A linearized DPLL calculus with clause learning

Holger Arnold*

March 8, 2009

Abstract

Many formal descriptions of DPLL-based SAT algorithms either do not include all essential proof techniques applied by modern SAT solvers or are bound to particular heuristics or data structures. This makes it difficult to analyze proof-theoretic properties or the search complexity of these algorithms. In this paper we try to improve this situation by developing a nondeterministic proof calculus that models the functioning of SAT algorithms based on the DPLL calculus with clause learning. This calculus is independent of implementation details yet precise enough to enable a formal analysis of realistic DPLL-based SAT algorithms.

1 Introduction

The problem of determining whether a propositional logic formula has a model is called the Boolean satisfiability (SAT) problem. Programs solving this problem (SAT solvers) have applications in the verification of hardware and software systems and many other areas. Most complete SAT solvers currently available are based on some variant of the DPLL calculus extended with clause learning. The calculus commonly referred to as *the* DPLL calculus is the propositional part of the calculus given by Davis et al. [6, 7] in the 1960s. The clause learning extension allows to add derived clauses to the original formula during proof search. As clause learning is typically initiated and directed by conflicts, the respective calculus is referred to as conflict-directed (or conflict-driven) backtracking and learning (CDBL). Clause learning is related to the superresolution calculus of Lieberherr [10] and other lemma learning methods. It had already been an established technique in the constraint programming community before Marques-Silva and Sakallah [12] introduced clause learning to the SAT world.

In contrast to other recent advances in the development of efficient SAT solvers, clause learning is not merely an implementation technique, but actually strengthens the resulting proof system in terms of proof complexity. Alekhnovich et al. [1] showed that the proof systems of general resolution and regular resolution can be exponentially separated. Based on this result, Beame et al. [3] constructed a family of formulas providing an exponential separation between the DPLL calculus extended by clause learning and the original DPLL calculus.

*Universität Potsdam, Institut für Informatik, August-Bebel-Str. 89, 14482 Potsdam, Germany. See http://holgerarnold.net for contact information.

The prevailing method of analyzing the proof complexity of DPLL-based calculi is based on the fact that for every search tree corresponding to a run of a DPLL algorithm on an unsatisfiable formula a resolution proof can be computed. This makes it possible to adapt proof complexity results that have been established for various resolution refinements to the DPLL calculus [3, 5, 9]. Although this method has been very successful, it is not sufficient for analyzing the *search* complexity of DPLL algorithms, as it considers only the end result of the search — the proof, — but not the complexity of finding it.[1]

In addition to its strength in proof-complexity, which is first of all a theoretical result, the DPLL calculus with clause learning has also turned out to be very efficient in practice. It is particularly suitable for solving SAT problems stemming from real-world applications. Considering the practical relevance of this calculus, it is surprising that there are only few formal descriptions of DPLL-based algorithms that contain all essential proof techniques but are not bound to specific heuristics or data structures. For instance, many articles on this subject explain in detail dynamic literal selection strategies such as "variable state independent decaying sum" and data structures for efficient constraint propagation such as "watched literal lists" [8, 11, 13, 15]. These details are important to implementers but are irrelevant for analyzing proof-theoretic properties or for proving bounds on the search complexity of the calculus.

In this paper we try to improve this situation. We develop a nondeterministic proof calculus that models the functioning of SAT algorithms based on the DPLL calculus with clause learning. This calculus is independent of implementation details like literal selection heuristics and data structures. Nevertheless, it describes the essential proof techniques implemented in modern SAT solvers sufficiently precise to enable a formal analysis of realistic DPLL-based SAT algorithms.

We develop our calculus first in a general form, called Linearized DPLL (LD), and prove that this calculus is sound and terminating (Section 2.1). We then show that the original DPLL calculus can be described as an instance of the LD calculus and prove that this specialization is proof confluent and complete (Section 2.2). After that we do the same for the CDBL calculus that is implemented in many current SAT solvers (Section 2.3). We particularly elaborate on the process of deriving implied clauses from conflicts by linear resolution. We also show how conflict graphs, which are often used as graphical representations of this derivation process, correspond to proof steps in a CDBL proof.

In many respects, our calculus is similar to the Abstract DPLL procedure by Nieuwenhuis et al. [14]. The two calculi differ, however, in their approach to clause learning. The calculus by Nieuwenhuis et al. models clause learning only on a very general level. It contains proof rules that basically allow to replace the current clause set by any logically equivalent set of clauses, but the calculus does not suggest a method for deriving the clauses to be added or removed. In contrast to this, the specialization of the LD calculus to the CDBL calculus, which is the subject of section 2.3, directly models the resolution process that current SAT solvers implement to derive implied clauses, but it is still general enough to cover different proof strategies. Examples of efficient implementations of this calculus, whose sources are publicly available, are the

[1] For instance, every satisfiable formula has a trivial proof that is just of a description of a model of the formula. Yet it can still be difficult for a DPLL algorithm to find it.

SAT solvers MiniSat [8], NanoSAT [4], and zChaff [13, 16].

1.1 Formal preliminaries

An *atomic formula* or *atom* is an element of some discrete set of propositional symbols. Atoms are also called propositional variables, but we do not use this terminology in this text. The operators $\neg$, $\wedge$, and $\vee$ denote logical negation, conjunction, and disjunction. If p is an atom, then p is a *positive literal* and $\neg p$ is a *negative literal.* The *complement* $\bar{l}$ of a literal l is $\neg p$ if $l = p$, and it is p if $l = \neg p$. A *clause* is a formula of the form $l_1 \vee \cdots \vee l_m$, where $l_1, \ldots, l_m$ are literals. A formula is in *conjunctive normal form* if it can be written as $C_1 \wedge \cdots \wedge C_n$, where $C_1, \ldots, C_n$ are clauses. A clause can be represented by the set of its literals, and a formula in conjunctive normal form can be represented by the set of its clauses. An *interpretation* is a mapping of formulas to truth values that is generated by a mapping of atomic formulas to truth values and is extended to non-atomic formulas by using the classical semantics of the logical operators. If an interpretation does not define truth values for all atoms of a formula, then the interpretation is *partial.* An interpretation M *satisfies* a formula F if M maps the formula to *true*; M is then called a *model* of F. If F is in conjunctive normal form, then M is a model of F if it maps at least one literal of each clause of F to *true*. A formula F *entails* a formula G, written $F \models G$, if every model of F is also a model of G. If $F \models \neg G$, then F is said to *falsify* G. Two formulas F and G are *logically equivalent*, written $F \equiv G$, if $F \models G$ and $G \models F$. Unless otherwise stated, we assume formulas to be in conjunctive normal form. There are efficient algorithms for computing from a formula F a formula F' in conjunctive normal form such that F' is satisfiable if and only if F is satisfiable (although F and F' will in general not be logically equivalent).

