2.9 Conflict Driven Clause Learning (CDCL)

The CDCL calculus tests satisfiability of a finite set N of propositional clauses. Similar to DPLL, Section 2.8, the CDCL calculus explicitely builds a candidate model for a clause set. If such a sequence of literals $L_1, \ldots, L_n$ satisfies the clause set N, it is done. If not, there is a false clause $C \in N$ with respect to $L_1, \ldots, L_n$. Now instead of just backtracking through the literals $L_1, \ldots, L_n$ as done in DPLL, CDCL generates an additional clause, called *learned clause*, that actually guarantees that the subsequence of $L_1, \ldots, L_n$ that caused C to be false will not be generated anymore. This causes CDCL to be exponentially more powerful in proof length than its predecessor DPLL, Section 2.8, or Tableau, Section 2.4, see Theorem 2.14.2. The learned clause is always a resolvent from clauses in N, so CDCL can be viewed as a combination of DPLL (Tableau) and Resolution. More precisely, it can be understood as a resolution variant where a partial model assumption triggers which resolvents are actually generated. In this regard it is similar to propositional Superposition, Section 2.7, where a model assumption generated out of an a priori total ordering on the propositional variables triggers the relevant resolution steps, see the proof of propositional superposition completeness, Theorem 2.7.11. I investigate the connection between model assumptions, proof length, completeness and orderings in Section 2.11.

For any clause set N, I assume that $\bot \notin N$ and that the clauses in N do not contain duplicate literal occurrences. Furthermore, duplicate literal occurrences are always silently removed during rule applications of the calculus. A CDCL problem state is a five-tuple $(M; N; U; k; D)$ where M a sequence of annotated literals, called a *trail*, N and U are sets of clauses, $k \in \mathbb{N}$, and D is a non-empty clause or $\top$ or $\bot$, called the *mode* of the state. The set N is initialized by the problem clauses where the set U contains all newly learned clauses that are consequences of clauses from N derived by resolution. In particular, the following states can be distinguished:

$(\epsilon; N; \emptyset; 0; \top)$	is the start state for some clause set N
$(M; N; U; k; \top)$	is a final state, if $M \models N$ and all literals from N are defined in M
$(M; N; U; k; \bot)$	is a final state, where N has no model
$(M; N; U; k; \top)$	is an intermediate model search state if $M \not\models N$
$(M; N; U; k; D)$	is a backtracking state if $D \notin \{\top, \bot\}$

Literals in $L \in M$ are either annotated with a number, a level, i.e., they have the form L^k meaning that L is the $k-th$ guessed decision literal, or they are annotated with a clause that forced the literal to become true. A literal L is of *level* k with respect to a problem state $(M; N; U; j; C)$ if L or $\text{comp}(L)$ occurs in M and L itself or the first decision literal left from L ($\text{comp}(L)$) in M is annotated with k. If there is no such decision literal then $k = 0$. A clause D is of *level* k with respect to a problem state $(M; N; U; j; C)$ if k is the maximal level of a literal in D. Recall C is a non-empty clause or $\top$ or $\bot$. The rules are

Propagate $(M; N; U; k; \top) \Rightarrow_{\text{CDCL}} (ML^{C \vee L}; N; U; k; \top)$
provided $C \vee L \in (N \cup U)$, $M \models \neg C$, and L is undefined in M

Decide $(M; N; U; k; \top) \Rightarrow_{\text{CDCL}} (ML^{k+1}; N; U; k+1; \top)$
provided L is undefined in M

Conflict $(M; N; U; k; \top) \Rightarrow_{\text{CDCL}} (M; N; U; k; D)$
provided $D \in (N \cup U)$ and $M \models \neg D$

Skip $(ML^{C \vee L}; N; U; k; D) \Rightarrow_{\text{CDCL}} (M; N; U; k; D)$
provided $D \notin \{\top, \bot\}$ and $\text{comp}(L)$ does not occur in D

Resolve $(ML^{C \vee L}; N; U; k; D \vee \text{comp}(L)) \Rightarrow_{\text{CDCL}} (M; N; U; k; D \vee C)$
provided D is of level k

Backtrack $(M_1 K^{i+1} M_2; N; U; k; D \vee L) \Rightarrow_{\text{CDCL}} (M_1 L^{D \vee L}; N; U \cup \{D \vee L\}; i; \top)$
provided L is of level k and D is of level i.

Restart $(M; N; U; k; \top) \Rightarrow_{\text{CDCL}} (\epsilon; N; U; 0; \top)$
provided $M \not\models N$

Forget $(M; N; U \uplus \{C\}; k; \top) \Rightarrow_{\text{CDCL}} (M; N; U; k; \top)$
provided $M \not\models N$

Compared to expositions of this calculus in the literature, e.g. [67], the above rule set is more concrete. It does not need a Fail rule anymore and 1-UIP backjumping [16] is build in. The clause $D \vee L$ immediately propagates after Backtracking.

Recall that $\bot$ denotes the empty clause, hence failure of searching for a model. The level of the empty clause $\bot$ is 0. The clause $D \vee L$ added in rule Backtrack to U is called a *learned clause*. When applying Resolve I silently assumed that duplicate literal occurrences are merged, i.e., the clause $D \vee \text{comp}(L)$ is always condensed (see Section 2.7). Compared to superposition, condensation is always applied eagerly without mentioning. The CDCL algorithm stops with a model M if neither Propagate nor Decide nor Conflict are applicable to a state $(M; N; U; k; \top)$, hence $M \models N$ and all literals of N are defined in M. The only possibility to generate a state $(M; N; U; k; \bot)$ is by the rule Resolve. So in case of detecting unsatisfiability the CDCL algorithm actually generates a resolution proof as a certificate. I will discuss this aspect in more detail in Section 2.11. In the special case of a unit clause L, the rule Propagate actually annotates the literal L with itself. So the propagated literals on the trail are annotated with the respective propagating clause and the decision literals with the respective level.

Obviously, the CDCL rule set does not terminate in general for a number of reasons. For example, starting with $(\epsilon; N; \emptyset; 0; \top)$ any combination of the rules Propagate, Decide and eventually Restart yields the start state again. Even after a successful application of Backtrack, exhaustive application of Forget followed by Restart again may produce the start state. So why these rules Forget and Restart? Actually, any modern SAT solver makes use of the two rules. The rule Forget is needed to get rid of "redundant" clauses. For otherwise, the number of clauses in $N \cup U$ may get too large to be processed anymore in an efficient way. The rule Restart makes sense with respect to a suitable heuristic (see Section 2.10.2) for selecting the decision literals. If applied properly, it helps the calculus to focus on a part of N where it currently can make progress [16]. I will further motivate the rules later on.

The original SAT literature [79, 53, 65, 16] does not contain a theoretical foundation for a redundancy concept for CDCL. Redundancy has been experimentally developed based on heuristics where completeness is guaranteed in many implementations by increasing the number of overall kept clauses over a run and by following the DPLL style systematic exploration of potential models. I will develop a theoretical foundation for CDCL redundancy in Section 2.11. There, I will also discuss clause minimization, Section 2.13.2, a technique to strenghten learned clauses implemented in almost all CDCL solvers.

The following examples show that if the CDCL rules are applied in an arbitrary way, then many unwanted phenomena can happen. Proofs can become longer than needed, the rules can produce stuck states and clauses are learned that are already contained in the set $N \cup U$. In order to overcome all these situations, a strategy prioritizing certain rule applications is eventually added.

Example 2.9.1 (CDCL Lengthy Proof)**.** Consider the clause set $N = \{P \vee Q, \neg P \vee Q, \neg Q\}$ which is unsatisfiable. The below is a CDCL derivation proving this fact. The chosen strategy for CDCL rule selection priorizing the rule Decide produces a lengthy proof.

