

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. 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. For example the statement:

“I will take an umbrella with me if it is raining or the weather forecast is bad”

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 fully determined by the *raining* and *weather* ones. In functional terms we can consider the truth value of the *umbrella* proposition as the output of the truth values of the other two. We can represent this by means of a simple block diagram (Figure 1).

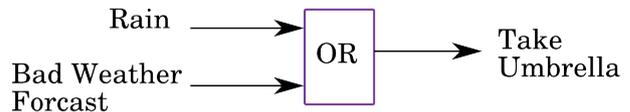


Figure 1: A simple Proposition

The meaning of the OR connective is that the corresponding output is TRUE if either one of the input propositions is TRUE, otherwise it is FALSE. 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:

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

We can make the propositions as complex as we require. For example, if we want to include the proposition *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 we have to state the proposition in a well structured way using brackets to indicate how the proposition is composed. The correct representation 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 Figure 2. To simplify the handling of complex binary connectives, the mathematician George 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 \cdot for the AND and $+$ for the OR connectives which we call Boolean operators. The NOT operator, which is unary, we will denote with a post fix prime, eg A' means NOT A.

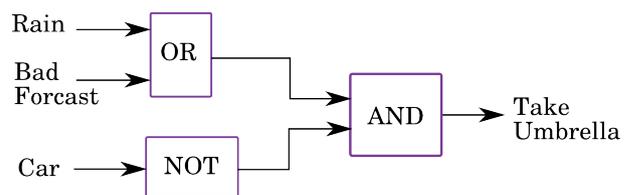


Figure 2: Another Proposition

(Alternatives that you may see in books are \wedge for AND, \vee for OR, and either over-score or prefix \neg for NOT). Sometimes, when the meaning is clear from the context, we may omit the AND symbol. Using the values 1 for TRUE and 0 for FALSE the truth tables of the three basic operators are as follows.

AND \cdot		
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

Boolean operations are carried out in a well defined order or “precedence”, which is defined as follows:

Operator	Symbol	Precedence
NOT	'	Highest
AND	·	Middle
OR	+	Lowest

Expressions inside brackets are always evaluated first, overriding the precedence order. The Boolean equation of the block diagram (Figure 2) in fully bracketed form is given by:

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

By taking advantage of the precedence rules, we can simplify it by removing brackets:

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

We can use the basic truth tables for AND, OR and NOT 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 given in the above equation for every possible input combination. We shall call this the Truth Table Method. 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	$X_1 = R + W$	$X_2 = C'$	$U = X_1 \cdot X_2$
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

Like all algebras, there are rules to manipulate Boolean expressions. The most simple are the rules that concern the unary operator NOT:

$$\begin{aligned} (A')' &= A \\ A \cdot A' &= 0 \\ A + A' &= 1 \end{aligned}$$

General rules like the distributive, commutative, and associative rules hold for the AND and OR binary operators as follows.

$$\begin{aligned} \text{Associative} \quad & (A \cdot B) \cdot C = A \cdot (B \cdot C) \\ & (A + B) + C = A + (B + C) \\ \text{Commutative} \quad & A \cdot B = B \cdot A \\ & A + B = B + A \\ \text{Distributive} \quad & A \cdot (B + C) = A \cdot B + A \cdot C \\ & A + (B \cdot C) = (A + B) \cdot (A + C) \quad (\text{the weird one!}) \end{aligned}$$

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

$$\begin{aligned} A \cdot A &= A \\ A + A &= A \end{aligned}$$

The second group uses Boolean constants 0 and 1:

$$\begin{aligned} A \cdot 0 &= 0 \\ A \cdot 1 &= A \\ A + 0 &= A \\ A + 1 &= 1 \end{aligned}$$

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

$$A + A \cdot (B) = A \quad (\text{proof: } A + A \cdot B = A \cdot (1 + B) = A \cdot 1 = A)$$

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:

$$\begin{aligned} (A + B)' &= A' \cdot B' \\ (A \cdot B)' &= A' + B' \end{aligned}$$

This theorem is widely used in Boolean logic design. The theorem holds for any number of terms, so:

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

and similarly:

$$(A \cdot B \cdot C \cdot \dots \cdot X)' = A' + B' + C' + \dots + X'$$

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:

$$A + A \cdot B = A$$

is:

$$A \cdot (A + B) = A \quad (\text{proof: } A \cdot A + A \cdot B = A + A \cdot B = A)$$

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 simplify the proposition *I am not taking an umbrella*.

$$\begin{aligned} & (U)' = (C' \cdot (W + R))' \\ \text{apply de Morgan's theorem} & \quad U' = (C')' + (W + R)' \\ \text{apply de Morgan's theorem again} & \quad U' = (C')' + W' \cdot R' \\ \text{and simplify} & \quad U' = C + W' \cdot R' \end{aligned}$$