2 The linearized DPLL calculus

The LD calculus can be used to find a model of a propositional logic formula in conjunctive normal form or to prove that no such model exists. The original DPLL calculus is usually presented as a branching calculus, which means that to close a proof it is generally necessary to close several distinct branches. A branching calculus, however, is not particularly suitable for being extended by clause learning. The problem is that the learned clauses are global information. This would necessitate a mechanism for transferring information between independent branches, which is in conflict with the local nature of the branches. Therefore we present the DPLL calculus in a linearized form.

During proof search, the LD calculus maintains a non-contradictory, non-redundant set of literals called the *context.*[2] It contains the literals that are assumed to be true in a given state of the proof and so constitutes a partial interpretation of the formula. This context is incrementally extended by assuming literals and deriving the consequences of these assumptions. Extending the context by an assumption is called a *decision*, and literals added to the context by this means are called *decision literals.* The process of deriving consequences

[2] The term *context* as well as the notation $M \vdash F$ have been adopted from the Model Evolution Calculus by Baumgartner and Tinelli [2].

of decisions is called *propagation*, and literals added to the context that way are called *propagated literals*.[3] The sequence of decision literals and propagated literals in a context forms a linear encoding of a position in the branch structure as it is present in a DPLL proof.

When the input formula is falsified by the current context, this situation is called a *conflict*. To resolve a conflict, some decisions and their consequences have to be retracted, which corresponds to the closure of one or more branches in a branching calculus. In a conflict-directed SAT solver the occurrence of a conflict initiates an analysis of its *reasons*, where a reason can be described by a set of literals whose conjunction falsifies the input formula. One or more clauses that prevent conflicts caused by the same reasons from occurring in subsequent proof steps are then added to the input formula. This process is referred to as *clause learning*. The proof search continues until either a model satisfying the input formula has been found or a conflict occurs that cannot be resolved. In the latter case, the formula is proved unsatisfiable, which corresponds to the situation where all branches have been closed in a branching calculus.

The LD calculus manipulates sequents of the form $M \vdash F$, where M is a context and F is a set of clauses containing the clauses of the input formula and those learned in previous proof steps (and not deleted afterward). A context is a linearly ordered set of literals. When viewed as a formula, a context is interpreted as the conjunction of its literals. If l is a decision literal in a given context, this fact is pointed out by writing l^d. When a context is written in the form $M_0, l_1^d, M_1, \ldots, l_k^d, M_k$, it is implicitly assumed that $l_1^d, \ldots, l_k^d$ are the only decision literals in that context.[4] The decision literals partition the elements of a context into *decision levels*. If $M_0, l_1^d, M_1, \ldots, l_k^d, M_k$ is a context with decision literals $l_1^d, \ldots, l_k^d$, then the literals in M_0 are assigned decision level 0, and for all $i \in \{1, \ldots, k\}$, the literals in M_i and l_i^d are assigned decision level i. The decision level of a proof state $M \vdash F$ is the maximum of the decision levels of literals in M. If S is a set of literals, then $\overline{S} = \{\bar{l} \mid l \in S\}$ denotes the set of complements of literals in S. By $F|_M$ we denote the formula F *evaluated* under the context M, which is defined by

$$F|_M = \{C \setminus \overline{M} \mid C \in F \text{ and } C \cap M = \emptyset\}.$$

A formula F is satisfiable by an interpretation that is consistent with a context M if and only if $F|_M$ is satisfiable. A formula F is falsified by the context M if and only if $F|_M$ contains the empty clause. The set of literals occurring in a formula F is denoted by $\mathit{lit}(F)$. In the following, contexts are designated by the letter M, formulas by F, clauses by C, and literals by l.

2.1 The LD calculus

In this subsection we present the rules of the LD calculus. It is formulated very similarly to the system described by Nieuwenhuis et al. [14]. The calculus is devised in such a way that the original DPLL calculus as well as the CDBL calculus implemented in SAT solvers built on clause learning can be described

[3] Although the terms “decision literal” and “propagated literal” are commonly used, they are somewhat misleading because they are *not* referring to properties of the literals, but to their role within the context.

[4] As customary, a one-element set $\{x\}$ is denoted by x and a (disjunctive) set union $A \cup B$ is denoted by A, B where the meaning is clear from the context.

as instances of the LD calculus. The relation between these calculi is formally characterized in subsections 2.2 and 2.3.

Definition 1. The *LD calculus (Linearized DPLL)* is defined by the following set of rules:

$$\textsf{Start} \quad \emptyset \vdash F$$

$$\textsf{Unsatisfiable} \quad \frac{M, M' \vdash F}{\Box} \quad \text{if } \begin{cases} \emptyset \in F|_M \text{ and} \\ M \text{ contains no decision literals} \end{cases}$$

$$\textsf{Decide}(l) \quad \frac{M \vdash F}{M, l^d \vdash F} \quad \text{if } \begin{cases} l \in \mathit{lit}(F) \text{ and} \\ M \cap \{l, \bar{l}\} = \emptyset \end{cases}$$

$$\textsf{Propagate}(l) \quad \frac{M \vdash F}{M, l \vdash F} \quad \text{if } \{l\} \in F|_M$$

$$\textsf{Back}(C, l', l) \quad \frac{M, l^d, M' \vdash F}{M, l' \vdash F} \quad \text{if } \begin{cases} C, l' \subseteq \mathit{lit}(F), \\ F \models C \vee l', \\ M \cap \{l', \bar{l'}\} = \emptyset, \text{ and} \\ \{C\}|_M = \{\emptyset\} \end{cases}$$

$$\textsf{Learn}(C) \quad \frac{M \vdash F}{M \vdash F, C} \quad \text{if } \begin{cases} C \subseteq \mathit{lit}(F) \text{ and} \\ F \models C \end{cases}$$

$$\textsf{Delete}(C) \quad \frac{M \vdash F, C}{M \vdash F} \quad \text{if } F \models C$$

A proof of the formula F in the LD calculus starts with $\emptyset \vdash F$ and is closed when either $\Box$ has been derived by means of the Unsatisfiable rule, in which case F has been proved unsatisfiable, or a state $M \vdash F'$ has been derived such that $F'|_M = \emptyset$. In the latter case F is satisfiable and every interpretation consistent with M is a model of F.

