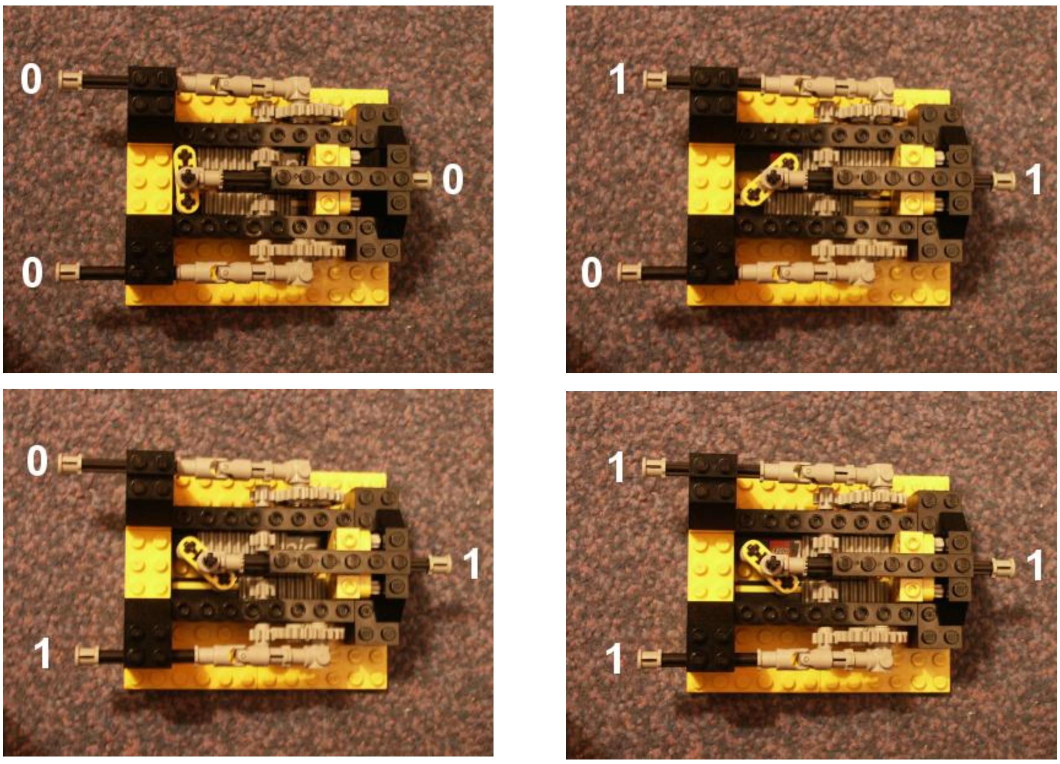
Switching Circuits & Logic Design

Jie-Hong Roland Jiang
江介宏

Department of Electrical Engineering
National Taiwan University

Fall 2013

§2 Boolean Algebra

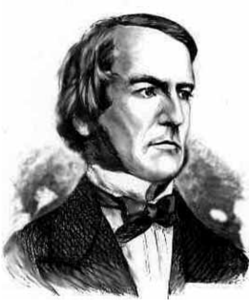


Outline

- Introduction
- Basic operations
- Boolean expressions and truth tables
- Basic theorems
- Commutative, associative, and distributive laws
- Simplification theorems
- Multiplying out and factoring
- DeMorgan's laws

Introduction

- Boolean algebra is the mathematical foundation of logic design
  - George Boole (1847)
    - logic + algebra → Boolean algebra



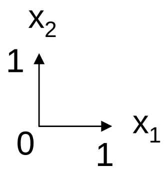
  - Claude Shannon (1939)
    - Boolean algebra ↔ logic design

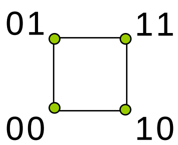
Introduction

- Boolean (switching) variable $x \in \{0,1\}$
  - 0, 1 are abstract symbols
    - They may correspond to {false, true} in logic, {off, on} of a switch, {low voltage, high voltage} of a CMOS circuit, or other meanings
- Boolean space $\{0,1\}^n$
  - The configuration space of all possible {0,1} assignments to n Boolean variables

