Boolean Algebra

George Boole (1815-1864) English mathematician and philosopher.

- Defined on the set B={0,1}
- Binary operators: AND, OR { • , + }
- Unary operation: Complement (NOT) { ' }

a • b

a \ b	0	1
0	0	0
1	0	1

a + b

a \ b	0	1
0	0	1
1	1	1

Complement

a	a'
0	1
1	0

- **Axioms (Laws):**

Under assumption a , b ∈ B

1. Closure:	a + b ∈ B	a • b ∈ B
2. Commutative:	a + b = b + a	a • b = b • a
3. Associative:	a + (b + c) = (a + b) + c	a • (b • c) = (a • b) • c
4. Identity :	a + 0 = a	a • 1 = a
5. Distributive:	a + (b • c) = (a + b) • (a + c)	a • (b + c) = (a • b) + (a • c)
6. Inverse:	a + a' = 1	a • a' = 0

Properties and Theorems:

These properties and theorems are derived from the operations and axioms of the Boolean algebra.
They can be proven by using the axioms.

1. Annihilator (or Dominance):
   a + 1 = 1 a • 0 = 0
2. Involution: (a')' = a
3. Idempotency:
   a+a+a+….+a = a a•a•a •… •a = a
4. Absorption: (Proof in 2.4)
   a + a·b = a a·(a+b) = a
5. De Morgan Theorem: *Augustus De Morgan (1806 – 1871)*
   (a + b)' = a' • b' (a • b)' = a' + b'
5. General De Morgan Theorem:
   f '(X1,X2,...,Xn,0,1,+,•) = f(X1',X2',...,Xn',1,0,•,+)
   - It establishes relations between binary operations (AND, OR): • and +

6. The Duality principle

A dual of a logical expression is obtained by replacing, • by +, + by •, 0 by 1, and 1 by 0, without changing the variables.

$$a + b + 0 \ldots \Leftrightarrow a \bullet b \bullet 1 \ldots$$

Duals of all proven theorems are also valid.

If we can prove a theorem than we know that its dual is also true.

Example:

Absorption:

If we can prove the theorem $a + a \cdot b = a$, than its dual $a \cdot (a+b) = a$ is also true.

Note that in previous slides axioms and theorems are presented with their duals.

General Duality Principle:

$$f(X1,X2,\ldots,Xn,0,1,+,\bullet) \Leftrightarrow f^D(X1,X2,\ldots,Xn,1,0,\bullet,+)$$

- It is different from the De Morgan Theorem.
  - It establishes relation between proofs of theorems.
  - It is not used to convert one expression into another. Duals are not equal.

Precedence Order Of Operators:

From higher to lower precedence:

1. Parenthesis, 2. Complement (Negation), 3. AND, 4. OR

Proving the theorems:

a) With Axioms

Example:

Theorem: $X \bullet Y + X \bullet Y' = X$

Proof:

Distributive	$X \bullet Y + X \bullet Y'$	$= X \bullet (Y + Y')$
Complement	$X \bullet (Y + Y')$	$= X \bullet (1)$
Identity	$X \bullet (1)$	$= X$ ✓

Example:

Theorem: $X + X \bullet Y = X$ (*Absorption*)

Proof:

Identity	$X + X \bullet Y$	$= X \bullet 1 + X \bullet Y$
Distributive	$X \bullet 1 + X \bullet Y$	$= X \bullet (1 + Y)$
Annihilator	$X \bullet (1 + Y)$	$= X \bullet (1)$
Identity	$X \bullet (1)$	$= X$ ✓

Proving the theorems: **b) With the truth table**

De Morgan:

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

X	Y	X'	Y'	(X + Y)'	X' • Y'
0	0	1	1	1	1
0	1	1	0	0	0
1	0	0	1	0	0
1	1	0	0	0	0

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

X	Y	X'	Y'	(X • Y)'	X' + Y'
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	0	0

Although truth tables may contain many rows, they can be filled and compared by computer programs in very short times.

