2.5 Improving the CNF Transformation

The goal

"Given a formula F, find an *equivalent* formula G in CNF"

is unpractical.

But if we relax the requirement to

"Given a formula F, find an *equisatisfiable* formula G in CNF"

we can get an efficient transformation.

Literature:

Andreas Nonnengart and Christoph Weidenbach: Computing small clause normal forms, in *Handbook of Automated Reasoning*, pages 335-367. Elsevier, 2001.

Christoph Weidenbach: Automated Reasoning (Chapter 2). Textbook draft, available for registered participants in the lecture Nextcloud (same link as for the online session recordings), 2021.

Tseitin Transformation

Proposition 2.13 *A formula $H[F]_p$ is satisfiable if and only if $H[Q]_p \wedge (Q \leftrightarrow F)$ is satisfiable, where Q is a new propositional variable that works as an abbreviation for F.*

Proof. (i) "$\Rightarrow$": Suppose that the Π-formula $H[F]_p$ is satisfiable. Let $\mathcal{A}$ be a Π-valuation such that $\mathcal{A}(H[F]_p) = 1$. Let Q be a new propositional variable (that is, a variable that is not contained in Π). Let $\Pi' = \Pi \cup \{Q\}$ and let $\mathcal{A}'$ be the Π'-valuation defined by $\mathcal{A}'(P) = \mathcal{A}(P)$ for all $P \in \Pi$ and $\mathcal{A}'(Q) = \mathcal{A}(F)$. Since $H[F]_p$ is a Π-formula, we have $\mathcal{A}'(H[F]_p) = \mathcal{A}(H[F]_p) = 1$ and $\mathcal{A}'(F) = \mathcal{A}(F)$. Therefore $\mathcal{A}'(Q) = \mathcal{A}'(F)$ and by Prop. 2.8 $\mathcal{A}'(H[Q]_p) = \mathcal{A}'(H[F]_p) = 1$, thus $\mathcal{A}'(H[Q]_p \wedge (Q \leftrightarrow F)) = 1$.

(ii) "$\Leftarrow$": Let $\Pi' = \Pi \cup \{Q\}$. Suppose that the Π'-formula $H[Q]_p \wedge (Q \leftrightarrow F)$ is satisfiable. Let $\mathcal{A}'$ be a Π'-valuation such that $\mathcal{A}'(H[Q]_p \wedge (Q \leftrightarrow F)) = 1$. Then $\mathcal{A}'(H[Q]_p) = 1$ and $\mathcal{A}'(Q) = \mathcal{A}'(F)$, so by Prop. 2.8 $\mathcal{A}'(H[F]_p) = \mathcal{A}'(H[Q]_p) = 1$. $\square$

Satisfiability-preserving CNF transformation (Tseitin 1970):

Apply Prop. 2.13 recursively bottom-up to all subformulas F in the original formula (except $\bot$, $\top$, and atomic formulas). This introduces a linear number of new propositional variables Q and definitions $Q \leftrightarrow F$.

Convert the resulting conjunction to CNF. This increases the size only by an additional factor, since each formula $Q \leftrightarrow F$ yields at most four clauses in the CNF.

Polarity-based CNF Transformation

A further improvement is possible by taking the *polarity* of the subformula F into account (Plaisted and Greenbaum 1986):

Proposition 2.14 *Let $\mathcal{A}$ be a valuation, let F and G be formulas, and let $H = H[F]_p$ be a formula in which F occurs as a subformula at position p.*

If $\mathrm{pol}(H,p) = 1$ and $\mathcal{A}(F) \leq \mathcal{A}(G)$, then $\mathcal{A}(H[F]_p) \leq \mathcal{A}(H[G]_p)$.

If $\mathrm{pol}(H,p) = -1$ and $\mathcal{A}(F) \geq \mathcal{A}(G)$, then $\mathcal{A}(H[F]_p) \leq \mathcal{A}(H[G]_p)$.

Proof. Exercise. □

Let Q be a propositional variable not occurring in $H[F]_p$.

Define the formula $\mathrm{def}(H,p,Q,F)$ by

- $(Q \rightarrow F)$, if $\mathrm{pol}(H,p) = 1$,
- $(F \rightarrow Q)$, if $\mathrm{pol}(H,p) = -1$,
- $(Q \leftrightarrow F)$, if $\mathrm{pol}(H,p) = 0$.

