# An Exponential Separation Between Deterministic CDCL and DPLL Solvers

**Sahil Samar**
School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA

**Marc Vinyals**
School of Computer Science, University of Auckland, New Zealand

**Vijay Ganesh**
School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA

## Abstract

We prove that there exists a deterministic configuration of Conflict-Driven Clause Learning (CDCL) SAT solvers using a variant of the VSIDS branching heuristic that solves instances of the Ordering Principle (OP) CNF formulas in time polynomial in $n$, where $n$ is the number of variables in such formulas. Since tree-like resolution is known to have an exponential lower bound for proof size for OP formulas, it follows that CDCL under this configuration has an exponential separation with any solver that is polynomially equivalent to tree-like resolution and therefore any configuration of DPLL SAT solvers.

**2012 ACM Subject Classification** Theory of computation → Logic

**Keywords and phrases** SAT solvers, CDCL, Proof Systems, VSIDS

**Digital Object Identifier** 10.4230/LIPIcs.SAT.2026.30

**Related Version** *Full Version*: `https://arxiv.org/abs/2603.16156` [19]

**Acknowledgements** We would like to thank Vincent Liew for his help in verifying the work in this paper. Additionally, we want to thank the anonymous reviewers for their valuable feedback.

## 1 Introduction

Although SAT is the quintessential NP-complete problem, modern SAT solvers routinely handle real-world instances with millions of variables and clauses [11]. Much of this success is attributable to clause learning [20] and conflict-driven branching techniques such as VSIDS [15] and LRB [14], reinforced by extensive benchmarking and ablation studies [12, 9]. A natural question is whether this empirical success has a theoretical explanation.

The best theoretical evidence to date comes from equivalences to proof systems: assuming optimal heuristics, DPLL is polynomially equivalent to tree-like resolution, while CDCL is polynomially equivalent to general resolution [17, 2]. Since there exist formulas that have polynomially long resolution proofs but require exponentially long tree-like resolution proofs, as a result of the separation between proof systems we obtain an exponential separation between algorithms.

But do these results explain the efficiency of *practical* solvers? The equivalence and separation results only apply to nondeterministic algorithms using optimal heuristics given by an unrealistic oracle. The fact that a short proof exists does not mean that a deterministic algorithm is necessarily able to find it. Whether that is possible is called the automatability problem [7], and it is an active area of study. In the case of resolution, a breakthrough result proved that no deterministic algorithm can always find short resolution proofs unless $\mathsf{P} = \mathsf{NP}$ [3].

More precisely, only one heuristic in CDCL needs to be nondeterministic for it to simulate resolution: the branching heuristic, which determines the next variable to branch on. A randomized heuristic is enough for CDCL to simulate bounded-width resolution [2], and if

the heuristic is fixed and deterministic then it is known unconditionally – independently of whether $\mathsf{P} = \mathsf{NP}$ – that CDCL cannot always find short resolution proofs when the branching heuristic is VSIDS-like [22] or ordered [16].

This leaves open the central question we address: is there an exponential separation between DPLL and CDCL when CDCL uses a practical, fully deterministic branching heuristic?

Two families of formulas that exponentially separate tree-like and general resolution, and therefore are candidates to answer our question, are pebbling formulas and the Ordering Principle. The first family of formulas has been studied in a proof system more closely resembling CDCL [10], but still assuming an optimal decision heuristic. The second family is what concerns us.

The complexity of resolution proofs of the ordering principle has been widely studied. The formulas were introduced as a *tricky* family that could be solved using certain symmetry breaking properties but appeared hard for resolution [13]. They were later shown to have in fact short resolution proofs [21] but to require exponentially long tree-like resolution proofs [6]. Furthermore, a modified version of these formulas was used to exponentially separate regular from general resolution [1].

What makes these formulas tricky is that any resolution proofs require learning clauses containing many variables, which is usually an indicator of hardness. In a well-defined sense, these formulas are just at the boundary of formulas that are easy versus hard for resolution: a formula over $n$ variables that requires clauses containing $\Omega(\sqrt{n})$ variables to prove, such as the Ordering Principle, requires exponentially long tree-like resolution proofs, and a formula whose proof requires clauses containing $\Omega(n^{1/2+\epsilon})$ variables requires exponentially long resolution proofs. This makes the Ordering Principle a very interesting case study for the capabilities of any resolution-based algorithm. Adding to the mystery, empirical work shows that the Glucose SAT solver with VSIDS decay factor 0.6 solves OP instances in polynomial time, but not with a decay factor of 0.95 [9].

In this paper, we prove that CDCL with VSIDS solves the Ordering Principle in polynomial time. To our knowledge, this is the first result showing a polynomial upper bound for a completely deterministic configuration of CDCL solvers, simultaneously exponentially separating it from all configurations of DPLL SAT solvers. To be more precise, we use VSIDS with decay factor at most 1/2 (which is equivalent to a variant of VMTF), fixed phase value selection, restarts after every conflict, no clause deletion, and 1UIP clause learning.

Specifying a deterministic CDCL configuration requires a tie-breaking rule for VSIDS, and any such rule must appeal to some ordering on variables. The natural source of this ordering is the DIMACS representation of the formula, which is what real solvers receive as input. Our analysis assumes a column-major variable ordering; to extend the result to arbitrary DIMACS representations, we equip the solver with a deterministic polynomial time pre-processor that recovers this ordering, making the solver's behavior invariant to the choice of representation. This invariance is what licenses our stating the results in terms of abstract OP formulas: under it, analyzing the solver on the abstract formula is equivalent to analyzing it on every DIMACS representation.

An immediate corollary of our result, given that there is an exponential lower bound on the size of tree-like resolution proofs of the Ordering Principle, and that the DPLL algorithm is captured by tree-like resolution, is that there is an exponential separation between deterministic CDCL and nondeterministic DPLL.

# 2 Preliminaries

## 2.1 Ordering Principle

$$A_n = \bigwedge_{\substack{1\le i,j,k\le n \\ i\neq j, k\neq i, j\neq k}} (\neg P_{i,j} \vee \neg P_{j,k} \vee P_{i,k}), \qquad \text{(transitivity)}$$

$$B_n = \bigwedge_{\substack{1\le i,j\le n \\ i\neq j}} (\neg P_{i,j} \vee \neg P_{j,i}), \qquad \text{(antisymmetry)}$$

$$D_n = \bigwedge_{1\le j\le n} \bigvee_{\substack{1\le i\le n \\ i\neq j}} P_{i,j} \qquad \text{(non-minimality)}$$

$$A(i,j,k) = \neg P_{i,j} \vee \neg P_{j,k} \vee P_{i,k} \qquad \text{(clause in } A_n\text{)}$$

$$B(i,j) = \neg P_{i,j} \vee \neg P_{j,i} \qquad \text{(clause in } B_n\text{)}$$

$$D(j) = \bigvee_{\substack{1\le i\le n \\ i\neq j}} P_{i,j} \qquad \text{(clause in } D_n\text{)}$$

An Ordering Principle instance $OP_n$ is defined as $A_n \wedge B_n \wedge D_n$. In particular, the variables in the formula are $P_{i,j}$ s.t. $1 \le i, j \le n$ and $i \neq j$. For a variable $P_{i,j}$, we often will call $i$ the row and $j$ the column. An $OP_n$ instance can be thought of as describing $n$ elements that are ordered in some way where $P_{i,j}$ encodes that element $i$ is smaller than element $j$, and such that the elements satisfy transitivity and antisymmetry conditions, but also with the property that no element is minimal (and thus, the formula is UNSAT).

## 2.2 VSIDS

First introduced by the authors of the Chaff solver [15], VSIDS (Variable State Independent Decaying Sum) is a dynamic branching heuristic for SAT solvers that, upon learning a conflict clause, assigns a score to each variable that appears in the conflict clause (or some variation of the implication graph), decays the scores of all variables at regular intervals, and then branches on the highest scoring variable not currently on the assignment trail. There have been many variants of the heuristic since its first introduction.

The variant of VSIDS that we consider is identical to that used in the first version of MiniSAT [8], i.e., it scores *variables* not literals, bumps the variables in the *final* learned clause, rather than all or some subset of clauses involved in conflict analysis, and decays all variable scores by a multiplicative factor *after each conflict*. There is a single hyperparameter for the heuristic, which is the decay factor $\delta$. The scores are all initialized to 0, and updated after each conflict.

Let the score of variable $x$ after conflict $t$ be $q(x,t)$. Then, the update formula is:

$$q(x,t) = b(x,t) + \delta q(x,t-1)$$

where $b(x,t) = 1$ if variable $x$ participated in conflict $t$ and $b(x,t) = 0$ otherwise. A variable "participated in a conflict" if it was a part of the resulting learned clause of the conflict.

