Boolean Algebra

MIEET 1º ano

Peter Stallinga UAlg 2011

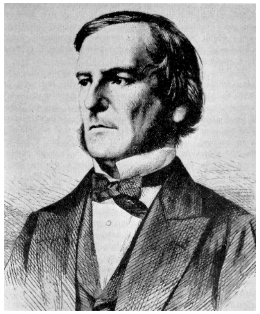
Age old question ….

“To be or not to be ….

… that is the question”

- William Shakespeare

Boolean algebra; 'logic'



Boolean Algebra of George Boole

ℕ: Natural numbers {1, 2, 3, …}, for countable and existing objects

ℤ: Integer numbers {…, -2, -2, 0, 1, 2, …}

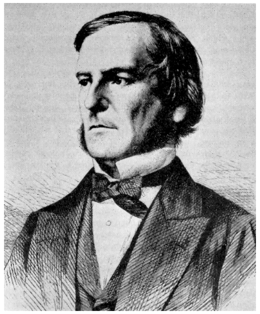
ℚ: Numbers resulting from fraction *a*/*b*, *a* and *b* € ℤ

ℝ: All real numbers

ℂ: All complex numbers *a* + *i b*, *a* and *b* € ℝ

𝔹: {0, 1} or any binary combination. Two possibilities!

Boolean algebra; 'logic'



Boolean Algebra of George Boole

$\mathbb{B}$: {0, 1} or any binary combination. Two possibilities!

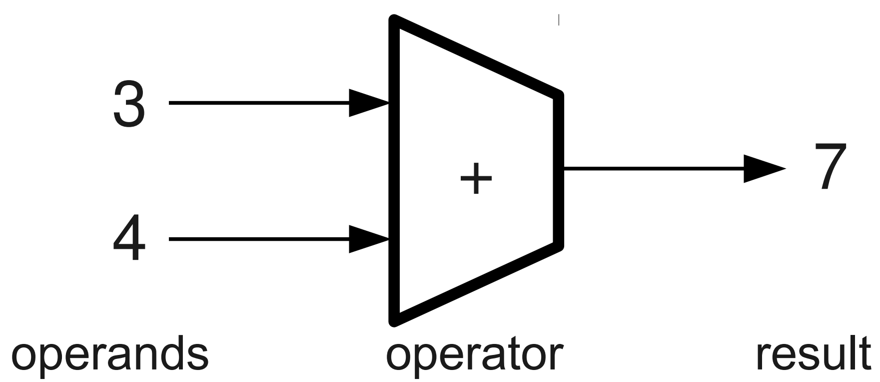
Since Boolean algebra works with **values that can have two possibilities** and the basic ingredient of computers is the **binary digital-electronics 'port'***, **Boolean Algebra is very adequate for computer science and informatics**

*The physical implementation of the numbers 0 and 1 can be anything 'binary'
(0 / 5 V), (1 kΩ / 10 kΩ), (0 pC / 1 pC)
Just a matter of **convention**. (Ex. RS232: “1” = -12 V, “0” = +12 V)

Boolean 'operator'

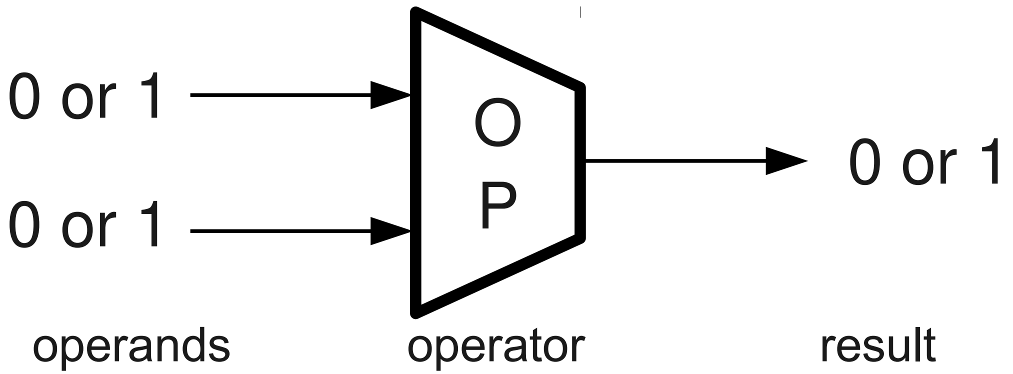
Compare with $y = 3 + 4$

$y = \mathbf{3 + 4}$ **Operation**: 'adding two numbers'
$y = 3 \mathbf{+} 4$ **Operator**: '+'
$y = \mathbf{3} + \mathbf{4}$ **Operands**: the objects used in the operation
$y = \mathbf{3 + 4}$ **Expression**: Something resulting in a value
$\mathbf{y = 3 + 4}$ **Instruction**. Attributing a value to a variable



