COMP 122/L Week 6

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′ == ` $\overline{\texttt{A}}$ ` == 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

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

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`

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`

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`

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`

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`

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

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

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

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
```

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

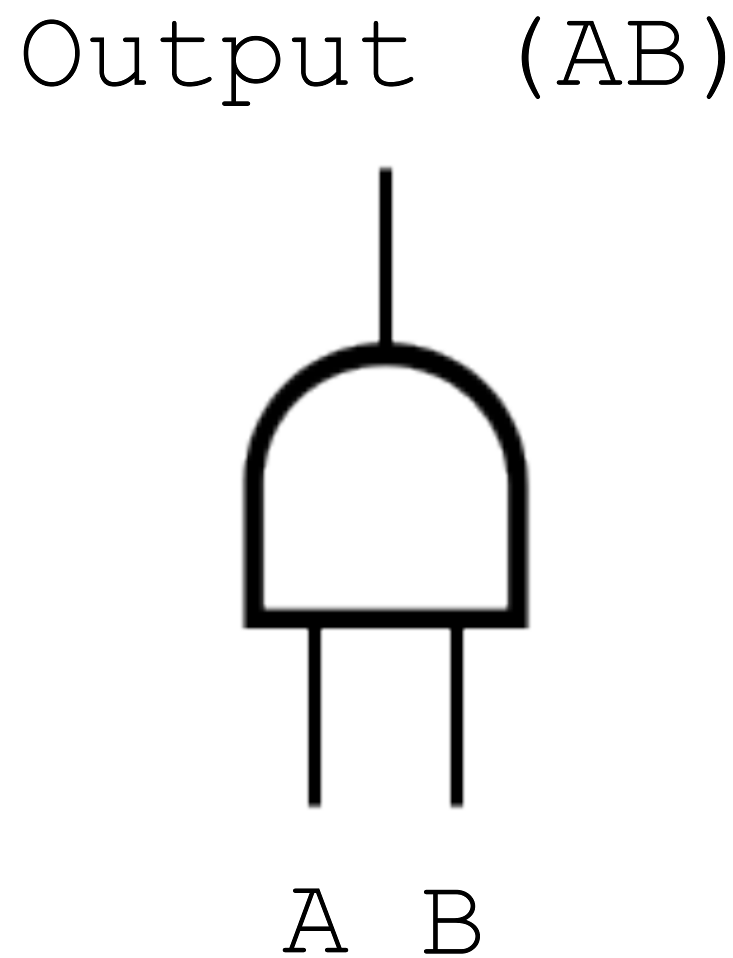
