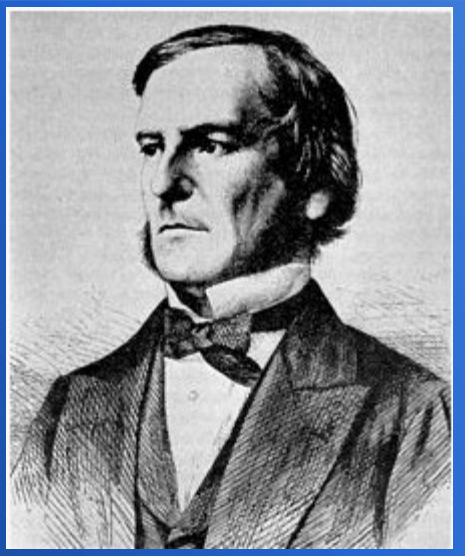
Boolean Algebra



George Boole 2 November 1815 – 8 December 1864) was an English mathematician, philosopher and logician.

Appointed in 1849 as the first professor of mathematics at Queen's College, Cork in Ireland.

Developed a new form of mathematics that is now known as Boolean Algebra. Boole's intention was to use mathematical techniques to represent and rigorously test logical and philosophical arguments. Boole established a new mathematical field known as symbolic logic.

The basic mathematics needed for the study of the logic design of digital systems is Boolean algebra.

Boolean algebra has many other applications including set theory and mathematical logic, but we will study its application to switching circuits.

In the 1930s, while studying switching circuits, **Claude Shannon** observed that one could also apply the rules of Boole's algebra in this setting, and he introduced switching algebra as a way to analyse and design circuits by algebraic means.

Because all of the switching devices which we will use are essentially two-state devices (such as a transistor with high or low output voltage), we will study the special case of Boolean algebra in which all of the variables assume only one of two values.

This two-valued Boolean algebra can be referred to as switching algebra.

Summary of basic logic functions

Several different notations commonly used:-

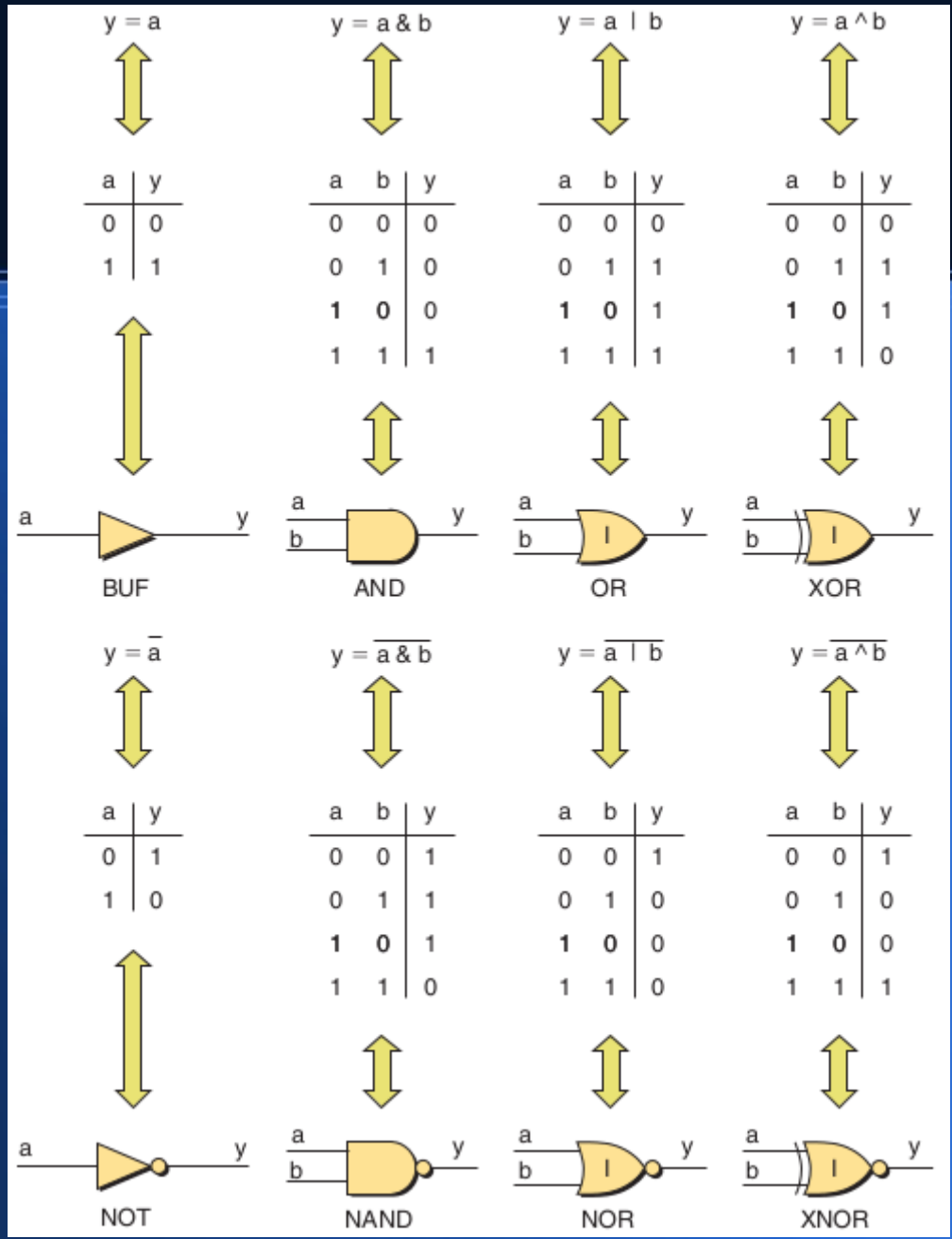
AND A&B A.B AB

OR A|B A+B

NOT $\overline{AB}$ (AB)'

EOR A^B $A \oplus B$

i.e. 5A 5 x A ordinary algebra



DEMORGAN TRANSFORMATIONS

Augustus De Morgan (27 June 1806 – 18 March 1871) was a British mathematician and logician. He formulated De Morgan's laws and introduced the term mathematical induction, making its idea rigorous. The crater De Morgan on the Moon is named after him

The Duality Principle, also called De Morgan duality, asserts that Boolean algebra is unchanged when all dual pairs are interchanged.

There is nothing magical about the choice of symbols for the values of Boolean algebra. We could rename 0 and 1 to say α and β, and as long as we did so consistently throughout it would still be Boolean algebra, albeit with some obvious cosmetic differences.

But suppose we rename 0 and 1 to 1 and 0 respectively. Then it would still be Boolean algebra, and moreover operating on the same values. However it would not be identical to our original Boolean algebra because now we find & behaving the way | used to do and vice versa.

So there are still some cosmetic differences to show that we've been fiddling with the notation, despite the fact that we're still using 0s and 1s.

In addition to interchanging the names of the values we also interchange the names of the two binary operations, now there is no trace of what we have done. The end product is completely indistinguishable from what we started with. We might notice that the columns for x&y and x|y in the truth tables had changed places, but that switch is immaterial.

When values and operations can be paired up in a way that leaves everything important unchanged when all pairs are switched simultaneously, we call the members of each pair dual to each other.

Thus 0 and 1 are dual, and & and | are dual.

A truth table (also called a table of combinations) specifies the values of a Boolean expression for every possible combination of values of the variables in the expression.

$A\ B\ C$	B'	AB'	$AB' + C$	$A + C$	$B' + C$	$(A + C)(B' + C)$
0 0 0	1	0	0	0	1	0
0 0 1	1	0	1	1	1	1
0 1 0	0	0	0	0	0	0
0 1 1	0	0	1	1	1	1
1 0 0	1	1	1	1	1	1
1 0 1	1	1	1	1	1	1
1 1 0	0	0	0	1	0	0
1 1 1	0	0	1	1	1	1

The name truth table comes from a similar table which is used in symbolic logic to list the truth or falsity of a statement under all possible conditions.