In the remainder of this subsection the LD calculus is proved sound and terminating. Naturally, these properties carry over to any instance of the LD calculus.

2.1.1 Soundness of LD

We begin with some simple technical observations. If $M \vdash F'$ can be derived from $\emptyset \vdash F$ in the LD calculus, then the conditions of the rules trivially ensure that F' and F are logically equivalent, $\mathit{lit}(F')$ and $\mathit{lit}(M)$ are both subsets of $\mathit{lit}(F)$, and M contains at most one literal of each atom occurring in F (note that a literal l can only be appended to a context M if $M \cap \{l, \bar{l}\} = \emptyset$, as for all literals $l \in M$, $F|_M$ contains neither l nor $\bar{l}$). The latter means that the context is non-redundant and non-contradictory.

The following lemma contains the main argument in the soundness proof. It states that propagated literals are entailed by the conjunction of the input formula and the decision literals assigned at previous decision levels.

Lemma 2. *If $M \vdash F$ can be derived in the LD calculus such that the context M has the form $M_0, l_1^d, M_1, \ldots, l_k^d, M_k$, where $l_1^d, \ldots, l_k^d$ are the decision literals of M, then $F, l_1^d, \ldots, l_i^d \models M_i$ for all $i \in \{0, \ldots, k\}$.*

Proof. We prove the statement by induction on the length of the derivation of $M \vdash F$. If the proof of $M \vdash F$ consists only of a Start step, then $M = \emptyset$, and the statement holds trivially. Now assume that $M \vdash F$ has been derived in $m > 1$ steps and that the statement holds for all derivations of length less than m. We split our argument depending on the rule applied in the last step of the proof of $M \vdash F$, which must be an application of one of the rules Decide, Propagate, Back, Learn, or Delete:

1. If the rule applied in the last step of the proof is Decide, then this step can be written as

$$\mathsf{Decide}(l_k) \quad \frac{M' \vdash F}{M', l_k^d \vdash F},$$

where $M' = M_0, l_1^d, M_1, \ldots, M_{k-1}$. Thus $M_k = \emptyset$ and the statement follows from the induction hypothesis.

2. In case the rule applied in the last step of the proof is Propagate, this step can be written as

$$\mathsf{Propagate}(l) \quad \frac{M' \vdash F}{M', l \vdash F},$$

where

$$\begin{aligned} M' &= M_0, l_1^d, M_1, \ldots, l_k^d, M_k', \\ M_k &= M_k', l, \end{aligned}$$

such that $\{l\} \in F|_{M'}$. By the induction hypothesis, $F, l_1^d, \ldots, l_k^d$ entails the context $M_0, \ldots, M_{k-1}, M_k'$. Further, any model of F consistent with M' must map the literal l to *true*. Consequently, $F, l_1^d, \ldots, l_k^d$ entails l and hence it also entails M_k.

3. If the rule applied in the last step of the proof is Back, then the proof must contain a state $M' \vdash F'$ derived in less than m steps such that the derivation of $M \vdash F$ can be written in the form

$$\begin{array}{c} \vdots \\ M' \vdash F' \\ \vdots \end{array}$$

$$\mathsf{Back}(C, l', l) \quad \frac{M', l^d, M'' \vdash F}{M', l' \vdash F},$$

where

$$\begin{aligned} M' &= M_0, l_1^d, M_1, \ldots, l_k^d, M_k', \\ M_k &= M_k', l', \end{aligned}$$

such that $F \models C \vee l'$ and $\{C\}|_{M'} = \{\emptyset\}$. By the induction hypothesis, $F, l_1^d, \ldots, l_k^d$ entails the context $M_0, \ldots, M_{k-1}, M_k'$. Because of $F \models C \vee l'$ and $\{C\}|_{M'} = \{\emptyset\}$, any model of F that is consistent with M' must map the literal l' to *true*. Consequently, $F, l_1^d, \ldots, l_k^d$ entails l' and hence it also entails M_k.

4. If the rule applied in the last step of the proof is Learn or Delete, then this step can either be written as

$$\mathsf{Learn}(C)\ \frac{M \vdash F'}{M \vdash F} \qquad \text{or as} \qquad \mathsf{Delete}(C)\ \frac{M \vdash F'}{M \vdash F},$$

where $F' \equiv F$. The statement then follows from the induction hypothesis. □

Proposition 3 (Soundness)**.** *If $M, M' \vdash F'$ can be derived from $\emptyset \vdash F$ in the LD calculus such that $\emptyset \in F'|_M$ and M contains no decision literals, then F is unsatisfiable. If $M \vdash F'$ can be derived from $\emptyset \vdash F$ in the LD calculus such that $F'|_M = \emptyset$, then F is satisfiable and every interpretation consistent with M is a model of F.*

Proof. Assume that $M, M' \vdash F'$ can be derived from $\emptyset \vdash F$ such that $\emptyset \in F'|_M$ and M contains no decision literals. Then no interpretation consistent with M can satisfy F', as no interpretation can satisfy the empty clause. But lemma 2 shows that M is entailed by F', as M contains no decision literals. Consequently, F' must be unsatisfiable. Because F and F' are equivalent, F must be unsatisfiable as well.

On the other hand, assume that $M \vdash F'$ can be derived from $\emptyset \vdash F$ such that $F'|_M = \emptyset$. As the context M is non-contradictory, it generates a partial interpretation that maps at least one literal in each clause of F' to *true*. Hence every interpretation consistent with M is a model of F', and therefore also a model of F. □

2.1.2 Termination of LD

The following proposition shows that the LD calculus can actually be used to build a decision procedure for clausal propositional logic.