Proposition 2.15 *Let Q be a propositional variable not occurring in $H[F]_p$. Then $H[F]_p$ is satisfiable if and only if $H[Q]_p \wedge \mathrm{def}(H,p,Q,F)$ is satisfiable.*

Proof. ($\Rightarrow$) Since $H[F]_p$ is satisfiable, there exists a Π-valuation $\mathcal{A}$ such that $\mathcal{A} \models H[F]_p$. Let $\Pi' = \Pi \cup \{Q\}$ and define the Π'-valuation $\mathcal{A}'$ by $\mathcal{A}'(P) = \mathcal{A}(P)$ for $P \in \Pi$ and $\mathcal{A}'(Q) = \mathcal{A}(F)$. Obviously $\mathcal{A}'(\mathrm{def}(H,p,Q,F)) = 1$; moreover $\mathcal{A}'(H[Q]_p) = \mathcal{A}'(H[F]_p) = \mathcal{A}(H[F]_p) = 1$ by Prop. 2.8, so $H[Q]_p \wedge \mathrm{def}(H,p,Q,F)$ is satisfiable.

($\Leftarrow$) Let $\mathcal{A}$ be a valuation such that $\mathcal{A} \models H[Q]_p \wedge \mathrm{def}(H,p,Q,F)$. So $\mathcal{A}(H[Q]_p) = 1$ and $\mathcal{A}(\mathrm{def}(H,p,Q,F)) = 1$. We will show that $\mathcal{A} \models H[F]_p$.

If $\mathrm{pol}(H,p) = 0$, then $\mathrm{def}(H,p,Q,F) = (Q \leftrightarrow F)$, so $\mathcal{A}(Q) = \mathcal{A}(F)$, hence $\mathcal{A}(H[F]_p) = \mathcal{A}(H[Q]_p) = 1$ by Prop. 2.8.

If $\mathrm{pol}(H,p) = 1$, then $\mathrm{def}(H,p,Q,F) = (Q \rightarrow F)$, so $\mathcal{A}(Q) \leq \mathcal{A}(F)$. By Prop. 2.14, $\mathcal{A}(H[F]_p) \geq \mathcal{A}(H[Q]_p) = 1$, so $\mathcal{A}(H[F]_p) = 1$.

If $\mathrm{pol}(H,p) = -1$, then $\mathrm{def}(H,p,Q,F) = (F \rightarrow Q)$, so $\mathcal{A}(F) \leq \mathcal{A}(Q)$. By Prop. 2.14, $\mathcal{A}(H[F]_p) \geq \mathcal{A}(H[Q]_p) = 1$, so $\mathcal{A}(H[F]_p) = 1$. □

Optimized CNF

Not every introduction of a definition for a subformula leads to a smaller CNF.

The number of potentially generated clauses is a good indicator for useful CNF transformations.

The functions $\nu(F)$ and $\bar{\nu}(F)$ give us upper bounds for the number of clauses in $\mathrm{cnf}(F)$ and $\mathrm{cnf}(\neg F)$ using a naive CNF transformation.

G	$\nu(G)$	$\bar{\nu}(G)$
$P, \top, \bot$	1	1
$F_1 \land F_2$	$\nu(F_1) + \nu(F_2)$	$\bar{\nu}(F_1)\bar{\nu}(F_2)$
$F_1 \lor F_2$	$\nu(F_1)\nu(F_2)$	$\bar{\nu}(F_1) + \bar{\nu}(F_2)$
$\neg F_1$	$\bar{\nu}(F_1)$	$\nu(F_1)$
$F_1 \rightarrow F_2$	$\bar{\nu}(F_1)\nu(F_2)$	$\nu(F_1) + \bar{\nu}(F_2)$
$F_1 \leftrightarrow F_2$	$\nu(F_1)\bar{\nu}(F_2) + \bar{\nu}(F_1)\nu(F_2)$	$\nu(F_1)\nu(F_2) + \bar{\nu}(F_1)\bar{\nu}(F_2)$

A better CNF transformation (Nonnengart and Weidenbach 2001):

