ (with $D_0 = \langle \rangle$).
- A CLR *state* S 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 CLR state (Δ, Γ, D_k).

[2]The clause α is known as a conflict clause and m as the backtrack level. We will define them formally later.

```
input  : CNF formula $\Delta$
output: A solution of $\Delta$ or unsat if $\Delta$ is not satisfiable
$D \leftarrow \langle\rangle$ // decision literals
$\Gamma \leftarrow$ true // learned clauses
while true do
    if $S = (\Delta, \Gamma, D)$ is 1–inconsistent then
        // there is a conflict.
        if $D = \langle\rangle$ then
            return unsat
        end
        $\alpha \leftarrow$ a conflict clause of $S$
        $m \leftarrow$ the assertion level of $\alpha$
        $D \leftarrow D_m$ // the first $m$ decisions
        $\Gamma \leftarrow \Gamma \wedge \alpha$
    else
        // there is no conflict.
        if time to restart then
            $D \leftarrow \langle\rangle$
            $S \leftarrow (\Delta, \Gamma, D)$
        end
        Choose a literal $\ell$ such that $S \nvdash_1 \ell$ and $S \nvdash_1 \neg\ell$
        if $\ell =$ null then
            return $D$ // satisfiable
        end
        $D \leftarrow D, \ell$
    end
end
```
Algorithm 1: CLR: Clause-learning SAT solver with restarts.

- A CNF Δ is *1–inconsistent* iff $\Delta \vdash_1 \mathsf{false}$. It is *1–consistent* otherwise. A CLR 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.

- Given a CLR state $S = (\Delta, \Gamma, D)$, we write $S \vdash_1 \ell$ if $\Delta \wedge \Gamma \wedge D \vdash_1 \ell$ and write $S \nvdash_1 \ell$ if $\Delta \wedge \Gamma \wedge D \nvdash_1 \ell$. A literal ℓ is *implied by state* $S = (\Delta, \Gamma, D)$ *at level* k iff k is the smallest integer for which $S_k \vdash_1 \ell$. We say that the *implication level* of literals ℓ and $\neg\ell$ is k in this case.

- 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, $S_{i-1} \nvdash_1 \ell_i$ and $S_{i-1} \nvdash_1 \neg\ell_i$.

The notion of normal states prohibits CLR 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 15 of Algorithm 1 is always normal. Therefore, from now on, we will assume that every CLR state is normal.

We are now ready to define the last two notions used in Algorithm 1: conflict clause and assertion level. Conventionally, the notion of conflict clause is usually defined in terms of a cut in an implication graph [30, 5]. An *implication graph* is a graph that captures the implication relationship between current variable assignments. Each node in the graph corresponds to a literal currently assigned to true (by decision or by unit resolution). A directed edge from ℓ' to ℓ exists iff $\neg\ell'$ appears in the unit clause that implies ℓ (i.e., the assignment $\ell' = \mathsf{true}$ directly contributes to the implication $\ell = \mathsf{true}$). For a more complete definition of implication graph, see [30], for example.

The results that we will present in this work do not depend on the notion of implication graph, which is just one way of deriving conflict clauses. In what follows, we present a definition of conflict clauses in a more general sense. This definition focuses on the logical semantics of conflict clause and closely follows the graphical definition in [30] and [5].

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

1. $\Delta \wedge \Gamma \wedge \neg\alpha \vdash_1 \mathsf{false}$. *That is, we can show that α is implied by $\Delta \wedge \Gamma$ using just unit resolution.*
2. *For each literal ℓ_i, $S \vdash_1 \neg\ell_i$. That is, each literal $\neg\ell_i$ is a decision or an implication discovered by unit resolution in state S.*

The *assertion level* of a conflict clause is defined to be the second largest implication level of any literal in the clause with respect to the 1–inconsistent CLR state at the time the clause is derived. If every literal in the clause has the same implication level, the assertion level is defined to be zero.

In [5], 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 CDCL SAT solvers [5]. 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. We present this claim formally in Proposition 4.

Every 1–inconsistent CLR state has at least one conflict clause associated with it. The following proposition states this result, which ensures that the execution of Line 8 of Algorithm 1 will always succeed.

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

Proof of Proposition 1 ($\rightarrow$) If $S = (\Delta, \Gamma, D)$ is 1–inconsistent then the clause $\neg D$ (the negations of the current decisions) is a conflict clause for S. ($\leftarrow$) Let a conflict clause C of $S = (\Delta, \Gamma, D)$ be given. By the definition of conflict clause, $\Delta \wedge \Gamma \wedge D \vdash_1 \neg C$. Moreover, by the same definition, we have $\Delta \wedge \Gamma \wedge \neg C \vdash_1 \mathsf{false}$. Hence, S is 1–inconsistent. $\square$

This completes our description of CLR.

