

Boolean Algebra

Boolean algebra is one of the fundamental mathematical tools used in switching theory. We begin with a brief review of its basic properties.

Definition: A *set* is a collection of objects.

Example:

$X = \{a, b, c\}$ is a finite set. $a \in X$ indicates membership of an element in a set.

Example:

$Y = Z^+ = \{ \text{all positive integers} \}$ is an infinite set.

Definition: If two sets X and Y have exactly the same elements they are said to be equal ($X = Y$).

Definition: A binary operation on a set X which has the property of **closure** maps $(a, b) \rightarrow c$ where $a, b, c \in X$. (A closed unary operation maps $a \rightarrow b$ where $a, b \in X$.)

Definition: A set of elements B and two associated closed binary operations, $(\bullet, +)$, form a *Boolean algebra* if and only if the following independent postulates are satisfied:

(A1) \bullet and $+$ are commutative.

$$\begin{aligned}a + b &= b + a \\ a \bullet b &= b \bullet a\end{aligned}$$

(A2) \bullet and $+$ distribute over each other.

$$\begin{aligned}a \bullet (b + c) &= (a \bullet b) + (a \bullet c) \\ a + b \bullet c &= (a + b) \bullet (a + c)\end{aligned}$$

(A3) There exists in B identity elements for $+$ and \bullet denoted 0 and 1 respectively such that for all $a \in B$

$$\begin{aligned}0 + a &= a \\ 1 \bullet a &= a\end{aligned}$$

(A4) For each $a \in B$ there exists $a' \in B$ such that

$$\begin{aligned}a + a' &= 1 \\ a \bullet a' &= 0\end{aligned}$$

- There are many other sets of axioms that can also be used to define a Boolean algebra. By definition, one set can be deduced from the others (although the derivation is not necessarily obvious.)
- Note associativity

$$(a + b) + c = a + (b + c)$$
$$(ab)c = a(bc)$$

is not mentioned in the axioms above. It can be deduced from the axioms and is often included in the set redundantly for the sake of clarity.

- The uniqueness of a' can be deduced from the axioms.

Note the differences from the more familiar algebra of addition and multiplication on the real numbers.

- $a + (b \cdot c) = (a + b) \cdot (a + c)$ does not hold for real numbers using addition and multiplication
- a' does not exist as defined in A4

Example:

{B = the set of all finite sets of integers from the closed interval [0, 100]}

(This includes the empty set \emptyset .)

- is taken to be set intersection \cap

- + is taken to be set union \cup

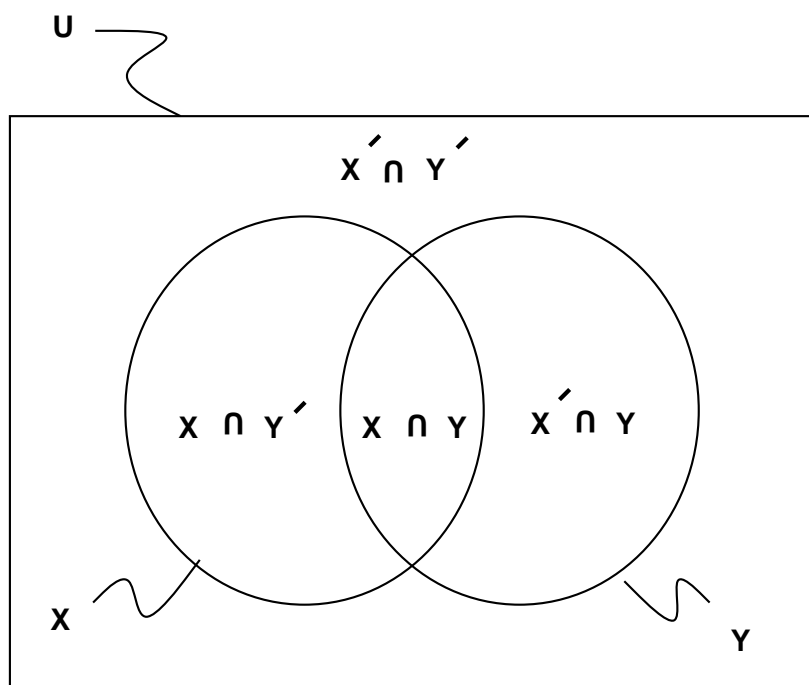
- ' is taken to be set complement, i.e., $A' = U - A$ or everything in U not also in A where U is the universal set (in this case $\{0, 1, \dots, 100\}$).

It is easy to verify that this yields a Boolean algebra with

- 0 identified with \emptyset
- 1 with U

Set algebras are useful because:

1. It is easy to visualize set operations (and therefore identities and proofs) via Venn diagrams.



2. [Whitesitt 1961] For any Boolean algebra there is a universal set U such that the algebra of subsets of U has the same algebraic structure. Therefore, proofs based on inclusions and other set concepts can be used.

