

# OCR Computer Science AS Level

## 1.4.3 Boolean Algebra

### Advanced Notes



**Specification:**

**1.4.3 a)**

- Define problems using Boolean logic

**1.4.3 b)**

- Manipulate Boolean expressions
  - Karnaugh maps to simplify Boolean expressions





**1.4.3 c)**

- Use logic gate diagrams and truth tables



## Logic Gate Diagrams and Truth Tables

Problems can be defined using **Boolean logic** in **Boolean equations**. A Boolean equation can equate to either True or False, but not both. There are **four operations** we need to cover: conjunction, disjunction, negation and exclusive disjunction.

Operation	Conjunction	Disjunction	Negation	Exclusive Disjunction
Logic gate				
	AND	OR	NOT	XOR
Symbol	$\wedge$	$\vee$	$\neg$	$\underline{\vee}$

### Truth tables

A truth table is a table showing **every possible permutation** of inputs to a logic gate and the corresponding output. Inputs are usually labeled A, B, C etc.

For example, below is the truth table for an AND gate (conjunction) which is True (1) only when both inputs are True, otherwise the output is False (0).

#### Conjunction (AND)

A conjunction is applied to two **literals** (or inputs) to produce a single output. The truth table for an AND gate is shown to the right. Conjunction can be thought of as applying **multiplication** to its binary inputs. In terms of Boolean logic, the truth table represents the expression  $A \wedge B = Y$ .

#### AND

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

#### Disjunction (OR)

Like conjunction, disjunction operates on two literals and produces a single output. The truth table for an OR gate is shown to the right. Disjunction can be thought of as applying **addition** to its inputs, as long as one input is True then the output is True. The truth table shown above is equivalent to the boolean expression  $A \vee B = Y$ .

#### OR

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



### Negation (NOT)

In contrast to conjunction and disjunction, negation is only applied to **one literal**, and simply reverses the truth value of the input. For example, NOT 1 is 0. The truth table is the same as the expression  $\neg A = Y$ .

#### NOT

A	Y
0	1
1	0

### Exclusive Disjunction (XOR)

Also known as **exclusive OR**, hence XOR, exclusive disjunction is similar to disjunction but differs when both inputs are True. Exclusive disjunction only outputs True when **exactly** one input is True. Otherwise the output is False.

#### XOR

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

The truth table is the same as the expression  $A \vee B = Y$ .

## Combining Boolean Operations

Boolean equations are made by combining Boolean operators. This is done in the same way that standard mathematical operators are combined. For example, the Boolean equation  $\neg(A \wedge (B \vee C))$  can be used to describe a certain combination of the variables A, B and C. A truth table can be made for this equation by building up the parts one by one as follows:

A	B	C	$B \vee C$	$A \wedge (B \vee C)$	$\neg(A \wedge (B \vee C))$
0	0	0	0	0	1
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	1	0	1
1	0	0	0	0	1
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	1	0



## Manipulating Boolean Expressions

Sometimes a really long Boolean expression is identical to (has the **same truth table** as) another, shorter expression. It tends to be **desirable to use the shorter versions**, and there are a variety of methods which can be used to simplify them.

### Karnaugh Maps

A Karnaugh map can be used to simplify Boolean expressions. The tables are filled in corresponding to the expression's truth table, like so:

A	B	Y
0	0	w
0	1	x
1	0	y
1	1	z

	A	0	1
B	0	w	y
1	x	z	

A Karnaugh map can also be used for a truth table with **three or four variables**. It's important that the values in the columns are written using **Gray code**. That is, they can only ever differ by **one bit** between adjacent columns and rows, including wraparound (eg. from the last column to the first), as highlighted in the top line of the map below.

	AB	00	01	11	10
CD	00	0	0	1	1
	01	0	1	0	0
	11	1	0	1	0
	10	0	1	1	1



To simplify a Boolean expression, first write your truth table as a Karnaugh map. Then highlight all of the 1s in the map with a rectangle. The larger the rectangle you can highlight at once the better. Bear in mind that only groups of 1s with edges equal to a **power of 2** (1, 2 or 4 in a row) can be highlighted, and wraparound is included.

Take for example the Boolean expression:

$$Y = (\neg A \wedge \neg B \wedge \neg C) \vee (A \wedge \neg B \wedge C) \vee (A \wedge B \wedge \neg C) \vee (A \wedge \neg B \wedge \neg C) \vee (A \wedge \neg B \wedge C)$$

The truth table for this expression can then be converted to a Karnaugh map as shown below.

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

		AB					
		00	01	11	10		
C	0	0	1	1	0		
	1	0	1	0	0		

		AB					
		00	01	11	10		
C	0	0	1	1	0		
	1	0	1	0	0		

Once a Karnaugh map has been made, any 1s can be highlighted using **as few rectangles as possible**, in this case - two rectangles must be used. Overlapping is **good**.

Using the highlighting, our expression can now be **simplified**. From the first rectangle, we can see that C doesn't affect the output when A is false and B is true. From the second, we can see that A doesn't affect the output when B is true and C is false.

From these two highlighted portions, we can significantly reduce the original expression to simply:  $Y = (\neg A \wedge B) \vee (B \wedge \neg C)$

