Chapter 4
Propositional Logic: Resolution

- The method of resolution dates at least to a 1938 dissertation of Archie Blake on Boolean algebra, and was famously employed by Martin Davis and Hilary Putnam in 1960 in a method of automatic theorem-proving. In 1965, John Alan Robinson[1] showed how to apply resolution efficiently to predicate logic.
- An efficient method for searching for proofs, especially when extended to first-order logic.[2]
- Widely used in automatic theorem provers and in logic programming.
- Based on a representation of formulas in 'conjunctive normal form' . . .

4.1 Conjunctive normal form

A 'normal form' is a particular syntactical form into which any formula can be transformed while preserving logical equivalence.

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

Example 4.2

- $(\neg p \vee q \vee r) \wedge (\neg q \vee r) \wedge (\neg r)$ – in conjunctive normal form;
- $(\neg p \vee q \vee r) \wedge ((p \wedge \neg q) \vee r) \wedge (\neg r)$ – not in CNF: $p \wedge \neg q$ is a conjunction within a disjunction;
- $(\neg p \vee q \vee r) \wedge \neg(\neg q \vee r) \wedge (\neg r)$ – not in CNF because second conjunct is a negated disjunction.

□

[1]Not to be confused with Julia Robinson, who is also famous for work on logical decision procedures, much of which was carried out in collaboration with Davis, Putnam, and others.

[2]In a precise technical sense concerning worst-case complexity, resolution is more efficient than the tableau method.

Theorem 4.3 Every formula in propositional logic can be transformed into a logically equivalent formula in CNF.

Proof: Perform the following steps, in sequence, each of which preserves logical equivalence:

1. Eliminate all operators other than negation, conjunction and disjunction:

$$\begin{aligned} A \Leftrightarrow B &\equiv \quad (A \Rightarrow B) \wedge (B \Rightarrow A) \\ A \oplus B &\equiv \neg(A \Rightarrow B) \vee \neg(B \Rightarrow A) \\ A \Rightarrow B &\equiv \neg A \vee B \\ A \uparrow B &\equiv \neg(A \wedge B) \\ A \downarrow B &\equiv \neg(A \vee B) \end{aligned}$$

2. Drive negations inward via De Morgan's laws:

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

   Repeat as long as possible, at which point they will appear only before atoms or atoms preceded by negations.

3. Eliminate sequences of negations:

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

   The only negation symbols will then appear in literals.

4. Use the distributive laws to eliminate conjunctions within disjunctions:

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

$\square$

Example 4.4

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

$\square$

4.2 Clausal form

- a notational variant of conjunctive normal form.

Definition 4.5

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

$\blacksquare$

Corollary 4.6 Every propositional formula can be transformed into a logically equivalent formula in clausal form.

Proof: By Theorem 4.3, every $\phi \in \mathscr{F}$ can be transformed in to a logically equivalent formula ϕ' in CNF. Transform each disjunction in ϕ' into a set of literals (a clause) and the formula as a whole into the set of these clauses.

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 $A \vee A \equiv A$. $\square$

Example 4.4 revisited:

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

□

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 first and last disjunctions collapse into a single clause, and in the third disjunction, multiple occurrences of p and $\neg p$ collapse into single literals within the third clause. □

Trivial clauses

In the context of clausal form we'll say that complementary pairs of literals *clash*.

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

Since a trivial clause is valid, its removal from a set of clauses preserves logical equivalence.

Lemma 4.9 Let S be a set of clauses, and let $C \in S$ be a trivial clause. Then $S \setminus \{C\} \equiv S$.

Proof: Under any interpretation, C has the truth value T, so its removal does not affect the truth value of the implicit conjunction S. □

From now on, assume that all trivial clauses have been deleted from formulas in clausal form.

The empty clause and the empty set of clauses

Convention: The empty clause, $\Box$ is unsatisfiable, while the empty set of clauses, $\emptyset$ is valid.

Rationale: Think of a disjunction as being satisfied when *there exists* a disjunct that is satisfied; and a conjunction as satisfied whenever *all* conjuncts are satisfied.

Then, the empty clause can never be satisfied, because it contains no disjuncts; while the empty set of clauses is vacuously satisfied.

This convention agrees with standard mathematical conventions. $\Box$

More intuitively, disjunctions become harder to satisfy as disjuncts are removed, while conjunctions become easier to satisfy as conjuncts are removed.

