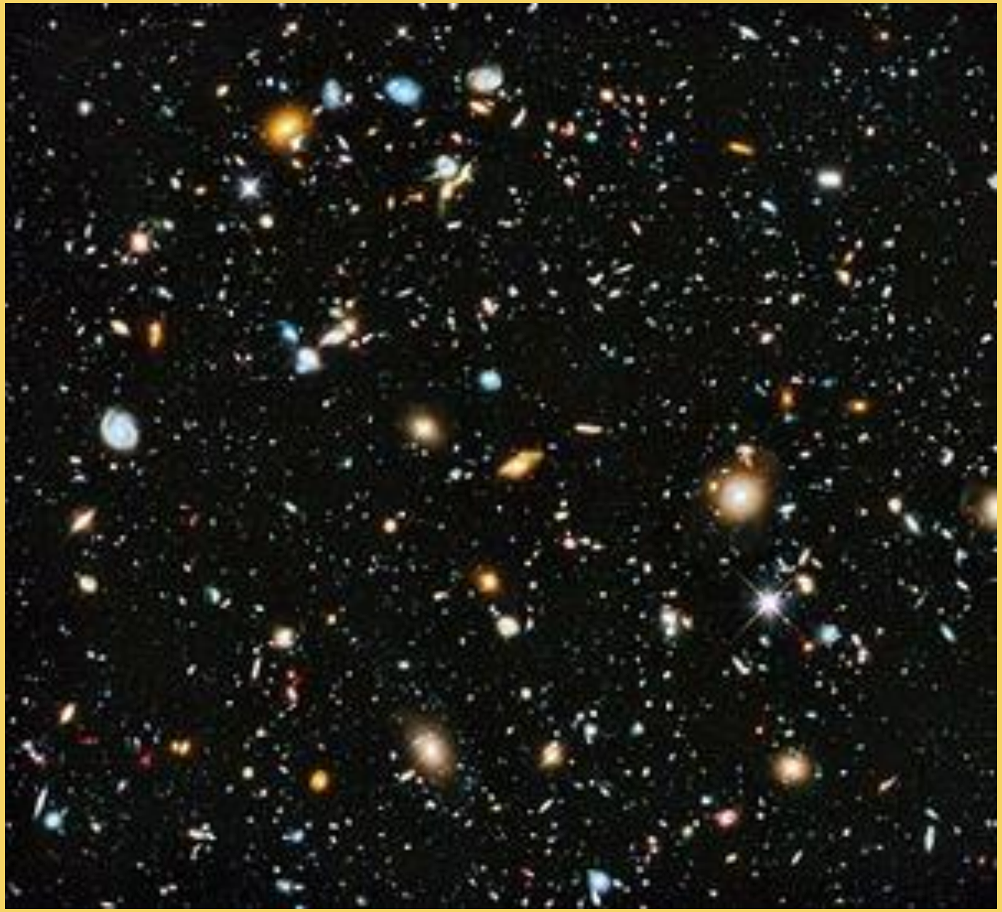


Photo: https://en.wikipedia.org/wiki/Universe

HW 4 due this Friday, Feb 20 (10:00pm)
Midterm 1 next Tuesday

Quiz 5?

Scope: reg. exp., DFA, NFA

Class 10: Universal Circuits

University of Virginia
CS3120: DMT2
https://weikailin.github.io/cs3120-toc
Wei-Kai Lin

Plan

Universal Circuits

Textbook [TCS] Section 3 and 4

https://introtcs.org/public/lec_03_computation.htm

Circuits and universal

Module 2: Circuits

A simple model of computation

Based on Boolean logical ‘gates’:

OR(a, b): outputs 1 iff a=1 or b=1

AND(a, b): outputs 1 iff a=1 and b=1

NOT(b): outputs 1 iff b=0

Output 0 otherwise

Write some “programs”