Visual representation of the expression consisting of the single operation 'adding two numbers' with the two operands '3' and '4' resulting in the value '7'

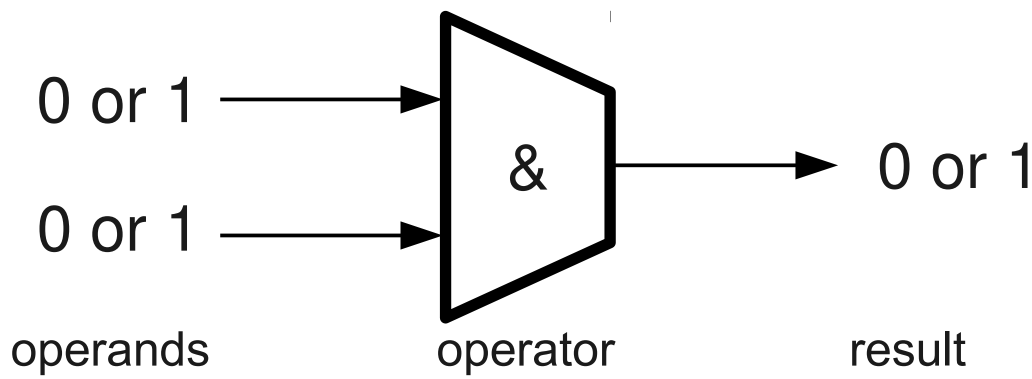
Boolean 'operator'



For Boolean Algebra operations with 2 inputs and 1 output, there exist only **16** possible operations

We can put them in a so-called **truth table**, which specifies the output of the operation for all possible combinations of inputs

Boolean 'operator' AND; truth table

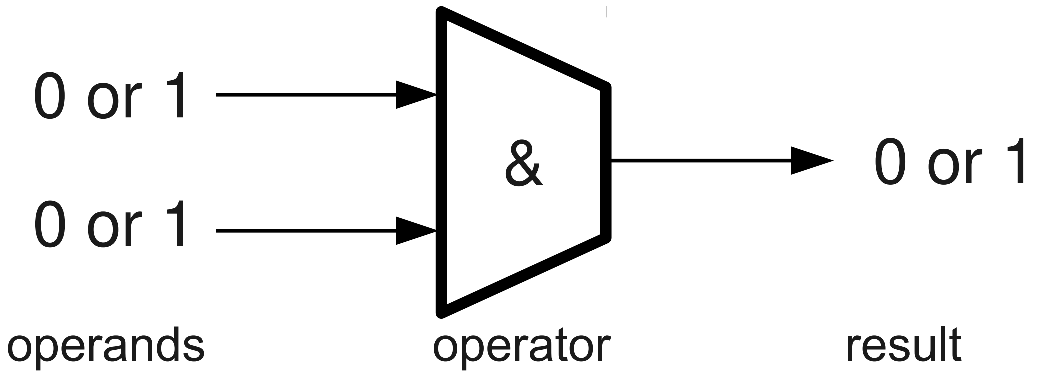



```
x y |x&y
---------
0 0 | 0
0 1 | 0
1 0 | 0
1 1 | 1
```

If we now use the convention that
'0' is by definition 'false'
'1' is by definition 'true'
we can say
"(x&y) is true if x is true **and** y is true"

This way we have a link to **human logic** and it explains the name for the operation '**and**'

All Boolean operations (2 in, 1 out)

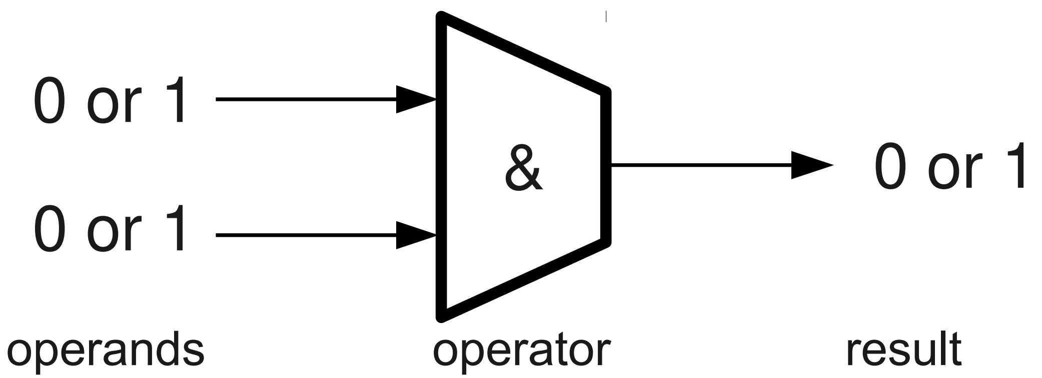



```
x y |                    out
--------------------------------------------------
0 0 |
0 1 |
1 0 |
1 1 |
```