Theorems are also used to minimize the logical expressions:

Example:

$$
\begin{aligned}
Z &= A'\,B\,C + A\,B'\,C + A\,B\,C' + A\,B\,C \\
&= A'\,B\,C + A\,B'\,C + A\,B\,C' + A\,B\,C + A\,B\,C \\
&= A'\,B\,C + A\,B\,C + A\,B'\,C + A\,B\,C' + A\,B\,C \\
&= (A' + A)\,B\,C + A\,B'\,C + A\,B\,C' + A\,B\,C \\
&= (1)\,B\,C + A\,B'\,C + A\,B\,C' + A\,B\,C \\
&= B\,C + A\,B'\,C + A\,B\,C' + A\,B\,C + A\,B\,C \\
&= B\,C + A\,B'\,C + A\,B\,C + A\,B\,C' + A\,B\,C \\
&= B\,C + A\,(B' + B)\,C + A\,B\,C' + A\,B\,C \\
&= B\,C + A\,(1)\,C + A\,B\,C' + A\,B\,C \\
&= B\,C + A\,C + A\,B\,(C' + C) \\
&= B\,C + A\,C + A\,B\,(1) \\
&= B\,C + A\,C + A\,B
\end{aligned}
$$

Logical (Boolean) Expressions

A logical expression, is a finite combination of variables, constants and operators that are well-formed according to the rules.

It is represented as E(X), where X= (x_1, x_2, ... x_n) and each $x_i \in \{0,1\}$.

If E_1 and E_2 are logical expressions than, E_1', E_2', $E_1 + E_2$, $E_1 \bullet E_2$ and all possible combinations are also logical expressions.

Normal Forms of Logical expressions:

Each logical expression can be written in two special forms.

1. **Disjunctive normal form (DNF): ΣΠ**

   Logical sum of logical products (SOP). OR of ANDs.

   Example: bc'+ad+a'b

   The OR operation is also called the *logical disjunction*.

2. **Conjunctive normal form (CNF): ΠΣ**

   Logical product of logical sums (POS). AND of ORs.

   Example: (a+b+c')(a+d)(a'+b)

   The AND operation is also called the *logical conjunction*.

Any logical expression can be written in CNF (POS form) and DNF (SOP form).
Any expression in CNF can be converted to DNF and vice versa.

Value of a logical expression:

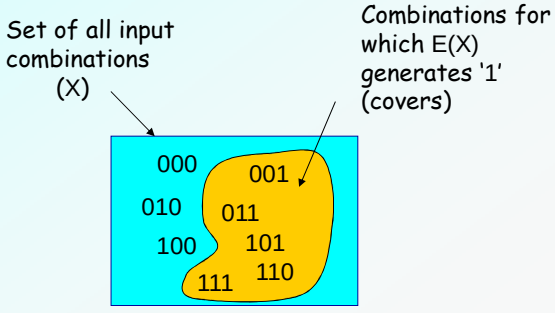
An expression E(X), generates for each combination of the input vector X=(x_1, ... x_n), a value from the set of B={0,1}.

These values constitute the truth table of the expression.

Example: $E(X) = x_1x_2 + x_3$

Truth table of the expression

X= x_1	x_2	x_3	E(X)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



"Order Relation" between expressions:

To explain some properties of the logical expressions the following order relation can be used.

An order relation between elements of set B={0,1}: $0 < 1$

Read as "0 precedes 1" or "0 is smaller than 1".

According to this, an order relation between X vectors can be defined as follows:

If all elements of X1 precede (are smaller than) elements in the same order of vector X2 than $X1 \leq X2$.

Example:

X1=1001 , X2 = 1101

$X1 \leq X2$.

The order relation may not exist between all vectors.

Example: X1=0011 , X2 = 1001

The order relation is not valid between X1 and X2.

Order relation between expressions:

$E_1(X) \leq E_2(X)$ denotes that, values of E_1 generated by combinations of input X are always smaller than (or equal to) all values of E_2 generated by the same input combinations.

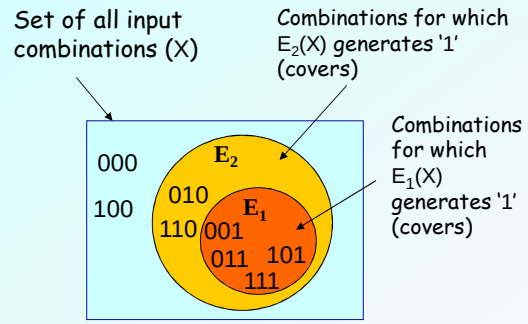
Example :

x_1	x_2	x_3	$E_1(X)$		$E_2(X)$
0	0	0	0	=	0
0	0	1	1	=	1
0	1	0	0	<	1
0	1	1	1	=	1
1	0	0	0	=	0
1	0	1	1	=	1
1	1	0	0	<	1
1	1	1	1	=	1

For each input combination, where $E_1(X)$ generates 1, $E_2(X)$ also generates 1. (This is a special case.)