Circuit takes two inputs and produces one output

`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

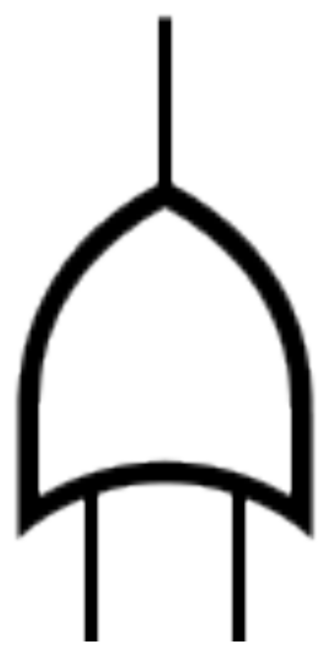
`A + B`

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)

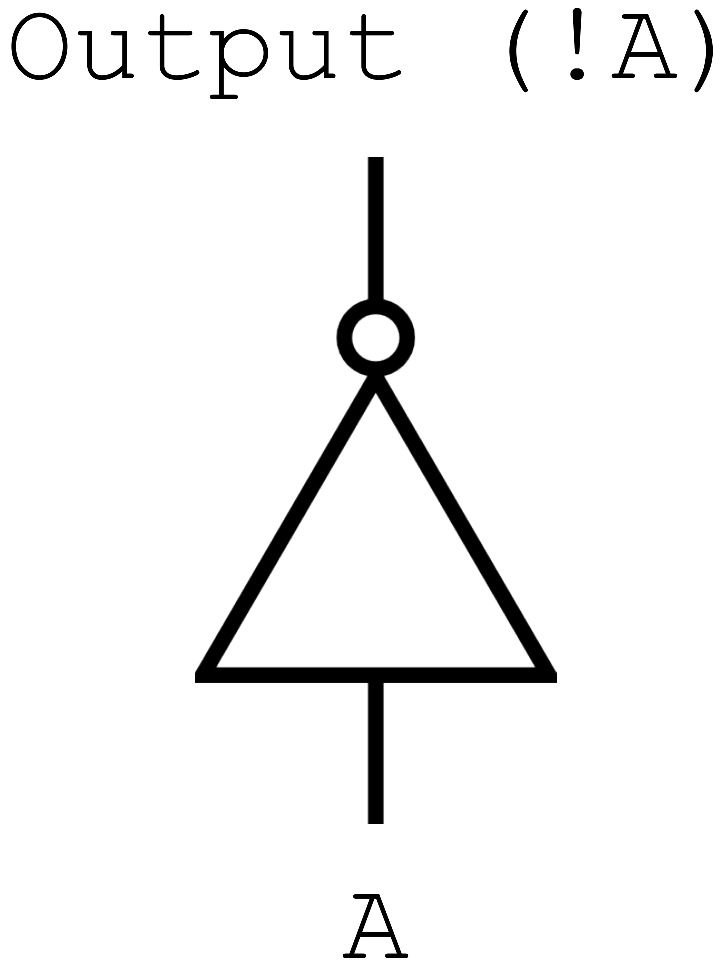
Circuit takes one input and produces one output

`!A`

NOT (Inverter)

Circuit takes one input and produces one output

!A



Formula to Circuit

Formula to Circuit

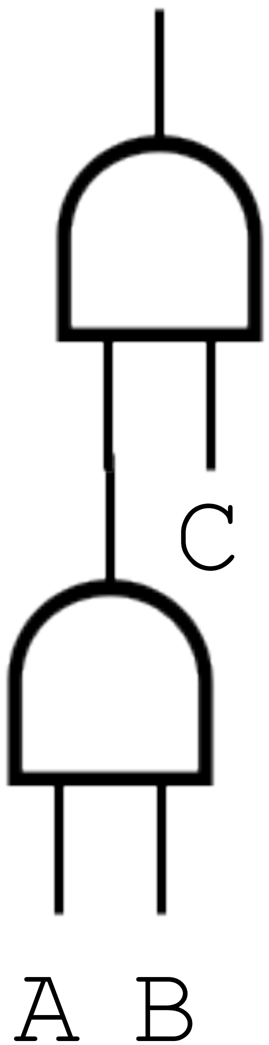
`(AB)C`

Formula to Circuit

(AB) C

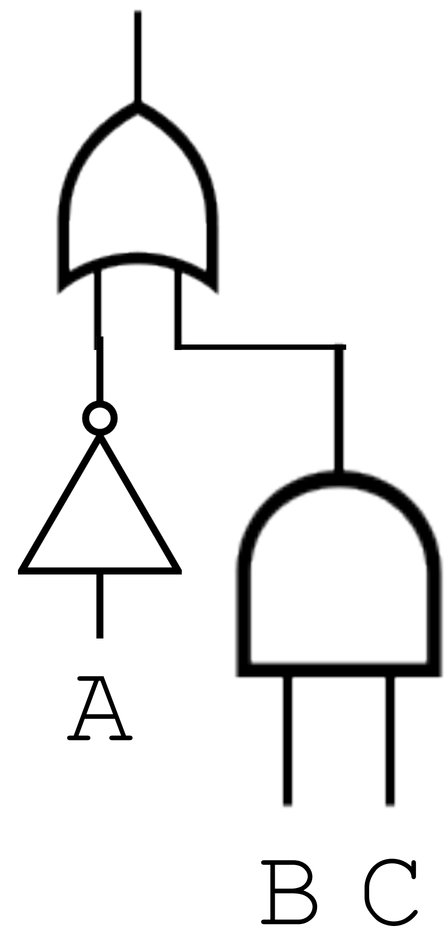
Formula to Circuit

(AB) C

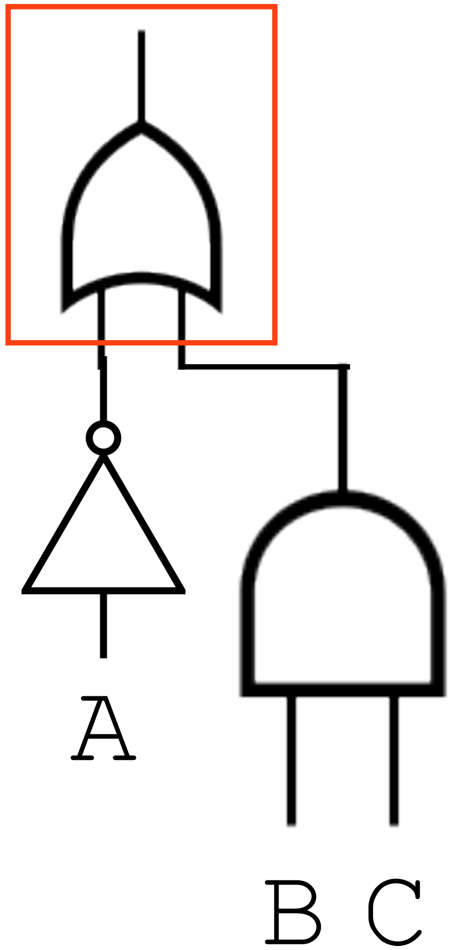


Circuit to Formula

Circuit to Formula

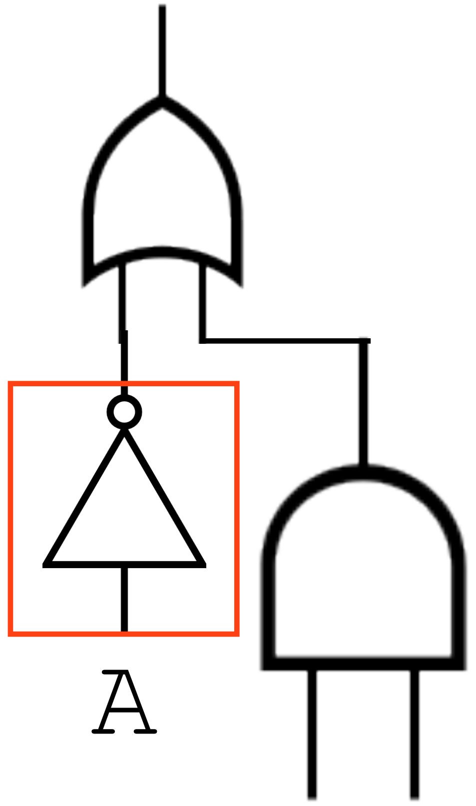


Circuit to Formula



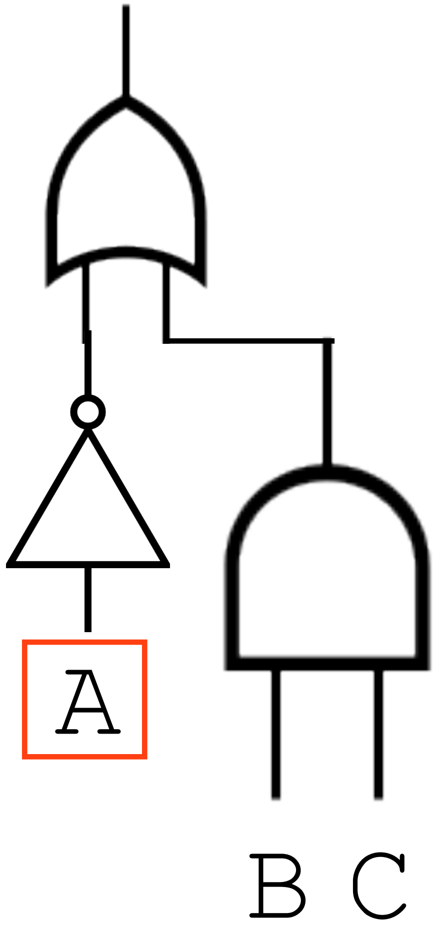
??? + ???

Circuit to Formula



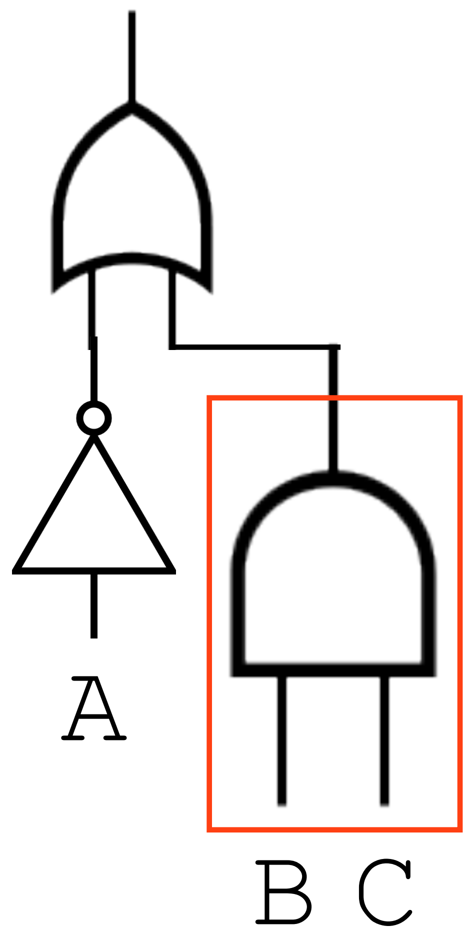
!??? + ???

Circuit to Formula



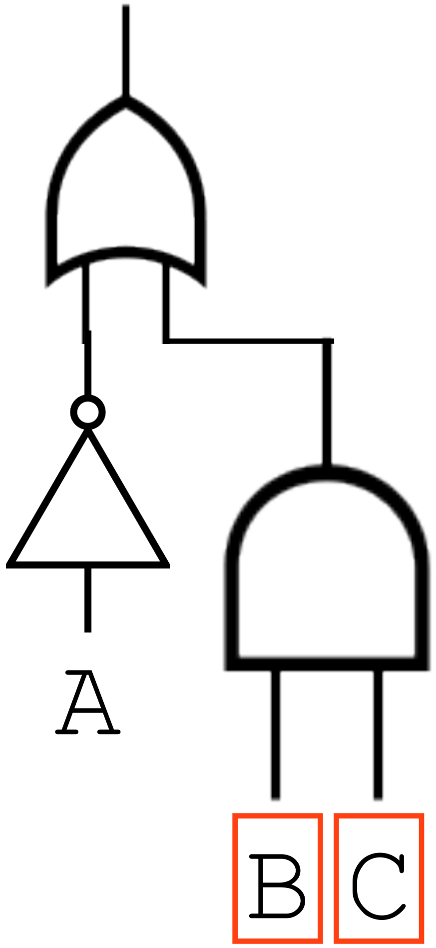
!A + ???

Circuit to Formula



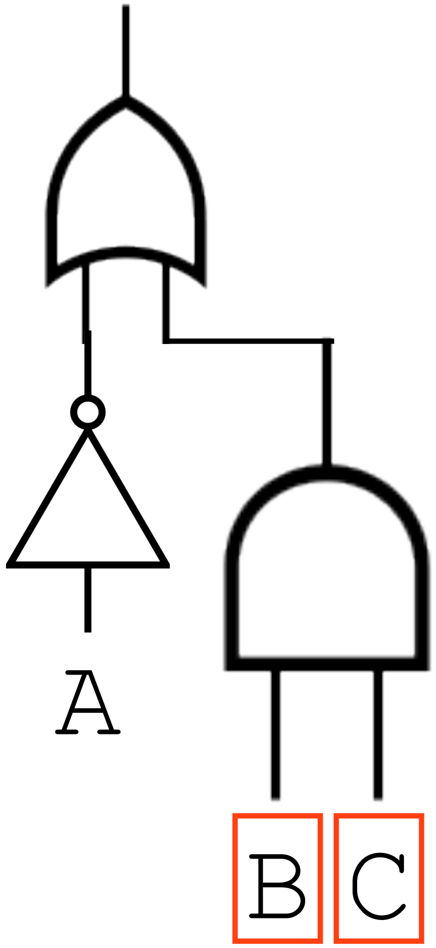
!A + (???)(???)

Circuit to Formula



!A + (B)(C)

Circuit to Formula



!A + BC

Overview

- Circuit minimization
  - Boolean algebra
  - Karnaugh maps

Circuit Minimization

Motivation

- Unnecessarily large programs: bad
- Unnecessarily large circuits: Very Bad™
  - Why?

Motivation

- Unnecessarily large programs: bad
- Unnecessarily large circuits: Very Bad™
  - Why?
    - Bigger circuits = bigger chips = higher cost (non-linear too!)
    - Longer circuits = more time needed to move electrons through = slower

Simplification

- Real-world formulas can often be simplified, according to algebraic rules
  - How might we simplify the following?

```
R = A*B + !A*B
```

Simplification

- Real-world formulas can often be simplified, according to algebraic rules
  - How might we simplify the following?

```
R = A*B + !A*B
R = B(A + !A)
R = B(true)
R = B
```

Simplification Trick

- Look for products that differ only in one variable
  - One product has the original variable (`A`)
  - The other product has the other variable (`!A`)

```
R = A*B + !A*B
```

Additional Example 1

```
!ABCD + ABCD + !AB!CD + AB!CD
```

Additional Example 1

`!ABCD + ABCD + !AB!CD + AB!CD`

`BCD(A + !A) + !AB!CD + AB!CD`

Additional Example 1

```
!ABCD + ABCD + !AB!CD + AB!CD

BCD(A + !A) + !AB!CD + AB!CD

BCD + !AB!CD + AB!CD
```

Additional Example 1

```
!ABCD + ABCD + !AB!CD + AB!CD

BCD(A + !A) + !AB!CD + AB!CD

BCD + !AB!CD + AB!CD

BCD + B!CD(!A + A)
```

Additional Example 1

```
!ABCD + ABCD + !AB!CD + AB!CD

BCD(A + !A) + !AB!CD + AB!CD

BCD + !AB!CD + AB!CD

BCD + B!CD(!A + A)

BCD + B!CD
```

Additional Example 1

```
!ABCD + ABCD + !AB!CD + AB!CD

BCD(A + !A) + !AB!CD + AB!CD

BCD + !AB!CD + AB!CD

BCD + B!CD(!A + A)

BCD + B!CD

BD(C + !C)
```

Additional Example 1

```
!ABCD + ABCD + !AB!CD + AB!CD

BCD(A + !A) + !AB!CD + AB!CD

BCD + !AB!CD + AB!CD

BCD + B!CD(!A + A)

BCD + B!CD

BD(C + !C)
BD
```

Additional Example 2

`!A!BC + A!B!C + !ABC + !AB!C + A!BC`

Additional Example 2

`!A!BC + A!B!C + !ABC + !AB!C + A!BC`

`!A!BC + A!BC + A!B!C + !ABC + !AB!C`

Additional Example 2

```
!A!BC + A!B!C + !ABC + !AB!C + A!BC

!A!BC + A!BC + A!B!C + !ABC + !AB!C

!BC(A + !A) + A!B!C + !ABC + !AB!C
```

Additional Example 2

```
!A!BC + A!B!C + !ABC + !AB!C + A!BC

!A!BC + A!BC + A!B!C + !ABC + !AB!C

!BC(A + !A) + A!B!C + !ABC + !AB!C

!BC + A!B!C + !ABC + !AB!C
```

Additional Example 2

```
!A!BC + A!B!C + !ABC + !AB!C + A!BC

!A!BC + A!BC + A!B!C + !ABC + !AB!C

!BC(A + !A) + A!B!C + !ABC + !AB!C

!BC + A!B!C + !ABC + !AB!C

!BC + A!B!C + !AB(C + !C)
```

Additional Example 2

```
!A!BC + A!B!C + !ABC + !AB!C + A!BC

!A!BC + A!BC + A!B!C + !ABC + !AB!C

!BC(A + !A) + A!B!C + !ABC + !AB!C

!BC + A!B!C + !ABC + !AB!C

!BC + A!B!C + !AB(C + !C)

!BC + A!B!C + !AB
```

De Morgan’s Laws

Also potentially useful for simplification

De Morgan's Laws

Also potentially useful for simplification

```
!(A + B) = !A!B
```

De Morgan's Laws

Also potentially useful for simplification

```
!(A + B) = !A!B

!(AB) = !A + !B
```

De Morgan Example

```
!(X + Y)!(!X + Z)
```

De Morgan Example

```
!(X + Y)!(!X + Z)

    !A          !B
```

De Morgan Example

`!(X + Y)!(!X + Z)`

`!A` `!B`

De Morgan Example

`!(X + Y)!(!X + Z)`

`!A` `!B`

From De Morgan's Law:

`!(A + B) = !A!B`

De Morgan Example

```
!(X + Y)!(!X + Z)
```

```
    !A          !B
```

From De Morgan's Law:

```
!(A + B) = !A!B
```

```
!(X + Y + !X + Z)
```

De Morgan Example

```
!(X + Y)!(!X + Z)

    !A          !B
```

From De Morgan's Law:

```
!(A + B) = !A!B

!(X + Y + !X + Z)

!(X + !X + Y + Z)
```

De Morgan Example

```
!(X + Y)!(!X + Z)

   !A          !B
```

From De Morgan's Law:

```
!(A + B) = !A!B

!(X + Y + !X + Z)

!(X + !X + Y + Z)

!(true + Y + Z)
```

De Morgan Example

`!(X + Y)!(!X + Z)`

`!A                !B`

From De Morgan's Law:

`!(A + B) = !A!B`

`!(X + Y + !X + Z)`

`!(X + !X + Y + Z)`

`!(true + Y + Z)`

`!(true)`

De Morgan Example

```
!(X + Y)!(!X + Z)

   !A          !B
```

From De Morgan's Law:

```
!(A + B) = !A!B

!(X + Y + !X + Z)

!(X + !X + Y + Z)

!(true + Y + Z)

!(true)

false
```

Scaling Up

- Performing this sort of algebraic manipulation by hand can be tricky
- We can use *Karnaugh maps* to make it immediately apparent as to what can be simplified

Example