Proposition 4 (Termination)**.** *Every derivation in the LD calculus that contains no infinite sequence of consecutive Learn and Delete steps is finite.*

Proof. When one of the rules Decide, Propagate, and Back is applied to a proof state with decision level d, then either the context is extended by a literal or the number of literals with a decision level less than d is increased. But the number of literals in the context is bounded, as the context is non-redundant and contains only literals of the input formula. Therefore the length of a derivation without Learn and Delete steps is bounded as well.

The formalization of this argument has been adopted from Zhang [15]. Let $n(M,k)$ denote the number of literals with decision level k in the context M and consider the discrete function f_m defined as

$$f_m(M) = \sum_{k=0}^{m} \frac{n(M,k)}{(m+1)^k}.$$

Note that $1/(m+1)^k > m/(m+1)^{k+1}$ for all k with $0 < k \leq m$. Let M and M' be two contexts occurring in a derivation. Then $f_m(M) > f_m(M')$ if and only if there exists a decision level k with $0 \leq k < m$ such that $n(M,k) > n(M',k)$ and $n(M,k') = n(M',k')$ for any $k' < k$.

Consider the sequence $\emptyset, M_1, M_2, \ldots$ of contexts in a proof derived from of $\emptyset \vdash F$, and let m be the number of atoms in F. If M_{i+1} is a context that arises from a context M_i by an application of one of the rules Decide and Propagate, then $f_m(M_{i+1}) > f_m(M_i)$ because M_i must be a subset of M_{i+1}. If M_{i+1} arises from M_i by an application of the Back rule, then M_i must have the form M', l^d, M'' for some decision literal l with decision level k and M_{i+1} must have the form M', l' for some propagated literal l'. But then M_{i+1} contains one more literal with decision level $k-1$ than M_i and therefore $f_m(M_{i+1}) > f_m(M_i)$. Thus in a derivation without applications of the rules Learn and Delete the value of f_m is strictly increasing. But the range of f_m is bounded from above by m; this value is reached when all literals are assigned decision level 0. Hence every derivation without applications of the rules Learn and Delete must be finite.

The rules Learn and Delete do not modify the context of the state they are applied to. Consequently, every infinite derivation in the LD calculus must contain an infinite sequence of consecutive Learn and Delete steps. □

2.2 The DPLL calculus

In this subsection we show that the original calculus by Davis et al. [6, 7] can be described in a linearized form as an instance of the LD calculus.

Definition 5. The *DPLL calculus (Davis-Putnam-Logemann-Loveland)* is defined by the rules Start, Unsatisfiable, Decide, and Propagate from the LD calculus (see definition 1) and the following rule:

$$\mathsf{Back}_{\mathrm{DP}}(l) \quad \frac{M, l^d, M' \vdash F}{M, \bar{l} \vdash F} \quad \text{if } \begin{cases} \emptyset \in F|_{M,l^d,M'} \text{ and} \\ M' \text{ contains no decision literals} \end{cases}$$

All rules of the DPLL calculus leave the clause set unmodified and no learning is done at all. What has been omitted from the DPLL calculus, compared with Davis and Putnam's original formulation, is the Affirmative-Negative rule (or Pure-Literal rule, as it is called today) that allows the propagation of literals occurring only positively or only negatively in the formula. The reason for this omission is that such literals are not entailed by the conjunction of formula and context (cf. lemma 2), and handling them would unnecessarily complicate the conditions of the rules. Most implementations of the calculus do not use this rule anyway.

2.2.1 Soundness of DPLL

The following lemma shows that the DPLL calculus is an instance of the LD calculus. As a result, it inherits the soundness and termination properties from the LD calculus (propositions 3 and 4).

Lemma 6. *If $M \vdash F$ can be derived in the DPLL calculus, then every $\mathsf{Back}_{\mathrm{DP}}$ step in the derivation can be replaced by a Back step such that the resulting derivation is a valid proof in the LD calculus.*

Proof. We proof the statement by induction on the length of the derivation. A derivation of length 1 can only consist of a Start step, for which the statement holds trivially. Now assume that $M \vdash F$ has been derived in $m > 1$ steps and that the statement holds for all derivations of length less than m. We need only consider the case where the last step of the derivation is an application of the Back rule. The derivation of $M \vdash F$ can then be written as

$$\mathsf{Back}_{\mathrm{DP}}(l) \quad \frac{\begin{matrix}\vdots \\ M', l^d, M'' \vdash F\end{matrix}}{M', \bar{l} \vdash F},$$

where $M = M', \bar{l}$, such that $\emptyset \in F|_{M', l^d, M''}$ and M'' contains no decision literals. Because M', l^d, M'' falsifies F and M'' is entailed by F, M', l^d by lemma 2, the formula F is already falsified by M', l^d. Let $C = \overline{M'}$. Then $\{C\}|_{M'} = \{\emptyset\}$, $F \models C \vee \bar{l}$, and $C \subseteq \mathit{lit}(F)$. By the induction hypothesis, $M' \cap \{l, \bar{l}\} = \emptyset$. Thus the above derivation can be replaced by

$$\mathsf{Back}(C, \bar{l}, l) \quad \frac{\begin{matrix}\vdots \\ M', l^d, M'' \vdash F\end{matrix}}{M', \bar{l} \vdash F},$$

which is a valid proof in the LD calculus. □

2.2.2 Proof confluence and completeness of DPLL

The following lemma shows that as long as a DPLL proof has not been closed, there is always a rule of the calculus that can be applied, which is a necessary condition for a proof confluent calculus.

Lemma 7. *If $M \vdash F$ has been derived in the DPLL calculus and $F|_M \neq \emptyset$, then at least one rule of the calculus can be applied to this state.*

Proof. Assume that M falsifies F. If M contains no decision literals, then the $\mathsf{Unsatisfiable}$ rule can be applied. If M contains a decision literal, then it can be written as $M = M', l^d, M''$ such that M'' contains no decision literals. In this case, the Back rule can be applied.

On the other hand, assume that M does not falsify F. Because of $F|_M \neq \emptyset$, the formula F must contain at least one literal l such that $M \cap \{l, \bar{l}\} = \emptyset$. If F contains a clause C such that $\{C\}|_M = \{l'\}$ for some literal l', then the $\mathsf{Propagate}$ rule can be applied. If F contains no such clause, then the Decide rule can be applied. □

It should be emphasized that the proof of lemma 7 is constructive and gives rise to an algorithm for deciding the satisfiability of propositional clause sets by creating derivations in the DPLL calculus. Whenever there are multiple possible choices in the algorithm, the actual decision taken has no influence on the correctness and proof confluence of the procedure (a property referred to as don't-care non-determinism), although different choices might result in different proof lengths. We can now conclude that the DPLL calculus is a proof confluent and complete calculus for clausal propositional logic.