We can use a truth table to specify the output values for a circuit of logic gates in terms of the values of the input variables

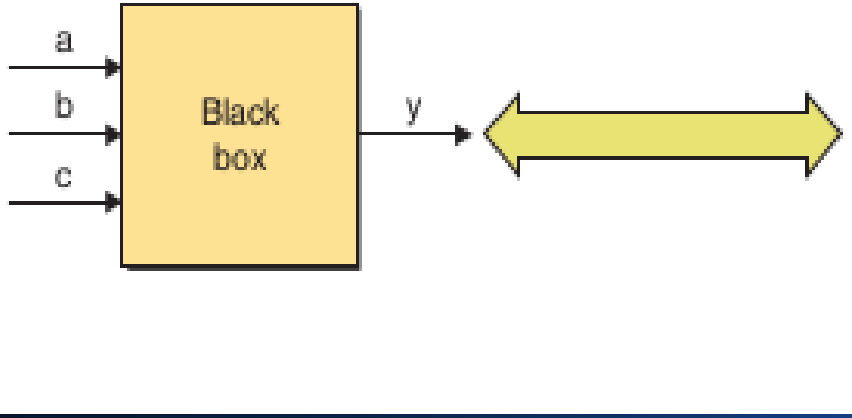
If an expression has n variables, and each variable can have the value 0 or 1, the number of different combinations of values of the variables is

$$2 \times 2 \times 2 \times \ldots = 2^n$$

Therefore, a truth table for an n-variable expression will have 2^n rows

Black Box

A designer will often specify portions of a design using truth tables, and determine how to implement these functions as logic gates later. The designer may start by representing a function as a black box with an associated truth table



a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

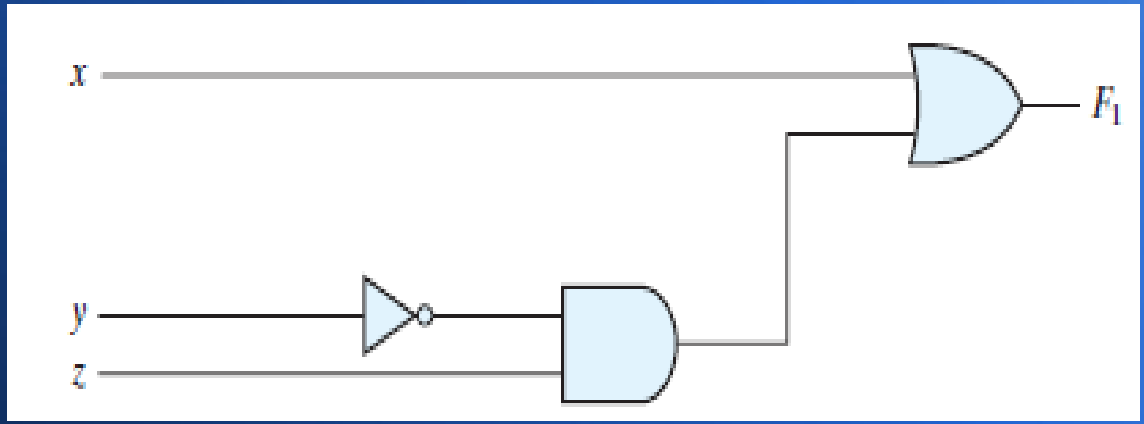
A boolean function described by an algebraic expression consists of binary variables, the constants 0 and 1, and the logic operation symbols. For a given value of the binary variables, the function can be equal to either 1 or 0.

$F_1 = x + y'z$

Truth Tables for F_1 and F_2

x	y	z	F_1	F_2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0

Truth Tables

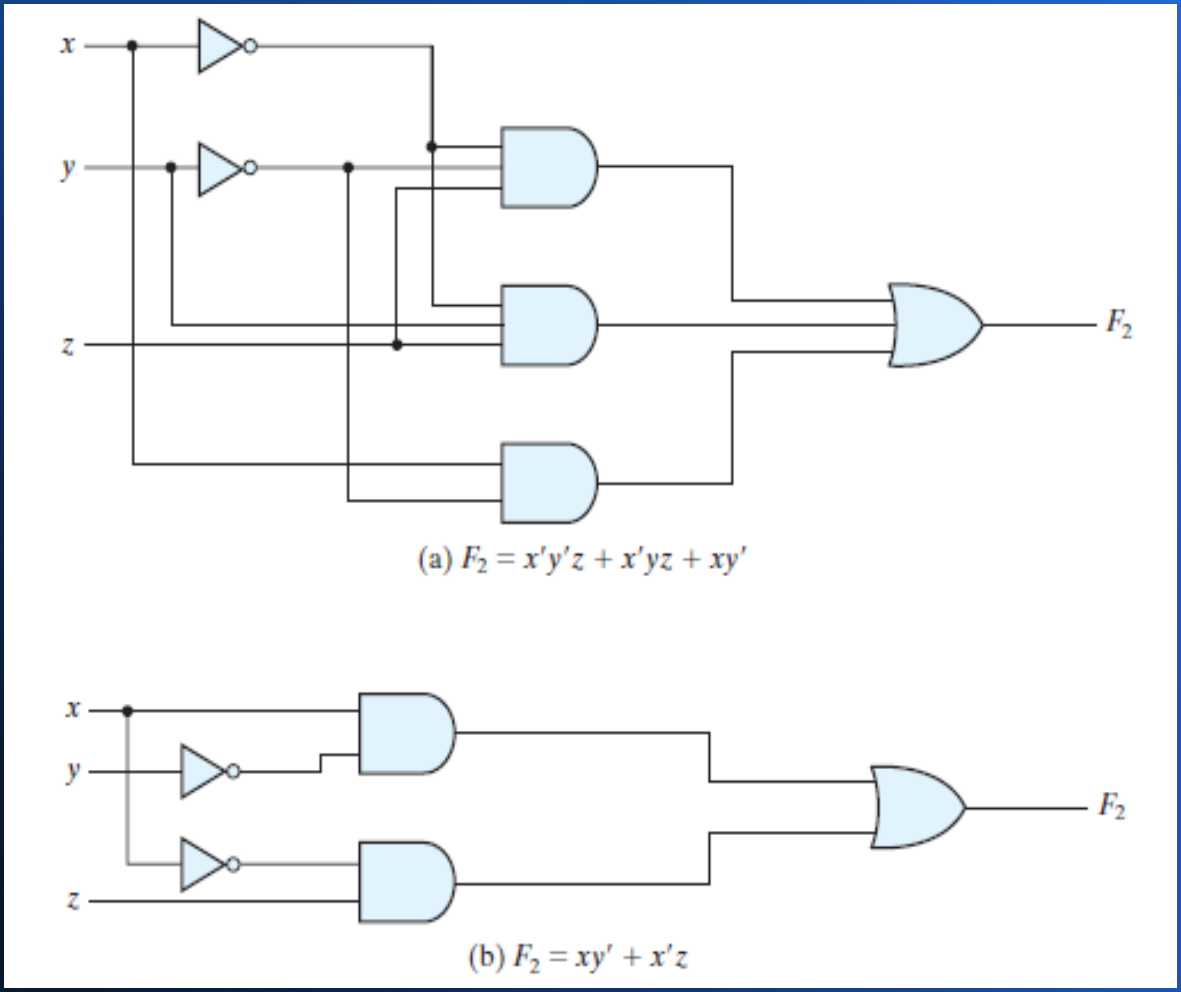


Gate implementation

There is only one way that a Boolean function can be represented in a truth table. However, when the function is in algebraic form, it can be expressed in a variety of ways, all of which have equivalent logic.

Here is a key fact that motivates our use of Boolean algebra: By manipulating a Boolean expression according to the rules of Boolean algebra, it is sometimes possible to obtain a simpler expression for the same function and thus reduce the number of gates in the circuit and the number of inputs to the gate.