Switching Algebra

Definition: A *switching algebra* is a Boolean algebra with the set

$$B = \{0, 1\}$$

and the following closed unary and binary operations in precedence order (x and y are switching variables that can take on the values 0 or 1):

(i) NOT (complement) denoted $'$ (\neg)

x	x'
0	1
1	0

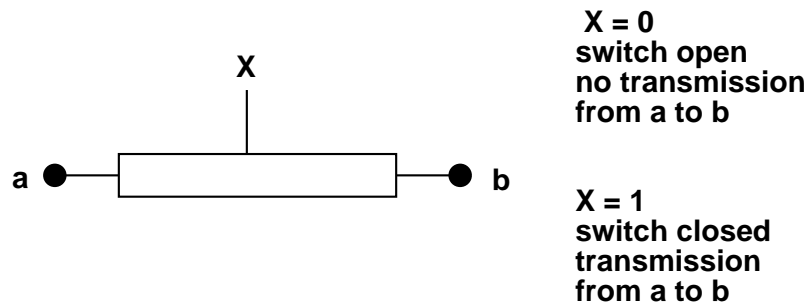
(ii) AND (product, conjunction) denoted \bullet (\wedge)

x	y	$x \bullet y$
0	0	0
0	1	0
1	0	0
1	1	1

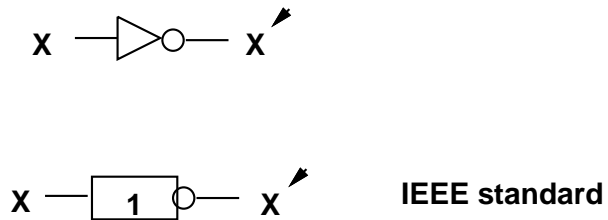
(iii) OR (sum, disjunction) denoted $+$ (\vee)

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

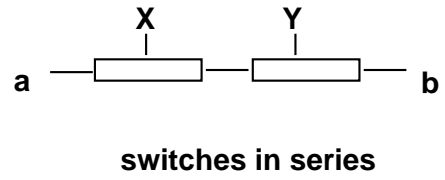
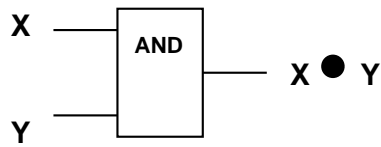
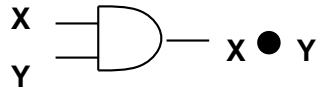
Switching variables are used to specify the states on gate inputs and outputs in gate networks and the state of a switch (open or closed) in a switch network.



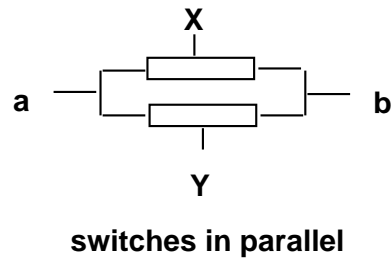
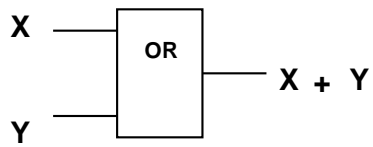
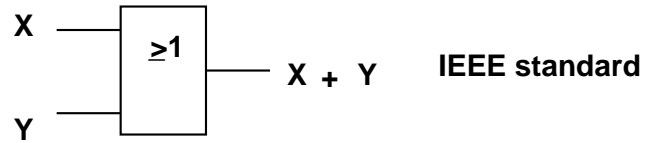
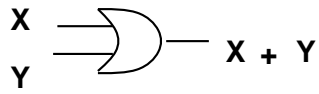
NOT



AND



OR



Combinational Networks

Definition: A *combinational network* is a logical network of gates or switches without memory, i.e., outputs depend only on present inputs.

An acyclic gate or switch network is a combinational network.

We wish to analyze (1-3) and synthesize (4) combinational networks relative to their steady state behavior.

1. given a gate or switch network, derive a switching expression that defines the network's function
2. given two networks determine if they are equivalent
3. given a network derive a more efficient equivalent network
4. given a logical function derive and optimize a network that realizes the function

Definition: A *switching expression* is either

a constant – 0 or 1

or a variable – x, y, z, \dots (also $x_1, x_2 \dots$) or its complement x'

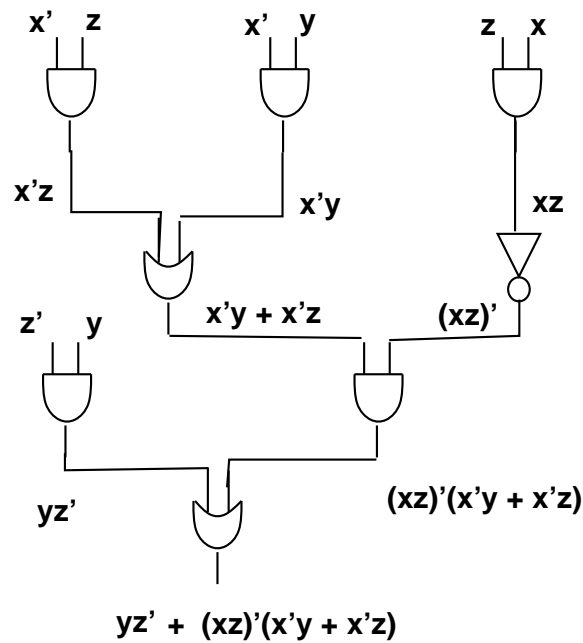
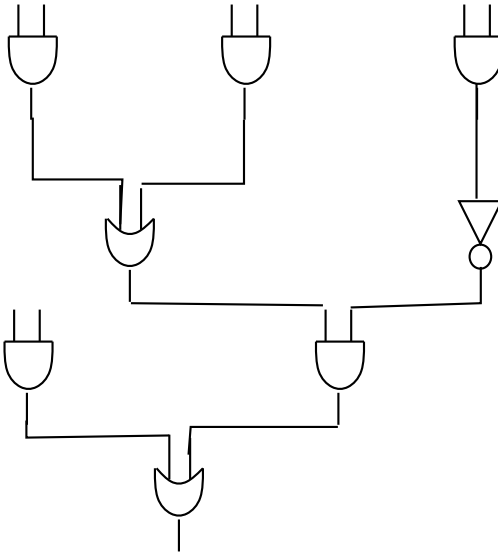
or $(E)'$ or $(E \bullet F)$ or $(E + F)$ where E and F are (shorter) expressions

The operator \bullet has precedence of evaluation over $+$ and is often omitted from the expression, i.e.,

$$\begin{aligned} w \bullet x + y \bullet z &= wx + yz \\ &= ((wx) + (yz)) \end{aligned}$$

The symbol for a switching variable, x , or its complement, x' are call **literals**. They are the basic symbolic building block of switching expressions.

So it is easy to build a switching expression from a gate or switch network. Once the expression is known the truth table can be evaluated.



Definition: Two switching expressions $E(x_1, \dots, x_n)$ and $F(x_1, \dots, x_n)$ are *equivalent*, written $E = F$, if

$$E(a_1, \dots, a_n) = F(a_1, \dots, a_n)$$

for all $(a_1, \dots, a_n) \in \{0, 1\}^n$, i.e., for all assignments of 0 and 1 to the variables.

If the switching expressions are the same then any networks implementing them have the same input to output relationship, i.e., seen from the outside they are equivalent. Hence, the one which has more acceptable cost can be used.

Equivalence of expressions can be proven by three basic techniques:

- Compute the result for each side of the identity for all possible assignments of 0 and 1 to the variables. This is called **perfect induction** and although not very elegant it is very effective given the fact that the algebra is defined with a set that has only two elements. It is also extremely easy to automate. (It is less useful in a true Boolean algebra.)
- Use set algebra to visualize and or prove the identity.
- Derive the identity from previously proven identities and the postulates of the switching algebra. This is often the most elegant. It can be the fastest. But it is also the hardest to automate.

$$E \rightarrow E_1 \rightarrow E_2 \rightarrow E_3 = F$$

Example: Test the hypothesis using perfect induction.
 (This is just one of the distribution laws of the algebra.)

$$x + yz = (x + y) \bullet (x + z)$$

x	y	z	$x + yz$	$x + y$	$x + z$	$(x + y) \bullet (x + z)$
0	0	0	0	0	0	0
0	0	1	0	0	1	0
0	1	0	0	1	0	0
0	1	1	1	1	1	1
1	0	0	1	1	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

The fourth and seventh columns are equal so the identity is proven.

Many useful basic properties can be deduced from the axioms. (Some axioms are included below for convenience.)

<i>identities</i>	$x \bullet 1 = x$	$x + 0 = x$
<i>nulls</i>	$x \bullet 0 = 0$	$x + 1 = 1$
<i>idempotency</i>	$x \bullet x = x$	$x + x = x$
<i>complements</i>	$x \bullet x' = 0$	$x + x' = 1$
<i>involution</i>	$(x')' = x$	
<i>associativity</i>	$x(yz) = (xy)z$	$x + (y + z) = (x + y) + z$
<i>distribution</i>	$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$

Note the relationship between the two columns. This is a demonstration of the principal of **duality**.

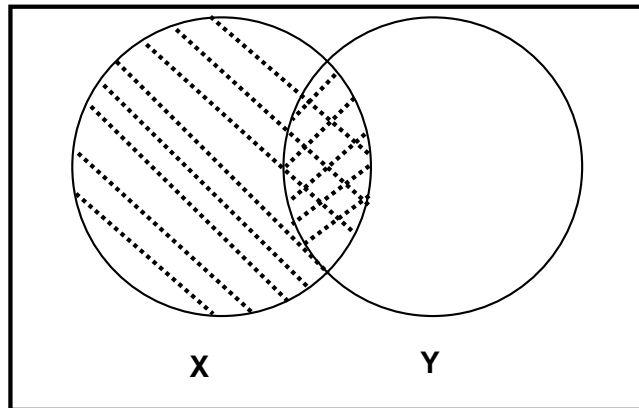
In a switching algebra, if all occurrences of $+$ and \bullet are swapped and all occurrences of 0 and 1 are swapped in a true identity the resulting identity is also true.

Care should be taken when applying duality to an expression when multiple levels of parentheses are involved.

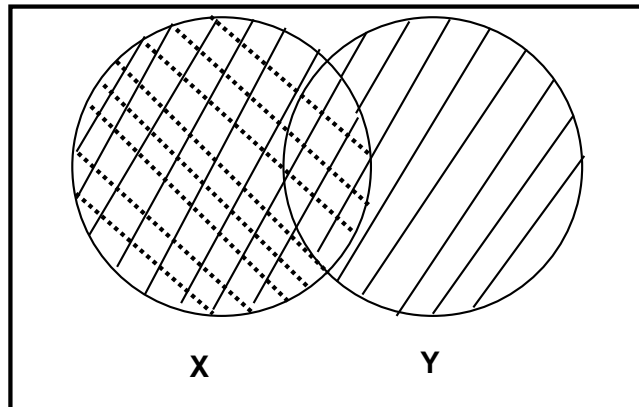
Some useful theorems that reduce the complexity of a switching expression.

Absorption: eliminates one of the terms

$$X + XY = X$$



$$X(X + Y) = X$$



absorption is particularly useful if Y happens to be a fairly complex expression.

$$wx'y + wx'y(wx + yz) = wx'y$$

Set theory proofs are obvious from the diagram.

We must show $X \cup (X \cap Y) \subseteq X$ and $X \subseteq X \cup (X \cap Y)$ in order to show equality of the sets.

Proof:

(1) $X \subseteq X \cup (X \cap Y)$.

Let $\sigma \in X$. $X \cup (X \cap Y)$ includes, by definition, all elements, μ , that are either $\mu \in X$ or $\mu \in X \cap Y$. Therefore, since $\sigma \in X$ we have $\sigma \in X \cup (X \cap Y)$.

(2) $X \cup (X \cap Y) \subseteq X$.

Let $\mu \in X \cup (X \cap Y)$. So either $\mu \in X$ or $\mu \in X \cap Y$.

The first, $\mu \in X$, trivially satisfies the inclusion we wish to prove.

If, on the other hand, $\mu \in X \cap Y$ then $\mu \in X$ and $\mu \in Y$.

So once again we have $\mu \in X$ trivially. \square

Algebraic proofs are

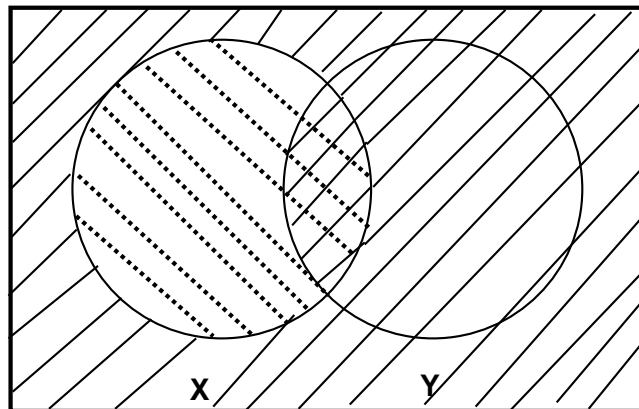
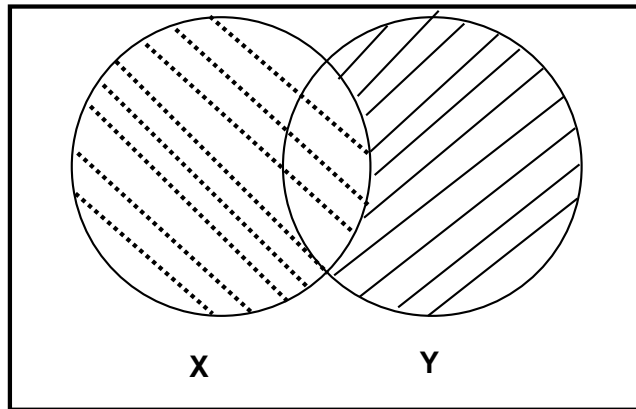
$$\begin{aligned}x + xy &= \\(\textit{ident.}) &= x \bullet 1 + xy \\(\textit{dist.}) &= x(1 + y) \\(\textit{null.}) &= x \bullet 1 \\(\textit{ident.}) &= x\end{aligned}$$

$$\begin{aligned}x(x + y) &= \\(\textit{dist.}) &= x + xy\end{aligned}$$

proceed as above

if one of the X literals is replaced by X'
the reduction is less successful but still
useful. one operation is removed

$$X + X'Y = X + Y$$



$$X(X' + Y) = XY$$

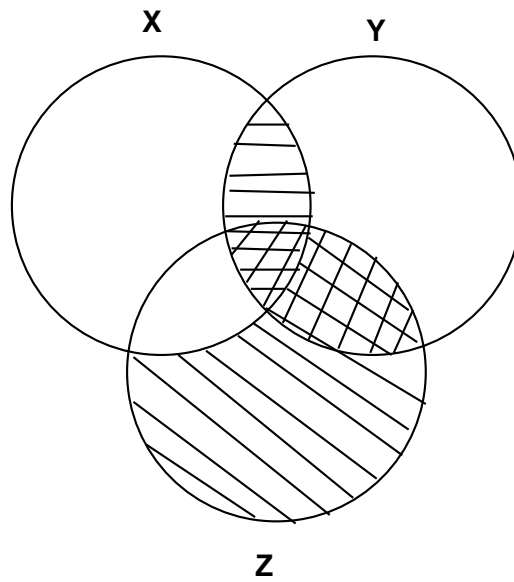
Set theory proofs are obvious from the diagram.

Algebraic proofs are

$$\begin{aligned}x + x'y &= \\(\textit{ident.}) &= x \bullet 1 + x'y \\(\textit{null.}) &= x(1 + y) + x'y \\(\textit{dist.}) &= x + xy + x'y \\(\textit{dist.}) &= x + (x + x')y \\(\textit{comp.}) &= x + 1 \bullet y \\(\textit{ident.}) &= x + y\end{aligned}$$

$$\begin{aligned}x(x' + y) &= \\(\textit{dist.}) &= xx' + xy \\(\textit{comp.}) &= 0 + xy \\(\textit{ident.}) &= xy\end{aligned}$$

Consensus: reduces a sum of products by one product, and a product of sums by one sum



$$XY + X'Z + YZ = XY + X'Z$$

look for a variable and its complement involved with two other variables

the dual expression is

$$(X+Y) (X'+Z) (Y+Z) = (X+Y) (X'+Z)$$

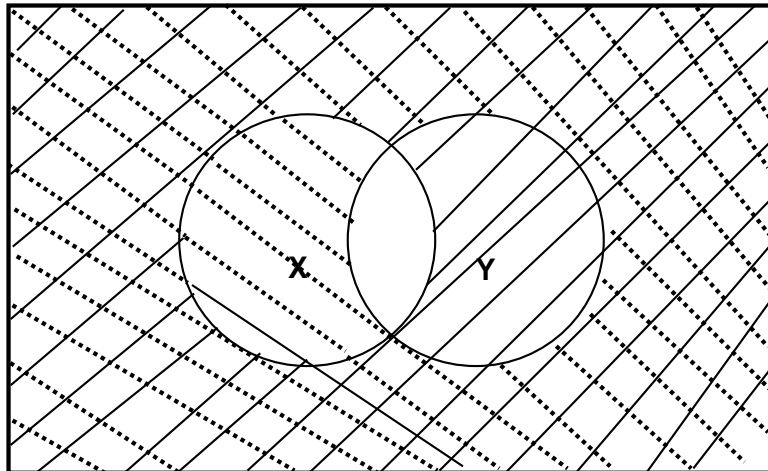
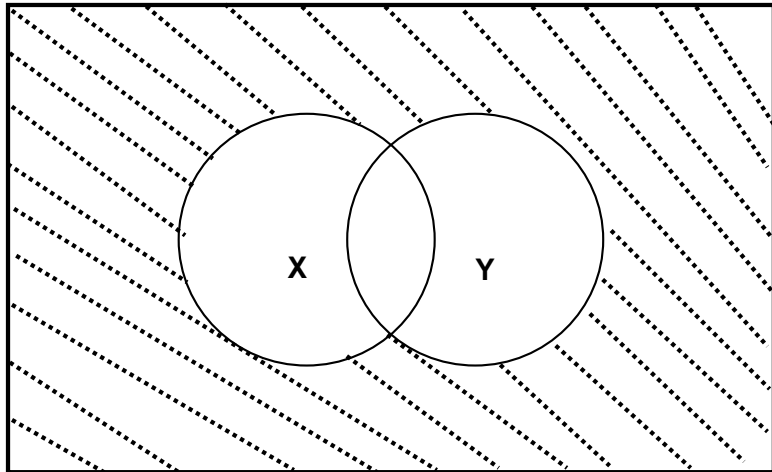
Consensus is one of the most important basic theorems of Boolean algebra. In fact, a generalization corresponds to an essential algebraic entity that is the building block of all of the basic theory of optimal synthesis.

The proof of the dual is left as an exercise.

$$\begin{aligned}
 xy + x'z + yz &= \\
 (\text{comp.}) &= xy(z + z') + x'z(y + y') + yz(x + x') \\
 (\text{dist.}) &= xyz + xyz' + x'yz + x'y'z + xyz + x'yz \\
 (\text{idem.}) &= xyz + xyz' + x'yz + x'y'z \\
 (\text{dist.}) &= xy(z + z') + x'z(y + y') \\
 (\text{comp.}) &= xy + x'z
 \end{aligned}$$

DeMorgan's Laws

$$(X + Y)' = X' Y'$$



Generalized Demorgan

DeMorgan's law can be extended to switching functions of n variables.

Let $F(x_1, \dots, x_n, 0, 1, +, \bullet)$ be a switching expression. The complement of the expression F' can be given by

$$F' = F(x'_1, \dots, x'_n, 1, 0, \bullet, +)$$

i.e., F' is obtained from F by replacing each variable and constant with its complement and by interchanging \bullet with $+$.

Examples:

$$\begin{aligned}(x_1 x_2 \cdots x_k)' &= x_1' + x_2' + \cdots + x_k' \\ (x_1 + x_2 + \cdots + x_k)' &= x_1' x_2' \cdots x_k'\end{aligned}$$

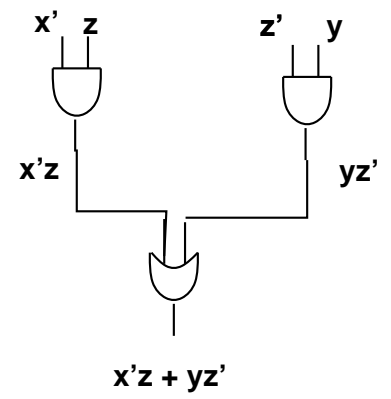
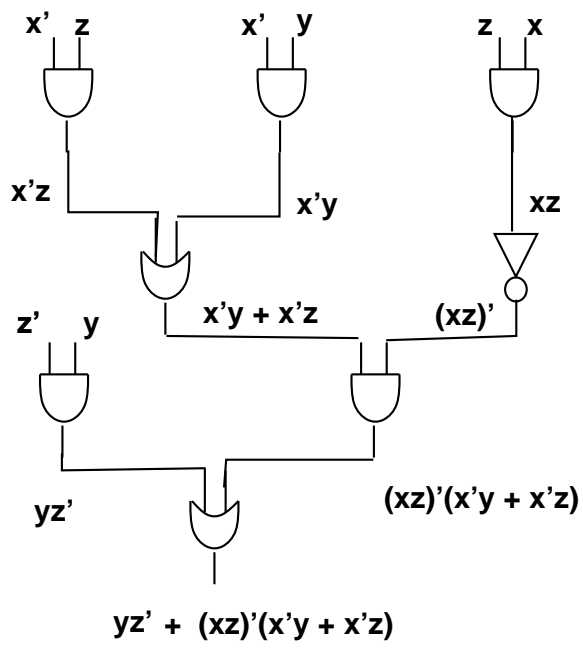
$$\begin{aligned}F &= w + x'y + (w + y')(x + w'z) + 0 \\ F' &= [w + x'y + (w + y')(x + w'z) + 0]' \\ &= (w)' \bullet (x'y)' \bullet [(w + y')(x + w'z)]' \bullet (0)' \\ &= w' \bullet (x + y') \bullet [(w + y')' + (x + w'z)'] \bullet 1 \\ &= w' \bullet (x + y') \bullet [w'y + (x' \bullet (w'z)')] \bullet 1 \\ &= w' \bullet (x + y') \bullet [w'y + (x' \bullet (w + z'))] \bullet 1 \\ &= w' \bullet (x + y') \bullet [w'y + (x' \bullet (w + z'))]\end{aligned}$$

We now have the tools to simplify the network given earlier.

$$\begin{aligned}
 yz' + (xz)'(x'y + x'z) &= \\
 \text{DeMorgan} &= yz' + (x' + z')(x'y + x'z) \\
 \text{Dist.} &= yz' + (x' + z')x'(y + z) \\
 \text{Dist.} &= yz' + (x' + x'z')(y + z) \\
 \text{Absor.} &= yz' + x'(y + z) \\
 \text{Dist.} &= yz' + x'y + x'z \\
 \text{Cons.} &= yz' + x'z
 \end{aligned}$$

So we have two equivalent networks. One with fewer gates and delay.

Equivalent networks



- one network has many switching expressions
- simpler expression can generate a new network
- all have equivalent input to output relationships
- need to characterize network forms that arise from input to output relation
- need to show how to analyze other gates

Definition: A **switching function** on n variables is an assignment of a value of 0 or 1 to each of the possible combinations of values of the variables. This assignment can be unambiguously specified by means of a **truth table**, i.e., two different truth tables do not implement the same switching function.

x	y	z	$f(x, y, z)$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Theorem: Every switching expression defines a switching function f . The converse holds for a switching algebra: For each switching function there are switching expressions that evaluate f . (This is not true for general Boolean algebras.)

We have already seen how to produce the switching function, i.e., truth table, from a network or switching expression.

Now consider how to create a switching expression which implements a given switching function.

Consider a row of the table where $f = 1$, e.g., the first row. Suppose we want to write a product involving x, y and z and/or their complements that evaluates to 1 **only** when x, y and z take on their values from that row, i.e., all 0.

Consider $x'y'z'$. Note the only time all terms in the product are 1 (and therefore the product is 1) is when $x, y,$ and z are all 0.

We can create such a product for any row that has $f = 1$ by merely including x, y or z if the value of the variable in the row is 1 and the complement of the variables that are 0.

For the second row, for example, we get $x'y'z$ which evaluates to 1 only when $(x, y, z) = (0, 0, 1)$ as desired.

The product, thus defined by each row, is called a **minterm** or **fundamental product**.

It follows that $f = 1$ for an assignment of (x, y, z) if any of the minterms evaluate to 1 for that same assignment.

A switching expression for f , called **the canonical sum of products**, can be constructed from the sum of the fundamental products (minterms) for whose associated assignment $f = 1$. These minterms are often called the **1-minterms** of f .

For the table above,

$$f(x, y, z) = x'y'z' + x'y'z + x'yz + xy'z' + xy'z$$

Note the position of the complements (as 0) and the variables (as 1) induce the binary representation of the row numbers (starting at 0) of the assignments for which $f = 1$.

If the rows of the truth table are ordered based on interpreting the binary pattern of the inputs as a row number then the canonical product induces the notation

$$f(x, y, z) = \sum(0, 1, 3, 4, 5)$$

where the row numbers for which $f = 1$ are listed.

Note that one can get the canonical sum notation for f' by using the remaining minterm numbers, i.e., those that have $f = 0$ in the original table:

$$f'(x, y, z) = \sum(2, 6, 7)$$

Definition: The **canonical product of sums** is formed by taking the product of *maxterms* or *fundamental sums* corresponding to rows where $f = 0$. The maxterm for a given row is the sum of literals produced by using the complement for variables whose input for that row is 1 and the uncomplemented variable for those whose value is 0.

Example: The maxterm for the third row of the table (formally row with binary index 010) is

$$x + y' + z$$

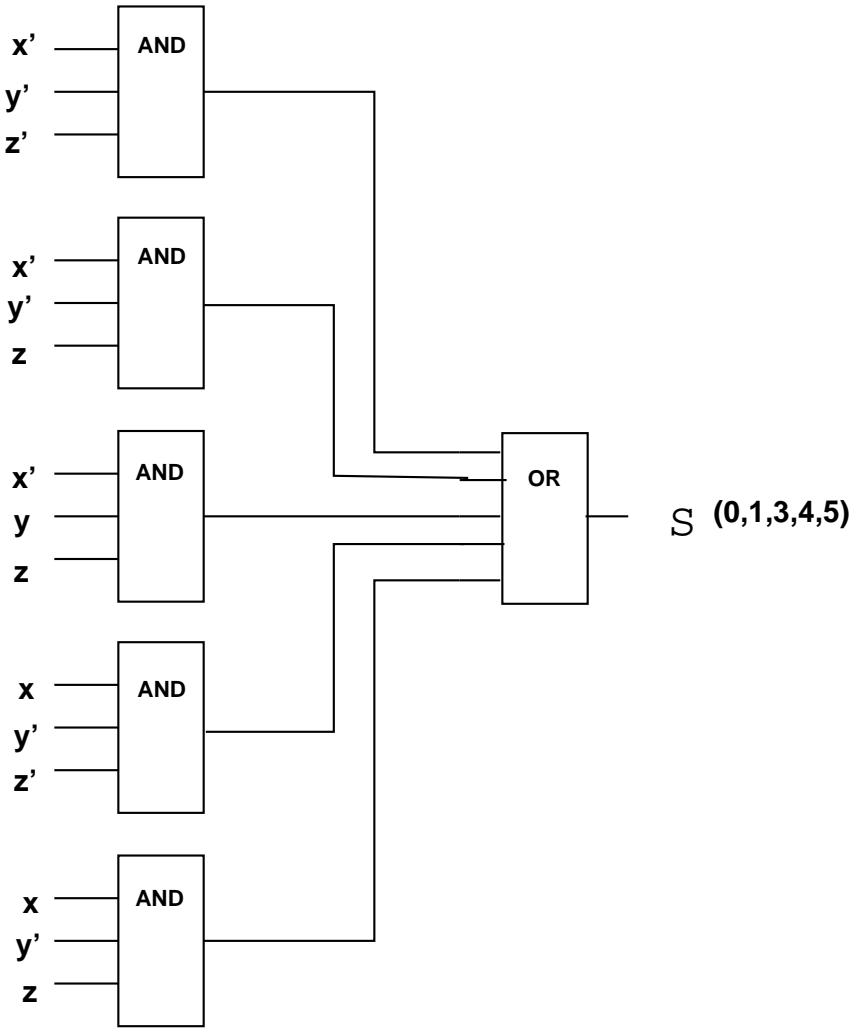
The canonical product is denoted by the indices of the rows for which $f = 0$

$$\begin{aligned} f(x, y, z) &= \prod(2, 6, 7) \\ &= (x + y' + z)(x' + y' + z)(x' + y' + z') \end{aligned}$$

Forms that Specify a Switching Function of n variables

- Canonical sum of products
 - one product of literals for each 1-minterm
 - each product has n literals (one for each variable)
- Canonical product of sums
 - one sum of literals for each 0-maxterm
 - each sum has n literals (one for each variable)
- Standard forms:
 - sum of products – each product may have fewer than n literals and there may be fewer than 1 product for each 1-minterm.
 - product of sums – each sum may have fewer than n literals and there may be fewer than 1 sum for each 0-maxterm.
- Nonstandard form – Any switching expression that is not one of the above.

These forms correspond to particular switching network structures.



Switching networks may consist of gates other than *AND*, *OR* and *NOT*. Any truth table can be used to define the function of a gate. (See text for examples)

However, if we are given a set of such gates and are allowed to implement switching functions via networks consisting of **only** those gates, we need to determine if all switching functions admit such an implementation.

Definition: A set of operations is **complete** if every switching function can be expressed in terms of operations from this set only.

Examples:

- { AND, OR, NOT } is complete by construction

- { AND, NOT } is complete by DeMorgan –

$$E + F = (E'F')'$$

- { OR, NOT } is also complete by DeMorgan –

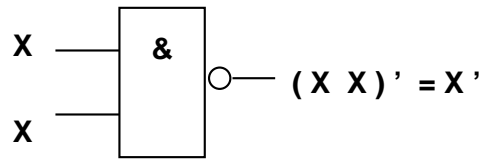
$$EF = (E' + F')'$$

- { AND, OR } is *not complete* – no NOT function

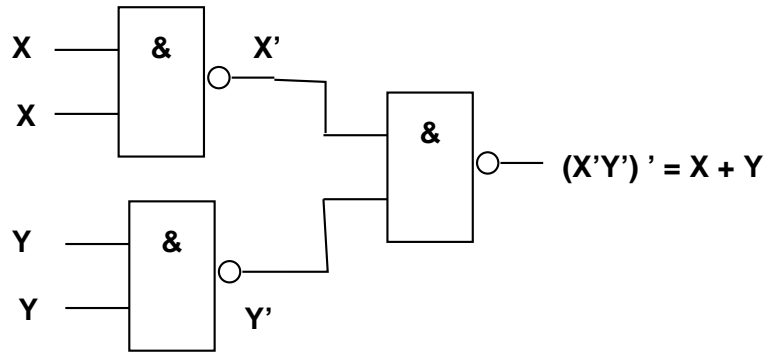
Note constants 0 and 1 cannot be assumed to exist as input to gates unless explicitly listed in set. (If they are not you have to be able to create them from an arbitrary switching variable and the gates in the set, e.g, $1 = x + x'$ can be constructed for the set { AND, OR, NOT }.)

The NAND gate is complete

$$\text{NAND} \longrightarrow (X Y)'$$



NOT can be implemented



OR can be implemented

Analysis and design techniques specifically targeting NAND gate realizations will be discussed later

The NAND gate is complete

$$\text{NAND} \longrightarrow (X Y)'$$

A two-level AND/OR network from the canonical sum is easily converted to a two-level NAND circuit

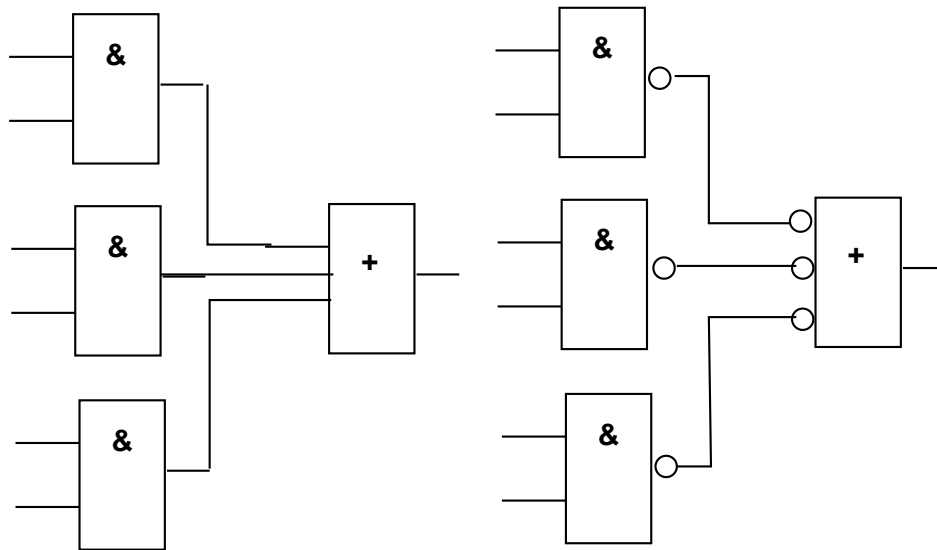
Recall that De Morgan says

$$(XY)' = (X' + Y')$$

or in terms of gates



Conversion is trivial



Other gate definitions and the VLSI technology used to implement them are discussed fairly well in the text.