Proposition 8 (Proof confluence and completeness)**.** *Let $\emptyset \vdash F, \ldots, M \vdash F$ be derivation in the DPLL calculus. If F is unsatisfiable, then $\Box$ can be derived from $M \vdash F$. If F is satisfiable, then $M' \vdash F$ can be derived from $M \vdash F$ such that $F|_{M'} = \emptyset$.*

Proof. By lemma 6 and propositions 3 and 4, the DPLL calculus is sound and terminating (note that the representation of a DPLL proof in the LD calculus contains no Learn or Delete steps). By lemma 7, the derivation $\emptyset \vdash F, \ldots, M \vdash F$ can be continued as long as the proof is not closed, and because the calculus is terminating, it will eventually be closed. By the soundness of the calculus, the result will be $\square$ if F is unsatisfiable, and it will be a state $M' \vdash F$ such that $F|_{M'} = \emptyset$ if F is satisfiable (where $M' = M$ in case the proof had already been closed). □

2.3 Conflict-directed backtracking and learning

In this subsection we show that the conflict-directed backtracking calculus with clause learning that it is implemented in many current SAT solvers can also be described as an instance of the LD calculus. Clause learning is typically induced and directed by conflicts in these solvers, and the learned clauses are derived from a conflicting clause by linear resolution.[5]

Definition 9. The *CDBL calculus (conflict-directed backtracking and learning)* is defined by the rules Start, Unsatisfiable, Decide, and Propagate from the LD calculus (see definition 1) and the following rules:

$$\textsf{Conflict}(C) \quad \frac{M \vdash F}{C \mid M \vdash F} \quad \text{if} \begin{cases} C \in F, \\ \{C\}|_M = \{\emptyset\}, \text{ and} \\ M \text{ contains a decision literal} \end{cases}$$

$$\textsf{Resolve}(C', l) \quad \frac{C \vee \bar{l} \mid M, l, M' \vdash F}{C \vee C' \mid M, l, M' \vdash F} \quad \text{if} \begin{cases} C' \subseteq \mathit{lit}(F), \\ F \models C' \vee l, \text{ and} \\ \{C'\}|_M = \{\emptyset\} \end{cases}$$

$$\textsf{Back-and-Learn}(l', l) \quad \frac{C \vee l' \mid M, l^d, M' \vdash F}{M, l' \vdash F, C \vee l'} \quad \text{if} \begin{cases} M \cap \{l', \overline{l'}\} = \emptyset \text{ and} \\ \{C\}|_M = \{\emptyset\} \end{cases}$$

When a conflict occurs, the proof state is extended by a clause falsified by the current context. This clause forms the starting point for a sequence of linear resolution steps used to derive a clause that is then added as a lemma to the original formula. The family of learning schemes defined by above rules can be referred to as *UIP learning* because the linear resolution derivation always ends at a so-called *unique implication point* (a term coined by Marques-Silva and Sakallah [12]). Of course, many extensions of this scheme are possible and are actually implemented in various SAT solvers. A straightforward extension would be to learn more than one clause per conflict, for example by adding a subset of all clauses derived in the resolution steps.

The Delete rule has been deliberately omitted from the calculus to simplify the conditions of the remaining rules. The results proved in this subsection also hold for the CDBL calculus extended by Delete, provided that the conditions of the rules are tightened such that the Conflict rule must be applied whenever possible. SAT solvers usually implement the calculus in this way.

[5] A resolution derivation is called linear if each resolution step except the first involves the resolvent of the previous step.

2.3.1 Soundness of CDBL

The following lemma shows that in a CDBL derivation every sequence of proof steps applying the rules Conflict, Resolve, and Back-and-Learn, which are not available in the LD calculus, can be replaced by applications of the rules Back and Learn. This means that the CDBL calculus can (in an extended sense) also be considered an instance of the LD calculus. Using the result that any such sequence of proof steps is finite, which is shown later in the proof of lemma 11, it follows that the CDBL calculus inherits the soundness and termination properties from the LD calculus (propositions 3 and 4).

Lemma 10. *If $M \vdash F$ can be derived in the CDBL calculus, then every sequence starting with a Conflict step, followed by a (possibly empty) sequence of Resolve steps, and ending with a Back-and-Learn step in the derivation can be replaced by a Back step and a Learn step such that the resulting derivation is a valid proof in the LD calculus.*

Proof. Without loss of generality we assume that the derivation of $M \vdash F$ contains only one sequence starting with a Conflict step, followed by a (possibly empty) sequence of Resolve steps, and ending with a Back-and-Learn step and that the Back-and-Learn step is the last step in the proof. The general result follows by induction on the length of the proof. The derivation of $M \vdash F$ can then be written as

$$\begin{array}{rc}
 & \vdots \\
\begin{array}{r} \mathsf{Conflict}(C_0 \vee \bar{l}_0) \\ \mathsf{Resolve}(C_0', l_0) \end{array} & \begin{array}{c} M' \vdash F' \\ \hline C_0 \vee \bar{l}_0 \mid M_0, l_0, M_0' \vdash F' \\ \hline C_1 \vee \bar{l}_1 \mid M_1, l_1, M_1' \vdash F' \end{array} \\
 & \vdots \\
\mathsf{Back\text{-}and\text{-}Learn}(\bar{l}_k) & \begin{array}{c} C_k \vee \bar{l}_k \mid M_k, l^d, M_k' \vdash F' \\ \hline M_k, \bar{l}_k \vdash F', C_k \vee \bar{l}_k \end{array}
\end{array}$$