$F_2 = x'y'z + x'yz + xy' = x'z(y' + y) + xy' = x'z + xy'$

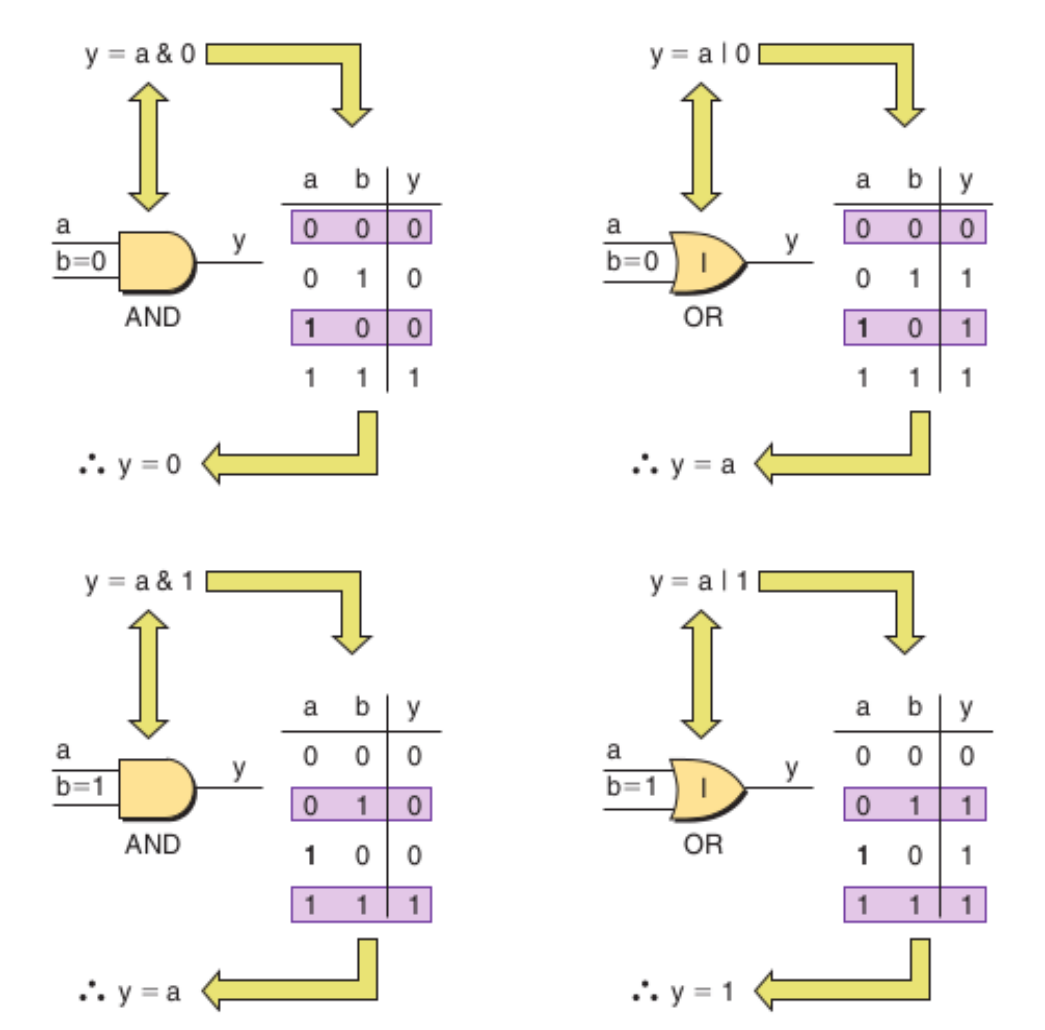


There are 2^(2^n) functions for n binary variables. Thus, for two variables, n = 2, and the number of possible Boolean functions is 16.

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

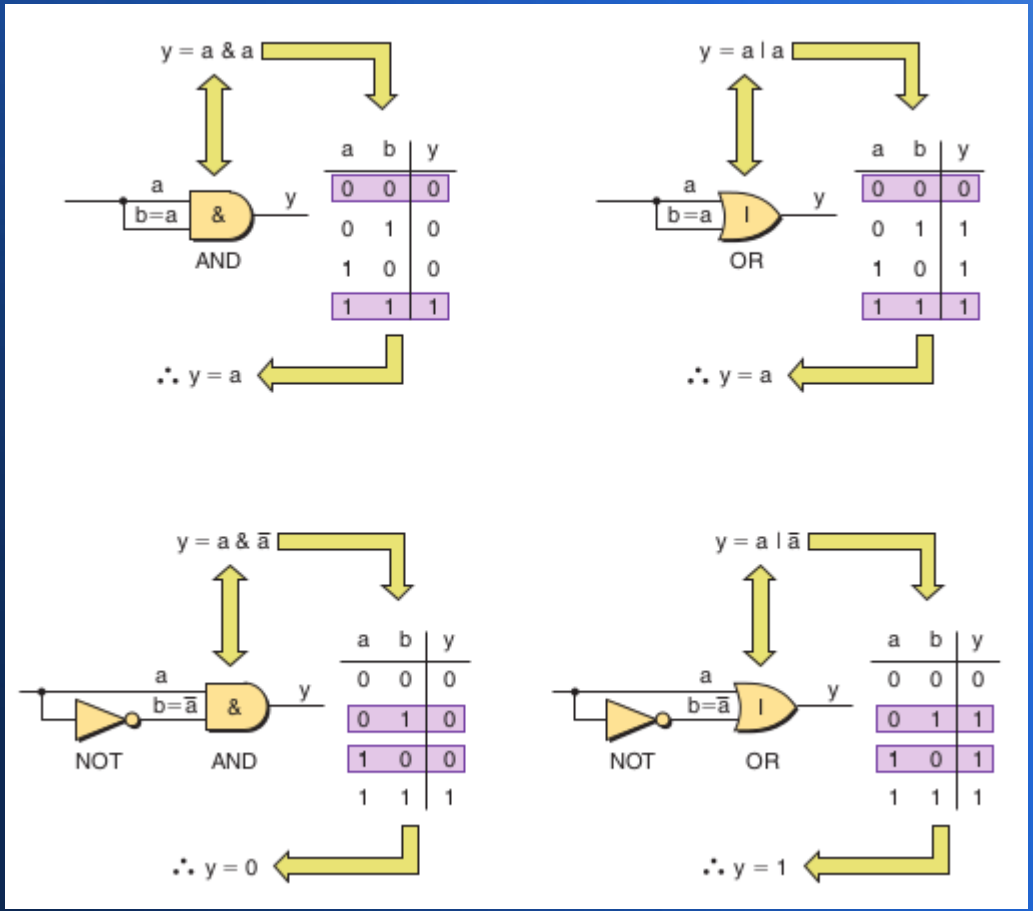
Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y, then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x, then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

