Lecture 1: An Introduction to Boolean Algebra

The operation of almost all modern digital computers is based on two-valued or binary systems. Binary systems were known in the ancient Chinese civilisation and by the classical Greek philosophers who created a well structured binary system, called propositional logic, in which propositions may be TRUE or FALSE, and are stated as functions of other propositions which are connected by the three basic logical connectives: AND, OR, and NOT. Hence the statement

IF *it is raining* OR *the weather forecast is bad* THEN *I will take an umbrella with me*

connects the proposition *I will take an umbrella with me* functionally to the two propositions *it is raining* and *the weather forecast is bad* . We can see that the 'umbrella' proposition can be considered as output while the other two as inputs and consequently a simple block diagram of this rule can be drawn (Diagram 1.1). The meaning of the OR connective is that the output is TRUE if either one of the input propositions is TRUE. Since there are only two possible values for any proposition, we can easily calculate a truth value for *I will take an umbrella* for all possible input conditions. This produces the Truth Table of the basic OR function as:

Raining	Bad Forecast	Umbrella
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

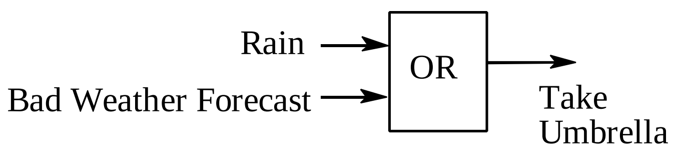

Diagram 1.1

We can make the propositons as complex as we require. For example, if we want to include the propostion *I will take the car*, we may make a statement such as: If I do not take the car then I will take the umbrella if it is raining or the weather forecast is bad. However, to find the correct block diagram and binary equations we have to state the proposition in a well structured way using brackets to indicate how the proposition is composed. The correct statement is:

(*Take Umbrella*) = (NOT (*Take Car*)) AND ((*Bad Forecast*) OR (*Raining*))

Notice that we have changed the IF verbal construction into an equation with binary variables. The block diagram is shown in Diagram 1.2.

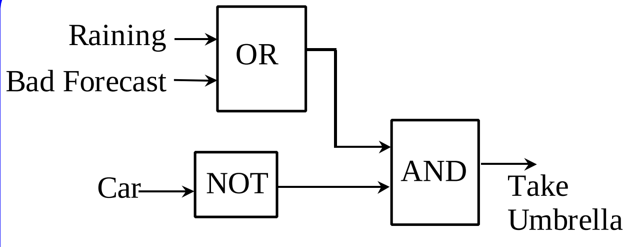

Diagram 1.2

To simplify the handling of complex binary connectives, the mathematician Boole developed Boolean Algebra in the last century, using ordinary algebraic notation, and 1 for TRUE and 0 for FALSE. In this course we will use the symbol • for the AND and + for the OR connectives or Boolean operators. The NOT, which is a unary operator, we will denote with a post fix prime, eg A' means NOT A. (Alternatives that you may see in books are $\wedge$ for AND, $\vee$ for OR, and either overscore or prefix $\neg$ for NOT) . Using the values 1 for TRUE and 0 for FALSE the truth tables of the three basic operators are given in Diagram 1.3.

AND
•

A	B	R
0	0	0
0	1	0
1	0	0
1	1	1

OR
+

A	B	R
0	0	0
0	1	1
1	0	1
1	1	1

NOT
'

A	R
0	1
1	0

Diagram 1.3: Truth Tables

The precedence of the operators is:

OPERATOR	SYMBOL	PRECEDENCE
NOT	'	Highest
AND	•	Middle
OR	+	Lowest

The Boolean equation of the above block diagram (1.2) in fully bracketed form is given by:

$U = ((C')\bullet((W)+(R))$

or, accepting the precedence rules, in simpler form it is:

$U = C' \bullet(W + R)$

We can use these basic truth tables to evaluate the overall truth table of a more complex expression. For example, to find out whether we should take our umbrella or not we can evaluate the overall truth of the proposition using a table as shown in Diagram 1.4. We shall call this the Truth Table Method. Note that, in this case, there are eight possible different combinations of input values since there are three independent inputs and $8 = 2^3$.

R	W	C	X1=R+W	X2=C'	U=X1•X2
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	0	0
Inputs			Partial Results		Output

Diagram 1.4: The Umbrella Truth Table

Once one has defined a notation for an algebra, the rules to manipulate expressions follow. The most simple are the rules that concern the unary operator NOT:

$(A')' = A$

$A \bullet A' = 0$

$A + A' = 1$

General rules like the distributive, commutative, and associative rules hold for the AND and OR binary operators (except one weird one) so that:

Distributive:	$A\bullet(B+C) = A\bullet B + A\bullet C$	
	$A+(B\bullet C) = (A+B)\bullet(A+C)$	(the weird one!)
Commutative:	$A\bullet B = B\bullet A$	
	$A+B = B+A$	
Associative:	$(A\bullet B)\bullet C = A\bullet(B\bullet C)$	
	$(A+B)+C = A+(B+C)$	

In addition, there are simplification rules for Boolean equations. There are three important groups of simplification rules. The first one uses just one variable:

$A\bullet A = A$

$A+A = A$

The second group uses Boolean constants 0 and 1:

$A \bullet 0 = 0$

$A \bullet 1 = A$

$A + 0 = A$

$A + 1 = 1$

The third group involves two or more variables and contains a large number of possible simplification rules (or theorems) such as:

$A + A\bullet(B) = A$ (proof: $A + A\bullet B = A\bullet(1+B) = A\bullet 1 = A$)

Note that in this expression either A or B may stand for any complex Boolean expression.

There are two important rules which constitute de Morgan's theorem:

$(A+B)' = A' \bullet B'$

$(A\bullet B)' = A' + B'$

This theorem is widely used in Boolean logic design. Stated in words it is: *To "invert" (negate) a Boolean expression, you replace the AND operator with the OR operator (or vice versa) and invert the individual terms*. The theorem holds for any number of terms, so:

$(A+B+C)' = ((A+B)+C)' = ((A+B)')\bullet C' = A'\bullet B'\bullet C'$

and similarly:

$(A\bullet B\bullet C\bullet \ldots\bullet X)' = A' + B' + C' + \ldots + X'$

You may have noticed by now that rules are often given in pairs. It makes sense that in a binary system there is some kind of symmetry between the two operators. For Boolean algebra this symmetry is called Duality. Every equation has its dual which one can generate by replacing the AND operators with ORs (and vice versa) and the constants 0 with 1s (and vice versa).

For example, the dual equation of the important simplifying rule:

$A + A\bullet B = A$

is:

$A\bullet(A+B) = A$ (proof: $A\bullet A + A\bullet B = A + A\bullet B = A$)

Do not mix up or get confused between a dual expression which is generated by the above rules and the complement (or inverted) expression which is generated by applying the NOT operator. The rules are similar, but they mean very different things.

Finally, let us consider the proposition (I am not taking an umbrella), or:

$(U)' = (C'\bullet(W+R))'$

Apply de Morgan's theorem

$U' = (C')' + (W+R)'$

Apply de Morgan's theorem again

$U' = (C')' + W'\bullet R'$

And simplify $U' = C + W'\bullet R'$