where $F = F' \cup \{C_k \vee \bar{l}_k\}$, $M = M_k, \bar{l}_k$, and $M' = M_k, l^d, M_k' = M_i, l_i, M_i'$ for all $i \in \{0, \dots, k-1\}$. By the conditions of the rules Conflict and Resolve, the formula F' contains the clause $C_0 \vee \bar{l}_0$ and for all $i \in \{0, \dots, k-1\}$, the clause $C_{i+1} \vee \bar{l}_{i+1}$ arises by resolving $C_i \vee \bar{l}_i$ with a clause $C_i' \vee l_i$ entailed by F' on the literal l_i. By the soundness of resolution [6], $F' \models C_k \vee \bar{l}_k$, and trivially $C_k \vee \bar{l}_k \subseteq \mathit{lit}(F')$. The condition of the Back-and-Learn rule ensures that $M_k \cap \{l_k, \bar{l}_k\} = \emptyset$ and $\{C_k\}|_{M_k} = \{\emptyset\}$. The above derivation of $M \vdash F$ can therefore be replaced by

$$\begin{array}{rc}
 & \vdots \\
\begin{array}{r} \mathsf{Back}(C_k, \bar{l}_k, l) \\ \mathsf{Learn}(C_k \vee \bar{l}_k) \end{array} & \begin{array}{c} M_k, l^d, M_k' \vdash F' \\ \hline M_k, \bar{l}_k \vdash F' \\ \hline M_k, \bar{l}_k \vdash F', C_k \vee \bar{l}_k \end{array}
\end{array}$$

which is a valid proof in the LD calculus. □

2.3.2 Conflict resolution

The following lemma shows that when a conflict occurs in a CDBL proof, either the formula to be proved is unsatisfiable and the proof can be closed or a sub-proof resolving the conflict and extending the formula to be proved by an implied clause can be produced.

Lemma 11. *If $M \vdash F$ can be derived in the CDBL calculus such that F contains a clause C, where $\{C\}|_M = \{\emptyset\}$, then either the proof can be closed by applying the* Unsatisfiable *rule or a sub-proof starting with a* Conflict *step, followed by a (possibly empty) finite sequence of* Resolve *steps, and ending with a* Back-and-Learn *step can be derived from $M \vdash F$.*

Proof. As $\{C\}|_M = \{\emptyset\}$, every literal of C must be mapped to *false* by M. Let l_0 be the last literal in M (recall that contexts are *ordered* sets) whose complement occurs in C. Then C and M can be written as $C = C_0 \vee \bar{l}_0$ and $M = M_0, l_0, M_0'$, where $\overline{C}_0 \cap M_0' = \emptyset$. If the context M_0, l_0 contains no decision literals, then the Unsatisfiable rule can be applied to $M \vdash F$ because C is already falsified by M_0, l_0. If the context M_0, l_0 contains a decision literal, then a sub-proof can be started by applying the Conflict rule:

$$\mathsf{Conflict}(C_0 \vee \bar{l}_0) \qquad \frac{M \vdash F}{C_0 \vee \bar{l}_0 \mid M_0, l_0, M_0' \vdash F}$$

$$\vdots$$

Consider a state $S_i = C_i \vee \bar{l}_i \mid M_i, l_i, M_i' \vdash F$ derived in the sub-proof such that

$$F \models C_i \vee \bar{l}_i, \quad \{C_i\}|_{M_i} = \{\emptyset\}, \quad \overline{C_i} \cap M_i' = \emptyset. \qquad (*)$$

The first state S_0, derived by means of the Conflict rule, fulfills these conditions, and we will see that this invariant is preserved by applications of the Resolve rule.

If all literals in C_i have a strictly lower decision level than l_i, then the state S_i can also be written as $S_i = C_i \vee \bar{l}_i \mid L_i, l^d, L_i', l_i, M_i' \vdash F$ such that $\overline{C_i} \cap \{l, L_i'\} = \emptyset$ and L_i' contains no decision literals. Then $L_i \cap \{l_i, \bar{l}_i\} = \emptyset$ and $\{C_i\}|_{L_i} = \{\emptyset\}$, which means that the sub-proof can be closed by applying the Back-and-Learn rule:

$$\vdots$$

$$\mathsf{Back\text{-}and\text{-}Learn}(\bar{l}_i, l) \qquad \frac{C_i \vee \bar{l}_i \mid L_i, l^d, L_i', l_i, M_i' \vdash F}{L_i, \bar{l}_i \vdash F, C_i \vee \bar{l}_i}$$

On the other hand, if there are literals in C_i with the same decision level as l_i (note that C_i cannot contain literals with higher decision levels than l_i because $\overline{C_i} \cap M_i' = \emptyset$), then l_i must be a propagated literal, which means that there must be a clause $C_i' \vee l_i$ entailed by F such that $\{C_i'\}|_{M_i} = \{\emptyset\}$, as otherwise the literal l_i could not have been propagated. Thus the Resolve rule can be applied to the state S_i:

$$\begin{array}{ll} & \vdots \\ \mathsf{Resolve}(C_i', l_i) & \dfrac{C_i \vee \bar{l}_i \mid M_i, l_i, M_i' \vdash F}{C_i \vee C_i' \mid M_i, l_i, M_i' \vdash F} \\ & \vdots \end{array}$$

By the soundness of resolution, $F \models C_i \vee C_i'$. Further, $\{C_i \vee C_i'\}|_{M_i} = \{\emptyset\}$ and $\overline{C_i \vee C_i'} \cap \{l_i, M_i'\} = \emptyset$. This means that the result of the Resolve step can be written in the form $S_{i+1} = C_{i+1} \vee \bar{l}_{i+1} \mid M_{i+1}, l_{i+1}, M_{i+1}' \vdash F$ such that l_{i+1} is the last literal in M whose complement occurs in $C_i \vee C_i'$ and S_{i+1} fulfills the invariant $(*)$. The above Resolve step can therefore be written as

$$\begin{array}{ll} & \vdots \\ \mathsf{Resolve}(C_i', l_i) & \dfrac{C_i \vee \bar{l}_i \mid M_i, l_i, M_i' \vdash F}{C_{i+1} \vee \bar{l}_{i+1} \mid M_{i+1}, l_{i+1}, M_{i+1}' \vdash F} \\ & \vdots \end{array}$$

All literals of $C_i \vee C_i'$ precede l_i in the context M, hence l_{i+1} also precedes l_i in M. On the other hand, the clause $C_i \vee C_i'$ still contains a literal with the same decision level as l_i. Thus in any sequence $S_0, S_1, S_2, \ldots$ of states generated as described above by an initial Conflict step and a sequence of Resolve steps there is a state S_k such that no further Resolve step, but only a Back-and-Learn step closing the sub-proof can be applied to S_k. □

