Advanced CNF Algorithm

For the formula

\[ P_1 \leftrightarrow (P_2 \leftrightarrow (P_3 \leftrightarrow (\ldots (P_{n-1} \leftrightarrow P_n) \ldots))) \]

the basic CNF algorithm generates a CNF with \(2^{n-1}\) clauses.
2.5.4 Proposition (Renaming Variables)

Let $P$ be a propositional variable not occurring in $\psi[\phi]_p$.

1. If $\text{pol}(\psi, p) = 1$, then $\psi[\phi]_p$ is satisfiable if and only if $\psi[P]_p \land (P \rightarrow \phi)$ is satisfiable.

2. If $\text{pol}(\psi, p) = -1$, then $\psi[\phi]_p$ is satisfiable if and only if $\psi[P]_p \land (\phi \rightarrow P)$ is satisfiable.

3. If $\text{pol}(\psi, p) = 0$, then $\psi[\phi]_p$ is satisfiable if and only if $\psi[P]_p \land (P \leftrightarrow \phi)$ is satisfiable.
Renaming

**SimpleRenaming**

\[ \phi \Rightarrow \text{SimpRen} \phi[P_1]_{p_1} [P_2]_{p_2} \ldots [P_n]_{p_n} \land \]
\[ \text{def}(\phi, p_1, P_1) \land \ldots \land \text{def}(\phi[P_1]_{p_1} [P_2]_{p_2} \ldots [P_{n-1}]_{p_{n-1}}, p_n, P_n) \]

provided \( \{p_1, \ldots, p_n\} \subset \text{pos}(\phi) \) and for all \( i, i + j \) either \( p_i \parallel p_{i+j} \) or \( p_i > p_{i+j} \) and the \( P_i \) are different and new to \( \phi \).

Simple choice: choose \( \{p_1, \ldots, p_n\} \) to be all non-literal and non-negation positions of \( \phi \).
Renaming Definition

\[
def(\psi, p, P) := \begin{cases} 
(P \rightarrow \psi|_p) & \text{if } \text{pol}(\psi, p) = 1 \\
(\psi|_p \rightarrow P) & \text{if } \text{pol}(\psi, p) = -1 \\
(P \leftrightarrow \psi|_p) & \text{if } \text{pol}(\psi, p) = 0
\end{cases}
\]
Obvious Positions

A smaller set of positions from $\phi$, called *obvious positions*, is still preventing the explosion and given by the rules:

(i) $p$ is an obvious position if $\phi|_p$ is an equivalence and there is a position $q < p$ such that $\phi|_q$ is either an equivalence or disjunctive in $\phi$ or

(ii) $pq$ is an obvious position if $\phi|_{pq}$ is a conjunctive formula in $\phi$, $\phi|_p$ is a disjunctive formula in $\phi$, $q \neq \epsilon$, and for all positions $r$ with $p < r < pq$ the formula $\phi|_r$ is not a conjunctive formula.

A formula $\phi|_p$ is conjunctive in $\phi$ if $\phi|_p$ is a conjunction and $\text{pol}(\phi, p) \in \{0, 1\}$ or $\phi|_p$ is a disjunction or implication and $\text{pol}(\phi, p) \in \{0, -1\}$.

Analogously, a formula $\phi|_p$ is disjunctive in $\phi$ if $\phi|_p$ is a disjunction or implication and $\text{pol}(\phi, p) \in \{0, 1\}$ or $\phi|_p$ is a conjunction and $\text{pol}(\phi, p) \in \{0, -1\}$.
Polarity Dependent Equivalence
Elimination

**ElimEquiv1** \[ \chi[(\phi \leftrightarrow \psi)]_p \Rightarrow_{\text{ACNF}} \chi[(\phi \rightarrow \psi) \land (\psi \rightarrow \phi)]_p \]
providing \( \text{pol}(\chi, p) \in \{0, 1\} \)

**ElimEquiv2** \[ \chi[(\phi \leftrightarrow \psi)]_p \Rightarrow_{\text{ACNF}} \chi[(\phi \land \psi) \lor (\neg \phi \land \neg \psi)]_p \]
providing \( \text{pol}(\chi, p) = -1 \)
Extra ⊤, ⊥ Elimination Rules

\[
\begin{align*}
\text{ElimTB7} & : \chi[\phi \rightarrow \bot]_p \Rightarrow \text{ACNF} \chi[\neg \phi]_p \\
\text{ElimTB8} & : \chi[\bot \rightarrow \phi]_p \Rightarrow \text{ACNF} \chi[\top]_p \\
\text{ElimTB9} & : \chi[\phi \rightarrow \top]_p \Rightarrow \text{ACNF} \chi[\top]_p \\
\text{ElimTB10} & : \chi[\top \rightarrow \phi]_p \Rightarrow \text{ACNF} \chi[\phi]_p \\
\text{ElimTB11} & : \chi[\phi \leftrightarrow \bot]_p \Rightarrow \text{ACNF} \chi[\neg \phi]_p \\
\text{ElimTB12} & : \chi[\phi \leftrightarrow \top]_p \Rightarrow \text{ACNF} \chi[\phi]_p
\end{align*}
\]

where the two rules ElimTB11, ElimTB12 for equivalences are applied with respect to commutativity of ↔.
Advanced CNF Algorithm