$(\epsilon; N; \emptyset; 0; \top)$
$\Rightarrow^{\text{Decide}}_{\text{CDCL}}$ $(P^1; N; \emptyset; 1; \top)$
$\Rightarrow^{\text{Decide}}_{\text{CDCL}}$ $(P^1 \neg Q^2; N; \emptyset; 2; \top)$
$\Rightarrow^{\text{Conflict}}_{\text{CDCL}}$ $(P^1 \neg Q^2; N; \emptyset; 2; \neg P \vee Q)$
$\Rightarrow^{\text{Backtrack}}_{\text{CDCL}}$ $(P^1 Q^{\neg P \vee Q}; N; \{\neg P \vee Q\}; 1; \top)$
$\Rightarrow^{\text{Conflict}}_{\text{CDCL}}$ $(P^1 Q^{\neg P \vee Q}; N; \{\neg P \vee Q\}; 1; \neg Q)$
$\Rightarrow^{\text{Backtrack}}_{\text{CDCL}}$ $(\neg Q^{\neg Q}; N; \{\neg P \vee Q, \neg Q\}; 0; \top)$
$\Rightarrow^{\text{Decide}}_{\text{CDCL}}$ $(\neg Q^{\neg Q} P^1; N; \{\neg P \vee Q, \neg Q\}; 1; \top)$
$\Rightarrow^{\text{Conflict}}_{\text{CDCL}}$ $(\neg Q^{\neg Q} P^1; N; \{\neg P \vee Q, \neg Q\}; 1; \neg P \vee Q)$
$\Rightarrow^{\text{Backtrack}}_{\text{CDCL}}$ $(\neg Q^{\neg Q} \neg P^{\neg P \vee Q}; N; \{\neg P \vee Q, \neg Q\}; 0; \top)$
$\Rightarrow^{\text{Conflict}}_{\text{CDCL}}$ $(\neg Q^{\neg Q} \neg P^{\neg P \vee Q}; N; \{\neg P \vee Q, \neg Q\}; 0; P \vee Q)$
$\Rightarrow^{\text{Resolve}}_{\text{CDCL}}$ $(\neg Q^{\neg Q}; N; \{\neg P \vee Q, \neg Q\}; 0; Q)$
$\Rightarrow^{\text{Resolve}}_{\text{CDCL}}$ $(\epsilon; N; \{\neg P \vee Q, \neg Q\}; 0; \bot)$

Example 2.9.2 (CDCL Short Proof)**.** Consider again the clause set $N = \{P \vee Q, \neg P \vee Q, \neg Q\}$ from Example 2.9.1. For the following CDCL derivation the rules Propagate and Conflict are preferred over the other rules.

$(\epsilon; N; \emptyset; 0; \top)$
$\Rightarrow^{\text{Propagate}}_{\text{CDCL}}$ $(\neg Q^{\neg Q}; N; \emptyset; 0; \top)$
$\Rightarrow^{\text{Propagate}}_{\text{CDCL}}$ $(\neg Q^{\neg Q} P^{Q \vee P}; N; \emptyset; 0; \top)$
$\Rightarrow^{\text{Conflict}}_{\text{CDCL}}$ $(\neg Q^{\neg Q} P^{Q \vee P}; N; \emptyset; 0; \neg P \vee Q)$
$\Rightarrow^{\text{Resolve}}_{\text{CDCL}}$ $(\neg Q^{\neg Q}; N; \emptyset; 0; Q)$
$\Rightarrow^{\text{Resolve}}_{\text{CDCL}}$ $(\epsilon; N; \emptyset; 0; \bot)$

Example 2.9.3 (CDCL Stuck)**.** The CDCL calculus can even get stuck, i.e., a sequence of rule applications leads to a state where no rule is applicable anymore, but the state does neither indicate satisfiability, nor unsatisfiability. Consider a clause set $N = \{Q \vee P, \neg P \vee \neg R, \ldots\}$ and the derivation

$(\epsilon; N; \emptyset; 0; \top)$
$\Rightarrow^{\text{Decide}}_{\text{CDCL}}$ $(P^1; N; \emptyset; 1; \top)$
$\Rightarrow^{\text{Decide}}_{\text{CDCL}}$ $(P^1 R^2; N; \emptyset; 2; \top)$
$\Rightarrow^{\text{Decide}}_{\text{CDCL}}$ $(P^1 R^2 Q^3; N; \emptyset; 3; \top)$
$\Rightarrow^{\text{Conflict}}_{\text{CDCL}}$ $(P^1 R^2 Q^3; N; \emptyset; 3; \neg P \vee \neg R)$.

Obviously, neither Skip nor Resolve are applicable to the final state. Backtracking is not applicable as well because $\neg P \vee \neg R$ is of level 2 and the actual level of the final state is 3.

C

Stuck states could be prevented in various ways. Either by following a particular strategy in case several rules are applicable. The reasonable strategy below, Definition 2.9.5, has this property. Or by changing the rules themselves. For example, if Skip is modified to remove decision literals from the trail, this can also prevent stuck states, see Exercise **??**.

Example 2.9.4 (CDCL Redundancy)**.** The CDCL calculus can also produce redundant clauses, in particular learn a clause that is already contained in $N \cup U$. Consider again a clause set $N = \{Q \vee P, \neg P \vee \neg R, \ldots\}$ and the derivation

$$
\begin{aligned}
&(\epsilon; N; \emptyset; 0; \top)\\
\Rightarrow^{\text{Decide}}_{\text{CDCL}} \quad &(P^1; N; \emptyset; 1; \top)\\
\Rightarrow^{\text{Decide}}_{\text{CDCL}} \quad &(P^1 R^2; N; \emptyset; 2; \top)\\
\Rightarrow^{\text{Conflict}}_{\text{CDCL}} \quad &(P^1 R^2; N; \emptyset; 2; \neg P \vee \neg R).\\
\Rightarrow^{\text{Backtrack}}_{\text{CDCL}} \quad &(P^1 \neg R^{\neg P \vee \neg R}; N; \{\neg P \vee \neg R\}; 1; \top)
\end{aligned}
$$

where the clause $\neg P \vee \neg R$ is learned although it is already contained in N.

In an implementation the rule Conflict is preferred over the rule Propagate and both over all other rules. Exactly this strategy has been used in Example 2.9.2 and is called *reasonable* below. A further ingredient of a state-of-the-art implementation is a dynamic heuristic which literal is actually used by the rule Decide. This heuristic typically depends on the literals resolved on by the rule Resolve or contained in eventually learned clause. All these literals "get a bonus", see Section 2.10.2 for details.

Definition 2.9.5 (Reasonable CDCL Strategy)**.** A CDCL strategy is *reasonable* if the rules Conflict and Propagate are always preferred over all other rules.

Proposition 2.9.6 (CDCL Basic Properties)**.** Consider a CDCL state $(M; N; U; k; C)$ derived from a start state $(\epsilon, N, \emptyset, 0, \top)$ by any strategy but without using the rules Restart and Forget. Then the following properties hold:

1. M is consistent.
2. All learned clauses are entailed by N.
3. If $C \notin \{\top, \bot\}$ then $M \models \neg C$.
4. If $C = \top$ and M contains only propagated literals then for each valuation $\mathcal{A}$ with $\mathcal{A} \models N$ it holds that $\mathcal{A} \models M$.
5. If $C = \top$, M contains only propagated literals and $M \models \neg D$ for some $D \in (N \cup U)$ then N is unsatisfiable.
6. If $C = \bot$ then CDCL terminates and N is unsatisfiable.
7. k is the maximal level of a literal in M.
8. Each infinite derivation

$$(\epsilon; N; \emptyset; 0; \top) \Rightarrow_{\text{CDCL}} (M_1; N; U_1; k_1; D_1) \Rightarrow_{\text{CDCL}} \ldots$$

contains an infinite number of Backtrack applications.

Proof. 1. M is consistent if it does does not contain L and $\text{comp}(L)$ at the same time. The only rules that add literals are Propagate, Decide, and Backtrack. The rules Propagate and Decide only add undefined literals to M. By an inductive argument this holds also for Backtrack as it just removes literals from M and flips one literal already contained in M.

2. A learned clause is always a resolvent of clauses from $N \cup U$ and eventually added to U where U is initially empty. By soundness of resolution (Theorem 2.6.1) and an inductive argument it is entailed by N.

3. A clause $C \notin \{\top, \bot\}$ can only occur after Conflict where $M \models \neg C$. The rule Skip does not change C and only deletes propagated literals from M that are not contained in C. By an inductive argument, if the rule Resolve is applied to a state $(M' \,\text{comp}(L)^{D' \vee \text{comp}(L)}; N; U; k; D \vee L)$ where $C = D \vee L$ resulting in $(M'; N; U; k; D \vee D')$ then $M' \models \neg D$ because $M' \models \neg C$ and $M' \models \neg D'$ because $\text{comp}(L)$ was propagated with respect to M' and $D' \vee \text{comp}(L)$.

4. Proof by induction on the number n of propagated literals in M. Let $M = L_1, \ldots, L_n, L_{n+1}$. There are two rules that could have added L_{n+1}. (i) rule Propagate: in this case there is a clause $C = D \vee L_{n+1}$ where L_{n+1} was undefined in M and $M \models \neg D$. By induction hypothesis for each valuation $\mathcal{A}$ with $\mathcal{A} \models N$ it holds that $\mathcal{A}(L_i) = 1$ for all $i \in \{1, \ldots, n\}$. Since all literals in D appear negated in M with the induction hypothesis it holds that all those literals must have the truth value 1 in any valuation $\mathcal{A}$. Therefore, for the clause C to be true L_{n+1} must be true as well in any valuation. It follows that for each valuation $\mathcal{A}$ it holds that $\mathcal{A}(L_i) = 1$ for all $i \in \{1, \ldots, n+1\}$. (ii) rule Backtrack: the state $(M_1 K^{i+1} M_2; N; U; k; D \vee L_{n+1})$ where $M \models \neg(D \vee L_{n+1})$ (by Proposition 2.9.6.3) and $M_1 = L_1 \ldots L_n$ with only propagated literals results in $(M_1 L_{n+1}^{D \vee L_{n+1}}; N; U; i; \top)$. With the induction hypothesis for each valuation $\mathcal{A}$ with $\mathcal{A} \models N$ it holds that $\mathcal{A}(L_i) = 1$ for all $1 \leq i \leq n$, i.e., in particular for each literal L in D it holds $\mathcal{A}(L) = 0$ since each literal in D appears negated in M_1. Thus, for each each valuation $\mathcal{A}$ with $\mathcal{A} \models N$ it holds $\mathcal{A}(L_{n+1}) = 1$.

5. Let $D = K_1 \vee \ldots \vee K_m$. Since $M \models \neg D$ it holds that $\text{comp}(K_i) \in M$ for all $1 \leq i \leq m$. With Proposition 2.9.6.4 for each valuation $\mathcal{A}$ with $\mathcal{A} \models N$ it holds that $\mathcal{A}(L) = 1$ for all $L \in M$. Thus in particular it holds that $\mathcal{A}(\text{comp}(K_i)) = 1$ for all $1 \leq i \leq m$. Therefore D is always false under any valuation $\mathcal{A}$ and N is unsatisfiable.

6. By the definition of the rules the state $(M; N; U; k; \bot)$ can only be reached if the rule Conflict sets the mode of a state $(M'; N; U; k; \top)$ to a conflict clause D. Then Resolve is eventually used to derive $\bot$. By Proposition 2.9.6-2 it follows that N is unsatisfiable.

7. By induction on the number of rule applications. Actually, I prove something stronger, for any state $(M; N; U; k; D)$ reachable from a start state, k is the maximal level of literal in M and M includes exactly one literal L^i for $1 \leq i \leq k$ in increasing order from left to right. For the initial state $(\epsilon; N; \emptyset; 0; \top)$ the trail M is empty so $k = 0$ is fine. The only rules that manipulate decision

literals in M or k are Decide, Backtrack, and Restart. Restart is fine as well. For Decide the level k is increased to $k+1$ and a literal L^{k+1} is added right to M, so Decide is also fine. For Backtrack all decision literals including K^{i+1} are deleted from right to left from the trail, so by induction hypothesis it includes a decision literal of level i. But this is exactly the level that Backtrack sets for the new state.

8. Proof by contradiction. Assume Backtrack is applied only finitely often in the infinite trace. Then there exists an i and a state $(M_i; N; U_i; k_i; D_i)$ such that there is no Backtrack application beyond this state. Propagate and Decide can only be applied as long as there are undefined literals in M. Since N is finite there can only be finitely many rule applications of Propagate and Decide.

By definition the application of the rules Skip, Resolve and Backtrack is preceded by an application of the rule Conflict. The rules Skip and Resolve can only be applied finitely often until a decision literal is the rightmost literal of M, or M becomes empty. For the rule Conflict to be applied infinitely often mode has to change back to $\top$. But the only rule that changes the mode to $\top$ is Backtrack which is not applied anymore. A contradiction. □

Lemma 2.9.7 (CDCL Redundancy)**.** Consider a CDCL derivation by a reasonable strategy. Then CDCL never learns a clause contained in $N \cup U$.

Proof. By contradiction. Assume CDCL learns the same clause twice, i.e., it reaches a state $(M; N; U; k; D \vee L)$ where Backtracking is applicable and $D \vee L \in (N \cup U)$. More precisely, the state has the form $(M_1 K^{i+1} M_2' K_1^k K_2 \dots K_n; N; U; k; D \vee L)$ where the K_j, $j > 1$ are propagated literals that do not occur complemented in D, as for otherwise D cannot be of level i. Furthermore, one of the K_j is the complement of L. But now, because $D \vee L$ is false in $M_1 K^{i+1} M_2' K_1^k K_2 \dots K_n$ and $D \vee L \in (N \cup U)$ instead of deciding K_1^k the literal L should have been propagated by a reasonable strategy. A contradiction. Note that none of the K_j can be annotated with $D \vee L$. □

Corollary 2.9.8 (CDCL Redundancy)**.** Consider a CDCL derivation by a reasonable strategy. Then CDCL never learns a clause subsumed by a clause in $N \cup U$.

Proof. Exercise **??**. □

A reasonable strategy is mandatory for the above result. Example 2.9.4 shows that prioritizing Decide over Propagate can result in learning a clause contained in $N \cup U$. The below example shows that even if CDCL starts Conflict with a redundant clause, one that is already subsumed, the eventually learned clause is not redundant.

Example 2.9.9 (CDCL Subsumed Conflict)**.** Consider the clause set $N = \{\neg P \vee Q, \neg P \vee R, \neg P \vee \neg R \vee \neg Q, \neg P \vee \neg R \vee Q, \dots\}$ and a derivation

$$
\begin{array}{ll}
(\epsilon; N; \emptyset; 0; \top) & \\
\Rightarrow^{\text{Decide}}_{\text{CDCL}} & (P^1; N; \emptyset; 1; \top) \\
\Rightarrow^{\text{Propagate}}_{\text{CDCL}} & (P^1 R^{\neg P \vee R}; N; \emptyset; 1; \top) \\
\Rightarrow^{\text{Propagate}}_{\text{CDCL}} & (P^1 R^{\neg P \vee R} \neg Q^{\neg P \vee \neg R \vee \neg Q}; N; \emptyset; 1; \top) \\
\Rightarrow^{\text{Conflict}}_{\text{CDCL}} & (P^1 R^{\neg P \vee R} \neg Q^{\neg P \vee \neg R \vee \neg Q}; N; \emptyset; 1; \neg P \vee \neg R \vee Q) \\
\Rightarrow^{\text{Resolve}}_{\text{CDCL}} & (P^1 R^{\neg P \vee R}; N; \emptyset; 1; \neg P \vee \neg R) \\
\Rightarrow^{\text{Resolve}}_{\text{CDCL}} & (P^1; N; \emptyset; 1; \neg P) \\
\Rightarrow^{\text{Backtrack}}_{\text{CDCL}} & (\neg P^{\neg P}; N; \{\neg P\}; 0; \top)
\end{array}
$$

Note that although the conflict clause $\neg P \vee \neg R \vee Q$ is subsumed by $\neg P \vee Q$, the eventually learned clause $\neg P$ is not redundant.

Lemma 2.9.10 (CDCL Soundness)**.** In a reasonable CDCL derivation, CDCL can only terminate in two different types of final states: $(M; N; U; k; \top)$ where $M \models N$ and $(M; N; U; k; \bot)$ where N is unsatisfiable.

Proof. If CDCL terminates with $(M; N; U; k; \top)$ then all literals of N are defined in M and Conflict is not applicable, i.e., for all clauses $C \in N$ it holds $M \models C$, so $M \models N$. In addition if CDCL terminates with $(M; N; U; k; \bot)$ then by Proposition 2.9.6.2 the clause set N is unsatisfiable.

What remains is to show that with a reasonable strategy CDCL cannot get stuck, see Example 2.9.3. I prove that no stuck state can reached by contradiction. Assume that CDCL terminates in a terminating state $(M_1 K^{i+1} M_2' K_1^k K_2 \dots K_n; N; U; k; D \vee L)$, where the K_i, $i > 1$, are propagated literals. If $\text{comp}(K_n) \neq L$ and $n > 1$ then Skip is applicable. If $\text{comp}(K_n) = L$ then either Resolve or Backtrack is applicable. Since neither Skip, Resolve, or Backtrack are applicable, it holds $n = 1$ and the complement of K_1^k does not occur in $D \vee L$. But then $M_1 K^{i+1} M_2' \models \neg(D \vee L)$ so the decision on K_1^k contradicts a reasonable strategy. □

Proposition 2.9.11 (CDCL Soundness)**.** The rules of the CDCL algorithm are sound: (i) if CDCL terminates from $(\epsilon; N; \emptyset; 0; \top)$ in the state $(M; N; U; k; \top)$, then N is satisfiable, (ii) if CDCL terminates from $(\epsilon; N; \emptyset; 0; \top)$ in the state $(M; N; U; k; \bot)$, then N is unsatisfiable.

Proof. (i) by Proposition 2.9.6.1 the sequence M is always consistent. Since $(M; N; U; k; \top)$ is a final state neither Decide nor Propagate can be applied and hence all atoms of literals in N occur in M. Furthermore, the state is not a result of an application of the Conflict rule nor is the Conflict rule applicable. Therefore, there is no $D \in (N \cup U)$ with $M \models \neg D$ and since all literals in N are defined in M this means $M \models N$, so N is satisfiable.

(ii) the only rule that can produce $(M; N; U; k; \bot)$ is Conflict, where $\bot \in (N \cup U)$. By Proposition 2.9.6.2 all learned clauses are entailed by N, hence N is unsatisfiable. □

Proposition 2.9.12 (CDCL Strong Completeness)**.** The CDCL rule set is complete: for any valuation M with $M \models N$ there is a reasonable sequence of rule

applications generating $(M'; N; U; k; \top)$ as a final state, where M and M' only differ in the order of literals.

Proof. By induction on the length of M. Assume we have already reached a state $(M'; N; U; k; \top)$ where $M' \subset M$. If Propagate is applicable to $(M'; N; U; k; \top)$ extending it to $(M'L^{C \vee L}; N; U; k; \top)$ then $L \in M$. For otherwise, I pick a literal $L \in M$ that is not defined in M' and apply Decide yielding $(M'L^{k+1}; N; U; k+1; \top)$. The rule Conflict is not applicable, because $M \models N$ and $M' \subset M$. □

Proposition 2.9.13 (CDCL Termination)**.** Assume the algorithm CDCL with all rules except Restart and Forget is applied using a reasonable strategy. Then it terminates in a state $(M; N; U; k; D)$ with $D \in \{\top, \bot\}$.

Proof. By Lemma 2.9.10 if CDCL terminates using a reasonable strategy then $D \in \{\top, \bot\}$. I show termination by contradiction. By Proposition 2.9.6.8 an infinite run includes infinitely many Backtrack applications. By Lemma 2.9.7 each learned clause does not occur in $N \cup U$. But there are only finitely many different condensed clauses with respect to the finite signature contained in N. A contradiction. □

The CDCL rule set does not terminate in general. This is due to the rules Restart and Forget. If they are applied only finitely often then the algorithm terminates. Then at some point of the derivation the final application of Restart and Forget occurred. From this point onwards Proposition 2.9.13 applies, provided a reasonable strategy.

Example 2.9.14 (CDCL Termination I)**.** Consider the clause set $N = \{P \vee Q, \neg P \vee Q, \neg Q\}$. The CDCL algorithm does not terminate due to the rule Restart.

$$
\begin{aligned}
&(\epsilon; N; \emptyset; 0; \top)\\
\Rightarrow_{\text{CDCL}}^{\text{Propagate}} \quad &(\neg Q^{\neg Q}; N; \emptyset; 0; \top)\\
\Rightarrow_{\text{CDCL}}^{\text{Propagate}} \quad &(\neg Q^{\neg Q} P^{Q \vee P}; N; \emptyset; 0; \top)\\
\Rightarrow_{\text{CDCL}}^{\text{Restart}} \quad &(\epsilon; N; \emptyset; 0; \top)\\
\Rightarrow_{\text{CDCL}}^{\text{Propagate}} \quad &(\neg Q^{\neg Q}; N; \emptyset; 0; \top)\\
\Rightarrow_{\text{CDCL}}^{\text{Propagate}} \quad &(\neg Q^{\neg Q} P^{Q \vee P}; N; \emptyset; 0; \top)\\
\Rightarrow_{\text{CDCL}}^{\text{Restart}} \quad &(\epsilon; N; \emptyset; 0; \top)\\
\Rightarrow_{\text{CDCL}} \quad &\dots
\end{aligned}
$$

Example 2.9.15 (CDCL Termination II)**.** Consider the clause set $N = \{\neg P \vee Q \vee \neg R, \neg P \vee Q \vee R\}$. The CDCL algorithm does not terminate due to the rule Forget.

$$
\begin{array}{ll}
& (\epsilon; N; \emptyset; 0; \top) \\
\Rightarrow_{\text{CDCL}}^{\text{Decide}} & (P^1; N; \emptyset; 1; \top) \\
\Rightarrow_{\text{CDCL}}^{\text{Decide}} & (P^1 \neg Q^2; N; \emptyset; 2; \top) \\
\Rightarrow_{\text{CDCL}}^{\text{Propagate}} & (P^1 \neg Q^2 \neg R^{\neg P \vee Q \vee \neg R}; N; \emptyset; 2; \top) \\
\Rightarrow_{\text{CDCL}}^{\text{Conflict}} & (P^1 \neg Q^2 \neg R^{\neg P \vee Q \vee \neg R}; N; \emptyset; 2; \neg P \vee Q \vee R) \\
\Rightarrow_{\text{CDCL}}^{\text{Resolve}} & (P^1 \neg Q^2; N; \emptyset; 2; \neg P \vee Q) \\
\Rightarrow_{\text{CDCL}}^{\text{Backtrack}} & (P^1; N; \{\neg P \vee Q\}; 1; \top) \\
\Rightarrow_{\text{CDCL}}^{\text{Forget}} & (P^1; N; \emptyset; 1; \top) \\
\Rightarrow_{\text{CDCL}}^{\text{Decide}} & (P^1 \neg Q^2; N; \emptyset; 2; \top) \\
\Rightarrow_{\text{CDCL}}^{\text{Propagate}} & (P^1 \neg Q^2 \neg R^{\neg P \vee Q \vee \neg R}; N; \emptyset; 2; \top) \\
\Rightarrow_{\text{CDCL}}^{\text{Conflict}} & (P^1 \neg Q^2 \neg R^{\neg P \vee Q \vee \neg R}; N; \emptyset; 2; \neg P \vee Q \vee R) \\
\Rightarrow_{\text{CDCL}} & \dots
\end{array}
$$

As an alternative and positive proof of Proposition 2.9.13 the CDCL termination can be shown by assigning a well-founded measure μ and proving that it decreases with each rule application except for the rules Restart and Forget. Let n be the number of propositional variables in N. The range for the measure μ is $\mathbb{N} \times \{0, 1\} \times \mathbb{N}$.

$$
\mu((M; N; U; k; D)) = \begin{cases} (3^n - 1 - |U|, 1, n - |M|) & , D = \top \\ (3^n - 1 - |U|, 0, |M|) & , else \end{cases}
$$

The well-founded ordering is the lexicographic extension of $<$ to triples. What remains to be shown is that each rule application except Restart and Forget decreases μ. This is done via a case analysis over the rules:

Propagate:

$$
\begin{aligned}
\mu((M; N; U; k; \top)) &= (3^n - 1 - |U|, 1, n - |M|) \\
&> (3^n - 1 - |U|, 1, n - |ML^{C \vee L}|) \\
&= \mu((ML^{C \vee L}; N; U; k; \top))
\end{aligned}
$$

Decide:

$$
\begin{aligned}
\mu((M; N; U; k; \top)) &= (3^n - 1 - |U|, 1, n - |M|) \\
&> (3^n - 1 - |U|, 1, n - |ML^{k+1}|) \\
&= \mu((ML^{k+1}; N; U; k; \top))
\end{aligned}
$$

Conflict:

$$
\begin{aligned}
\mu((M; N; U; k; \top)) &= (3^n - 1 - |U|, 1, n - |M|) \\
&> (3^n - 1 - |U|, 0, |M|) \\
&= \mu((M; N; U; k; D))
\end{aligned}
$$

Skip:

$$
\begin{aligned}
\mu((ML^{C \vee L}; N; U; k; D)) &= (3^n - 1 - |U|, 0, |ML^{C \vee L}|) \\
&> (3^n - 1 - |U|, 0, |M|) \\
&= \mu((M; N; U; k; D))
\end{aligned}
$$

Resolve:

$$\begin{aligned}\mu((ML^{C\vee L}; N; U; k; D \vee \neg L)) &= (3^n - 1 - |U|, 0, |ML^{C\vee L}|)\\ &> (3^n - 1 - |U|, 0, |M|)\\ &= \mu((M; N; U; k; D \vee C))\end{aligned}$$

Backtrack:

$$\begin{aligned}\mu((M_1K^{i+1}M_2; N; U; k; D \vee L)) &= (3^n - 1 - |U|, 0, |M_1K^{i+1}M_2|)\\ &> (3^n - 1 - |U \cup \{D \vee L\}|, 1, n - |M_1L^{D\vee L}|)\\ &= \mu((M_1L^{D\vee L}; N; U \cup \{D \vee L\}; i; \top))\end{aligned}$$

Recall that the strict inequation for Backtrack only holds with respect to a reasonable strategy, see Lemma 2.9.7.

Another proof of termination is possible by considering the number of learned clauses.

Theorem 2.9.16 (CDCL Termination: Learned Clauses)**.** A CDCL run from $(\epsilon; N; \emptyset; 0; \top)$ without rule Restart terminates and learns at most 2^n clauses, where n is the number of propositional variables in N.

Proof. Clauses are only learned at Backtrack applications. Without Restart, the number of Backtrack applications is limited to 2^n. By Proposition 2.9.6.8 any infinite CDCL run contains infinitely many Backtrack applications. □

The CDCL calculus is also *incremental* in the following sense: assume a CDCL run on a clause set N results in a trail M that constitutes a model for N. If satisfiability of N together with a new clause C needs to be tested, then either $M \models C$ and a satisfying model is found, or $M \models \neg C$. In the latter case there is a minimal subsequence $M'L \subseteq M$ such that $M'L \models \neg C$. Now if L is a decision literal then M is replaced with $M' \operatorname{comp}(L)^C$ and CDCL continues with from $N \cup \{C\}$ this state. If L is a propagated literal, then Conflict is applicable to $N \cup \{C\}$ and CDCL continues from this state.

Note that the example problem from Figure 2.6 depicting a closed tableau where the optimal closed tableau is exponentially smaller, will be solved by CDCL without the exponential overhead. If closedness does not depend on the K_j, K'_j literals, then even after deciding (propagating) these literals and later on the L_i, L'_i literals, the conflict clauses as well as the learned clauses will not contain any K_j, K'_j literal. Therefore one run through a L_i, L'_i subtree yields the empty clause via CDCL.

2.10 Implementing CDCL

The performance of modern SAT solvers is sensitive to at least six components [16]: (i) preprocessing [36], (ii) conflict-driven clause learning (CDCL) [79,

53, 67, 84], (iii) heuristics for decision variable selection [65, 14], (iv) heuristics for restarting [49, 6, 15], (v) heuristics for learned clauses deletion (forgetting) [5, 50], and (vi) reduction during a CDCL run [51], typically called *inprocessing* in the SAT context. All these components are inter-connected, so if one of them is deactivated, or the interplay is not correctly adjusted then efficiency drops down.

In addition, for an efficient CDCL implementation the underlying data-structure and algorithms play a crucial part. There is a significant gap between the rule based CDCL calculus discussed in Section 2.9 and an actual implementation. In this section I explain some of the main techniques.

The most important data-structure of a CDCL implementation is the *2-watched literal* scheme, introduced in the next section. For choosing the next decision variables a suitable heuristic is important. One of the standard heuristics is called *VSIDS* (Variable State Independent Decaying Sum) end explained in Section 2.10.2. Furthermore, the decision for choosing the most reasonable clause to be learned after a discovered conflict is handled by the notion of *UIPs* (Unique Implication Points). Actually, the CDCL calculus of Section 2.9 has learning a UIP clause build in. Finally, a suitable Restart and Forget strategy, the deletion policy, is essential for an efficient long term behavior of the calculus, discussed in Section 2.10.4.

2.10.1 Lazy Data Structure: 2-Watched Literals (2WL)

For applying the rule Propagate, all literals in a clause except one need to be false with respect to the current trail. For applying the rule Conflict all literals need to be false with respect to the current trail. Thus as long as at least two literals of a clause are undefined with respect to the current trail or at least one literal is true with respect to the current trail the clause can be discarded by the CDCL calculus rules.

This results in the well-known *2-watched literals* idea where only two literals of a clause are indexed. Indexing is needed in general, in order to efficiently access relevant clauses. After a decision or propagation of some literal L all clauses containing $\mathrm{comp}(L)$ need to be visited in order to check for rule Propagate and Conflict. Simply traversing all clauses in $N \cup U$ is too inefficient. A complete index assigns to every literal the sequences of clauses containing the literal. The 2-watched literal index reduces this to indexing only the first two literals of a clause. The invariant for the 2-watched literals with respect to the current trail is:

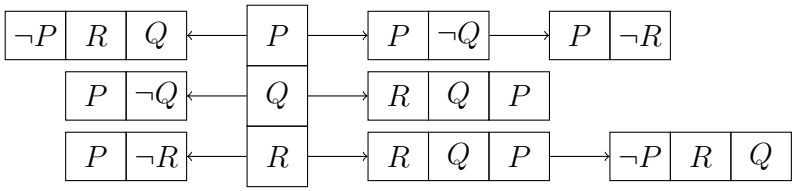
Invariant 2.10.1 (2-Watched Literal Indexing)**.** If one of the watched literals is false and the other watched literal is not true, then all other literals of the clause are false.

There are two major advantages of indexing a clause by 2WL: (i) when a literal of the clause that is not watched changes its truth value nothing needs to be done (ii) applying rule Backtrack does not result in any update on the 2WL data structure, because literals are only changed to status undefined.

For example, consider the clause set

$$N = \{P \vee \neg R,\ P \vee \neg Q,\ R \vee Q \vee P,\ \neg P \vee R \vee Q\}$$

resulting in the below 2WL index structure. Clauses are written as boxes suggesting an array implementation and the leftmost two literals are the 2WL. The entry point of the overall index is an array shown in the middle. It is indexed by propositional variables and has two entries for each variable: to the right a sequences of clauses where the positive literal of that variable is watched and to the left a sequences of clauses where the negative literal of that variable is watched. Every clause has 2-watched literals so it is linked twice in the 2WL index structure. Literals are represented by natural numbers or integers.



Note that the clause $R \vee Q \vee P$ is not indexed at literal P and the clause $\neg P \vee R \vee Q$ is not indexed at literal Q.

Although the above figure suggests two copies of each clause in the 2WL index structure, of course, there is only one shared clause. Unit clauses are considered separately, outside the index structure, since they can be immediately used for reduction. In practice, two literal clauses are considered separately as well, because they employ more efficient update and check procedures. For simplicity, the examples treat unit and two literal clauses as any other clause. Clauses, sequences of clauses and the propositional variable index structure are all implemented via arrays.

Now consider a CDCL run deciding $\neg P$:

$$\begin{array}{ll} & (\epsilon; N; \emptyset; 0; \top) \\ \Rightarrow^{\text{Decide}}_{\text{CDCL}} & (\neg P^1; N; \emptyset; 1; \top) \end{array}$$

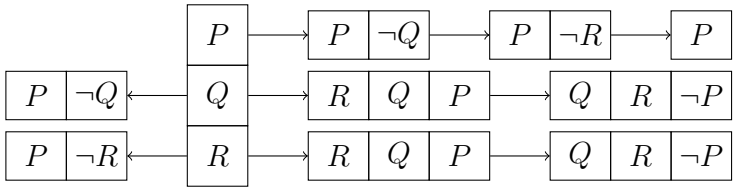
The algorithm jumps through variable P to its clause sequence on the right, because all clauses to the left of P are true. It finds the two literal clauses $P \vee \neg R$, $P \vee \neg Q$ that need not to be updated but result in the propagation of $\neg R$ and $\neg Q$. Updating means reestablishing the above 2WL Invariant 2.10.1 if it is violated. The algorithm propagates both $\neg R$ and $\neg Q$ and then runs the update procedure for both literals.

$$\begin{array}{ll} \Rightarrow^{\text{Propagate}}_{\text{CDCL}} & (\neg P^1 \neg R^{P \vee \neg R}; N; \emptyset; 1; \top) \\ \Rightarrow^{\text{Propagate}}_{\text{CDCL}} & (\neg P^1 \neg R^{P \vee \neg R} \neg Q^{P \vee \neg Q}; N; \emptyset; 1; \top) \end{array}$$

The algorithm jumps through variable R to its clause sequence on the right. The first clause it finds is $R \vee Q \vee P$ which is already false with respect to the trail. So Conflict and Resolve and finally Backtrack is applied.

$$\begin{aligned}
&\Rightarrow^{\text{Conflict}}_{\text{CDCL}} \quad (\neg P^1 \neg R^{P \vee \neg R} \neg Q^{P \vee \neg Q}; N; \emptyset; 1; R \vee Q \vee P) \\
&\Rightarrow^{\text{Resolve}}_{\text{CDCL}} \quad (\neg P^1 \neg R^{P \vee \neg R}; N; \emptyset; 1; R \vee P) \\
&\Rightarrow^{\text{Resolve}}_{\text{CDCL}} \quad (\neg P^1; N; \emptyset; 1; P) \\
&\Rightarrow^{\text{Backtrack}}_{\text{CDCL}} \quad (P^P; N; \{P\}; 0; \top)
\end{aligned}$$

The overall CDCL rule sequence does not result in any updates of the 2WL index data structure. It is always the case that executing the rules Skip, Resolve, or Backtrack does not need any update on the 2WL indexing, because the Invariant 2.10.1 is robust against removing literals from the trail. After backtracking and establishing P on the trail, the algorithm jumps through variable P to its clause sequence on the left. This time the clause $\neg P \vee R \vee Q$ violates the 2WL invariant since Q and R are undefined but $\neg P$ is false. The 2WL for this clause are changed by exchanging the literals $\neg P$ and Q and updating the 2WL index data structure.



Finally, two applications of Decide on Q and R result in a model for N without any further changes to the 2WL structure, because P is already true.

$$\begin{aligned}
&\Rightarrow^{\text{Decide}}_{\text{CDCL}} \quad (P^P Q^1; N; \{P\}; 1; \top) \\
&\Rightarrow^{\text{Decide}}_{\text{CDCL}} \quad (P^P Q^1 R^2; N; \{P\}; 2; \top)
\end{aligned}$$

An alternative derivation to a model would be to first decide on $\neg Q$.

$$\begin{aligned}
&\Rightarrow^{\text{Decide}}_{\text{CDCL}} \quad (P^P \neg Q^1; N; \{P\}; 1; \top) \\
&\Rightarrow^{\text{Propagate}}_{\text{CDCL}} \quad (P^P \neg Q^1 R^{Q \vee R \vee \neg P}; N; \{P\}; 1; \top)
\end{aligned}$$

Then R is propagated by the clause $Q \vee R \vee \neg P$ but before the clause 2WL index is changed by flipping the second and third literal in the clause $R \vee Q \vee P$ and moving it to the index P in the right sequence. Eventually, also this sequence results in a model for N.

In general, the update procedure considers the following cases, assuming a literal L is set to true by Propagate or Decide. All clauses where L is watched can be ignored, they are true anyway. So the algorithm traverses all clauses where $\text{comp}(L)$ is watched. If for some clause the other watched literal is true, nothing needs to be done. If the other literal is false or undefined, then the

clause is searched for an undefined or true literal beyond the watched literals. If it exists, the undefined/true literal and comp(L) are exchanged and the clause is removed from the comp(L) sequence and moved to the sequence of the other literal. If it does not exist, then if the second watched literal is undefined, the clause propagates that literal. If the second watched literal is false as well, then a conflict clause is found.

During the update procedure for the 2WL structure often several literals are found propagating and their respective sequences need to be considered sequentially. Still it is very useful to update their truth value already at the time they are detected.

One advantage of the 2WL data structure is that during the loop Propagate, Decide, Conflict only clauses where an involved literal is watched need to be considered. The other is that the rule Backtrack does not result in any update for the literals whose truth value is turned to undefined.

Theorem 2.10.2 (2WL and Backtrack)**.** Consider a reasonable CDCL run on a 2WL data structure satisfying the 2WL Invariant 2.10.1 before any application of rule Decide and a Backtrack application:

$(M_1 K^{i+1} M_2; N; U; k; D \vee L) \Rightarrow_{\text{Backtrack}} (M_1 L^{D \vee L}; N; U \cup \{D \vee L\}; i; \top)$.

Then the 2WL invariant is preserved except for potentially L.

Proof. First, note that $D \vee L$ is propagating L by construction, so the comp(L) sequence needs to be updated. Now assume that there is a clause $K_1 \vee K_2 \vee C$ with K_1, K_2 watched that violates the 2WL invariant and both $K_i \neq \text{comp}(L)$. This means without loss of generality that K_1 is false in $M_1 L$ and K_2 is not true and there are literals in C which are not false with respect to $M_1 L$. Note that both $K_i \neq L$ because for otherwise one of them is true. So the truth value of both K_i is determined by M_1. The above described update procedure only exchanges false literals with true or undefined literals. The rule Backtrack, except for L, only changes truth values of literals to undefined. Thus already before K^{i+1} was decided, the clause $K_1 \vee K_2 \vee C$ did violate the 2WL invariant, contradicting the assumption. □

2.10.2 Dynamic Decision Heuristic: VSIDS

The actual literal selection by the rule Decide greatly influences the number of transition steps until CDCL derives a finite state. For example, the proof of Proposition 2.8.7 presents a nice decision heuristic for some satisfiable clause set, provided the actual model is known in advance.

In general, the idea of the decision heuristic is to focus the search. Thinking of a set of clauses as a graph, where the clauses are the nodes and edges are drawn between complementary literals, the search should focus on a single connected component of the graph. Inside this component the search should focus on a small highly connected part that enables conflicts on the basis of a few decisions or has a model. For a simple example, consider the clause set $N = N_1 \uplus N_2 \uplus \{P \vee Q\} \uplus N_3$, where N_1 does not share any propositional variable with N_2, N_3,

the atom P does not occur in N_3 and the atom Q does not occur in N_2. Then an ideal decision heuristic will first focus on the atoms in N_1 before considering any atom from the rest of the clause set, or the other way round. Furthermore, atoms in N_3 should not be decided until all atoms in N_2 have been decided, or the other way round. Unfortunately, deriving the necessary graph properties for this behavior is, in general, as hard as the SAT problem. The way out are heuristics that can be efficiently computed and dynamically adapt towards the above described behavior. The *VSIDS* (Variable State Independent Decaying Sum) heuristic is such a heuristic [65, 44].

