CHAPTER 1

BOOLEAN ALGEBRA

CONTENTS

1 INTRODUCTION

1.1 Boolean Algebra is a mathematical system for manipulating propositions, where a proposition is any statement which is either true or false. It was devised as a technique for undertaking logical analysis. It is also suitable for analysing other situations or systems which comprise elements which can be in only one of two possible states. It is therefore used in computing (where binary arithmetic is used) and in the analysis of switching circuits.

1.2 Boolean Algebra provides a formal procedure for reducing a complex system of logical dependencies to its simplest form.

2 NOTATION

2.1 Upper case letters denote statements, e.g. 'A' could mean that a particular piece of hardware is functioning (up). $\overline{A}$ denotes the converse of statement A, e.g. the item is failed (down).

2.2 Several symbols are used to denote mathematical functions that are performed on such statements. Note that that many programmable systems (specifically those using weakly typed* languages) use numbers to represent Boolean states which conflict with the standard presented here. 0 is often used for true and -1 for false although any other value is also treated as false.

Traditional symbol	Simple symbol	Definition	Example / comment
□	+	The **union** or '**or**' operator provides a result of true when either or both statements linked by the operator are true	A □ B = A + B = A or B
□	.	The **intersect** or '**and**' operator provides a result of true when, and only when, both statements linked by the operator are true	A □ B = A . B = A and B
true	1	**Truth** or, more colloquially, 'certainty'	$A + \overline{A} = 1$ it is certain that A is true or that A is false.
false	0	**Falsity** or, more colloquially, 'impossibility'	$A . \overline{A} = 0$ it is impossible that A is true and A is false at the same time. An item cannot be 'up' and 'down' at the same time.

Table 1: Notation used in Boolean Algebra

* Weekly typed languages are programming languages which do not strictly enforce the consistency of variable type usage. For example a variable may be set to an inter value and later used as if it were a Boolean variable (True or False). Strongly typed languages detect this as an error during completion.

3 AXIOMS AND THEOREMS

3.1 Simple combinations

$$0 + 0 = 0 \quad \text{(i)}$$
$$0 \cdot 0 = 0 \quad \text{(ii)}$$
$$0 + 1 = 1 \quad \text{(iii)}$$
$$0 \cdot 1 = 0 \quad \text{(iv)}$$
$$1 + 0 = 1 \quad \text{(v)}$$
$$1 \cdot 0 = 0 \quad \text{(vi)}$$
$$1 + 1 = A \quad \text{(vii)}$$
$$1 \cdot 1 = 1 \quad \text{(viii)}$$
$$A + 0 = A \quad \text{(ix)}$$
$$A \cdot 0 = 0 \quad \text{(x)}$$
$$A + 1 = 1 \quad \text{(xi)}$$
$$A \cdot 1 = A \quad \text{(xii)}$$
$$A + \overline{A} = 1 \quad \text{(xiii)}$$
$$A \cdot \overline{A} = 0 \quad \text{(xiv)}$$

3.2 Associativity and order

$$A + B = B + A \quad \text{(xv)}$$
$$A \cdot B = B \cdot A \quad \text{(xvi)}$$
$$A + (B \cdot C) = (A + B) \cdot (A + C) \quad \text{(xvii)}$$
$$(A + B) \cdot C = (A \cdot C) + (B \cdot C) \quad \text{(xviii)}$$

3.3 Combinations

$$\overline{(\overline{A})} = A \quad \text{(xix)}$$
$$A + (A + B) = A + B \quad \text{(xx)}$$
$$A \cdot (A + B) = A \quad \text{(xxi)}$$
$$A + (A \cdot B) = A \quad \text{(xxii)}$$
$$A.(A \cdot B) = A \cdot B \quad \text{(xxiii)}$$
$$A \cdot B = \overline{(\overline{A} + \overline{B})} \quad \text{(xxiv)}$$
$$\overline{A}.\overline{B} = \overline{A + B} \quad \text{(xxv)}$$
$$A + B = \overline{(\overline{A}.\overline{B})} \quad \text{(xxvi)}$$

$$\overline{A} + \overline{B} = \overline{A.B} \qquad \text{(xxvii)}$$

$$\text{If } A = B, \quad \text{then } A + C = B + C \text{ and } A \,.\, C = B \,.\, C \qquad \text{(xxvi)}$$

3.4 Complementary Principle

The complementary principle (also known as DeMoivre's Theorem) is expressed in the theorems xxiv to xxvii above.

If $S = f(A, \overline{A}, B, \overline{B}, C, \text{etc ...}, 0, 1)$

then $\overline{S} = g(\overline{A}, A, \overline{B}, B, \overline{C}, \text{etc ...}, 1, 0)$

where the function g is obtained from the function f by negating all individual statements (i.e. $X \rightarrow \overline{X}$, $\overline{X} \rightarrow X$), by changing '+' to '.' and '.' to '+',. and by changing 0 to 1 and 1 to 0.

e.g. if $S = (A + B) \,.\, (B.\{C + D\})$

then $\overline{S} = (\overline{A} \,.\, \overline{B}) + (\overline{B} + \{\overline{C} \,.\, \overline{D}\})$

3.5 Operator precedence

3.5.1 As with the arithmetic of numbers there is a precedence of operators with respect to the order of evaluation. This is:

1. the contents of in brackets;
2. the intersect ('and') operator is evaluated;
3. the union ('or') operator.

3.5.2 Brackets should be used to avoid ambiguity. For example, do not write A+B . C but instead write either (A + B).C or A+(B . C) whichever is appropriate. This is particularly important when applying the complementary principle.

3.6 Differences from numeric algebra

3.6.1 Note that the above theorems are not all equivalent to 'ordinary' algebra, and in particular *the converse of (xvi) is not true,*

i.e. it is NOT necessarily true that if A + B = A+C then B = C

NOR is it necessarily true that if A . B = A . C then B = C

However, it *is* true that if both A + B = A + C

and A . B = A . C

then B = C

Note also that in the above notation the plus sign (OR) indicates an inclusive OR, i.e. A + B means A or B *or both*. An exclusive OR, i.e. A OR B but *not* both, is written $(A.\overline{B})+(B.\overline{A})$. That is, (A AND not B) OR (B AND not A).

4 SET THEORY

4.1 Although the theorems in Section 3 are not proved in this Manual, their truth can be determined by the reader if it is accepted that Boolean Algebra may be represented by the methods of Set Theory given below. For the purposes of this leaflet a 'set' may be regarded as a collection of points on a plane. The propositions of Boolean Algebra can then be represented pictorially by what are known as Venn Diagrams, as explained below.

4.2 '1' in Boolean Algebra is equivalent in Set Theory to the Universal set or the set of points on an infinite plane. It is represented in the Figures below by a rectangle. All other sets lie within this set. Propositions A, B, C, etc., are equivalent in Set Theory to regions within closed curves, and $\overline{A}$, $\overline{B}$, $\overline{C}$, etc., are the complement of these with respect to the universal set (see Figure 1).

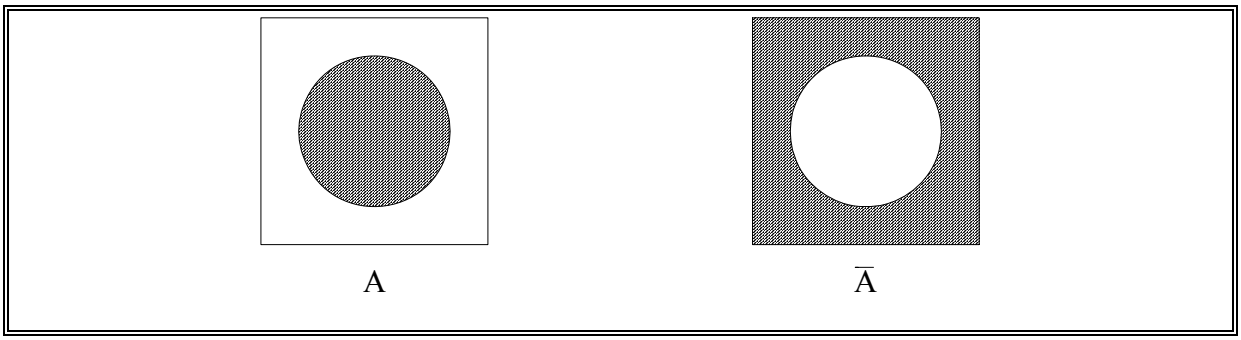


Figure 1: Venn diagrams for A and $\overline{A}$

4.3 A + B is equivalent to the region covered by A or B, i.e. the 'union' of A and B. A . B is equivalent to the overlapping region or intersection of A and B (see Figure 2).