If $E_1(X) \leq E_2(X)$:

1. $E_1(X) + E_2(X) = E_2(X)$
2. $E_1(X) \bullet E_2(X) = E_1(X)$

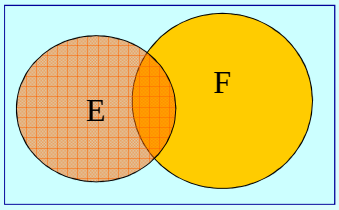


If $E_1(X) \leq E_2(X)$

$E_1(X)$ implies $E_2(X)$, $E_1(X) \Rightarrow E_2(X)$,

$E_2(X)$ covers $E_1(X)$.

The order relation (≤) may not be valid between all expressions.



IF E and F are two logical expressions the following inequalities are always true:

$E \cdot F \le E \le E+F$

$E \cdot F \le F \le E+F$

Absorption Properties of expressions:

$E+E \cdot F = E$ dual $E(E+F) = E$

Proof: $E(E+F) = EE+EF = E+EF = E(1+F) = E$

$E+E' \cdot F = E+F$ dual $E(E'+F) = E \cdot F$

Proof: $E+E'F = (E+E')(E+F) = 1(E+F) = E+F$

There properties are used to minimize (simplify) logical expressions.

The Consensus theorem

Assume that E_1 and E_2 are two expressions, which do not include the literal x_1 :
$E_1(x_2, ... x_n)$ and $E_2(x_2, ... x_m)$

We can create a new expression by multiplying one term by x_1 and the other one by the complement of x_1 .

$$E = x_1 E_1 + x_1' E_2$$

Here, x_1 is called a **biform** variable, because it occurs both positively (as itself) and negatively (as complement) in the expression.

Examples: $x_1(x_2+x_3') + x_1'(x_3+x_4)$,
$x_1 x_2 x_3 + x_1' x_4 x_5$

•Given a pair of product terms that include a biform variable, **the consensus term** is formed by multiplying two original terms together, leaving out the selected variable and its complement.

• E_1E_2 is **the consensus term** of $xE_1 + x'E_2$ with respect to the variable x.

Example: Consensus of abc + a'cd (respect to a) is bccd = **bcd**

Theorem: The consensus term is redundant and can be eliminated.

$$xE_1 + x'E_2 + E_1E_2 = xE_1 + x'E_2$$

The Consensus theorem (dual)

According to the duality principle the consensus theorem is also valid for expressions written in product of sums (POS) form.

Assume that E_1 and E_2 are two expressions, which do not include the literal x_1:
$E_1(x_2, ... x_n)$ and $E_2(x_2, ... x_m)$

We can create a new expression by adding x_1 to one term and the complement of x_1 to the other one.

$$E = (\mathbf{x_1}+E_1)(\mathbf{x_1'}+E_2)$$

Here, x_1 is a **biform** variable.

Examples: $(x_1+x_2+x_3')(x_1'+x_3+x_4)$,
$(x_1+x_2x_3')(x_1'+x_3x_4)$

•Given a pair of sums that include a biform variable, **the consensus term** is formed by adding two original terms together, leaving out the selected variable and its complement.

• $\mathbf{E_1 + E_2}$ is **the consensus term** of $(x+E_1)(x'+E_2)$ with respect to the variable x.

Example: Consensus of (a+b+c) (a'+c+d) is: b+c+c+d = **b+c+d**

Theorem: The consensus term is redundant and can be eliminated.

$$(x+E_1)(x'+E_2)(E_1+E_2) = (x+E_1)(x'+E_2)$$

Example: Minimization of a logical expression using the consensus theorem

F(A, B, C)	= A'B'C + A'BC + AB'C + ABC + ABC'	
	= A'B'C + A'BC + AB'C + ABC + ABC' + AB	Consensus (respect to C) is added
	= A'B'C + A'BC + AB'C + ABC + ABC' + AB	Absorption
	= A'B'C + A'BC + AB'C + AB	
	= A'B'C + A'BC + A'C + AB'C + AB	Consensus (respect to B) is added
	= A'B'C + A'BC + A'C + AB'C + AB	Absorption
	= A'C + AB'C + AB	
	= A'C + AB'C + AB + AC	Consensus (B) is added Absorption
	= A'C + AB + AC	
	= A'C + AB + AC + C	Consensus (A) is added Absorption
	= AB + C	

