CS3102 Theory of Computation

www.cs.virginia.edu/~njb2b/cstheory/s2020

Warm up:

How might we compare models of computing?

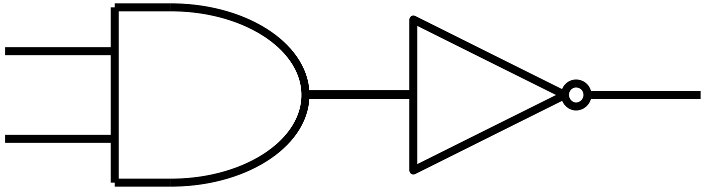
By what metrics might we say model A is “better” than model B?

Logistics

- Exercise 0 was due last week
  - Didn’t complete it? No problem (this time)! Just do it soon. Ask for an extension on the assignment page.
- First Quiz was due today
  - Didn’t complete it? No problem (this time)! Ask for an extension on the assignment page.
- Exercise 1 is out.

Last Time

- Boolean Circuits as a model of computing
- Components:
  - Inputs (how many?)
  - Gates (how many?)
  - Outputs (how many?)



- Important: Each circuit receives an input of a fixed size
  - Function of form $\{0,1\}^n \to \{0,1\}^m$ for n inputs and m outputs
  - What is the size of the domain?

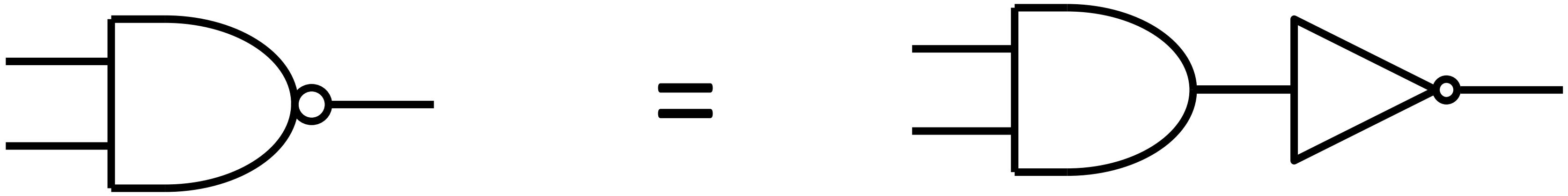
Defining the AON circuit model

- Define how to represent a computation
  - And/Or/Not circuit:
    - Number of inputs
    - Number of outputs
    - Gates and their labels
    - Wires connecting the above
- Define how to perform an execution
  - For each component, find its value once all its inputs are defined
  - Inputs start of with their value defined
  - Things labelled as output are the result

NAND with AON

- Build a circuit for $NAND$
  - $NAND(a, b) = \neg(a \wedge b)$

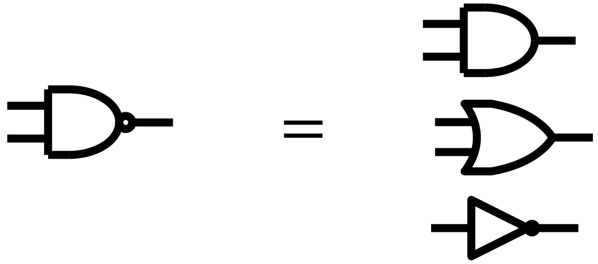
Input	Output
00	1
01	1
10	1
11	0



Today

- Comparing models of computing
  - And/Or/Not circuits vs NAND circuits
  - Circuits vs languages

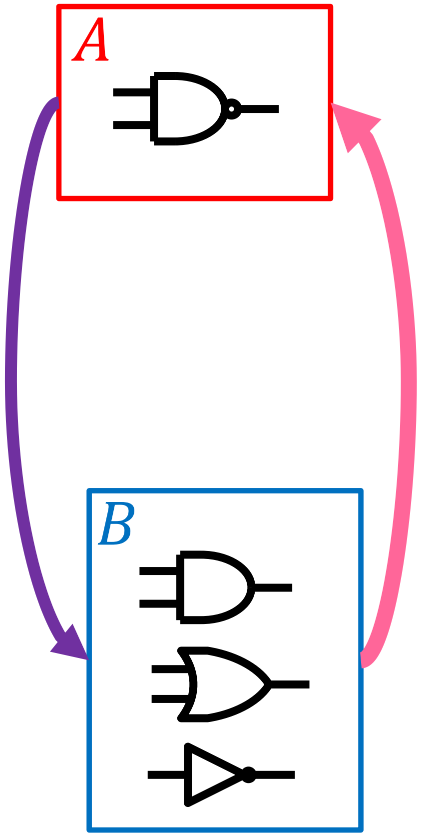
NAND Circuits



- The set of functions we can compute with $NAND$ gates only is the same as the set of functions we can compute with circuits AND, OR, NOT gates.
  - These computing models are "equivalent"
- How do we show this?

