COMP 122/L Lecture 17

Kyle Dewey

Outline

- Boolean formulas and truth tables
- Introduction to circuits

Boolean Formulas and Truth Tables

Boolean?

- Binary: `true` and `false`
  - Abbreviation: `1` and `0`
  - Easy for a circuit: on or off
- Serves as the building block for all digital circuits

Basic Operation: AND

`AB == A AND B`

Basic Operation: AND

`AB == A AND B`

`true` only if both `A` and `B` are `true`

Basic Operation: AND

`AB == A AND B`

`true` **only if both** `A` **and** `B` **are** `true`

Truth Table:

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

Basic Operation: OR

```
A + B == A OR B
```

Basic Operation: OR

`A + B == A OR B`

`false` only if both `A` and `B` are `false`

Basic Operation: OR

`A + B == A OR B`

`false` **only if both** `A` **and** `B` **are** `false`

Truth Table:

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

Basic Operation: NOT

`!A  ==  A′  ==` $\overline{A}$ `==  NOT A`

Basic Operation: NOT

$!A == A' == \overline{A} ==$ NOT A

Flip the result of the operand

Basic Operation: NOT

$!A == A' == \overline{A} ==$ NOT A

Flip the result of the operand

Truth Table:

A	!A
0	1
1	0

AND, OR, and NOT

- Serve as the basis for everything we will do in this class
- As simple as they are, they can do just about everything we want

Truth Table to Formula

- Idea: for every output in the truth table which has a `1`, write an AND which corresponds to it
- String them together with OR

Truth Table to Formula

- Idea: for every output in the truth table which has a `1`, write an AND which corresponds to it
- String them together with OR

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	1

–For example, consider this table

Truth Table to Formula

- Idea: for every output in the truth table which has a `1`, write an AND which corresponds to it
- String them together with OR

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	1

–First 1 in the table

Truth Table to Formula

- Idea: for every output in the truth table which has a `1`, write an AND which corresponds to it
- String them together with OR

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	1

`!A!B`

-This corresponds to !A!B
-That is, the output is set to 1 when !A!B is true (meaning when A = 0 and B = 0)

Truth Table to Formula

- Idea: for every output in the truth table which has a `1`, write an AND which corresponds to it
- String them together with OR

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	1

`!A!B`

–Second 1 in the table

Truth Table to Formula

- Idea: for every output in the truth table which has a `1`, write an AND which corresponds to it
- String them together with OR

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	1

`!A!B`

AB

-This corresponds to AB

Truth Table to Formula

- Idea: for every output in the truth table which has a `1`, write an AND which corresponds to it
- String them together with OR

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	1

`!A!B + AB`

-Finally, string them together with OR

Truth Table to Formula

- Idea: for every output in the truth table which has a `1`, write an AND which corresponds to it
- String them together with OR

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	1

`Out = !A!B + AB`

-Out is equal to this formula

Sum of Products Notation

This formula is in *sum of products* notation:

```
Out = !A!B + AB
```

Sum of Products Notation

This formula is in *sum of products* notation:

`Out = !A!B + AB`

Sum

Sum of Products Notation

This formula is in *sum of products* notation:

```
Out = !A!B + AB
```

Sum

Products

Sum of Products Notation

This formula is in *sum of products* notation:

```
Out = !A!B + AB
```

Sum

Products

Very closely related to the sort of sums and products you're more familiar with...more on that later.

Bigger Operations

Adding single bits with a carry-in and a carry-out (Cout)

Bigger Operations

Adding single bits with a carry-in and a carry-out (Cout)

0 0 +0 -- 0 Cout: 0	0 0 +1 -- 1 Cout: 0	0 1 +0 -- 1 Cout: 0	0 1 +1 -- 0 Cout: 1
1 0 +0 -- 1 Cout: 0	1 0 +1 -- 0 Cout: 1	1 1 +0 -- 0 Cout: 1	1 1 +1 -- 1 Cout: 1

Single Bit Addition as a Truth Table

Inputs?

