Lecture 2

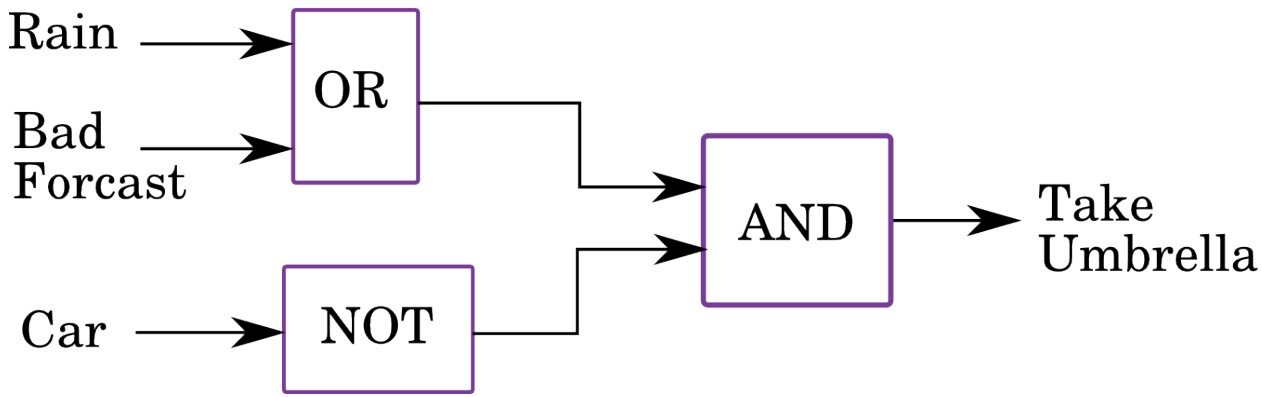
Gates, Circuits and Boolean Functions

In this lecture we will:

- Introduce an electronic representation of Boolean operators called digital gates.
- Define a schematic representation for digital gates.
- Use digital gates to create practical circuits.
- Define two more basic Boolean operators: XOR and XNOR.
- Introduce the concept of the Control Variable in a digital circuit.

Digital Gates

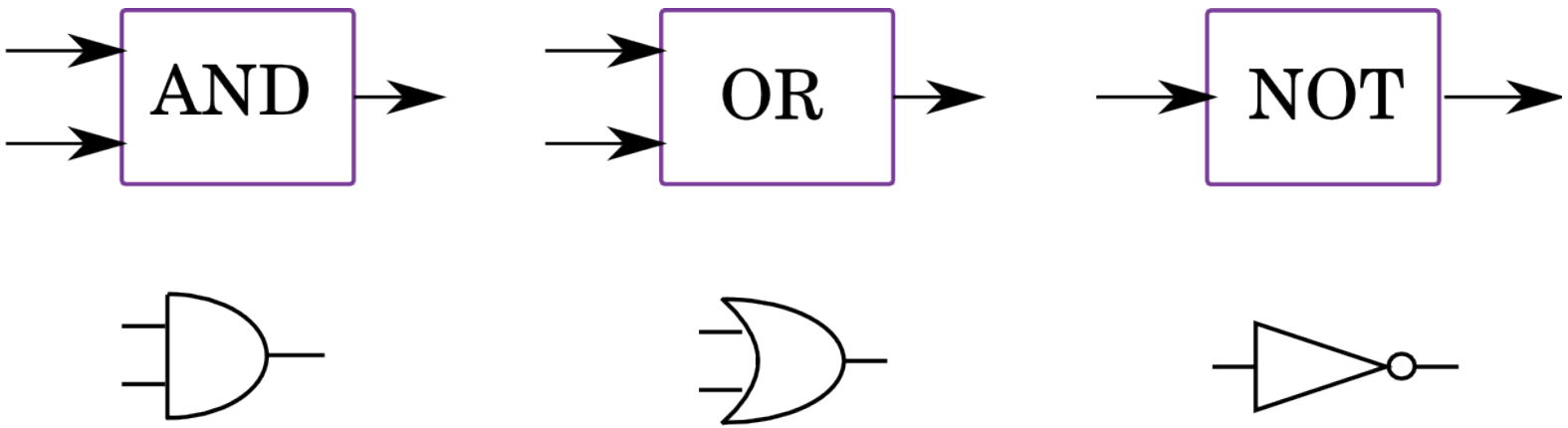
In the last lecture we looked at Boolean algebra from a mathematical point of view, but we also introduced a diagramatic representation of Boolean equations, for example:



This diagramatic representation describes an electronic circuit that will implement the equation, and in the circuit the individual Boolean operators are called digital gates.

Schematic Representation of Boolean Operators

Instead of labelled boxes, a standard set of easy-to-recognise symbols is normally used to represent boolean functions.

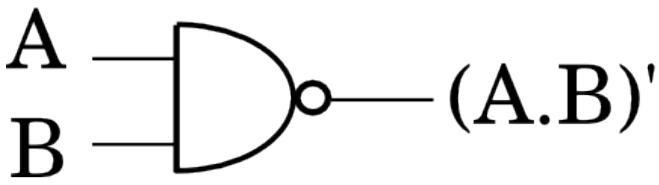


A circle is all that is required to indicate NOT. The triangle is provided to indicate the input/output direction.

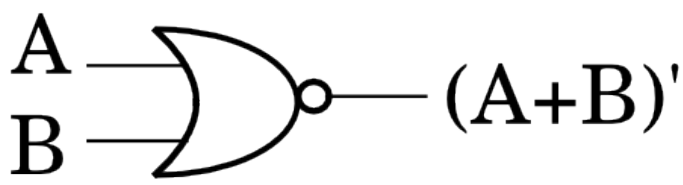
Inverting functions

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

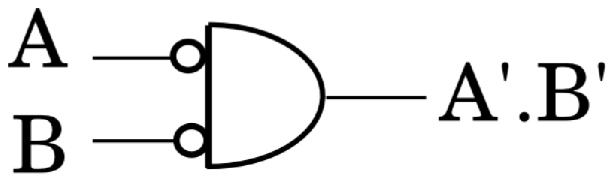
NAND



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

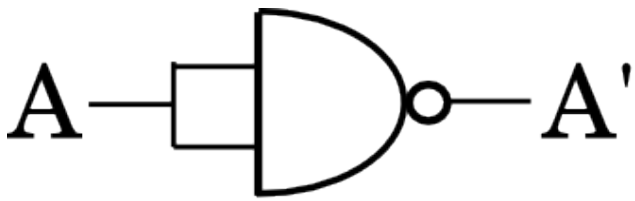
(A set of Boolean functions f1,f2, is complete if and only if any Boolean function can be generated by a combination these functions.)

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 Indempotent law:

$A \bullet A = A$ therefore $(A \bullet A)' = A'$

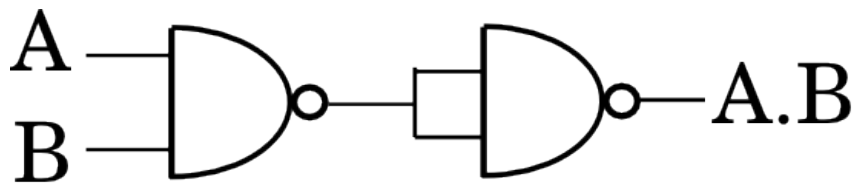


The NAND gate is all we need

To create an AND gate we apply the Involution law:

$$(A')' = A$$

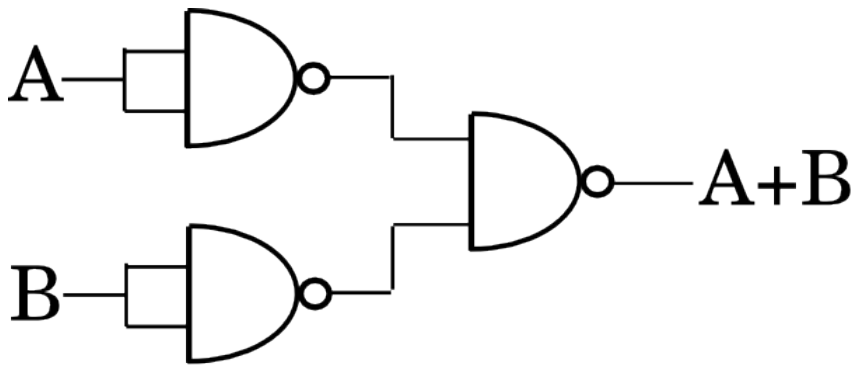
making use of our newly designed inverter:



The NAND gate is all we need

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

$$A+B = (A' \,.\, B')'$$



We can just invert the output to make a NOR gate.

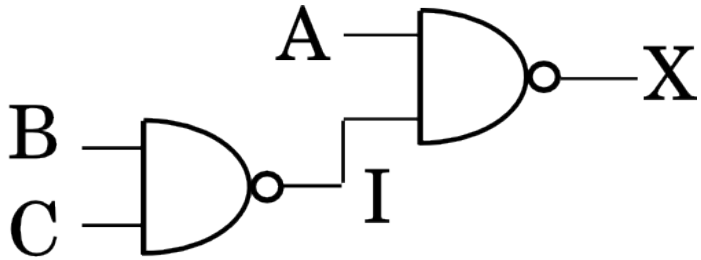
Logicians call NAND the Sheffer Stroke

Problem Time

Can you build every other gate using just the NOR gate?

Building more complex circuits

What happens if we cascade two NAND gates?



We can analyse this circuit using Boolean Algebra:

$I = (B.C)'$

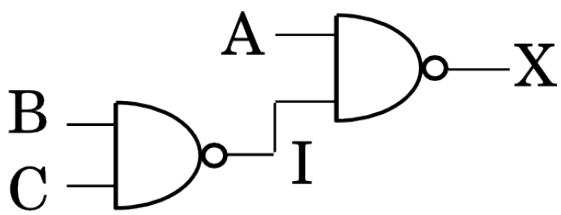
$X = (A.I)' = (A.(B.C)')'$

Applying de Morgan we get:

$X = A' + B.C$

Building more complex circuits

An alternative way of analysing the circuit is to build a truth table:

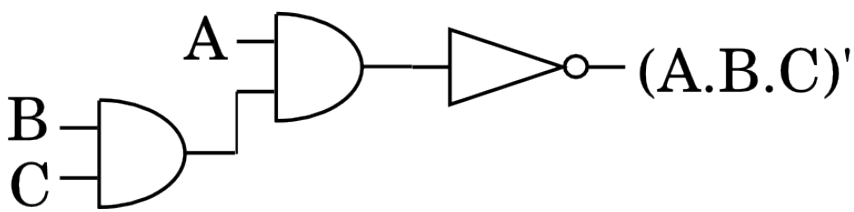


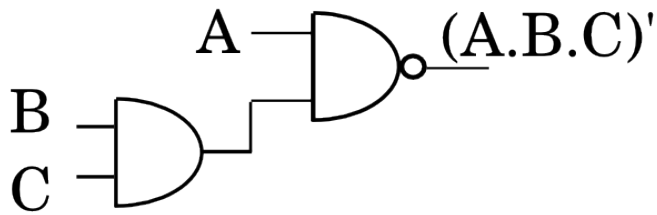
A	B	C	I = $(B \cdot C)'$	X = $(A \cdot I)'$
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Building a three input NAND gate

Cascading two 2-input NAND gates does not do the job, but we can design a 3 input NAND gate using Boolean algebra:

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

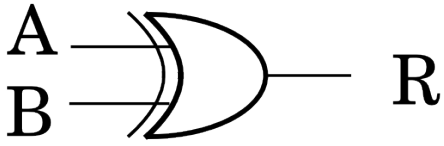




Two new gates

Here we define two new gates which can be very useful:

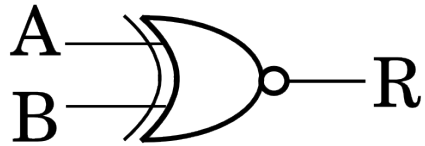
Exclusive Or (XOR)



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

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 using only NANDs.