Combining a single variable with a logic 0 or a logic 1

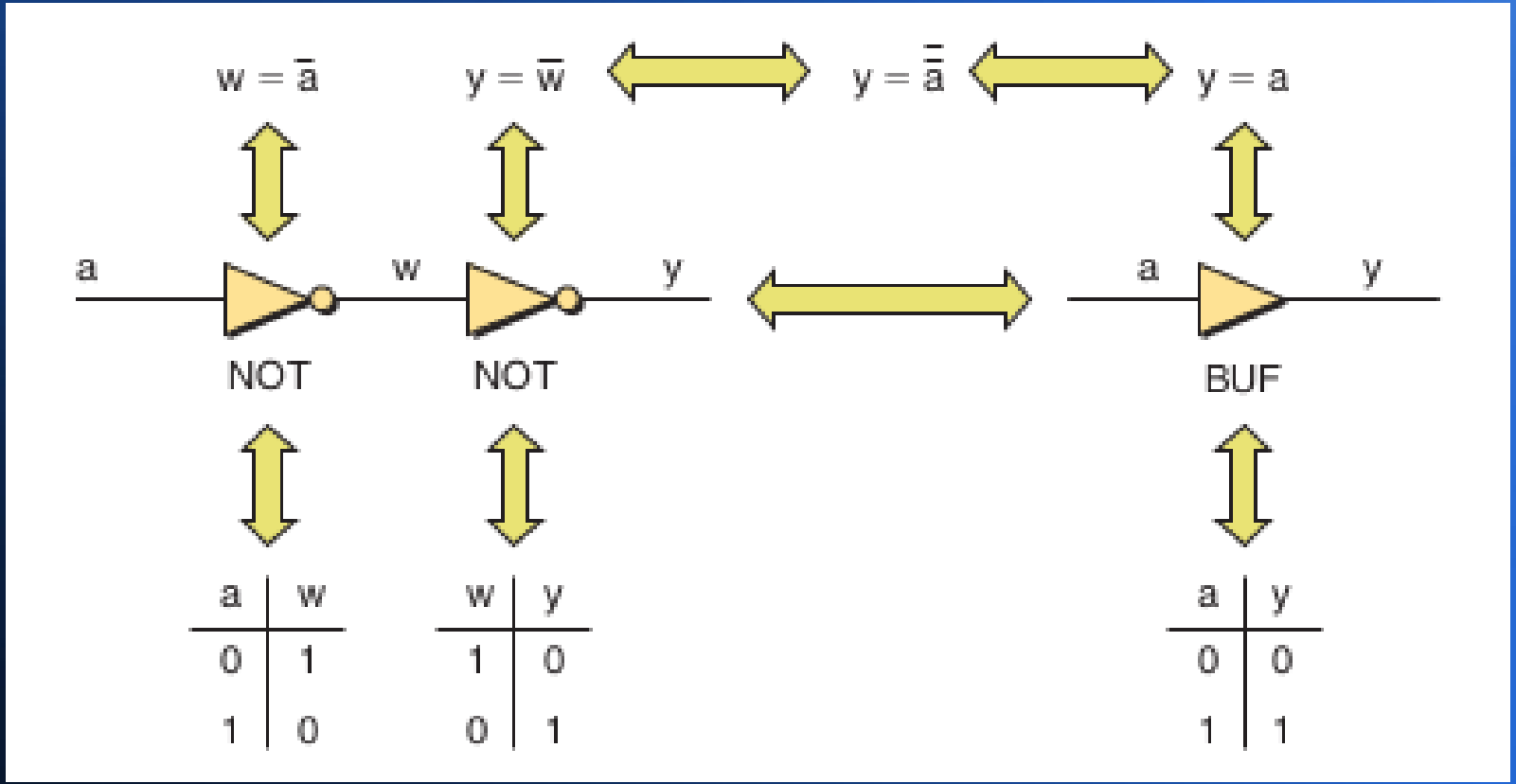


The idempotent rules derived from the combination of a single variable with itself.

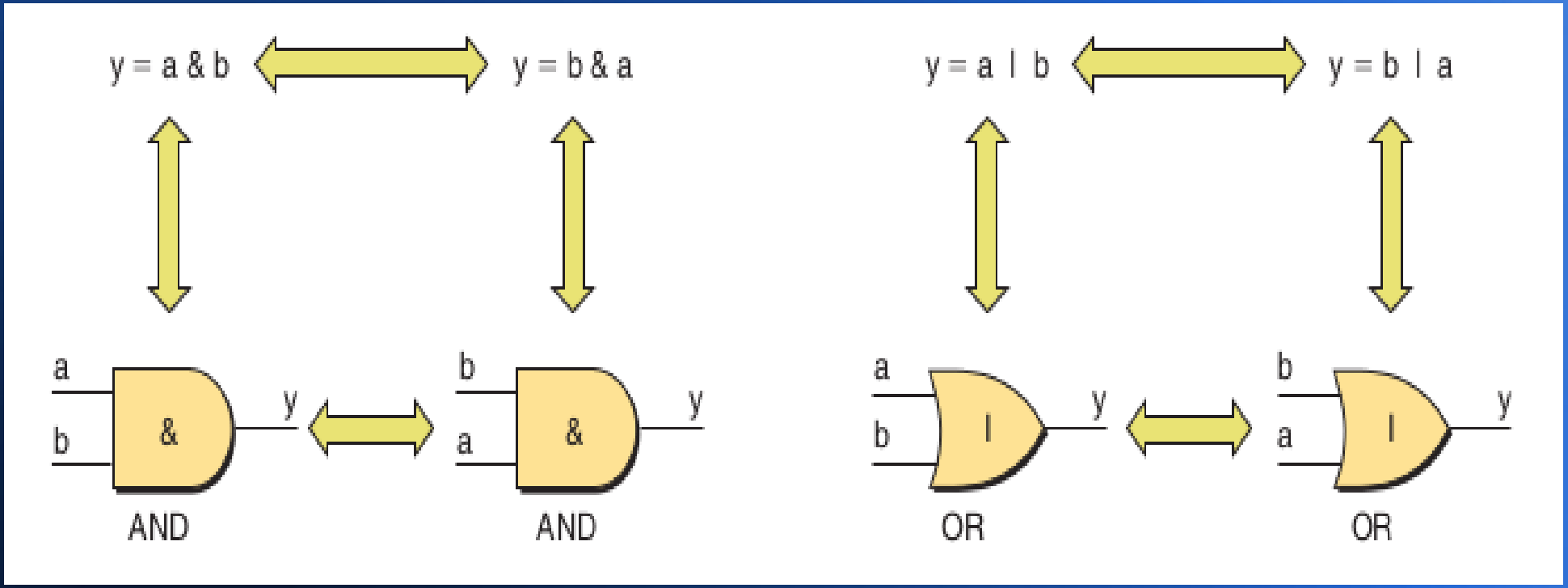
The rules derived from the combination of a single variable with the inverse of itself are known as the complementary rules



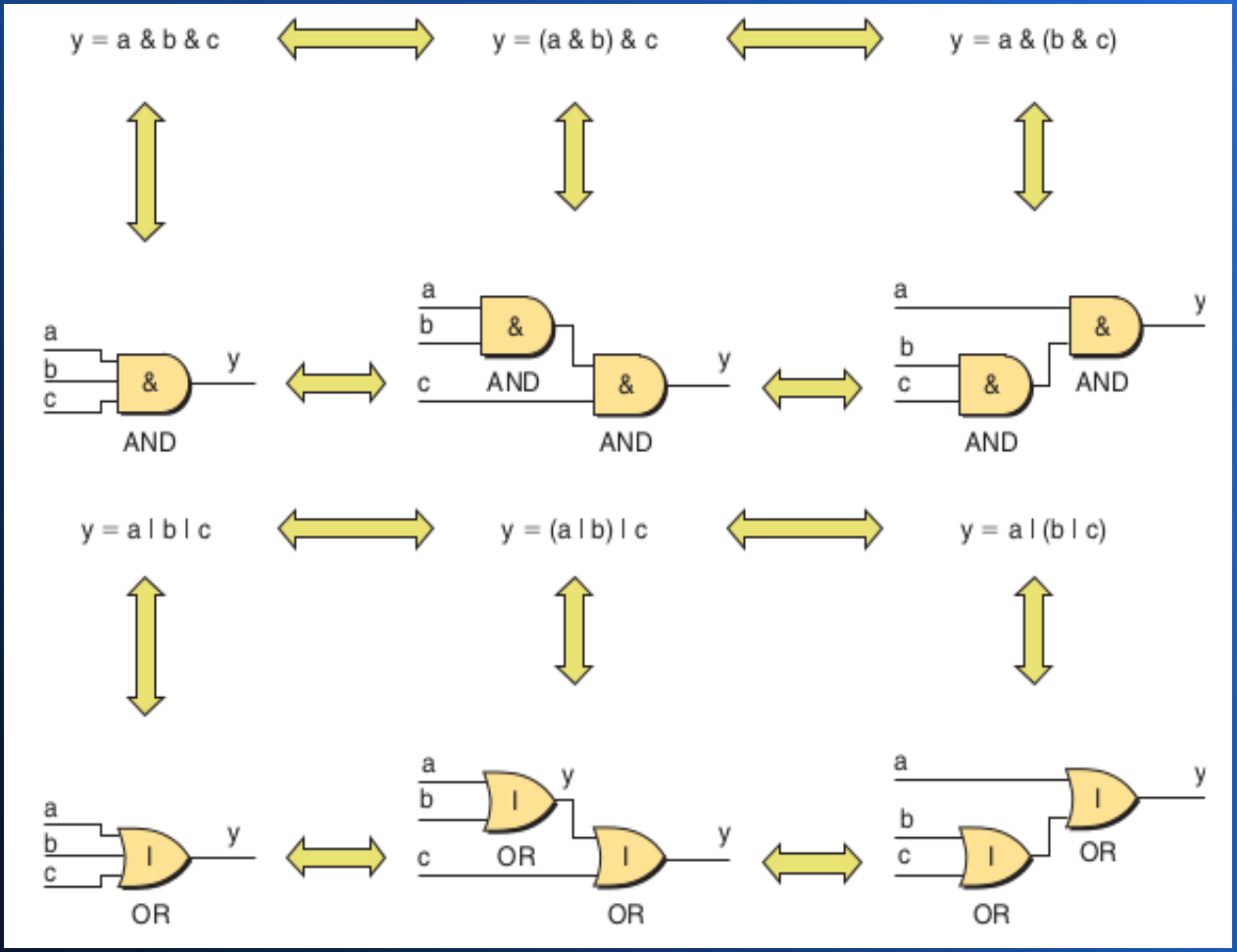
The involution rule states that an even number of inversions cancel each other out.



The commutative rules state that the order in which variables are specified will not affect the result of an AND or OR operation.



The associative rules state that the order in which pairs of variables are associated together will not affect the result of multiple AND or OR operations.



PRECEDENCE OF OPERATORS

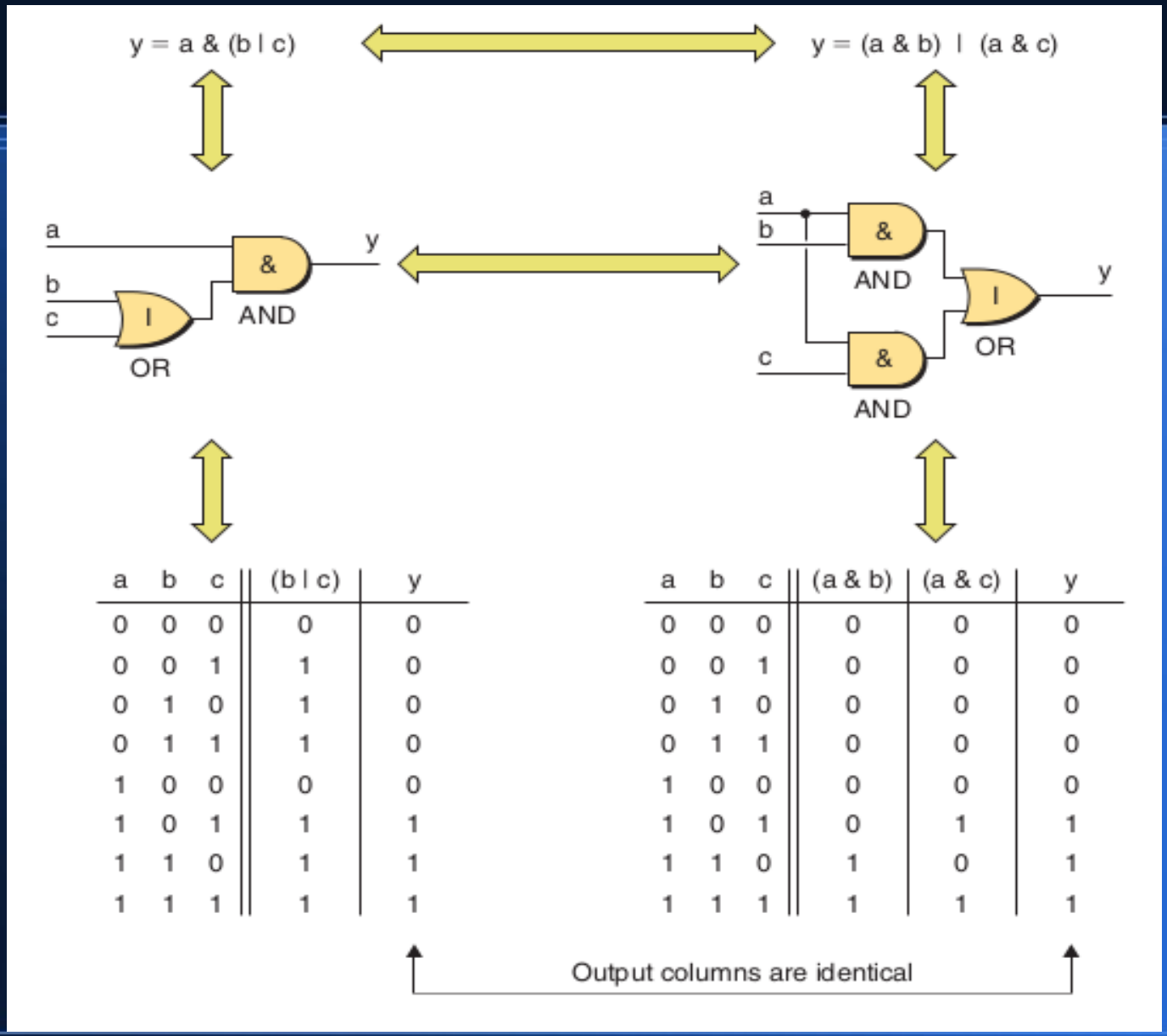
PRECEDENCE OF OPERATORS - in Boolean Algebra, the & (AND) operator has a higher precedence than the | (OR) Operator, like multiplication.

$$a \mid b \,\&\, c \equiv a \mid (b \,\&\, c)$$

Due to the similarities between these arithmetic and logical operators, the & (AND) operator is known as a logical multiplication or product, while the | (OR) operator is known as a logical addition or sum.

The first distributive rule – AND distributes over OR, like multiplication over addition.

$2 \times (1 + 3) =$
$(2 \times 1) + (2 \times 3)$

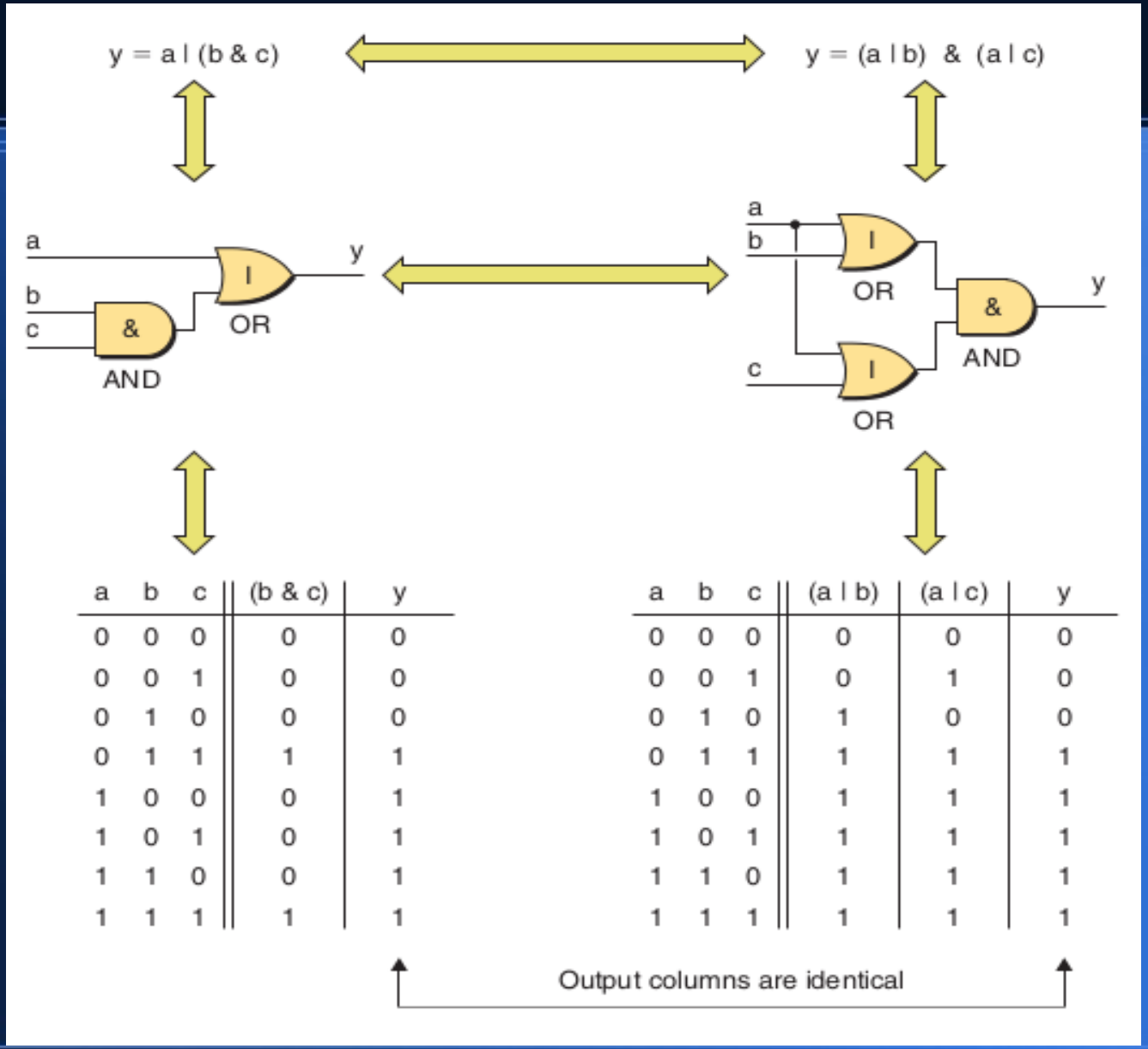


a	b	c	(b | c)	y
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

a	b	c	(a & b)	(a & c)	y
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

The second distributive rule – OR distributes over AND.

$2 + (1 \times 3) \neq (2 + 1) \times (2 + 3)$



a	b	c	(b & c)	y
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

a	b	c	(a | b)	(a | c)	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Proof of the second distributive law

$$(X + Y)(X + Z) = X(X + Z) + Y(X + Z) = XX + XZ + YX + YZ$$

$$= X + XZ + XY + YZ = X \cdot 1 + XZ + XY + YZ$$

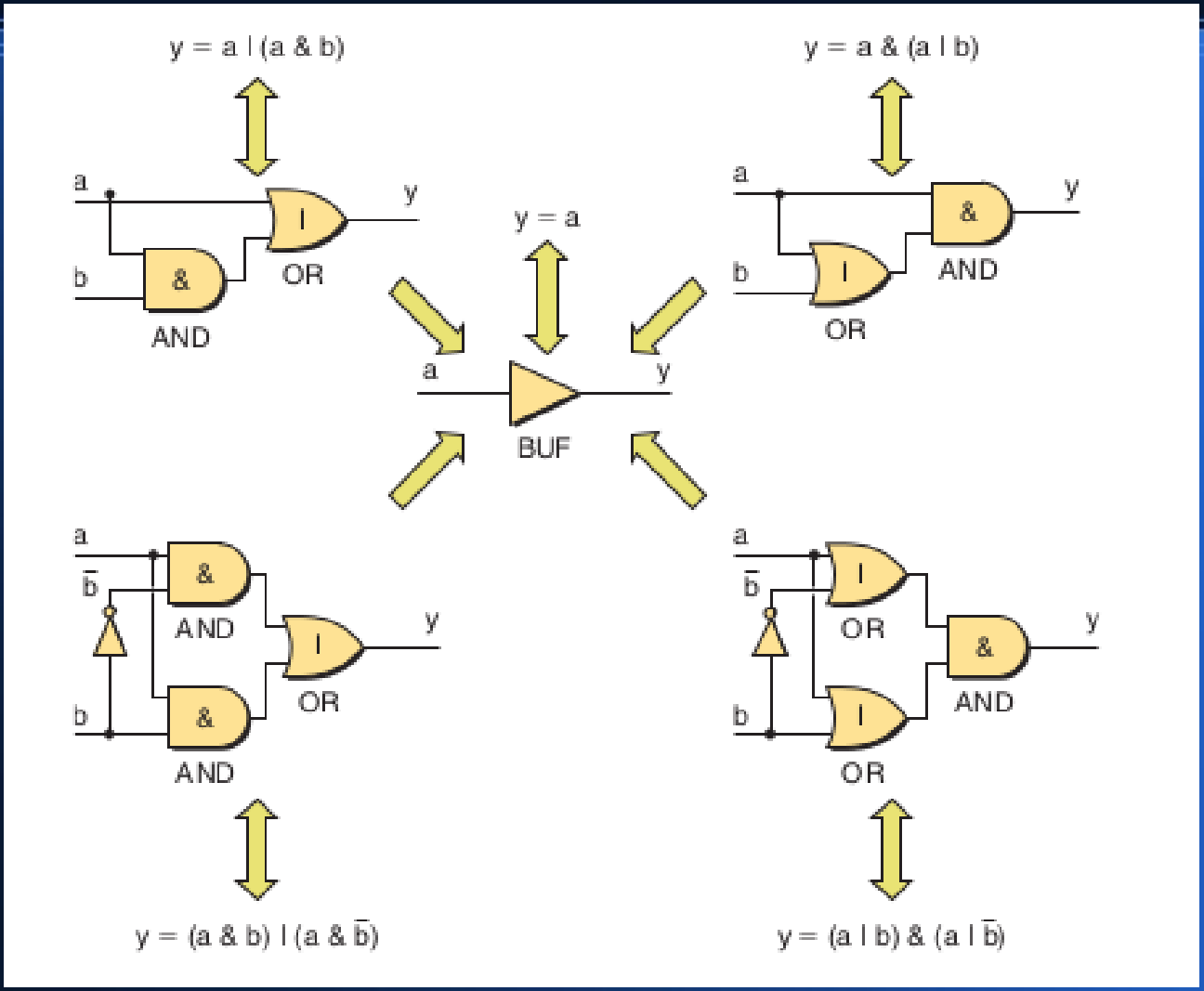
$$= X(1 + Z + Y) + YZ = X \cdot 1 + YZ = X + YZ$$

The ordinary distributive law states that the AND operation distributes over OR, while the second distributive law states that OR distributes over AND.

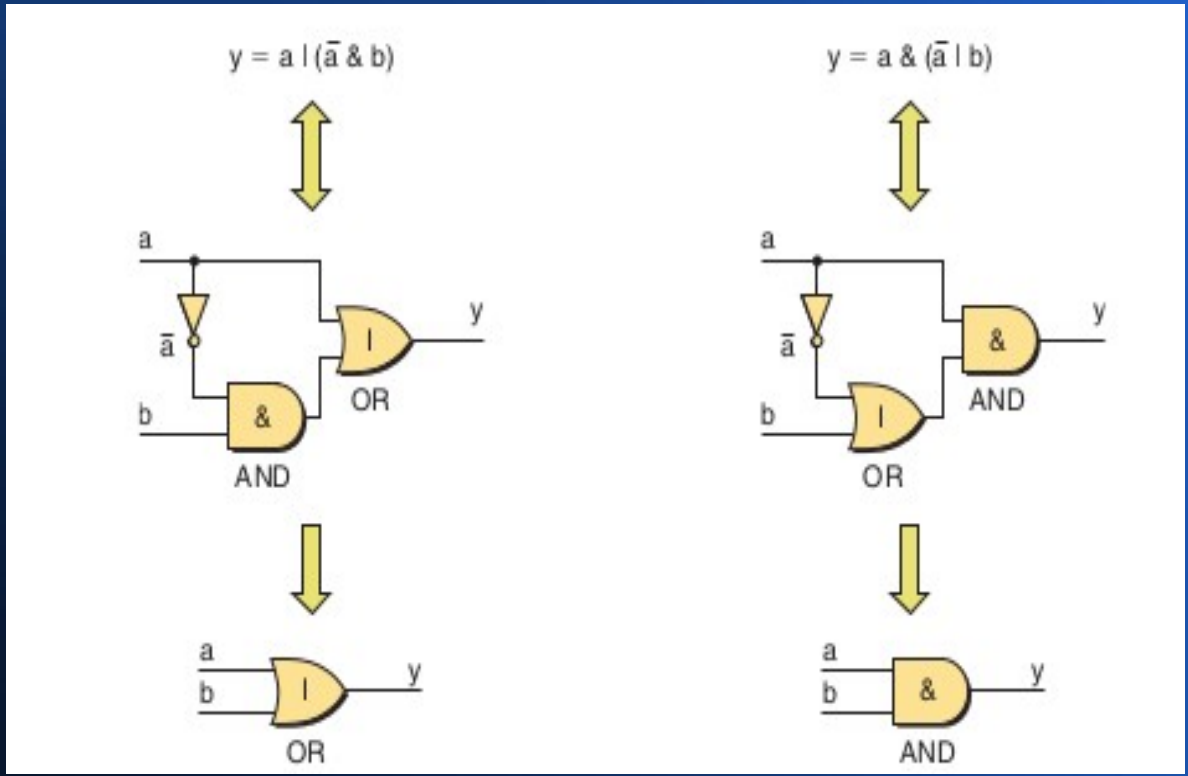
This second law is very useful in manipulating Boolean expressions. In particular, an expression like A + BC, which cannot be factored in ordinary algebra, is easily factored using the second distributive law

$A + BC = (A + B)(A + C)$

There are a number of simplification rules that can be used to reduce the complexity of Boolean expressions.