NAND(a, b) = NOT(AND(a, b))

a	b	NAND
0	0	1
0	1	1
1	0	1
1	1	0

XOR(a, b) = AND(OR(a, b), NAND(a, b))

a	b	XOR
0	0	0
0	1	1
1	0	1
1	1	0

A formal programming language

AON Straightline programs

Python-like language

Define functions that take Boolean inputs

Use AND/OR/NOT within

Assign results of AND/OR/NOT to variables

The result of variables can be used later as inputs

Return some of the obtained result(s) as output

More things to program

1-bit addition, ADD(a, b) :

c = OR(a, b)
d = AND(a, b)
e = NOT(d)
return = AND(c, e)

1-bit addition with carry, ADD(a, b) outputs (c, d)

carry = AND (a, b)

return (carry , AND (c, e))

Towards Algorithms

Example: “median"

Median is 1 if at least half of inputs are 1

Math definition of MED on 3 inputs: $MED(a_1, a_2, a_3)$

(a_1, a_2, a_3)

Computing MED using And/Or/Not

$p_1 = AND(a_1, a_2)$

$p_2 = AND(a_2, a_3)$

$p_3 = AND(a_3, a_1)$

$o_1 = OR(p_1, p_2)$

$o_2 = OR(o_1, p_3)$

return o_2

NAND Straightline Programs

Like AON straightline programs

Difference: we can only use NAND

YES

AND using NAND

OR ‐ ‐ ‐ ‐ —

NOT ‐ ‐ —

NOT(a):
return NAND(a, a)

AND(a, b):
c = NAND(a, b)
ret NOT(c)

NAND Straightline = AON Straightline

AON to NAND

x = NAND(a,b)

Becomes

temp = AND(a,b)

x = NOT(temp)

NAND to AON

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)

NAND Straightline = AON Straightline

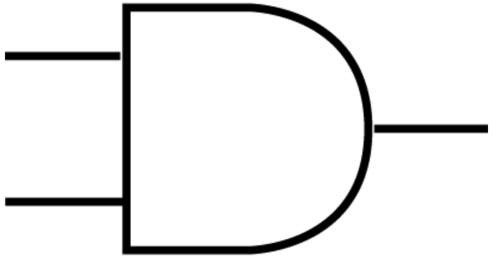
What could be benefits of each of them?

