CE 474 - LOGIC OF COMPUTER SCIENCE

Lecture Note 7 — Resolution & SAT Solving

Quick overview / motivation

SAT solving — deciding the satisfiability of propositional formulas — is a cornerstone of modern automated reasoning. SAT solvers power hardware verification, software model checking, planning in AI, constraint solving (scheduling), and even puzzle solving (Sudoku). In practice, we feed solvers a standard form (CNF); algorithms like **resolution** and **DPLL** (and DPLL-based solvers) then search for satisfying assignments or show unsatisfiability.

1. Learning objectives

By the end of this handout you should be able to:

- Convert arbitrary propositional formulas into **Conjunctive Normal Form (CNF)**.
- Apply the **resolution** inference rule to (attempt to) derive contradictions.
- Understand the high-level operation of the **DPLL** procedure.
- Sketch how modern SAT solvers extend DPLL with learning and heuristics.

Warm-up

What is the CNF of $(p \to q) \land (\neg q \to r)$? (Do it mentally: replace $\to$ with $\neg$ or $\lor$ and simplify.)

2. Conjunctive Normal Form (CNF)

Definition. A formula is in CNF if it is a conjunction of clauses, each clause being a disjunction of literals (a literal is an atomic proposition or its negation). Example:

$$(\neg p \lor q \lor r) \ \land \ (q \lor \neg p)$$

Conversion recipe (practical steps)

To convert an arbitrary formula to CNF:

1. Eliminate $\leftrightarrow$ and $\to$:

$$A \leftrightarrow B \Rightarrow (A \to B) \land (B \to A), \quad A \to B \Rightarrow \neg A \lor B$$

2. Push negations inward using De Morgan and double-negation:

$$\neg(A \land B) \equiv \neg A \lor \neg B, \quad \neg(\neg A) \equiv A$$

3. (Optional for large formulas) Apply Tseitin renaming: introduce fresh variables for subformulas to keep size linear.
4. Distribute $\vee$ over $\wedge$ to get a conjunction of disjunctions:

$$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$$

Small inline diagram: CNF pipeline

Worked example (short)

Convert $(p \to (q \vee r)) \wedge (\neg q \to \neg p)$ to CNF.

$$p \to (q \vee r) \equiv \neg p \vee q \vee r$$
$$\neg q \to \neg p \equiv q \vee \neg p$$

CNF: $(\neg p \vee q \vee r) \wedge (q \vee \neg p)$. (Already in clause form.)

Checkpoint

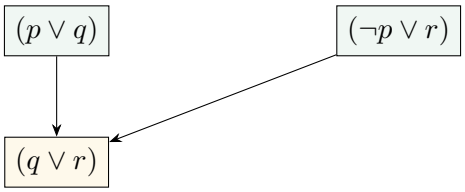
Convert $(p \leftrightarrow q) \to r$ to CNF. Show the intermediate forms.

3. Resolution: single inference rule for CNF

Resolution rule. From clauses $(C \vee p)$ and $(D \vee \neg p)$ infer the *resolvent* $(C \vee D)$. Symbolically:

$$\frac{C \vee p \qquad D \vee \neg p}{C \vee D} \text{ (resolve on } p\text{)}$$

Visual: resolution step



resolve on p

Use: Repeated resolution can derive new clauses. A CNF formula is **unsatisfiable** iff resolution can derive the empty clause $\square$.

Practice

Resolve $(p \vee r)$ and $(\neg p \vee q)$ on p. What is the resolvent?

4. DPLL: search + propagation (high level)

DPLL (Davis–Putnam–Logemann–Loveland) is the backbone of many SAT solvers. It interleaves search (decisions) with *unit propagation* and optional heuristics.

Main ingredients

Unit propagation
If a clause has exactly one unassigned literal ℓ, assign ℓ so the clause is satisfied. Repeat until no unit clauses.

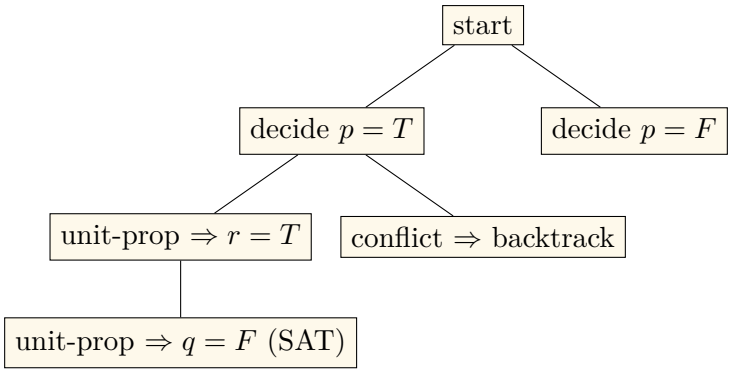
Pure literal elimination
If a variable occurs only as positive (or only negative), set it to make its clauses true (optional).

Decision
Pick an unassigned variable and assign a value (branch).

Backtracking
If a contradiction (empty clause) appears, undo the last decisions and try the other polarity (or backjump if using learning).

Termination
If all clauses satisfied → SAT. If all choices exhausted → UNSAT.

Small DPLL search sketch (inline diagram)



Example DPLL run (brief)

Formula: $(p \vee q) \wedge (\neg p \vee r) \wedge (\neg q \vee \neg r)$.

1. No unit clauses initially.
2. Decide $p = T$.
3. Clause $(\neg p \vee r)$ becomes unit (r) ⇒ set $r = T$.
4. Clause $(\neg q \vee \neg r)$ becomes unit $(\neg q)$ ⇒ set $q = F$.
5. All clauses evaluate to true under $p = T, r = T, q = F$ ⇒ SAT. A satisfying model is $\{p = T, q = F, r = T\}$.

5. Worked resolution example (manual)

Check satisfiability of

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

- Resolve $(p \vee q)$ and $(\neg p \vee r)$ on $p \to (q \vee r)$.
- Resolve $(q \vee r)$ with $(\neg q \vee \neg r)$ on $q \to (r \vee \neg r)$ (tautology).
- Tautologies do not help produce the empty clause. Try alternative resolutions:
  - Resolve $(p \vee q)$ with $(\neg q \vee \neg r)$ on $q \to (p \vee \neg r)$.
  - Resolve $(p \vee \neg r)$ with $(\neg p \vee r)$ on $p \to (\neg r \vee r)$ (tautology).
- No empty clause derived formula is satisfiable (consistent with DPLL run above).

6. Practical notes and modern solvers

Modern SAT solvers (CDCL solvers) extend DPLL with:

- **Conflict-driven clause learning (CDCL)**: when a conflict occurs, analyze it and add a learned clause to prevent the same mistake.
- **Non-chronological backtracking (backjumping)**: jump to the decision level that caused the conflict.
- **Heuristics**: variable selection heuristics (e.g., VSIDS) to decide branching order.
- **Restarts** and other practical tuning techniques.

7. Check Your Understanding

Exercises

1. Convert $(p \leftrightarrow (q \wedge \neg r)) \to s$ into CNF (show steps). Optionally use Tseitin renaming for readability.
2. For CNF $(p \vee q \vee r) \wedge (\neg p \vee q) \wedge (\neg q \vee r)$, simulate a DPLL run (decisions, unit propagations).
3. Use resolution to derive a resolvent from $(p \vee a \vee b)$ and $(\neg p \vee \neg a)$. What happens if you continue resolving?

8. Summary and takeaways

- **CNF** is the standard input for SAT and resolution.
- **Resolution** is simple and complete for propositional logic (can derive contradiction if unsatisfiable), but may blow up combinatorially.
- **DPLL** is practical: interleave search with propagation to find models quickly in practice.
- **Modern solvers** (CDCL) add learning and heuristics that make SAT fast on real-world benchmarks.

Further reading

- Huth, M., & Ryan, M. (2004). *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press.
- Biere, A., Heule, M., van Maaren, H., & Walsh, T. (eds.). (2009). *Handbook of Satisfiability*. IOS Press.
- Nieuwenhuis, R., Oliveras, A., & Tinelli, C. (2006). "Solving SAT and SMT by Lazy Grounding." Proceedings of IJCAR.