It is also well known that VSIDS with decay factor at most $1/2$ is equivalent to the Variable Move to Front (VMTF) branching heuristic [18, 5]. Thus, we can replace VSIDS with VMTF for this proof, and the argument would work the same.

## 2.3 CDCL Configuration

The SAT solver configuration we analyze in this paper is the following:

- VSIDS Branching heuristic (as described in the previous section) with decay factor $\delta \leq 1/2$
- Fixed phase value selection: when making decisions, always assign variables to false.
- Restart after every conflict
- No clause deletion scheme
- 1UIP Clause Learning scheme

We denote the database of learned clauses by $\Gamma$. We assume that at any point during the solver run, $\Gamma$ contains all of the clauses the solver learned up to that point in the same order in which they were learned.

We will briefly discuss the rationale behind the configuration. Our proof heavily uses the fact that the decay factor is at most $1/2$, which is perhaps not surprising in view of how an aggressive decay factor has also been experimentally observed to be required to solve instances of the Ordering Principle (as discussed in the introduction). Most assumptions we make about the configuration are inspired by practice or fairly standard in theoretical papers, except perhaps fixed phase value selection. Frequent restarts and no clause deletions are required in the existing proofs that nondeterministic CDCL simulates resolution [17] and CDCL with randomized branching simulates bounded-width resolution [2], while 1UIP is the de-facto standard clause learning heuristic in modern CDCL SAT solvers. We want to point out that the separation proved in this paper may still hold without restarts; this is left to future work.

The only detail left to specify for the solver is how we break ties when there is more than one variable with the highest VSIDS score. We define the tie-break rule as follows: Let the variables $P_{i,j}$ of the OP instance be in column-major order, i.e.

$$P_{2,1}, ..., P_{n,1}, P_{1,2}, ..., P_{n,2}, ..., P_{1,n}, ..., P_{n-1,n}$$

When branching, if there is a tie for the highest VSIDS score, we break the tie based on this ordering (i.e. the variable chosen is the one with the highest score and whichever comes first in the above ordering).

The tie-breaking rule arises naturally from treating the solver's input as a DIMACS representation rather than an abstract propositional formula, a perspective we develop in Section 2.4 as the appropriate setting for analyzing deterministic solvers. Under this view, the variable ordering in the representation determines the tie-breaking rule. To make the solver's behavior invariant to the choice of representation, Section 2.5 presents a polynomial time pre-processing algorithm that makes the solver behave identically on all valid DIMACS representations of the same formula. All results in Section 3 assume this pre-processor is applied.

## 2.4 DIMACS Representation

In practice, SAT solvers take in DIMACS representations of CNF boolean formulas as input. DIMACS representations map each variable to a unique number, encode each clause as a sequence of numbers corresponding to the variables in the clause, and list each clause sequentially. Thus, a given DIMACS representation gives a formula additional structure in 3 main ways: an ordering of the variables, an ordering of the clauses, and an ordering of variables within clauses. When studying the complexity of deterministic SAT solvers, it

makes sense to consider this structure that SAT solvers have access to in practice. Thus, we propose that the theoretical treatment of deterministic SAT solvers should have the input be DIMACS representations rather than arbitrary propositional formulas. Then, the deterministic solver can be analyzed with the structure that those representations impose on the formulas. In many cases, simply changing the representation of a formula can have a significant impact on the solver behavior [4], and so we find it natural to treat solvers in this way.

For the purposes of this paper, we are primarily concerned with the VSIDS branching heuristic, and thus the aspect of the DIMACS representation that we care about is the ordering of the variables. In particular, when solving OP instances, if VSIDS scores are tied we wish to choose variables in column-major order. So, the DIMACS representation that we consider maps the variables by enumerating them in column-major order (i.e. assign $P_{2,1}$ to 1, $P_{3,1}$ to 2, and so on); any ordering of clauses and any ordering of variables within clauses is allowed. Given this representation, the column-major tie-breaking rule is simply the rule of choosing the lowest index variable – which is the standard rule used in practice by solvers like Cadical or Kissat [5]. Next, we discuss a way to recover the column-major variable ordering from an arbitrary DIMACS representation of OP.

## 2.5 Pre-processing

For some classes of formulas, a polynomial time pre-processing algorithm can (using only the inherent structure of the formulas) derive enough information such that the solver behaves the same way regardless of DIMACS representation. We describe such a pre-processing algorithm for recovering the column-major variable ordering for the Ordering Principle.

Consider an arbitrary CNF formula $F$ (not necessarily an OP formula). Below we give a polytime pre-processor $P$ that takes as input $F$ and produces as output a variable ordering for $F$ with the following property: if $F$ happens to be an OP formula, then $P$ constructs a valid labeling of its variables and returns the column-major order with respect to that labeling; otherwise, $P$ returns an arbitrary order.

1. First, sort the clauses in ascending order by the number of variables in them.
2. Next, let the number of total variables be $N$. Solve for $n \cdot (n-1) = N$. If $n$ is not an integer, return an arbitrary order. Otherwise, proceed.
3. Now, consider the $n$ largest clauses. If there are less than $n$ clauses, then return an arbitrary order. For a valid OP instance, these are the non-minimality clauses; each one corresponds to a distinct column. Prescribe to each of those $n$ clauses a unique index $j$, $1 \leq j \leq n$. Assign column index $j$ to all variables appearing in clause $j$. For an OP instance, each variable in the formula appears in exactly one of these clauses (if not, return an arbitrary order). Now, every variable has a valid column index assigned.
4. Next, consider the $n \cdot (n-1)/2$ smallest clauses. If there are not enough clauses, return an arbitrary order. For a valid OP instance, these are the antisymmetry clauses; furthermore, each variable in the formula should appear in exactly one of these clauses. Consider one of these clauses, which is of the form $\neg x_{i_1,j} \vee \neg x_{i_2,k}$, where $i_1$ and $i_2$ are the respective unknown row indices and $j$ and $k$ are the known column indices from the previous step. Now, we assign $i_1$ to be $k$ and $i_2$ to be $j$. We iterate over all of these clauses and repeat this step. If any of these clauses has more than two variables or there is a collision, return an arbitrary order. Now, for a valid OP instance, every variable has both a valid row and valid column index assigned.
5. Finally, now that each variable has a valid row and valid column index, we return the column-major ordering of the variables based on that labeling.

Augmenting the CDCL solver with this pre-processor makes its behavior invariant to the choice of DIMACS representation for OP formulas: on any two representations of the same OP instance, the solver executes identically. This invariance is what justifies stating the results in Section 3 in terms of abstract OP formulas rather than specific DIMACS representations. More precisely, once a deterministic solver is invariant to representation, analyzing it on the abstract formula is equivalent to analyzing it on any (and hence every) DIMACS representation of that formula.

## 3 Main Result

We show that CDCL can determine the unsatisfiability of the Ordering Principle in polynomial time by a direct analysis of the behavior of the algorithm. That is, for each clause that the algorithm learns we show which decisions and unit propagations the solver does until reaching a conflict, building an implication graph along the way. Then we show which clause the conflict analysis heuristic learns when applied to said implication graph.

We first prove a useful invariant about the VSIDS branching heuristic under the CDCL configuration described in Section 2.3.

▶ **Lemma 1** (Focus Lemma). *Assume that the solver has learned at least one clause. Consider the set $S$ of variables in the most recent learned clause. During a run, when the solver has to branch, it must branch on an unassigned variable in $S$ and assign it to false, before it can branch on any other variables.*

**Proof.** First, observe that the variables in the most recently learned clause have a VSIDS score of at least 1. The reason is that their scores are increased by a value of 1 after the clause was learned by definition of VSIDS. Variables not in the most recent learned clause have VSIDS score at most $\sum_{i=1}^{\infty} \delta^i = \frac{\delta}{1-\delta}$. Observe that $\frac{\delta}{1-\delta} < 1$ for $\delta < \frac{1}{2}$. Furthermore, for any finite $k \in \mathbb{N}$, $\sum_{i=1}^{k} \frac{1}{2}^i < 1$. So, the variables in the most recent learned clause all have higher VSIDS score than any variable not in the most recent learned clause for $\delta \leq 1/2$. Recall that we assume the solver restarts after every conflict. Therefore, upon branching, the solver branches on unassigned variables in the newest learned clause first, before branching on any other variables. Finally, because of fixed phase value selection, the solver assigns the (unassigned) variables in the most recent learned clause to false. ◀

Informally, what the Focus Lemma says is that whenever the solver has to make a decision during a run, the solver cannot branch on a variable which does not appear in the newest learned clause *if there exists at least one unassigned variable in the newest learned clause.* Note that it could be the case that a variable in the most recent learned clause is already propagated to true before a conflict is derived, and the solver thus didn't assign false to it. The Focus Lemma says nothing about this.