The VSIDS heuristic initially assigns each propositional variable a number, its initial *score*, e.g., its number of occurrences in the clause set, or all 0 or 1 or some random value. This score is updated during a CDCL run. The rule Decide always picks a variable with maximal score. In case the picked variable was not decided before it heuristically chooses it positively or negatively, but then stores the sign for later use. After a Backtrack or Restart application, if the variable is selected again by Decide, the stored sign us used. This is called *phase saving*. If the variable was already decided, the stored sign is selected. Now either the CDCL run produces a model or a conflict and afterwards Skip and Resolve become applicable. With each application of Resolve, the score of the resolved literal is incremented by a bonus $b > 0$. Depending on the initial score, the bonus b starts with some positive value, e.g., maximum of half the initial maximal score and it is updated as well during search. When Backtrack is applied, the scores of the remaining atoms of the learned clause are also incremented by b and b itself is updated by multiplication with a fixed constant $c > 1$, e.g., $c = \frac{6}{5}$ and for this value of c the new value for b after Backtrack is $\lceil \frac{6}{5} b \rceil$. If the score is implemented by doubles then the ceiling function application is not needed. In addition, every k^{th} application of Decide a random variable is chosen, where a typical value for k is 200.

Of course, b grows exponentially in the number of Backtrack applications. Hence, the score function as well. So eventually there will be an overflow of the respective unsigned data structures. The bonus b only depends on the number of conflicts, c, and some initial value known at the start of the CDCL procedure. Furthermore, the bit-length of the used unsigned data structure is known as well. So the minimal number of Conflicts generating the overflow can be computed once in advance. A typical value for today's computer architectures and choices for c and some initial value is above 500. So every 500 conflicts the implementation has to take care of an potential overflow by rescaling b and the score function. This can be done by right-shifting the scores of all variables and b itself. Computationally, this is a neglectable overhead, so the VSIDS heuristic can be efficiently computed. The score function is often implemented as a priority queue. The Sat solver MiniSat [38], implements VSIDS via doubles. The default for the start score value is 0, the initial bonus value is $b = 2$. The bonus increase is $c = \frac{100}{95}$. It does not precompute a bound for rescaling, instead whenever an actual increase operation exceeds a value of 10^{100} all scores and b are scaled by 10^{-100}.

The choice of a random variable every k^{th} application of Decide is a backdoor for the case that the score function accidentally focuses on a too large complicated component of the clause graph, while there exist very small beneficial components. Think of the above example, where N_1 contains only a few clauses compared to the rest and is unsatisfiable, but the VSIDS heuristic starts with atoms from N_2, because they occur by far more frequently. So the random choice part of the heuristic improves its robustness.

2.10.3 Conflict Analysis, Learning, and 1-UIPs

The SAT literature [16] discusses the relationship between an appropriate analysis of the conflict, the eventually learned clause and so called first unit implication points (1-UIPs). This section clarifies these notions in relation to the CDCL calculus of Section 2.9.

Example 2.10.3. Consider the clause set $N = \{P_1 \vee P_2 \vee \neg Q_1,\ P_2 \vee \neg Q_2,\ Q_1 \vee Q_2 \vee Q_3,\ \neg Q_3 \vee P_3 \vee \neg Q_5,\ \neg Q_3 \vee \neg Q_4,\ Q_4 \vee Q_5\}$ and a CDCL run resulting in a first conflict:

$(\epsilon; N; \emptyset; 0; \top)$

$\Rightarrow^{*}_{\text{CDCL}}\ (\neg P_3^1 \neg P_1^2 \neg P_2^3 \neg Q_2^{P_2 \vee \neg Q_2} \neg Q_1^{P_1 \vee P_2 \vee \neg Q_1} Q_3^{Q_1 \vee Q_2 \vee Q_3} \neg Q_4^{\neg Q_3 \vee \neg Q_4} \neg Q_5^{\neg Q_3 \vee P_3 \vee \neg Q_5}; N; \emptyset; 3; Q_4 \vee Q_5)$

The information of the dependencies between decision literals and propagated literals can now be represented by a directed, acyclic graph, called the *implication graph* presented in Figure 2.17. The decision literals $\neg P_3^1 \neg P_1^2 \neg P_2^3$ are the sources of the implication graph, the eventual false clause $Q_4 \vee Q_5$ the sink.

The CDCL calculus, Section 2.9, derives out of the above state in two steps the learned clause $P_3 \vee \neg Q_3$ and backtracks:

$\Rightarrow^{\text{Resolve}}_{\text{CDCL}}\ (\neg P_3^1 \neg P_1^2 \neg P_2^3 \neg Q_2^{P_2 \vee \neg Q_2} \neg Q_1^{P_1 \vee P_2 \vee \neg Q_1} Q_3^{Q_1 \vee Q_2 \vee Q_3} \neg Q_4^{\neg Q_3 \vee \neg Q_4}; N; \emptyset; 3; \neg Q_3 \vee P_3 \vee Q_4)$

$\Rightarrow^{\text{Resolve}}_{\text{CDCL}}\ (\neg P_3^1 \neg P_1^2 \neg P_2^3 \neg Q_2^{P_2 \vee \neg Q_2} \neg Q_1^{P_1 \vee P_2 \vee \neg Q_1} Q_3^{Q_1 \vee Q_2 \vee Q_3}; N; \emptyset; 3; \neg Q_3 \vee P_3)$

$\Rightarrow^{\text{Backtrack}}_{\text{CDCL}}\ (\neg P_3^1 \neg Q_3^{Q_3 \vee P_3}; N; \{Q_3 \vee P_3\}; 1; \top)$

Stopping the resolution process at clause $Q_3 \vee P_3$ is forced by the Backtrack rule and hence my version of the CDCL calculus. But this is not the only choice of generating a clause for backtracking. For example, further resolution steps can be performed:

$\Rightarrow^{\text{Resolve}}_{\text{CDCL}}\ (\neg P_3^1 \neg P_1^2 \neg P_2^3 \neg Q_2^{P_2 \vee \neg Q_2} \neg Q_1^{P_1 \vee P_2 \vee \neg Q_1} Q_3^{Q_1 \vee Q_2 \vee Q_3}; N; \emptyset; 3; \neg Q_3 \vee P_3)$

$\Rightarrow\ (\neg P_3^1 \neg P_1^2 \neg P_2^3 \neg Q_2^{P_2 \vee \neg Q_2} \neg Q_1^{P_1 \vee P_2 \vee \neg Q_1}; N; \emptyset; 3; Q_1 \vee Q_2 \vee P_3)$

$\Rightarrow\ (\neg P_3^1 \neg P_1^2 \neg P_2^3 \neg Q_2^{P_2 \vee \neg Q_2}; N; \emptyset; 3; P_1 \vee P_2 \vee Q_2 \vee P_3)$

$\Rightarrow\ (\neg P_3^1 \neg P_1^2 \neg P_2^3; N; \emptyset; 3; P_1 \vee P_2 \vee P_3)$

and then a backtrack step

$\Rightarrow\ (\neg P_3^1 \neg P_1^2 P_2^{P_1 \vee P_2 \vee P_3}; N; \{P_1 \vee P_2 \vee P_3\}; 2; \top)$

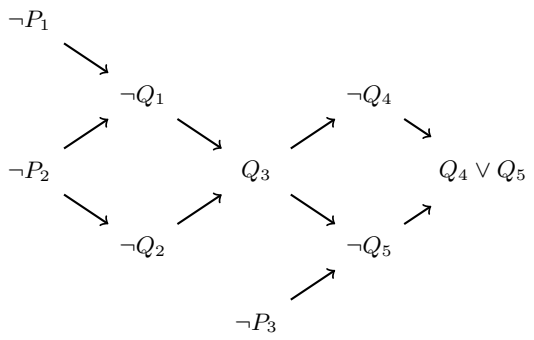


Figure 2.17: The implication graph for Example 2.10.3.