How many different possibilities are there for "2-in,1-out" binary ports?*

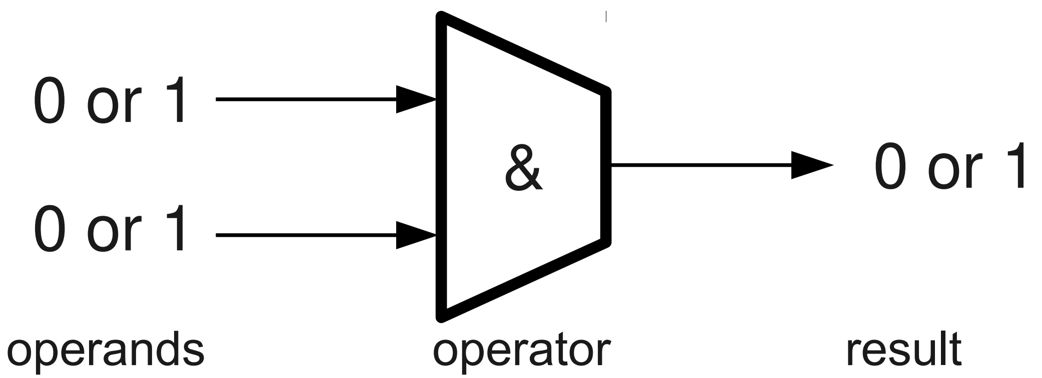
*: Homework: and for 2-in, 1-out *ternary* ports (0, 1, 2)?

All 16 Boolean operations (2 in, 1 out)



x	y	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

All 16 Boolean operations (2 in, 1 out)



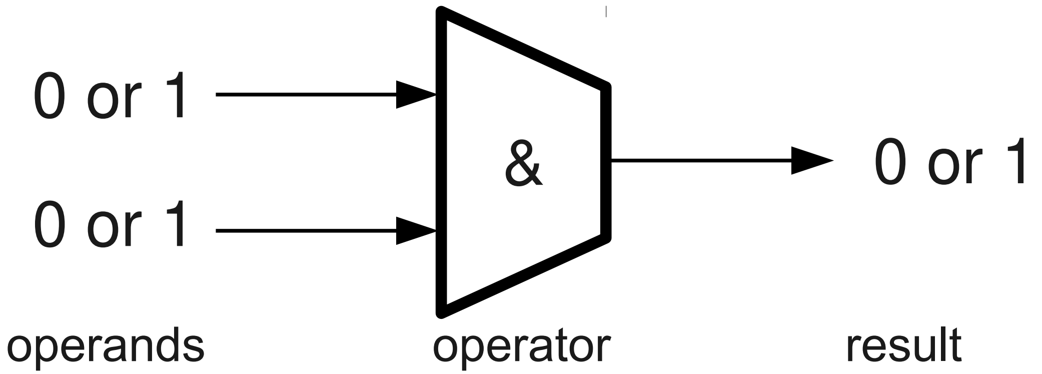
x	y		01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
0	0	|	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	|	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	|	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	|	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

AND → logic name of operation

& → symbolic name of operator

2 → number of effective operands

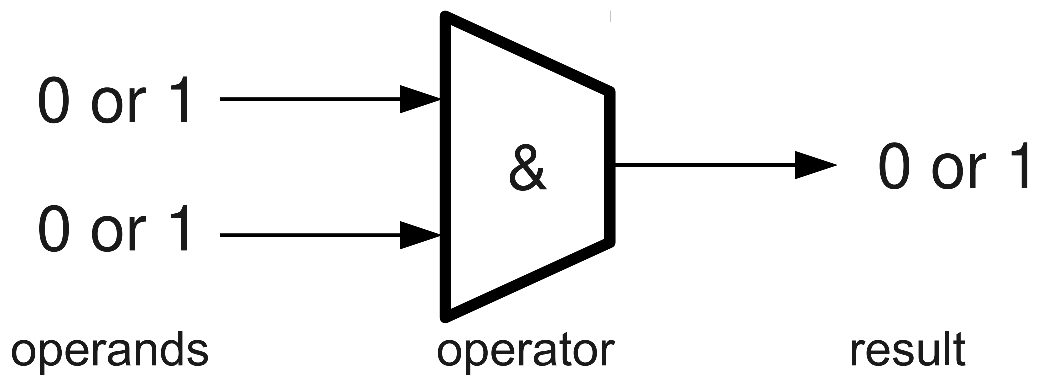
All 16 Boolean operations (2 in, 1 out)