Equivalence of Computing Models

- Computing Model A and Computing Model B are "equivalent" if they compute the same set of functions
  – Any function that can be implemented with A can also be implemented with B, and vice-versa
- To show:
  – How to take an implementation of A and **convert it into** an implementation of B (which computes the same function)
  – How to take an implementation of B and **convert it into** an implementation of A (which computes the same function)



AND/OR/NOT using NAND

- AND

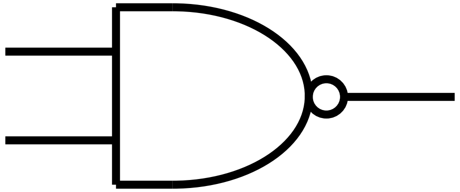
- OR

- NOT

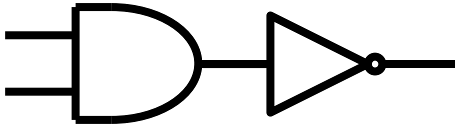
NAND = AON

NAND to AON

Everywhere
you see:

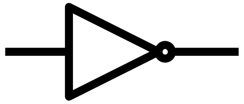


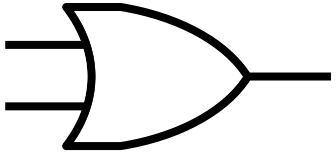
Instead put:

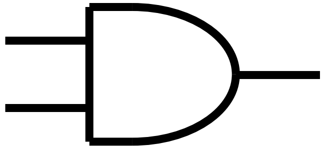


AON to NAND

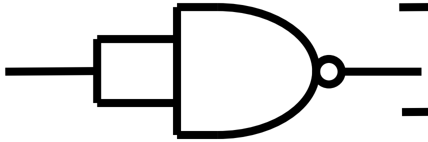
Everywhere
you see:

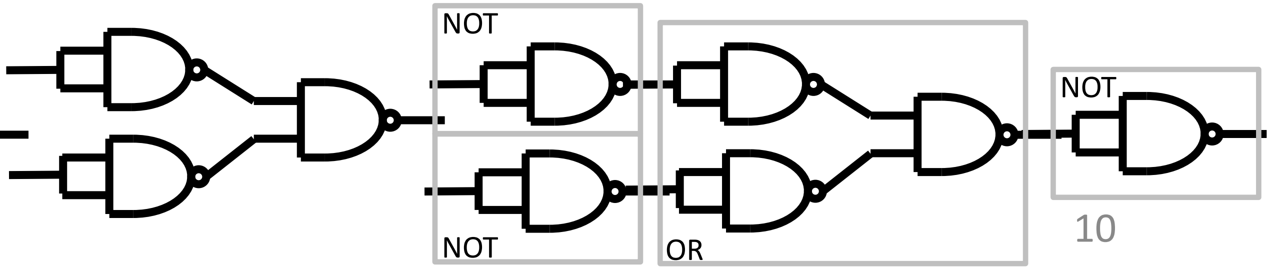




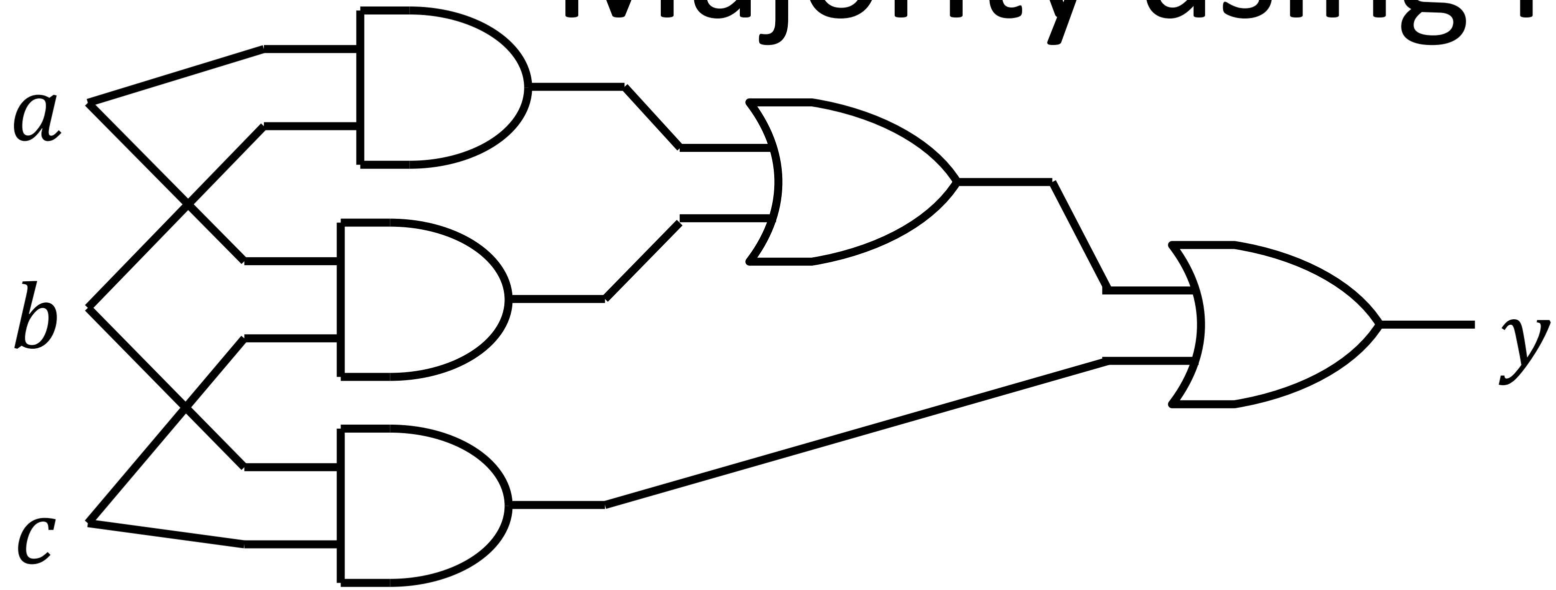


Instead put:

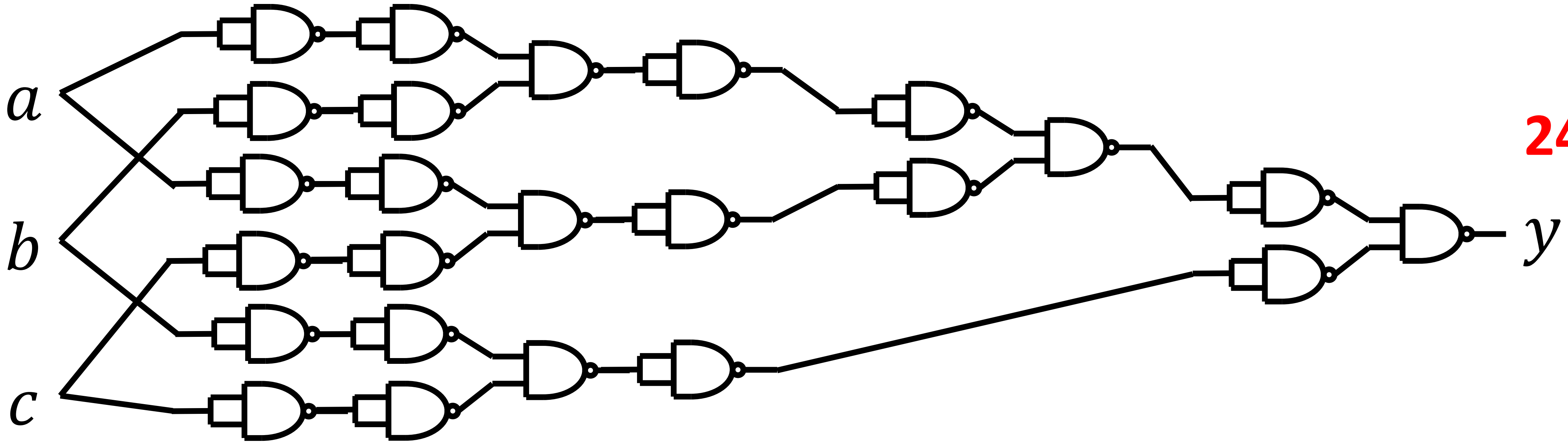




Majority using NAND



5 gates



24 gates

Takeaway

- We now have a hardware-based model of computing to work with
  – Actually two!
- Meant to be similar to how CPUs operate
- We've already made proofs about models of computation!
- While some models are equivalent in what they can compute, they may not be in how efficiently they can do it
- **Next time:** a software-like model of computing