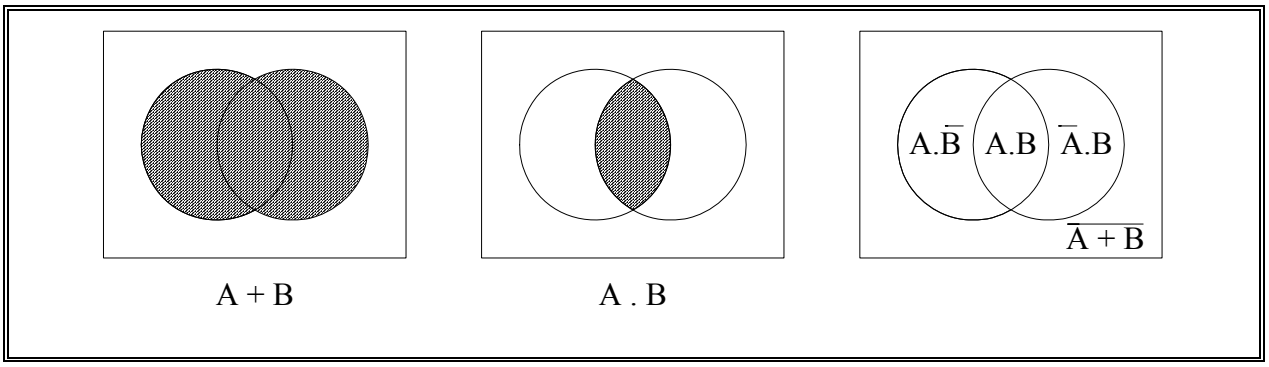


Figure 2: Venn Diagrams for A + B, A . B and $\overline{A + B}$

4.4 Thus, for example, the 'proof' of theorem ix above is as follows:

Left Hand Side = A+(B . C)

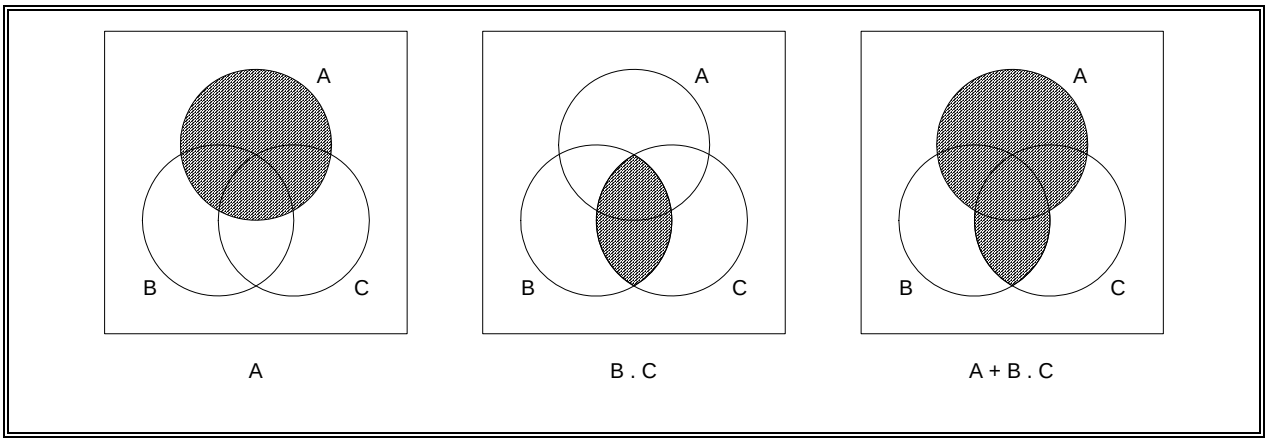


Figure 3: Venn Diagrams for A+(B . C)

Right Hand Side = (A + B) . (A + C)

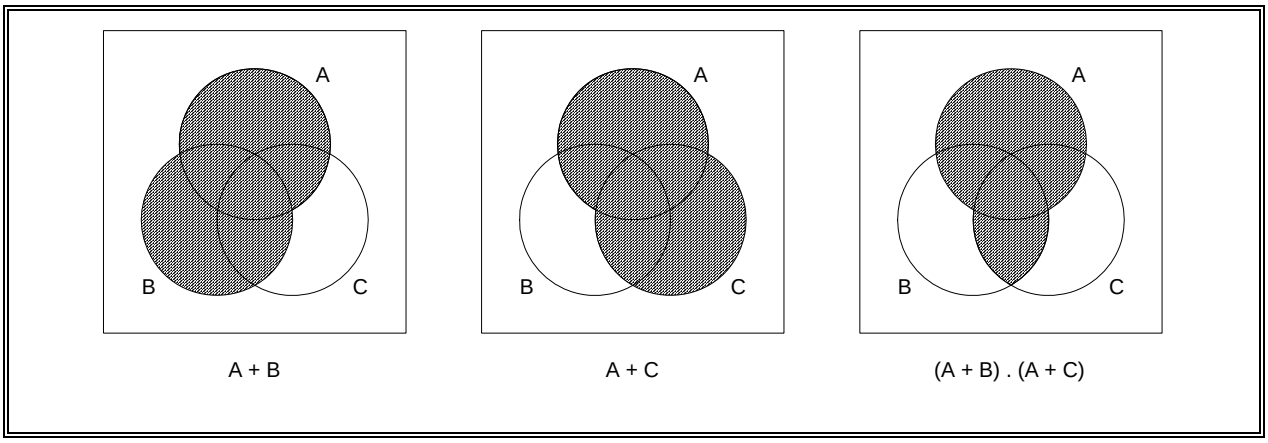


Figure 4: Venn Diagrams for (A + B) . (A + C)

The final diagrams of Figures 3 and 4 are identical, 'proving' the equality of the two expressions beneath them and hence theorem ix.

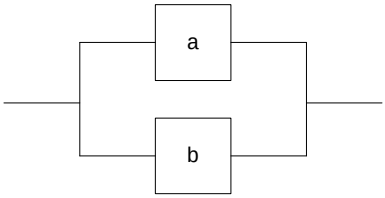
5 APPLICATION

5.1 Reliability Block Diagrams (RBDs)

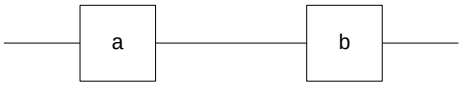
5.1.1 Often it is not easy to determine by inspection or common sense consideration of a system just what is its RBD, and further to determine the simplest RBD. Even when one produces such a diagram one cannot be certain that it covers all eventualities simply by inspecting it, except for the simplest systems. A surer method is to employ Boolean Algebra.

5.1.2 The method of attack is to use the techniques described below to produce a Boolean expression for the system up (or down) states; then reduce this expression using the theorems of section 3 to the simplest form for the system up state; then convert this to an RBD. In doing this conversion it must be remembered that '+' (OR) indicates redundancy, and '.' (AND) indicates elements in series.

Thus A + B is



A . B is



where lower case letters represent the hardware associated with statements A, B, etc.

5.1.3 In general, if there are fewer down states than up states, it will be simpler to attack the problem by analysing the down states, then apply the complementary principle (para 3.3).

5.1.4 The problem of producing an RBD may be posed generally in one of two forms. Either:

- There will be a clear statement of the conditions for the system to be considered up (or down); or
- There will be no such statement, and the task might be, for example, to produce the RBD for a given piece of hardware.

5.1.5 For the first form, the Boolean expression for the system up (or down) state can be obtained directly from the statement of the problem, by expressing it in symbolic form.

5.1.6 For the latter form, the recommended method is to set up a truth table for the system (see Pt3Ch38), identify the system up (or down) states in the truth table (in consultation with designers and/or users), and thereby obtain an expression for the system up or down state. For an n element system there are 2^n lines or system states in the truth table, a number which rapidly becomes large as n increases. However, this problem may be alleviated if the system is broken down into independent groups of elements, and each group is analysed separately. Also, it will be evident from the following examples that the task is not necessarily as arduous as it appears at first sight.

5.2 Fault / Success Tree Analysis

5.2.1 The analysis techniques of Fault Tree Analysis (FTA) and Success Tree Analysis (STA) are based on the logical combination of events and states. For example: the event of *Man under ladder is hit by paint pot* is a combination of the state *Man is under ladder* **and** the event *Paint pot drops from top of ladder*. This latter event is a combination of *Paint pot accidentally falls from top of ladder* **or** *Paint pot deliberately dropped from top of ladder*. The main elements of Boolean Algebra form the basis for this technique.

5.2.2 FTA and STA also provide a graphical representation of the combinatorial logic and ways of addressing the frequency and probability of events. These are, however, outside the scope of Boolean Algebra (see Pt3Ch29 for FTA/STA and Pt4Ch6 for probability).

5.2.3 It is possible to use negation in FTA and STA but this should be avoided if possible. The negating of a state or event, by taking the converse (*No man under ladder*), introduces a level of complexity that reduces the benefit to be gained from the simple representation of the scenario. Negation changes a branch of a Fault Tree into a Success Tree and vice-versa.

5.2.4 FTA and STA apply restrictions to the way in which events and states may be combined. This occurs because the statements of Boolean Algebra are used to describe events and states. Events are, in theory, instantaneous and the probability of two happening at exactly the same time is negligible. It is essential that functions (**and** and **or**) are only applied to combinations of statements for which they are mathematically correct.

5.3 Functional system control

5.3.1 Boolean Algebra forms a major element of many modern control systems. The subject of digital electronics is based on its principles and uses many extensions to the basic functions presented in this chapter. These allow negation, storage of data and even delay (time is not considered in basic Boolean Algebra unless explicitly written into the description).

5.4 Legal Documents

5.4.1 Boolean Algebra also has application to the simplification of conditions in legal documents, contracts and specifications, etc.

Intentional blank page