x	y	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		0	AND														1
		0	&														1
		0	2														0

Silly! Operands not used

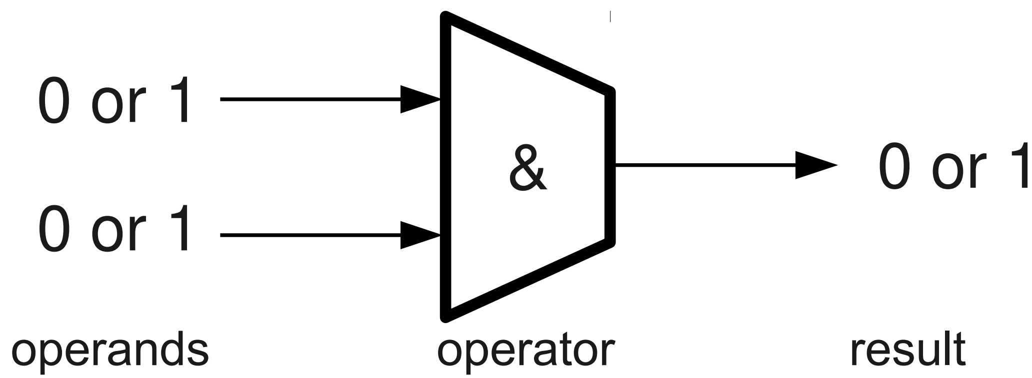
All 16 Boolean operations (2 in, 1 out)



<table>
<tr><th>x</th><th>y</th><th>01</th><th>02</th><th>03</th><th>04</th><th>05</th><th>06</th><th>07</th><th>08</th><th>09</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td></td><td></td><td>0</td><td>AND</td><td></td><td></td><td></td><td></td><td></td><td>OR</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td></tr>
<tr><td></td><td></td><td>0</td><td>&</td><td></td><td></td><td></td><td></td><td></td><td>|</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td></tr>
<tr><td></td><td></td><td>0</td><td>2</td><td></td><td></td><td></td><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr>
</table>

x or y true, or both

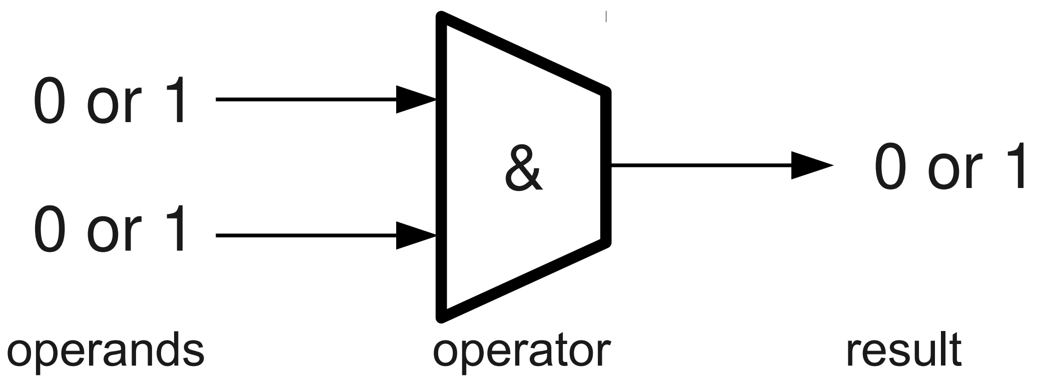
All 16 Boolean operations (2 in, 1 out)



x	y	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		0	AND					XOR	OR								1
		0	&					xor	|								1
		0	2					2	2								0

exclusive OR: x or y true, but not both

All 16 Boolean operations (2 in, 1 out)

