CS 121: Lecture 4
Defining Computation: Circuits

Madhu Sudan

https://madhu.seas.Harvard.edu/courses/Fall2020

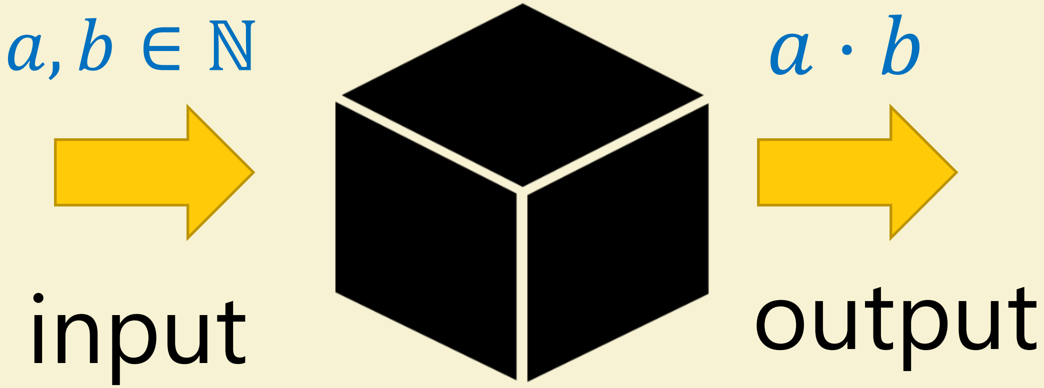
Book: https://introtcs.org

How to contact us {
- The whole staff (faster response): CS 121 Piazza
- Only the course heads (slower): cs121.fall2020.course.heads@gmail.com

Reminder

- Homework 1 due Thursday!
- CS 121.5: Sasha Golovnev on “circuit lower bounds” on Thursday. 
- Reminder: Sign up for active participation, Lectures 8-11.
- Other modes of participation: Sections+OH+Piazza!
    - TFs standing by!!

What



Function

How

Input: $a, b \in \mathbb{N}$

Operations:

$res \leftarrow 0$

for $i = 1 \dots \#digits(a)$:

  for $j = 1 \dots \#digits(b)$:

    $res \leftarrow res + 10^{i+j} a_i b_j$

return res

Formula/Algorithm/
Program/Circuit/..

How

Input: $a, b \in \mathbb{N}$

Operations:

$res \leftarrow 0$

for $i = 1 \ldots \#digits(a)$:

  for $j = 1 \ldots \#digits(b)$:

    $res \leftarrow res + 10^{i+j} a_i b_j$

return res

Formula/Algorithm/Program/Circuit/…

$f\colon \{0,1\}^n \to \{0,1\}^m$ finite function

Example:

x	$f(x)$
000	0
001	1
010	1
011	0
100	1
101	0
110	0
111	1

Compute f **:**

map x to $f(x)$ using sequence of "basic operations".

Today's goals

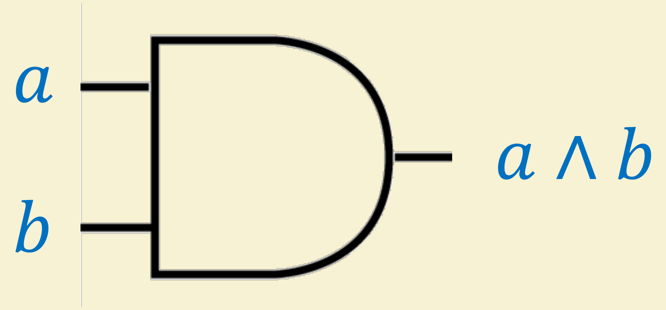
- Define "basic operations".
- Formally define *"f can be computed using the basic operations"*
- Formally define *"f can be computed using $\leq s$ operations"*