3.2. Clause-Learning Schemes

CLR can employ various *learning schemes* to derive conflict clauses (Line 8). In our context, it is sufficient to view a learning scheme as a function that produces a conflict clause for every 1–inconsistent CLR state. While Algorithm 1 allows any learning scheme to be used, in practice, CDCL SAT solvers 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* CLR *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.*

One nice property of asserting clauses is that they are guaranteed to produce new implications as soon as the solver backtracks to their assertion levels. In the following sections (Sections 4, 5), we will first present results based on a certain class of learning schemes that always produce asserting clauses. We refer to these learning schemes as *asserting learning schemes*. Later, in Section 6, we will discuss generalized results, which apply to some non-asserting learning schemes as well.

Given a learning scheme LS, we use CLR_{LS} to denote the SAT algorithm obtained by applying LS on Line 8 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 the branching heuristic and the restart policy.

In this work, we utilize the notion of extended branching sequence (introduced in [5]) to capture these two sources of non-determinism. An *extended branching sequence* (or just branching sequence, for short) 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 $x =$ true, the second decision should be $y =$ false, then the solver should restart, and set $x =$ false next (unit resolution and conflict analysis are applied normally between these steps). Given such a sequence σ, CLR(Δ, σ) refers to the CLR 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 13 and 21 should be made based on the next element in the sequence). While not necessary, we assume for simplicity that the solver skips any decision in σ if its variable is already set to the value of the decision. This is different from branching on assigned variables (as done in [5]), which allows the solver to set variables to values opposite to what have already been assigned.

After CLR makes some decisions, its state may be 1–inconsistent (Line 4). In this case, the algorithm will keep executing Lines 5-11 until its state becomes 1–consistent, at which point it is ready to make another decision or restart. For simplicity, we insist that CLR(Δ, σ) always refer to the state at the end of this process, which is 1–consistent (unless the knowledge base already contains the empty clause). Moreover, we use the notation KB($\mathsf{CLR}(\Delta, \sigma)$) to refer to the knowledge base (original and learned clauses) of the CLR state $\mathsf{CLR}(\Delta, \sigma)$.

3.4. CLR *as a Proof System*

Conflict-driven clause-learning SAT solvers can be viewed as a proof system that contains all refutation proofs obtainable by executing the solvers according to some decision heuristic and restart policy. If we view each conflict clause as the result of a resolution derivation, we can combine these derivations into a refutation proof once the solvers finish solving an unsatisfiable problem. 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 derivation 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 CNF variables. We will later justify this claim in Proposition 4. In the following definition, which defines a 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 and 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 the unit refutation τ at the end of the proof (see above) is always at most linear in the number of variables. Since this part of the refutation proof can only contribute a linear factor and in general will be much smaller than the rest of the proof (i.e., the derivations of learned clauses), we omit it from our future discussion and proofs for simplicity.

To help formalize our discussion later, we present a definition of a clause-learning resolution derivation induced by an extended branching sequence.

Definition 6. *Let $C_1, C_2, \ldots, C_k$ be the conflict clauses produced by executing CLR_{LS} on CNF Δ according to branching sequence σ, and let S^i be the 1–inconsistent CLR state during the execution for which C_i is a conflict clause. Moreover, let Π_i be a trivial resolution derivation of clause C_i from $KB(S^i)$ (the knowledge base at the time C_i was derived), then we say that σ <u>induces</u> a resolution derivation DER$_{LS}$(Δ, σ) = $\Pi_1, \Pi_2, ..., \Pi_k$.*

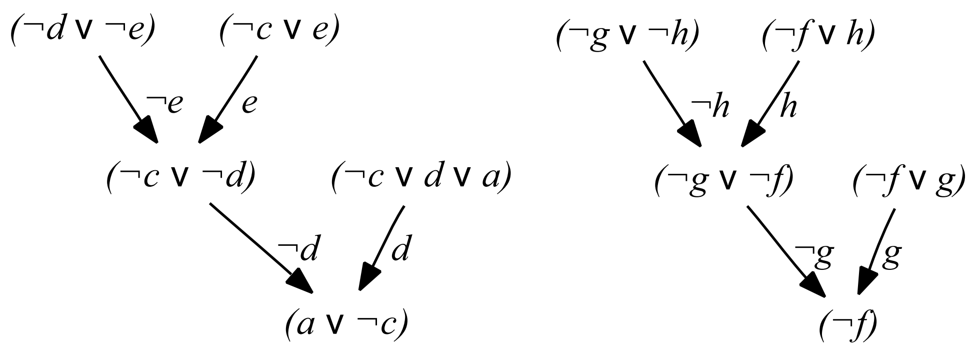

Figure 1: (Left) A trivial resolution derivations of $(a \vee \neg c)$. (Right) A trivial resolution derivation of $(\neg f)$.