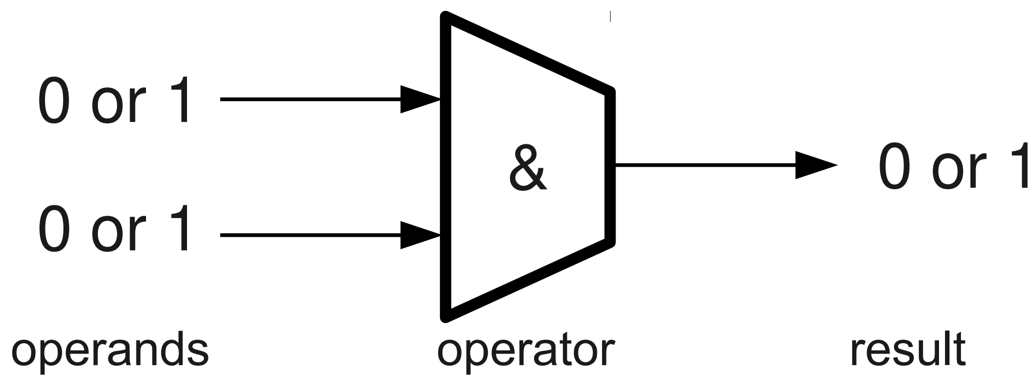


x	y	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		0	AND		X		Y	XOR	OR			NOT		NOT			1
		0	&		x		y	xor	|			!y		!x			1
		0	2		1		1	2	2			1		1			0

One operand. Copy (silly!)

Only one operand used. But useful!

All 16 Boolean operations (2 in, 1 out)

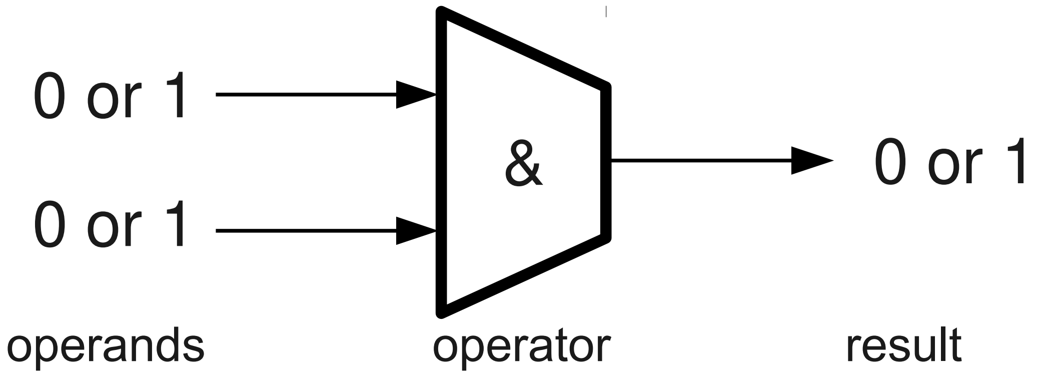


x	y	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Inverted versions

01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
0	AND		X		Y	XOR	OR	NOR	NXO	NOT		NOT		NAND	1
0	&		x		y	xor	|	!|	!xr	!y		!x		!&	1
0	2		1		1	2	2	2	2	1		1		2	0

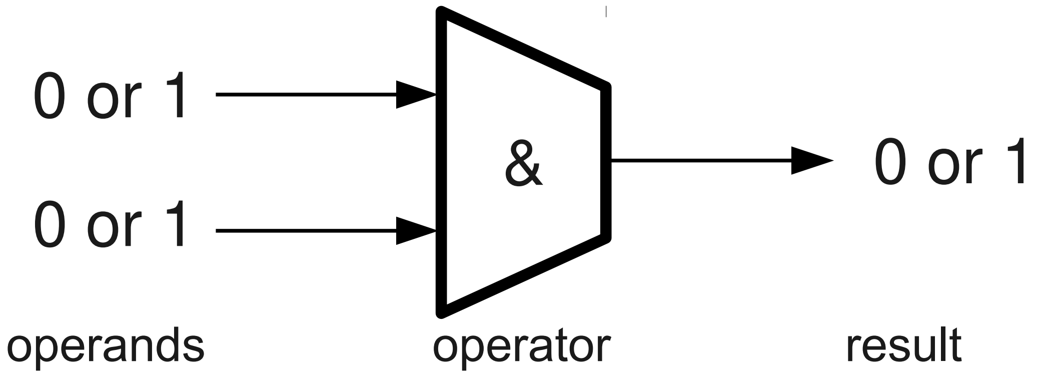
All 16 Boolean operations (2 in, 1 out)



x	y	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		0	AND		X		Y	XOR	OR	NOR	EQ	NOT		NOT		NAND	1
		0	&		x		y	xor	|	!|	==	!y		!x		!&	1
		0	2		1		1	2	2	2	2	1		1		2	0

x and y equal? (EQ)