Logical (Boolean) Functions

Logical functions are defined on the input set B^n (vectors with n binary variables.
There are 3 types of logical functions:

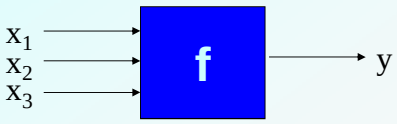
1. Simple (basic) functions: Multiple inputs, single output

$y = f(X): B^n \rightarrow B$

$\forall X^0 \in B^n ; \exists! \; y^0 \in B ; y=f(X)$

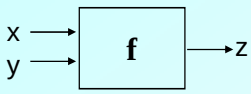
Example:

x_1	x_2	x_3	y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



There are $2^{(2^n)}$ possible basic logical functions for n binary variables (inputs).

There are 16 possible basic logical functions for 2 binary variables (inputs).



The truth table of 16 possible basic logical functions for 2 binary variables (F0–F15):

Inputs		Functions															
X	Y	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		0	X AND Y	X	Y	X XOR Y	X OR Y			X NOR Y (X OR Y)'	X = Y	Y'		X'		X NAND Y (X AND Y)'	1

2. General functions: Multiple inputs, multiple outputs

$Y = f(X): B^n \rightarrow B^m$, $X=(x_1, ... x_n)$, $Y=(y_1, ... y_m)$,

Example:

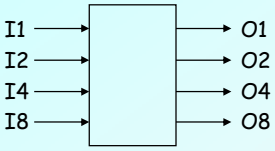
x_1 x_2 x_3	y_1 y_2
0 0 0	1 1
0 0 1	1 0
0 1 0	0 0
0 1 1	0 0
1 0 0	1 1
1 0 1	0 1
1 1 0	0 1
1 1 1	1 0

3. Incompletely specified functions: They have unspecified outputs for some input combinations.

The input-sequence of the function that the designer does not care about is called **don't-care** condition.

A don't-care input condition usually never happens, or differences in that don't result in any changes to the output.

Example: A function that increments BCD numbers



For these input combinations the output values of the circuit (function) are not specified. They can not happen.
Don't-care conditions are presented by an X or Φ.

I8	I4	I2	I1	O8	O4	O2	O1
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Representation of Logical (Boolean) Functions

There are different ways to represent (express) the same logical function.

While designing logical circuits the proper representation method can be used.

Representation with truth table:

Output values for all possible input combinations (variables) are written in table.

The input columns are usually constructed in the order of binary counting.

Numbered (Indexed) Representation:

Input variables are encoded as binary numbers.

We can assign a decimal number for each input combination according to its binary value.

To represent the function we can list the number of input combinations for witch the function generates the logical value "1" (or logical "0" or "Φ").

Example: Representation of a completely specified basic logical function:

Num	Input x_1 x_2	Output y
0	0 0	1
1	0 1	0
2	1 0	1
3	1 1	0

$y = f(x_1,x_2) = \cup_1(0,2)$ ∪ denotes "union" or "set of".

The order of the variables is important.
It must be the same as in the truth table.
Otherwise the numbers of the combinations will change.

$y = f(\mathbf{x_2,x_1}) = \cup_1(0,1)$

The same function can represented with "0"-genereting combinations.

$y = f(x_1,x_2) = \cup_0(1,3)$

Example: Representation of a completely specified general logical function :
The numbered representation is applied to all outputs.

Num	x_1	x_2	y_1	y_2
0	0	0	1	1
1	0	1	0	1
2	1	0	1	0
3	1	1	0	0

$y_1 = f(x_1,x_2) = \cup_1(0,2)$

$y_2 = f(x_1,x_2) = \cup_1(0,1)$

Representation of the same function with "0"-genereting combinations:

$y_1 = f(x_1,x_2) = \cup_0(1,3)$

$y_2 = f(x_1,x_2) = \cup_0(2,3)$

Example: Representation of an incompletely general logical function:

In this case writing only 1-generating or 0-generating input combinations is not sufficient.

At least two different groups must be presented.

No	x_1	x_2	y_1	y_2
0	0	0	1	1
1	0	1	0	Φ
2	1	0	Φ	0
3	1	1	0	Φ

$y_1 = f(x_1,x_2) = \cup_1(0) + \cup_0(1,3)$

or $y_1 = f(x_1,x_2) = \cup_1(0) + \cup_\Phi(2)$

or $y_1 = f(x_1,x_2) = \cup_0(1,3) + \cup_\Phi(2)$

$y_2 = f(x_1,x_2) = \cup_1(0) + \cup_0(2)$

or $y_2 = f(x_1,x_2) = \cup_1(0) + \cup_\Phi(1,3)$

or $y_2 = f(x_1,x_2) = \cup_0(2) + \cup_\Phi(1,3)$

Graphical Representation

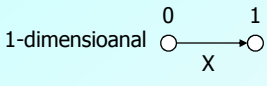
The input variables (combinations) of a logical function are elements of the set B^n. These variables can be represented as vertices of an n-dimensional hyper cube.

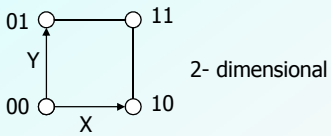
Vertices that correspond to 1-generating combinations are colored (marked).

Number of inputs of the function defines the dimension of the cube.

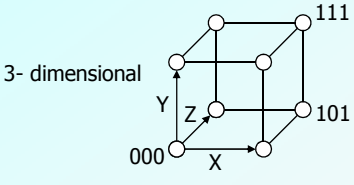
n-bit input → n-dimensional cube

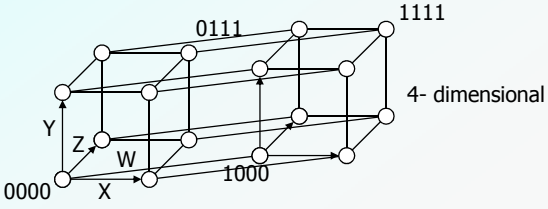
Boolean Cubes:





2- dimensional

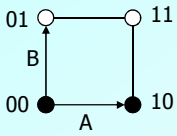




4- dimensional

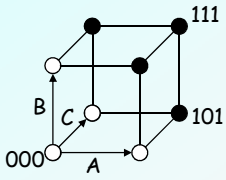
Example:

A	B	F
0	0	1
0	1	0
1	0	1
1	1	0



Example:

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



With the increase in the variables (inputs) it gets harder to draw a Boolean cube. Therefore Boolean cubes are not practical to represent Boolean functions with many inputs.

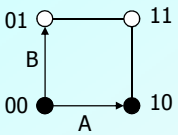
However, they make it easier to see some properties (especially the adjacent combinations) of the functions, and explain further topics.

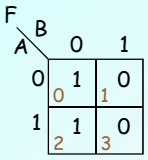
Karnaugh Maps (*Maurice Karnaugh (1924-), American physicist*)

Karnaugh maps are used to represent and simplify Boolean functions.

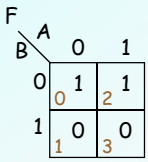
The Boolean variables and results are transferred (generally from a truth table) into a table in a matrix form.

Num.	A	B	F
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	0



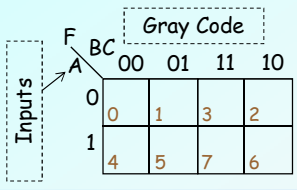


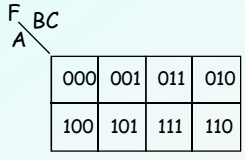
or

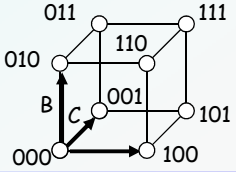


Cells of the table are ordered according to the principles of **Gray** code in which only one variable changes in between adjacent (horizontal and vertical) squares.

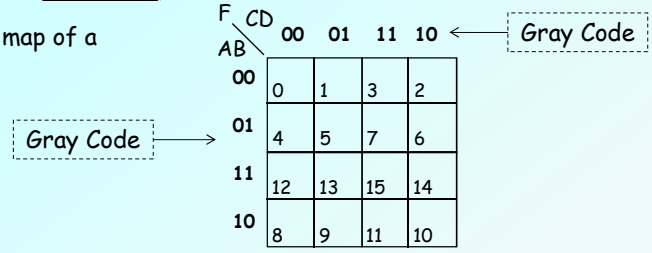
Format of the Karnaugh map of a function with 3 inputs:





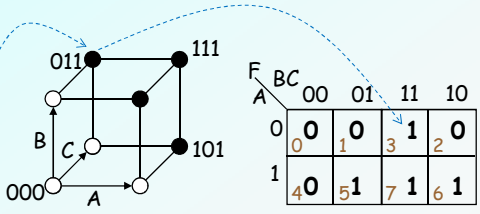


Format of the Karnaugh map of a function with 4 inputs:



Example: The Karnaugh map of a function with 3 inputs :

No	A	B	C	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1



Karnaugh maps will be used in further lessons to simplify Boolean functions.