Notation

- A clause $\{p, \neg q, r\}$ will be abbreviated $p\overline{q}r$, by eliminating the curly braces and commas, and letting $\overline{q}$ stand for $\neg q$.
- In this notation, the formula of example 4.7 becomes $\{pr, \overline{q}\overline{p}q, p\overline{p}q\}$ (where two of the clauses are trivial).
- If l is a literal, then l^c is its complement: if $l = p$, then $l^c = \overline{p}$ and if $l = \overline{p}$ then $l^c = p$.
- The concept of an interpretation is generalized to literals. Let l be a literal containing the atomic proposition p (that is, either $l = p$ or $l = \overline{p}$). Then
  - $\mathscr{I}(l) = T$, if $l = p$ and $\mathscr{I}(p) = T$,
  - $\mathscr{I}(l) = F$, if $l = p$ and $\mathscr{I}(p) = F$,
  - $\mathscr{I}(l) = F$, if $l = \overline{p}$ and $\mathscr{I}(p) = T$,
  - $\mathscr{I}(l) = T$, if $l = \overline{p}$ and $\mathscr{I}(p) = F$.

The restriction of CNF to 3CNF

– skipped, for brevity.

4.3 Resolution rule

- When used for checking validity, resolution is a a “refutation” procedure, used to check whether the formula’s negation (in clausal form) is unsatisfiable.
- It consists of a sequence of applications of the *resolution rule* to a set of clauses.
- Each application preserves satisfiability of the set of clauses: if the original set is satisfiable, so is that produced by an application of the resolution rule.
- The empty clause is eventually obtained if and only if the original set of clauses is unsatisfiable.

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

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

The original clauses C_1 and C_2 are called the *parent clauses* of C.

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

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

□

Resolution is only performed if the pair of literals clash on *exactly* one pair of complementary literals: otherwise, their resolvent is a trivial clause, and can be ignored.

Lemma 4.16 If two clauses clash on more than one literal, then their resolvent is a trivial clause.

Proof: Consider a pair of clauses $\{l_1, l_2\} \cup C_1$ and $\{l_1^c, l_2^c\} \cup C_2$, and resolve them with respect to the pair $\{l_1, l_1^c\}$. The resolvent is the trivial clause $\{l_2, l_2^c\} \cup C_1 \cup C_2$ □

Because trivial clauses can be deleted from a set of clauses without affecting its truth value, we do not perform resolution on clauses that clash on more than one pair of literals.

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

Proof: Suppose that C_1 and C_2 clash on the pair of literals l and l^c, and are satisfied by some interpretation $\mathscr{I}$, where, say, $\mathscr{I}(l) = T$. Then, if l belongs to C_1 and l^c to C_2, we must have $\mathscr{I}(l')$, for some other literal $l' \in C_2$. But then $l' \in C$, so C is satisfied.

Conversely, suppose that C is satisfied by some interpretation $\mathscr{I}$ for C. Then there must be some literal $l' \in C_1 \cup C_2$ other than l or l^c that is satisfied by $\mathscr{I}$. If, say, $l' \in C_1$, then extend $\mathscr{I}$ to an interpretation for $\{C_1, C_2\}$ by setting either $\mathscr{I}(l) = T$ if $l \in C_2$ or $\mathscr{I}(l^c) = T$ otherwise. The extended interpretation then satisfies both C_1 and C_2. $\square$

Algorithm 4.18 (Resolution procedure)
Input: A set of clauses S
Output: S is declared either satisfiable or unsatisfiable.
Initialize $S_0 = S$.
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 $\{C_1, C_2\}$ 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, let $S_{i+1} = S_i$.
- Terminate the procedure if
  - $C = \square$, or
  - All pairs of clashing clauses have been resolved.

Example 4.19 Let S be the set of clauses in the following list:

1. p
2. $\overline{p}q$
3. $\overline{r}$
4. $\overline{p}\overline{q}r$

Here is a resolution derivation of $\square$ from S, where the numbers of the resolved clauses are listed for each resolvent:

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

$\square$

The derivation is represented as a tree in Figure 4.1. The clauses of S label the leaves of the tree, and the resolvents label interior nodes whose children are the parent clauses of the resolution.