num of gates

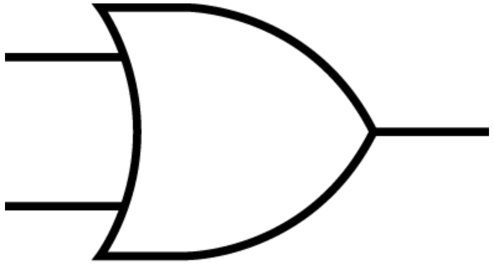
Circuits

Another approach: Boolean Circuits

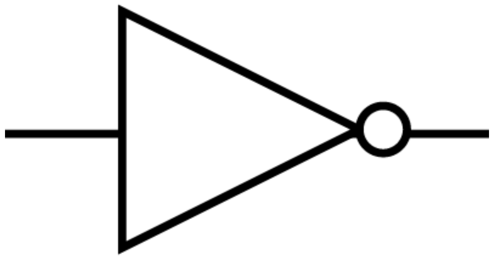
AND



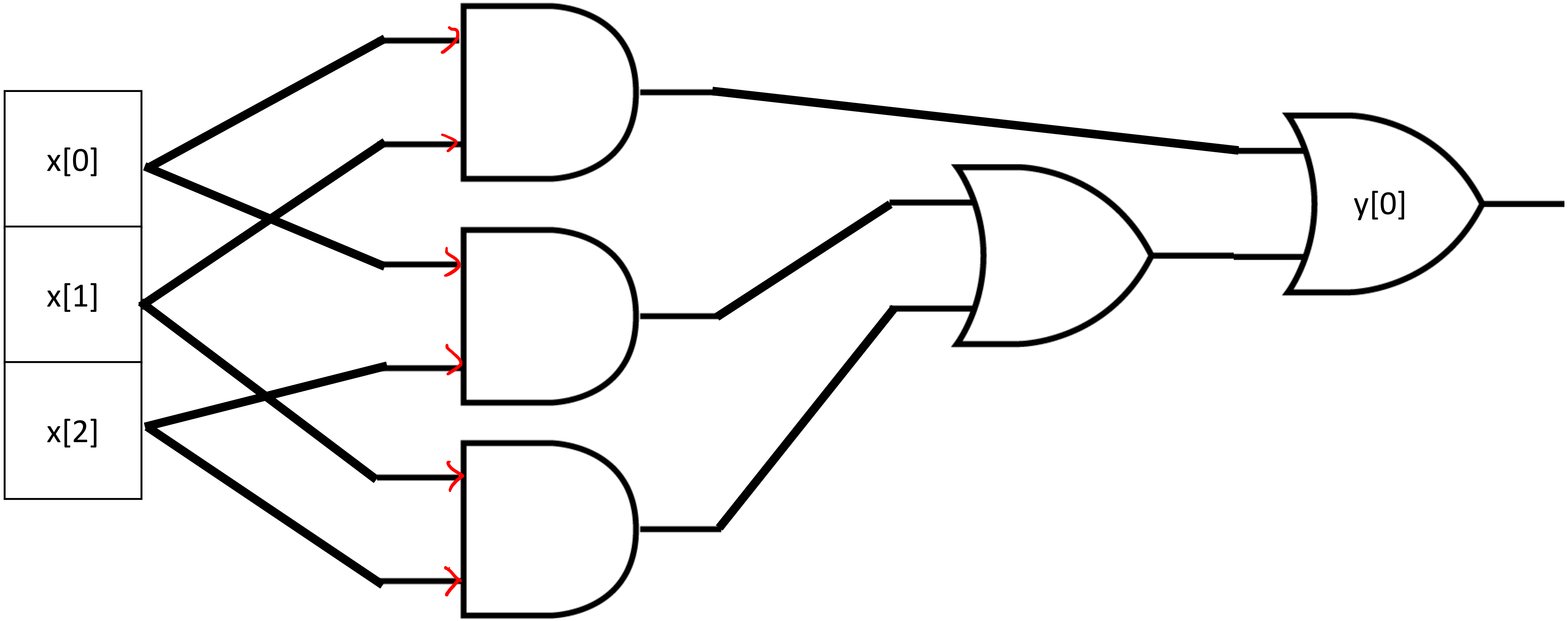
OR



NOT



Median with Boolean Circuits



Formal Definition of Boolean Circuits

A Boolean circuit with *n* inputs, *m* outputs, and *s* gates is a *directed acyclic graph*

Exactly *n* nodes have no in-neighbors (these are inputs, label them *x[0], …, x[n-1]*)

All remaining *s* nodes have a label AND, OR, NOT. AND and OR gates have two in-neighbors, NOT gates have one in-neighbor

Exactly *m* gates are denoted as outputs (label them y*[0], …, y[m-1]*)

