

Lecture 2:

Gates, Circuits and Boolean Functions

In this lecture we will:

Represent Boolean functions as Digital Gates.

Define a schematic representation for Digital Gates.

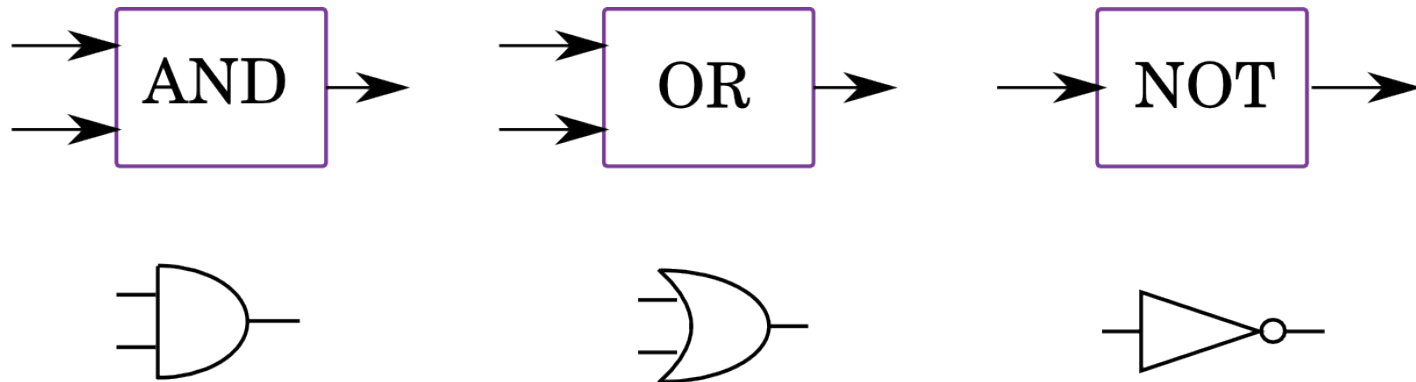
Implement Digital Gates as practical circuits.

Define two more basic Boolean functions (gates): the XOR and XNOR gates.

Introduce the concept of the Control Variable.

Schematic Representation of Boolean Functions

A standard set of easy-to-recognise symbols is used to represent boolean functions.

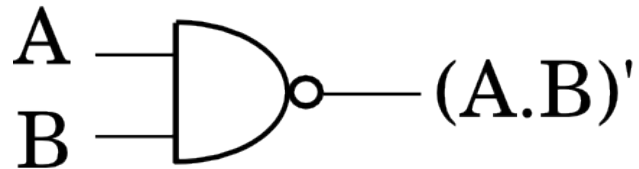


A circle is all that is required to indicate NOT.
The triangle is provided to indicate the input/output direction.

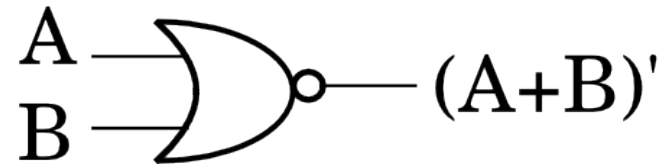
Inverting functions

A circle can be added to the AND and OR symbol outputs to create NAND and NOR gates.

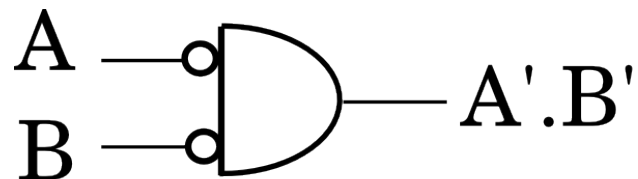
NAND



NOR



Circles can also be placed at the inputs to these gates, but this does not create a recognised Boolean function.



Building blocks for practical circuits

Note that NAND/NOR are commonly used building blocks for most circuits:

NAND / NOR can easily be constructed from transistors as we will see in lecture 5.

NAND is complete

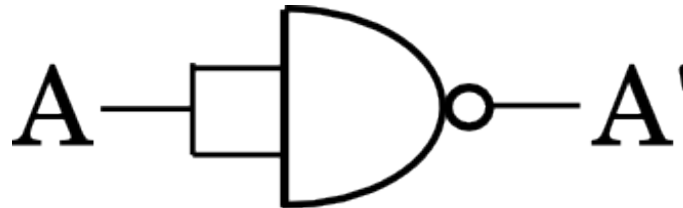
(A set of Boolean functions f_1, f_2, \dots is “complete” if and only if any Boolean function can be generated by a combination these functions.)

The NAND gate is all we need

It is possible to build all other gates out of NAND gates.

We can create a NOT gate using the Idempotent law:

$A.A = A$ therefore $(A.A)' = A'$

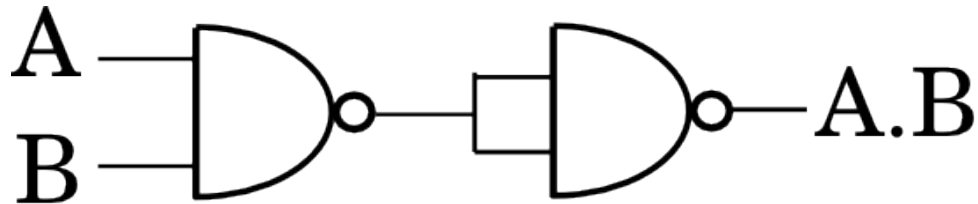


The NAND gate is all we need

To create an AND gate we apply the Involution law,

$$(A')' = A$$

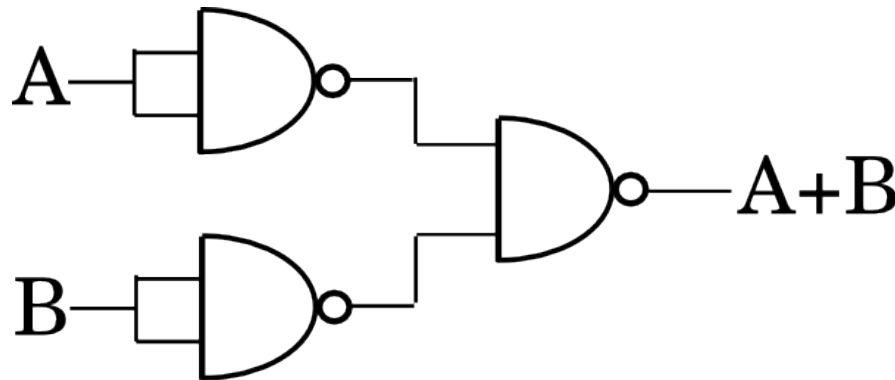
making use of our newly designed inverter:



The NAND gate is all we need

To make an OR gate we need to apply de Morgan's theorem:

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



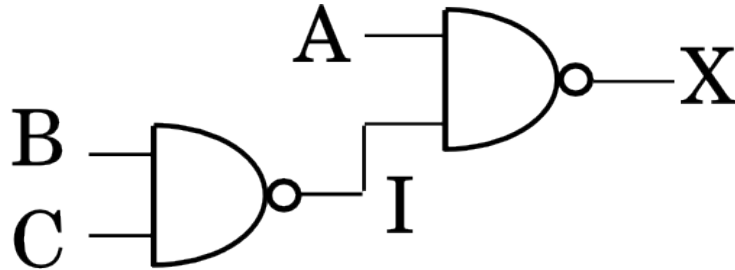
We can just invert the output to make a NOR gate

Problem time

Can you build every gate you need from the NOR gate?

Building more complex circuits

Consider cascading two NAND gates



What circuit have we created? The first approach is to use Boolean Algebra to find out:

$$I = (B.C)'$$

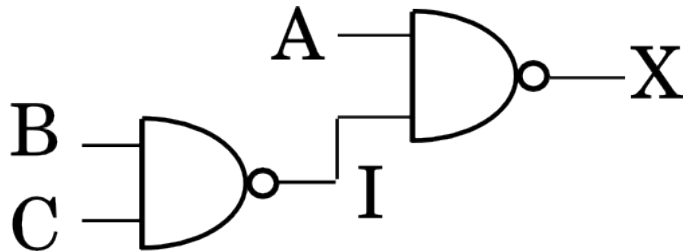
$$X = (A.I)' = (A.(B.C)')'$$

Applying de Morgan we get:

$$X = A' + B.C$$

Building more complex circuits

Alternatively we can construct a truth table.

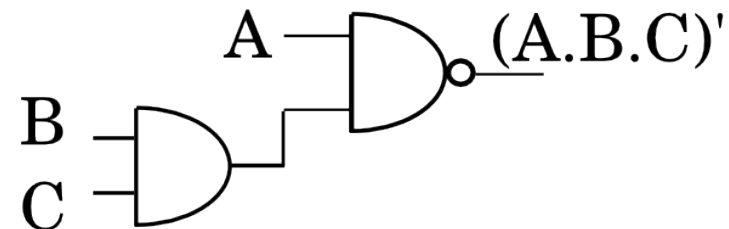
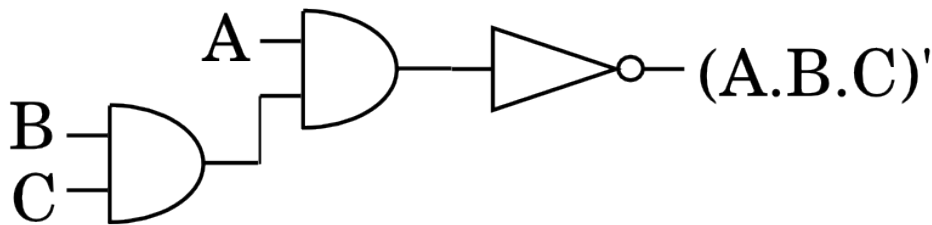


A	B	C	$I = (B \cdot C)'$	$X = (A \cdot I)'$
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Building a three input NAND gate

We can do this by noting how a three input NAND gate is written as a boolean expression:

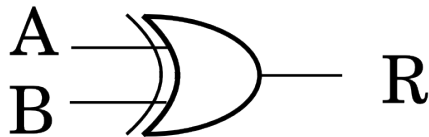
$$X = (A.B.C)' = (A.(B.C))'$$



Two new gates

Here are two new gates that can be very useful

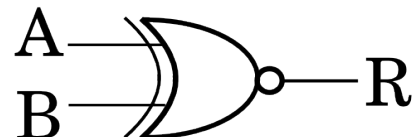
Exclusive Or (XOR)



$$R = A.B' + A'.B$$

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive Nor (XNOR)



$$R = A'.B' + A.B$$

A	B	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

Building an XOR gate from NAND gates

Since the NAND gate is complete we should be able to construct an XOR gate from it.

We can start with the Boolean equation:

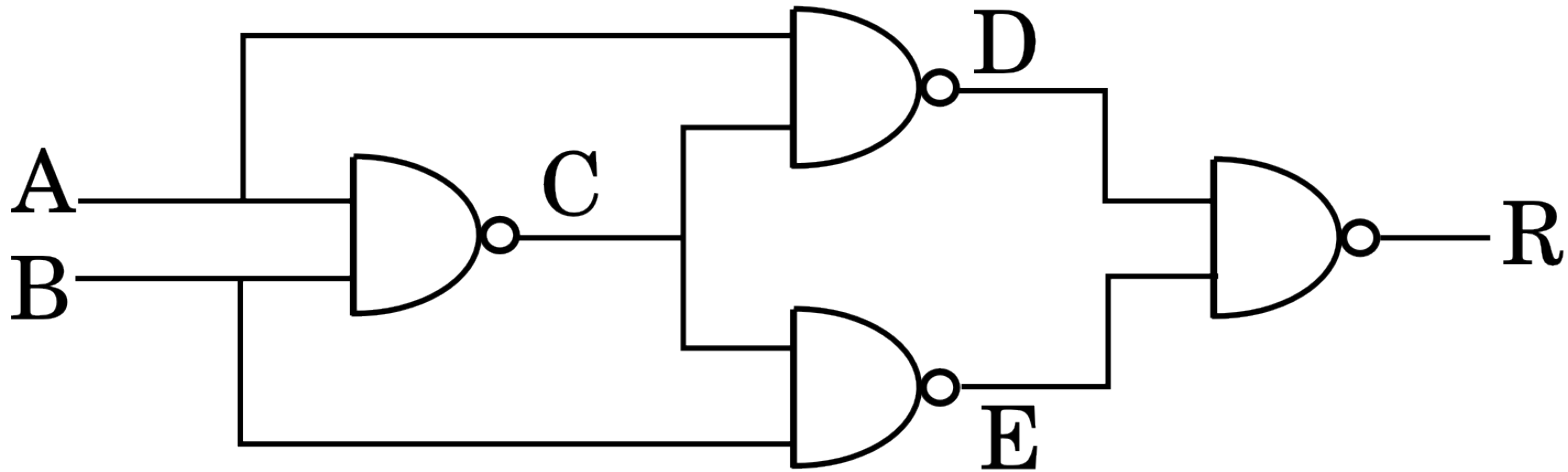
$$A \text{ XOR } B = A'.B + A.B'$$

And use de Morgan's theorem

$$A \text{ XOR } B = ((A'.B)' . (A . B')')'$$

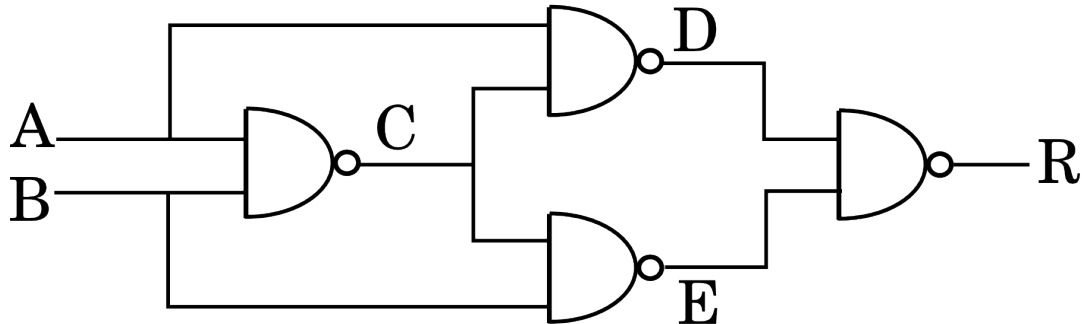
This does the job, but is it the best circuit?

No: this circuit is smaller



Problem - How do we prove that this circuit implements the correct boolean function?

The Logician's solution



$$R = (D \cdot E)' = ((A \cdot C)' \cdot (B \cdot C)')' = ((A \cdot (A \cdot B)')' \cdot (B \cdot (A \cdot B)')')'$$

Apply De Morgan $R = (A \cdot (A \cdot B)') + (B \cdot (A \cdot B)')$

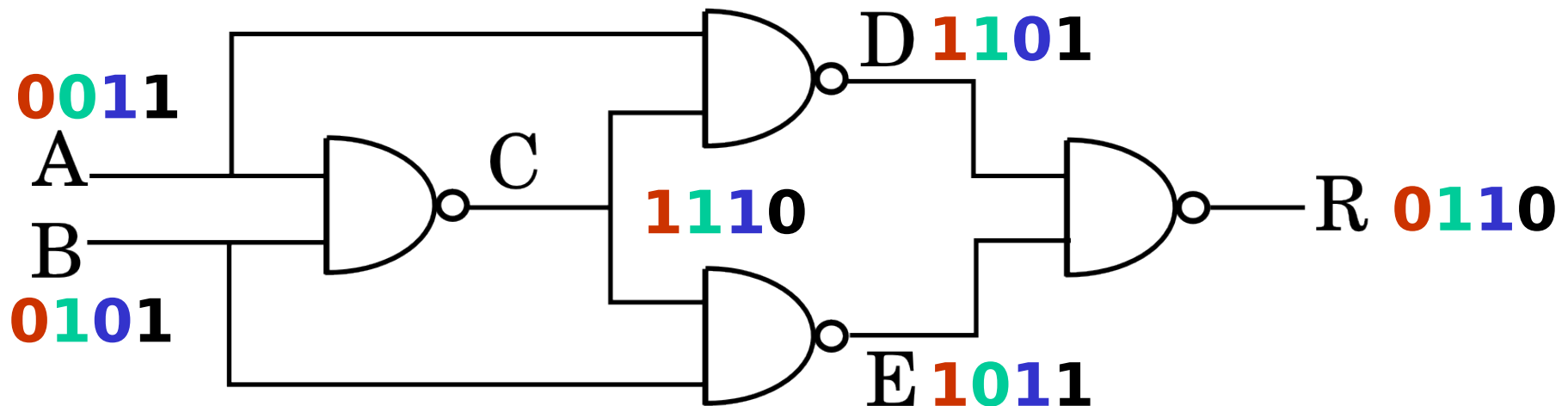
Apply De Morgan $R = A \cdot (A' + B') + B \cdot (A' + B')$

Distributivity $R = A \cdot A' + A \cdot B' + B \cdot A' + B \cdot B'$

Simplify $R = A \cdot B' + B \cdot A'$

The Hardware Engineer's solution

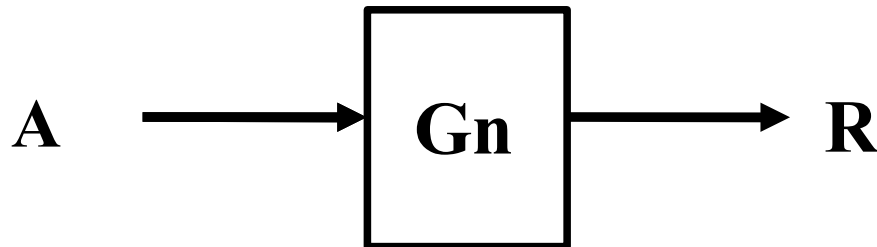
Work round the circuit to construct the input/output truth table, and verify that it is the same as the XOR gate



How many possible gates are there?

So far we have seen ONE one-input gate and SIX two-input gates, but in theory there are 4 possible one-input gates and 16 possible two-input gates.

Are there any more useful one-input gates?



We enumerate all possibilities

A	G0	G1	G2	G3
0	0	0	1	1
1	0	1	0	1

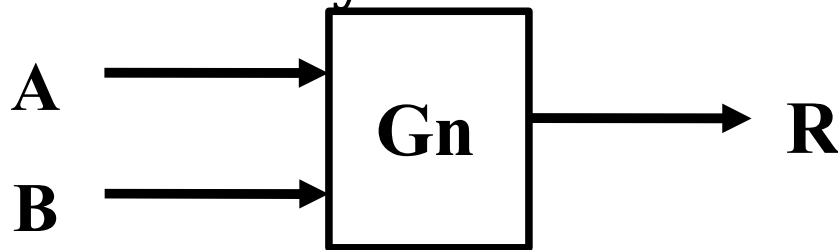
Gate G2 corresponds to the NOT function ie $R2=A'$. None of the others have useful functionality?

	G0	G1	G2	G3
R =	0	A	A'	1

What are the possible 2 input gates?

For two inputs we have 4 possible input values and therefore sixteen possible two input gates.

As before we can enumerate all the possibilities and see what they do.



The sixteen possible two-input gates

AB	G0	G1	G2	G3	G4	G5	G6	G7
00	0	0	0	0	0	0	0	0
01	0	0	0	0	1	1	1	1
10	0	0	1	1	0	0	1	1
11	0	1	0	1	0	1	0	1
R =	0	AND		A		B	XOR	OR

AB	G8	G9	G10	G11	G12	G13	G14	G15
00	1	1	1	1	1	1	1	1
01	0	0	0	0	1	1	1	1
10	0	0	1	1	0	0	1	1
11	0	1	0	1	0	1	0	1
R =	NOR	XNOR	B'		A'		NAND	1

Designing circuits with certain gates only

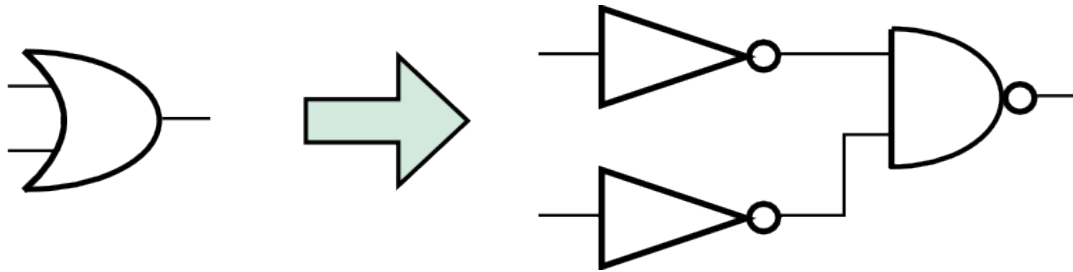
Of all the useful gates, NAND and NOR are simpler and smaller to build and faster in operation.

When designing a circuit to calculate a Boolean function, it then makes sense to try and engineer it by using these gates only.

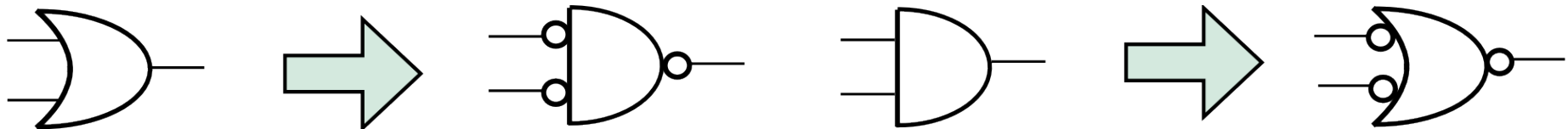
To do that it is helpful to be able to manipulate the schematic circuit diagram.

Graphical Application of de Morgan

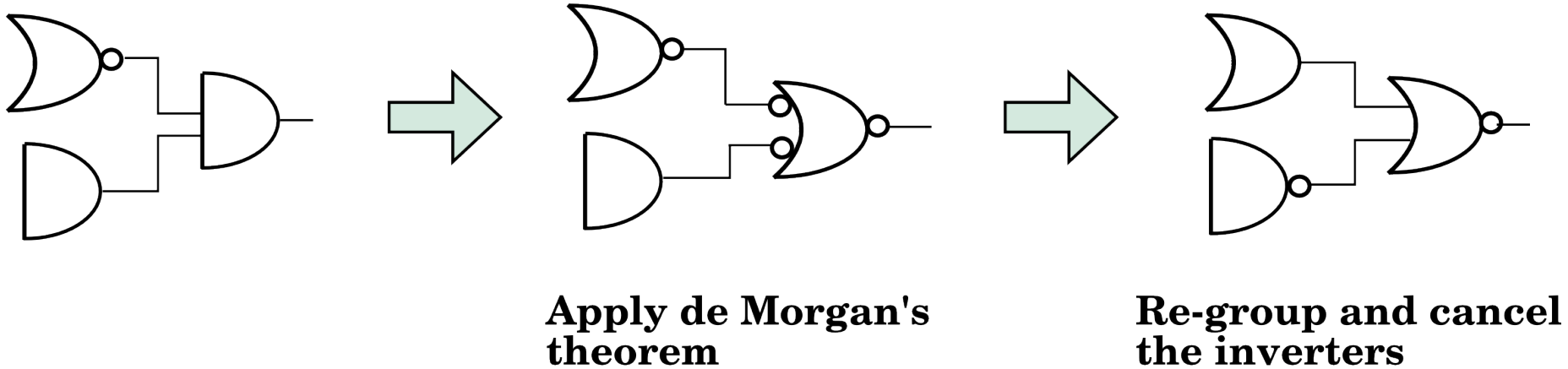
de Morgan's theorem can be applied graphically,
eg $A+B = (A'.B')'$ has the circuit equivalent:



or, just using circles for inversion:



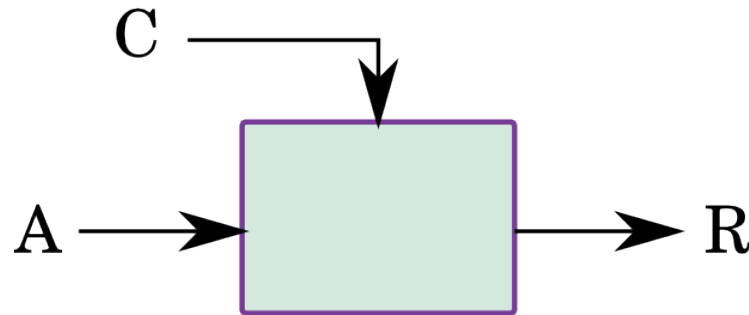
Symbolic circuit manipulation example



The transformation increases the number of NAND/NOR gates by 1.

Control and Data Variables

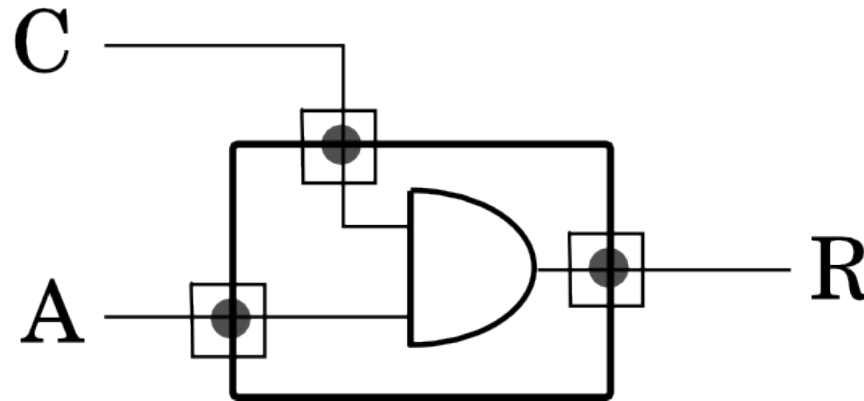
Suppose we want to design a controlled function circuit. For example when $C=0$ the output is 0, and when $C=1$ the output is A



This is the same as a 2 input 1 output gate BUT functionally we interpret it to have: 1 data input, 1 control input and 1 output.

Control and Data Variables

We can construct a truth table from the specification, and we find that it can be implemented with just one AND gate.



This explains why the name gate is used.

An important control circuit

Now consider a circuit with two data inputs A and B and one control input C and one output defined as follows:

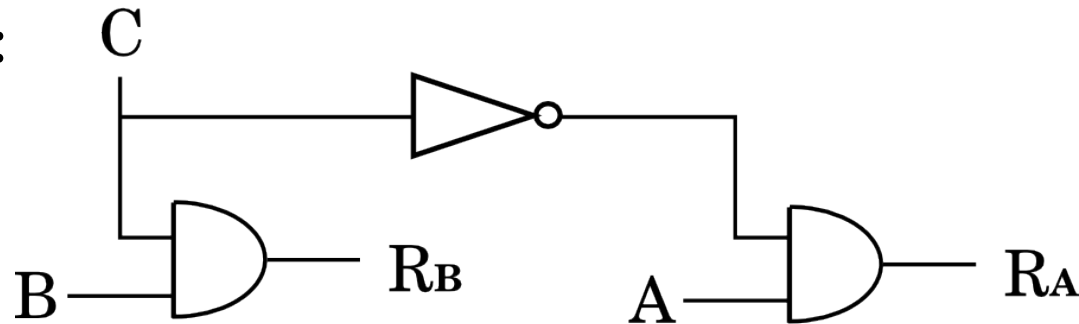
if $C=0$ the output $R=A$

if $C=1$ the output $R=B$

This is in essence a digital switch and is called a multiplexer.

Designing the multiplexer

We use the AND function to gate the A and B inputs:



And combine the two outputs with an OR since $A+0=A$ and $B+0=B$