As the end result is to reduce the number of logic gates required to implement the expression, the process of simplification is also known as minimization.



Each of the Simplification theorems can be proved by using a truth table, or they can be proved algebraically starting with the basic theorems.

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

$$X(X + Y) = XX + XY = X + XY = X$$

Simplify $Z = A'BC + A'$

let $X = A'$ and $Y = BC$.

$$Z = X + XY = X = A'$$

SUMMARY – BASIC THEOREMS

The following basic laws and theorems of Boolean algebra involve only a single variable:

Operations with 0 and 1:

$$X + 0 = X \qquad X \cdot 1 = X$$

$$X + 1 = 1 \qquad X \cdot 0 = 0$$

Idempotent laws

$$X + X = X \qquad X \cdot X = X$$

Involution law

$$(X')' = X$$

Laws of complementarity

$$X + X' = 1 \qquad X \cdot X' = 0$$

Commutative

$$XY = YX \qquad X + Y = Y + X$$

Associative

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

Distributive

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

Simplification theorems

$$XY + XY' = X \qquad (X + Y)(X + Y') = X$$
$$X + XY = X \qquad X(X + Y) = X$$
$$(X + Y')Y = XY \qquad XY' + Y = X + Y$$

We define a literal to be a single variable within a term, in complemented or uncomplemented form.

Simplify the following Boolean functions to a minimum number of literals.

1. $x(x' + y) = xx' + xy = 0 + xy = xy.$
2. $x + x'y = (x + x')(x + y) = 1(x + y) = x + y.$
3. $(x + y)(x + y') = x + xy + xy' + yy' = x(1 + y + y') = x.$
4. $$\begin{aligned} xy + x'z + yz &= xy + x'z + yz(x + x') \\ &= xy + x'z + xyz + x'yz \\ &= xy(1 + z) + x'z(1 + y) \\ &= xy + x'z. \end{aligned}$$
5. $(x + y)(x' + z)(y + z) = (x + y)(x' + z)$

5 is duality principle of 4

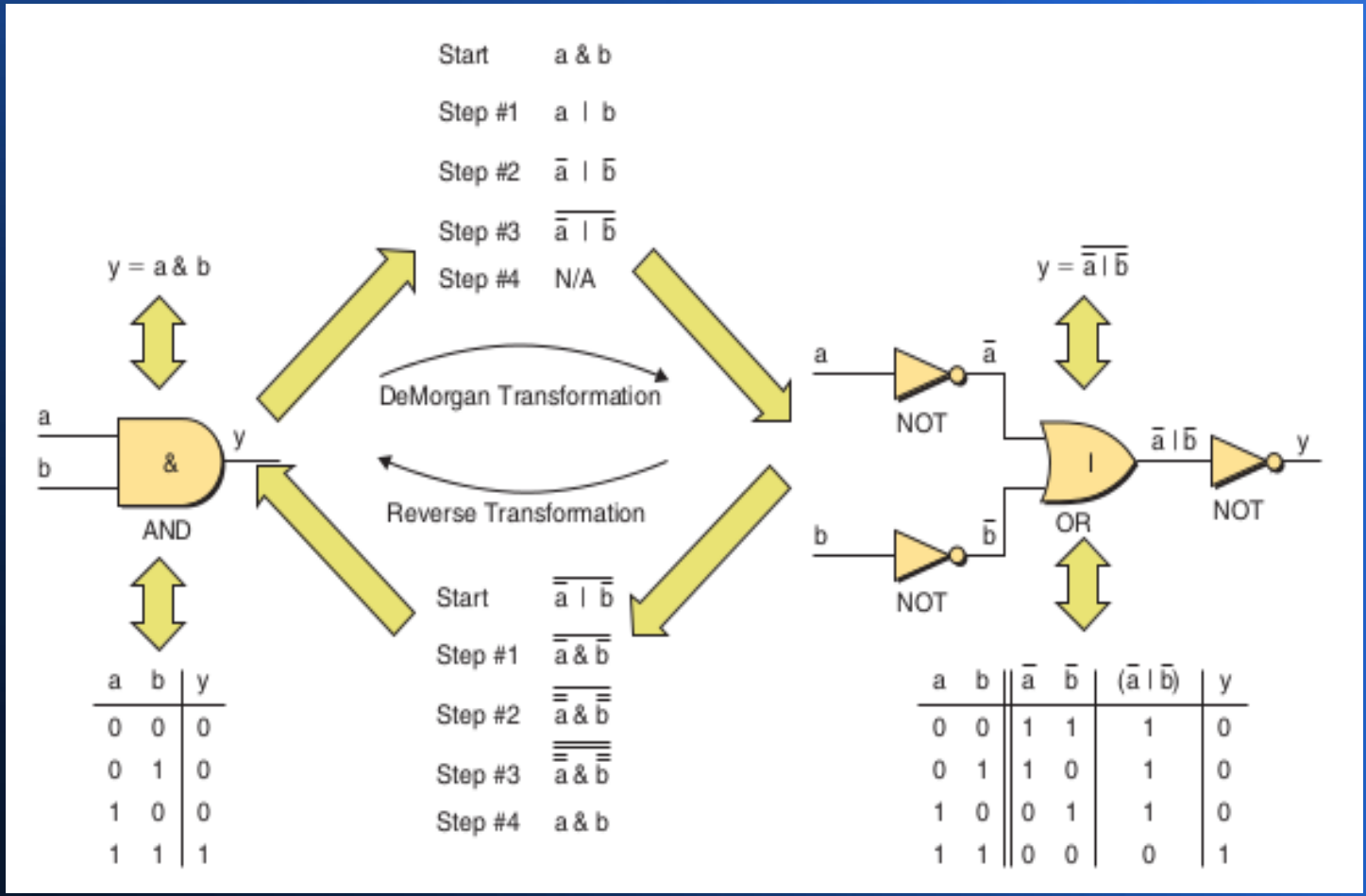
The rules we now attribute to DeMorgan were known in a more primitive form by William of Ockham in the 14th century. To celebrate Ockham's position in history, the OCCAM computer programming language was named in his honor.

William of Ockham (c. 1288 – c. 1348) was an English Franciscan friar and scholastic philosopher, who is believed to have been born in Ockham, a small village in Surrey. He is considered to be one of the major figures of medieval thought and was at the centre of the major intellectual and political controversies of the fourteenth century