Computing with a Boolean Circuit

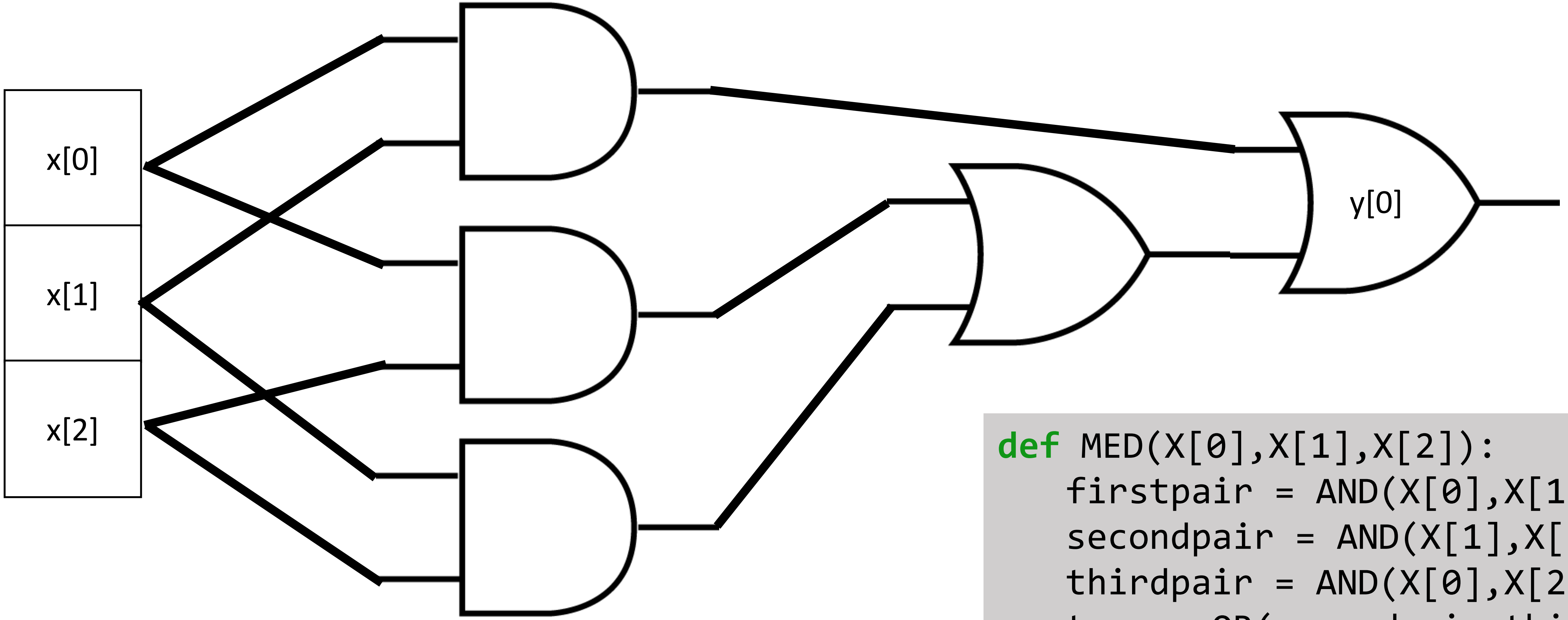
Assign gates into *layers* such that gate *x* appears before gate *y* whenever x has an outgoing edge that's an incoming edge of *y*

For each input node *x[i]*, assign its value to be bit *i* in the input

For each gate (considered in order of layers), assign as its value the result of its labelled operation applied to its in-neighbor(s)

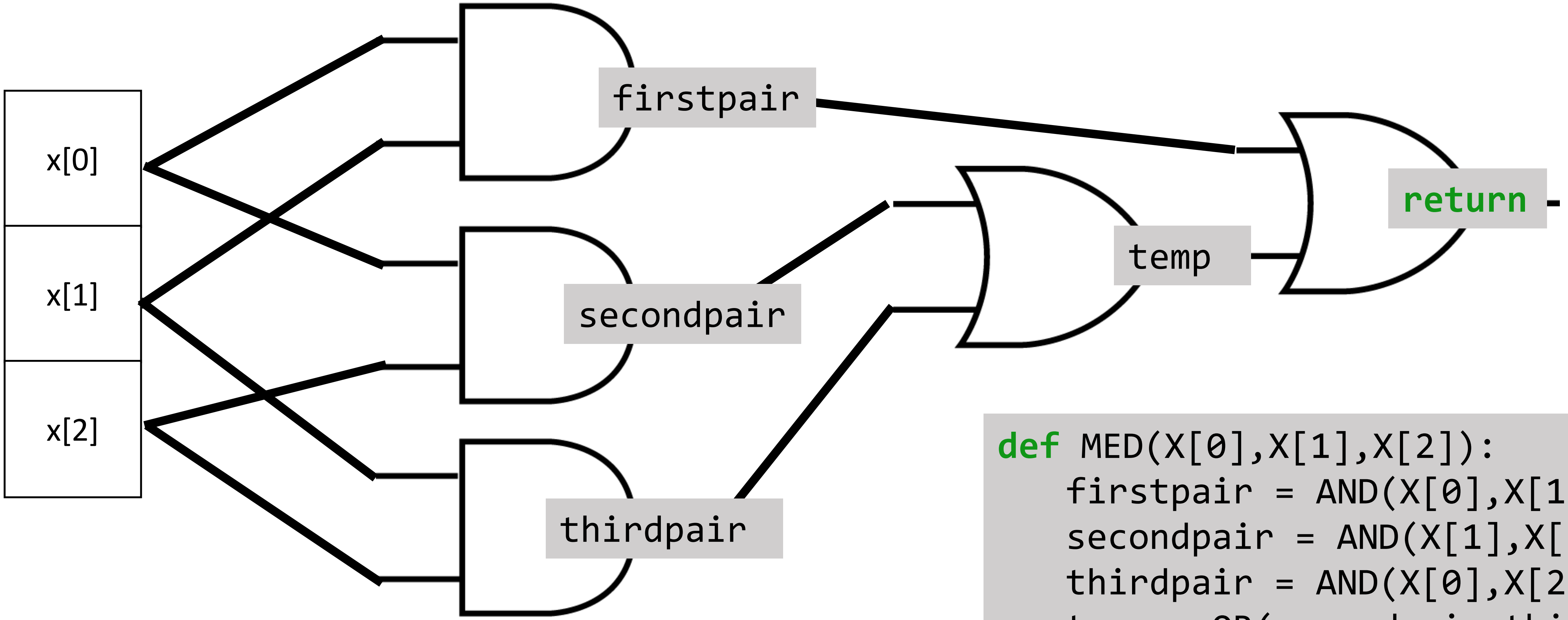
The result is the bit-string given by the values of the output gates