    E.g.,

    the Boolean space spanned by (x_1,x_2) is $\{0,1\}^2 = \{0,1\}\times\{0,1\} = \{00, 01, 10, 11\}$

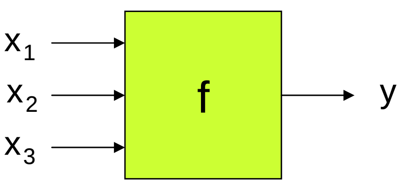


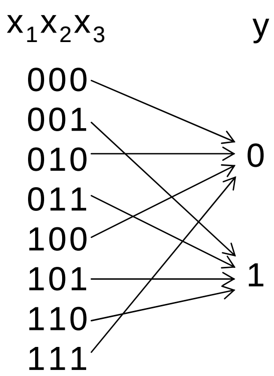


Introduction

- Boolean function $f(x_1, x_2, \ldots, x_n)$ is a mapping: $\{0,1\}^n \rightarrow \{0,1\}$, where x_i's are Boolean variables

E.g.,

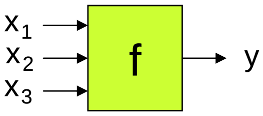




How many Boolean functions of n variables are there?

Introduction

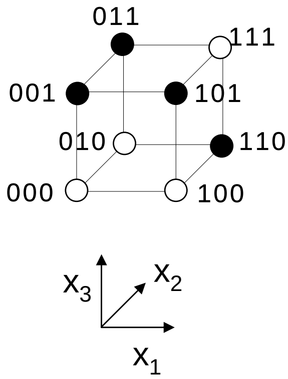
- There are many different ways to represent a Boolean function
  - E.g., truth tables, Boolean expressions (formulas), logic circuits, Binary Decision Diagrams, combinatorial cubes, …



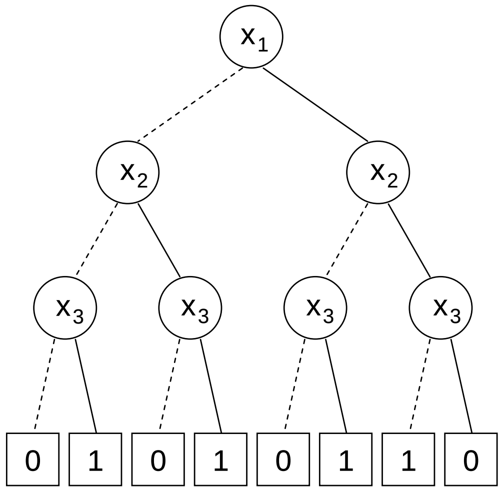
Truth table

$x_1x_2x_3$	y
000	0
001	1
010	0
011	1
100	0
101	1
110	1
111	0

Combinatorial cube



Binary decision diagram



Introduction

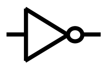
- Different Boolean-function representations have their own strengths and weaknesses
  - They affect the computational efficiency of Boolean manipulations in logic synthesis, hardware/software verification, and many other applications

- Truth tables, Boolean expressions, and logic circuits will be our main use in representing Boolean functions
  - Boolean expressions and logic circuits are closely related
    - They are built up from *logic operators* and *Boolean variables*

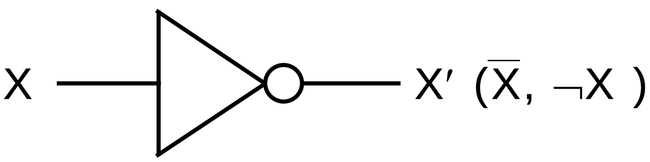
Basic Operations

- Three most basic operations in Boolean algebra: {AND, OR, NOT}
  - They form a **functionally complete** set of operations, that is, any Boolean functions can be constructed using these three operations (why?)
  - Are {AND, NOT} functionally complete?

Basic Operations
NOT

- NOT (complement, or inverse)
  - Notation: “ ′ ”, “ ¯ ”,or “¬”
  - Logic gate symbol: 

$$\begin{cases} 0' = 1 \\ 1' = 0 \end{cases} \qquad \begin{cases} X' = 1 \text{ if and only if } X = 0 \\ X' = 0 \text{ if and only if } X = 1 \end{cases}$$

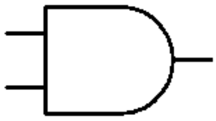


NOT-gate, inverter

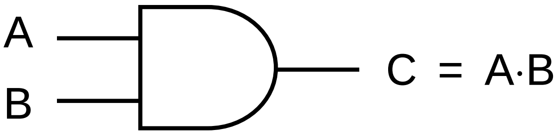
X	X′
0	1
1	0

Basic Operations
AND

- AND (conjunction)
  - Notation: " · ", "∧"
  - Logic gate symbol:



$$\begin{cases} 0 \cdot 0 = 0 \\ 0 \cdot 1 = 0 \\ 1 \cdot 0 = 0 \\ 1 \cdot 1 = 1 \end{cases}$$

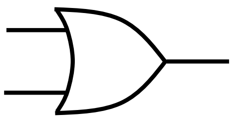


AND-gate

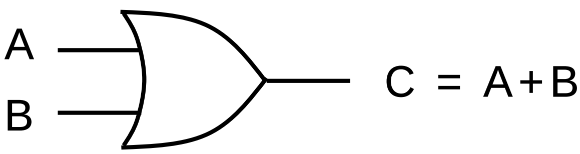
AB	C=A·B
00	0
01	0
10	0
11	1

Basic Operations
OR

- OR (disjunction)
  - Notation: "+", "∨"
  - Logic gate symbol:



$$\begin{cases} 0 + 0 = 0 \\ 0 + 1 = 1 \\ 1 + 0 = 1 \\ 1 + 1 = 1 \end{cases}$$

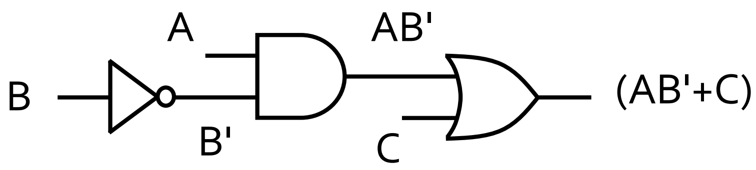


OR-gate

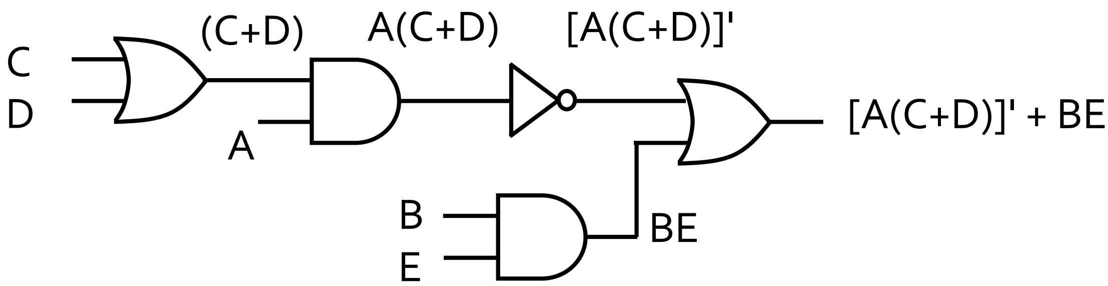
AB	C=A+B
00	0
01	1
10	1
11	1

Boolean Expressions & Logic Circuits

- AB′+C



- [A(C+D)]′+BE



Boolean Expressions & Logic Circuits

- Given a Boolean expression, we can construct a functionally equivalent logic circuit (not unique)
- Given a logic circuit, we can derive a Boolean expression of the corresponding Boolean function
- Given a Boolean expression or logic circuit, we can derive the truth table of the corresponding Boolean function