2.3.3 Conflict resolution using the conflict graph

The proof of the preceding lemma describes an effective strategy for resolving conflicts occurring in a CDBL proof. How this strategy works can be clearly visualized by means of the *conflict graph.*

Definition 12. Consider the following derivation of a conflict in the CDBL calculus:

$$\begin{array}{ll} & \vdots \\ \mathsf{Conflict}(C_0 \vee \bar{l}_0) & \dfrac{M \vdash F}{C_0 \vee \bar{l}_0 \mid M_0, l_0, M_0' \vdash F}, \end{array}$$

where $M = M_0, l_0, M_0'$ and l_0 is the last literal in M whose complement occurs in the clause $C_0 \vee \bar{l}_0$. Then $F \models C_0 \vee \bar{l}_0$, $\{C_0\}|_{M_0} = \{\emptyset\}$, and $\overline{C_0} \cap M_0' = \emptyset$. A *conflict graph* associated with this conflict is a minimal directed graph G, whose nodes are literals, satisfying the following conditions:

1. The graph G contains the nodes $l_0, \bar{l}_0$, and the edge $l_0 \to \bar{l}_0$.
2. Let $C_0 = l_1 \vee \cdots \vee l_k$. Because of $\{C_0\}|_{M_0} = \{\emptyset\}$, the context M_0 must contain the literals $\bar{l}_1, \ldots, \bar{l}_k$. For every $i \in \{1, \ldots, k\}$ the graph G contains the node $\bar{l}_i$ and the edge $\bar{l}_i \to \bar{l}_0$.
3. For every node l of G that is a propagated literal in M let $C_l = l_1 \vee \cdots \vee l_k \vee l$ be a clause in F such that the literals $\bar{l}_1, \ldots, \bar{l}_k$ precede l in M. The formula F must contain such a clause, as otherwise l could not have been propagated. For every $i \in \{1, \ldots, k\}$ the graph G contains the node $\bar{l}_i$ and the edge $\bar{l}_i \to l$.

A conflict graph encodes dependencies between the literals in the context that are related to the conflict to be analyzed. As decision literals do not depend on other literals, every decision literal becomes a node without incoming edges in the conflict graph. For every propagated literal, however, there must be a reason clause that caused the propagation of this literal. The propagated literal therefore depends on the complements of the other literals in the reason clause, which must precede it in the context. In the conflict graph this dependency is expressed by edges running from the literals it depends on to the propagated literal. Since there can be more than one reason for every propagation, many conflict graphs can be associated with a single conflict. The choice of a particular graph is part of the strategy of an implementation of the calculus.

Using the conflict graph the derivation of an implied clause by linear resolution can be illustrated in the following way: Each of the clauses, denoted by $C_i \vee \bar{l}_i$ in the proof of lemma 11, derived by an application of either the Conflict rule or the Resolve rule corresponds to a subset R_i of the nodes of the conflict graph G satisfying the following conditions:

1. R_i contains the node l_0,

2. R_i does not contain a node that is a decision literal in M, and

3. no node contained in R_i occurs in the context M at a smaller position than a node of G not contained in R_i.

The clause $C_i \vee \bar{l}_i$, corresponding to the set R_i, is the set of the complements of the start nodes of edges incoming to R_i, *i.e.*, edges $l \rightarrow l'$ of the conflict graph such that $l \notin R_i$ and $l' \in R_i$. Every Resolve step can be viewed as an extension of the set R_i by one node of the conflict graph in the reverse order of their occurrence in the context M, resulting in the set R_{i+1}.

The resolution process can be finished (corresponding to an application of the Back-and-Learn rule) when the set of start nodes of edges incoming to R_i contains only a single literal that has the same decision level as l_0 (note that all literals in the set R_i have the same decision level, as R_i is never extended by a decision literal). Such a literal is referred to as a *unique implication point* of the conflict graph. A unique implication point has the property that it is contained in every path in the conflict graph running from the decision literal with the maximal decision level to the literals l_0 or $\bar{l}_0$. Every conflict graph contains at least one unique implication point because the decision literal with the maximal decision level in the conflict graph is a unique implication point.[6] This guarantees that the resolution process eventually terminates. The end of the resolution derivation at a unique implication point coincides with an application of the Back-and-Learn rule in the proof.

Figure 1 shows an example of a conflict graph. It belongs to a proof state $M \vdash F$ where the formula F contains the clauses $C_1 = (\bar{c} \vee \bar{d} \vee e)$, $C_2 = (\bar{a} \vee \bar{e} \vee f)$, $C_3 = (\bar{b} \vee \bar{e} \vee g)$, $C_4 = (\bar{f} \vee \bar{g} \vee h)$, $C_5 = (\bar{f} \vee \bar{g} \vee \bar{h})$, and the context M has the form $M = a^{(1)} \dots b^{(3)} \dots c^{(4)} \dots d^{(6)} \dots e \dots f \dots g \dots h$ (decision literals are annotated with their decision levels). The formula F and the context M are conflicting because $\{C_5\}|_M = \{\emptyset\}$. The shaded areas contain (in order of decreasing lightness) the node sets that correspond to the clauses $(\bar{f} \vee \bar{g} \vee \bar{h})$,

[6] A conflict graph may also contain more than one unique implication point, which adds a degree of freedom to the calculus, but makes the term "unique" a bit deceptive.

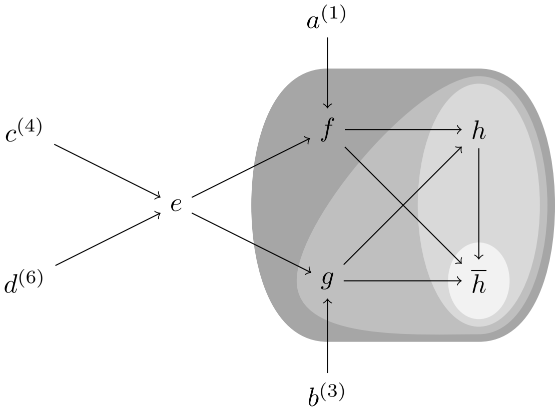

Figure 1: Example of a conflict graph. The current context contains the decision literals a, b, c, d, and the propagated literals e, f, g, h. A conflict occurred because the formula to be proved contains the clause $(\overline{f} \vee \overline{g} \vee \overline{h})$, which is falsified by the current context. The shaded areas contain (in order of decreasing lightness) the node sets that correspond to the clauses $(\overline{f} \vee \overline{g} \vee \overline{h})$, $(\overline{f} \vee \overline{g})$, $(\overline{b} \vee \overline{e} \vee \overline{f})$, and $(\overline{a} \vee \overline{b} \vee \overline{e})$, which are successively derived in the linear resolution steps until the unique implication point e is reached.