Median with Boolean Circuits
→ AON straightline program




```
def MED(X[0],X[1],X[2]):
    firstpair = AND(X[0],X[1])
    secondpair = AND(X[1],X[2])
    thirdpair = AND(X[0],X[2])
    temp = OR(secondpair,thirdpair)
    return OR(firstpair,temp)
```

Median with Boolean Circuits → AON straightline program




```
def MED(X[0],X[1],X[2]):
    firstpair = AND(X[0],X[1])
    secondpair = AND(X[1],X[2])
    thirdpair = AND(X[0],X[2])
    temp = OR(secondpair,thirdpair)
    return OR(firstpair,temp)
```

Circuits equivalent to AON Straightline

How do we show this?

- **Show how to convert any circuit to an AON straightline that computes the same function**
- **Show how to convert any AON straightline to a circuit that computes the same function**

Circuit to Straightline (saw an example)

Circuit inputs are straightline inputs already

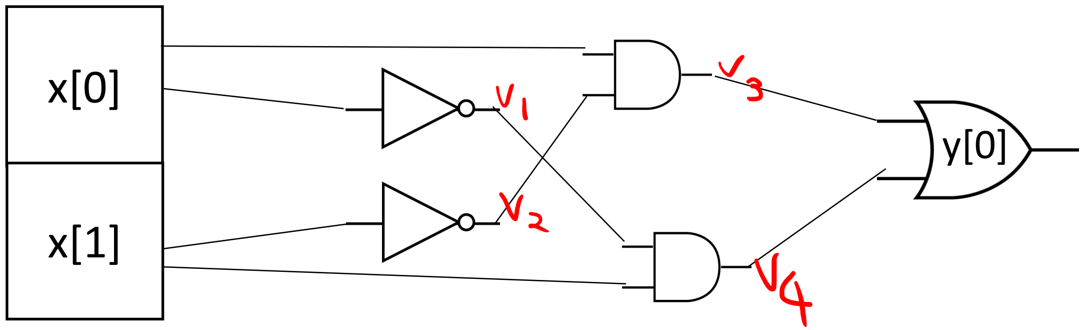
Each gate gets a variable

Value of that variable is result of applying the operation of that gate to the variables of the in-neighbors

Output is variable values of the output gates

Another example: XOR

Circuit



0	0	0
0	1	1
1	0	1
1	1	0

Program

```
ckt(x0, x1)
v1 = NOT(x0)
v2 = NOT(x1)
v3 = AND(x0, v2)
v4 = AND(x1, v1)
ret OR(v3, v4)
```

AON-Straightline to Circuit

Straightline inputs become circuit inputs

Each variable becomes a gate

The type of gate is given by the RHS of the assignment in the AON program

The in-neighbors of the gate are the gates represented by the operand variables

The output gates are the ones represented by return variables

Observations (2^{nd} one as important)

Everything function computable by a circuit is also computable by a straightline program (and vice-versa)

Every function computable by a straightline program with *s* variables is computable by a circuit with *s* gates (and the converse)

Universality

What does it mean for a Gate Set to be *Universal*?

