Boolean Logic. Resolution Method

Definition. *Propositional, or Boolean, formulae* are constructed from a countable set of variables $\mathrm{Var} = \{p, q, r, \ldots\}$ using one unary connective $\neg$ (negation) and the following binary connectives: $\vee$ (disjunction), $\wedge$ (conjunction), $\to$ (implication). Formally, the set Fm of Boolean formulae is the smallest set such that

- $\mathrm{Var} \subset \mathrm{Fm}$;
- if $A, B \in \mathrm{Fm}$, then $\neg A, (A \vee B), (A \wedge B), (A \to B) \in \mathrm{Fm}$.

In short, this definition can be expressed by the following *context-free grammar (Backus–Naur form):*

$$\mathrm{Fm} ::= \mathrm{Var} \mid \neg\mathrm{Fm} \mid (\mathrm{Fm} \vee \mathrm{Fm}) \mid (\mathrm{Fm} \wedge \mathrm{Fm}) \mid (\mathrm{Fm} \to \mathrm{Fm})$$

Any function $\alpha\colon \mathrm{Var} \to \{0, 1\}$ is called a *truth assignment.* A truth assignment α can be propagated to an assignment on formulae, $\bar{\alpha}\colon \mathrm{Fm} \to \{0, 1\}$, using the following *truth tables:*

$\bar{\alpha}(A)$	$\bar{\alpha}(\neg A)$
0	1
1	0

$\bar{\alpha}(A)$	$\bar{\alpha}(B)$	$\bar{\alpha}(A \vee B)$
0	0	0
0	1	1
1	0	1
1	1	1

$\bar{\alpha}(A)$	$\bar{\alpha}(B)$	$\bar{\alpha}(A \wedge B)$
0	0	0
0	1	0
1	0	0
1	1	1

$\bar{\alpha}(A)$	$\bar{\alpha}(B)$	$\bar{\alpha}(A \to B)$
0	0	1
0	1	1
1	0	0
1	1	1

Definition. An assignment α is a *satisfying* one form formula A, if $\bar{\alpha}(A) = 1$. A formula A is *satisfiable*, if there exists a satisfying assignment α for A. A formula A is a *tautology,* if any assignment satisfies A.

Definition. Equivalences: we write $A \equiv B$ to denote the fact that $A \to B$ and $B \to A$ are both tautologies. In other words, $A \equiv B$ if and only if $\alpha(A) = \alpha(B)$ for any α.

It is important to know the following equivalences:

- $A \vee B \equiv B \vee A$, $A \wedge B \equiv B \wedge A$ (commutativity);
- $(A \vee B) \vee C \equiv A \vee (B \vee C)$, $(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$ (associativity);
- $A \vee A \equiv A \equiv A \wedge A$ (idempotence);
- $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$, $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$ (distributivity);
- $\neg(A \vee B) \equiv \neg A \wedge \neg B$, $\neg(A \wedge B) \equiv \neg A \vee \neg B$ (de Morgan laws);
- $A \to B \equiv \neg A \vee B$, $\neg(A \to B) \equiv A \wedge \neg B$;
- $A \to (B \wedge C) \equiv (A \to B) \wedge (A \to C)$, $(A \vee B) \to C \equiv (A \to C) \wedge (B \to C)$;
- $A \to (B \to C) \equiv (A \wedge B) \to C$;
- $\neg\neg A \equiv A$ (double negation law).

The questions of being a tautology and satisfiability of Boolean formulae are dual: a formula A is satisfiable if and only if its negation, $\neg A$, is **not** a tautology. Throughout this course, we pay more attention to satisfiability, and consider only formulae in conjunctive normal form (CNF).

A CNF is a conjunction of *elementary disjunctions,* or *clauses,* that is, disjunctions of variables and their negations. Any formula can be transformed into an equivalent formula in CNF, using the equivalences listed above. For example, a CNF for $(p \to (q \to r)) \wedge (p \vee \neg(q \vee r))$ is $(\neg p \vee \neg q \vee r) \wedge (p \vee \neg q) \wedge (p \vee \neg r)$. This CNF can be satisfied, for example, by $\alpha(p) = \alpha(q) = \alpha(r) = 1$.

We consider a CNF as a finite set of clauses which should be satisfied simultaneously. Our calculus is very simple, including only one rule, called *resolution inference:*

$$\frac{A \vee p \quad B \vee \neg p}{A \vee B}$$

This rule allows adding clause $A \vee B$ to the CNF, provided clauses $A \vee p$ and $B \vee \neg p$ are already there. The empty clause $\varnothing$ plays a specific rôle: being an empty disjunction, it is intepreted as **false,** and since the CNF is the conjunction of its clauses, **an empty clause falsifies the whole formula.**

Note that if a clause happens to include both q and $\neg q$ for some variable q (say, one comes from A and the other comes from B), then the clause can be removed from the CNF by *tertium non datur:* $q \vee \neg q$ is tautologically true, and so is $C \vee q \vee \neg q$ for any C.

The resolution inference rule satisfies the following correctness condition:

Lemma 1. *If α is a satisfying assignment for $A \vee p$ and $B \vee \neg p$, then it is also a satisfying assignment for $A \vee B$.*

Proof. If $\alpha(p) = 1$, then $\bar{\alpha}(B)$ should be 1 (otherwise $\bar{\alpha}(B \vee \neg p) = 0$, which is not the case), and so is $\bar{\alpha}(A \vee B)$. If $\alpha(p) = 0$, then $\bar{\alpha}(A) = 1$, whence $\bar{\alpha}(A \vee B) = 1$. □

The empty clause is not satisfiable. Thus, if one can derive $\varnothing$ from a CNF using resolution inference, then the CNF is not satisfiable. In fact, it is a criterion:

Theorem 2. *A CNF is satisfiable if and only if one cannot derive $\varnothing$ from it using resolution inference.*

This is the *soundness and completeness theorem* for propositional resolution calculus.

Proof. The "only if" part (soundness) comes from Lemma 1 as discussed above.

For the "if" part (completeness), proceed by induction on the number of variables used in the CNF. The base case includes only one variable, p. The only non-trivial clauses here are p and $\neg p$. If the CNF includes both of them, then by resolution inference we derive the empty clause. Contradiction. Otherwise, the CNF gets satisfied by assigning $\alpha(p) = 0$ (if p is not in the CNF) or $\alpha(p) = 1$ (otherwise).

For the induction step, take a variable q. Denote the CNF by S and consider two other CNFs, S^- and S^+. The CNF S^+ includes all clauses of S which do not include $\neg q$, with q removed. Dually, for S^- we take all clauses which do not include q, with $\neg q$ removed. Suppose that S is not satisfiable. We claim that so are both S^+ and S^-. Indeed, if α satisfies S^+, then augmenting it by stipulating $\alpha(q) = 0$ satisfies S, and if β satisfies S^-, then taking $\beta(q) = 1$ satisfies S.

Both S^+ and S^- include less variables, then S, so we can apply the induction hypothesis. Since they are both not satisfiable, there exist resolution inference derivations which derive $\varnothing$ from S^+ and S^-. If one returns the removed occurrences of q to S^+ and the removed occurrences of $\neg q$ to S^-, this derivations would yield clause q or $\varnothing$ and $\neg q$ or $\varnothing$, respectively. Notice that these derivations are valid in the original CNF S. Now an application of resolution inference for q and $\neg q$ yields $\varnothing$. □

This theorem suggests the following **resolution algorithm** for checking satisfiability of a CNF:

1. *Saturation.* Apply resolution inference, until it stops generating new clauses.
2. *Checking.* The original CNF is satisfiable if and only if the resulting saturated CNF does not include the empty clause.

Let us estimate the complexity of resolution algorithm. For the general case, saturation can generate exponentially many clauses (see exercises), thus the algorithm will take exponential time to execute. By k-CNF we denote a CNF in which each clause includes at most k literals (variables / negations of variables). Even for a 3-CNF, however, saturation can lead to exponential growth. We consider the special case of 2-CNF. It is easy to see that applying resolution inference to a 2-CNF yields again a 2-CNF. Suppose the original CNF included n variables: $p_1, \ldots, p_n$. Then the maximal possible number of clauses can be counted as follows. There are $2n$ literals: $p_1, \ldots, p_n, \neg p_1, \ldots, \neg p_n$. Clauses of a 2-CNF can include the empty one (1 clause), $2n$ one-literal clauses, and $\frac{2n \cdot (2n-1)}{2}$ two-literal clauses, which is $O(n^2)$ in total.

The process of saturation can be organised in a way that each pair of clauses gets processed only once: first we process all pairs of original CNF clauses; after adding a new clause, we process it in pair with each old one. Processing a pair means that the algorithm tries to apply resolution to the pair. If it succeeds, it checks whether the clause obtained is a new one. If yes, the new clause gets added to the CNF. A naïve implementation of this requires comparing the new clause with $O(n^2)$ existing ones. The check whether resolution inference could be applied to a pair of clauses is implemented in $O(1)$ (recall that each clause includes 2 or less literals). The algorithm stops either if it reaches the empty clause, or when saturation stops (no new clauses get added). Thus, we get an algorithm of complexity $O(n^4)$, that is, **checking satisfiability for 2-CNF can be done in polynomial time.**