We can start with the Boolean equation:

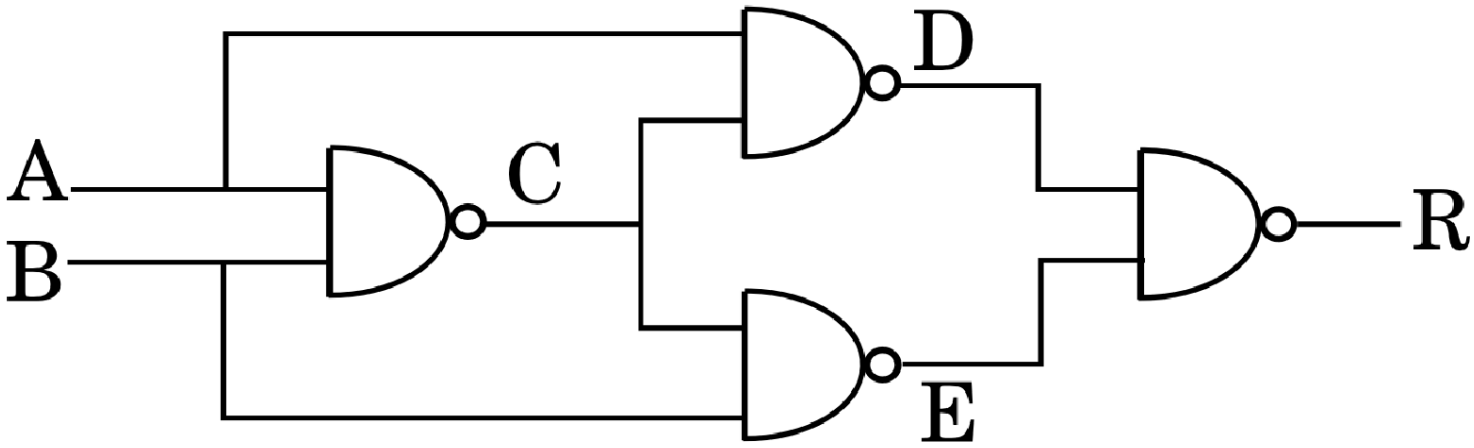
$$A \text{ XOR } B = A'.B + A.B'$$

and use de Morgan's theorem

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

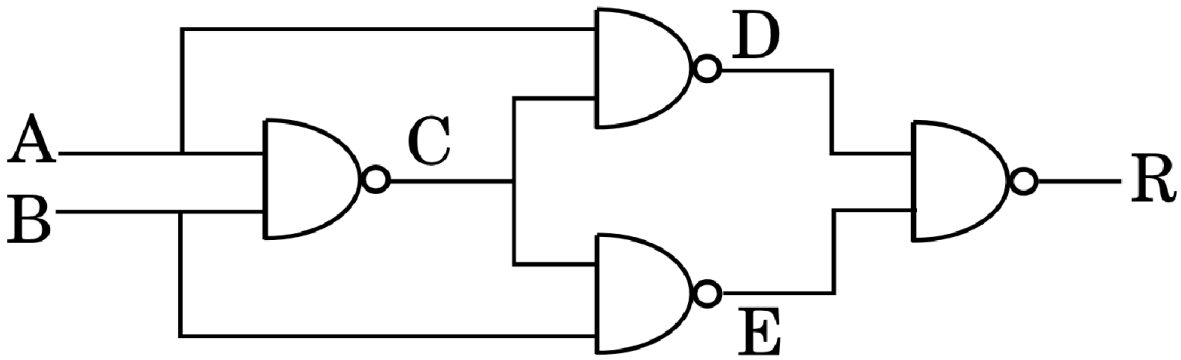
This does the job, but it it the best circuit?