A circuit-like programming language

- Define how to represent a computation
  - Inputs as positional arguments
  - Outputs as return statements
  - Variable assignments using boolean operators AND/OR/NOT
- Define how to perform an execution
  - Evaluate each variable assignment sequentially

AON-Straightline

MAJ with our language

- English:
  - Return 1 if at least 2 inputs are 1, 0 otherwise
- Math:
  - $MAJ(a,b,c) = (a \wedge b) \vee (a \wedge c) \vee (b \wedge c)$
- AON-CIRC:

```
def MAJ(a, b, c):
    first_two = AND(a,b)
    last_two = AND(b,c)
    first_last = AND(a,c)
    temp = OR(first_two, first_last)
    return OR(temp, last_two)
```

With your neighbors

- Write AON-straightline programs:
  - $NAND(a, b)$
  - $XOR(a, b)$

$NAND$

Input	Output
00	1
01	1
10	1
11	0

XOR

Input	Output
00	0
01	1
10	1
11	0

With your neighbors

- Write AON-straightline programs:
  - $NAND(a, b)$

```
def NAND(a, b):
    a_and_b = AND(a,b)
    return NOT(a_and_b)
```

  - $XOR(a, b)$

```
def XOR(a, b):
    not_a = NOT(a)
    not_b = NOT(b)
    only_a = AND(a, not_b)
    only_b = AND(b, not_a)
    return OR(only_a, only_b)
```

$NAND$

Input	Output
00	1
01	1
10	1
11	0

XOR

Input	Output
00	0
01	1
10	1
11	0

AON-Straightline = NAND-Straightline

- Show any function I can implement in the AON-Straightline language can be implemented such that the only operation is NAND

NAND Straightline = AON Straightline

NAND -> AON

AON -> NAND

NAND Straightline = AON Straightline

NAND -> AON

x = NAND(a,b)

Becomes

temp = AND(a,b)

x = NOT(temp)

AON -> NAND

x = NOT(a)

Becomes

x= NAND(a,a)

x = AND(a,b)

Becomes

temp= NAND(a,b)

x=NAND(temp,temp)

x = OR(a,b)

Becomes

t1 = NAND(a,a)

t2 = NAND(b,b)

x= NAND(t1,t2)

Circuits equivalent to AON Straightline

- How do we show this?

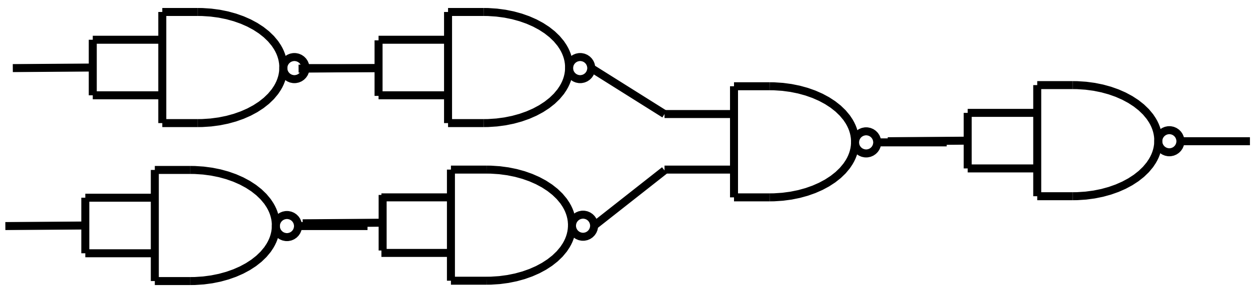
Convert Expression of each into the other

- NAND-Straightline Components
  - Inputs as positional arguments
  - Outputs as return statements
  - Variable assignments using boolean operator NAND
- NAND-Circuit Components
  - Number of inputs
  - Number of outputs
  - Gates and their labels
  - Wires connecting the above

Circuit to Straightline

- Inputs as positional arguments
  - Come from:
- Outputs as return statements
  - Come from:
- Variable assignments using boolean operator NAND
  - Come from:

Circuit to Straightline



Straightline to Circuit

- Number of inputs
  - Come from:
- Number of outputs
  - Come from:
- Gates and their labels
  - Come from:
- Wires connecting the above
  - Come from:

Straightline to Circuit

```
def AND(a,b):
    not_a = NAND(a,a)
    not_b = NAND(b,b)
    or1 = NAND(nota, nota)
    or2 = NAND(notb, notb)
    a_or_b = NAND(or1, or2)
    a_and_b = NAND(a_or_b, a_or_b)
    return a_and_b
```