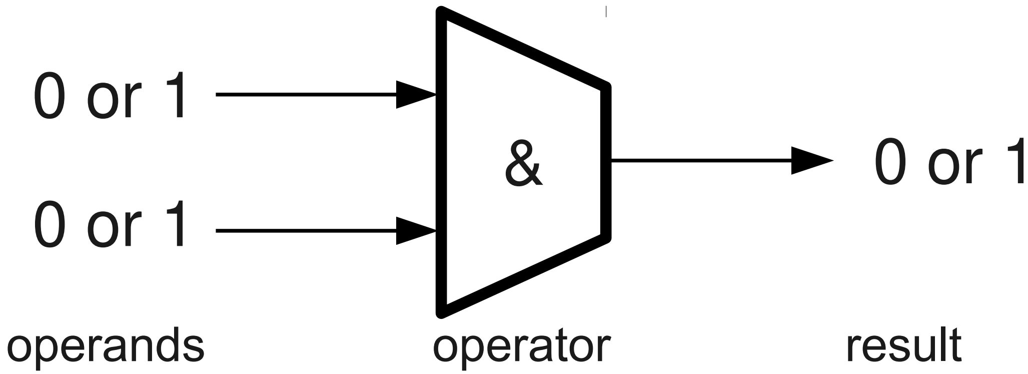
All 16 Boolean operations (2 in, 1 out)



x	y	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

The rest have no simple link to human logic
Example: case 14: “If x is true, copy y, else 1”

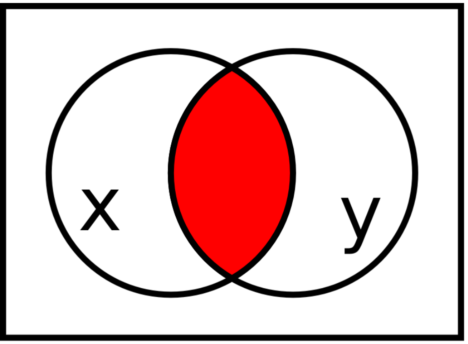
5 useful Boolean operations



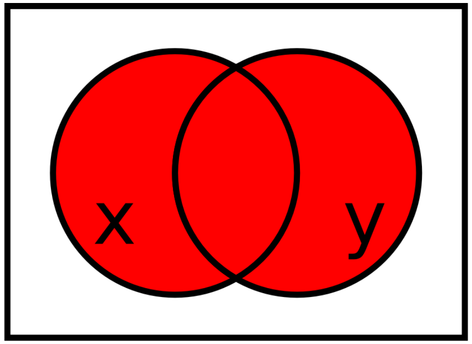
x	y	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
			AND					XOR	OR			NOT		NOT			

We remain with 5 useful Boolean operations for programming (MatLab)
EQ (==), AND (&), OR (|), XOR (xor), NOT (!)

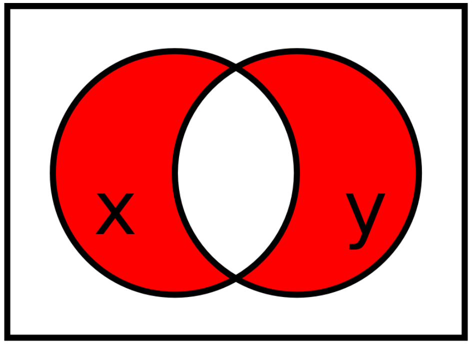
5 useful Boolean operations



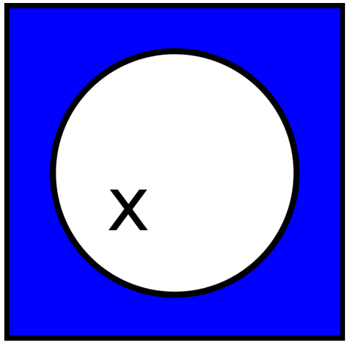
x&y



x|y



x xor y



!x

x	y	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
			AND					XOR	OR			NOT		NOT			

We remain with 5 useful Boolean operations for programming (MatLab)
EQ (==), AND (&), OR (|), XOR (xor), NOT (!)

