5.5 Boolean Forms and Free Boolean Algebra

Boolean Forms

In Boolean algebra, Boolean forms (or Boolean expressions) are mathematical expressions built using Boolean variables and the operations **AND** (conjunction), **OR** (disjunction), and **NOT** (negation). These expressions represent logic functions and are used to model logical relationships and operations in digital circuits, computer science, and mathematics.

A **Boolean form** can be expressed as:

1. **Boolean Variables**: These are symbols (often represented as letters like AAA, BBB, XXX, etc.) that can take values of either 0 (false) or 1 (true).
2. **Boolean Operations**:
   - **AND** (·\cdot· or ∧\land∧): The result is true (1) only if both operands are true.
   - **OR** (+++ or ∨\lor∨): The result is true (1) if at least one operand is true.
   - **NOT** (¬\neg¬, ¯\overline{}, or ∼\sim∼): This negates the value of the operand. If the operand is 0, the result is 1, and vice versa.
3. **Canonical Forms**: Boolean expressions can be written in two important canonical forms:
   - **Sum of Products (SOP)**: A logical OR of ANDed terms. Each term is a product of literals (variables or their negations). Example: (A·B)+(¬A·C)(A \cdot B) + (\neg A \cdot C)(A·B)+(¬A·C).
   - **Product of Sums (POS)**: A logical AND of ORed terms. Example: (A+B)·(¬A+C)(A + B) \cdot (\neg A + C)(A+B)·(¬A+C).

Free Boolean Algebra

A **Free Boolean algebra** is a special kind of Boolean algebra that is abstract and not tied to a specific set of elements or values. It provides a formal structure for reasoning about Boolean expressions and allows the creation of Boolean functions from a set of free variables.

In more formal terms:

- A **Boolean algebra** is a set BBB equipped with two binary operations (AND, OR), a unary operation (NOT), and two constants (0 and 1) that satisfy the standard Boolean laws such as:
  - **Commutativity**: A+B=B+AA + B = B + AA+B=B+A, A·B=B·AA \cdot B = B \cdot AA·B=B·A
  - **Associativity**: (A+B)+C=A+(B+C)(A + B) + C = A + (B + C)(A+B)+C=A+(B+C), (A·B)·C=A·(B·C)(A \cdot B) \cdot C = A \cdot (B \cdot C)(A·B)·C=A·(B·C)
  - **Distributivity**: A·(B+C)=(A·B)+(A·C)A \cdot (B + C) = (A \cdot B) + (A \cdot C)A·(B+C)=(A·B)+(A·C), A+(B·C)=(A+B)·(A+C)A + (B \cdot C) = (A + B) \cdot (A + C)A+(B·C)=(A+B)·(A+C)
  - **Identity elements**: A+0=AA + 0 = AA+0=A, A·1=AA \cdot 1 = AA·1=A

    - **Complementation**: A+¬A=1A + \neg A = 1A+¬A=1, A·¬A=0A \cdot \neg A = 0A·¬A=0
- A **Free Boolean algebra** is one that is generated from a set of Boolean variables, but it does not impose any additional relations or constraints beyond the Boolean operations and identities.

In simple terms:

- A **Free Boolean algebra** has no relations other than the basic Boolean operations and laws, meaning it has no predefined values for the elements.
- The elements of a free Boolean algebra are formal symbols that can be manipulated algebraically, but they don't correspond to specific "truth values" (0 or 1) unless interpreted in a particular model.

Construction of a Free Boolean Algebra

A Free Boolean algebra can be constructed from a set of generators. If we take a set X={x1,x2,…,xn}X = \{ x_1, x_2, \dots, x_n \}X={x1,x2,…,xn} of Boolean variables, the free Boolean algebra over XXX is the algebra that consists of all Boolean expressions that can be formed from XXX using the Boolean operations.

The elements of a Free Boolean algebra are all possible Boolean combinations of the variables in the set. There are no additional relations, and the only rules are those that come from the properties of Boolean operations.

In the free Boolean algebra, every Boolean function can be seen as an algebraic expression formed by the free variables.

Applications and Importance

- **Designing Digital Circuits**: Free Boolean algebra is useful in simplifying Boolean expressions, which are central to the design of digital circuits (like AND, OR, NOT gates).
- **Theoretical Computer Science**: Boolean algebra and its free algebra versions are essential for understanding logic, algorithms, and data structures, especially in fields like formal verification, logic programming, and automata theory.
- **Optimization**: Boolean algebra plays a significant role in simplifying and optimizing logic expressions to reduce the number of gates needed in circuit design, which is crucial in hardware development.

Example of a Free Boolean Algebra:

Let's say we have a free Boolean algebra generated by two variables AAA and BBB. The elements of this algebra would consist of all possible Boolean expressions involving AAA and BBB. Some of the expressions could be:

- AAA
- ¬A\neg A¬A
- A·BA \cdot BA·B
- A+BA + BA+B
- ¬(A·B)\neg (A \cdot B)¬(A·B)
- (A+¬B)·(¬A+B)(A + \neg B) \cdot (\neg A + B)(A+¬B)·(¬A+B)

These are all distinct elements in the free Boolean algebra generated by AAA and BBB. None of these are tied to a specific truth value until you assign specific values (0 or 1) to AAA and BBB.

Boolean algebra is a mathematical structure used to work with logical values (usually 0 and 1) and operations (AND, OR, NOT). It's often used in computer science, electrical engineering, and digital circuit design. In Boolean algebra, expressions and equations are simplified to form minimal representations that are efficient for computation.