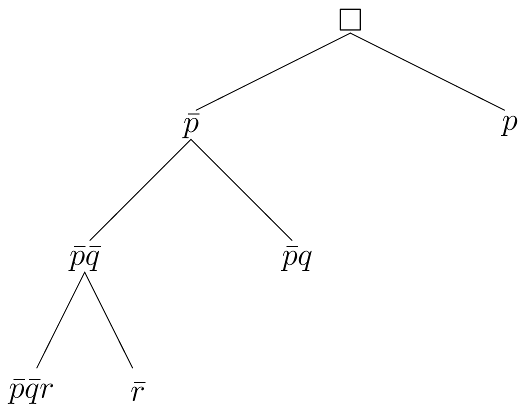

Figure 4.1: Resolution refutation as a tree

The same clause may label many nodes of the tree, because in general a given clause can be resolved with many other clauses.

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

- Because $\Box$ is unsatisfiable, it follows by Theorem 4.17 that if there exists a resolution refutation of S then S is unsatisfiable.
- The set of clauses S of Example 4.19 is the clausal form of the negation of an instance of Axiom scheme 2 of $\mathscr{H}$. Because its negation is unsatisfiable, the axiom is valid.

4.4 Soundness and Completeness of Resolution

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

Proof: By induction using Theorem 4.17 (exercise). $\Box$

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

Corollary 4.22 (Soundness) Let S be a set of clauses. If there is a refutation by resolution for S then S is unsatisfiable.

Proof: Immediate from Theorem 4.21. $\Box$

Theorem 4.23 (Completeness) If a set of clauses is unsatisfiable then the empty clause will be derived by the resolution procedure.

Proof sketch: The proof is based on *semantic trees*. Suppose a set S of clauses is given, and let $p_1,\ p_2,\ \ldots,\ p_n$ be a list of the atomic propositions appearing in S. A semantic tree is a complete binary tree such that, for every $0 < i \leq n$, every left-branching edge from a node at depth $i-1$ is labelled p_i, and every right-branching edge is labelled $\overline{p}_i$.

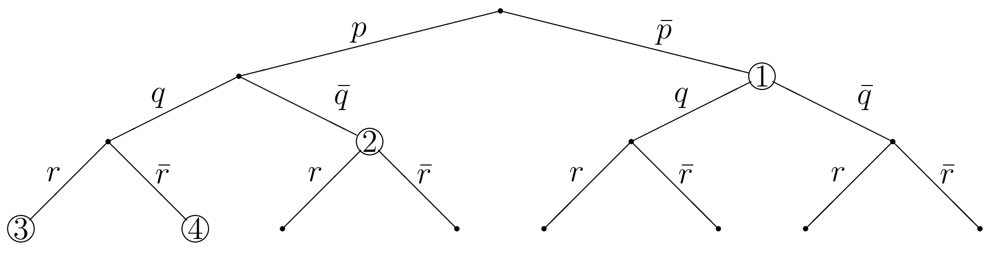

Figure 4.3: Semantic tree

- Every branch b determines an interpretation $\mathscr{I}_b$ for S that assigns T to all the literals labelling its edges.
- Conversely, every interpretation determines a branch.
- The branch b is *closed* if the corresponding truth value $v_b(S)$ is F; otherwise, b is *open*.
- The tree $\mathscr{T}$ is *closed* if all branches are closed; otherwise, it is *open*.
- The semantic tree $\mathscr{T}$ for a set of clauses S is closed if and only if the set S is unsatisfiable.
- Partial branches that start at the root but end above a leaf define partial interpretations.
- If $\mathscr{T}$ is closed, then, along a given branch, the shortest such partial branch that falsifies S ends in a *failure node*.
- A clause falsified by that partial branch is *associated* with the failure node.
- Figure 4.3 shows the numbers of the clauses from Example 4.19 that are associated with the failure nodes of the set of clauses S of that example.
- A clause C associated with a failure node is a subset of the complements of the literals labelling the partial branch from the root to the failure node.
- If $\mathscr{T}$ is closed, then the failure nodes form a 'frontier' that 'cuts across the tree from side to side' (see Fig. 4.3):
  - Because resolution only adds clauses to S, this frontier can only move higher.
  - As long as the frontier hasn't reached the root, there must be failure nodes that are siblings.

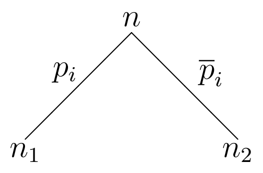

Figure 4.4: Sibling failure nodes

- Clauses associated with sibling failure nodes at depth i clash on the single pair of literals p_i and $\overline{p}_i$ ...
- and their resolvent is falsified by the partial branch ending at the parent (because it's a subset of the complements of the literals labelling that partial branch).
- So after a finite number of resolution steps, the frontier must reach the root, indicating that the empty clause has been generated.

