OCR Computer Science A 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 the following rules to derive or simplify statements in Boolean algebra:
  - De Morgan's Laws
  - Distribution
  - Association
  - Commutation
  - Double negation

1.4.3 d)

- Using logic gate diagrams and truth tables

1.4.3 e)

- The logic associated with D type flip flops, half and full adders

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	$\wedge$	$\vee$	$\neg$	$\veebar$

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.

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

XOR

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

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*

B \ A	0	1
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.

CD \ AB	00	01	11	10
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.

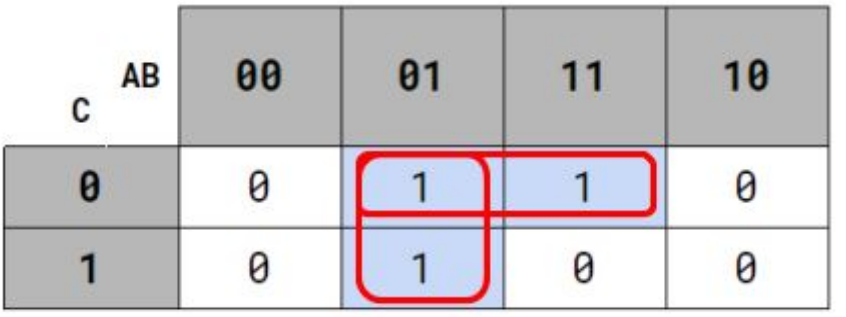
Take for example the Boolean expression:

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

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

C \ AB	00	01	11	10
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: $Y = (\neg A \wedge B) \vee (B \wedge \neg C)$

Simplifying Boolean Algebra

Just like algebra in mathematics, there are a variety of rules which can be used to simplify Boolean algebra.

De Morgan's Laws

De Morgan's laws involve breaking a negation and changing the operator between two literals.

$$\neg(A \wedge B) \equiv \neg A \vee \neg B$$
$$\neg(A \vee B) \equiv \neg A \wedge \neg B$$

They are used when a negation applies to the whole of an operator between two literals, and result in two negated literals acted upon by a different operator.

Conjunction is replaced by disjunction and vice versa.

Distribution

Another two useful laws, distribution applies to conjunction over disjunction as well as disjunction of conjunction.

Conjunction over disjunction:

$$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$$

Disjunction of conjunction:

$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$$

It's also important to note that distribution can be carried out over the same operator:

$$A \wedge (B \wedge C) \equiv (A \wedge B) \wedge (A \wedge C)$$
$$A \vee (B \vee C) \equiv (A \vee B) \vee (A \vee C)$$

Association

Associative laws involve the addition or removal of brackets and reordering of literals in a Boolean expression.

$$(A \wedge B) \wedge C \equiv A \wedge (B \wedge C) \equiv A \wedge B \wedge C$$
$$(A \vee B) \vee C \equiv A \vee (B \vee C) \equiv A \vee B \vee C$$

Commutation

Laws of commutation show that the order of literals around an operator does not matter.

$$A \vee B \equiv B \vee A$$
$$A \wedge B \equiv B \wedge A$$

Double Negation

If you negate a literal twice, you can remove both negations and retain the same truth value. `NOT NOT A` is the same as `A`.

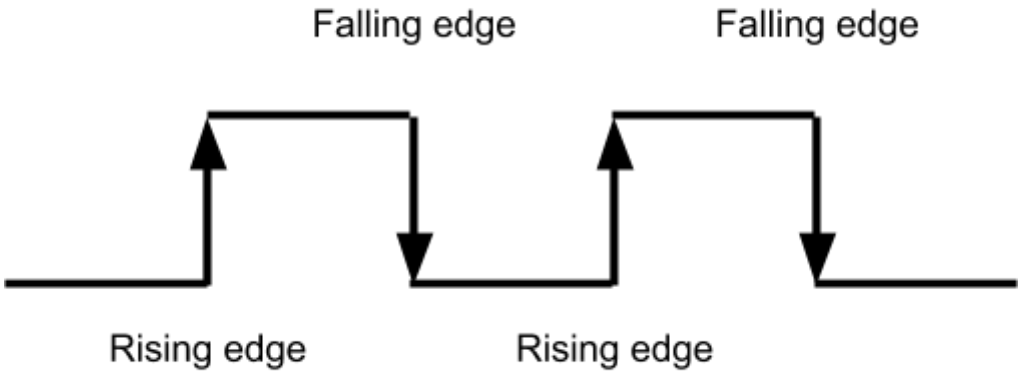
$$\neg\neg A \equiv A$$

Logic Circuits

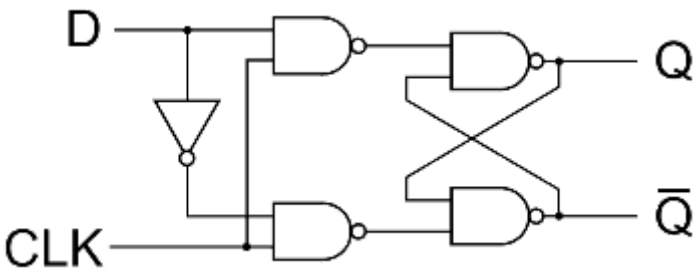
There are three circuits that you need to be familiar with. D-Type Flip Flops, Half Adders and Full Adders.

D-Type Flip Flops

A flip flop is a type of logic circuit which can store the value of one bit. A flip flop has two inputs, a control signal and a clock input. A clock is a regular pulse generated by the CPU which is used to coordinate the computer's components.



A clock pulse rises and falls as shown in the diagram, with edges labelled rising or falling. The output of a D-type flip flop can only change at a rising edge, the start of a clock tick.



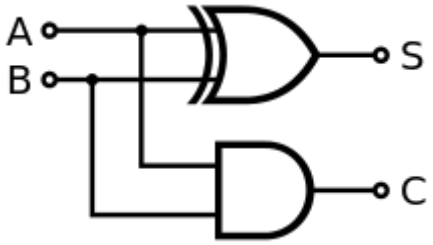
This is the logic circuit for a D-type flip flop. It uses four NAND gates and updates the value of Q to the value of D whenever the clock (CLK) ticks, on a rising edge. The value of Q is the stored value. Remember, a flip-flop is a basic unit of memory and sets of flip-flops can be combined to form registers.

Adders

An adder is a logic circuit which adds together the number of inputs which are true, and outputs that number in binary. There are two adder circuits you need to know: half and full.

Half Adder

A half adder has two inputs, A and B, and two outputs, Sum and Carry. The circuit is formed from just two logic gates: AND and XOR.

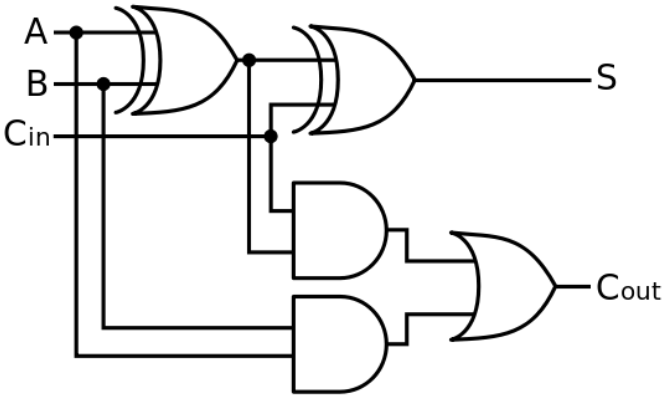


The corresponding truth table is below. When both A and B are false, both outputs are false. When one of A or B is true, Sum (S) is true and when both inputs are true, Carry (C) is true.

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Full Adder

A full adder is similar to a half adder, but has an additional input, allowing for a carry in to be represented. The full adder logic circuit is formed from two XOR gates, two AND gates and an OR gate as shown in the diagram.



The truth table for a full adder is shown below. There are eight rows as opposed to the four rows of the half adder due to the additional third input, C_{in}.

A	B	C_{in}	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Because the full adder has a carry input, the circuits can be chained together to form what's known as a ripple adder. At each stage, B and C_{in} can be connected to the previous adder's S and C_{out}, and a new input can be attached to A.