This notion captures the resolution proof produced, by CDCL SAT solvers, as a result of executing a certain branching sequence. Note that, according to this definition, the derivation induced by a branching sequence may not form one connected resolution derivation. To give an example, consider the following CNF:

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

and the branching sequence $\sigma = \langle \neg a, \neg b, \neg e \rangle$. Let $1^{st}UIP$ be the first UIP learning scheme [30]. We will now show the resolution derivation produced by executing $\mathsf{CLR}_{1^{st}UIP}$ on Δ according to σ.

The solver begins by setting $a = \mathsf{false}$, followed by $b = \mathsf{false}$. At this point, unit resolution will be able to detect a conflict (using the clauses on the first line) and $\mathsf{CLR}_{1^{st}UIP}$ will derive $(a \vee \neg c)$ as the conflict clause. Figure 1 (left) shows a trivial resolution derivation of this clause.

After that, the solver will undo $b = \mathsf{true}$ and assert this clause. According to σ, the solver now sets $e = \mathsf{false}$ as the next decision. This decision will lead to another conflict (using the clauses on the second row). From this, $1^{st}UIP$ will derive $(\neg f)$ as the conflict clause. Figure 1 (right) shows a trivial resolution derivation of this clause. Overall, the resolution derivation produced is

$$\begin{aligned}DER_{1^{st}UIP}(\Delta, \sigma) = &(\neg d \vee \neg e), (\neg c \vee e), (\neg c \vee \neg d), (\neg c \vee d \vee a), (a \vee \neg c),\\ &(\neg g \vee \neg h), (\neg f \vee h), (\neg g \vee \neg f), (\neg f \vee g), (\neg f).\end{aligned}$$

4. Ingredients for the Main Result

In this section, we present three results that will be utilized in the proof of the main result. These results, which provide insights on the power of CLR, are made possible by two important concepts: *1–empowerment* and *1–provability*. These concepts allow us to formalize the ability of CLR to simulate general resolution. We first give definitions of these notions before presenting the results. The first notion, 1–empowerment, 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 [24].

Definition 7 (1–Empowerment [24]). *Let $\alpha \Rightarrow \ell$ be a clause where ℓ is some literal 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. $\Delta \wedge \alpha \not\vdash_1 \ell$*: 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 contains no empowering literal is said to be absorbed by* Δ*.*[3]

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 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' is absorbed by 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 [24].

The second key notion, called 1–provability, is related to the difficulty of showing that a clause is entailed by a CNF.

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

Definition 8 (1–Provability). *Given a CNF Δ, clause C is* <u>*1–provable*</u> *with respect to Δ iff $\Delta \wedge \neg C \vdash_1 \mathsf{false}$.*

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, the clause (a) is not 1–provable with respect to Δ. 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.[4]

We are now ready to present the results. 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 2. *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 proof of this proposition is accomplished with the help of the following two lemmas.

Lemma 1. *Let $C_1 = \alpha \vee \ell$, $C_2 = \beta \vee \neg \ell$ be two clauses implied by Δ. 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 $C = \alpha \vee \beta$. If asserting $\neg\alpha$ or $\neg\beta$ result in a 1–inconsistent CNF, C must be trivially 1–provable. Otherwise, since both C_1, C_2 are implied by Δ and are not 1–empowering, we know that $\Delta \wedge \neg\alpha \vdash_1 \ell$ and $\Delta \wedge \neg\beta \vdash_1 \neg\ell$. 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. *Every resolution derivation of a clause that is not 1–provable with respect to Δ from Δ must contain a clause which is both 1–provable and 1–empowering with respect to Δ.*

[4]The concept of 1–provability is not new as it is indeed the underlying principle of the resolution proof verification method proposed in [13].

Proof Let $\Pi = C_1, ..., C_n$ be the derivation 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. $\square$

Proof of Proposition 2 The empty clause is not 1–provable with respect to Δ, because Δ is 1–consistent. Lemma 2 implies that the refutation proof Π, which derives the empty clause, must contain a clause that is both 1–empowering and 1–provable. $\square$

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 while producing a resolution proof of polynomial size. This result essentially states that, given any 1–empowering and 1–provable clause C, we can 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 3. *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 such that the following holds:*

1. $KB(\mathsf{CLR}_{AS}(\Delta, \sigma))$ *absorbs* C
2. $|DER_{AS}(\Delta, \sigma)| \in O(n^4)$

Proof of Proposition 3 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 CLR state right after asserting δ must be 1–consistent. At this point, because C is 1–empowering, ℓ must not be implied by unit resolution. Moreover, $\neg\ell$ cannot be implied by unit resolution either, otherwise the current state must be 1–inconsistent (because C is 1–provable). This would be a contradiction to the fact that C is 1–empowering. Hence, CLR can select $\neg\ell$ as the next decision. Asserting δ and $\neg\ell$ falsifies C by construction. Since C is 1–provable with respect to the current knowledge base, these assertions must result in a 1–inconsistent CLR state S.

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 CLR 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 every time a conflict is dealt with, the algorithm backtracks by at least one level. Hence, by Proposition 4, 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 C remains 1–empowering and 1–provable 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 $O(n^4)$. $\square$

This result shows that any 1–empowering and 1–provable clause can be efficiently absorbed by CLR with asserting clauses. So, we will call such a clause CLR-*easy* 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.

Proposition 4. *Let $S = (\Delta, \Gamma, D)$ be a 1–inconsistent* CLR *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 $\Delta \wedge \Gamma$.*

Proof of Proposition 4 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 graph in Section 2 of [30]). 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 [5] implies that every conflict clause produced by it, C' in particular, can be derived using a trivial resolution proof. $\square$

This result can be viewed as a variation of Proposition 4 of [5] for our definition of conflict clauses and 1–empowering clauses. 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 learning scheme that always derives 1–empowering clauses (including any schemes yet to be conceived) can essentially be "implemented" with the kind of resolution derivation already employed by CDCL 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 CDCL SAT solvers p-simulates general resolution.

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

In practice, the *first UIP* learning scheme [21, 30] (denoted $1^{st}UIP$ below), is by far the most popular asserting learning scheme.

Corollary 1. $\mathsf{CLR}_{1^{st}UIP}$ *p-simulates general resolution.*

We now present the proof of Theorem 1. In contrast to the proofs presented in [5], [15], and [7], we do not try to simulate the derivation of *every* clause in the given resolution proof. Instead, we force CLR to go after CLR-easy clauses only.

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 2, we can always find a clause C in Π that is CLR-easy with respect to Σ. Since we are using an asserting learning scheme, Proposition 3 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-easy 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 3, 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 $O(n^4|\Pi|)$. $\square$

The next result shows that not only can we construct a CLR refutation proof with size 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-easy 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 asserting $\neg C_i$ 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-easy clause is still polynomial in the sizes of the proof and of the CNF.

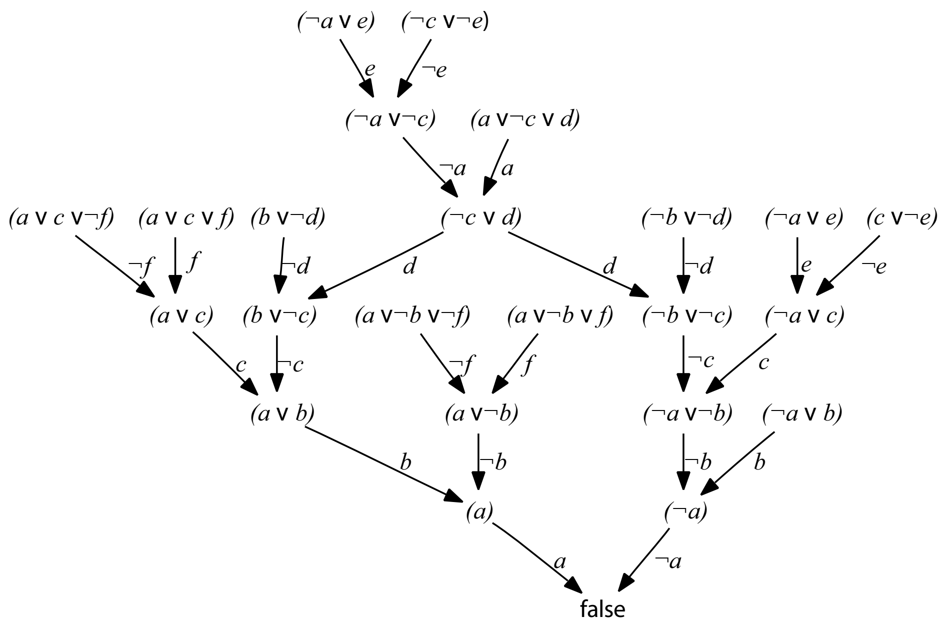

Figure 2: A refutation proof of Δ. Each resolvent has two incoming edges from its resolved clauses (original clauses have no incoming edges). Each edge is annotated with the resolved literal of the corresponding resolved clause.

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

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

Initially, one of the CLR-easy clauses in Π is $(\neg b \vee \neg c)$ and $\neg c$ is the empowering literal. If the solver assigns $b = \textsf{true}$ and then $c = \textsf{true}$, 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-easy 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-easy 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. Generalization of the Main Result

In this section, we generalize our main result to some variations of clause-learning SAT solvers that have been studied in the past. In particular, we show that the main result stated in Theorem 1 also holds for certain non-asserting learning schemes and for restart policies with a certain property. This demonstrates that our result is applicable to a wide range of practical SAT algorithms.

6.1. Beyond Asserting Clauses

The proof of Theorem 1 reveals that 1–empowerment is a key to the power of CLR. Yet, the main result presented in the previous section requires that the clauses learned by the solvers always be asserting (every asserting clause is also 1–empowering). One obvious question that could be asked is whether any other 1–empowering learning scheme will be sufficient for CLR to achieve its full potentials. In this section, we will show that, indeed, a certain class of 1–empowering learning schemes also allows CLR to p-simulate general resolution. We first define this class of learning schemes then present the result.

Definition 9. *Let α and β be disjunctions of zero or more literals. The conflict clause $C = (\alpha \vee \beta \vee \ell)$ of a CLR state $S = (\Delta, \Gamma, D)$ is $\underline{k\text{–asserting}}$ if there are exactly k literals $(\beta \vee \ell)$ falsified at level $|D|$ and $\Delta \wedge \Gamma \wedge \overline{D_{|D|-1}} \wedge \neg\beta \nvdash_1 \ell$.*

In other words, a k–asserting clause contains exactly k literals falsified at the last level. Moreover, the clause is capable of producing a new implication (ℓ) under the set of decisions right before the conflict. Notice that the second condition implies that a k–asserting clause is also 1–empowering, because, by the definition of conflict clauses, $\neg\alpha$ must be implied from $\Delta \wedge \Gamma \wedge D_{|D|-1}$ (using unit resolution). Moreover, a literal (ℓ) falsified at the last level is an empowering literal of the clause. Note that the second condition above could also be interpreted as follows. Under the assignment $D_{|D|-1}$, any k–asserting clause reduces (i.e., simplifies) to a clause $(\beta \vee \ell)$ that is 1–empowering with respect to $\Delta \wedge \Gamma \wedge D_{|D|-1}$. The notion of k–asserting clause generalizes the notion of asserting clause. In the case of asserting clause β is the empty clause.

Definition 10. *A learning scheme LS is k–bounded-asserting if, for every 1–inconsistent* CLR *state S, there exists $i \leq k$ such that LS produces an i–asserting clause for S.*

Essentially, a k–bounded-asserting learning scheme always learns an i–asserting clause, where $i \leq k$, for every conflict. Clearly, every asserting learning scheme is 1–bounded-asserting. In [24], a learning scheme called *bi-asserting clause* was proposed and showed to yield much smaller conflict clauses, which significantly improved performance of the solver on unsatisfiable formulas. This learning scheme could be viewed as a practical variation of the 2–bounded-asserting learning scheme.

In order to prove the result which generalizes Theorem 1, we need a generalized version of Proposition 3.

Proposition 5. *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 k–bounded-asserting learning scheme LS, there exists an extended branching sequence σ such that the following holds:*

1. $KB(\mathsf{CLR}_{LS}(\Delta, \sigma))$ *absorbs* C
2. $|DER_{LS}(\Delta, \sigma)| \in O(n^{k+3})$

Proof of Proposition 5 Let $C = (\alpha \vee \ell)$ be the clause in question and ℓ be an empowering literal. The proof has the same structure as the proof of Proposition 3. First, we construct an extended branching sequence δ from the negation of the clause C in the same manner. As soon as δ and $\neg\ell$ are asserted, the solver must be in a 1–inconsistent state.

Let D be the k–asserting clause derived by the learning scheme. If $\Delta \wedge D$ absorbs C, we are done. Otherwise, we add D to the knowledge base. As in the proof of Proposition 3, the solver will go through at most n conflicts to get into a 1–consistent state and produce resolution proof of size $O(n^2)$ here. Once it is in a 1–consistent state, we restart, and repeat this whole process. Now, instead of arguing that this process can only be repeated $O(n)$ times (as in the proof of Proposition 3), we will show that it can go on for at most $O(n^k)$ times.

Since every clause learned during this process is k–asserting, it must simplify to a 1–empowering clause C' under the decision sequence δ (by definition). This implies that each of the clauses learned during this process must reduce to a distinct sub-clause once δ is asserted, otherwise the clause that was learned later cannot be 1–empowering with respect to the existing knowledge base.

This key observation implies that there can only be so many clauses that we can learn during this process. In particular, the number of different clauses learned is at most the number of distinct clauses of size $\leq k$. Therefore, this process can be repeated at most $O(n^k)$ times. Clearly, the size of the proof produced so far is in $O(n^{k+2})$. At this point, we know that ℓ can no longer be an empowering literal of C (in the current knowledge base). In order to completely absorb C, we need to repeat this procedure for each empowering literal of C. Since there are $\leq n$ literals in C, the total proof size is in $O(n^{k+3})$. □

Again, when $k = 1$, the result in this proposition become identical to that of Proposition 3. With this result, we are now ready show that CLR with any k–bounded-asserting learning scheme (where k is a constant) can also p-simulate general resolution.

Theorem 3. CLR *with any k–bounded-asserting learning scheme p-simulates general resolution.*

Proof of Theorem 3 This result can be immediately obtained by applying Proposition 5 instead of Proposition 3 in the proof of Theorem 1. □

In our proof, the size of the proof produced by CLR will be bounded by $O(n^{k+3}|\Pi|)$, where Π is the given general resolution proof. When $k = 1$ (asserting learning scheme), this result reduces to that of Theorem 1.