Single Bit Addition as a Truth Table

Inputs?

Carry-in, first operand bit, second operand bit.

Single Bit Addition as a Truth Table

Inputs?

Carry-in, first operand bit, second operand bit.

Outputs?

Single Bit Addition as a Truth Table

Inputs?

Carry-in, first operand bit, second operand bit.

Outputs?

Result bit, carry-out bit.

Single Bit Addition as a Truth Table

A	B	Cin	R	Cout
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Single Bit Addition as a Truth Table

```
  0
  0
+0
--
```

A	B	Cin	R	Cout
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Single Bit Addition as a Truth Table

```
  0
  0
+0
--
  0
```
Cout: 0

A	B	Cin	R	Cout
0	0	0	0	0
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Single Bit Addition as a Truth Table

A	B	Cin	R	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Single Bit Addition as a Truth Table

A	B	Cin	R	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Single Bit Addition as a Truth Table

A	B	Cin	R	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Single Bit Addition as a Truth Table

A	B	Cin	R	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1		
1	1	0		
1	1	1		

Single Bit Addition as a Truth Table

A	B	Cin	R	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0		
1	1	1		

Single Bit Addition as a Truth Table

A	B	Cin	R	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1		

Single Bit Addition as a Truth Table

A	B	Cin	R	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Single Bit Addition as a Formula

Single Bit Addition as a Formula

A	B	Cin	R	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

–If we take the truth table from before...

Single Bit Addition as a Formula

A	B	Cin	R	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

-Need a formula for each output
-Start with R (arbitrary; could also start at Cout)

Single Bit Addition as a Formula

A	B	Cin	R	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

–We have these products

Single Bit Addition as a Formula

A	B	Cin	R	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

R = !A!BCin +
!AB!Cin +
A!B!Cin +
ABCin

–We have these products

Single Bit Addition as a Formula

A	B	Cin	R	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

```
R = !A!BCin +
   !AB!Cin +
   A!B!Cin +
     ABCin
```

Single Bit Addition as a Formula

A	B	Cin	R	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

R = !A!BCin +
!AB!Cin +
A!B!Cin +
ABCin

Single Bit Addition as a Formula

A	B	Cin	R	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

```
R = !A!BCin +
!AB!Cin +
A!B!Cin +
ABCin
```

```
Cout = !ABCin +
A!BCin +
AB!Cin +
ABCin
```

Circuits

Circuits

- AND, OR, and NOT can be implemented with physical hardware
  - Therefore, anything representable with AND, OR, and NOT can be turned into a hardware device

AND Gate

Circuit takes two inputs and produces one output

AND Gate

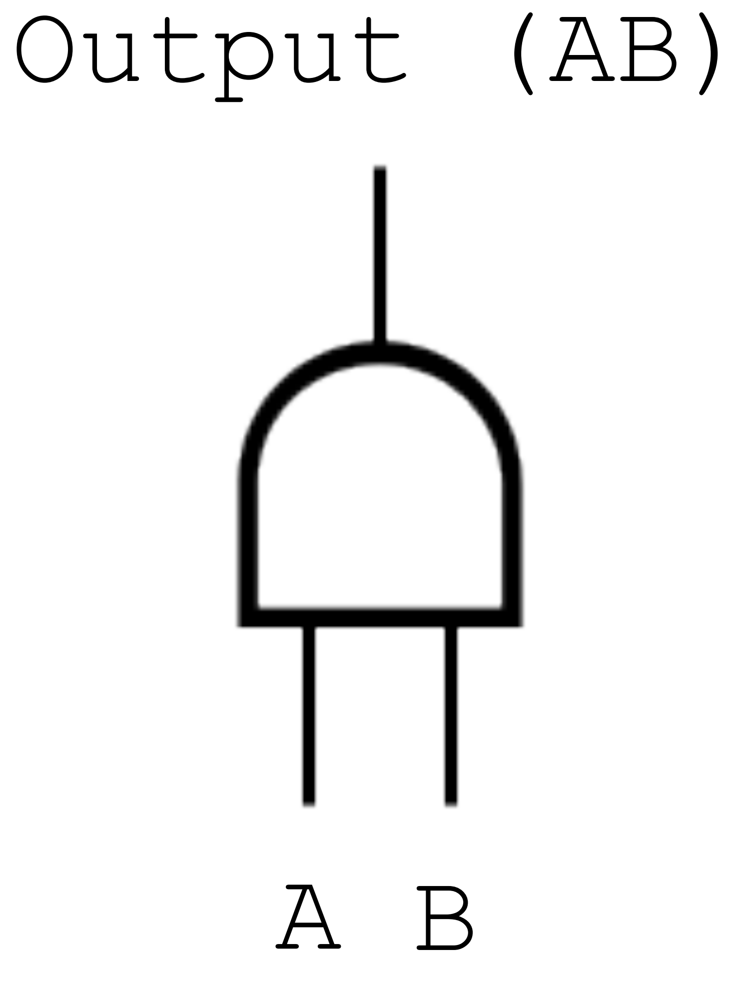
Circuit takes two inputs and produces one output