Boolean Algebra in MatLab

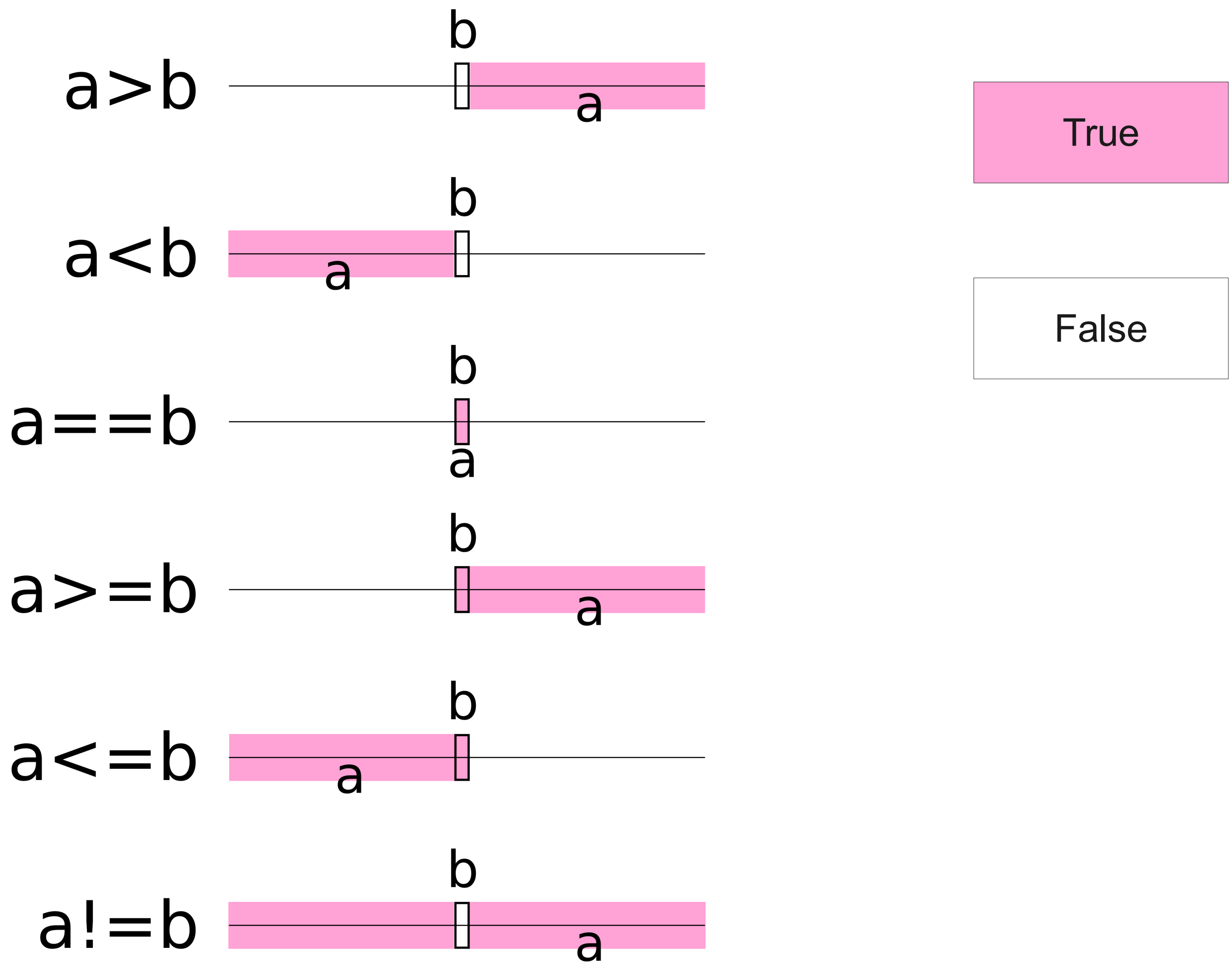
In MatLab:

false = 0

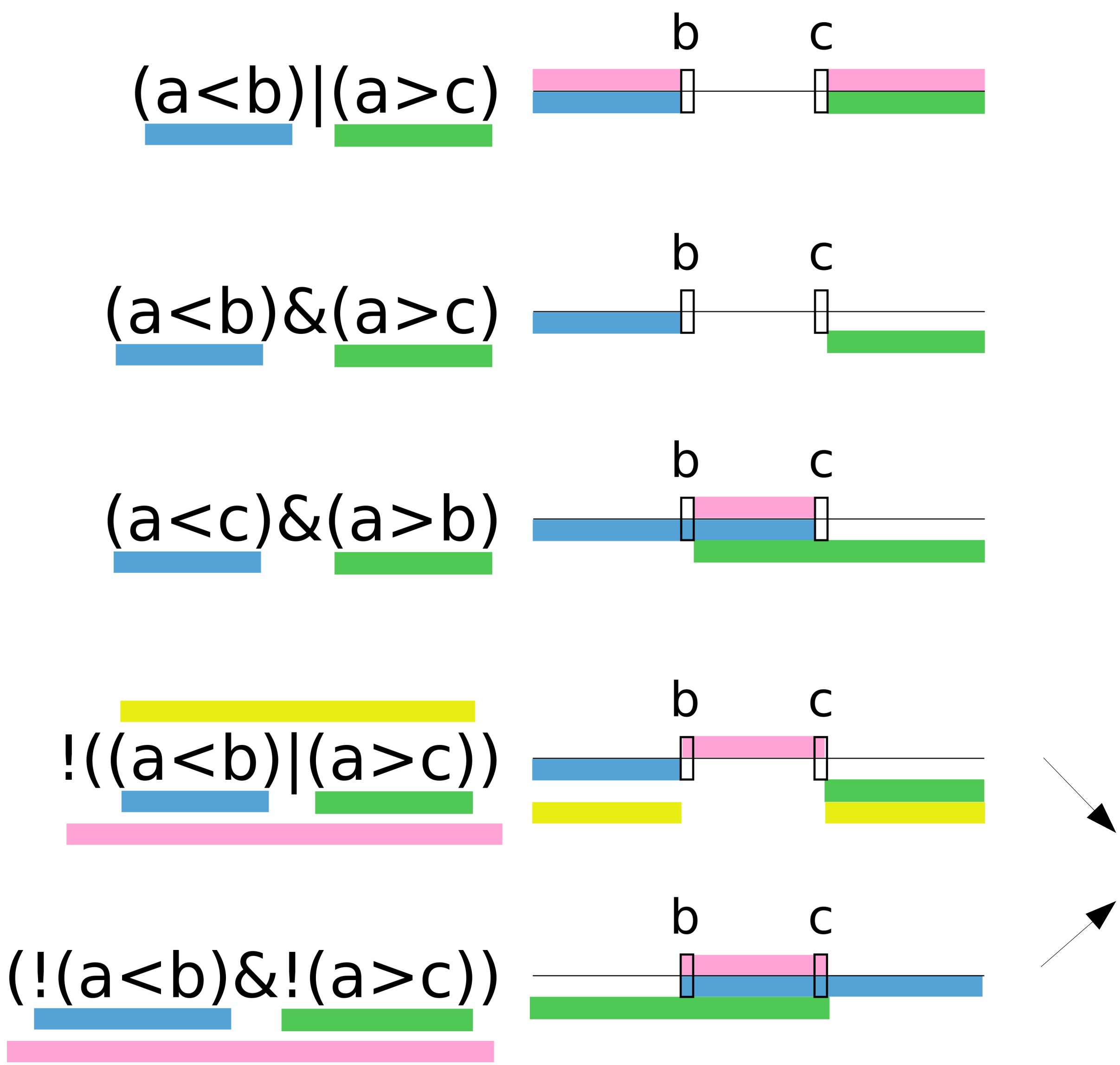
true = everything not 0

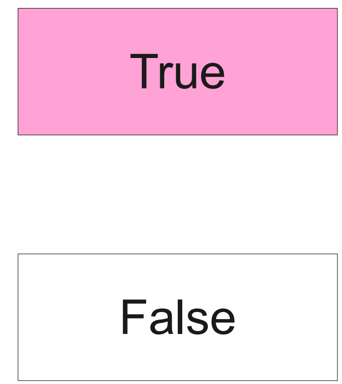
```
octave:1> !65
ans = 0
octave:2> !0
ans = 1
octave:3> 65|2
ans = 1
octave:3> 65&2
ans = 1
octave:3> 65|0
ans = 1
octave:3> 65&0
ans = 0
```

Boolean Algebra: Comparisons



Boolean Algebra: Comparisons





Example of De Morgan's Law
(Remember lectures of Digital Systems:
not(x or y) = not(x) and not(y))

Boolean Algebra: if

if (Boolean expression)
  MatLab instruction(s)
endif

```
x = 1;
if (x == 1)
  disp ("one");
elseif (x == 2)
  disp ("two");
else
  disp ("not one or two");
endif
```

*: 'disp' is 'display'

Boole vs. Shakespear showdown ...

"To be or not to be ....
... that is the question"

- William Shakespeare

```
(2==b) | !(2==b)
```

The answer of George Boole:
→ see exercises!