Common Boolean Operations:

1. **AND (Conjunction):**
   - Denoted by · (multiplication) or sometimes just by adjacency.
   - Truth table:
     A·B=1A \cdot B = 1A·B=1 if both A=1A = 1A=1 and B=1B = 1B=1, otherwise 000.
2. **OR (Disjunction):**
   - Denoted by + (addition).
   - Truth table:
     A+B=1A + B = 1A+B=1 if either A=1A = 1A=1 or B=1B = 1B=1 (or both), otherwise 000.
3. **NOT (Negation):**
   - Denoted by A¯\overline{A}A or A′A'A′.
   - A¯=1\overline{A} = 1A=1 if A=0A = 0A=0, and A¯=0\overline{A} = 0A=0 if A=1A = 1A=1.

Basic Boolean Laws and Properties:

1. **Identity Law:**
   - A·1=AA \cdot 1 = AA·1=A
   - A+0=AA + 0 = AA+0=A
2. **Null Law:**
   - A·0=0A \cdot 0 = 0A·0=0
   - A+1=1A + 1 = 1A+1=1
3. **Domination Law:**
   - A·0=0A \cdot 0 = 0A·0=0
   - A+1=1A + 1 = 1A+1=1
4. **Idempotent Law:**
   - A·A=AA \cdot A = AA·A=A
   - A+A=AA + A = AA+A=A

5. **Complement Law:**
   - $A \cdot \overline{A} = 0$
   - $A + \overline{A} = 1$
6. **Distributive Law:**
   - $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
   - $A + (B \cdot C) = (A + B) \cdot (A + C)$
7. **De Morgan's Laws:**
   - $\overline{A \cdot B} = \overline{A} + \overline{B}$
   - $\overline{A + B} = \overline{A} \cdot \overline{B}$
8. **Double Negation:**
   - $\overline{\overline{A}} = A$
9. **Absorption Law:**
   - $A \cdot (A + B) = A$
   - $A + (A \cdot B) = A$

Boolean Forms:

1. **Sum of Products (SOP):**
   - A Boolean expression in which several terms are OR-ed together, where each term is a product (AND) of literals (variables or their negations).

   Example: $(A \cdot B) + (A' \cdot C)$

2. **Product of Sums (POS):**
   - A Boolean expression in which several terms are AND-ed together, where each term is a sum (OR) of literals.

   Example: $(A + B) \cdot (A' + C)$

Simplification of Boolean Expressions:

1. **Using Boolean laws** to reduce expressions to simpler forms.
2. **Truth Tables** are sometimes used to verify expressions.
3. **Karnaugh Maps (K-Maps)** provide a graphical method to simplify Boolean expressions.
4.

Examples and Problems:

Problem 1: Simplify the Boolean Expression

Expression: $A \cdot (A + B)$

Solution: Using the Absorption Law: $A \cdot (A + B) = A$

Problem 2: Simplify the Boolean Expression

Expression: A·A‾+A·BA \cdot \overline{A} + A \cdot BA·A+A·B

Solution: By the Complement Law, A·A‾=0A \cdot \overline{A} = 0A·A=0, so: A·A‾+A·B=0+A·B=A·BA \cdot \overline{A} + A \cdot B = 0 + A \cdot B = A \cdot BA·A+A·B=0+A·B=A·B

Problem 3: Simplify using De Morgan's Law

Expression: A·B‾\overline{A \cdot B}A·B

Solution: By De Morgan’s Law: A·B‾=A‾+B‾\overline{A \cdot B} = \overline{A} + \overline{B}A·B=A+B

Problem 4: Express in Sum of Products (SOP)

Expression: A+(B·C)A + (B \cdot C)A+(B·C)

Solution: The expression is already in SOP form, as it is the sum (OR) of AAA and B·CB \cdot CB·C.

Problem 5: Simplify the Boolean Expression

Expression: (A+B)·(A+C)(A + B) \cdot (A + C)(A+B)·(A+C)

Solution: Using the Distributive Law: (A+B)·(A+C)=A+(B·C)(A + B) \cdot (A + C) = A + (B \cdot C)(A+B)·(A+C)=A+(B·C)

Problem 6: Construct and Simplify a Boolean Expression using a Truth Table

Let's create a truth table for the expression: A·B‾+A′·BA \cdot \overline{B} + A' \cdot BA·B+A′·B

A	B	B‾\overline{B}B	A·B‾A \cdot \overline{B}A·B	A′A'A′	A′·BA' \cdot BA′·B	A·B‾+A′·BA \cdot \overline{B} + A' \cdot BA·B+A′·B
0	0	1	0	1	0	0
0	1	0	0	1	1	1
1	0	1	1	0	0	1

A	B	B¯\overline{B}B	A·B¯A \cdot \overline{B}A·B	A′A'A′	A′·BA' \cdot BA′·B	A·B¯+A′·BA \cdot \overline{B} + A' \cdot BA·B+A′·B
1	1	0	0	0	0	0

From the truth table, the simplified Boolean expression is: A·B¯+A′·BA \cdot \overline{B} + A' \cdot BA·B+A′·B This is a form of the **XOR** (exclusive OR) operation.

Problem 7: Simplify the Boolean Expression Using K-map

Expression: A·B+A′·B′A \cdot B + A' \cdot B'A·B+A′·B′

We can represent this expression using a K-map for two variables (A, B):

A \ B	0	1
0	1	0
1	0	1

In the K-map, we have two ones in the cells (0,0) and (1,1). Grouping these together gives: A·B+A′·B′=A⊕BA \cdot B + A' \cdot B' = A \oplus BA·B+A′·B′=A⊕B This is the XOR (exclusive OR) operation.