Basic Operations

$AND, OR : \{0,1\}^2 \to \{0,1\}, NOT : \{0,1\} \to \{0,1\}$

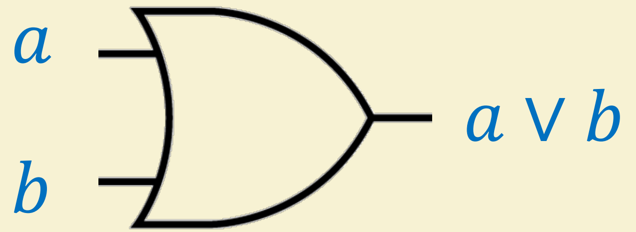
$$AND(a,b) = a \wedge b = \begin{cases} 1, & a = b = 1 \\ 0, & otherwise \end{cases}$$

a, b	$a \wedge b$
00	0
01	0
10	0
11	1



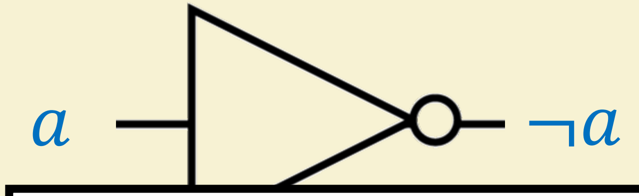
$$OR(a,b) = a \vee b = \begin{cases} 0, & a = b = 0 \\ 1, & otherwise \end{cases}$$

a, b	$a \vee b$
00	0
01	1
10	1
11	1



$$NOT(a) = \neg a = \overline{a} = \begin{cases} 1, & a = 0 \\ 0, & a = 1 \end{cases}$$

a	$\neg a$
0	1
1	0



Continue on Jupyter

Circuit = Sequence of Basic Operations

- Two equivalent ways of thinking:
  - As a graph
  - As a “straightline” program (no loops – simple sequence of instructions).

(AON)-Circuit Computing Majority

Circuit = Sequence of Basic Operations

Two equivalent ways of thinking:

As a graph

As a "straightline" program (no loops – simple sequence of instructions).

Exercise Break 1

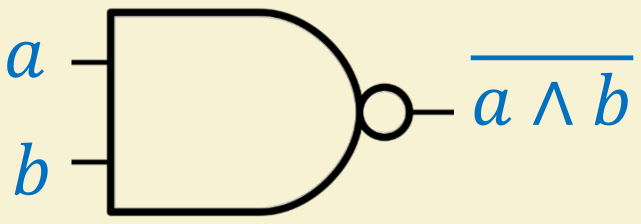
- Describe circuit computing Exactly-2:$\{0,1\}^3 \to \{0,1\}$:
  - Exactly-2$(x_0, x_1, x_2) = 1 \quad \Leftrightarrow \quad x_0 + x_1 + x_2 = 2$
  - How many gates did you use?

- Food for thought/If you have extra time:
  - How would you extend to circuit computing Exactly-m:$\{0,1\}^n \to \{0,1\}$ given by Exactly-$m(x_0 \ldots x_{n-1}) = 1 \Leftrightarrow x_0 + \cdots + x_{n-1} = m$.
  - How many gates would Exactly-m require? (say if $m \cong \frac{n}{2}$)

NAND Operation

$$NAND(a,b) = \overline{a \wedge b} = \begin{cases} 0, & a = b = 1 \\ 1, & otherwise \end{cases}$$

a, b	$\overline{a \wedge b}$
00	1
01	1
10	1
11	0



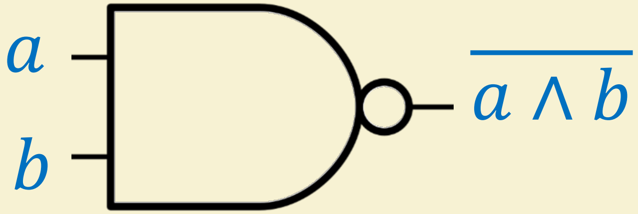
Exercise (Part 1): Show how to compute $NAND$ using AND, OR, NOT

Corollary: If we can compute f using combinations of $NAND$ then we can compute f using AND/OR/NOT

NAND Operation

$$NAND(a,b) = \overline{a \wedge b} = \begin{cases} 0, & a = b = 1 \\ 1, & otherwise \end{cases}$$

a, b	$\overline{a \wedge b}$
00	1
01	1
10	1
11	0



Exercise (Part 2): Show how to compute (1) NOT , (2) AND , (3) OR using NAND

Corollary: If we can compute f using combinations of $AND/OR/NOT$ then we can compute f using $NAND$

f computable by $\leq s$ $NANDs$ ⇒ f computable by $\leq 2s$ $AND/OR/NOT$ ⇒ f computable by $\leq 6s$ $NANDs$

Exercise Break 2:

Exercise 1: Compute NOT, AND, OR using NAND

(how many gates per operation?)

Exercise 2: Compute NAND using AND, OR, NOT

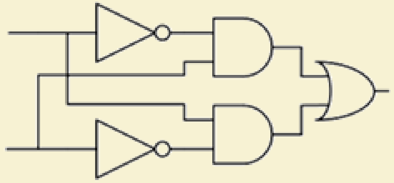
(how many gates per operation?)

Food for thought:

- Did you use all three gates in Exercise 2? If not what do you learn from this?

Boolean Circuits

(with ∧,∨, ¬ gates)



⇄

AON-CIRC straightline programs

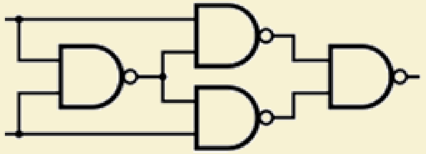
(with AND, OR, NOT operations)

```
t1     = AND(X[0],X[1])
notx0  = NOT(X[0])
t2     = AND(notx0,X[2])
Y[0]   = OR(t1,t2)
```

⇅ ⇅

NAND Circuits

(with $\overline{\wedge}$ gates)



⇄

NAND-CIRC straightline programs

(with NAND operation)

```
u = NAND(X[0],X[1])
v = NAND(X[0],u)
w = NAND(X[1],u)
Y[0] = NAND(v,w)
```

⇅ ⇅

...

Universality-1

- Have seen $\{AND, OR, NOT\} \equiv \{NAND\}$
  - $f: \{0,1\}^n \to \{0,1\}^m$ has $\{AND, OR, NOT\}$-circuit iff it has $\{NAND\}$-circuit
- Is $\{NOT\} \equiv \{NAND\}$?
- Is $\{AND, OR\} \equiv \{NAND\}$?
- Is $\{AND, NOT\} \equiv \{NAND\}$?
- Is $\{AND, XOR\} \equiv \{NAND\}$? Is $\{AND, XOR\} \equiv \{AND, OR, NOT\}$?
  - Not immediate, but same answer!

Universality-2

- Let $S = \{f_1, .., f_\ell\}$ be Boolean functions, $f_i: \{0,1\}^{k_i} \to \{0,1\}$.
- S-circuit is a sequence of operations where each operation is of the form $z = f_i(y_1, \dots, y_{k_i})$ - where $y_1, \dots, y_{k_i}$ input or previously computed.

- S is (NAND-)Universal iff there exists an S-circuit computing NAND(x_0, x_1).
- Can define $\{AND, OR, NOT\}$-Universal similarly.
  - ($\exists$ S-circuit computing AND, $\exists$ S-circuit computing OR, $\exists$ S-circuit computing NOT)
  - S is $\{AND, OR, NOT\}$-Universal iff S is $\{NAND\}$-Universal.
  - So … abbreviate to "S is Universal".

Summary:

- "Basic operations": $\{NAND\}$ (or equivalently $AON = \{AND, OR, NOT\}$).
- "f can be computed with basic operations": ∃ NAND-circuit computing f
  - Or equivalently NAND-CIRC program, or AON circuit, or AON-CIRC program.
- "f can be computed with $\leq s$ basic operations":
  - ∃ NAND-circuit program with $\leq s$ gates computing f
  - Or equivalently NAND-CIRC program with $\leq s$ lines
- S is <u>universal</u> iff $\exists S$-circuit computing NAND.
  - Exercise: S is universal $\Rightarrow \exists c$ such that the following holds for every f
    - if f can be computed with $\leq s$ basic operations, then f can be computed by an S-circuit with at most cs gates.

Next Lecture

- Every function $f:\{0,1\}^n \to \{0,1\}$ can be computed by basic operations.
- NAND-universality is really universal!!

- Every function f can be computed by circuit with $O(n2^n)$-gates. (complexity upper bound)
- Some (most!) functions require $\Omega\left(\frac{2^n}{n^2}\right)$-gates. (Complexity lower bound. Limits of circuits!)