Step 1: Exhaustively apply modulo commutativity of $\leftrightarrow$ and associativity/commutativity of $\land$, $\lor$:

$$\begin{aligned}
H[(F \land \top)]_p &\Rightarrow_{\mathrm{OCNF}} H[F]_p \\
H[(F \lor \bot)]_p &\Rightarrow_{\mathrm{OCNF}} H[F]_p \\
H[(F \leftrightarrow \bot)]_p &\Rightarrow_{\mathrm{OCNF}} H[\neg F]_p \\
H[(F \leftrightarrow \top)]_p &\Rightarrow_{\mathrm{OCNF}} H[F]_p \\
H[(F \lor \top)]_p &\Rightarrow_{\mathrm{OCNF}} H[\top]_p \\
H[(F \land \bot)]_p &\Rightarrow_{\mathrm{OCNF}} H[\bot]_p
\end{aligned}$$

$$\begin{aligned}
H[(F \land F)]_p &\Rightarrow_{\mathrm{OCNF}} H[F]_p \\
H[(F \lor F)]_p &\Rightarrow_{\mathrm{OCNF}} H[F]_p \\
H[(F \land (F \lor G))]_p &\Rightarrow_{\mathrm{OCNF}} H[F]_p \\
H[(F \lor (F \land G))]_p &\Rightarrow_{\mathrm{OCNF}} H[F]_p \\
H[(F \land \neg F)]_p &\Rightarrow_{\mathrm{OCNF}} H[\bot]_p \\
H[(F \lor \neg F)]_p &\Rightarrow_{\mathrm{OCNF}} H[\top]_p \\
H[\neg\top]_p &\Rightarrow_{\mathrm{OCNF}} H[\bot]_p \\
H[\neg\bot]_p &\Rightarrow_{\mathrm{OCNF}} H[\top]_p
\end{aligned}$$

$$
\begin{aligned}
H[(F \to \bot)]_p &\Rightarrow_{\mathrm{OCNF}} H[\neg F]_p \\
H[(F \to \top)]_p &\Rightarrow_{\mathrm{OCNF}} H[\top]_p \\
H[(\bot \to F)]_p &\Rightarrow_{\mathrm{OCNF}} H[\top]_p \\
H[(\top \to F)]_p &\Rightarrow_{\mathrm{OCNF}} H[F]_p
\end{aligned}
$$

Note: Applying the absorption laws exhaustively modulo associativity/commutativity of $\wedge$ and $\vee$ is expensive. In practice, it is sufficient to apply them only in those cases that are easy to detect.

Step 2: Introduce top-down fresh variables for beneficial subformulas:

$$H[F]_p \Rightarrow_{\mathrm{OCNF}} H[Q]_p \wedge \mathrm{def}(H,p,Q,F)$$

where Q is new to $H[F]_p$ and $\nu(H[F]_p) > \nu(H[Q]_p \wedge \mathrm{def}(H,p,Q,F))$.

Remark: Although computing ν is not practical in general, the test $\nu(H[F]_p) > \nu(H[Q]_p \wedge \mathrm{def}(H,p,Q,F))$ can be computed in constant time.

Step 3: Eliminate equivalences dependent on their polarity:

$$H[F \leftrightarrow G]_p \Rightarrow_{\mathrm{OCNF}} H[(F \to G) \wedge (G \to F)]_p$$

if $\mathrm{pol}(F,p) = 1$ or $\mathrm{pol}(F,p) = 0$.

$$H[F \leftrightarrow G]_p \Rightarrow_{\mathrm{OCNF}} H[(F \wedge G) \vee (\neg F \wedge \neg G)]_p$$

if $\mathrm{pol}(F,p) = -1$.

Step 4: Apply steps 2, 3, 4, 5 of $\Rightarrow_{\mathrm{CNF}}$

Remark: The $\Rightarrow_{\mathrm{OCNF}}$ algorithm is already close to a state of the art algorithm, but some additional redundancy tests and simplification mechanisms are missing.

2.6 The DPLL Procedure

Goal:
Given a propositional formula in CNF (or alternatively, a finite set N of clauses), check whether it is satisfiable (and optionally: output *one* solution, if it is satisfiable).

Preliminaries

Recall:

$\mathcal{A} \models N$ if and only if $\mathcal{A} \models C$ for all clauses C in N.

$\mathcal{A} \models C$ if and only if $\mathcal{A} \models L$ for some literal $L \in C$.

Assumptions:

Clauses contain neither duplicated literals nor complementary literals.

The order of literals in a clause is irrelevant.

$\Rightarrow$ Clauses behave like *sets* of literals.

Notation:

We use the notation $C \vee L$ to denote a clause with some literal L and a clause rest C. Here L need *not* be the last literal of the clause and C may be empty.

$\overline{L}$ is the complementary literal of L, i. e., $\overline{P} = \neg P$ and $\overline{\neg P} = P$.

Partial Valuations

Since we will construct satisfying valuations incrementally, we consider *partial valuations* (that is, partial mappings $\mathcal{A} : \Pi \to \{0, 1\}$).

Every partial valuation $\mathcal{A}$ corresponds to a set M of literals that does not contain complementary literals, and vice versa:

$\mathcal{A}(L)$ is true, if $L \in M$.

$\mathcal{A}(L)$ is false, if $\overline{L} \in M$.

$\mathcal{A}(L)$ is undefined, if neither $L \in M$ nor $\overline{L} \in M$.

We will use $\mathcal{A}$ and M interchangeably.

A clause is true under a partial valuation $\mathcal{A}$ (or under a set M of literals) if one of its literals is true; it is false (or *"conflicting"*) if all its literals are false; otherwise it is undefined (or *"unresolved"*).

Unit Clauses

Observation:
Let $\mathcal{A}$ be a partial valuation. If the set N contains a clause C, such that all literals but one in C are false under $\mathcal{A}$, then the following properties are equivalent:

- there is a valuation that is a model of N and extends $\mathcal{A}$.
- there is a valuation that is a model of N and extends $\mathcal{A}$ and makes the remaining literal L of C true.

C is called a *unit clause;* L is called a *unit literal.*

Pure Literals

One more observation:
Let $\mathcal{A}$ be a partial valuation and P a variable that is undefined under $\mathcal{A}$. If P occurs only positively (or only negatively) in the unresolved clauses in N, then the following properties are equivalent:

- there is a valuation that is a model of N and extends $\mathcal{A}$.
- there is a valuation that is a model of N and extends $\mathcal{A}$ and assigns 1 (0) to P.

P is called a *pure literal.*

The Davis-Putnam-Logemann-Loveland Procedure

```
boolean DPLL(literal set $M$, clause set $N$) {
    if (all clauses in $N$ are true under $M$) return true;
    elsif (some clause in $N$ is false under $M$) return false;
    elsif ($N$ contains unit literal $P$) return DPLL($M \cup \{P\}$, $N$);
    elsif ($N$ contains unit literal $\neg P$) return DPLL($M \cup \{\neg P\}$, $N$);
    elsif ($N$ contains pure literal $P$) return DPLL($M \cup \{P\}$, $N$);
    elsif ($N$ contains pure literal $\neg P$) return DPLL($M \cup \{\neg P\}$, $N$);
    else {
        let $P$ be some undefined variable in $N$;
        if (DPLL($M \cup \{\neg P\}$, $N$)) return true;
        else return DPLL($M \cup \{P\}$, $N$);
    }
}
```

Initially, DPLL is called with an empty literal set and the clause set N.

2.7 From DPLL to CDCL

The DPLL procedure can be improved significantly:

The pure literal check is only done while preprocessing (otherwise is too expensive).

The algorithm is implemented iteratively $\Rightarrow$ the backtrack stack is managed explicitly (it may be possible and useful to backtrack more than one level).

Information is reused by conflict analysis and learning.

The branching variable is not chosen randomly.

Under certain circumstances, the procedure is restarted.

Literature:

Lintao Zhang and Sharad Malik: The Quest for Efficient Boolean Satisfiability Solvers, Proc. CADE-18, LNAI 2392, pp. 295–312, Springer, 2002.

Robert Nieuwenhuis, Albert Oliveras, Cesare Tinelli: Solving SAT and SAT Modulo Theories – From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T), pp. 937–977, Journal of the ACM, 53(6), 2006.

Armin Biere, Marijn Heule, Hans van Maaren, Toby Walsh (eds.): Handbook of Satisfiability, IOS Press, 2009

Daniel Le Berre's slides at VTSA'09: `http://www.mpi-inf.mpg.de/vtsa09/`.

Conflict Analysis and Learning

Conflict analysis serves a dual purpose:

Backjumping (non-chronological backtracking): If we detect that the conflict is independent of some earlier branch, we can skip over that backtrack level.

Learning: By deriving a new clause from the conflict that is added to the current set of clauses, we can reuse information that is obtained in one branch in further branches. (Note: This may produce a large number of new clauses; therefore it may become necessary to delete some of them afterwards to save space.)

These ideas are implemented in all modern SAT solvers.

Because of the importance of clause learning the algorithm is now called CDCL: Conflict Driven Clause Learning.

Formalizing CDCL

We model the improved DPLL procedure by a transition relation $\Rightarrow_{\text{CDCL}}$ on a set of states.

States:

- *fail*
- $M \parallel N$,

where M is a *list of annotated literals ("trail")* and N is a set of clauses.

Annotated literal:

- L: deduced literal, due to unit propagation.
- L^{d}: decision literal (guessed literal).

Unit Propagate:

$M \parallel N \cup \{C \vee L\} \;\Rightarrow_{\text{CDCL}}\; M\ L \parallel N \cup \{C \vee L\}$

if C is false under M and L is undefined under M.

Decide:

$M \parallel N \;\Rightarrow_{\text{CDCL}}\; M\ L^{\text{d}} \parallel N$

if L is undefined under M and contained in N.

Fail:

$M \parallel N \cup \{C\} \;\Rightarrow_{\text{CDCL}}\; \textit{fail}$

if C is false under M and M contains no decision literals.

Backjump:

$M'\ L^{\text{d}}\ M'' \parallel N \;\Rightarrow_{\text{CDCL}}\; M'\ L' \parallel N$

if there is some "backjump clause" $C \vee L'$ such that
$N \models C \vee L'$,
C is false under M', and
L' is undefined under M'.

We will see later that the Backjump rule is always applicable, if the list of literals M contains at least one decision literal and some clause in N is false under M.

There are many possible backjump clauses. One candidate: $\overline{L_1} \vee \ldots \vee \overline{L_n}$, where the L_i are all the decision literals in $M'\ L^{\text{d}}\ M''$. (But usually there are better choices.)

Lemma 2.16 *If we reach a state $M \parallel N$ starting from $\varepsilon \parallel N$, then:*

(1) M does not contain complementary literals.

(2) Every deduced literal L in M follows from N and decision literals occurring before L in M.

Proof. By induction on the length of the derivation. □

Lemma 2.17 *Every derivation starting from $\varepsilon \parallel N$ terminates.*

Proof. Let $M \parallel N$ and $M' \parallel N'$ be two CDCL states, such that $M = M_0 L_1^{\mathrm{d}} M_1 \dots L_k^{\mathrm{d}} M_k$ and $M' = M'_0 L_1'^{\mathrm{d}} M'_1 \dots L_{k'}'^{\mathrm{d}} M'_{k'}$. We define a relation $\succ$ on lists of annotated literals by $M \succ M'$ if and only if

(i) there is some j such that $0 \leq j \leq \min(k, k')$, $|M_i| = |M'_i|$ for all $0 \leq i < j$, and $|M_j| < |M'_j|$, or

(ii) $|M_i| = |M'_i|$ for all $0 < i \leq k < k'$ and $|M| < |M'|$.

It is routine to check that $\succ$ is irreflexive and transitive, hence a strict partial ordering, and that for every CDCL step $M \parallel N \Rightarrow_{\text{CDCL}} M' \parallel N'$ we have $M \succ M'$. Moreover, the set of propositional variables in N is finite, and each of these variables can occur at most once in a literal list (positively or negatively, with or without a d-superscript). So there are only finitely many literal lists that can occur in a CDCL derivation. Consequently, if there were an infinite CDCL derivation, there would be some cycle $M \parallel N \Rightarrow^{+}_{\text{CDCL}} M \parallel N'$, so by transitivity $M \succ M$, but that would contradict the irreflexivity of $\succ$.

Lemma 2.18 *Suppose that we reach a state $M \parallel N$ starting from $\varepsilon \parallel N$ such that some clause $D \in N$ is false under M. Then:*

(1) If M does not contain any decision literal, then "Fail" is applicable.

(2) Otherwise, "Backjump" is applicable.