$(\overline{f} \vee \overline{g})$, $(\overline{b} \vee \overline{e} \vee \overline{f})$, and $(\overline{a} \vee \overline{b} \vee \overline{e})$, which are successively derived in the linear resolution steps. The literal e is a unique implication point of this conflict graph because all paths from the node $d^{(6)}$ to one of the nodes h and $\overline{h}$ run through it. Equivalently, it is the only literal in the set $\{a, b, e\}$, corresponding to the derived clause $(\overline{a} \vee \overline{b} \vee \overline{e})$, that has the same decision level as the literal $\overline{h}$.

2.3.4 Proof confluence and completeness of CDBL

The following lemma shows that as long as a CDBL proof has not been closed, there is always a rule of the calculus that can be applied.

Lemma 13. *If $M \vdash F$ has been derived in the CDBL calculus and $F|_M \neq \emptyset$, then at least one of the rules of the calculus can be applied to this state.*

Proof. Assume that M falsifies F. Then F contains a clause C such that $\{C\}|_M = \{\emptyset\}$. If all elements of M whose complements are contained in C have decision level 0, then the Unsatisfiable rule can be applied. Otherwise, a sub-proof starting with a Conflict step, followed by a (possibly empty) sequence of Resolve steps, and ending with a Back-and-Learn step can be derived from $M \vdash F$, as shown in the proof of lemma 11.

On the other hand, assume that M does not falsify F. Because $F|_M \neq \emptyset$, F must contain at least one literal l such that $M \cap \{l, \overline{l}\} = \emptyset$. If F contains a clause C such that $\{C\}|_M = \{l'\}$ for some literal l', then the Propagate rule can be applied. If F contains no such clause, then the Decide rule can be applied. □

Note again that the proofs of lemmas 11 and 13 are constructive and therefore immediately suggest an algorithm for deciding the satisfiability of propositional clause sets by creating derivations in the CDBL calculus. Now we can conclude that the CDBL calculus is a proof confluent and complete calculus for clausal propositional logic.

Proposition 14 (Proof confluence and completeness)**.** *Let $\emptyset \vdash F, \ldots, M \vdash F'$ be a derivation in the CDBL calculus. If F is unsatisfiable, then $\Box$ can be derived from $M \vdash F'$. If F is satisfiable, then $M' \vdash F''$ can be derived from $M \vdash F'$ such that $F''|_{M'} = \emptyset$.*

Proof. In the representation of a CDBL proof in the LD calculus every Learn step is accompanied by a Back step. Hence it is not possible to create a CDBL derivation whose representation in the LD calculus contains an infinite sequence of consecutive Learn and Delete steps. By lemma 10 and propositions 3 and 4, the CDBL calculus is sound and terminating. By lemma 13, the derivation $\emptyset \vdash F, \ldots, M \vdash F'$ can be continued as long as the proof is not closed, and because the calculus is terminating, it will eventually be closed. By the soundness of the calculus, the result will be $\Box$ if F is unsatisfiable, and it will be a state $M' \vdash F''$ such that $F''|_{M'} = \emptyset$ if F is satisfiable (where $M' = M$ in case the proof had already been closed). $\Box$

Acknowledgment. I wish to thank Tim Richter for carefully reading a preliminary version of this paper and for providing many comments on how to improve it.

References

1. Michael Alekhnovich, Jan Johannsen, Toniann Pitassi, and Alasdair Urquhart. An exponential separation between regular and general resolution. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 448–456. ACM, 2002.

2. Peter Baumgartner and Cesare Tinelli. The model evolution calculus. In Franz Baader, editor, *Automated Deduction — CADE-19, 19th International Conference on Automated Deduction, Proceedings*, volume 2741 of *Lecture Notes in Computer Science*, pages 350–364. Springer, 2003.

3. Paul Beame, Henry Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.

4. Armin Biere. The evolution from Limmat to NanoSAT. Technical Report 444, Department of Computer Science, ETH Zürich, 2004. NanoSAT is available at `http://fmv.jku.at/nanosat`.

5. Joshua Buresh-Oppenheim and Toniann Pitassi. The complexity of resolution refinements. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, pages 138–147. IEEE Computer Society, 2003.

6. Martin Davis and Hillary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

7. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

8. Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, Proceedings*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003. MiniSat is available at `http://www.minisat.se`.

9. Allen Van Gelder. Pool resolution and its relation to regular resolution and DPLL with clause learning. In Geoff Sutcliffe and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence and Reasoning, 12th International Conference, Proceedings*, volume 3835 of *Lecture Notes in Computer Science*, pages 580–594. Springer, 2005.

10. Karl Lieberherr. Complexity of superresolution. *Notices of the American Mathematical Society*, 24:A–433, 1977.

11. Inês Lynce and João P. Marques-Silva. Efficient data structures for backtrack search SAT solvers. *Annals of Mathematics and Artificial Intelligence*, 43(1):137–152, 2005.

12. João P. Marques-Silva and Karem A. Sakallah. GRASP — a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227. IEEE Computer Society, 1996.

13. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001*, pages 530–535. ACM, 2001.

14. Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Abstract DPLL and abstract DPLL modulo theories. In Franz Baader and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence and Reasoning, 11th International Conference, Proceedings*, volume 3452 of *Lecture Notes in Computer Science*, pages 36–50. Springer, 2005.

15. Lintao Zhang. *Searching for Truth: Techniques for Satisfiability of Boolean Formulas.* PhD thesis, Princeton University, Department of Electrical Engineering, 2003.

16. Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design*, pages 279–285. IEEE Press, 2001. zChaff is available at `http://www.princeton.edu/~chaff/zchaff.html`.