Reading 06

7.1 Basic Properties of Boolean Algebra

- Basic Properties of Boolean algebra:

Table 7.1.1: A few basic properties of Boolean algebra.

Property	Name	Description
a(b + c) = ab + ac	***Distributive (for AND)***	Same as multiplication in regular algebra
a + a' = 1	***Complement***	Clearly one of a, a' must be 1 1 + 0 = 1 0 + 1 = 1
a · 1 = a	***Identity***	Result of a · 1 is always a's value *0* · 1 = *0* *1* · 1 = *1*

- More Properties!

Table 7.1.2: More properties.

Property	Name	Description
ab = ba	***Commutative (for AND)***	Same as multiplication for regular algebra
a + b = b + a	***Commutative (for OR)***	Same as addition for regular algebra
a + 1 = *1*	***Null elements***	OR only needs one 1 to evaluate to 1 a = 0 0 + 1 = *1* a = 1 1 + 1 = *1*
a + a = a aa = a	***Idempotent***	0 + 0 = 0 1 + 1 = 1 0 · 0 = 0 1 · 1 = 1

- More Properties!

Table 7.1.3: Commonly used basic properties of Boolean algebra.

Property	Name	Description
a(b + c) = ab + ac a + (bc) = (a + b)(a + c)	Distributive (AND) Distributive (OR)	(AND) Same as multiplication in regular algebra (OR) Different from regular algebra
ab = ba a + b = b + a	Commutative	Variable order does not matter. Good practice is to sort variables alphabetically.
(ab)c = a(bc) (a + b) + c = a + (b + c)	Associative	Same as regular algebra
aa' = 0 a + a' = 1	Complement (AND) Complement (OR)	(AND) Clearly one of a, a' must be 0 1 · 0 = 0 · 1 = 0 (OR) Clearly one of a, a' must be 1: 1 + 0 = 0 + 1 = 1.
a · 1 = a a + 0 = a	Identity (AND) Identity (OR)	(AND) Result of a · 1 is always a's value: *0* · 1 = *0* *1* · 1 = *1*. (OR) Result of a + 0 is always a's value: *0* + 0 = *0* *1* + 0 = *1*.
a · 0 = 0 a + 1 = 1	Null elements	Result doesn't depend on the value of a.
a · a = a a + a = a	Idempotent	Duplicate values can be removed.
(a')' = a	Involution	(0')' = (1)' = 0 (1')' = (0)' = 1
(ab)' = a' + b' (a + b)' = a'b'	DeMorgan's law	*Discussed in another section*

7.2 DeMorgan's Law

- DeMorgan's law, a Boolean algebra property for contemplating an expression, comes in 2 forms:
  1. $(a+b)' = a'b'$
  2. $(ab)' = a' + b'$

- Each literal is complemented
- the AND/ORs swap

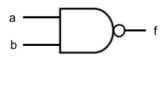
- $y = (ab)'$ is NOT in sum-of-products form
- $y = a' + b'$ IS in sum-of-products form.

7.3 NAND / NOR (universal gates)

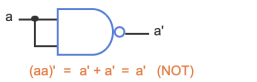
- A NAND gate is the opposite (the NOT, hence the "N") of an AND gate, outputting a 0 if all inputs are 1s; otherwise the output is 1.

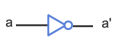
Figure 7.3.1: NAND truth table and gate.

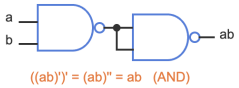
a	b	f
0	0	1
0	1	1
1	0	1
1	1	0

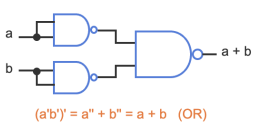


- NAND gates are universal gates.
- A universal gate is a single gate type that can implement any combinational circuit.
  - ↳ NAND can implement NOT, AND, and OR as shown below:







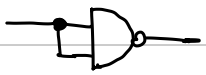


Because NAND can implement NOT, AND, and OR, any circuit can be implemented with just NANDs. NAND is thus a universal gate.

- A chip with prefabricated gates is called a gate-array ASIC. ASIC is short for application-specific integrated circuit.

- Converting to NAND gates:

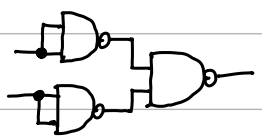
  ↳ Tying together the inputs of a NAND gate is equivalent to a NOT gate.

  ↳ 

  ↳ Inverting the output of a NAND gate is equivalent to an AND gate.

  ↳ 

  ↳ Inverting the inputs of a NAND gate is equivalent to an OR gate.

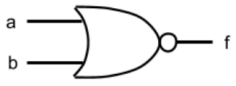
  ↳ 

- A NOR gate is the opposite of an OR gate, outputting 0 if any of the inputs are 1s; otherwise the output is 1.

  ↳ NOR is also a universal gate

Figure 7.3.2: NOR truth table and gate.

a	b	f
0	0	1
0	1	0
1	0	0
1	1	0



7.5 Sum-of-Products Form

- A product term consists of one or more variables, like ab'c, joined by AND.
- An expression in sum-of-products form consists of product terms joined by OR, such as ab'c' + ab

- Because similarities with regular algebra, convention uses the word "product" for AND and "sum" for OR; hence the name "sum-of-products"

- Converting sum-of-products to a circuit:

  - A sum-of-products equation can be easily converted to a two-level circuit, consisting of a column of AND gates (one gate per product term), followed by an OR gate (the NOT gates preceding the AND gates aren't considered a level.)

- A processor executes computer programs.
  - Devices such as keyboards or USB ports surrounding a processor may request the processor to execute a sub-program on behalf of that device, a request known as an interrupt.

7.6 Sum-of-minterms form

- A canonical form of a Boolean equation is a standard equation form for a function.
- Sum-of-minterms form is a canonical form of a Boolean equation where the right-side expression is a sum of products with each product a unique minterm.
- A minterm is a product term having exactly one literal for every function variable.
- A literal is a variable appearance, in true or complemented form, in an expression, such as b or b'.

Transforming to sum-of-minterms

- A sum-of-products equation can be transformed to sum-of-minterms form by multiplying each product term by $(v + v')$ for any missing variable v to create the minterm (removing redundant minterms)
  - ↳ Note: $(v + v')$ is 1.

Ex Equation → Sum-of-Minterms form

$y = (a+c)b$

$y = ac + bc$

$y = ab(1) + bc(1)$

$y = ab(c+c') + bc(a+a')$

$y = abc + abc' + abc + a'bc$

$y = a'bc + abc' + abc$

- Because sum-of-minterms form is canonical, you can put two different equations into sum-of-minterms form to see if they represent the same function.

- A compact function notation represents each minterm by a number.
  - Given that $ab'c$ is 1 if $a/b/c$ are $1/0/1$, that minterm is represented as m_5 because 101 in binary is 5 in decimal.

Ex: $a'bc' = m2$ because $a'bc'$ is 1 when $a/b/c$ are $0/1/0$. $010_2 \rightarrow 2_{10}$

Ex: $abc' = m6$ because abc' is 1 when $a/b/c$ is $1/1/0$. $110_2 \rightarrow 6_{10}$

Ex $f(a,b,c) = ab$

→ $ab(c+c')$

→ $abc + abc'$

→ $m7 + m6$

→ $m6 + m7$ (since minterms are usually in ascending order)