This generalized result is interesting because it shows that a much broader class of conflict clauses may be used by CLR without compromising its strengths. One potential benefit of considering a larger set of clauses is that the set may contain more clauses with some desirable properties (e.g., having fewer literals, containing fewer implication levels [3]), which may allow the solver to derive fewer clauses in order to refute the input formula.

6.2. Practical Restart Policies

In the proof of our main result (Theorem 1), we require the solver to restart after deriving each conflict clause. While this is a valid restart policy, it is not necessary for CLR to p-simulate general resolution.

In this section, we present a generalization of our main result with respect to the restart policy used by CLR. In particular, we identify a class of restart policies, which includes policies used by many state-of-the-art solvers, that is sufficient to achieve the main result.

A restart policy is simply a set of rules, usually based on the number of conflicts experienced by the solver, that determine when the solver should restart. To simplify our discussion, in this section, we will assume that the choice on Line 13 of Algorithm 1 (whether to restart or not) is determined by a restart policy, instead of by the given branching sequence. As a result, the branching sequences used in the following discussion will no longer contain any restart symbols. In this work, we consider only restart policies that determine their restarting points based on the number of conflicts experienced by the solver. As reflected in Algorithm 1, we only allow CLR to restart when the CLR state is 1–consistent. This is also how restarting is usually implemented in practice. As a result, it is possible for CLR to be in a 1–inconsistent state when there are enough conflicts to trigger a restart. In such a situation, the solver cannot restart yet. Rather, it has to resolve the conflict(s) by going through Lines 8-11 of Algorithm 1, possibly multiple times. Once the CLR state is 1–consistent, the solver can restart (Line 13). Since no new decision is made during this process, the number of additional conflicts experienced before the solver gets into a 1–consistent state is no more than the number of variables.

We use $\phi(i)$ to denote the number of conflicts between the $(i-1)^{\text{th}}$ and the i^{th} restarts as specified by the considered restart policy. Note that $\phi(1)$ denotes the number of conflicts before the first restart. The following definition characterizes how frequently a restart policy restarts.

Definition 11. *Given a function $f : \mathbb{N} \to \mathbb{N}$, a restart policy is $\underline{f\text{–spaced}}$ if $\phi(i) \leq f(i)$, for all $i \geq 1$.*

The above notion allows us to talk about the frequency of restarts of different policies. Roughly speaking, if a policy is f–spaced, it means that its restart intervals cannot grow more quickly than f does. If f is a polynomial function, an f–spaced restart policy is one whose restart intervals grow at most polynomially with the number of restarts. The proof system of CLR that utilizes restart policy RP contains all refutation proofs that CLR can produce with RP in effect on Line 13 of Algorithm 1. In the following result, we use the phrase "CLR with a restart policy RP" to refer to the proof system obtained by the aforementioned method.

Theorem 4. *Given any polynomial f,* CLR *with a k–bounded-asserting learning scheme and an f–spaced restart policy can p-simulate general resolution.*

Proof of Theorem 4 Given any function f, it is not hard to see that we can always find a monotonically increasing function f' such that $f' \geq f$. Any f–spaced restart policy must trivially be f'–spaced as well. Hence, from now on, we assume that f is monotonically increasing.

Let RP be any f–spaced restart policy. Because CLR produces a resolution proof of size at most n (number of variables) at each conflict and RP is f–spaced, CLR may encounter at most $f(i) + n$ conflicts between the $(i-1)^{\text{th}}$ and i^{th} restarts (recall that the solver may need to deal with up to n conflicts to get into a state in which it can restart). Therefore, the total resolution proof induced by CLR during a restart interval has size $n(f(i) + n)$, where i is the index of the restart. Since f is monotonically increasing, this size of the proof induced by CLR during any restart interval can be upper-bounded by $n(f(X) + n)$, where X is the total number of restarts CLR performed.

According to the proof of Theorem 3, the solver only needs to restart $O(n^{k+1})$ times to completely absorb a clause. Since there are at most $|\Pi|$ clauses to absorb, the maximum number of restarts the solver needs to perform is in $O(n^{k+1}|\Pi|)$. Therefore, the size of the refutation proof produced by CLR using a k–bounded-asserting learning scheme and restart policy RP is in $O(n^{k+2}|\Pi|(f(n^{k+1}|\Pi|) + n))$, which is polynomial in the sizes of Π and Δ. $\square$

Notice that when $k = 1$ and $f(i) = 1$, the size of the proof becomes $O(n^4|\Pi|)$, which is the same as the result obtained in the proof of Theorem 1.

This result shows that any polynomially-spaced restart policy is sufficient to allow CLR to p-simulate general resolution. For example, the restart policy used in the proof of Theorem 1 is f–spaced for $f(i) = 1$ (i.e., restart every 1 conflict). Many commonly-used restart policies are polynomially-spaced. For instance:

- Arithmetic policies. These policies restart every X conflicts and this interval is incremented by Y after each restart. These policies are f–spaced with $f(i) = X + (i-1)Y$. Examples of solvers utilizing this type of policies are: zChaff 2004 ($X = 700, Y = 0$) [19], Berkmin ($X = 550, Y = 0$) [12], Siege ($X = 16000, Y = 0$) [27], and Eureka ($X = 2000, Y = 0$) [22].

- Luby policies [17]. The restart intervals of this type of policies are based on a series proposed in [18], which can be defined recursively as:

$$\phi(i) = \begin{cases} 2^{k+1} & \text{if } i = 2^k - 1 \\ \phi(i - 2^{k-1} + 1) & \text{if } 2^{k-1} \leq i < 2^k - 1 \end{cases}$$

  In practice, the series is scaled by a constant X (known as the *unit run*). This type of policies is f–spaced with $f(i) = 2X(i+1)$. Examples of solvers with this policy are Tinisat ($X = 512$) [16], Rsat ($X = 512$) [23], and Minisat 2.0 ($X = 100$) [10].

7. Related Work

Early work in this direction was published by Beame *et al* in [5]. In that work, the authors showed that a slight variation of CDCL 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.

In [29], van Gelder proposed a different proof system called POOL for studying CDCL SAT solvers as resolution engines. In that work, the author focused on understanding the strength of POOL and using it to model CDCL SAT solvers. The author did not directly compare the power of the solvers against general resolution.

Nevertheless, POOL later became a basis of the work by Hertel *et al* [15], which proved that CDCL SAT solvers can *effectively* p-simulate general resolution. In other words, the authors showed that, with an additional preprocessing step, CDCL SAT solvers can become as strong as general resolution. While the preprocessing is deterministic and independent of the proof being simulated, it does not simplify the input formula and can be regarded as an extra component not utilized by any solver in practice.

Buss *et al* [7] also presented a similar argument. The authors showed that with a preprocessing step (different from the one in [15]), a generalized version of clause-learning SAT solvers can p-simulate general resolution. Apart from requiring an extra preprocessing step, the proof also needed the solver to make decisions on assigned variables.

One feature that all previous attempts in this direction had in common was that their proofs essentially forced the solver to derive every clause (or a variation of every clause) in the given general resolution proof. Unfortunately, CDCL SAT solvers can only derive clauses that are 1–empowering and 1–provable with respect to the current knowledge base. As a result, the previous work had to resort to less practical models in order to get around this constraint. In particular, allowing the solver to make decisions on assigned variables (as done in [5]) enables the resulting solver to learn clauses that are not 1–empowering. The preprocessor used by [15] transforms every clause in the original formula in order to artificially create 1–empowerment. Roughly speaking, as the solver goes after every clause C in the given general resolution proof, the solver is forced to derive C', which is a transformed version of C instead. The transformation makes sure that, even though C may not be 1–empowering with respect to the current knowledge base, C' will be. In [7], the authors showed that several proof systems based on CDCL SAT solvers can effectively p-simulate general resolution. The proof required that some new variables be introduced into the given CNF first. Moreover, the proof essentially relied on the fact that the solver was allowed to use intermediate resolvents (which are not learned by normal solvers) in the derivation of future clauses. Note that while these intermediate resolvents allow the proof to simulate the derivation of every clause in the given resolution proof, they may not be 1–empowering.

8. Conclusions and Discussion

In this paper, we proved that a broad class of CDCL SAT solvers corresponds to proof systems that are as powerful as general resolution. Included in this class is the well-known CDCL SAT algorithm that learns asserting clauses. This result improves on previous results by avoiding the needs for additional degree 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 CDCL 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 CDCL SAT solvers, as used in practice for many years now, are capable of polynomially simulating any resolution proof, given the right branching and restarting heuristics.

Our result also allows us to finally view modern CDCL SAT solvers as full-fledged resolution engines that try to look for short refutation proofs. Our main theorem (Theorem 1) shows that the components discussed in this paper are already sufficient to achieve the full power of clause-learning SAT solvers. According to this perspective, any techniques not used in the proof of Theorem 1 (including existing techniques such as CNF preprocessing, and any future techniques) can now be viewed as simply heuristics for guiding resolution. This suggests that we should probably construct these components (including the branching heuristic and restart policy) from resolution point-of-view rather than the search point-of-view, which is usually taken in practice.

Finally, in spite of our result, more theoretical work still remains to be done in this research direction. The construction of our proof relies heavily on the ability of the solver to restart at will. It still remains an open question whether far-backtracking (or even chronological backtracking) without restarting is sufficient for CDCL SAT solvers to be as strong as general resolution.

References

[1] ATSERIAS, A., FICHTE, J. K., AND THURLEY, M. Clause-learning algorithms with many restarts and bounded-width resolution. In *Proceedings of 12th 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 *Proceedings of 11th International Conference on Theory and Applications of Satisfiability Testing (SAT'08)* (2008), pp. 21–27.

[3] AUDEMARD, G., AND SIMON, L. Predicting learnt clauses quality in modern sat solver. In *Proceedings of 21st International Joint Conference on Artificial Intelligence (IJCAI'09)* (July 2009), pp. 399–404.

[4] BAYARDO, R. J. J., AND SCHRAG, R. C. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of 14th National Conference on Artificial Intelligence (AAAI-97)* (1997), pp. 203–208.

[5] BEAME, P., KAUTZ, H., AND SABHARWAL, A. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research 22* (2004), 319–351.

[6] BEN-SASSON, E., AND WIGDERSON, A. Short proofs are narrow—resolution made simple. *Journal of the ACM 48*, 2 (2001), 149–169.

[7] 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).

[8] COOK, S. A., AND RECKHOW, R. A. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic 44*, 1 (1979), 36–50.

[9] DAVIS, M., LOGEMANN, G., AND LOVELAND, D. A machine program for theorem-proving. *Commun. ACM 5*, 7 (1962), 394–397.

[10] EÉN, N., AND SÖRENSSON, N. *MiniSat v2.0 (Beta) Solver Description*. http://fmv.jku.at/sat-race-2006/descriptions/27-minisat2.pdf.

[11] EÉN, N., AND SÖRENSSON, N. An extensible sat-solver. In *SAT'03* (2003), pp. 502–518.

[12] GOLDBERG, E., AND NOVIKOV, Y. Berkmin: A fast and robust sat solver. In *Proceedings of the Design Automation and Test in Europe (DATE'02)* (2002), pp. 142–149.

[13] GOLDBERG, E., AND NOVIKOV, Y. Verification of proofs of unsatisfiability for cnf formulas. In *Proceedings of the Design Automation and Test in Europe (DATE'03)* (2003), pp. 10886–10891.

[14] GOMES, C. P., SELMAN, B., AND CRATO, N. Heavy-tailed distributions in combinatorial search. In *Proceedings of 3rd International Conference on Principles and Practice of Constraint Programming (CP-97)* (1997), pp. 121–135.

[15] HERTEL, P., BACCHUS, F., PITASSI, T., AND VAN GELDER, A. Clause learning can effectively p-simulate general propositional resolution. In *Proceedings of 23rd AAAI Conference on Artificial Intelligence (AAAI-08)* (2008), pp. 283–290.

[16] HUANG, J. A case for simple sat solvers. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP-07)* (2007), pp. 839–846.

[17] HUANG, J. The effect of restarts on the efficiency of clause learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)* (2007), pp. 2318–2323.

[18] LUBY, M., SINCLAIR, A., AND ZUCKERMAN, D. Optimal speedup of las vegas algorithms. *Information Processing Letters 47* (1993), 173–180.

[19] MAHAJAN, Y., FU, Z., AND MALIK, S. zchaff2004: An efficient sat solver. In *Proceedings of 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04) (Selected Papers)* (2004), pp. 360–375.

[20] MARQUES-SILVA, J. P., AND SAKALLAH, K. A. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of International Conference on Computer-Aided Design (ICCAD '96)* (1996), pp. 220–227.

[21] MOSKEWICZ, M., MADIGAN, C., ZHAO, Y., ZHANG, L., AND MALIK, S. Chaff: Engineering an efficient sat solver. In *Proceedings of 38th Design Automation Conference (DAC'01)* (2001).

[22] NADEL, A., GORDON, M., PALTI, A., , AND HANNA, Z. Eureka-2006 SAT solver. Solver description for SAT-Race 2006.

[23] PIPATSRISAWAT, K., AND DARWICHE, A. *RSat 2.0: SAT Solver Description*, 2007.

[24] PIPATSRISAWAT, K., AND DARWICHE, A. A new clause learning scheme for efficient unsatisfiability proofs. In *Proceedings of 23rd AAAI Conference on Artificial Intelligence (AAAI-08)* (2008), pp. 1481–1484.

[25] PIPATSRISAWAT, K., AND DARWICHE, A. On the power of clause-learning SAT solvers with restarts. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP-09)* (September 2009), pp. 654–668.

[26] ROBINSON, J. A. A machine-oriented logic based on the resolution principle. *Journal of the ACM 12*, 1 (1965), 23–41.

[27] RYAN, L. Efficient Algorithms for Clause-Learning SAT Solvers. Master's thesis, Simon Fraser University, 2004.

[28] SANG, T., BEAME, P., AND KAUTZ, H. Heuristics for fast exact model counting. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT'05),* (2005), pp. 226–240.

[29] VAN GELDER, A. Pool resolution and its relation to regular resolution and dpll with clause learning. In *Proceedings of 12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'05)* (2005), pp. 580–594.

[30] ZHANG, L., MADIGAN, C. F., MOSKEWICZ, M. W., AND MALIK, S. Efficient conflict driven learning in boolean satisfiability solver. In *Proceedings of International Conference on Computer-Aided Design (ICCAD'01)* (2001), pp. 279–285.