Algebraic Representation (Expressions) and Canonical Forms

A real-world digital problem can be specified by a truth table.

For example; assume that input variable A represents that the door of a car is open, B represents that the key is inserted, then the truth table can specify the action Z to be taken, where Z=1 means that the alarm sounds.

Num.	A	B	Z
0	0	0	0
1	0	1	0
2	1	0	0
3	1	1	1

Truth tables of real-world digital problems are more complicated.

To handle digital problems and implement the solutions using logical gates, we need to obtain the **expressions** of the related Boolean function.

Logical expressions of the Boolean functions can be obtained in **canonical forms** from their truth tables.

There are two types of canonical forms:

- 1st Canonical form: SOP (**ΣΠ**) form.

Consists of a sum of products each correspond to a "1"-generating combination.

- 2nd Canonical form: POS (**ΠΣ**) form.

Consists of a product of sums each correspond to a "0"-generating combination.

1st Canonical Form: Sum of Products

- The 1st canonical form consists of a sum of product terms each correspond to a row in the truth table with the output "1".
- For a Boolean function of n variables, a product term in which each of the n variables appears once (in either its complemented or uncomplemented form) is called a **minterm**.
- For example a function with 3 variables (a, b, c) has 8 minterms:

  a'b'c', a'b'c, a'bc', a'bc, ab'c', ab'c, abc', abc

- Each minterm covers only one "true" row of the truth table.
- The 1st canonical form of a function is the sum of minterms.
- To find the minterms,
  - Locate all rows in the truth table where output is "1".
  - To generate the individual minterms, substitute variables (for example A, B or C) for ones (of the inputs) and complements of variables (A', B', or C') for zeros in the truth table. (See the example in 2.28)
- A Boolean function may have many logical expressions. They produce the same result given the same inputs.
- But the expression in 1st canonical (standard) form is unique.

Example:

"True" (1) combinations: F = 001 011 101 110 111
Sum of Minters: F = A'B'C + A'BC + AB'C + ABC' + ABC

A	B	C	F	F'
0	0	0	0	1
0	0	1	1	0
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	0

The 1st canonical form of the complement of the function can be obtained similarly by considering the "false" (0) rows:

F' = A'B'C' + A'BC' + AB'C'

Indexing minterms:

We assign each minterm an index (number) based on binary encoding of the variables.

For example, we assign the index 5 to the minterm ab'c (101) and denote that minterm as m5.

A	B	C	Minterms
0	0	0	A'B'C' m0
0	0	1	A'B'C m1
0	1	0	A'BC' m2
0	1	1	A'BC m3
1	0	0	AB'C' m4
1	0	1	AB'C m5
1	1	0	ABC' m6
1	1	1	ABC m7

Representation of minterms with 3 variables

Expression of function F in (2.28) in 1st canonical form:

$F(A, B, C) = \Sigma m(1,3,5,6,7)$

$= m1 + m3 + m5 + m6 + m7$

$= A'B'C + A'BC + AB'C + ABC' + ABC$

$F = \Sigma_{A, B, C}(1,3,5,6,7)$ (Sum of minterms)

Canonical forms are usually not the simplest (optimal) algebraic expression of the function.

The can be usually simplified.

Simplification:

$F(A, B, C) = A'B'C + A'BC + AB'C + ABC + ABC'$

$= (A'B' + A'B + AB' + AB)C + ABC'$

$= ((A' + A)(B' + B))C + ABC'$

$= C + ABC'$

$= ABC' + C$

$= AB + C$

2nd Canonical Form: Product of Sums

- The 2nd canonical form consists of a product of sum terms each correspond to a row in the truth table with the output "0".
- For a Boolean function of n variables, a sum term in which each of the n variables appears once (in either its complemented or uncomplemented form) is called a **maxterm**.
- For example a function with 3 variables (a, b, c) has 8 maxterms:

  a+b+c, a+b+c', a+b'+c, a+b'+c', a'+b+c, a'+b+c', a'+b'+c, a'+b'+c'
- Each maxterm is "false" for exactly one combination of inputs.
- The 2nd canonical form of a function is the product of maxterms.
- To find the maxterms,
  - Locate all rows in the truth table where output is "0".
  - To generate the individual maxterms, substitute variables (for example A, B, or C) for zeros (of the inputs) and complements of variables (A', B', or C') for ones in the truth table.
- Each Boolean function has a unique 2nd canonical (standard) form.

Example:

"False" (0) generating combinations: F = *000* *010* *100*

Product of maxterms: F = (*A* + B + *C*) (*A* + B' + *C*) (*A*' + B + *C*)

A	B	*C*	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

The 2nd canonical form of the complement of the function can be obtained similarly by considering the "true" (1) rows:

F' = (A + B + C') (A + B' + C') (A' + B + C') (A' + B' + C) (A' + B' + C')

Indexing maxterms:

We assign each maxterm an index (number) based on binary encoding of the variables. For example, we assign the index 5 to the maxterm a'+b+c' (101) and denote that maxterm as M5.

Canonical forms are usually not the simplest (optimal) expression of the function. The can be usually simplified.

A	B	*C*	maxterms	
0	0	0	*A*+B+*C*	M0
0	0	1	*A*+B+*C*'	M1
0	1	0	*A*+B'+*C*	M2
0	1	1	*A*+B'+*C*'	M3
1	0	0	*A*'+B+*C*	M4
1	0	1	*A*'+B+*C*'	M5
1	1	0	*A*'+B'+*C*	M6
1	1	1	*A*'+B'+*C*'	M7

Representation of maxterms with 3 variables

Example: 2nd canonical form of F:

F(A, B, C) = ΠM(0,2,4)
= M0 • M2 • M4
= (A + B + C) (A + B' + C) (A' + B + C)

F = $\Pi_{A,B,C}$(0,2,4) product of maxterms.

Minimization:

F(A, B, C) = (A+B+C) (A+B'+C) (A'+B+C)
= (A+C)(B+B') (A'+B+C)
= (A+C) (A'+B+C)
= (A+C) (A'+B+C) (B+C) (consensus)
= (A + C) (B + C)

Conversions Between Canonical Forms

- From 1st (sum of minterms) form to 2nd (product to maxterms) form
  - Replace the minterms with maxterms, and assign them numbers of minterms that don't appear in the 1st canonical form.
  - F(A,B,C) = Σm(1,3,5,6,7) = ΠM(0,2,4)
- From 2nd (product to maxterms) form to 1st (sum of minterms) form
  - Replace the maxterms with minterms, and assign them numbers of maxterms that don't appear in the 2nd canonical form.
  - F(A,B,C) = ΠM(0,2,4) = Σm(1,3,5,6,7)
- Finding the complement of the function in sum of minterms form
  - Select the minterms that don't appear in the 1st canonical form.
  - F(A,B,C) = Σm(1,3,5,6,7) F'(A,B,C) = Σm(0,2,4)
- Finding the complement of the function in product of maxterms form
  - Select the maxterms that don't appear in the 2nd canonical form.
  - F(A,B,C) = ΠM(0,2,4) F'(A,B,C) = ΠM(1,3,5,6,7)

Canonical Forms and the De Morgan Theorem

Applying the De Morgan theorem to the complement of the function in the 1st canonical form generates the expression in the 2nd canonical form.

- Complement of the function in SOP form

  F' = A'B'C' + A'BC' + AB'C'
  - De Morgan

  (F')' = (A'B'C' + A'BC' + AB'C')'

  F = (A + B + C) (A + B' + C) (A' + B + C) 2nd canonical form is obtained.

Applying the De Morgan theorem to the complement of the function in the 2nd canonical form generates the expression in the 1st canonical form.

- Complement of the function in POS form

  F' = (A + B + C') (A + B' + C') (A' + B + C') (A' + B' + C') (A' + B' + C')
  - De Morgan

  (F')' = ((A + B + C')(A + B' + C')(A' + B + C')(A' + B' + C)(A' + B' + C'))'

  F = A'B'C + A'BC + AB'C + ABC' + ABC 1st canonical form is obtained.