□

Example 4.28 Again, Figure 4.3 shows the failure nodes of the set S and the numbers of the associated clauses for Example 4.19.

For instance, clause 3, $\overline{r}$, is associated with the failure node numbered 3, because $\{\overline{r}\}$ is a subset of $\{\overline{p}, \overline{q}, \overline{r}\}$, the set of complements of literals labelling the branch.

Clause 5, the resolvent $\overline{pq}$ of clauses 3 and 4 ($\overline{pqr}$), is associated with the parent of nodes 3 and 4, which becomes a failure node after clauses 3 and 4 are resolved.

The parent of that node and of node 2 becomes a failure node after resolution of clauses 2 and 5; and finally, the root node becomes a failure node after resolution of clauses 1 and 6. In both cases, the resolvent is associated with the failure node. □

4.5 Hard examples for resolution

We'll skip the details of this section, but discuss some of its significance. (Students won't be tested on this discussion.)

Resolution is so efficient that it was difficult to find problems that were ‘hard’ for it to solve.[3] The importance of the issue was both practical and theoretical.

The problem (SAT) of deciding satisfiability of a propositional formula is NP-complete: it’s in the class NP (nondeterministic polynomial-time) because it can be solved in the affirmative by a nondeterministic algorithm (guess an interpretation and check whether the formula is satisfied – if so, answer “yes”) in a number of steps that is upper-bounded by a polynomial in any reasonable measure of the size of the formula; it’s NP-*complete*, meaning ‘as hard as any problem in NP,’ because any decision problem in NP can be reduced to that of satisfiability by a deterministic, polynomial-time algorithm. Hence, if there were to exist a deterministic polynomial-time algorithm for satisfiability, there would exist one for any problem in NP (which would imply that P = NP, which would be extremely surprising and disruptive).

The problem of deciding unsatisfiability is that of deciding whether a given formula lies within the set-theoretic *complement* of the satisfiable formulas. It is therefore the *complement* of a decision problem (namely, SAT) that belongs to NP; it is said to belong to the class co-NP, and it is co-NP-complete. If a nondeterministic algorithm for checking unsatisfiability – such as resolution – always ran in polynomial time, that would mean that co-NP $\subseteq$ NP. But that would imply that the complements of the problems in co-NP were contained in the set of complements of problems in NP: that is, NP $\subseteq$ co-NP, so we would in fact have NP = co-NP.

That, too, would be a surprising result. Yet, by the early 1980s, no one had shown that a resolution refutation ever took longer than polynomial time.[4] This distinguished resolution from earlier attempts at automatic theorem-proving, which were clearly exponential in complexity.

A superpolynomial lower bound was first achieved by Armin Haken in his Ph.D. thesis of 1984 (University of Illinois at Urbana-Champaign). Haken essentially wrote down formulas that asserted that $n + 1$ ‘pigeons’ could fit individually into n ‘pigeonholes’ – which were unsatisfiable, by the pigeonhole principle – and showed that, while the encodings of the formulas as sets of clauses grew in length like n^3, the lengths of their shortest resolution refutations grew exponentially with n.

[3]In a precise technical sense, any tableau refutation can be translated into a resolution refutation in polynomial time; the reverse is not possible.

[4]except in a restricted form called “regular resolution.” Though he doesn’t cite himself, the author of the textbook contributed to this research.

Alasdair Urquhart of the University of Toronto strengthened Haken's result in 1987, using sequences of formulas based on 'expander-graph' constructions like those described in the textbook. Efficient encodings of the formulas as sets of clauses $S(n)$ grow linearly with n, but Urquhart obtained a lower bound on the length of the length of their resolution refutations that is exponential in n. Moreover, he showed that, in a standard axiomatic system for propositional logic, the same sets of clauses had refutations of polynomial length: the reason the examples were 'hard' for resolution is that resolution deals relatively inefficiently with formulas involving $\iff$ and $\oplus$.

4.6 Chapter summary

- Resolution is a (generally) efficient decision procedure for unsatisfiability in propositional logic ...
- or a refutation proof procedure.
- It is based on a clausal form, which is a set representation of conjunctive normal form.
- At each step, it produces a new clause, the *resolvent* of two clauses that clash on a pair of literals.
- It eventually yields the empty clause if and only if the original set of clauses is unsatisfiable.