No: this circuit is smaller



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

The Logician's Solution

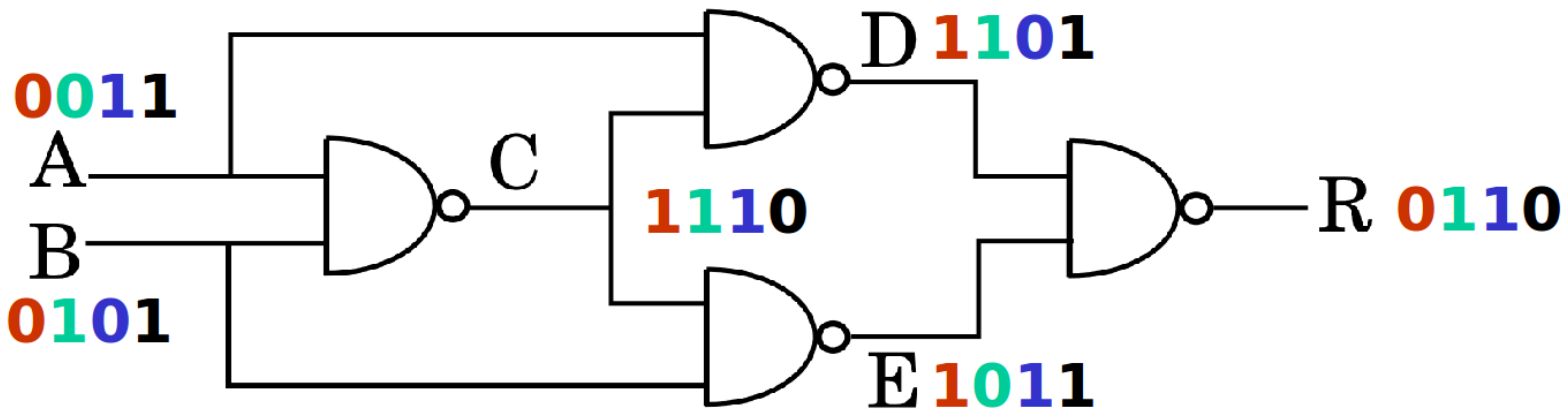


$R = (D.E)' = ((A.C)'.(B.C)')' = ((A.(A.B)')'.(B.(A.B)')')'$

Apply de Morgan	$R = (A.(A.B)') + (B.(A.B)')$
Apply de Morgan	$R = A.(A'+B') + B.(A'+B')$
Distributivity	$R = A.A'+A.B' + B.A'+B.B'$
Simplify	$R = A.B' + B.A'$

The Hardware Engineer's Solution

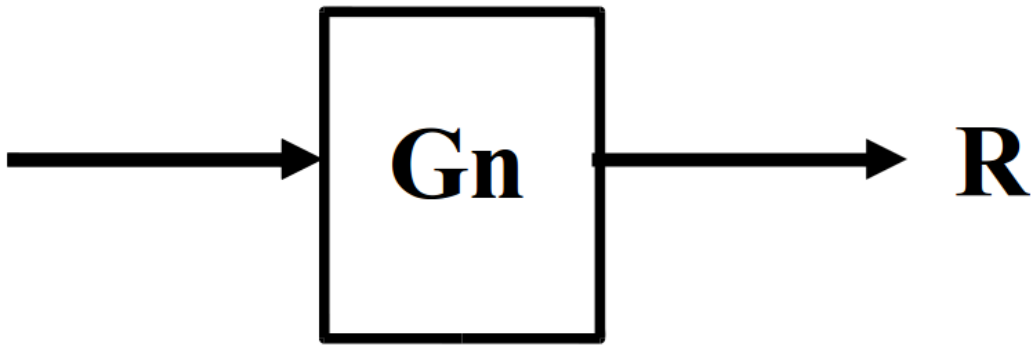
Work round the circuit to construct the input/output truth table, and verify that it is the same as the XOR gate.



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 FOUR possible one-input gates and SIXTEEN possible two-input gates.

Are there any more useful one-input gates?



We enumerate all one-input gate possibilities

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

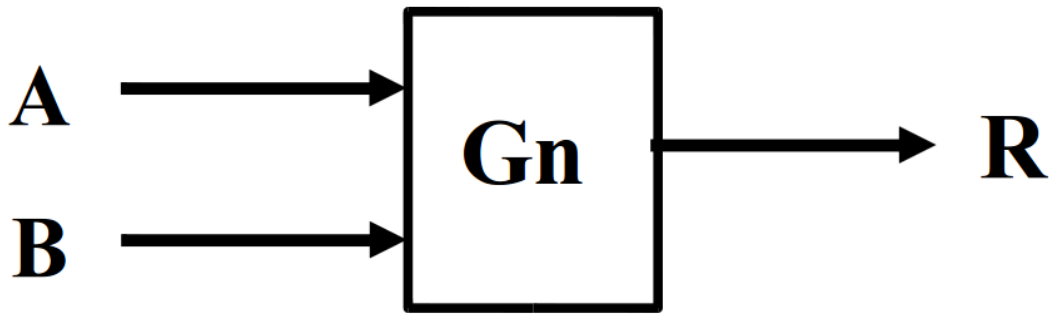
Clearly G2 in the above table is the inverter, and the other three possible gates are useless.

We can also define the gate function using algebra with the following table:

	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 using just NAND and NOR

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

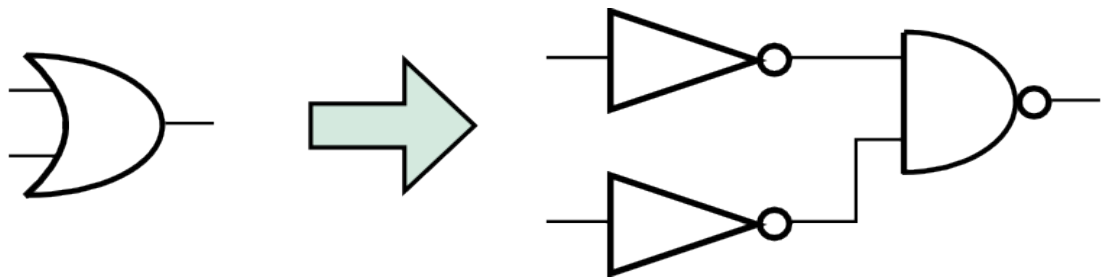
When designing a circuit to calculate a Boolean function, it makes sense to try and engineer it by using these gates only.

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

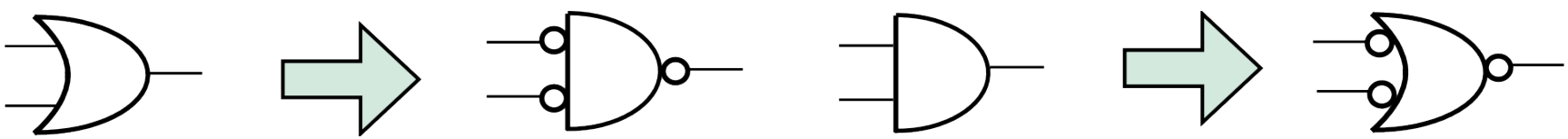
Applying de Morgan's Theorem Grapically

de Morgan's theorem can be applied graphically.

$A+B = (A'.B')'$ has the circuit equivalent:

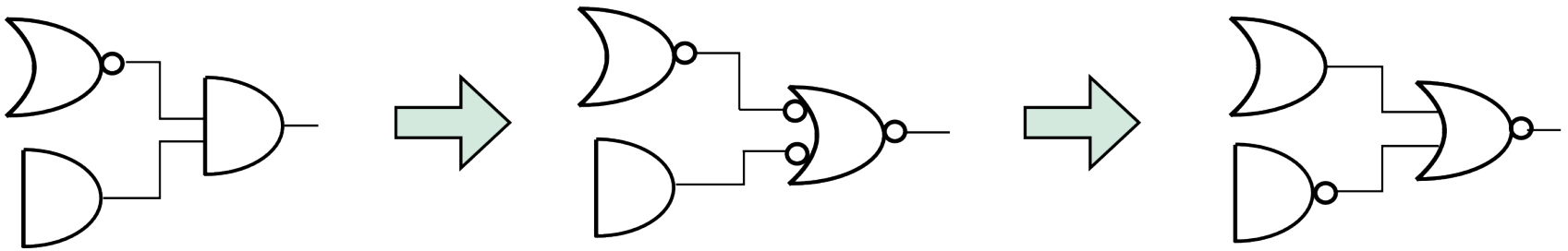


For simplicity we can use just circles for inversion:



Manipulating Circuit Diagrams

Often circuits can be simplified through symbolic manipulation



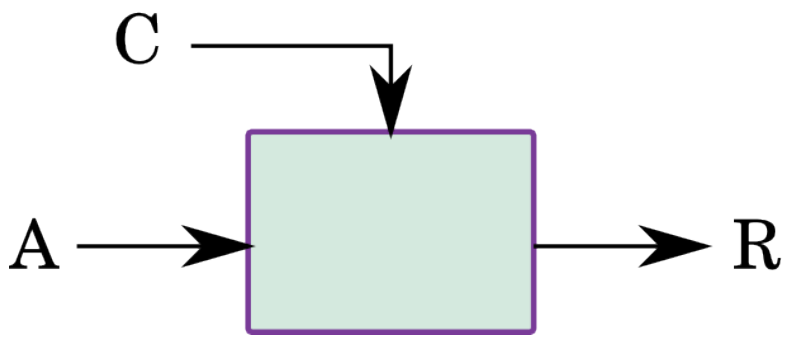
Apply de Morgan's theorem

Re-group and cancel the inverters

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

Control and Data Variables

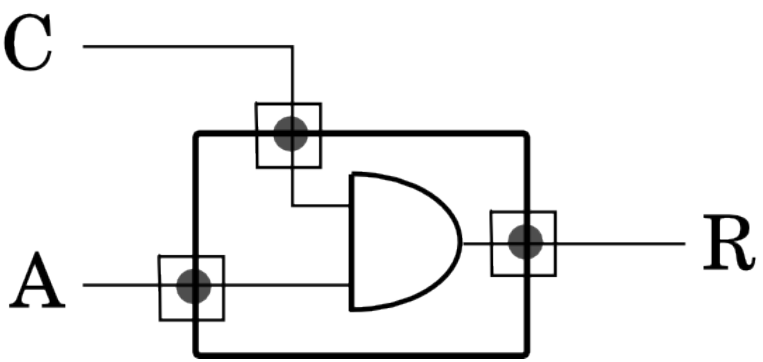
Suppose we want to design a controlled function circuit. For example when C=0 the output is 0, and when C=1 the output is A:



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

If we construct a truth table from the specification, we find that the above circuit can be implemented with just one AND gate:



This example shows why the name "gate" is used.

An Important Control Circuit

Consider a circuit with two data inputs A and B and once 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.

Implementing a Multiplexer

We use the Boolean AND function to "gate" the signals from A and B individually, and combine the result with an OR gate. We are using the rule that A+0=A and 0+B=B.

The complete circuit is: