Boolean Algebra and Logic Gates

CS207 Chapter 2

James YU
yujq3@sustech.edu.cn

Department of Computer Science and Engineering
Southern University of Science and Technology

Sept. 14, 2022

Boolean Algebra

- The previous binary logic is *two-valued Boolean algebra*.
  - On a set of two elements: 0 and 1.
  - With rules for the three binary operators: $+$, $\cdot$ and $'$.

$\begin{cases} + : \text{OR} \\ \cdot : \text{AND} \end{cases}$

binary logic

- Common properties:

and

  - $A + 0 = A$ and $A \cdot 1 = A$.
  - $A + 1 = 1$ and $A \cdot 0 = 0$.
  - $A + A' = 1$ and $A \cdot A' = 0$.
  - $A + A = A$ and $A \cdot A = A$.
  - $(A')' = A$.

try to memorize it

a boolean variable: 0/1

Postulates

If you postulate something, you suggest it as the basis for a theory, argument, or calculation, or assume that it is the basis. [formal]

Something that is handy is useful.

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- **Closure**: A set S is closed with respect to a binary operator if, for every pair of elements of S, the binary operator specifies a rule for obtaining a unique element of S.
- **Associative law**: $A + (B + C) = (A + B) + C$ and $A(BC) = (AB)C$.
- **Commutative law**: $A + B = B + A$ and $AB = BA$.
- **Identity element**: A set S is to have an identity element with respect to a binary operation $*$ on S, if there exists an element $E \in S$ with the property $E * A = A * E = A$.
  - Element 0 is an identity element of $+$, and 1 is an identity element of $\cdot$.
- **Distributive law**: $A(B + C) = AB + AC$ and $A + (BC) = (A + B)(A + C)$.
- **DeMorgan**: $(A + B)' = A'B'$ and $(AB)' = A' + B'$.
- **Absorption**: $A + AB = A$ and $A(A + B) = A$.

$* $ is NOT the "AND" but represent All boolean operator

you can prove it. But you can regarded it as the distributive law of "+" operator to "." operator

commonly used

= DeMorgan's rule

a little bit less used

$A + \bar{A}B = A + B$

Distributive Laws: $A(B+C) = AB + AC$

$A + BC = (A+B)(A+C)$

$(A+B)(A+C) = AA + AC + AB + BC$

① $AA = A$

NOT 0 !!!!

$\Rightarrow (A+B)(A+C) = A + AC + AB + BC$

② In Boolean Algebra, the order of precedence is

$\downarrow$
Parenthesis
NOT
AND
OR

③ $A + AB$

$= A(1+B)$

as $1+B = 1$

$A + AB = A$

$\Rightarrow (A+B)(A+C)$

$= AA + AC + AB + BC$

$= A + AC + AB + BC$

$= A + BC$

Absorption Laws: $A + AB = A$

proof: $A + AB = A(1 + B)$

$= A$

$A + \bar{A}B = A + B$

proof: $A + \bar{A}B = A + (A + B')'$

$= [A' \cdot (A + B')'']'$

$= [A'(A + B')]'$

$= [AA' + A'B']'$

$= [A'B']'$

$= A + B$

Duality property

A duality is a situation in which two opposite ideas or feelings exist at the same time.

If you deduce something or deduce that something is true, you reach that conclusion because of other things that you know to be true.

- Every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged.
- Change + to · and vice versa.
- Change 0 to 1 and vice versa.
  - $A + A' = 1 \rightarrow A \cdot A' = 0.$
  - $A + B = B + A \rightarrow AB = BA.$
  - $A(B + C) = AB + AC \rightarrow A + BC = (A + B)(A + C).$
  - $(A + B)' = A'B' \rightarrow (AB)' = A' + B'.$

Vice versa is used to indicate that the reverse of what you have said is true.

A vice is a habit which is regarded as a weakness in someone's character, but not usually as a serious fault.

not limits

4

??? = ???

n = n

Boolean function

- Binary variables have two values, either 0 or 1.
- A Boolean function is an expression formed with *binary variables*, the two *binary operators* **AND** and **OR**, one *unary operator* **NOT**, *parentheses* and *equal sign*.
- The value of a function may be 0 or 1, depending on the values of variables present in the Boolean function or expression.
- Example: $F = AB'C$. 3 literals
  - $F = 1$ when $A = C = 1$ and $B = 0$,
  - otherwise $F = 0$.

A AND $\overset{\text{NOT}}{(B)}$ AND C

Boolean function

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Boolean functions can also be represented by truth tables.
  - Tabular form of the values of a Boolean function according to the all possible values of its variables.
    (of data) consisting of or presented in columns or tables
- n number of variables $\rightarrow 2^n$ combinations of 1's and 0's
- One column representing function values according to the different combinations.
- Example: $F = AB + C$.

Independent, simple components of a logical statement are represented by either lowercase or capital letter variables. These variables are "independent" in that each variable can be either true or false independently of the others, and a truth table is a chart of all of the possibilities.

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

header

A header is text such as a name or a page number that can be automatically displayed at the top of each page of a printed document. Compare footer.

main body

Boolean function simplification

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- A Boolean function from an algebraic expression can be realized to a logic diagram composed of logic gates.
- Minimization of the number of literals and the number of terms leads to less complex circuits as well as less number of gates.
  - We first try use postulates and theorems of Boolean algebra to simplify.

现实意义

term: inside it, A. B is literals

6 literals
3 terms

$$F = AB + BC + B'C$$
$$= AB + C\underline{(B + B')}$$
$$= AB + C$$

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

$$F = XYZ + XY'Z + XYZ'$$
$$= XZ(Y + Y') + XY(Z + Z')$$
$$= XZ + XY = X(Y + Z)$$

- Each Boolean function has one representation in truth table, but a variety of ways in algebraic form.

if there are 100/1000 variable?

$\Rightarrow$ using computer to simplification

Algebraic manipulation = boolean function simplification

- Reduce the total number of terms and literals. object
- Usually not possible by hand for complex functions, use computer minimization program. eg. vivado
- More advanced techniques in the next lectures.

If you manipulate something that requires skill, such as a complicated piece of equipment or a difficult idea, you operate it or process it.

If you say that someone manipulates people, you disapprove of them because they skilfully force or persuade people to do what they want.

Boolean function complement

0→1 / 1→0 change the value

- Complement a Boolean function from F to F'.
  - Change 0's to 1's and vice versa in the truth table.
  - Use Use DeMorgan's theorem for multiple variables.
- Example: $F = x'yz' + x'y'z$.

prime

consisting of two parts, elements, or aspects:

Complement:

$$F' = (x'yz' + x'y'z)'$$
$$= (x'yz')'(x'y'z)'$$
$$= (x + y' + z)(x + y + z')$$

use de Morgan's Rule twice

Dual:

$$F^* = (x' + y + z')(x' + y' + z)$$

sum of products

product of summation

only in these 2 situation CAN we determin literals

Canonical forms

- Logical functions are generally expressed in terms of different combinations of logical variables with their true forms as well as the complement forms: x and x'.

  x x' true form

- An arbitrary logic function can be expressed in the following forms, called *canonical forms*:
  - *Sum of products* (SOP), and
  - *Product of sums* (POS).
- What are the products and sums?

If something has canonical status, it is accepted as having all the qualities that a thing of its kind should have.

Canonical forms

If one thing corresponds to another, there is a close similarity or connection between them. You can also say that two things correspond.

- The logical product of several variables on which a function depends is considered to be a product term.
  - Called minterms when all variables are involved: For x and y, xy, $x'y$, xy', and $x'y'$ are all the minterms.

special product term

a/a' b/b' c/c'

correspond

eg. 3 variable a.b.c : 8 minterm

- The logical sum of several variables on which a function depends is considered to be a sum term.
  - Called maxterms when all variables are involved: For x and y, $x + y$, $x' + y$, $x + y'$, and $x' + y'$ are all the maxterms.

sumterm

special sum term

- **SOP**: The logical sum of two or more logical product terms is referred to as a sum of products expression. $\longleftrightarrow$ sum of minterm (canonical SOP)
- **POS**: The logical product of two or more logical sum terms is referred to as a product of sums expression.

In **Sum of Products** (what you call ANDs) only one of the minterms must be true for the expression to be true. In **Product of Sums** (what you call ORs) all the maxterms must be true for the expression to be true.

In **Sum Of Products** (**SOP**), each term of the SOP expression is called a **"minterm"** because,

say, an **SOP** expression is given as: *F(X,Y,Z) = X'.Y'.Z + X.Y'.Z' + X.Y'.Z + X.Y.Z*

for this **SOP** expression to be **"1"** or *true* (being a **positive logic**), ***ANY*** *of the term of the expression should be 1. thus the word "minterm".*

i.e, **any** of the term (X'Y'Z) , (XY'Z') , (XY'Z) or (XYZ) being **1**, results in **F(X,Y,Z) to be 1**!! Thus they are called "minterms".

On the other hand, In **Product Of Sum** (**POS**), each term of the POS expression is called a "maxterm" because,

say an **POS** expression is given as: F(X,Y,Z) = (X+Y+Z).(X+Y'+Z).(X+Y'+Z').(X'+Y'+Z)

for this **POS** expression to be "**0**" (because **POS** is considered as a **negative logic** and we consider **0** terms), ***ALL*** *of the terms of the expression should be 0. thus the word "max term"!!*

i.e for **F(X,Y,Z) to be 0**, **each** of the terms (X+Y+Z), (X+Y'+Z), (X+Y'+Z') and (X'+Y'+Z) should be equal to "**0**", otherwise F won't be zero!!

Minterms

If you possess something, you have it or own it.

- In the minterm, a variable will possess the value 1 if it is in true or uncomplemented form, whereas, it contains the value 0 if it is in complemented form.

① in this table, 0 and 1 NOT represent the value of literals, but the FORM of variables.

② it is NOT truth table

A	B	C	Minterm
0	0	0	$A'B'C'$
0	0	1	$A'B'C$
0	1	0	$A'BC'$
0	1	1	$A'BC$
1	0	0	$AB'C'$
1	0	1	$AB'C$
1	1	0	ABC'
1	1	1	ABC

ONLY 0.0.1 can $A'B'C = 1$

- It possesses the value of 1 for only one combination of n input variables
  - The rest of the $2^n - 1$ combinations have the logic value of 0.

Minterms

product

- *Canonical SOP* expression, or *sum of minterms*: A Boolean function expressed as the logical sum of all the minterms from the rows of a truth table with value 1.

F=AB+C 对应 truth table 中 F 所在 column
在 A.B.C 为同一行的值所得的F值
而 Minterms column 写出的式子来自于左侧3列
中 A.B.C 为0时取 complement. 为1时取 tine value
所写出的 minterm 式子.

欲将F用 canonical SOP 去进行表示,
找到 truth table 中 F=1 时 A.B.C 值对应的 minterm
并将其相加即可

$\underbrace{\quad}_{\text{input}}$ A	B	C	F (output)	Minterms
0	0	0	0	$A'B'C'$ $(000)_2=(0)_{10}$
0	0	1	1	$A'B'C$
0	1	0	0	$A'BC'$
0	1	1	1	$A'BC$
1	0	0	0	$AB'C'$
1	0	1	1	$AB'C$
1	1	0	1	ABC'
1	1	1	1	ABC $(111)_2=(7)_{10}$

minterm
represent sum of

- $F = AB + C = A'B'C + A'BC + AB'C + ABC' + ABC = \sum(1,3,5,6,7)$.
  - A compact form by listing the corresponding decimal-equivalent codes of the minterms.

PREFERED form

- 我们写下使F=1的行对应的minterm组成的canonical SOP.
  这个SOP即F的SOP表示.
- why?
  在F=1的行中. 行上literals对应的值使F=1. 同时也使本行的minterm=1.
  且这组literals的值只使本行的minterm=1. 其他行的minterm=0.
  而SOP为sum of product. sum ⇒ "+" ⇒ OR
  由于这些minterm之间为OR. 所以只要有一个minterm=1. 则F=1.
  而在每一组使F=1的literals值中. 都有且仅有唯一minterm=1. 即这组literals对应的minterm.

- 表示minterm的方法
  $F=\Sigma(1,2,\cdots)$
  到第三章终于记起这从哪里来的了.
  注意此处的序号 $\begin{cases}\text{从0开始标} \\ \text{literals的变化类似Gray Code}\end{cases}$

Minterms

- The canonical sum of products form of a logic function can be obtained by using the following procedure.

  To retain something means to continue to have that thing

  1. Check each term in the given logic function. Retain if it is a minterm, continue to examine the next term in the same manner.
  2. Examine for the variables that are missing in each product which is not a minterm.
  3. If the missing variable in the minterm is X, multiply that minterm with $(X + X')$.
     - Example: $A + B \to A(B + B') + B(A + A')$
  4. Multiply all the products and discard the redundant terms.

Minterms

- Example: $F(A,B,C,D) = AB + ACD$.

$$
\begin{aligned}
F(A,B,C,D) &= AB + ACD \\
&= AB(C+C')(D+D') + ACD(B+B') \\
&= (ABC + ABC')(D+D') + ABCD + AB'CD \\
&= ABCD + ABCD' + ABC'D + ABC'D' + ABCD + AB'CD \\
&= ABCD + ABCD' + ABC'D + ABC'D' + AB'CD
\end{aligned}
$$

3 3

① check we have how many variable
double

② rewrite : look up the terms
if is minterm or not : AB: X ACD: X

③ multiply the missing variable with $(x+x')=1$

$x+x$ means
x OR x
$\Rightarrow$ simply it with x

Maxterms

- In the maxterm, a variable will possess the value 0, if it is in true or uncomplemented form, whereas, it contains the value 1, if it is in complemented form.

minterm : 1 ⇒ uncomplement
0 ⇒ complement

A	B	C	Maxterm
0	0	0	$A+B+C$
0	0	1	$A+B+C'$
0	1	0	$A+B'+C$
0	1	1	$A+B'+C'$
1	0	0	$A'+B+C$
1	0	1	$A'+B+C'$
1	1	0	$A'+B'+C$
1	1	1	$A'+B'+C'$

minterm: 1

- It possesses the value of 0 for only one combination of n input variables
  - The rest of the $2^n - 1$ combinations have the logic value of 1.

Maxterms

- *Canonical POS* expression, or *product of maxterms*: A Boolean function expressed as the logical product of all the maxterms from the rows of a truth table with value `0`.

  binary zero   fast   capital sigma

- $F = (A+B+C)(A+B'+C)(\underset{1}{A'}+\underset{0}{B}+\underset{1}{C'}) = \prod(0,2,5)$.
  - A compact form by listing the corresponding decimal-equivalent codes of the maxterms.

Maxterms

- Example: $F(A, B, C, \cancel{D}) = A + B'C$.

$$
\begin{aligned}
F(A, B, C, \cancel{D}) &= A + B'C \\
&= (A + B')(A + C) \\
&= (A + B' + CC')(A + C + BB') \\
&= (A + B' + C)(A + B' + C')(A + B + C)(A + B' + C)
\end{aligned}
$$

no maxterm

the sum of product

rewrite to product form

$xx' = 0$

using the distributive property: $X + YZ = (X + Y)(X + Z)$

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

$[(A+B') + CC']$

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

Derive from a truth table

a row will in either minterm oR maxtem. But NOT in both or in none

A	B	C	F	Minterm (F=1)	Maxterm (F=0)
0	0	0	0		$A+B+C$
0	0	1	0		$A+B+C'$
0	1	0	1	$A'BC'$	
0	1	1	0		$A+B'+C'$
1	0	0	1	$AB'C'$	
1	0	1	1	$AB'C$	
1	1	0	1	ABC'	
1	1	1	0		$A'+B'+C'$

- The final **canonical SOP** for the output F is derived by summing or performing an **OR** operation of the four product terms as shown below:
  - $F = A'BC' + AB'C' + AB'C + ABC' = \sum(2,4,5,6)$.
- The final **canonical POS** for the output F is derived by summing or performing an **AND** operation of the four sum terms as shown below:
  - $F = (A+B+C)(A+B+C')(A+B'+C')(A'+B'+C') = \prod(0,1,3,7)$.

· Minterms & Maxterms

① derive canonical SOP & POS of F

canonical SOP: write the sum of minterm on which row F = 1.

canonical POS: write the product of maxterm on which row F = 0.

why: As SOP is the SUM (OR), so if there exist 1, F = 1. But in OR, if there is a 0, F might NOT equals 0.

i.e. As F is a sum, a 1 can determine its value, but a 0 can't.

But for a POS, a 0 can determine F to be 0.

② minterm : $1 \Rightarrow A \quad 0 \Rightarrow A'$

maxterm : $1 \Rightarrow A' \quad 0 \Rightarrow A$

why: As minterm is the logical product of all variables, it is a PRODUCT.

Conversion between minterms and maxterms

- Minterms are the complement of corresponding maxterms: $m_i = M_i'$.
  - Example: $A' + B' + C' = (ABC)'$.

minterms maxterms

$$
\begin{aligned}
F(A,B,C) &= \sum(2,4,5,6) = m_2 + m_4 + m_5 + m_6 \\
&= A'BC' + AB'C' + AB'C + ABC' \\
F'(A,B,C) &= \sum(0,1,3,7) = m_0 + m_1 + m_3 + m_7 \\
F(A,B,C) &= \left(F'(A,B,C)\right)' = (m_0 + m_1 + m_3 + m_7)' \\
&= m_0' m_1' m_3' m_7' \\
&= M_0 M_1 M_3 M_7 \\
&= \prod(0,1,3,7) \\
&= (A+B+C)(A+B+C')(A+B'+C')(A'+B'+C').
\end{aligned}
$$

eg. $m_0 = A'B'C'$ $M_0 = A+B+C$

$M_0' = A'B'C' = m_0$

$\Rightarrow$ $m_i = M_i'$ exist for ANY i

Other logic operators

- When the binary operators AND and OR are applied on two variables A and B, they form two Boolean functions AB and $A + B$ respectively.

not intuitive

you have an intuitive idea or feeling about something, you feel that it is true although you have no evidence or proof of it.

to human being

Other logic operators

not need to remember all

- When the three operators AND, OR, and NOT are applied on two variables A and B, they form 16 Boolean functions:

x	y	$F = x\ ?\ Y$
0	0	0/1
0	1	
1	0	
1	1	

the answer has $2^4 = 16$ kinds

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y, then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x, then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

Digital logic gates

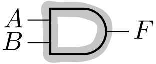
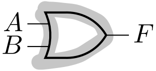
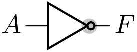
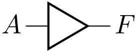
南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- As Boolean functions are expressed in terms of AND, OR, and NOT operations, it is easier to implement the Boolean functions with these basic types of gates.
  - It is possible to construct other types of logic gates.
- The following factors are to be considered for construction of other types of gates.

  the state or degree of being easily or conveniently done:
  - The **feasibility** and economy of producing the gate with physical parameters.
  - The possibility of **extending** to more than two inputs.
  - The basic properties of the binary operator such as **commutability** and **associability**.
  - The ability of the gate to **implement Boolean functions** alone or in conjunction with other gates.

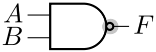
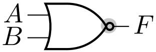
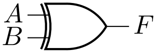
the action or an instance of two or more events or things occurring at the same point in time or space

Digital logic gates

			A	B	F
AND		$F = AB$	0 0 1 1	0 1 0 1	0 0 0 1
OR		$F = A + B$	0 0 1 1	0 1 0 1	0 1 1 1
NOT		$F = A'$	0 1	- -	1 0
Buffer		$F = A$	0 1	- -	0 1

when the line is too long the voltage will drop
Buffer helps to increase the voltage

Digital logic gates

			A	B	F
NAND		$F = (AB)'$	0	0	1
			0	1	1
			1	0	1
			1	1	0
NOR		$F = (A + B)'$	0	0	1
			0	1	0
			1	0	0
			1	1	0
XOR		$F = AB' + A'B$ $= A \oplus B$	0	0	0
			0	1	1
			1	0	1
			1	1	0

make SR latches

NAND. NOR. XOR 中. 仅XOR 有特殊运算符号 $\oplus$

a little expensive to manufacture

also called different checker: only A&B is different it produce 1

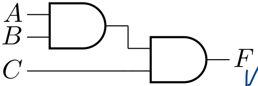
Multiple input logic gates

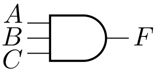
南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

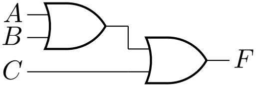
- A gate can be extended to have multiple inputs if its binary operation is commutative and associative.
- AND and OR gates are both commutative and associative.
  - $F = ABC = (AB)C$.
  - $F = A + B + C = (A + B) + C$.

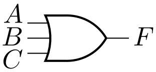
$A+B = B+A$ $\quad$ $(AB)C = A(BC)$

$AB = BA$ $\quad$ $(A+B)+C = A+(B+C)$





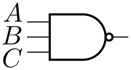




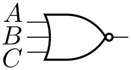
Multiple input logic gates

more commonly used because cheaper

- The NAND and NOR functions are the complements of AND and OR functions respectively. $A+B=B+A$
  - They are commutative, but **not associative**. ⟺ AND & OR are both com & ass
  - $((AB)'C)' \neq (A(BC)')'$: does not follow associativity.
  - $((A+B)'+C)' \neq (A+(B+C)')'$: does not follow associativity.
- We modify the definition of multi-input NAND and NOR:



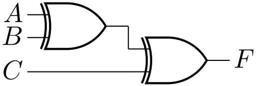
$F = (ABC)' = A' + B' + C'$

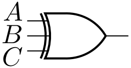


$F = (A+B+C)' = A'B'C'$

Multiple input logic gates

- The XOR gates and equivalence gates both possess commutative and associative properties.
  - Gate output is low when even numbers of 1's are applied to the inputs, and when the number of 1's is odd the output is logic 0.
  - Multiple-input exclusive-OR and equivalence gates are uncommon in practice.





$F = A \oplus B \oplus C$

orpluse / circle pluse

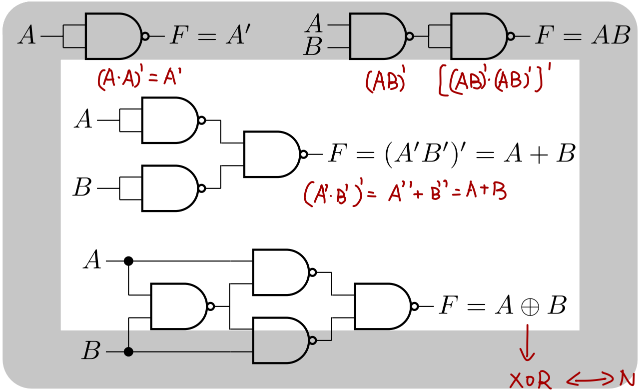
K

Universal gates

If you describe a task or problem as tricky, you mean that it is difficult to do or deal with.

tricky { ① cheap
but used { ② universal

- NAND gates and NOR gates are called *universal gates* or *universal building blocks*.
  - Any type of gates or logic functions can be implemented by these gates.



XOR ⟷ NOR

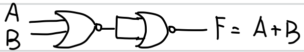
$1 \oplus 1 = 0$ $(1+1)' = 0$

$0 \oplus 0 = 0$ $(0+0)' = 1$

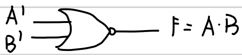
$1 \oplus 0 = 1$ $(1+0)' = 0$

NOR: $(A+B)' = A' \cdot B'$

$A+B$: $[(A+B)' + (A+B)']' = (A+B)'' \cdot (A+B)'' = A+B$

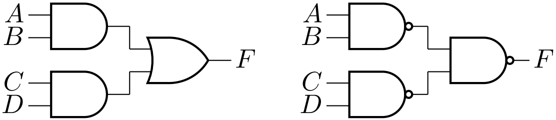


$A \cdot B$: $(A'+B')' = A'' \cdot B'' = A \cdot B$



Universal gates

- Universal gates are easier to fabricate with electronic components.
  - Also reduce the number of varieties of gates.
- Example: $F = AB + CD$ requires two AND and one OR gates.
  - Or three NAND gates.
  - $F = AB + CD = ((AB + CD)')' = ((AB)'(CD)')'$



2 level of gate