```
R = A*B + !A*B
```

Example

R = A*B + !A*B

A	B	O
0	0	0
0	1	1
1	0	0
1	1	1

Example

R = A*B + !A*B

A	B	O
0	0	0
0	1	1
1	0	0
1	1	1

A \ B	0	1
0	0	1
1	0	1

Example

R = A*B + !A*B

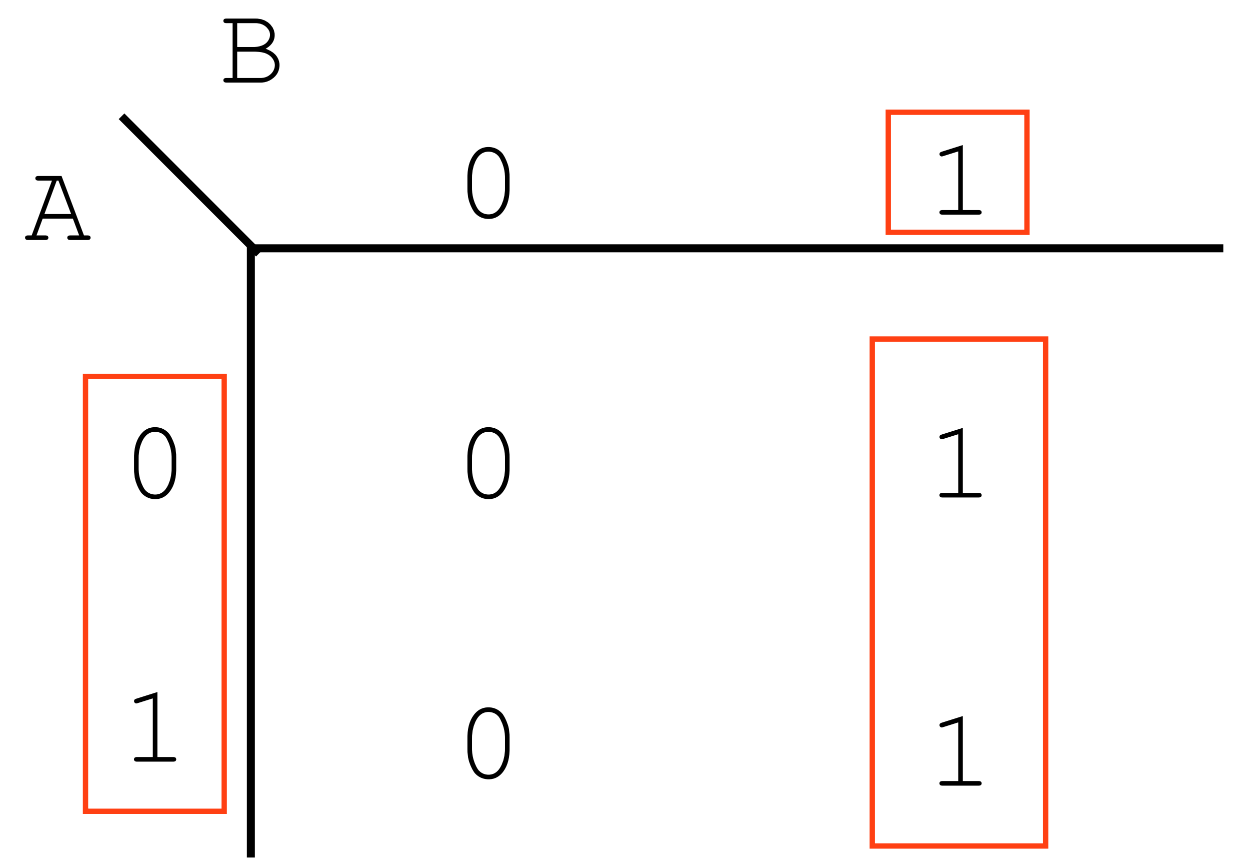
A	B	O
0	0	0
0	1	1
1	0	0
1	1	1

A \ B	0	1
0	0	1
1	0	1

Example

R = A*B + !A*B

A	B	O
0	0	0
0	1	1
1	0	0
1	1	1

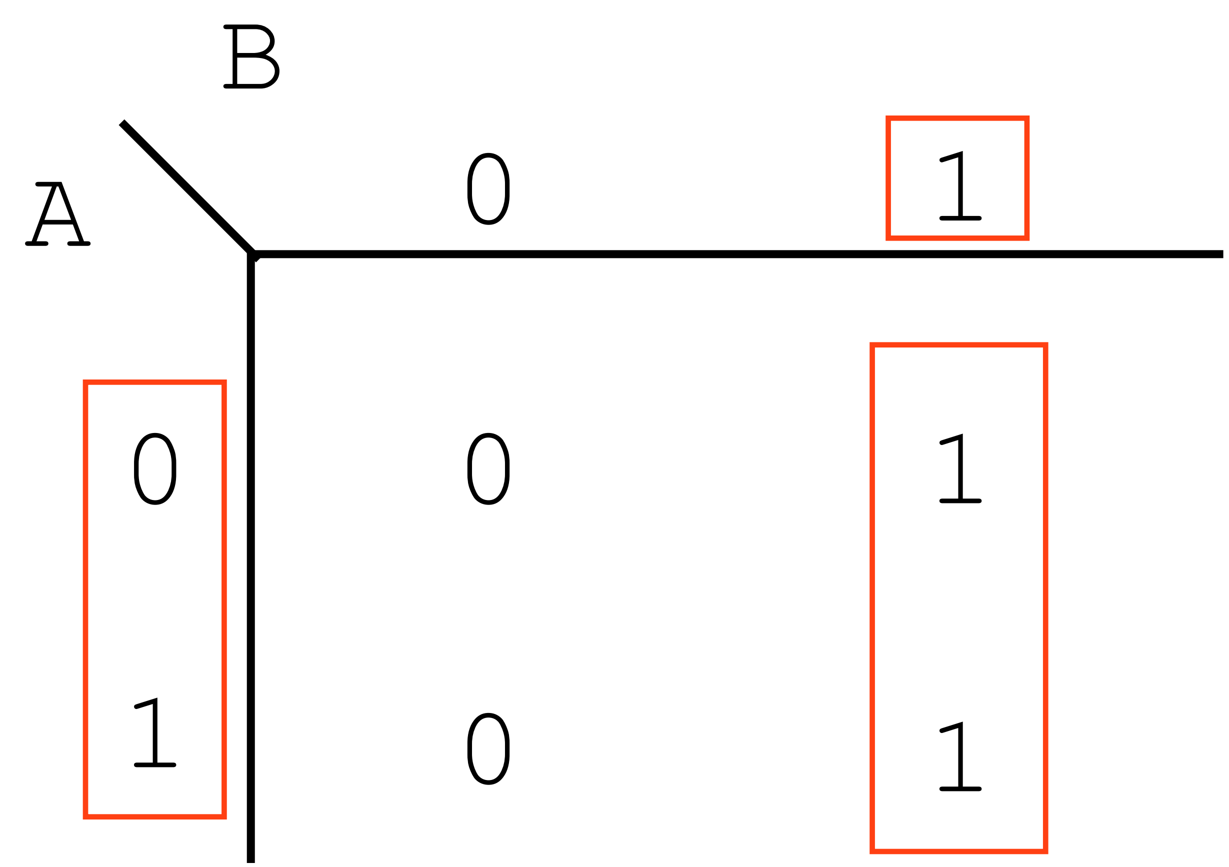


Example

R = A*B + !A*B

A	B	O
0	0	0
0	1	1
1	0	0
1	1	1

R = B



Three Variables

- We can scale this up to three variables, by combining two variables on one axis
- The combined axis must be arranged such that only one bit changes per position

A \ BC	00	01	11	10
0	?	?	?	?
1	?	?	?	?

Three Variable Example

R = !A!BC + !ABC + A!BC + ABC

R = !A!BC + !ABC + A!BC + ABC

A	B	C	R
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	0
1	1	1	1

R = !A!BC + !ABC + A!BC + ABC

A	B	C	R
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	0
1	1	1	1

A \ BC	00	01	11	10
0	0	1	1	0
1	0	1	1	0

R = !A!BC + !ABC + A!BC + ABC

A	B	C	R
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	0
1	1	1	1

A \ BC	00	01	11	10
0	0	1	1	0
1	0	1	1	0

$$R = !A!BC + !ABC + A!BC + ABC$$

A	B	C	R
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	0
1	1	1	1

R = C

A \ BC	00	01	11	10
0	0	1	1	0
1	0	1	1	0

Another Three Variable Example

R = !A!B!C + !A!BC + !ABC + !AB!C + A!B!C + AB!C

R = !A!B!C + !A!BC + !ABC +
!AB!C + A!B!C + AB!C

A	B	C	R
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

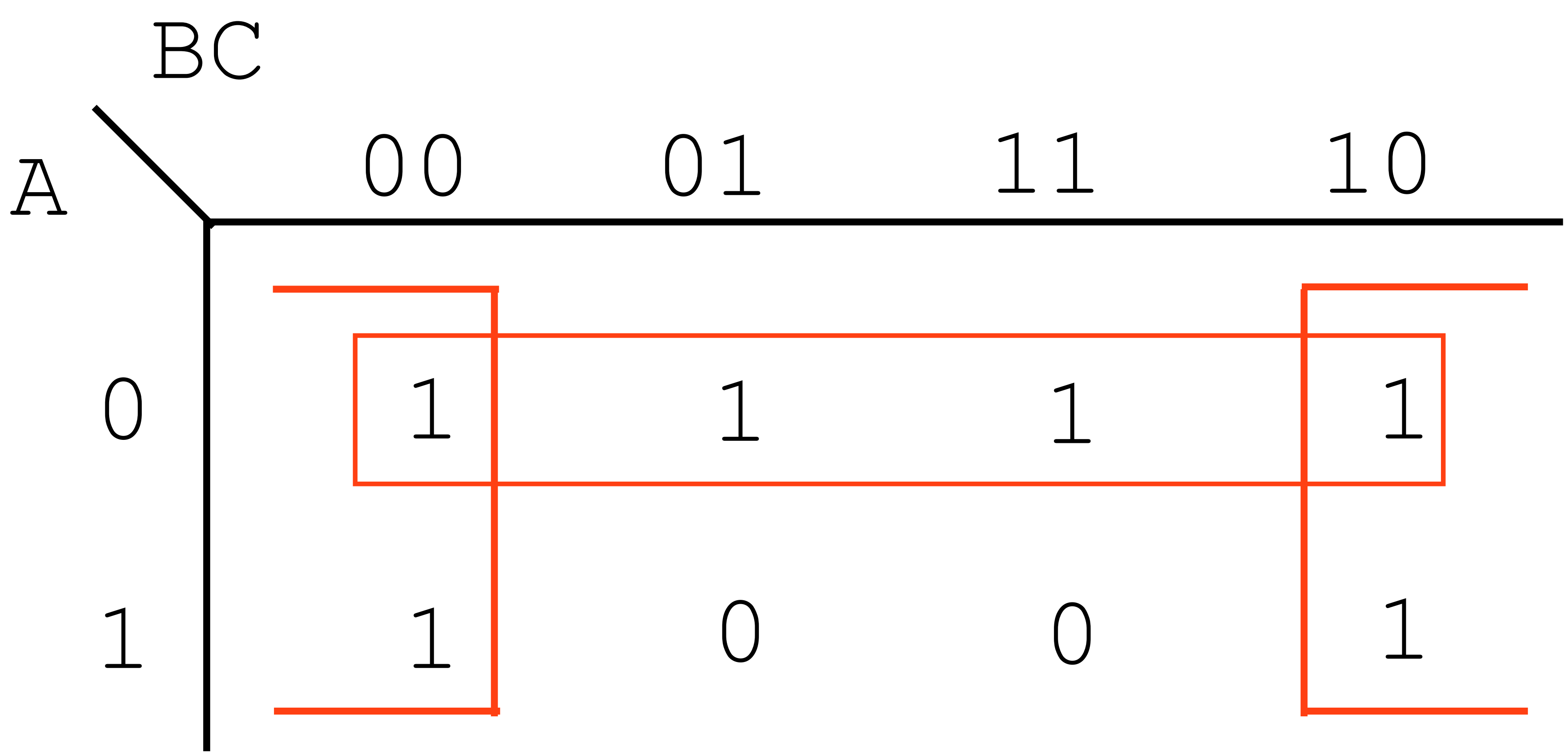
$$R = !A!B!C + !A!BC + !ABC + !AB!C + A!B!C + AB!C$$

A	B	C	R
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

A \ BC	00	01	11	10
0	1	1	1	1
1	1	0	0	1

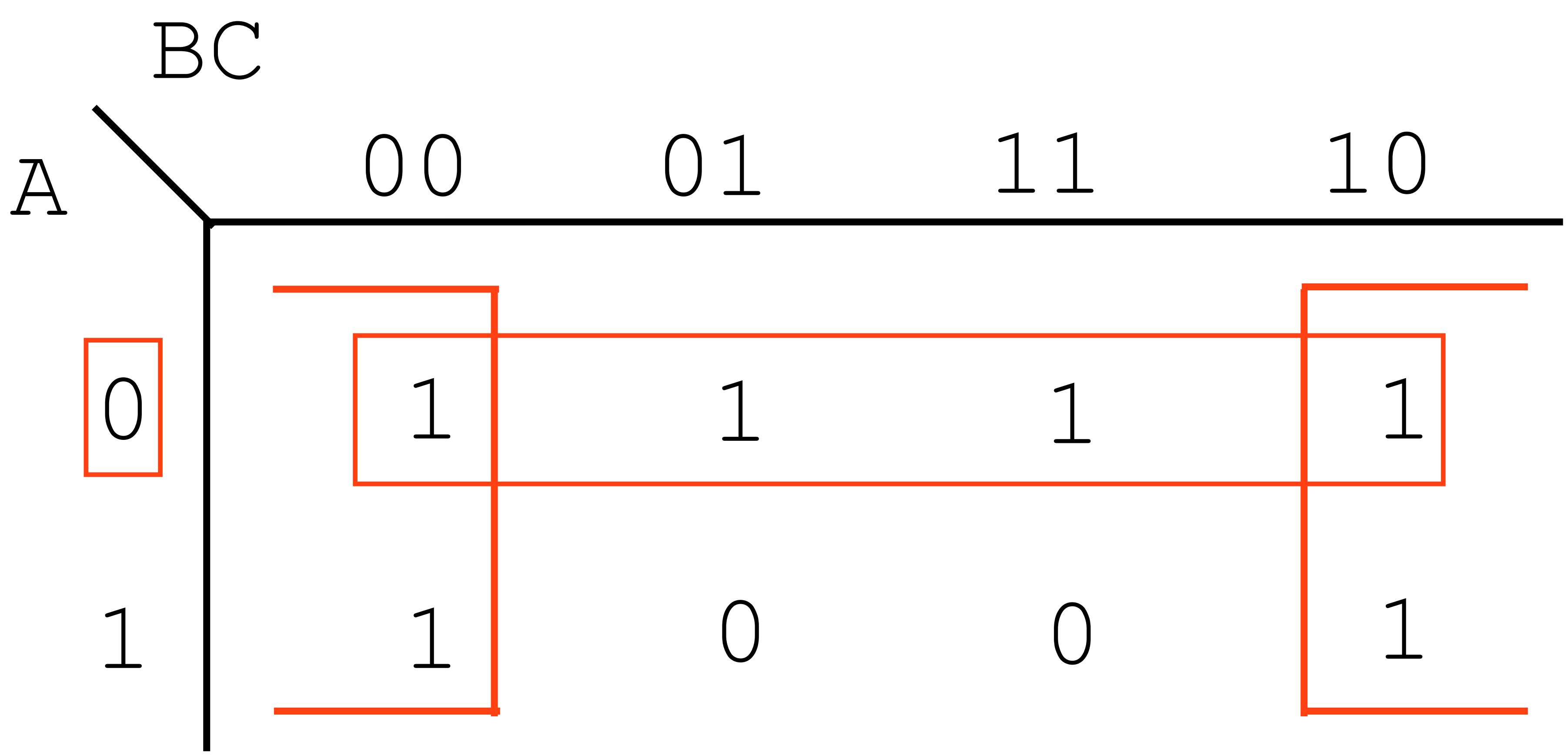
R = !A!B!C + !A!BC + !ABC +
!AB!C + A!B!C + AB!C

A	B	C	R
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



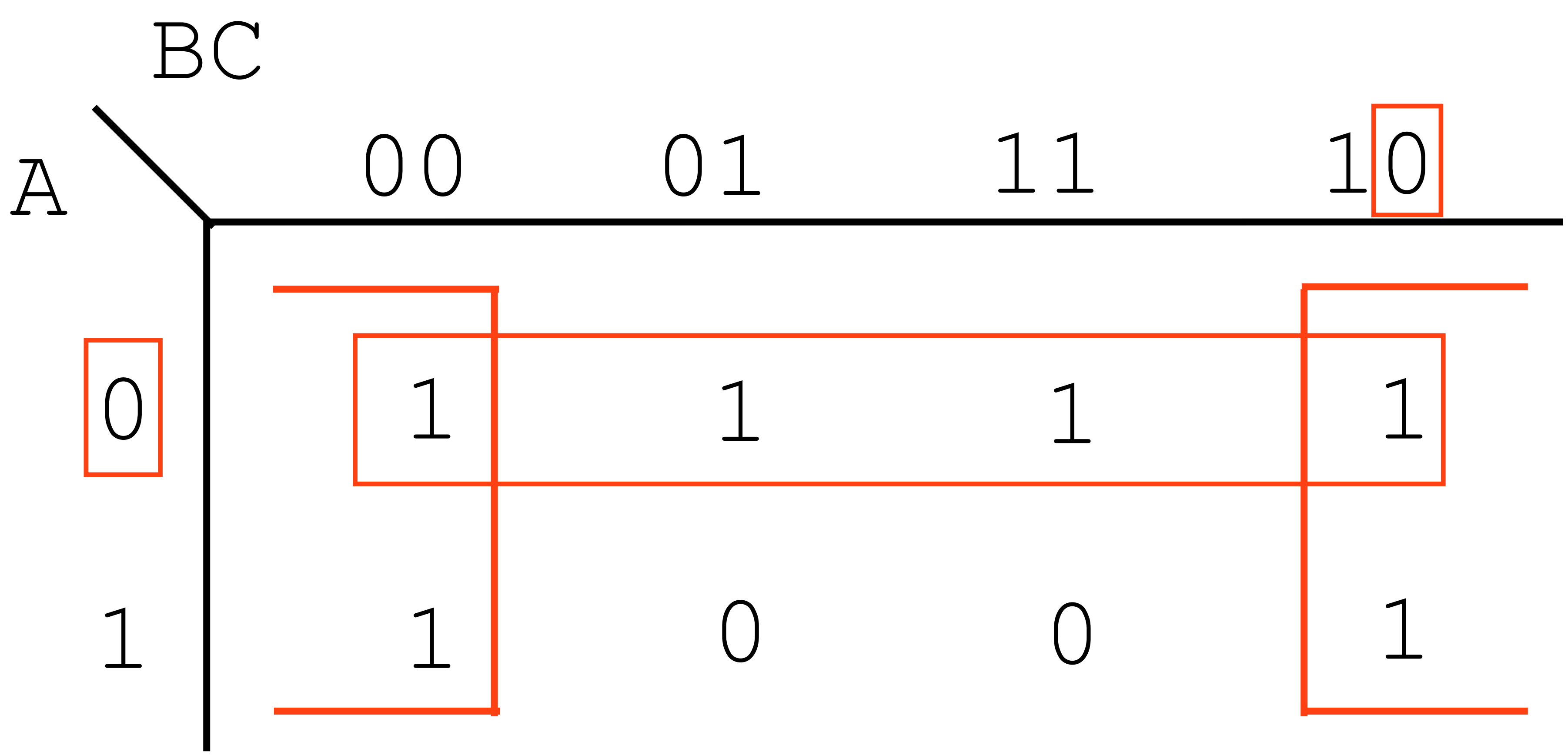
R = !A!B!C + !A!BC + !ABC + !AB!C + A!B!C + AB!C

A	B	C	R
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



R = !A!B!C + !A!BC + !ABC + !AB!C + A!B!C + AB!C

A	B	C	R
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



$$R = !A!B!C + !A!BC + !ABC + !AB!C + A!B!C + AB!C$$

A	B	C	R
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$$R = !A + !C$$

A \ BC	00	01	11	10
0	1	1	1	1
1	1	0	0	1

Four Variable Example

R = !A!B!C!D + !A!B!CD + !A!BC!D + !ABC!D + A!B!C!D + A!B!CD + A!BC!D

R = !A!B!C!D + !A!B!CD + !A!BC!D + !ABC!D + A!B!C!D + A!B!CD + A!BC!D

AB \ CD	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

R = !A!B!C!D + !A!B!CD + !A!BC!D + !ABC!D + A!B!C!D + A!B!CD + A!BC!D

AB \ CD	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

R = !A!B!C!D + !A!B!CD + !A!BC!D + !ABC!D + A!B!C!D + A!B!CD + A!BC!D

R =!B!C

AB \ CD	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

R = !A!B!C!D + !A!B!CD + !A!BC!D + !ABC!D + A!B!C!D + A!B!CD + A!BC!D

R =!B!C + !B!D

AB \ CD	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

R = !A!B!C!D + !A!B!CD + !A!BC!D + !ABC!D + A!B!C!D + A!B!CD + A!BC!D

R =!B!C + !B!D + !AC!D

AB \ CD	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

K-Map Rules in Summary (1)

- Groups can contain only `1`s
- Only `1`s in adjacent groups are allowed (no diagonals)
- The number of `1`s in a group must be a power of two (1, 2, 4, 8...)
- The groups must be as large as legally possible

K-Map Rules in Summary (2)

- All `1`s must belong to a group, even if it's a group of one element
- Overlapping groups are permitted
- Wrapping around the map is permitted
- Use the fewest number of groups possible

Revisiting Problem

```
!A!BC + A!B!C + !ABC + !AB!C + A!BC
```

Revisiting Problem

R = !A!BC + A!B!C + !ABC + !AB!C + A!BC

A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Revisiting Problem

R = !A!BC + A!B!C + !ABC + !AB!C + A!BC

A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

A \ BC	00	01	11	10
0	0	1	1	1
1	1	1	0	0

Revisiting Problem

R = !A!BC + A!B!C + !ABC + !AB!C + A!BC

A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

R = !AC

A \ BC	00	01	11	10
0	0	1	1	1
1	1	1	0	0

Revisiting Problem

R = !A!BC + A!B!C + !ABC + !AB!C + A!BC

A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

R = !AC + A!B

A \ BC	00	01	11	10
0	0	1	1	1
1	1	1	0	0

Revisiting Problem

R = !A!BC + A!B!C + !ABC + !AB!C + A!BC

A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

R = !AC + A!B + !AB!C

A \ BC	00	01	11	10
0	0	1	1	1
1	1	1	0	0

Difference

- Algebraic solution: `!BC + A!B!C + !AB`
- K-map solution: `!AC + A!B + !AB!C`
- Question: why might these differ?

Difference

- Algebraic solution: `!BC + A!B!C + !AB`
- K-map solution: `!AC + A!B + !AB!C`
- Question: why might these differ?
    - Both are *minimal*, in that they have the fewest number of products possible
    - Can be multiple minimal solutions

Difference

- Algebraic solution: `!BC + A!B!C + !AB`
- K-map solution: `!AC + A!B + !AB!C`
- Question: why might these differ?
  - Both are *minimal*, in that they have the fewest number of products possible
  - Can be multiple minimal solutions

Difference

Algebraic solution: `!BC + A!B!C + !AB`

K-map solution: `!AC + A!B + !AB!C`

A \ BC	00	01	11	10
0	0	1	1	1
1	1	1	0	0

Difference

Algebraic solution: `!BC + A!B!C + !AB`
K-map solution: `!BC + A!B!C + !AB`