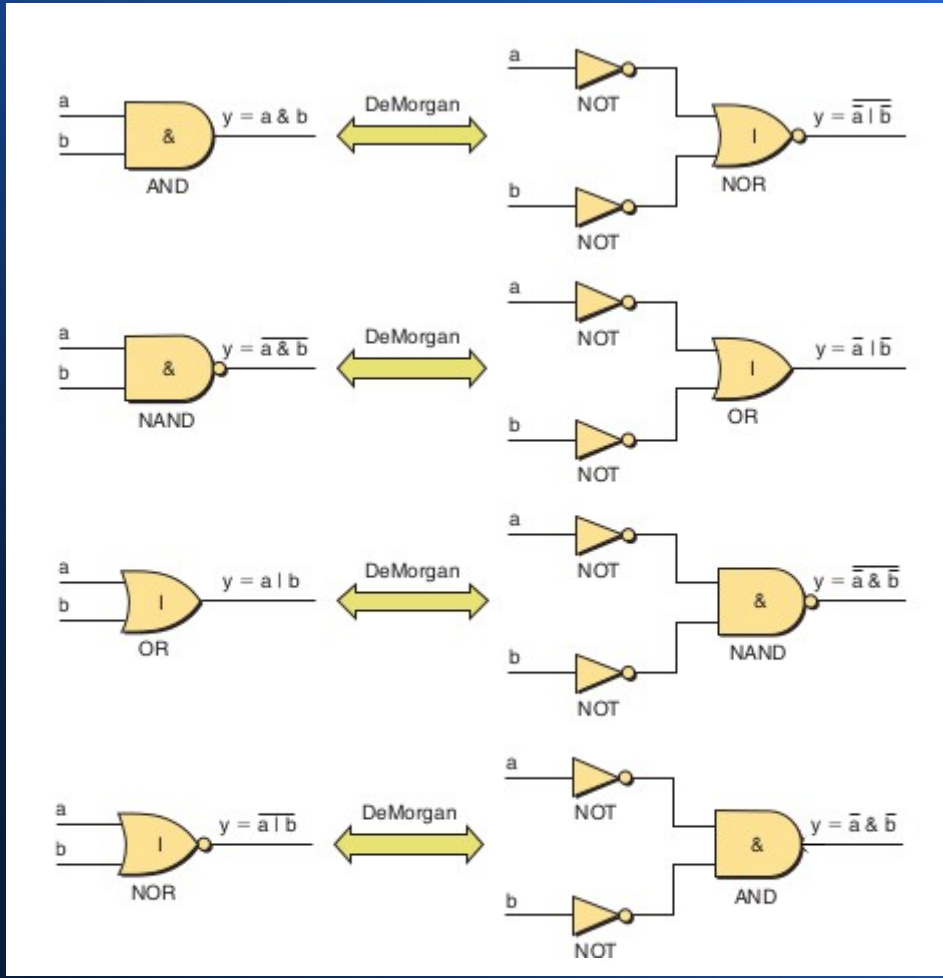
A DeMorgan Transformation comprises four steps:

1. Exchange all of the & (AND) operators for | (OR) operators and vice versa.
2. Invert all the variables; also exchange 0 s for 1 s and vice versa.
3. Invert the entire function.
4. Reduce any multiple inversions.

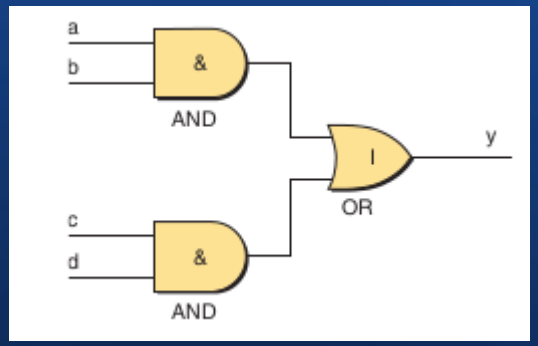
DeMorgan Transformation of an AND function

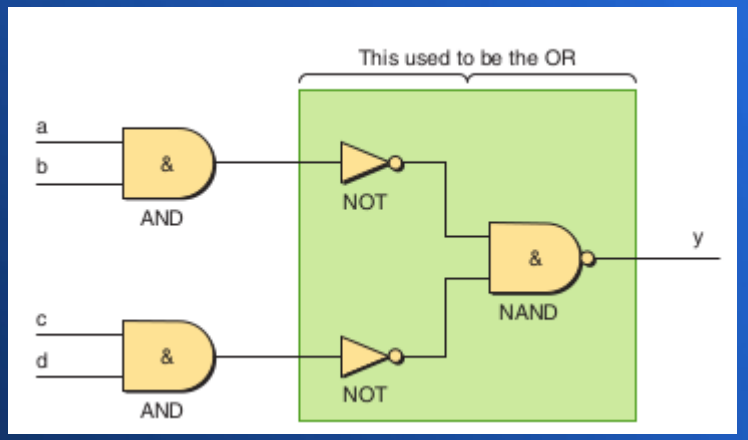


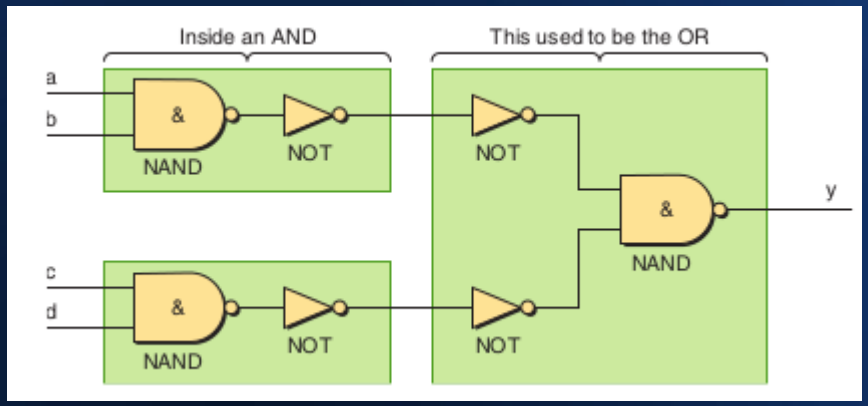
DeMorgan Transformations of AND, OR, NAND, and NOR functions.

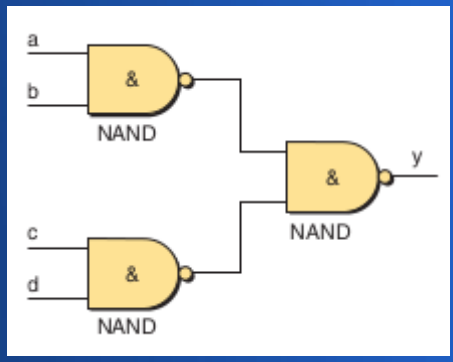


By means of a DeMorgan Transformation, we've taken our original circuit comprising two ANDs and an OR (with 18 transistors) and transformed it into a new version comprising three NANDs (with 14). This equates to around a 20% reduction in transistors.









DeMorgan's laws

$$(X + Y)' = X'\,Y'$$
$$(XY)' = X' + Y'$$

DeMorgan's laws are easily generalized to n variables:

$$(X_1 + X_2 + X_3 + \ldots + X_n)' = X_1'\,X_2'\,X_3'\ldots X_n'$$
$$(X_1X_2X_3\ldots X_n)' = X_1' + X_2' + X_3' + \ldots + X_n'$$

Referring to the OR operation as the logical sum and the AND operation as logical product, DeMorgan's laws can be stated as
The complement of the product is the sum of the complements.
The complement of the sum is the product of the complements

In the final expressions, the complement operation is applied only to single variables

$$
\begin{aligned}
[(AB' + C)D' + E]' &= [(AB' + C)D']'E' \\
&= [(AB' + C)' + D]E' \\
&= [(AB')'C' + D]E' \\
&= [(A' + B)C' + D]E'
\end{aligned}
$$

BOOLEAN ALGEBRA SUMMARY

Operations with 0 and 1:
1. $X + 0 = X$ 1D. $X \cdot 1 = X$
2. $X + 1 = 1$ 2D. $X \cdot 0 = 0$

Idempotent laws:
3. $X + X = X$ 3D. $X \cdot X = X$

Involution law:
4. $(X')' = X$

Laws of complementarity:
5. $X + X' = 1$ 5D. $X \cdot X' = 0$

Commutative laws:
6. $X + Y = Y + X$ 6D. $XY = YX$

Associative laws:
7. $(X + Y) + Z = X + (Y + Z) = X + Y + Z$ 7D. $(XY)Z = X(YZ) = XYZ$

Distributive laws:
8. $X(Y + Z) = XY + XZ$ 8D. $X + YZ = (X + Y)(X + Z)$

Simplification theorems:
9. $XY + XY' = X$ 9D. $(X + Y)(X + Y') = X$
10. $X + XY = X$ 10D. $X(X + Y) = X$
11. $(X + Y')Y = XY$ 11D. $XY' + Y = X + Y$

DeMorgan's laws:
12. $(X + Y + Z + \ldots)' = X'Y'Z'\ldots$
12D. $(XYZ\ldots)' = X' + Y' + Z' + \ldots$

The **duality principle** states that every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged.