Chapter 4
Propositional Logic: Resolution

The method of resolution, invented by J.A. Robinson in 1965, is an efficient method for searching for a proof. In this section, we introduce resolution for the propositional logic, though its advantages will not become apparent until it is extended to first-order logic. It is important to become familiar with resolution, because it is widely used in automatic theorem provers and it is also the basis of logic programming (Chap. 11).

4.1 Conjunctive Normal Form

Definition 4.1 A formula is in *conjunctive normal form (CNF)* iff it is a conjunction of disjunctions of literals. ■

Example 4.2 The formula:

$$(\neg p \vee q \vee r) \;\wedge\; (\neg q \vee r) \;\wedge\; (\neg r)$$

is in CNF while the formula:

$$(\neg p \vee q \vee r) \;\wedge\; ((p \wedge \neg q) \vee r) \;\wedge\; (\neg r)$$

is not in CNF, because $(p \wedge \neg q) \vee r$ is not a disjunction.

The formula:

$$(\neg p \vee q \vee r) \;\wedge\; \neg\,(\neg q \vee r) \;\wedge\; (\neg r)$$

is not in CNF because the second disjunction is negated. ■

Theorem 4.3 *Every formula in propositional logic can be transformed into an equivalent formula in CNF.*

Proof To convert an arbitrary formula to a formula in CNF perform the following steps, each of which preserves logical equivalence:

1. Eliminate all operators except for negation, conjunction and disjunction by substituting logically equivalent formulas:

$$\begin{aligned} A \leftrightarrow B &\equiv (A \rightarrow B) \wedge (B \rightarrow A), \\ A \oplus B &\equiv \neg\,(A \rightarrow B) \vee \neg\,(B \rightarrow A), \\ A \rightarrow B &\equiv \neg A \vee B, \\ A \uparrow B &\equiv \neg (A \wedge B), \\ A \downarrow B &\equiv \neg (A \vee B). \end{aligned}$$

2. Push negations inward using De Morgan's laws:

$$\begin{aligned} \neg\,(A \wedge B) &\equiv (\neg A \vee \neg B), \\ \neg\,(A \vee B) &\equiv (\neg A \wedge \neg B), \end{aligned}$$

   until they appear only before atomic propositions or atomic propositions preceded by negations.
3. Eliminate sequences of negations by deleting double negation operators:

$$\neg\neg A \equiv A.$$

4. The formula now consists of disjunctions and conjunctions of literals. Use the distributive laws:

$$\begin{aligned} A \vee (B \wedge C) &\equiv (A \vee B) \wedge (A \vee C), \\ (A \wedge B) \vee C &\equiv (A \vee C) \wedge (B \vee C) \end{aligned}$$

   to eliminate conjunctions within disjunctions. ■

Example 4.4 The following sequence of formulas shows the four steps applied to the formula $(\neg p \rightarrow \neg q) \rightarrow (p \rightarrow q)$:

$$\begin{aligned} (\neg p \rightarrow \neg q) \rightarrow (p \rightarrow q) &\equiv \neg\,(\neg\neg p \vee \neg q) \vee (\neg p \vee q) \\ &\equiv (\neg\neg\neg p \wedge \neg\neg q) \vee (\neg p \vee q) \\ &\equiv (\neg p \wedge q) \vee (\neg p \vee q) \\ &\equiv (\neg p \vee \neg p \vee q) \wedge (q \vee \neg p \vee q). \end{aligned}$$

■

4.2 Clausal Form

The clausal form of formula is a notational variant of CNF. Recall (Definition 2.57) that a *literal* is an atom or the negation of an atom.

Definition 4.5

- A *clause* is a set of literals.
- A clause is considered to be an implicit disjunction of its literals.
- A *unit clause* is a clause consisting of exactly one literal.
- The empty set of literals is the *empty clause*, denoted by $\square$.
- A formula in *clausal form* is a set of clauses.
- A formula is considered to be an implicit conjunction of its clauses.
- The formula that is the *empty set of clauses* is denoted by $\emptyset$. ■

The only significant difference between clausal form and the standard syntax is that clausal form is defined in terms of sets, while our standard syntax was defined in terms of trees. A node in a tree may have multiple children that are identical subtrees, but a set has only one occurrence of each of its elements. However, this difference is of no logical significance.

Corollary 4.6 *Every formula ϕ in propositional logic can be transformed into an logically equivalent formula in clausal form.*

Proof By Theorem 4.3, ϕ can be transformed into a logically equivalent formula ϕ' in CNF. Transform each disjunction in ϕ' into a clause (a set of literals) and ϕ' itself into the set of these clauses. Clearly, the transformation into sets will cause multiple occurrences of literals and clauses to collapse into single occurrences. Logical equivalence is preserved by idempotence: $A \wedge A \equiv A$ and $A \vee A \equiv A$. ■

Example 4.7 The CNF formula:

$$(p \vee r) \wedge (\neg q \vee \neg p \vee q) \wedge (p \vee \neg p \vee q \vee p \vee \neg p) \wedge (r \vee p)$$

is logically equivalent to its clausal form:

$$\{\{p, r\},\ \{\neg q, \neg p, q\},\ \{p, \neg p, q\}\}.$$

The clauses corresponding to the first and last disjunctions collapse into a single set, while in the third disjunction multiple occurrences of p and $\neg p$ have been collapsed to obtain the third clause. ■

Trivial Clauses

A formula in clausal form can be simplified by removing trivial clauses.

Definition 4.8 A clause if *trivial* if it contains a pair of clashing literals. ■

Since a trivial clause is valid ($p \vee \neg\, p \equiv true$), it can be removed from a set of clauses without changing the truth value of the formula.

Lemma 4.9 *Let S be a set of clauses and let $C \in S$ be a trivial clause. Then $S - \{C\}$ is logically equivalent to S.*

Proof Since a clause is an implicit disjunction, C is logically equivalent to a formula obtained by weakening, commutativity and associativity of a valid disjunction $p \vee \neg\, p$ (Theorems 3.34–3.35). Let $\mathscr{I}$ be any interpretation for $S - \{C\}$. Since $S - \{C\}$ is an implicit conjunction, the value $v_{\mathscr{I}}(S - \{C\})$ is not changed by adding the clause C, since $v_{\mathscr{I}}(C) = T$ and $A \wedge T \equiv A$. Therefore, $v_{\mathscr{I}}(S - \{C\}) = v_{\mathscr{I}}(S)$. Since $\mathscr{I}$ was arbitrary, it follows that $S - \{C\} \equiv S$. ■

Henceforth, we will assume that all trivial clauses have been deleted from formulas in clausal form.

The Empty Clause and the Empty Set of Clauses

The following results may be a bit hard to understand at first, but they are very important. The proof uses reasoning about vacuous sets.

Lemma 4.10

$\square$, the empty clause, is unsatisfiable. $\emptyset$, the empty set of clauses, is valid.

Proof A clause is satisfiable iff there is *some* interpretation under which *at least one literal* in the clause is true. Let $\mathscr{I}$ be an arbitrary interpretation. Since there are no literals in $\square$, there are *no* literals whose value is true under $\mathscr{I}$. But $\mathscr{I}$ was an arbitrary interpretation, so $\square$ is unsatisfiable.

A set of clauses is valid iff *every* clause in the set is true in *every* interpretation. But there are no clauses in $\emptyset$ that need be true, so $\emptyset$ is valid. ■

Notation

When working with clausal form, the following additional notational conventions will be used:

- An abbreviated notation will be used for a formula in clausal form. The set delimiters { and } are removed from each clause and a negated literal is denoted by a bar over the atomic proposition. In this notation, the formula in Example 4.7 becomes:

$$\{pr,\ \bar{q}\bar{p}q,\ p\bar{p}q\}.$$

- S is a formula in clausal form, C is a clause and l is a literal. The symbols will be subscripted and primed as necessary.
- If l is a literal l^c is its complement: if $l = p$ then $l^c = \bar{p}$ and if $l = \bar{p}$ then $l^c = p$.

- The concept of an interpretation is generalized to literals. Let l be a literal defined on the atomic proposition p, that is, l is p or l is $\bar{p}$. Then an interpretation $\mathscr{I}$ for a set of atomic propositions including p is extended to l as follows:
  - $\mathscr{I}(l) = T$, if $l = p$ and $\mathscr{I}(p) = T$,
  - $\mathscr{I}(l) = F$, if $l = p$ and $\mathscr{I}(p) = F$,
  - $\mathscr{I}(l) = T$, if $l = \bar{p}$ and $\mathscr{I}(p) = F$,
  - $\mathscr{I}(l) = F$, if $l = \bar{p}$ and $\mathscr{I}(p) = T$.

The Restriction of CNF to 3CNF *

Definition 4.11 A formula is in *3CNF* iff it is in CNF and each disjunction has exactly three literals. ■

The problem of finding a model for a formula in CNF belongs to an important class of problems called $\mathscr{NP}$-complete problems (Sect. 6.7). This important theoretical result holds even if the formulas are restricted to 3CNF. To prove this, an efficient algorithm is needed to transform a CNF formula into one in 3CNF.

Algorithm 4.12 (CNF to 3CNF)
Input: A formula in CNF.
Output: A formula in 3CNF.

For each disjunction $C_i = l_i^1 \vee l_i^2 \vee \cdots \vee l_i^{n_i}$, perform the appropriate transformation depending of the value of n_i:

- If $n_i = 1$, create two new atoms p_i^1, p_i^2 and replace C_i by:

$$(l_i^1 \vee p_i^1 \vee p_i^2) \wedge (l_i^1 \vee \neg\, p_i^1 \vee p_i^2) \wedge (l_i^1 \vee p_i^1 \vee \neg\, p_i^2) \wedge (l_i^1 \vee \neg\, p_i^1 \vee \neg\, p_i^2).$$

- If $n_i = 2$, create one new atom p_i^1 and replace C_i by:

$$(l_i^1 \vee l_i^2 \vee p_i^1) \wedge (l_i^1 \vee l_i^2 \vee \neg\, p_i^1).$$

- If $n_i = 3$, do nothing.
- If $n_i > 3$, create $n - 3$ new atoms $p_i^1, p_i^2, \ldots, p_i^{n-3}$ and replace C_i by:

$$(l_i^1 \vee l_i^2 \vee p_i^1) \wedge (\neg\, p_i^1 \vee l_i^3 \vee p_i^2) \wedge \cdots \wedge (\neg\, p_i^{n-3} \vee l_i^{n-1} \vee l_i^n).$$

■

We leave the proof of the following theorem as an exercise.

Theorem 4.13 *Let A be a formula in CNF and let A' be the formula in 3CNF constructed from A by Algorithm 4.12. Then A is satisfiable if and only if A' is satisfiable. The length of A' (the number of symbols in A') is a polynomial in the length of A.*

4.3 Resolution Rule

Resolution is a refutation procedure used to check if a formula in clausal form is unsatisfiable. The resolution procedure consists of a sequence of applications of the resolution rule to a set of clauses. The rule maintains satisfiability: if a set of clauses is satisfiable, so is the set of clauses produced by an application of the rule. Therefore, if the (unsatisfiable) empty clause is ever obtained, the original set of clauses must have been unsatisfiable.

Rule 4.14 (Resolution rule) *Let C_1, C_2 be clauses such that $l \in C_1$, $l^c \in C_2$. The clauses C_1, C_2 are said to be* clashing clauses *and to* clash *on the complementary pair of literals l, l^c. C, the* resolvent *of C_1 and C_2, is the clause*:

$$Res(C_1, C_2) = (C_1 - \{l\}) \cup (C_2 - \{l^c\}).$$

C_1 and C_2 are the parent clauses *of C.* ■

Example 4.15 The pair of clauses $C_1 = ab\bar{c}$ and $C_2 = bc\bar{e}$ clash on the pair of complementary literals c, $\bar{c}$. The resolvent is:

$$C = (ab\bar{c} - \{\bar{c}\}) \cup (bc\bar{e} - \{c\}) = ab \cup b\bar{e} = ab\bar{e}.$$

Recall that a clause is a set so duplicate literals are removed when taking the union: $\{a, b\} \cup \{b, \bar{e}\} = \{a, b, \bar{e}\}$. ■

Resolution is only performed if the pair of clauses clash on *exactly* one pair of complementary literals.

Lemma 4.16 *If two clauses clash on more than one literal, their resolvent is a trivial clause (Definition 4.8).*

Proof Consider a pair of clauses:

$$\{l_1, l_2\} \cup C_1, \qquad \{l_1^c, l_2^c\} \cup C_2,$$

and suppose that we perform the resolution rule because the clauses clash on the pair of literals $\{l_1, l_1^c\}$. The resolvent is the trivial clause:

$$\{l_2, l_2^c\} \cup C_1 \cup C_2.$$

■

It is not strictly incorrect to perform resolution on such clauses, but since trivial clauses contribute nothing to the satisfiability or unsatisfiability of a set of clauses (Theorem 4.9), we agree to delete them from any set of clauses and not to perform resolution on clauses with two clashing pairs of literals.

Theorem 4.17 *The resolvent C is satisfiable if and only if the parent clauses C_1 and C_2 are both satisfiable.*

Proof Let C_1 and C_2 be satisfiable under an interpretation $\mathscr{I}$. Since l, l^c are complementary, either $\mathscr{I}(l) = T$ or $\mathscr{I}(l^c) = T$. Suppose that $\mathscr{I}(l) = T$; then $\mathscr{I}(l^c) = F$ and C_2, the clause containing l^c, can be satisfied only if $\mathscr{I}(l') = T$ for some *other* literal $l' \in C_2, l' \neq l^c$. By construction in the resolution rule, $l' \in C$, so $\mathscr{I}$ is also a model for C. A symmetric argument holds if $\mathscr{I}(l^c) = T$.

Conversely, let $\mathscr{I}$ be an interpretation which satisfies C; then $\mathscr{I}(l') = T$ for at least one literal $l' \in C$. By the resolution rule, $l' \in C_1$ or $l' \in C_2$ (or both). If $l' \in C_1$, then $v_{\mathscr{I}}(C_1) = T$. Since neither $l \in C$ nor $l^c \in C$, $\mathscr{I}$ is not defined on either l or l^c, and we can extend $\mathscr{I}$ to an interpretation $\mathscr{I}'$ by defining $\mathscr{I}(l^c) = T$. Since $l^c \in C_2$, $v_{\mathscr{I}'}(C_2) = T$ and $v_{\mathscr{I}'}(C_1) = v_{\mathscr{I}}(C_1) = T$ (because $\mathscr{I}$ is an extension of v) so $\mathscr{I}'$ is a model for both C_1 and C_2. A symmetric argument holds if $l' \in C_2$. ∎

Algorithm 4.18 (Resolution procedure)
Input: A set of clauses S.
Output: S is satisfiable or unsatisfiable.

Let S be a set of clauses and define $S_0 = S$.

Repeat the following steps to obtain S_{i+1} from S_i until the procedure terminates as defined below:

- *Choose* a pair of clashing clauses $\{C_1, C_2\} \subseteq S_i$ that has not been chosen before.
- Compute $C = Res(C_1, C_2)$ according to the resolution rule.
- If C is not a trivial clause, let $S_{i+1} = S_i \cup \{C\}$; otherwise, $S_{i+1} = S_i$.

Terminate the procedure if:

- $C = \square$.
- All pairs of clashing clauses have be resolved. ∎

Example 4.19 Consider the set of clauses:

$$S = \{(1)\, p,\ (2)\, \bar{p}q,\ (3)\, \bar{r},\ (4)\, \bar{p}\bar{q}r\},$$

where the clauses have been numbered. Here is a resolution derivation of $\square$ from S, where the justification for each line is the pair of the numbers of the parent clauses that have been resolved to give the resolvent clause:

5.	$\bar{p}\bar{q}$	$3, 4$
6.	$\bar{p}$	$5, 2$
7.	$\square$	$6, 1$

∎

It is easier to read a resolution derivation if it is presented as a tree. Figure 4.1 shows the tree that represents the derivation of Example 4.19. The clauses of S label leaves, and the resolvents label interior nodes whose children are the parent clauses used in the resolution.

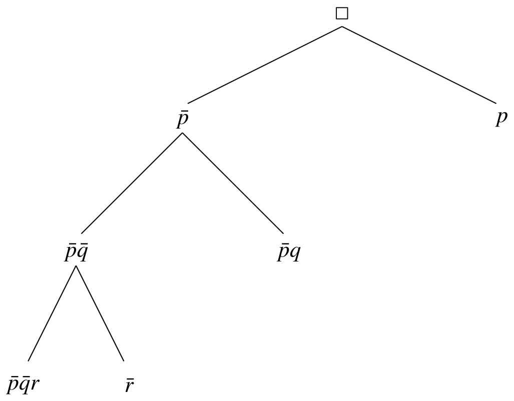

Fig. 4.1 A resolution refutation represented as a tree

Definition 4.20 A derivation of $\square$ from a set of clauses S is a *refutation by resolution* of S or a *resolution refutation* of S. ■

Since $\square$ is unsatisfiable, by Theorem 4.17 if there exists a refutation of S by resolution then S is unsatisfiable.

In Example 4.19, we derived the unsatisfiable clause $\square$, so we conclude that the set of clauses S is unsatisfiable. We leave it to the reader to check that S is the clausal form of $\neg A$ where A is an instance of Axiom 2 of $\mathscr{H}$ $(p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r))$. Since $\neg A$ is unsatisfiable, A is valid.

4.4 Soundness and Completeness of Resolution *

The soundness of resolution follows easily from Theorem 4.17, but completeness is rather difficult to prove, so you may want to skip the this section on your first reading.

Theorem 4.21 *If the set of clauses labeling the leaves of a resolution tree is satisfiable then the clause at the root is satisfiable.*

The proof is by induction using Theorem 4.17 and is left as an exercise.

The converse to Theorem 4.21 is not true because we have no way of ensuring that the extensions made to $\mathscr{I}$ on all branches are consistent. In the tree in Fig. 4.2, the set of clauses on the leaves $S = \{r, pq\bar{r}, \bar{r}, p\bar{q}r\}$ is not satisfiable even though the clause p at the root is satisfiable. Since S is unsatisfiable, it has a refutation: whenever the pair of clashing clauses r and $\bar{r}$ is chosen, the resolvent will be $\square$.

Resolution is a refutation procedure, so soundness and completeness are better expressed in terms of unsatisfiability, rather than validity.