

Logic

Aayush Bajaj

July 10, 2026

Contents

1	propositional logic	1
1.1	connectives	1
1.2	tautology, satisfiability, equivalence	2
1.3	normal forms	2
1.4	functional completeness	2
2	rules of inference	3
3	first-order logic	3
3.1	models and satisfaction	3
4	soundness and completeness	4
5	proof techniques	4
5.1	the well-ordering principle	5
6	SAT and resolution	5
6.1	the problem	5
6.2	resolution	5
6.3	a DPLL-lite solver	6
7	see also	7

logic is the study of valid inference: which conclusions are forced by which premises, independent of what the premises are about. it splits cleanly into a **syntax** (formulas, and rules for pushing them around) and a **semantics** (truth assignments, models), and the deepest theorems in the subject are exactly the ones that say when the two agree (Epp, Susanna S., 2019).

mathematics runs on logic the way programs run on a cpu — invisibly, until something segfaults

1 propositional logic

1.1 connectives

atomic propositions p, q, r are statements that are either true or false. compound propositions are built with the connectives $\neg$ (not), $\wedge$ (and), $\vee$ (or), $\oplus$ (exclusive or), $\rightarrow$ (implies), $\leftrightarrow$ (iff). semantics by truth table:

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \oplus q$	$p \rightarrow q$	$p \leftrightarrow q$
T	T	F	T	T	F	T	T
T	F	F	F	T	T	F	F
F	T	T	F	T	T	T	F
F	F	T	F	F	F	T	T

the only row that ever surprises anyone: $p \rightarrow q$ is **true** whenever p is false. “if pigs fly then i am the pope” is vacuously true. the implication is logically equivalent to $\neg p \vee q$, which is the form every SAT solver actually stores.

an implication with a false hypothesis promises nothing, so it cannot be caught lying

two equivalences worth memorising because they generate most simplification work:

- **de morgan:** $\neg(p \wedge q) \equiv \neg p \vee \neg q$ and $\neg(p \vee q) \equiv \neg p \wedge \neg q$
- **contrapositive:** $p \rightarrow q \equiv \neg q \rightarrow \neg p$ (but NOT $\equiv q \rightarrow p$, the converse)

1.2 tautology, satisfiability, equivalence

for a formula φ over n variables there are 2^n truth assignments. φ is

- a **tautology** if every assignment makes it true (e.g. $p \vee \neg p$),
- **satisfiable** if at least one does,
- a **contradiction** if none does (e.g. $p \wedge \neg p$),

and $\varphi \equiv \psi$ iff $\varphi \leftrightarrow \psi$ is a tautology. the three notions interlock: φ is a tautology iff $\neg\varphi$ is unsatisfiable — which is why theorem proving and SAT solving are the same problem wearing different hats.

1.3 normal forms

- **DNF** (disjunctive normal form): an OR of ANDs of literals. read it straight off the truth table — one conjunct per **true** row. for $p \oplus q$: rows TF and FT give $(p \wedge \neg q) \vee (\neg p \wedge q)$.
- **CNF** (conjunctive normal form): an AND of ORs (clauses). one clause per **false** row, negating that row’s assignment. for $p \oplus q$: rows TT and FF give $(\neg p \vee \neg q) \wedge (p \vee q)$.

every formula therefore has both a DNF and a CNF — but the truth-table construction can blow up exponentially. the tseitin transformation fixes this for CNF: by introducing one fresh variable per subformula it produces an **equisatisfiable** (not equivalent) CNF of linear size, which is why CNF is the universal input format for SAT solvers.

1.4 functional completeness

a connective set is **functionally complete** if it can express every boolean function. $\{\neg, \wedge, \vee\}$ is complete — the DNF construction above builds any function from its truth table. NAND alone ($p \uparrow q \equiv \neg(p \wedge q)$) is also complete:

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

each right-hand side uses only $\uparrow$, and together they simulate a complete set, so $\{\uparrow\}$ is complete. NOR ($\downarrow$) is complete by the dual argument; $\{\wedge, \vee\}$ without negation is not — it is monotone (flipping an input from F to T can never flip the output from T to F), so $\neg p$ is out of reach.

this is why you can fabricate an entire cpu from nand gates alone

2 rules of inference

semantics checks all 2^n rows; deduction builds arguments row-free. an inference rule licenses a conclusion from premises; a **proof** is a chain of rule applications. the standard toolkit:

rule	schema
modus ponens	$p, p \rightarrow q \vdash q$
modus tollens	$\neg q, p \rightarrow q \vdash \neg p$
hypothetical syllogism	$p \rightarrow q, q \rightarrow r \vdash p \rightarrow r$
disjunctive syllogism	$p \vee q, \neg p \vdash q$
addition	$p \vdash p \vee q$
simplification	$p \wedge q \vdash p$
conjunction	$p, q \vdash p \wedge q$
resolution	$p \vee q, \neg p \vee r \vdash q \vee r$

each rule is **sound**: whenever the premises are true, the conclusion is true (check with a truth table — modus ponens is the tautology $((p \rightarrow q) \wedge p) \rightarrow q$, machine-verified below). the two classic fallacies are the rules that are *not* here: affirming the consequent ($q, p \rightarrow q \therefore p$) and denying the antecedent ($\neg p, p \rightarrow q \therefore \neg q$) — both fail on the row $p = F, q = T$.

3 first-order logic

propositional logic cannot even say “all men are mortal” — its atoms have no insides. first-order logic (FOL) adds:

- **terms**: variables, constants, function applications $f(x, c)$
- **predicates**: $P(x), x < y$ — atomic formulas about terms
- **quantifiers**: $\forall x$ (for all), $\exists x$ (there exists)

3.1 models and satisfaction

a formula on its own is neither true nor false; truth is relative to a **model** (structure) $\mathcal{M} = (D, I)$: a non-empty domain D plus an interpretation I assigning meanings to the constant, function and predicate symbols. write $\mathcal{M} \models \varphi$ for “ φ is true in $\mathcal{M}$ ”, defined by recursion on the formula (tarski’s definition): $\mathcal{M} \models \forall x \varphi$ iff φ holds for every choice of x from D , and so on.

the same sentence changes truth value as the model changes — that is the whole point:

- $\forall x \exists y (y > x)$ — true over $\mathbb{Z}$, true over $\mathbb{N}$, false over any finite ordered domain.
- $\exists y \forall x (y \leq x)$ — false over $\mathbb{Z}$ (no least integer), true over $\mathbb{N}$ (namely $y = 0$).

quantifier order matters: $\forall x \exists y (y > x)$ (“everyone has someone above them”) is much weaker than $\exists y \forall x (y > x)$ (“someone is above everyone” — false in any

model with order irreflexive, since $y > y$ fails). negation pushes through quantifiers by the FOL de morgan laws:

$$\neg \forall x \varphi \equiv \exists x \neg \varphi, \quad \neg \exists x \varphi \equiv \forall x \neg \varphi. \quad (1)$$

a sentence is **valid** if true in every model ($\models \varphi$), **satisfiable** if true in some model. unlike the propositional case there is no truth-table check — the space of models is infinite — and by church and turing, validity of FOL sentences is undecidable.

4 soundness and completeness

fix a proof system (natural deduction, say) and write $\Gamma \vdash \varphi$ for “ φ is derivable from assumptions Γ ”, and $\Gamma \models \varphi$ for “every model of Γ is a model of φ ”. the two headline theorems:

- **soundness**: $\Gamma \vdash \varphi \implies \Gamma \models \varphi$. the calculus proves nothing false. proved by induction on derivations — each rule preserves truth.
- **completeness** (gödel, 1929): $\Gamma \models \varphi \implies \Gamma \vdash \varphi$, for first-order logic. every semantic consequence has a formal proof.¹

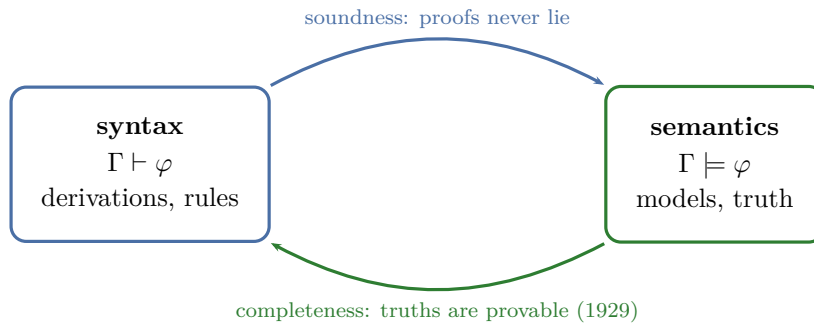


Figure 1: the two bridges between syntax and semantics. soundness is routine; completeness is gödel’s 1929 doctoral thesis.

a corollary of completeness worth its own line: **compactness** — if every finite subset of Γ has a model, so does Γ (any proof of a contradiction would use only finitely many assumptions). compactness powers non-standard analysis and half of model theory.

5 proof techniques

the working mathematician’s compiler targets. to prove $p \rightarrow q$ (Velleman, Daniel J., 2006):

- **direct**: assume p , chain forward to q . *example*: if m, n are even then $m + n$ is even. $m = 2a, n = 2b$, so $m + n = 2(a + b)$. done.

¹do not confuse this with gödel’s *incompleteness* theorems (1931), which concern **theories**, not the logic itself. first incompleteness: any consistent, effectively axiomatisable theory that interprets enough arithmetic (e.g. peano arithmetic) fails to prove some sentence that is true in the standard model $\mathbb{N}$ — the theory is negation-incomplete. second: no such theory proves its own consistency. there is no tension with the 1929 completeness theorem: completeness says the *deductive calculus* proves every sentence true in **all** models of the axioms; incompleteness says no effective axiom set pins down truth in the **one** intended model $\mathbb{N}$. the truths PA misses are true in $\mathbb{N}$ but false in some non-standard model of PA.

- **contrapositive:** prove $\neg q \rightarrow \neg p$ instead — logically the same statement, often structurally easier. *example:* if n^2 is even then n is even. contrapositive: n odd $\implies n^2$ odd. $n = 2k + 1 \implies n^2 = 2(2k^2 + 2k) + 1$. done. (the direct route from “ n^2 even” is awkward — you would have to factor n^2 .)
- **contradiction:** assume $p \wedge \neg q$, derive absurdity. *example:* $\sqrt{2}$ is irrational. suppose $\sqrt{2} = a/b$ in lowest terms. then $a^2 = 2b^2$, so a^2 is even, so a is even (previous bullet!), say $a = 2c$. then $4c^2 = 2b^2$, i.e. $b^2 = 2c^2$, so b is even too — contradicting lowest terms.
- **induction:** to prove $\forall n \geq n_0 P(n)$: establish $P(n_0)$ (**base**), and $P(k) \rightarrow P(k+1)$ for arbitrary $k \geq n_0$ (**step**). *example:* $\sum_{i=1}^n i = n(n+1)/2$. base: $n = 1$, both sides 1. step: $\sum_{i=1}^{k+1} i = k(k+1)/2 + (k+1) = (k+1)(k+2)/2$.
- **strong induction:** assume $P(n_0), \dots, P(k)$ all hold in the step. *example:* every integer $n \geq 2$ is a product of primes — if n is prime, done; else $n = ab$ with $2 \leq a, b < n$, and both factor by the strong hypothesis.

5.1 the well-ordering principle

WOP: every non-empty subset of $\mathbb{N}$ has a least element. this is equivalent to induction, and it is the engine inside “minimal counterexample” proofs: to prove $\forall n P(n)$, suppose the set of counterexamples $C = \{n : \neg P(n)\}$ is non-empty, take its least element m by WOP, and derive a contradiction — typically by manufacturing a smaller counterexample. one direction of the equivalence: assume induction, and let $C \subseteq \mathbb{N}$ have no least element; let $Q(n)$ say “no element of C is $\leq n$ ”; then $Q(0)$ holds (else 0 would be least in C) and $Q(k) \rightarrow Q(k+1)$ (else $k+1$ would be least), so by induction Q holds everywhere and C is empty.

the euclidean algorithm’s termination proof and the irrationality of root two are both secretly WOP arguments

6 SAT and resolution

6.1 the problem

SAT: given a CNF formula, decide whether some assignment satisfies it. this was the first problem ever proved NP-complete (the cook–levin theorem (Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. and Stein, Clifford, 2009)) — every problem in NP reduces to it — yet modern conflict-driven solvers routinely dispatch industrial instances with millions of variables. the gap between worst case and practice is one of the standing embarrassments (or miracles) of the field.

6.2 resolution

a single inference rule suffices for CNF refutation:

$$\frac{C \vee p \quad D \vee \neg p}{C \vee D} \quad (2)$$

resolve two clauses on a complementary literal pair, keep the union of the rest. resolution is sound, and **refutation-complete**: a CNF set is unsatisfiable iff the empty clause $\square$ is derivable from it by resolution.

worked example. show $\{p \vee q, \neg p \vee q, p \vee \neg q, \neg p \vee \neg q\}$ is unsatisfiable (it demands every combination of values for p and q be avoided at once):

1. resolve $p \vee q$ with $\neg p \vee q$ on p : get q .

2. resolve $p \vee \neg q$ with $\neg p \vee \neg q$ on p : get $\neg q$.
3. resolve q with $\neg q$: get $\perp$. unsatisfiable.

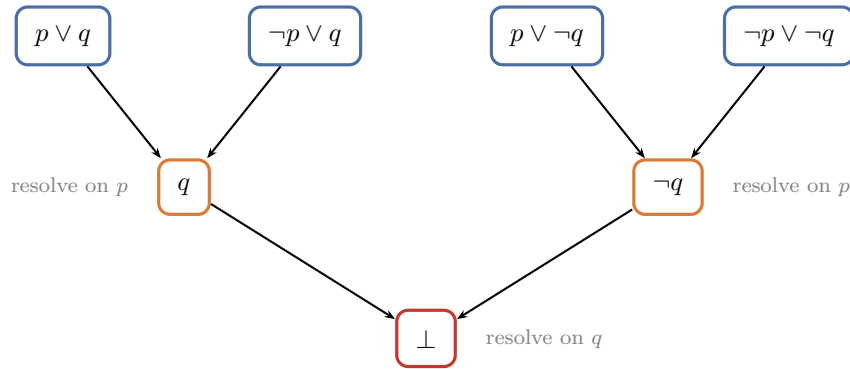


Figure 2: the refutation as a tree: four input clauses (top), two resolvents, and the empty clause. each junction resolves on one variable.

6.3 a DPLL-lite solver

the DPLL algorithm (1962, still the skeleton of every modern solver) is depth-first search plus one crucial deduction: **unit propagation** — a clause reduced to a single literal forces that literal's value, and the forcing cascades. below: a brute-force truth-table checker (to machine-verify modus ponens) and a miniature DPLL.

```

1  from itertools import product
2
3  # --- part 1: brute force over truth tables -----
4  def brute(n_vars, formula):
5      """return (tautology?, satisfiable?, #models) by trying all 2^n rows."""
6      models = [v for v in product([False, True], repeat=n_vars) if formula(*v)]
7      return len(models) == 2 ** n_vars, len(models) > 0, len(models)
8
9  mp = lambda p, q: ((not p or q) and p) <= q # ((p -> q) and p) -> q
10 xor3 = lambda p, q, r: (p ^ q ^ r) # parity on 3 bits
11
12 for name, n, f in [("modus ponens schema", 2, mp),
13                  ("p xor q xor r", 3, xor3)]:
14     taut, sat, k = brute(n, f)
15     print(f"{name:22s} tautology={taut:s:5s} satisfiable={sat:s:5s}
16           ↔ models={k}")
17
18 # --- part 2: DPLL-lite on CNF (clauses = frozensets of signed ints) -----
19 def unit_propagate(clauses, assign):
20     changed = True
21     while changed:
22         changed = False
23         for c in clauses:
24             if len(c) == 1:
25                 (lit,) = c
26                 assign[abs(lit)] = lit > 0
27                 clauses = simplify(clauses, lit)
28                 if clauses is None:
29                     return None, assign
30                 changed = True
31         break
32     return clauses, assign
33
34 def simplify(clauses, lit):
35     out = []
36     for c in clauses:

```

```

36     if lit in c:
37         continue # clause satisfied
38     if -lit in c:
39         c = c - {-lit} # literal falsified, drop it
40         if not c:
41             return None # empty clause: conflict
42     out.append(c)
43     return out
44
45 def dpll(clauses, assign=None, depth=0):
46     assign = dict(assign or {})
47     clauses, assign = unit_propagate([frozenset(c) for c in clauses], assign)
48     if clauses is None:
49         print(" " * depth + "conflict (empty clause)")
50         return None
51     if not clauses:
52         return assign # all clauses satisfied
53     lit = min(abs(l) for l in c for c in clauses) # smallest unassigned
54     ↪ var
55     print(" " * depth + f"branch on x{lit}")
56     for guess in (lit, -lit):
57         s = simplify(clauses, guess)
58         if s is not None:
59             r = dpll(s, {**assign, abs(guess): guess > 0}, depth + 1)
60             if r is not None:
61                 return r
62     return None
63
64 # the 4-clause unsatisfiable set from the resolution example (1=p, 2=q)
65 unsat = [{1, 2}, {-1, 2}, {1, -2}, {-1, -2}]
66 print("\nDPLL on {p|q, ~p|q, p|~q, ~p|~q}:")
67 print("result:", dpll(unsat))
68
69 # a satisfiable instance: (p|q|r) & (~p|q) & (~q|r) & (~r|p)
70 sat_inst = [{1, 2, 3}, {-1, 2}, {-2, 3}, {-3, 1}]
71 print("\nDPLL on {p|q|r, ~p|q, ~q|r, ~r|p}:")
72 print("result:", dpll(sat_inst))

```

```

modus ponens schema      tautology=True  satisfiable=True  models=4
p xor q xor r            tautology=False  satisfiable=True   models=4

```

DPLL on {p|q, ~p|q, p|~q, ~p|~q}:

branch on x1

conflict (empty clause)

conflict (empty clause)

result: None

DPLL on {p|q|r, ~p|q, ~q|r, ~r|p}:

branch on x1

result: {1: True, 2: True, 3: True}

read the traces: on the unsatisfiable set, guessing p either way triggers a unit-propagation cascade into the empty clause — the solver's two conflicts mirror the two halves of the resolution tree above. on the satisfiable instance, one branch on $p = \text{True}$ lets propagation finish the job: the clauses $\neg p \vee q$ and $\neg q \vee r$ fire as units, yielding the model $p = q = r = \text{True}$.

7 see also

- [sets, functions and sequences](#) — de morgan's laws in their set-theoretic costume

- **number theory** — induction and WOP doing real work (euclid’s algorithm, unique factorisation)
- **graph theory** — where “minimal counterexample” proofs earn their keep
- **classical algorithms** — backtracking, DPLL’s family tree

references

Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. and Stein, Clifford (2009). *Introduction to Algorithms*, MIT Press.

Epp, Susanna S. (2019). *Discrete Mathematics with Applications*, Cengage Learning.

Velleman, Daniel J. (2006). *How to Prove It: A Structured Approach*, Cambridge University Press.