Boolean Expressions & Logic Circuits

- A Boolean expression (logic circuit) gives a unique Boolean function
  - The converse is not true, that is, a Boolean function can be represented by different Boolean expressions (logic circuits)

- A truth table gives a unique Boolean function, and vice versa
  - Truth tables are **canonical** in representing Boolean functions
  - Can use truth tables to show the equivalence of two Boolean functions

Boolean Expressions & Truth Tables

Truth-table proof of AB′+C = (A+C)(B′+C)
(equivalence under all truth assignments)

ABC	B′	AB′	AB′+C	A+C	B′+C	(A+C)(B′+C)
000	1	0	0	0	1	0
001	1	0	1	1	1	1
010	0	0	0	0	0	0
011	0	0	1	1	1	1
100	1	1	1	1	1	1
101	1	1	1	1	1	1
110	0	0	0	1	0	0
111	0	0	1	1	1	1

Basic Theorems of Boolean Algebra

- Operations with 0 and 1:
  - $X + 0 = X \overset{\text{dual}}{\Longleftrightarrow} X \cdot 1 = X$
  - $X + 1 = 1 \Longleftrightarrow X \cdot 0 = 0$
- Idempotent laws
  - $X + X = X \Longleftrightarrow X \cdot X = X$

Duality: interchange "0" and "1" and interchange "+" and " · "

Basic Theorems of Boolean Algebra

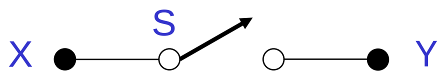
- Involution law
  - $(X')' = X$
- Laws of complementarity
  - $X + X' = 1 \Longleftrightarrow X \cdot X' = 0$

Applications to logic simplification

E.g., $(AB'+D)E+1 = 1$

$(AB'+D)(AB'+D)' = 0$

Boolean Algebra with Switches



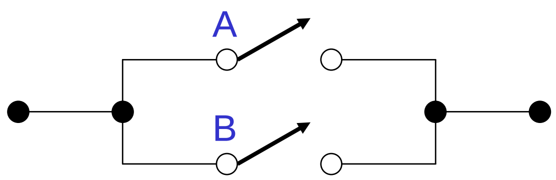
S = 0, switch open
S = 1, switch closed

X and Y are connected if and only if S = 1

The connectivity between X and Y is a function over S

Boolean Algebra with Switches

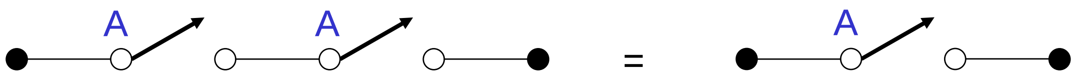
X and Y are connected if and only if A·B = 1



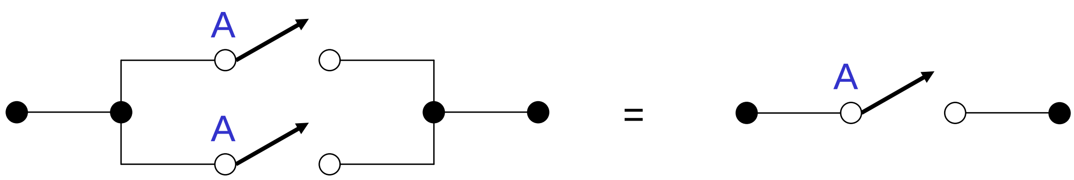
X and Y are connected if and only if A+B = 1

Boolean Algebra with Switches
Basic Theorems Revisited

- Idempotent laws



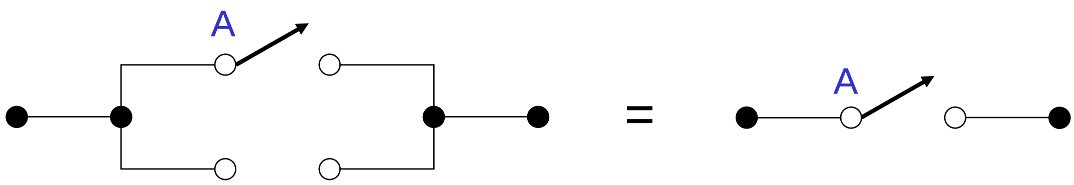
(A · A = A)



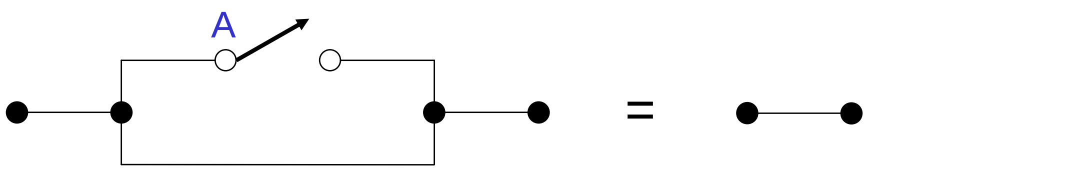
(A + A = A)

Boolean Algebra with Switches
Basic Theorems Revisited

- Operations with 0 and 1



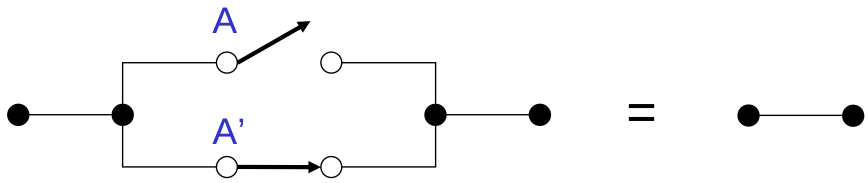
(A + 0 = A)



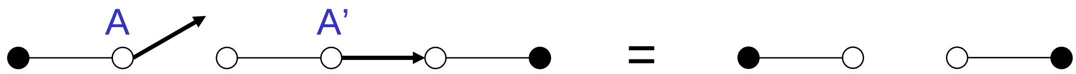
(A + 1 = 1)

Boolean Algebra with Switches
Basic Theorems Revisited

- Laws of complementarity



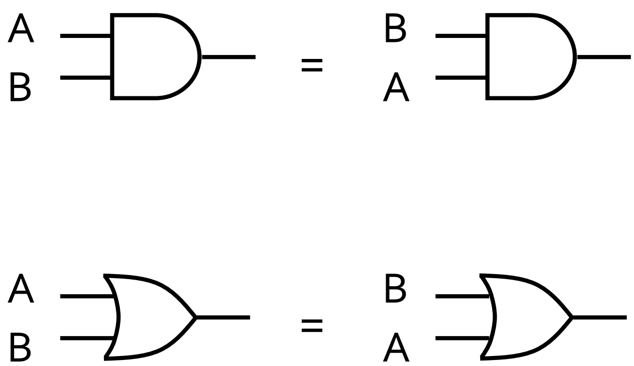
(A + A′ = 1)



(A · A′ = 0)

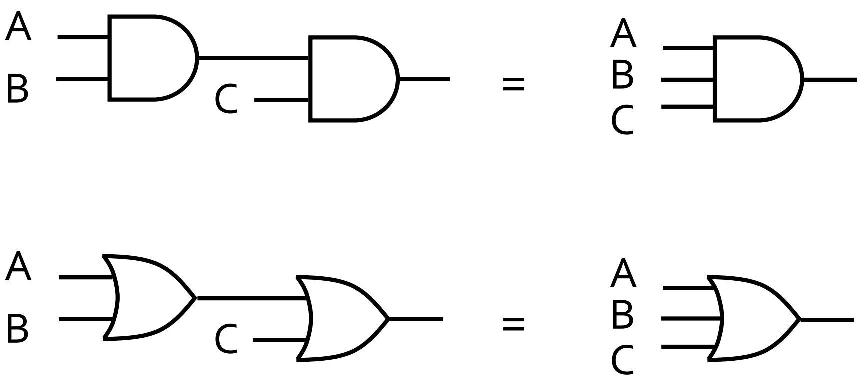
Commutative, Associative, and Distributive laws