The first state can be considered superior to the second state for two reasons. Firstly, the learned clause $Q_3 \vee P_3$ of the first state has less literals than the learned clause $P_1 \vee P_2 \vee P_3$ of the second state and hence the first state has a higher probability to eventually yield a shorter terminating CDCL derivation. Recall that CDCL with a reasonable strategy never learns a clause subsumed by an already existing clause, Corollary 2.9.8. The clause $Q_3 \vee P_3$ subsumes $\mathrm{O}(n)$-times more clauses than the clause $P_1 \vee P_2 \vee P_3$ in the set of all non-tautologous clauses, where n is the number of propositional variables in N. Secondly, the backtrack level of the first state is smaller than the backtrack level of the second state.

From the implication graph it can be seen that node Q_3 dominates the nodes $\neg P_1$, $\neg P_2$: every path from these nodes to the sink traverses Q_3. This is a graph-based argument that replacing P_1, P_2 in $P_1 \vee P_2 \vee P_3$ with Q_3 results in a consequence of the clause set. The node Q_3 is called a *unique implication point (UIP)* and the condition of the Backtrack rule ensures that the CDCL calculus of Section 2.9 always learns a clause with respect to a first UIP (1-UIP) searching from the sink towards the sources.

2.10.4 Restart and Forget

So far, I have not discussed much the CDCL rules Restart and Forget. In contrast, many properties of the CDCL calculus, Section 2.9, do not hold if these rules are applied in an arbitrary way. Still, they are very important for the efficiency of a SAT solver.

The rule Forget removes clauses from the set of learned clauses U. The more clauses are contained in $N \cup U$ the more time is needed for the algorithms for propagation and conflict detection. The potential number of learned clauses is exponential in the number of propositional variables of a problem. Furthermore, as soon as the size of the computer memory that is actually needed to perform propagation and conflict detection exceeds cache structures or even main mem-

ory of a computer, the SAT solver slows down to such an extend that it often becomes practically useless. In addition, although Theorem 2.11.4 shows that learned clauses are always non-redundant, this does not exclude that a learned clause causes previously learned clauses or even input clauses to be redundant. For example, looking at a sequence of learned clauses $C_1, C_2, \ldots$ of a "typical" CDCL run, learned clause C_i subsumes its predecessor C_{i-1} with a probability of about 10%.

A reasonable CDCL run without Restart, where every learned clause is immediately forgotten via Forget after Backtrack also results in a sound and complete algorithm. It violates the invariant that the clauses propagating literals are contained in $N \cup U$, but still all rules work as desired. The rule Backtrack never deletes propagated literals left from the leftmost decision literal, so the CDCL run terminates anyway. Typically, such a run degrades to a DPLL run refined with dependency directed backtracking, called *backjumping*. Learned clauses are not needed for completeness or termination, just for efficiency.

The typical way Forget is invoked in implementations is in combination with an application of Restart. For each of the learned clauses an activity score is computed, incorporating the activity score of its literals as used by the VSIDS heuristics, see Section 2.10.2. The second established criteria is the number of decision levels a learned clause depends on for propagation. Then learned clauses are sorted with respect to the two criteria and clauses with a low score are forgotten. It may be that a forgotten clause is the justification for some propagated literal on the trail. A restart clears this potential problem.

Explaining the usefulness of the rule Restart is more difficult. It is in depth not well understood. Engineering the restart criterion is still an active are of research [6, 15]. Firstly, looking at a CDCL run without restarts, after a reasonable number of conflicts, the n decision literals on the trail will not be the maximal n literals with respect to the VSIDS heuristic. For example, assume the run starts by a first decision P^1 but the atom P is afterwards not involved in any conflict, i.e., it is not the atom of a literal resolved upon and not contained in any learned clause. Still the fact that P is true may cause unneeded propagations. It may even result in a behavior where the solver does not concentrate on "one part of the problem" but on two or more, causing a worst case exponential number of useless propagations. The rule Restart is a means to get rid of such "confusing" literals on the trail that meanwhile have a low score with respect to the dynamically evolving VSIDS heuristic. Secondly, a restart changes the status of literals on the trail. If the VSIDS variable selection heuristic did not change much after the previous restart, a restart will eventually produce a similar trail. However, the role of some literals on the trail will change from a decision literal to a propagated literal and vice versa. Learned clauses typically contain decision literals. This means, even if a trail after a restart is almost identical to the previous one, the afterwards learned clauses may look quite different because of the changed role of literals on the trail.

I MiniSat [38] combines Restart and Forget as follows: there is always a limit c for the number of learned clauses. If c clauses are learned, then they are sorted with respect to an activity score. The $\frac{c}{2}$ clauses with lowest score are thrown away, c is increased by a constant and a Restart is performed. Recall that performing a restart is needed to clear the trail. The VSIDS heuristic together with phase saving directs the search towards the same state that was generated before the restart.

2.10.5 The Overall Algorithm and Further Heuristics & Strategies

Algorithm 5 presents a CDCL solver including most aspects discussed in previous sections. It implements a reasonable strategy and includes the incorporation of the VSIDS heuristic and restarts. It does not contain a heuristic for an initial VSIDS score. Typical solutions are to start with a score of 0 for all variables or to start with the number of variable occurrences in N. Similarly for an application of the rule Decide. For a variable with maximal VSIDS score either the positive or the negative literal can be decided. Again this can be implemented via a heuristic on the number of literal occurrences in N. Important is *phase saving*: once a literal has been decided, after removal from the trail due to Restart or Backtrack, if it is decided again, it is decided with the same sign.

The restart heuristic typically considers also unit clauses. Once a unit clause is learned a restart is performed immediately. Unit clauses always propagate, so their literals are collected during a run at the start of the trail. This applies as well to literals propagating solely because of unit clauses, i.e., at level 0.

2.11 Superposition and CDCL

At the time of this writing it is often believed that the superposition (resolution) calculus is not a good choice on SAT problems in practice. Most of the successful SAT solvers implemented in 2015 are based on CDCL. In this section I will develop some relationships between superposition and CDCL. Actually, CDCL can be considered as a superposition calculus with respect to a generalized model operator.

The goal of the original model operator, Definition 2.7.6, is to create minimal models with respect to positive literals, i.e., if $N_{\mathcal{I}} \models N$ for some N, then there is no $M' \subset N_{\mathcal{I}}$ such that $M' \models N$. However, if the goal generating minimal models is dropped, then there is more freedom to construct models while preserving the general properties of the superposition calculus, in particular, the notion of redundancy. The gained freedom can be used to be more flexible when generating a partial model with respect to a given set of clauses. For example, consider the two clauses in the clause set

$$N = \{P \vee Q,\ \neg P \vee R\}$$

Algorithm 5: CDCL(S)

```
Input  : An initial state (ϵ; N; ∅; 0; ⊤).
Output: A final state S = (M; N; U; k; ⊤) or S = (M; N; U; k; ⊥)
while (any rule applicable) do
    ifrule (Conflict(S)) then
        while (Skip(S) || Resolve(S)) do
            update VSIDS scores on resolved literals;
        update VSIDS scores on learned clause;
        Backtrack(S);
        if (potential VSIDS score overflow) then
            scale VSIDS scores;
        if (forget heuristic) then
            Forget(S) clauses ;
            Restart(S);
        else
            if (restart heuristic) then
                Restart(S);
    else
        ifrule (! Propagate(S)) then
            Decide(S) literal with max. VSIDS score;
return(S);
```

with precedence $R \prec Q \prec P$. The superposition model operator generates $N_{\mathcal{I}} = \{P\}$ which is not a model for N. However, this model can be extended to a model for N by adding R to it. The superposition model operator does not include R because it is not maximal in the second clause. Starting with a decision on P, the CDCL calculus derives the model P, R via propagation. In the sequel, I show that a generalized superposition model operator can in fact generate this model as well.

In addition to an ordering $\prec$ I assume a decision heuristic $\mathcal{H}$ that selects whether a literal should be productive, i.e., included in the model, or not.

Definition 2.11.1 (Heuristic-Based Partial Model Construction)**.** Given a clause set N, a set of propositional variables $M \subseteq \Sigma$, a total ordering $\prec$, and a variable heuristic $\mathcal{H} : \Sigma \to \{0, 1\}$, the (partial) model $N_M^{\mathcal{H}}$ for N with $P, Q \in M$