Boolean Algebra

Discrete Math, Fall 2025

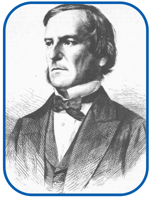
Konstantin Chukharev

Boolean Algebra

“Мы почитаем всех нулями,
А единицами — себя.”

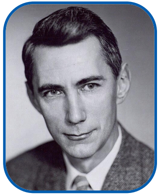
— А.С. Пушкин, «Евгений Онегин»


Gottfried Wilhelm Leibniz


George Boole


Augustus De Morgan


Charles Sanders Peirce


Claude Shannon

Definition and Basic Properties

Definition 1: A *Boolean algebra* is a bounded distributive lattice $(B, \vee, \wedge, \bot, \top)$ with complement $(\cdot)' : B \to B$ such that $x \vee x' = \top$ and $x \wedge x' = \bot$.

Example: $(\mathcal{P}(A), \cup, \cap, \varnothing, A)$ with $X' = A \setminus X$ is a Boolean algebra.

Example (Digital Circuit Design): Consider 3-bit binary values as Boolean algebra:

- Elements: $\{000, 001, 010, 011, 100, 101, 110, 111\}$
- Order: Bitwise comparison ($001 \leq 011$ since $0 \leq 0$, $0 \leq 1$, $1 \leq 1$)
- Join: Bitwise OR ($010 \vee 101 = 111$)
- Meet: Bitwise AND ($110 \wedge 101 = 100$)
- Complement: Bitwise NOT ($001' = 110$)

This directly corresponds to logic gates: OR, AND, NOT gates in computer processors.

Note: Logical reading: "join" $\mapsto \vee$, "meet" $\mapsto \wedge$, "complement" $\mapsto \neg$.

Example: Database Query Lattice

Example: A database has tables `Students`, `Courses`, `Enrollments`.

- Let Q_1 = "Computer Science students"
- Let Q_2 = "Students in Math courses"
- Let Q_3 = "Graduate students"

Consider queries as lattice elements ordered by result size (specificity).

Lattice Operations:

- $Q_1 \vee Q_2$ = "Students in CS OR Math courses" (larger result set)
- $Q_1 \wedge Q_2$ = "CS students taking Math courses" (smaller result set)
- $Q_1 \wedge Q_3$ = "Graduate CS students" (most specific)

Why this matters: Query optimizers use this structure to:

1. Find equivalent but more efficient queries.
2. Cache common subqueries.
3. Predict result set sizes for cost estimation.

Complement is Unique

Theorem 1: Complements are unique in a Boolean algebra.

Proof: Suppose for some element a we have *two* complements x and y.

$$
\begin{aligned}
x &= x \wedge \top && \top \text{ is the identity for } \wedge \\
&= x \wedge (a \vee y) && \text{by definition of complement: } \top = a \vee y \\
&= (x \wedge a) \vee (x \wedge y) && \wedge \text{ distributes over } \vee \\
&= \bot \vee (x \wedge y) && \text{by definition of complement: } \; x \wedge a = \bot \\
&= (a \wedge y) \vee (x \wedge y) && \text{by definition of complement: } \; \bot = a \wedge y \\
&= (a \vee x) \wedge y && \wedge \text{ distributes over } \vee \\
&= \top \wedge y && \text{by definition of complement: } \; a \vee x = \top \\
&= y && \top \text{ is the identity for } \wedge
\end{aligned}
$$

That is, $x = y$. □

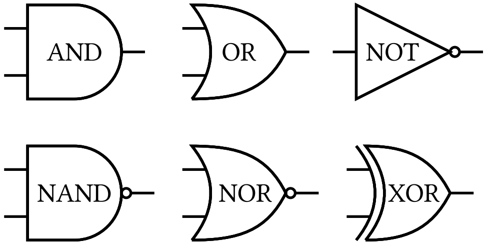
De Morgan's Laws

Theorem 2 (De Morgan): $(x \vee y)' = x' \wedge y'$ and $(x \wedge y)' = x' \vee y'$ in any Boolean algebra.

Digital Logic Circuits

Definition 2: A *logic gate* is a physical device that implements a Boolean function, taking binary inputs and producing a binary output.

Gate	Formula	Description
AND	$A \wedge B$	Outputs 1 only when both inputs are 1
OR	$A \vee B$	Outputs 1 when at least one input is 1
NOT	$\neg A$	Outputs the opposite of the input
NAND	$\neg(A \wedge B)$	Outputs 0 only when both inputs are 1
NOR	$\neg(A \vee B)$	Outputs 0 when at least one input is 1
XOR	$A \oplus B$	Outputs 1 when inputs differ
XNOR	$A \equiv B$	Outputs 1 when inputs are the same



Note: NAND and NOR gates are *universal* — any Boolean function can be implemented using only NAND gates (or only NOR gates). For example, to implement AND using NAND:

$$A \wedge B = \neg\neg(A \wedge B) = \neg(A \barwedge B) = (A \barwedge B) \barwedge (A \barwedge B)$$

Combinational Logic

Definition 3: A *combinational circuit* is a circuit where the output depends only on the current input values, without any memory or state.

