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 $\mathrm{pol}(\psi, p) = 1$, then $\psi[\phi]_p$ is satisfiable if and only if $\psi[P]_p \wedge (P \rightarrow \phi)$ is satisfiable.
2. If $\mathrm{pol}(\psi, p) = -1$, then $\psi[\phi]_p$ is satisfiable if and only if $\psi[P]_p \wedge (\phi \rightarrow P)$ is satisfiable.
3. If $\mathrm{pol}(\psi, p) = 0$, then $\psi[\phi]_p$ is satisfiable if and only if $\psi[P]_p \wedge (P \leftrightarrow \phi)$ 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 ϕ

Simple choice: choose $\{p_1, \ldots, p_n\}$ to be all non-literal and non-negation positions of ϕ.

Renaming Definition

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

Obvious Positions

A smaller set of positions from ϕ, 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 ϕ or

(ii) pq is an obvious position if $\phi|_{pq}$ is a conjunctive formula in ϕ, $\phi|_p$ is a disjunctive formula in ϕ, $q \neq \epsilon$, 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 ϕ if $\phi|_p$ is a conjunction and $\mathrm{pol}(\phi, p) \in \{0, 1\}$ or $\phi|_p$ is a disjunction or implication and $\mathrm{pol}(\phi, p) \in \{0, -1\}$.

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

Polarity Dependent Equivalence Elimination

ElimEquiv1 $\chi[(\phi \leftrightarrow \psi)]_p \Rightarrow_{\text{ACNF}} \chi[(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)]_p$
provided $\text{pol}(\chi, p) \in \{0, 1\}$

ElimEquiv2 $\chi[(\phi \leftrightarrow \psi)]_p \Rightarrow_{\text{ACNF}} \chi[(\phi \wedge \psi) \vee (\neg\phi \wedge \neg\psi)]_p$
provided $\text{pol}(\chi, p) = -1$

Extra $\top, \bot$ Elimination Rules

ElimTB7	$\chi[\phi \rightarrow \bot]_p$	$\Rightarrow_{\text{ACNF}}$	$\chi[\neg\phi]_p$
ElimTB8	$\chi[\bot \rightarrow \phi]_p$	$\Rightarrow_{\text{ACNF}}$	$\chi[\top]_p$
ElimTB9	$\chi[\phi \rightarrow \top]_p$	$\Rightarrow_{\text{ACNF}}$	$\chi[\top]_p$
ElimTB10	$\chi[\top \rightarrow \phi]_p$	$\Rightarrow_{\text{ACNF}}$	$\chi[\phi]_p$
ElimTB11	$\chi[\phi \leftrightarrow \bot]_p$	$\Rightarrow_{\text{ACNF}}$	$\chi[\neg\phi]_p$
ElimTB12	$\chi[\phi \leftrightarrow \top]_p$	$\Rightarrow_{\text{ACNF}}$	$\chi[\phi]_p$

where the two rules ElimTB11, ElimTB12 for equivalences are applied with respect to commutativity of $\leftrightarrow$.

Advanced CNF Algorithm

Algorithm: 3 acnf(ϕ)

Input : A formula ϕ.

Output : A formula ψ in CNF satisfiability preserving to ϕ.

whilerule *(***ElimTB1**(ϕ),...,**ElimTB12**(ϕ)*)* **do** ;

;

SimpleRenaming(ϕ) on obvious positions;

whilerule *(***ElimEquiv1**(ϕ),**ElimEquiv2**(ϕ)*)* **do** ;

;

whilerule *(***ElimImp**(ϕ)*)* **do** ;

;

whilerule *(***PushNeg1**(ϕ),...,**PushNeg3**(ϕ)*)* **do** ;

;

whilerule *(***PushDisj**(ϕ)*)* **do** ;

;

return ϕ;

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 $(N \uplus \{C_1 \lor P, C_2 \lor \neg P\}) \;\Rightarrow_{\mathrm{RES}}$
$(N \cup \{C_1 \lor P, C_2 \lor \neg P\} \cup \{C_1 \lor C_2\})$

Factoring $(N \uplus \{C \lor L \lor L\}) \;\Rightarrow_{\mathrm{RES}}$
$(N \cup \{C \lor L \lor L\} \cup \{C \lor L\})$

2.6.1 Theorem (Soundness & Completeness)

The resolution calculus is sound and complete:

$$N \text{ is unsatisfiable iff } N \Rightarrow^{*}_{\mathrm{RES}} N' \text{ and } \bot \in N' \text{ for some } N'$$

Resolution Reduction Rules

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

Tautology Deletion $(N \uplus \{C \vee P \vee \neg P\}) \Rightarrow_{\text{RES}} (N)$

Condensation $(N \uplus \{C_1 \vee L \vee L\}) \Rightarrow_{\text{RES}} (N \cup \{C_1 \vee L\})$

Subsumption Resolution $(N \uplus \{C_1 \vee L, C_2 \vee \text{comp}(L)\}) \Rightarrow_{\text{RES}}$
$(N \cup \{C_1 \vee L, C_2\})$
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 $\Rightarrow^{+}_{\mathrm{RES}}$ is well-founded.

The Overall Picture

Application System + Problem
System Algorithm + Implementation
Algorithm Calculus + Strategy
Calculus Logic + States + Rules
Logic Syntax + Semantics

Conflict Driven Clause Learning (CDCL)

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

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. (Exhaustive Condensation.)

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 $L_1, \ldots, L_n$, 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 $D \notin \{\top, \bot\}$