Additionally, observe that the Focus Lemma can be extended beyond just the newest learned clause. Under this configuration, we can view branching as ranking each variable based solely on the most recent learned clause it appears in (where higher rank means the variable appeared in a newer learned clause), and then picking the variable with highest rank (and using the tie-breaking rule if necessary). Thus, the Focus Lemma proves the generally known equivalence between VSIDS with decay factor at most 1/2 and VMTF (while also additionally prescribing the polarity of variable assignment due to the solver configuration).

▶ **Definition 2** (Ordered Variable Sequence and Prefix Clause). *For any column $j$, let $\mathcal{L}_j$ be the ordered sequence of variables in that column: $\mathcal{L}_j = \langle P_{1,j}, P_{2,j}, \ldots, P_{j-1,j}, P_{j+1,j}, \ldots, P_{n,j} \rangle$. Note that for $j = 1$, variable $P_{1,1}$ does not exist, so the sequence starts at $P_{2,1}$. Let $C(j,k)$ be the disjunction of the first $k$ variables in $\mathcal{L}_j$. Note that $D(j) = C(j, n-1)$.*

▶ **Theorem 3.** *For the CDCL solver configuration in Section 2.3, the solver always learns exactly the following clauses in this exact order on $OP_n$ instances for $n \geq 6$:*

***Head***

$C(1, n-2)$
$C(1, n-3)$

***Descending Cascade***

*For each $j$ descending from $n-2$ to 2 :*

$C(j, n-2)$
$C(j, n-3)$
$\vdots$
$C(j, j-1)$

*A triangular block of depth $n-j$.*
*Starts at length $n-2$, ends at length $j-1$.*

***Tail***

$C(1, n-4)$
$\vdots$
$C(1, 2)$

*Resumes column 1.*
*Ends with the pair $P_{2,1} \vee P_{3,1}$.*

To illustrate the pattern, for $OP_6$ (left) and $OP_7$ (right) the learned clauses are:

$P_{2,1}\ P_{3,1}\ P_{4,1}\ P_{5,1}$
$P_{2,1}\ P_{3,1}\ P_{4,1}$

$P_{1,4}\ P_{2,4}\ P_{3,4}\ P_{5,4}$
$P_{1,4}\ P_{2,4}\ P_{3,4}$
$P_{1,3}\ P_{2,3}\ P_{4,3}\ P_{5,3}$
$P_{1,3}\ P_{2,3}\ P_{4,3}$
$P_{1,3}\ P_{2,3}$
$P_{1,2}\ P_{3,2}\ P_{4,2}\ P_{5,2}$
$P_{1,2}\ P_{3,2}\ P_{4,2}$
$P_{1,2}\ P_{3,2}$
$P_{1,2}$

$P_{2,1}\ P_{3,1}$

$P_{2,1}\ P_{3,1}\ P_{4,1}\ P_{5,1}\ P_{6,1}$
$P_{2,1}\ P_{3,1}\ P_{4,1}\ P_{5,1}$

$P_{1,5}\ P_{2,5}\ P_{3,5}\ P_{4,5}\ P_{6,5}$
$P_{1,5}\ P_{2,5}\ P_{3,5}\ P_{4,5}$
$P_{1,4}\ P_{2,4}\ P_{3,4}\ P_{5,4}\ P_{6,4}$
$P_{1,4}\ P_{2,4}\ P_{3,4}\ P_{5,4}$
$P_{1,4}\ P_{2,4}\ P_{3,4}$
$P_{1,3}\ P_{2,3}\ P_{4,3}\ P_{5,3}\ P_{6,3}$
$P_{1,3}\ P_{2,3}\ P_{4,3}\ P_{5,3}$
$P_{1,3}\ P_{2,3}\ P_{4,3}$
$P_{1,3}\ P_{2,3}$
$P_{1,2}\ P_{3,2}\ P_{4,2}\ P_{5,2}\ P_{6,2}$
$P_{1,2}\ P_{3,2}\ P_{4,2}\ P_{5,2}$
$P_{1,2}\ P_{3,2}\ P_{4,2}$
$P_{1,2}\ P_{3,2}$
$P_{1,2}$

$P_{2,1}\ P_{3,1}\ P_{4,1}$
$P_{2,1}\ P_{3,1}$

**Proof.** The structure of our proof is to show that the solver first learns the **Head**, then transitions to and learns the **Descending Cascade**, and finally transitions to and learns the **Tail** and derives UNSAT. We inductively show that, given the solver has learned some prefix of the learned clauses in the theorem statement, it correctly learns the next clause. We show the corresponding implication graphs to learned clauses, and for each graph we show the 1UIP node and cut. First, we state a key invariant:

▶ **Lemma 4** (Equal Score Invariant). *For the CDCL solver configuration in Section 2.3, when solving OP instances, for each learned clause $C$ one of the following holds: (1)* ***all*** *variables in $C$ participate in a learned clause for the first time in $C$, (2)* ***all*** *variables in $C$ participate in a previously learned clause $C'$ and in no clause learned after $C'$ but before $C$.*

*It follows that the variables in $C$ all have the same VSIDS score right after the clause $C$ has been learned and the scores have been bumped.*

A consequence of the **Equal Score Invariant** is that we branch on variables in the most recent learned clauses (by the Focus Lemma) in column-major order (due to the tie-breaking rule). We will show that the **Equal Score Invariant** is maintained for all of the learned clauses.

1: $A(2, n, 1)$
2: $D(1)$
3: $D(1)$
4: $A(n-1, n, 1)$
5: $A(2, n, 1)$
6: $A(n-1, n, 1)$
7: $B(1, n)$
8: $D(n)$
9: $D(n)$
10: $D(n)$

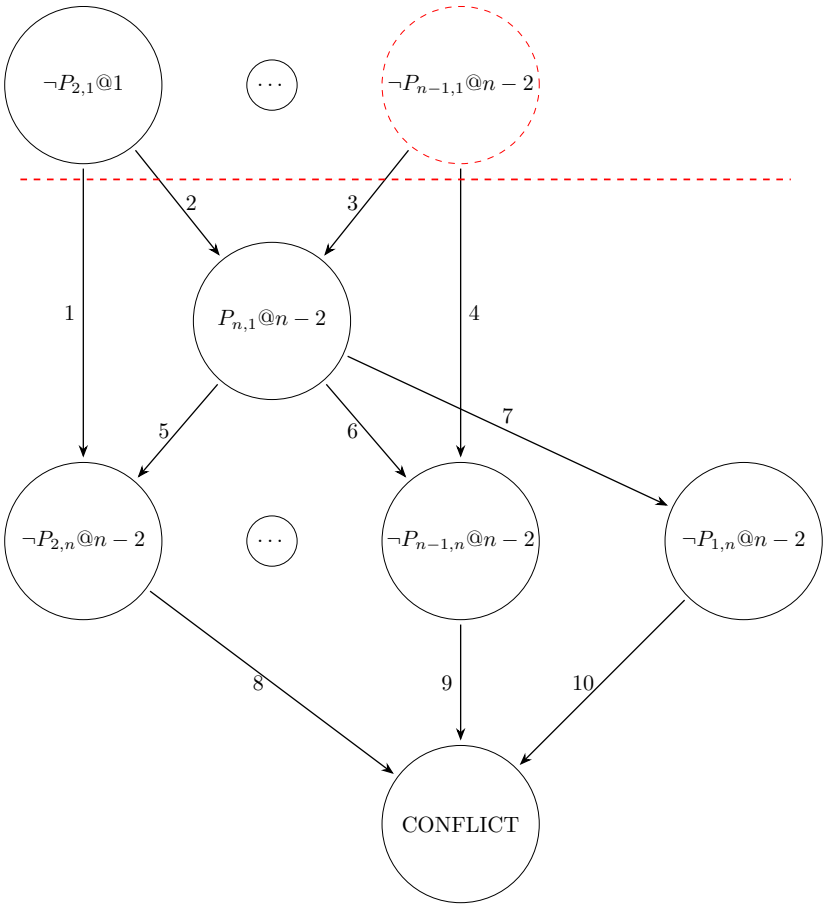


**Figure 1** First Head Clause Implication Graph. 1UIP node and cut in red.

At first, since $|\Gamma| = 0$ and all VSIDS scores are 0, the solver branches on the OP variables in column-major order; that is, the solver branches on $P_{2,1}, P_{3,1}, ..., P_{n-1,1}$ (and assigns them to false). After doing so, we get a conflict, and 1UIP derives the learned clause $C(1, n-2)$ (Figure 1). The **Equal Score Invariant** holds by Case 1. Then, by the Focus Lemma, the solver branches on $P_{2,1}, ..., P_{n-2,1}$ before any other variables. This then yields a conflict and 1UIP derives the learned clause $C(1, n-3)$ (Figure 2). Because each variable in this learned clause was also in the previous learned clause, the **Equal Score Invariant** is maintained. So, we see that the learned clause database always starts off with **Head**, and the **Equal Score Invariant** is maintained for the **Head** by Case 2.