- Commutative laws
  - X·Y = Y·X ⟺ X+Y = Y+X



Commutative, Associative, and Distributive laws

- Associative laws
  - (XY)Z = X(YZ) = XYZ ⟺
    (X+Y)+Z = X+(Y+Z) = X+Y+Z



Commutative, Associative, and Distributive laws

- Distributive laws
  - X(Y+Z) = XY+XZ ⟺ X+YZ = (X+Y)(X+Z)
    - The second equality is valid for Boolean algebra but not for ordinary algebra

  Proof.

  (X+Y)(X+Z) =
  XX+XZ+YX+YZ =
  X+XZ+XY+YZ =
  X·1+XZ+XY+YZ =
  X(1+Z+Y)+YZ =
  X·1+YZ =
  X+YZ

Simplification Theorems

- $XY + XY' = X \iff (X+Y)(X+Y') = X$
- $X+XY = X \iff X(X+Y) = X$

  Proof.

  $X+XY = X \cdot 1+XY = X(1+Y) = X \cdot 1 = X$

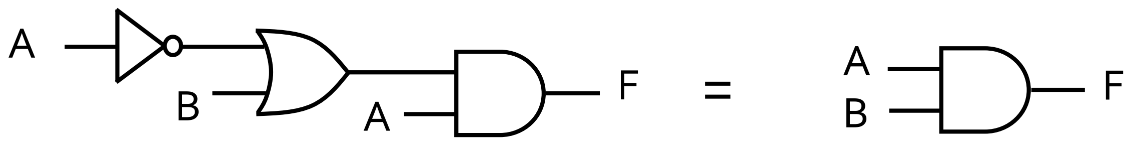
  $X(X+Y) = XX+XY = X+XY = X$
- $(X+Y')Y = XY \iff XY'+Y = X+Y$

  Proof.

  $Y+XY' = (Y+X)(Y+Y') = (Y+X) \cdot 1 = X+Y$

Logic Circuit Simplification

- $F = A(A'+B) = AA'+AB = 0+AB = AB$



Exercise (p.48)

- Simplify $Z = [A+B'C+D+EF]\ [A+B'C+(D+EF)']$
- Simplify $Z = (AB+C)(B'D+C'E')+(AB+C)'$

Multiplying Out and Factoring

- Sum-of-products (SOP), or Disjunctive Normal Form (DNF)
  - Sum of products of **literals** (a literal is a variable x or its complement x′)
    - E.g., ab′c+a′bd yes
      a+b+c yes
      abc yes
      a(b′+c)+a′bd no
  - Any Boolean function can be represented in the SOP form (Why?)

- Product-of-sums (POS), or Conjunctive Normal Form (CNF)
  - Product of **clauses** (a clause is a sum of literals)
    - E.g., (a+b+c)(a′+d) yes
      (a+b+c) yes
      (a)(b)(c) yes
      (a+bc)(a′+d) no
  - Any Boolean function can be represented in the POS form (Why?)

Multiplying Out

- SOP
  - When multiplying out an expression (to obtain an SOP), the 2nd distributive law
    (X+Y)(X+Z) = X+YZ
    can be applied first when possible to simply the expression

  E.g.,

  (A+BC)(A+D+E) = A+BC(D+E) = A+BCD+BCE

  X Y X Z X Y Z

  In contrast to,

  (A+BC)(A+D+E) = A+AD+AE+ABC+BCD+BCE

  = A(1+D+E+BC)+BCD+BCE = A+BCD+BCE

Factoring

- POS
  - Apply distributive laws

    $XY+XZ = X(Y+Z)$

    $X+YZ = (X+Y)(X+Z)$

    to factor an expression in the POS form
    - Any expression can be factored to the POS form
    - An expression cannot be further factored if and only if it is in the POS form

E.g.,

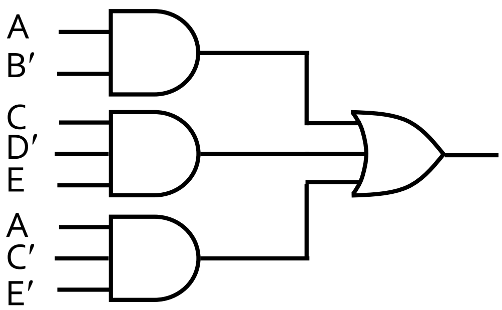
$$(A+B'CD) = (A+B')(A+CD) = (A+B')(A+C)(A+D)$$

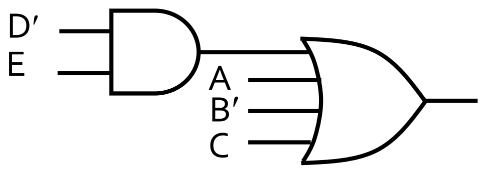
$$(AB'+CD) = (AB'+C)(AB'+D) = (A+C)(B'+C)(A+D)(B'+D)$$

Exercise (p.51): Factor $(C'D+C'E'+G'H)$

Multiplying Out and Factoring

- SOP in AND-OR circuit

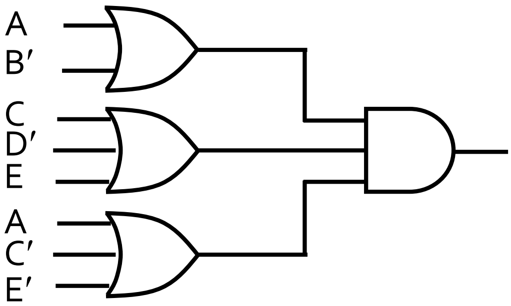


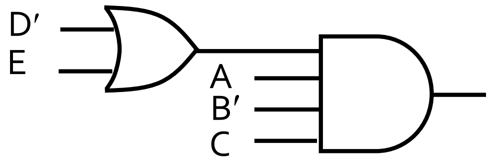


$AB'+CD'E+AC'E'$ $\qquad$ $A+B'+C+D'E$

Multiplying Out and Factoring

- POS in OR-AND circuit





(A+B′)(C+D′+E)(A+C′+E′)

AB′C(D′+E)

DeMorgan's Laws

- Complement by DeMorgan's laws
  - (X+Y)′ = X′ · Y′
  - (X · Y)′ = X′ + Y′

Proof by truth table

XY	X+Y	(X+Y)′	X′·Y′	XY	(X·Y)′	X′+Y′
00	0	1	1	0	1	1
01	1	0	0	0	1	1
10	1	0	0	0	1	1
11	1	0	0	1	0	0

Generalized DeMorgan's Laws

- $(X_1+X_2+\cdots+X_n)' = X_1' X_2' \cdots X_n'$
  - Complement of sum = product of complements
- $(X_1 X_2 \cdots X_n)' = X_1'+X_2'+\cdots+X_n'$
  - Complement of product = sum of complements

E.g.,

$[(A'+B)C']' = (A'+B)'+(C')' = AB'+C$

$[(AB'+C)D'+E]' = [(AB'+C)D']'E' = [(AB'+C)'+D]E' = [(AB')'C'+D]E' = [(A'+B)C'+D]E'$

Duality

- The dual F^D of an expression F is formed by replacing AND with OR, OR with AND, 0 with 1, and 1 with 0
  - F^D can also be obtained by complementing F and then complementing each individual variable

  E.g.,

  $(AB'+C)^D = (A+B')C$

- Equalities are preserved under duality, i.e.,

  F = G iff $F^D = G^D$ (justify prior theorems)

  E.g.,

  $X(Y+Z) = XY+XZ$ ⟺ dual ⟺ $X+YZ = (X+Y)(X+Z)$