Algorithm: 3 \( \text{acnf}(\phi) \)

Input : A formula \( \phi \).
Output : A formula \( \psi \) in CNF satisfiability preserving to \( \phi \).

whilerule (ElimTB1(\( \phi \)),..,ElimTB12(\( \phi \))) do ;

SimpleRenaming(\( \phi \)) on obvious positions;

whilerule (ElimEquiv1(\( \phi \)),ElimEquiv2(\( \phi \))) do ;

whilerule (ElimImp(\( \phi \))) do ;

whilerule (PushNeg1(\( \phi \)),..,PushNeg3(\( \phi \))) do ;

whilerule (PushDisj(\( \phi \))) do ;

return \( \psi \);
Propositional Resolution

The propositional resolution calculus operates on a set of clauses and tests unsatisfiability.

Recall that for clauses I switch between the notation as a disjunction, e.g., $P \lor Q \lor P \lor \neg R$, and the multiset notation, e.g., \{ $P, Q, P, \neg R$ \}. This makes no difference as we consider $\lor$ in the context of clauses always modulo AC. Note that $\bot$, the empty disjunction, corresponds to $\emptyset$, the empty multiset. Clauses are typically denoted by letters $C, D$, possibly with subscript.
Resolution Inference Rules

**Resolution**

\[
\begin{align*}
\text{Resolution} & \quad (N \cup \{ C_1 \lor P, C_2 \lor \neg P \}) \implies \text{RES} \\
& \quad (N \cup \{ C_1 \lor P, C_2 \lor \neg P \} \cup \{ C_1 \lor C_2 \})
\end{align*}
\]

**Factoring**

\[
\begin{align*}
\text{Factoring} & \quad (N \cup \{ C \lor L \lor L \}) \implies \text{RES} \\
& \quad (N \cup \{ C \lor L \lor L \} \cup \{ C \lor L \})
\end{align*}
\]
2.6.1 Theorem (Soundness & Completeness)

The resolution calculus is sound and complete: 

\( N \) is unsatisfiable iff \( N \Rightarrow^{*}_{\text{RES}} N' \) and \( \bot \in N' \) for some \( N' \)
Resolution Reduction Rules

Subsumption

\( (N \uplus \{ C_1, C_2 \}) \Rightarrow_{\text{RES}} (N \uplus \{ C_1 \}) \)
provided \( C_1 \subset C_2 \)

Tautology Deletion

\( (N \uplus \{ C \lor P \lor \neg P \}) \Rightarrow_{\text{RES}} (N) \)

Condensation

\( (N \uplus \{ C_1 \lor L \lor L \}) \Rightarrow_{\text{RES}} (N \uplus \{ C_1 \lor L \}) \)

Subsumption Resolution

\( (N \uplus \{ C_1 \lor L, C_2 \lor \text{comp}(L) \}) \Rightarrow_{\text{RES}} (N \cup \{ C_1 \lor L, C_2 \}) \)
where \( C_1 \subseteq C_2 \)
2.6.6 Theorem (Resolution Termination)

If reduction rules are preferred over inference rules and no inference rule is applied twice to the same clause(s), then $\Rightarrow^{+}_{\text{RES}}$ is well-founded.
## The Overall Picture

<table>
<thead>
<tr>
<th>Application</th>
<th>System + Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>Algorithm + Implementation</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Calculus + Strategy</td>
</tr>
<tr>
<td>Calculus</td>
<td>Logic + States + Rules</td>
</tr>
<tr>
<td>Logic</td>
<td>Syntax + Semantics</td>
</tr>
</tbody>
</table>
Conflict Driven Clause Learning (CDCL)

The CDCL calculus tests satisfiability of a finite set $N$ of propositional clauses.

I assume that $\bot \not\in N$ 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. (Exhaustive Condensation.)
The CDCL calculus explicitly 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 $L_1, \ldots, L_n$, CDCL generates in addition a new clause, called *learned clause* via resolution, 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 or Tableau.
CDCL State

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.
Modes of CDCL States

$$(\epsilon; N; \emptyset; 0; \top)$$ 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 \not\in \{\top, \bot\}$$