1: $A(2, n-1, 1)$
2: $C(1, n-2)$
3: $C(1, n-2)$
4: $A(n-2, n-1, 1)$
5: $B(1, n-1)$
6: $A(2, n-1, 1)$
7: $A(n-2, n-1, 1)$
8: $A(1, n, n-1)$
9: $D(n-1)$
10: $A(2, n, n-1)$
11: $D(n-1)$
12: $D(n-1)$
13: $A(n-2, n, n-1)$
14: $A(1, n, n-1)$
15: $A(2, n, n-1)$
16: $A(n-2, n, n-1)$
17: $B(n-1, n)$
18: $D(n)$
19: $D(n)$
20: $D(n)$
21: $D(n)$

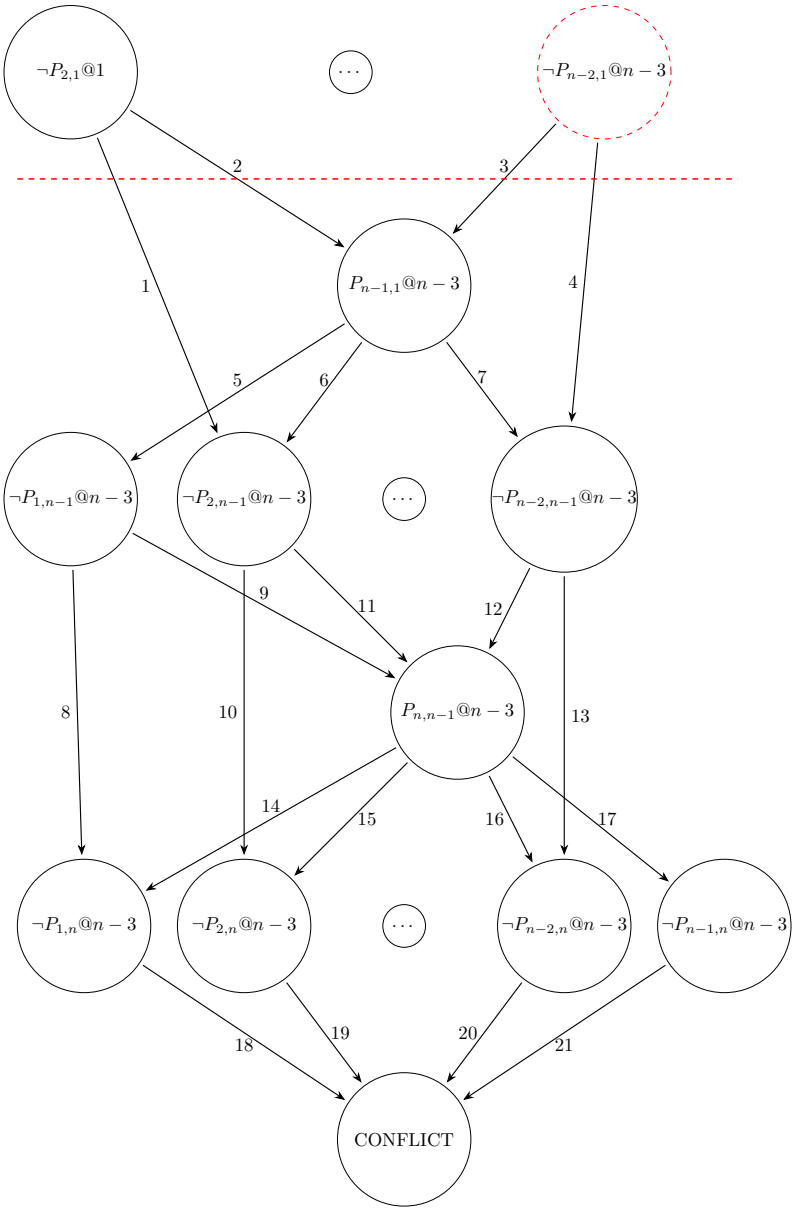


**Figure 2** Second Head Clause Implication Graph.

For the **Descending Cascade**, there are three facts that we show. The first is that the Descending Cascade actually *begins*, i.e., we learn $C(n-2, n-2)$ right after the head. The second is that we *cascade* within a single column, i.e., given that the solver has learned $C(j, n-2), ..., C(j, k)$ for $k > j-1$, the solver learns $C(j, k-1)$ next (Lemma 5). The third is that after learning $C(j, j-1)$ for $j > 2$, we *descend* to the next column and learn $C(j-1, n-2)$ (Lemma 6).

After learning the Head, by the Focus Lemma, we see that the solver next branches on $P_{2,1}, ..., P_{n-3,1}, P_{n-1,1}$. After doing so, 1UIP derives the learned clause $C(n-2, n-2)$ (Figure 3). None of the variables in the learned clause were involved in any of the previous learned clauses, so the **Equal Score Invariant** holds by case 1.

1: $A(2, n-2, 1)$
2: $C(1, n-3)$
3: $C(1, n-3)$
4: $A(n-3, n-2, 1)$
5: $A(n-1, n-2, 1)$
6: $B(1, n-2)$
7: $A(2, n-2, 1)$
8: $A(n-3, n-2, 1)$
9: $A(n-1, n-2, 1)$
10: $A(1, n, n-2)$
11: $D(n-2)$
12: $A(2, n, n-2)$
13: $D(n-2)$
14: $D(n-2)$
15: $A(n-3, n, n-2)$
16: $D(n-2)$
17: $A(n-1, n, n-2)$
18: $A(1, n, n-2)$
19: $A(2, n, n-2)$
20: $A(n-3, n, n-2)$
21: $A(n-1, n, n-2)$
22: $B(n-2, n)$
23: $D(n)$
24: $D(n)$
25: $D(n)$
26: $D(n)$
27: $D(n)$

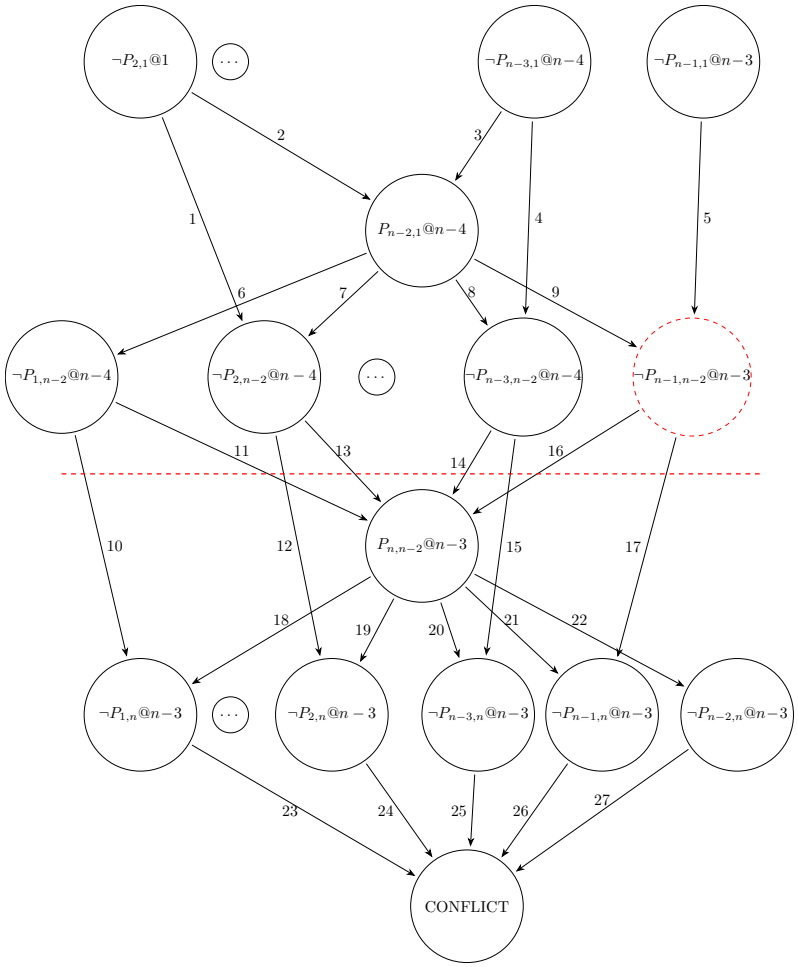


**Figure 3** First Descending Cascade Clause Implication Graph.

▶ **Lemma 5** (Cascade Lemma). *Assume that* $\Gamma$ *contains the* ***Head*** *clauses followed by* $C(j, n-2), ..., C(j, j-1)$ *for* $j = n-2, ..., l+1$*, and then* $C(l, n-2), ..., C(l, k)$ *for* $k > l-1$*. Additionally, assume that the* ***Equal Score Invariant*** *holds for all of those clauses. The next learned clause is* $C(l, k-1)$*, and the* ***Equal Score Invariant*** *still holds.*

There are 4 cases for the Cascade Lemma, shown below.