Example (Half Adder): Adds two single bits:

- Sum: $S = A \oplus B$
- Carry: $C = A \land B$

Example (Full Adder): Adds two bits plus a carry-in:

- Sum: $S = A \oplus B \oplus C_{\text{in}}$
- Carry-out: $C_{\text{out}} = (A \land B) \lor (C_{\text{in}} \land (A \oplus B))$

Sequential Logic and Memory

Definition 4: A *sequential circuit* is a circuit where the output depends on both current inputs and previous state (memory).

Example (Flip-Flops):

- **SR Latch**: Set-Reset memory element.
- **D Flip-Flop**: Data storage triggered by clock edge.
- **JK Flip-Flop**: Eliminates forbidden state of SR latch.
- **T Flip-Flop**: Toggle flip-flop for counters.

Normal Forms

Definition 5: A *literal* is a Boolean variable or its negation (e.g., x, $\neg x$).

Definition 6 (DNF): A Boolean formula is in *disjunctive normal form (DNF)* if it is a disjunction (OR) of *terms* — conjunctions (AND) of literals.

Example: $$f(x, y, z) = \underbrace{(x \land y \land \neg z)}_{\text{term}} \lor \underbrace{(\neg x \land z)}_{\text{term}} \lor \underbrace{(\neg y \land \neg z)}_{\text{term}} \lor \underbrace{x}_{\text{term}}$$

Definition 7 (CNF): A Boolean formula is in *conjunctive normal form (CNF)* if it is a conjunction (AND) of *clauses* — disjunctions (OR) of literals.

Example: $$f(x, y, z) = \underbrace{(x \lor y \lor \neg z)}_{\text{clause}} \land \underbrace{(\neg x \lor z)}_{\text{clause}} \land \underbrace{(\neg y \lor \neg z)}_{\text{clause}} \land \underbrace{x}_{\text{clause}}$$

Minterms and Maxterms

Definition 8 (Minterm and Maxterm):

- A *minterm* is a product (AND) of literals where each variable appears exactly once.
- A *maxterm* is a sum (OR) of literals where each variable appears exactly once.

Note: A minterm (maxterm) is a function that evaluates to 1 (0, respectively) for exactly one combination of variable values.

Example: $f(x, y, z) = x\overline{y}z$ is a minterm, and $g(x, y, z) = x + \overline{y} + z$ is a maxterm for variables x, y, z.

- $f(x, y, z) = 1$ only on input 101, i.e., $x = 1$, $y = 0$, $z = 1$, corresponding to the minterm $x\overline{y}z$.
- $g(x, y, z) = 0$ only on input 010, i.e., $x = 0$, $y = 1$, $z = 0$, corresponding to the maxterm $\overline{x} + y + \overline{z}$.

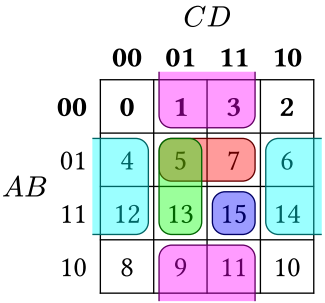
Canonical Forms

Definition 9 (SoP): Every Boolean function can be *uniquely* expressed as a *sum of minterms* (SoP, Sum of Products) corresponding to rows where the function evaluates to 1.

Definition 10 (PoS): Every Boolean function can be *uniquely* expressed as a *product of maxterms* (PoS, Product of Sums) corresponding to rows where the function evaluates to 0.

Karnaugh Maps

Definition 11: A *Karnaugh map* (K-map) is a graphical method for simplifying Boolean expressions by visually identifying adjacent minterms that can be combined.



Zhegalkin Polynomials

Definition 12: A *Zhegalkin polynomial* is a representation of a Boolean function as a polynomial over $GF(2)$ using XOR ($\oplus$) and AND ($\wedge$, often omitted) operations.

Theorem 3: Every Boolean function has a unique representation as a Zhegalkin polynomial:

$$f(x_1, \ldots, x_n) = \bigoplus_{S \subseteq \{1,\ldots,n\}} \left(a_S \prod_{i \in S} x_i \right)$$

where $a_S \in \{0, 1\}$ and $\oplus$ denotes XOR.

Example: $f(x, y) = x \vee y = x \oplus y \oplus xy$

Binary Decision Diagrams (BDDs)

Definition 13 (BDD): A *binary decision diagram (BDD)* is a directed acyclic graph representing a Boolean function, where each non-terminal node represents a variable test and edges represent variable assignments.

Definition 14 (ROBDD): A *reduced* ordered binary decision diagram (ROBDD) is an ordered BDD with a fixed variable ordering where:

- No variable appears more than once on any path
- No two nodes have identical low and high successors
- No node has identical low and high successors

Theorem 4: Every Boolean function has a unique reduced ordered binary decision diagram (ROBDD) representation for a given variable ordering.

TODO

- ...