AB

AND Gate

Circuit takes two inputs and produces one output

AB



OR Gate

Circuit takes two inputs and produces one output

OR Gate

Circuit takes two inputs and produces one output

`A + B`

OR Gate

Circuit takes two inputs and produces one output

A + B

Output (A + B)

A B

NOT (Inverter)

Circuit takes one input and produces one output

NOT (Inverter)

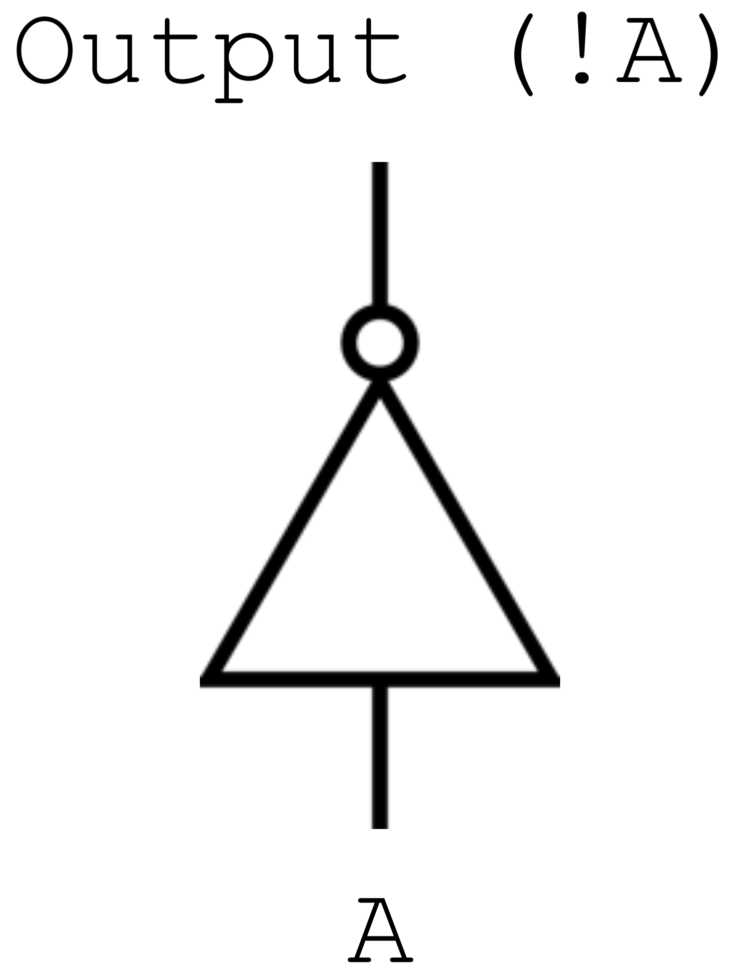
Circuit takes one input and produces one output

`!A`

NOT (Inverter)

Circuit takes one input and produces one output

`!A`



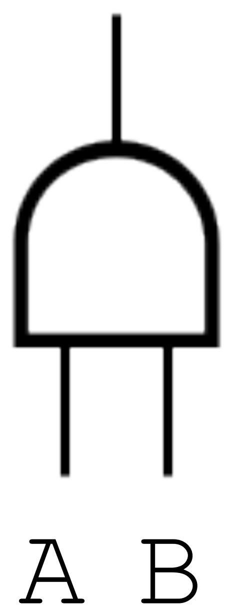
Formula to Circuit

Formula to Circuit

`(AB)C`

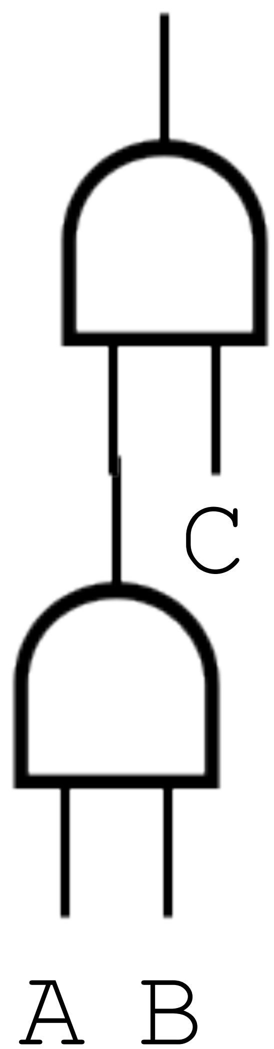
Formula to Circuit

(AB)C



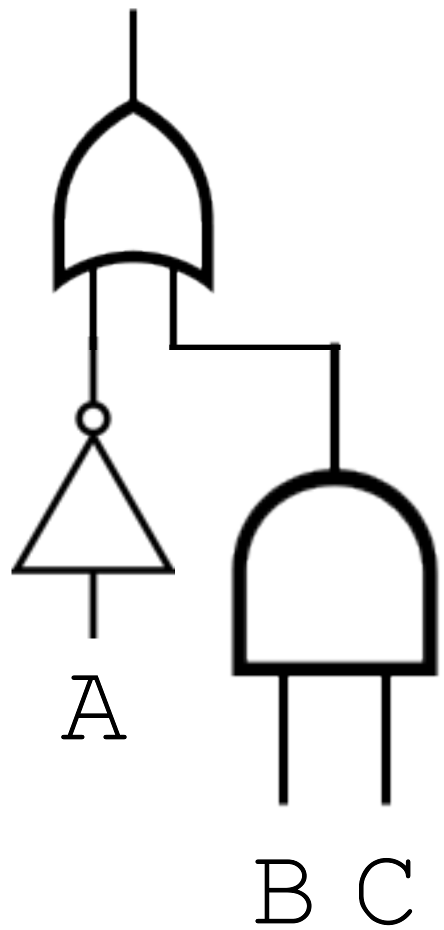
Formula to Circuit

(AB) C

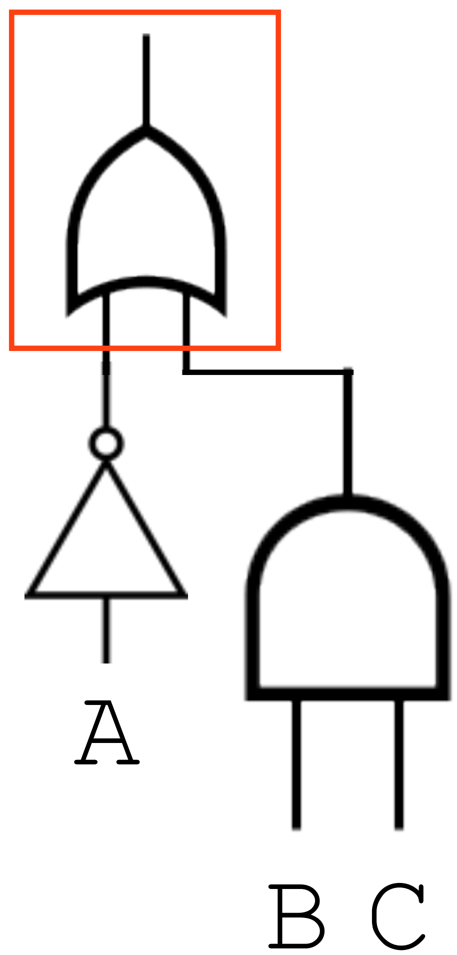


Circuit to Formula

Circuit to Formula

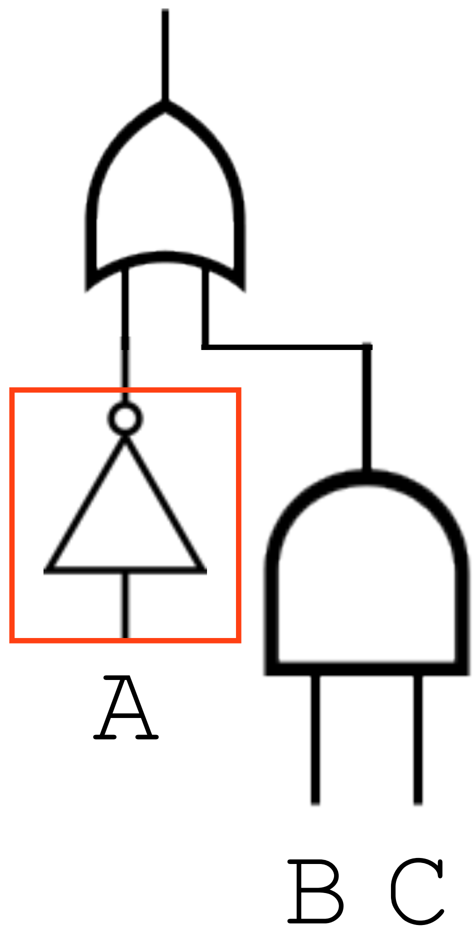


Circuit to Formula



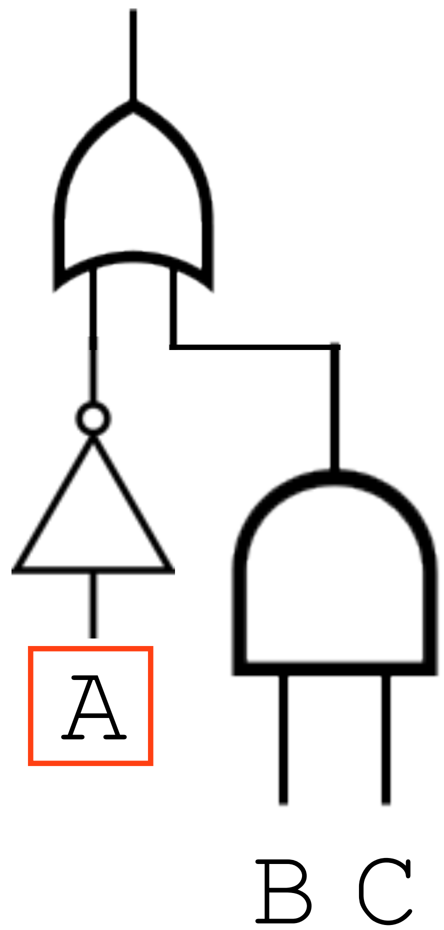
??? + ???

Circuit to Formula



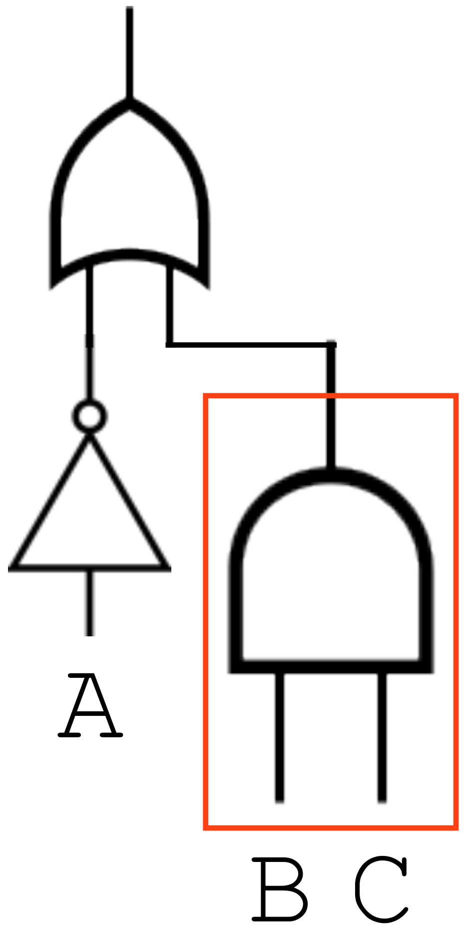

```
!??? + ???
```

Circuit to Formula



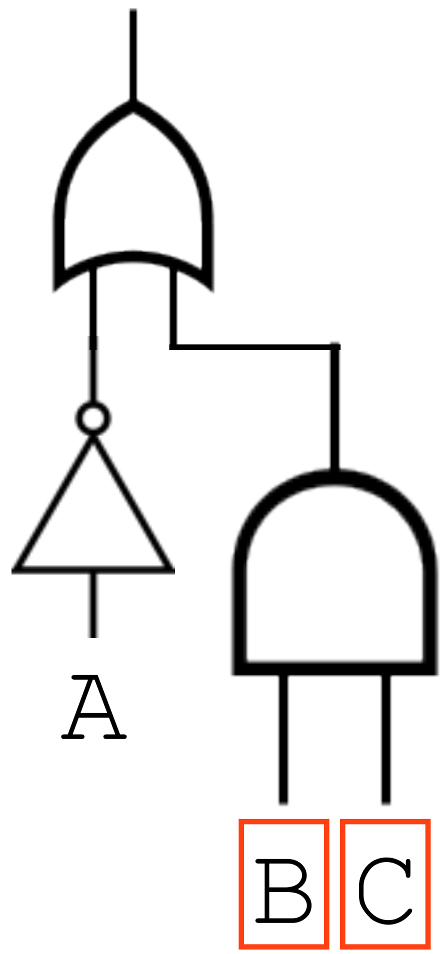
!A + ???

Circuit to Formula



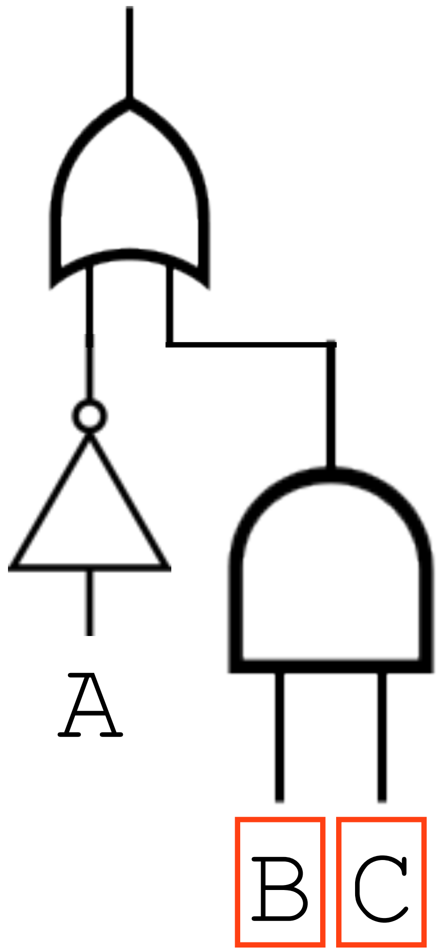
`!A + (???)(???)`

Circuit to Formula



!A + (B) (C)

Circuit to Formula



!A + BC