1.4 Normal Forms

We define *conjunctions* of formulas as follows:

$\bigwedge_{i=1}^{0} F_i = \top$.

$\bigwedge_{i=1}^{1} F_i = F_1$.

$\bigwedge_{i=1}^{n+1} F_i = \bigwedge_{i=1}^{n} F_i \wedge F_{n+1}$.

and analogously *disjunctions:*

$\bigvee_{i=1}^{0} F_i = \bot$.

$\bigvee_{i=1}^{1} F_i = F_1$.

$\bigvee_{i=1}^{n+1} F_i = \bigvee_{i=1}^{n} F_i \vee F_{n+1}$.

Literals and Clauses

A *literal* is either a propositional variable P or a negated propositional variable $\neg P$.

A *clause* is a (possibly empty) disjunction of literals.

CNF and DNF

A formula is in *conjunctive normal form (CNF, clause normal form)*, if it is a conjunction of disjunctions of literals (or in other words, a conjunction of clauses).

A formula is in *disjunctive normal form (DNF)*, if it is a disjunction of conjunctions of literals.

Warning: definitions in the literature differ:

are complementary literals permitted?
are duplicated literals permitted?
are empty disjunctions/conjunctions permitted?

Checking the validity of CNF formulas or the unsatisfiability of DNF formulas is easy:

A formula in CNF is valid, if and only if each of its disjunctions contains a pair of complementary literals P and $\neg P$.

Conversely, a formula in DNF is unsatisfiable, if and only if each of its conjunctions contains a pair of complementary literals P and $\neg P$.

On the other hand, checking the unsatisfiability of CNF formulas or the validity of DNF formulas is known to be coNP-complete.

Conversion to CNF/DNF

Proposition 1.7 *For every formula there is an equivalent formula in CNF (and also an equivalent formula in DNF).*

Proof. We consider the case of CNF.

Apply the following rules as long as possible (modulo associativity and commutativity of $\wedge$ and $\vee$):

Step 1: Eliminate equivalences:

$$(F \leftrightarrow G) \;\Rightarrow_K\; (F \rightarrow G) \wedge (G \rightarrow F)$$

Step 2: Eliminate implications:

$$(F \rightarrow G) \;\Rightarrow_K\; (\neg F \vee G)$$

Step 3: Push negations downward:

$$\begin{aligned} \neg(F \vee G) &\;\Rightarrow_K\; (\neg F \wedge \neg G) \\ \neg(F \wedge G) &\;\Rightarrow_K\; (\neg F \vee \neg G) \end{aligned}$$

Step 4: Eliminate multiple negations:

$$\neg\neg F \;\Rightarrow_K\; F$$

Step 5: Push disjunctions downward:

$$(F \wedge G) \vee H \;\Rightarrow_K\; (F \vee H) \wedge (G \vee H)$$

Step 6: Eliminate $\top$ and $\bot$:

$$\begin{aligned} (F \wedge \top) &\;\Rightarrow_K\; F \\ (F \wedge \bot) &\;\Rightarrow_K\; \bot \\ (F \vee \top) &\;\Rightarrow_K\; \top \\ (F \vee \bot) &\;\Rightarrow_K\; F \\ \neg\bot &\;\Rightarrow_K\; \top \\ \neg\top &\;\Rightarrow_K\; \bot \end{aligned}$$

Proving termination is easy for most of the steps; only step 3 and step 5 are a bit more complicated.

For step 3, we can prove termination in the following way: We define a function μ from formulas to positive integers such that $\mu(\bot) = \mu(\top) = \mu(P) = 1$, $\mu(\neg F) = 2\mu(F)$, $\mu(F \wedge G) = \mu(F \vee G) = \mu(F \rightarrow G) = \mu(F \leftrightarrow G) = \mu(F) + \mu(G) + 1$. Whenever a formula H' is the result of applying a rule of step 3 to a formula H, then $\mu(H) > \mu(H')$. Since μ takes only integer values and $\mu(H) \geq 1$ for all formulas H, step 3 must terminate.

Termination of step 5 is proved similarly using a function ν from formulas to positive integers such that $\nu(\bot) = \nu(\top) = \nu(P) = 1$, $\nu(\neg F) = \nu(F) + 1$, $\nu(F \wedge G) = \nu(F \rightarrow G) = \nu(F \leftrightarrow G) = \nu(F) + \nu(G) + 1$, and $\nu(F \vee G) = 2\nu(F)\nu(G)$. Again, if a formula H' is the result of applying a rule of step 5 to a formula H, then $\nu(H) > \nu(H')$. Since ν takes only integer values and Since $\nu(H) \geq 1$ for all formulas H, step 5 terminates, too.

The resulting formula is equivalent to the original one and in CNF.

The conversion of a formula to DNF works in the same way, except that conjunctions have to be pushed downward in step 5. $\square$

Complexity

Conversion to CNF (or DNF) may produce a formula whose size is *exponential* in the size of the original one.

Satisfiability-preserving Transformations

The goal

"find a formula G in CNF such that $F \models\!\!\!\!\!= G$"

is unpractical.

But if we relax the requirement to

"find a formula G in CNF such that $F \models \bot \Leftrightarrow G \models \bot$"

we can get an efficient transformation.

Idea: A formula $F[F']$ is satisfiable if and only if $F[P] \wedge (P \leftrightarrow F')$ is satisfiable (where P is a new propositional variable that works as an abbreviation for F').

We can use this rule recursively for all subformulas in the original formula (this introduces a linear number of new propositional variables).

Conversion of the resulting formula to CNF increases the size only by an additional factor (each formula $P \leftrightarrow F'$ gives rise to at most one application of the distributivity law).

Optimized Transformations

A further improvement is possible by taking the *polarity* of the subformula F into account.

Assume that F contains neither $\rightarrow$ nor $\leftrightarrow$. A subformula F' of F has *positive polarity* in F, if it occurs below an even number of negation signs; it has *negative polarity* in F, if it occurs below an odd number of negation signs.

Proposition 1.8 *Let $F[F']$ be a formula containing neither $\rightarrow$ nor $\leftrightarrow$; let P be a propositional variable not occurring in $F[F']$.*

If F' has positive polarity in F, then $F[F']$ is satisfiable if and only if $F[P] \wedge (P \rightarrow F')$ is satisfiable.

If F' has negative polarity in F, then $F[F']$ is satisfiable if and only if $F[P] \wedge (F' \rightarrow P)$ is satisfiable.

Proof. Exercise. □

1.5 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).

Assumption:
Clauses contain neither duplicated literals nor complementary literals.

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

Satisfiability of Clause Sets

$\mathcal{A} \models N$ 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 $L \in C$.

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 true (false) to P.

P is called a *pure literal.*

The Davis-Putnam-Logemann-Loveland Proc.

```
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 clause P) return DPLL(M ∪ {P}, N);
    elsif (N contains unit clause ¬P) return DPLL(M ∪ {¬P}, N);
    elsif (N contains pure literal P) return DPLL(M ∪ {P}, N);
    elsif (N contains pure literal ¬P) return DPLL(M ∪ {¬P}, N);
    else {
        let P be some undefined variable in N;
        if (DPLL(M ∪ {¬P}, N)) return true;
        else return DPLL(M ∪ {P}, N);
    }
}
```

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