1: $A(1, k+1, l)$
2: $C(l, k)$
3: $A(l-1, k+1, l)$
4: $C(l, k)$
5: $C(l, k)$
6: $A(l+1, k+1, l)$
7: $C(l, k)$
8: $A(k, k+1, l)$
9: $A(1, k+1, l)$
10: $A(l-1, k+1, l)$
11: $A(l+1, k+1, l)$
12: $A(k, k+1, l)$
13: $B(l, k+1)$
14: $A(1, k+2, k+1)$
15: $C(k+1, k)$
16: $A(l-1, k+2, k+1)$
17: $C(k+1, k)$
18: $C(k+1, k)$
19: $A(l+1, k+2, k+1)$
20: $C(k+1, k)$
21: $A(k, k+2, k+1)$
22: $C(k+1, k)$
23: $A(l, k+2, k+1)$
24: $A(1, k+2, k+1)$
25: $A(l-1, k+2, k+1)$
26: $A(l, k+2, k+1)$
27: $A(l+1, k+2, k+1)$
28: $A(k, k+2, k+1)$
29: $B(k+1, k+2)$
30: $C(k+2, k+1)$
31: $C(k+2, k+1)$
32: $C(k+2, k+1)$
33: $C(k+2, k+1)$
34: $C(k+2, k+1)$
35: $C(k+2, k+1)$

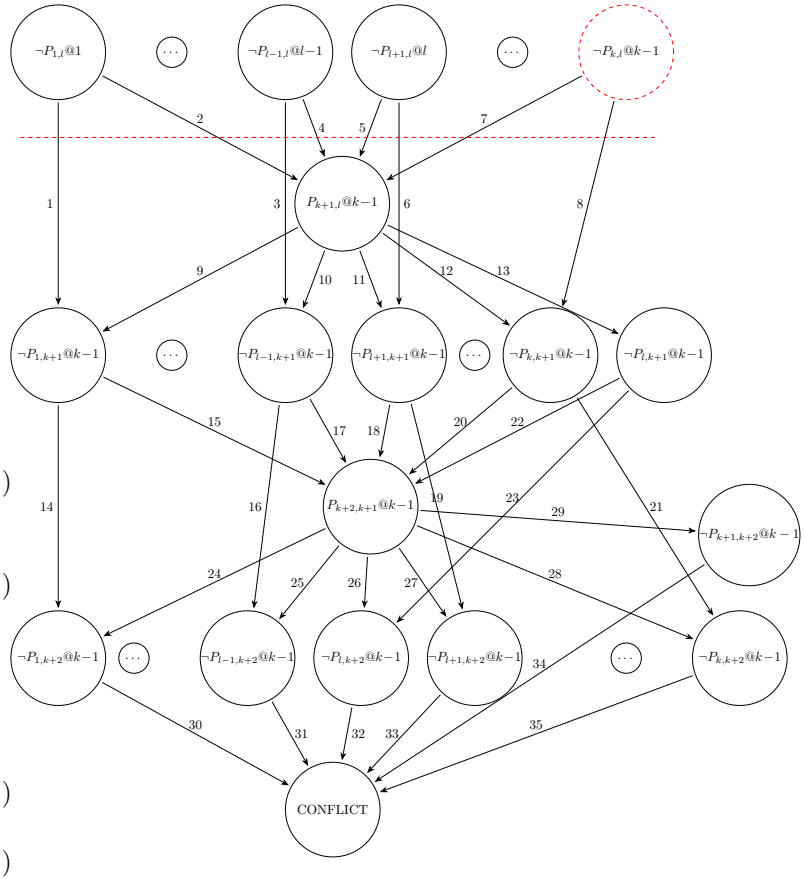


**Figure 4** Cascade Lemma ($k \neq l, k = n-2$) Implication Graph.

1: $A(1, k+1, l)$
2: $C(l, k)$
3: $C(l, k)$
4: $A(l-1, k+1, l)$
5: $A(1, k+1, l)$
6: $A(l-1, k+1, l)$
7: $B(l, k+1)$
8: $C(k+1, k)$
9: $C(k+1, k)$
10: $C(k+1, k)$

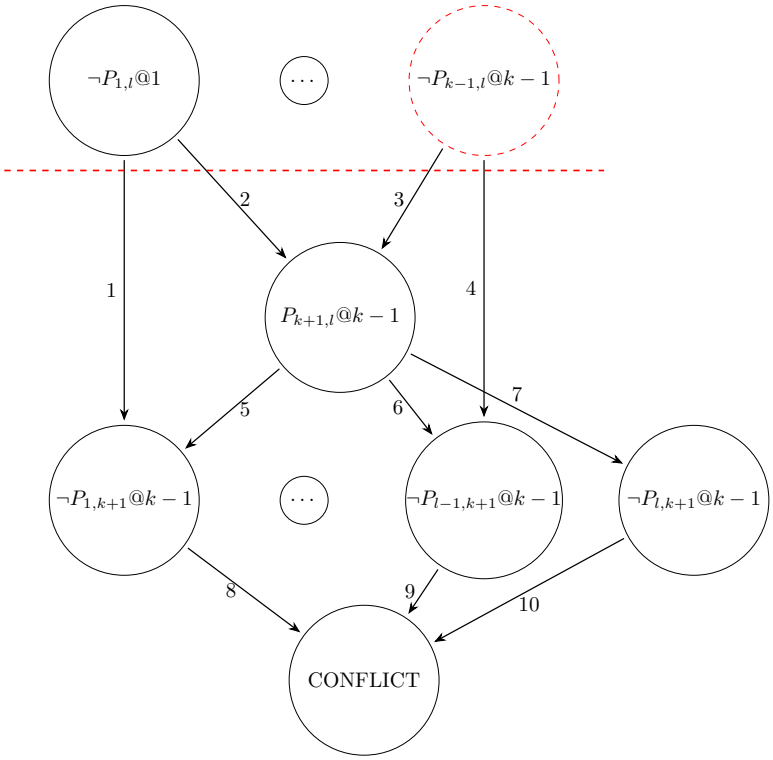


**Figure 5** Cascade Lemma ($k = l, k \neq n-2$) Implication Graph.

1: $A(1, k+1, l)$
2: $C(l, k)$
3: $A(l-1, k+1, l)$
4: $A(k-1, k+1, l)$
5: $C(l, k)$
6: $A(l+1, k+1, l)$
7: $C(l, k)$
8: $A(k, k+1, l)$
9: $A(1, k+1, l)$
10: $A(l-1, k+1, l)$
11: $A(l+1, k+1, l)$
12: $A(k, k+1, l)$
13: $B(l, k+1)$
14: $C(k+1, k)$
15: $C(k+1, k)$
16: $C(k+1, k)$
17: $C(k+1, k)$
18: $C(k+1, k)$

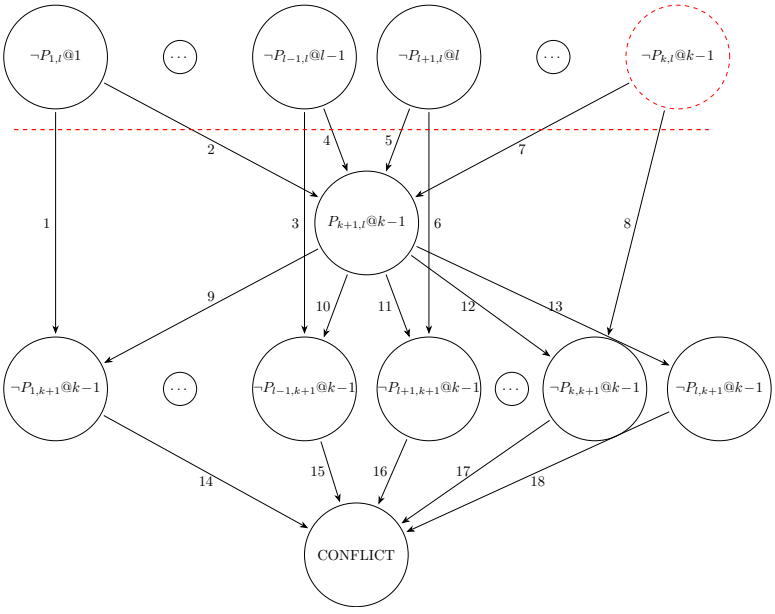


**Figure 6** Cascade Lemma ($l < k < n-2$) Implication Graph.

1: $A(1, k+1, l)$
2: $C(l, k)$
3: $C(l, k)$
4: $A(k-1, k+1, l)$
5: $A(1, k+1, l)$
6: $A(k-1, k+1, l)$
7: $B(k, k+1)$
8: $A(1, k+2, k+1)$
9: $C(k+1, k+1)$
10: $C(k+1, k+1)$
11: $A(k-1, k+2, k+1)$
12: $C(k+1, k+1)$
13: $A(k, k+2, k+1)$
14: $A(1, k+2, k+1)$
15: $A(k-1, k+2, k+1)$
16: $A(k, k+2, k+1)$
17: $B(k+1, k+2)$
18: $C(k+2, k+1)$
19: $C(k+2, k+1)$
20: $C(k+2, k+1)$
21: $C(k+2, k+1)$

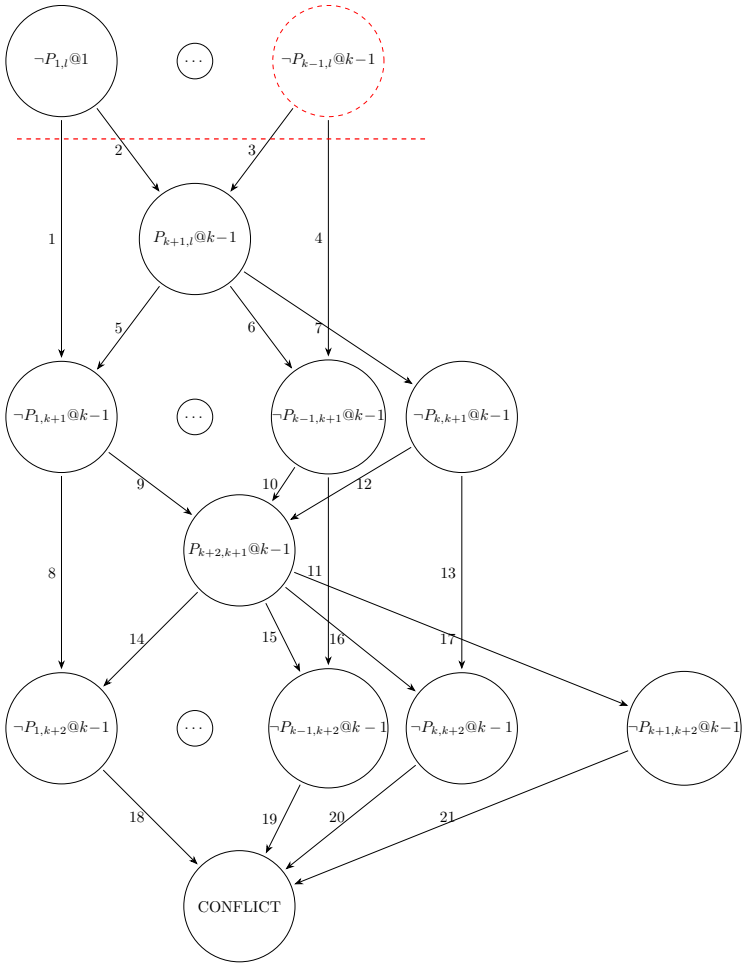


**Figure 7** Cascade Lemma ($k = l = n - 2$) Implication Graph.

The **Equal Score Invariant** holds by Case 2 for the Cascade Lemma.

▶ **Lemma 6** (Descending Lemma). *Assume that* $\Gamma$ *contains the* ***Head*** *clauses followed by* $C(j, n-2), ..., C(j, j-1)$ *for* $j = n-2, ..., l$, *for* $l > 2$. *Additionally, assume that the* ***Equal Score Invariant*** *holds for all of those clauses. Then, the next learned clause is* $C(l-1, n-2)$, *and the* ***Equal Score Invariant*** *still holds.*

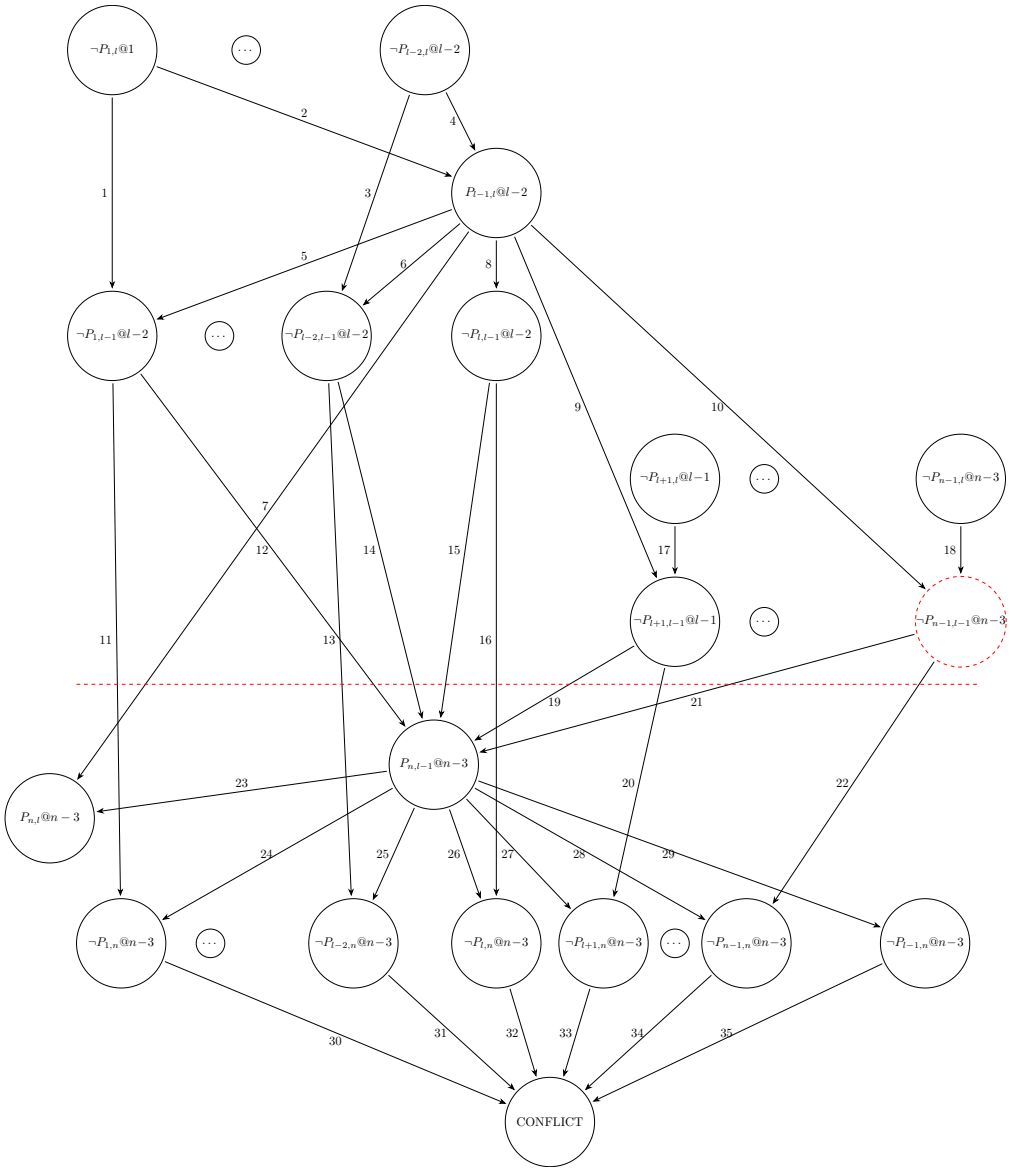


**Figure 8** Descending Lemma Implication Graph.

1: $A(1, l-1, l)$
2: $C(l, l-1)$
3: $A(l-2, l-1, l)$
4: $C(l, l-1)$
5: $A(1, l-1, l)$
6: $A(l-2, l-1, l)$
7: $A(n, l-1, l)$
8: $B(l-1, l)$
9: $A(l+1, l-1, l)$
10: $A(n-1, l-1, l)$
11: $A(1, n, l-1)$
12: $D(l-1)$
13: $A(l-2, n, l-1)$
14: $D(l-1)$
15: $D(l-1)$
16: $A(l, n, l-1)$
17: $A(l+1, l-1, l)$
18: $A(n-1, l-1, l)$
19: $D(l-1)$
20: $A(l+1, n, l-1)$
21: $D(l-1)$
22: $A(n-1, n, l-1)$
23: $A(n, l-1, l)$
24: $A(1, n, l-1)$
25: $A(l-2, n, l-1)$
26: $A(l, n, l-1)$
27: $A(l+1, n, l-1)$
28: $A(n-1, n, l-1)$
29: $B(l-1, n)$
30: $D(n)$
31: $D(n)$
32: $D(n)$
33: $D(n)$
34: $D(n)$
35: $D(n)$

The **Equal Score Invariant** holds by Case 1 for the Descending Lemma.

We now move on to the **Tail**. We first notice that the last learned clause in the Descending Cascade is always the unit clause $C(2,1) = P_{1,2}$. There are two things we need to show. The first is that the Tail begins, i.e., we learn $C(1, n-4)$ after learning $P_{1,2}$ (Lemma 7). The second is that after learning $C(1, n-4), ..., C(1, k)$ for $k > 2$, the next learned clause is $C(1, k-1)$ (Lemma 8).

▶ **Lemma 7** (Pivot Lemma). *Assume that* $\Gamma$ *contains the* ***Head*** *clauses followed by the* ***Descending Cascade*** *clauses, the last of which is the unit clause* $P_{1,2}$*. Additionally, assume that the* ***Equal Score Invariant*** *holds for all of those clauses. The next learned clause is* $C(1, n-4)$*, and the* ***Equal Score Invariant*** *still holds.*

1: $B(1,2)$
2: $A(3,1,2)$
3: $A(n-3,1,2)$
4: $A(3,1,2)$
5: $A(n-3,1,2)$
6: $A(2,n-2,1)$
7: $C(1,n-3)$
8: $A(3,n-2,1)$
9: $C(1,n-3)$
10: $C(1,n-3)$
11: $A(n-3,n-2,1)$
12: $A(2,n-2,1)$
13: $A(3,n-2,1)$
14: $A(n-3,n-2,1)$
15: $B(1,n-2)$
16: $A(l,n,l-1)$
17: $A(l+1,l-1,l)$
18: $A(n-1,l-1,l)$
19: $C(n-2,n-3)$

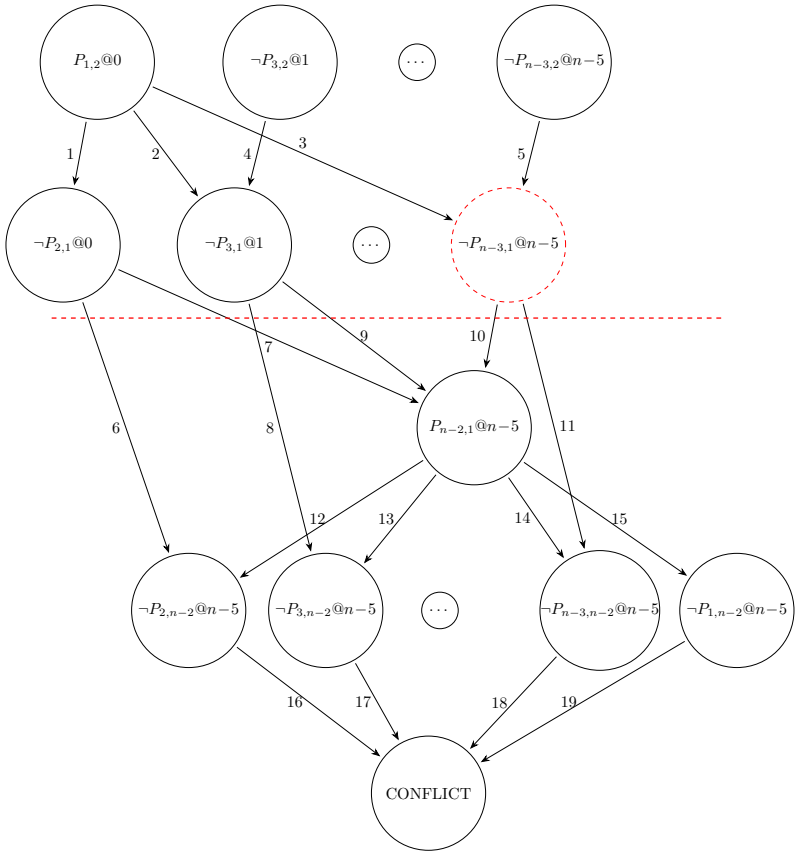


**Figure 9** Pivot Lemma Implication Graph.

The **Equal Score Invariant** holds by Case 2 for the Pivot Lemma.

▶ **Lemma 8** (Tail Lemma). *Assume that* $\Gamma$ *contains the* ***Head clauses*** *followed by the* ***Descending Cascade*** *clauses followed by the clauses* $C(1, n-4), ..., C(1, k)$ *for* $k > 2$*. Additionally, assume that the* ***Equal Score Invariant*** *holds for all of those clauses. The next learned clause is* $C(1, k-1)$*, and the* ***Equal Score Invariant*** *still holds.*

1: $A(k+1,1,2)$
2: $B(1,2)$
3: $A(2,1,k+1)$
4: $C(1,k)$
5: $A(3,k+1,1)$
6: $C(1,k)$
7: $C(1,k)$
8: $A(k,k+1,1)$
9: $A(k+1,1,2)$
10: $A(2,1,k+1)$
11: $A(3,k+1,1)$
12: $A(k,k+1,1)$
13: $B(1,k+1)$
14: $C(k,k+1)$
15: $C(k,k+1)$
16: $C(k,k+1)$
17: $C(k,k+1)$

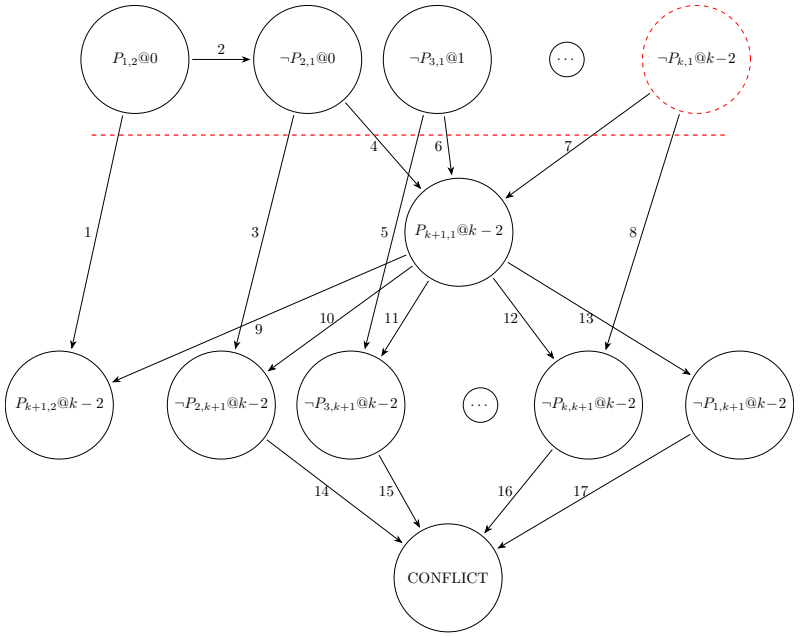


**Figure 10** Tail Lemma Implication Graph.

The **Equal Score Invariant** holds by Case 2 for the Tail Lemma.

The lemmas are enough to show that we learn the **Head**, **Descending Cascade**, and **Tail** clauses. To complete the proof of the theorem, we show that the CDCL solver derives UNSAT upon learning these clauses. As part of the **Descending Cascade**, the unit clause $P_{1,2}$ is learnt. Propagating it, we derive $\neg P_{2,1}$. Then, the clause $P_{2,1} \vee P_{3,1}$, which is in the **Tail**, becomes the unit clause $P_{3,1}$. Then the transitivity clause $A(2,3,1)$ becomes the unit clause $\neg P_{2,3}$. And, the antisymmetry clause $B(1,3)$ becomes the unit clause $\neg P_{1,3}$. We then see that the clause $P_{1,3} \vee P_{2,3}$, which is part of the **Descending Cascade**, is completely falsified. However, we did not have to make any decisions to arrive at this conflict, and therefore we derive UNSAT. ◀

A natural question is whether we crucially need tie-breaking throughout the proof. To answer this question we might try to generalize the proof in a way so that the order in which we branch does not matter, as long as our branching is column-focused (i.e. always branch in the same column as the previous decision if possible). This would result in perhaps processing columns in a different order but the proof would, plausibly, be equivalent up to symmetries. More precisely, we can assume that the first learned clause is $C(1,n-2)$, i.e. that we guarantee that we start off column-focused; then, we ask if the Focus Lemma and the rest of the configuration would result in the solver learning the essentially the same proof without further invoking the tie-breaking rule. The answer is negative due to the Descending Lemma, the mechanism by which we transition columns in the Descending Cascade, which requires that we initially propagate $P_{l-1,l}$ from the first $l-2$ decisions if we want to learn a clause in column $l-1$. So, the previous learned clauses cannot consist of arbitrary subsets of variables from column $l$ if we want to descend to column $l-1$. This suggests that tie-breaking does play an important role in this particular proof, and that further generalizations that removed tie-breaking would be more complex if not longer.

▶ **Theorem 9.** *The total number of conflicts for the deterministic configuration of CDCL SAT solvers in Section 2.3 to derive UNSAT for an $OP_n$ instance is exactly $\binom{n}{2} - 3$, for $n \geq 6$.*

**Proof.** The **Head** has 2 clauses, the **Descending Cascade** has $\sum_{j=2}^{n-2} n - j = n(n-3)/2$ clauses, and the **Tail** has $n - 5$ clauses. In total, then, there are $\frac{(n-3)(n+2)}{2} = \binom{n}{2} - 3$ clauses. ◀

▶ **Corollary 10.** *There exists a deterministic CDCL solver that has an exponential separation with any solver that is polynomially equivalent to tree-like resolution (e.g. DPLL with nondeterministic branching).*

**Proof.** It is known that tree-like resolution has an exponential lower bound for proof size for Ordering Principle instances [6]. We have shown the CDCL configuration described in Section 2.3 requires exactly $\binom{n}{2} - 3$ conflicts to derive UNSAT for $OP_n$ instances. It is well known that a polynomial number of conflicts implies polynomial runtime for CDCL SAT solvers. It then follows that this CDCL solver has an exponential separation with any solver that is polynomially equivalent to tree-like resolution. ◀

▶ Remark 11. A separation in the opposite direction also holds: there exists a class of formulas which can be solved efficiently by nondeterministic DPLL but not by VSIDS-like CDCL. This follows by noticing that the pitfall formulas that separate VSIDS-like CDCL from resolution [22] have short proofs in tree-like resolution.

## 4 Correctness

In the full version of the paper, we prove the correctness of all the implication graphs by showing their *completeness*. We say that an implication graph is *complete* if it is impossible for a different conflict to be learned under the same decisions [19].

Additionally, we experimentally verified our results using the "textbooksat" SAT solver [23] on instances up to $OP_{30}$, analyzing the exact learned clause structures and propagations on the same solver configuration. Our results were confirmed on all of those instances.

## 5 Conclusions

In this paper, we establish an exponential separation between a deterministic configuration of CDCL SAT solvers and any configuration of DPLL SAT solvers. Specifically, we show that the proposed CDCL configuration solves the OP class of formulas in polynomial time, even though tree-like resolution is known to admit only exponential-size proofs for these instances.

We hope this work proves useful for the future analysis of practical, deterministic SAT solvers. Our argument is considerably simplified by two key invariants (the Focus Lemma and the Equal Score Invariant) and we believe that identifying such invariants is essential for making deterministic analysis tractable in general. Moreover, treating the input to a deterministic solver as a DIMACS representation, rather than as an abstract propositional formula, helps bridge the gap between theory and practice: it allows structural information about the formula that is available in empirical settings to be specified precisely and reasoned about formally. Pairing this perspective with deterministic polynomial time pre-processing offers a natural way to recover invariance to the choice of DIMACS representation, and we expect it to be a useful tool in subsequent work.

## References


1 Michael Alekhnovich, Jan Johannsen, Toniann Pitassi, and Alasdair Urquhart. An exponential separation between regular and general resolution. *Theory of Computing*, 3(1):81–102, 2007. doi:10.4086/toc.2007.v003a005.

2 Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *J. Artif. Intell. Res.*, 40:353–373, 2011. doi:10.1613/jair.3152.

3 Albert Atserias and Moritz Müller. Automating resolution is NP-Hard. *J. ACM*, 67(5):31:1–31:17, 2020. doi:10.1145/3409472.

4 Gilles Audemard and Laurent Simon. Experimenting with small changes in conflict-driven clause learning algorithms. In *International Conference on Principles and Practice of Constraint Programming*, pages 630–634. Springer, 2008. doi:10.1007/978-3-540-85958-1_55.

5 Armin Biere and Andreas Fröhlich. Evaluating CDCL variable scoring schemes. In Marijn Heule and Sean A. Weaver, editors, *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, volume 9340 of *Lecture Notes in Computer Science*, pages 405–422. Springer, 2015. doi:10.1007/978-3-319-24318-4_29.

6 Maria Luisa Bonet and Nicola Galesi. Optimality of size-width tradeoffs for resolution. *Comput. Complex.*, 10(4):261–276, 2001. doi:10.1007/s000370100000.

7 Maria Luisa Bonet, Toniann Pitassi, and Ran Raz. On interpolation and automatization for Frege systems. *SIAM J. Comput.*, 29(6):1939–1967, 2000. doi:10.1137/S0097539798353230.

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, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003. doi:10.1007/978-3-540-24605-3_37.

9 Jan Elffers, Jesús Giráldez-Cru, Stephan Gocht, Jakob Nordström, and Laurent Simon. Seeking practical CDCL insights from theoretical sat benchmarks. In *IJCAI*, volume 18, pages 1300–1308, 2018. doi:10.24963/ijcai.2018/181.

10 Jan Elffers, Jan Johannsen, Massimo Lauria, Thomas Magnard, Jakob Nordström, and Marc Vinyals. Trade-offs between time and memory in a tighter model of CDCL SAT solvers. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 160–176. Springer, 2016. doi:10.1007/978-3-319-40970-2_11.

11 Vijay Ganesh and Moshe Y. Vardi. On the unreasonable effectiveness of SAT solvers. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 547–566. Cambridge University Press, 2020. doi:10.1017/9781108637435.032.

12 Hadi Katebi, Karem A. Sakallah, and João P. Marques Silva. Empirical study of the anatomy of modern sat solvers. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, volume 6695 of *Lecture Notes in Computer Science*, pages 343–356. Springer, Springer, 2011. doi:10.1007/978-3-642-21581-0_27.

13 Balakrishnan Krishnamurthy. Short proofs for tricky formulas. *Acta informatica*, 22(3):253–275, 1985. doi:10.1007/BF00265682.

14 Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Exponential recency weighted average branching heuristic for SAT solvers. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, pages 689–695, 2016. doi:10.1609/aaai.v30i1.10439.

15 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, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535. ACM, 2001. doi:10.1145/378239.379017.

**16** Nathan Mull, Shuo Pang, and Alexander A. Razborov. On CDCL-based proof systems with the ordered decision strategy. *SIAM J. Comput.*, 51(4):1368–1399, 2022. doi:10.1137/20m1362528.

**17** Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers with restarts. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings*, volume 5732 of *Lecture Notes in Computer Science*, pages 654–668. Springer, 2009. doi:10.1007/978-3-642-04244-7_51.

**18** Lawrence Ryan. Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University, 2004.

**19** Sahil Samar, Marc Vinyals, and Vijay Ganesh. An exponential separation between deterministic CDCL and DPLL solvers. *CoRR*, abs/2603.16156, 2026. doi:10.48550/arXiv.2603.16156.

**20** João P. Marques Silva and Karem A. Sakallah. GRASP - a new search algorithm for satisfiability. In Rob A. Rutenbar and Ralph H. J. M. Otten, editors, *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, San Jose, CA, USA, November 10-14, 1996*, pages 220–227. IEEE, IEEE Computer Society / ACM, 1996. doi:10.1109/ICCAD.1996.569607.

**21** Gunnar Stålmarck. Short resolution proofs for a sequence of tricky formulas. *Acta Informatica*, 33(3):277–280, 1996. doi:10.1007/s002360050044.

**22** Marc Vinyals. Hard examples for common variable decision heuristics. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1652–1659. AAAI Press, 2020. doi:10.1609/aaai.v34i02.5527.

**23** Marc Vinyals. Textbooksat. https://github.com/marcvinyals/textbooksat, 2026. Accessed: 2026-05-10.