The functions ν and $\bar{\nu}$ give us an overapproximation for the number of clauses generated by a formula that occurs positively/negatively.

G	$\nu(G)$	$\bar{\nu}(G)$
$P, \top, \bot$	1	1
$F_1 \wedge F_2$	$\nu(F_1) + \nu(F_2)$	$\bar{\nu}(F_1)\bar{\nu}(F_2)$
$F_1 \vee F_2$	$\nu(F_1)\nu(F_2)$	$\bar{\nu}(F_1) + \bar{\nu}(F_2)$
$\neg F_1$	$\bar{\nu}(F_1)$	$\nu(F_1)$
$F_1 \to F_2$	$\bar{\nu}(F_1)\nu(F_2)$	$\nu(F_1) + \bar{\nu}(F_2)$
$F_1 \leftrightarrow F_2$	$\nu(F_1)\bar{\nu}(F_2) + \bar{\nu}(F_1)\nu(F_2)$	$\nu(F_1)\nu(F_2) + \bar{\nu}(F_1)\bar{\nu}(F_2)$

A better CNF transformation:

Step 1: Exhaustively apply modulo commutativity of $\leftrightarrow$ and associativity/commutativity of $\wedge$, $\vee$:

$$
\begin{aligned}
H[(F \wedge \top)]_p &\Rightarrow_{\text{OCNF}} H[F]_p \\
H[(F \vee \bot)]_p &\Rightarrow_{\text{OCNF}} H[F]_p \\
H[(F \leftrightarrow \bot)]_p &\Rightarrow_{\text{OCNF}} H[\neg F]_p \\
H[(F \leftrightarrow \top)]_p &\Rightarrow_{\text{OCNF}} H[F]_p \\
H[(F \vee \top)]_p &\Rightarrow_{\text{OCNF}} H[\top]_p \\
H[(F \wedge \bot)]_p &\Rightarrow_{\text{OCNF}} H[\bot]_p
\end{aligned}
$$

$$
\begin{aligned}
H[(F \wedge F)]_p &\Rightarrow_{\text{OCNF}} H[F]_p \\
H[(F \vee F)]_p &\Rightarrow_{\text{OCNF}} H[F]_p \\
H[(F \wedge (F \vee G))]_p &\Rightarrow_{\text{OCNF}} H[F]_p \\
H[(F \vee (F \wedge G))]_p &\Rightarrow_{\text{OCNF}} H[F]_p \\
H[(F \wedge \neg F)]_p &\Rightarrow_{\text{OCNF}} H[\bot]_p \\
H[(F \vee \neg F)]_p &\Rightarrow_{\text{OCNF}} H[\top]_p \\
H[\neg\top]_p &\Rightarrow_{\text{OCNF}} H[\bot]_p \\
H[\neg\bot]_p &\Rightarrow_{\text{OCNF}} H[\top]_p
\end{aligned}
$$

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

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:

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

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

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

Step 3: Eliminate equivalences dependent on their polarity:

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

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($M \cup \{P\}$, N)) return true;
    else return DPLL($M \cup \{\neg P\}$, N);
  }
}

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