Proof. (1) Obvious.

(2) Let $L_1, \dots, L_n$ be the decision literals occurring in M (in this order). Since $M \models \neg D$, we obtain, by Lemma 2.16, $N \cup \{L_1, \dots, L_n\} \models \neg D$. Since $D \in N$, this is a contradiction, so $N \cup \{L_1, \dots, L_n\}$ is unsatisfiable. Consequently, $N \models \overline{L_1} \vee \dots \vee \overline{L_n}$. Now let $C = \overline{L_1} \vee \dots \vee \overline{L_{n-1}}$, $L' = \overline{L_n}$, $L = L_n$, and let M' be the list of all literals of M occurring before L_n, then the condition of "Backjump" is satisfied. □

Theorem 2.19 *Suppose that we reach a final state starting from $\varepsilon \parallel N$.*

(1) If the final state is $M \parallel N$, then N is satisfiable and M is a model of N.

(2) If the final state is fail, then N is unsatisfiable.

Proof. (1) Observe that the "Decide" rule is applicable as long as literals in N are undefined under M. Hence, in a final state, all literals must be defined. Furthermore, in a final state, no clause in N can be false under M, otherwise "Fail" or "Backjump" would be applicable. Hence M is a model of every clause in N.

(2) If we reach *fail*, then in the previous step we must have reached a state $M \parallel N$ such that some $C \in N$ is false under M and M contains no decision literals. By part (2) of Lemma 2.16, every literal in M follows from N. On the other hand, $C \in N$, so N must be unsatisfiable. □

Getting Better Backjump Clauses

Suppose that we have reached a state $M \parallel N$ such that some clause $C \in N$ (or entailed by N) is false under M.

Consequently, every literal of C is the complement of some literal in M.

(1) If every literal in C is the complement of a decision literal of M, then C is either a backjump clause or $\bot$.

(2) Otherwise, $C = C' \vee \overline{L}$, such that L is a deduced literal.

    For every deduced literal L, there is a (unit or backjump) clause $D \vee L$, such that $N \models D \vee L$ and D is false under M.

    Then $N \models D \vee C'$ and $D \vee C'$ is also false under M. ($D \vee C'$ is a *resolvent* of $C' \vee \overline{L}$ and $D \vee L$.)

As long as we are in case (2), we can repeat this transformation with the new clause $D \vee C'$.

This process must terminate:

Define an ordering $\succ$ on literals so that $\overline{L_2} \succ \overline{L_1}$ if L_2 occurs right of L_1 on the trail.

The trail is finite, so $\succ$ is well-founded, and therefore $\succ_{\text{mul}}$ is well-founded.

Then the multiset of literals in $C' \vee \overline{L}$ is larger than the multiset of literals in $D \vee C'$ with respect to $\succ_{\text{mul}}$.

So we must eventually reach case (1): We obtain a backjump clause or the empty clause.

In practice, it is not necessary to continue until case (1) is reached. Usually, one resolves the literals in the reverse order in which they were added to M and stops as soon as one obtains a clause in which all literals but one are complements of literals occurring in M before the last decision literal. (This is a backjump clause.)
$\Rightarrow$ 1UIP (first unique implication point) strategy.

Learning Clauses

Backjump clauses are good candidates for learning.

To model learning, the CDCL system is extended by the following two rules:

Learn:

$M \parallel N \ \Rightarrow_{\text{CDCL}}\ M \parallel N \cup \{C\}$

if $N \models C$.

Forget:

$M \parallel N \cup \{C\} \ \Rightarrow_{\text{CDCL}}\ M \parallel N$

if $N \models C$.

If we ensure that no clause is learned infinitely often, then termination is guaranteed.

The other properties of the basic CDCL system hold also for the extended system.

Restart

Runtimes of CDCL-style procedures depend extremely on the choice of branching variables.

If no solution is found within a certain time limit, it can be useful to *restart* from scratch with an adapted variable selection heuristics. Learned clauses, however, are kept.

In addition, it is useful to restart after a unit clause has been learned.

The restart rule is typically applied after a certain number of clauses have been learned or a unit is derived:

Restart:

$M \parallel N \ \Rightarrow_{\text{CDCL}}\ \varepsilon \parallel N$

If Restart is only applied finitely often, termination is guaranteed.