Theorem 3.12 (NAND is a universal operation)

For every Boolean circuit C of s gates, there exists a NAND circuit C' of at most $3s$ gates that computes the same function as C.

Definition 3.20 (General straight-line programs)

Let $\mathcal{F} = \{f_0, \ldots, f_{t-1}\}$ be a finite collection of Boolean functions, such that $f_i : \{0,1\}^{k_i} \to \{0,1\}$ for some $k_i \in \mathbb{N}$. An $\mathcal{F}$ *program* is a sequence of lines, each of which assigns to some variable the result of applying some $f_i \in \mathcal{F}$ to k_i other variables. As above, we use `X[` i `]` and `Y[` j `]` to denote the input and output variables.

We say that $\mathcal{F}$ is a *universal set of operations* (also known as a universal gate set) if there exists a $\mathcal{F}$ program to compute the function $NAND$.

Set of gates.

(Informal) Definition. We say a $\boxed{\text{computation model}}$ is ***universal*** if for **any** finite function $f: \{0,1\}^n \to \{0,1\}^m$, there is an "instance" of the model that computes f.

Theorem:
AON circuits is universal.

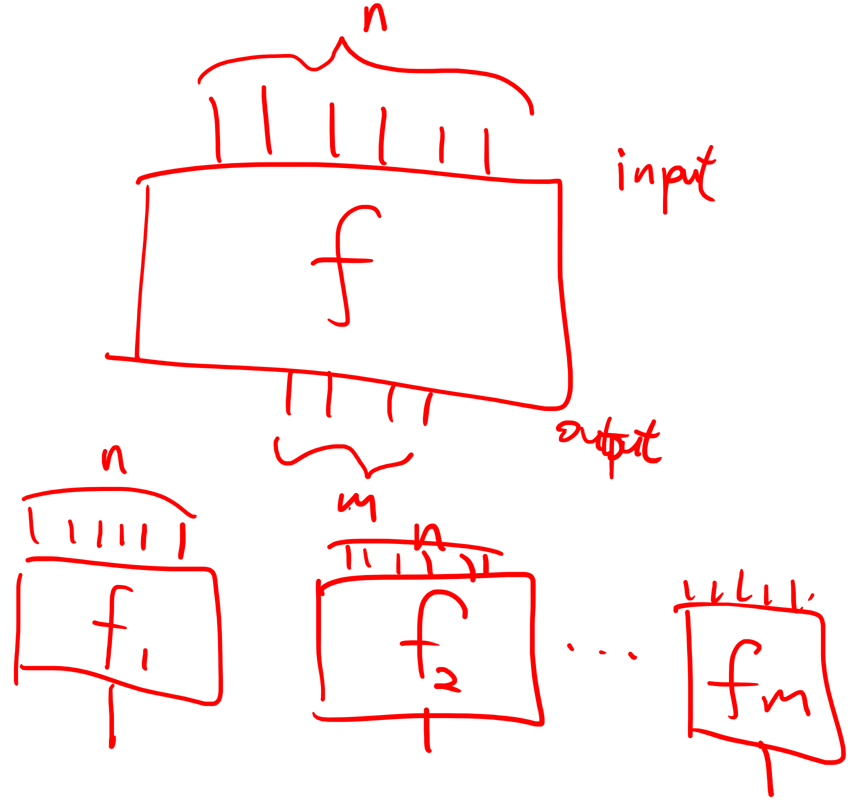
Corollary:
NAND is universal.

Goal: Compute $f: \{0,1\}^n \to \{0,1\}^m$

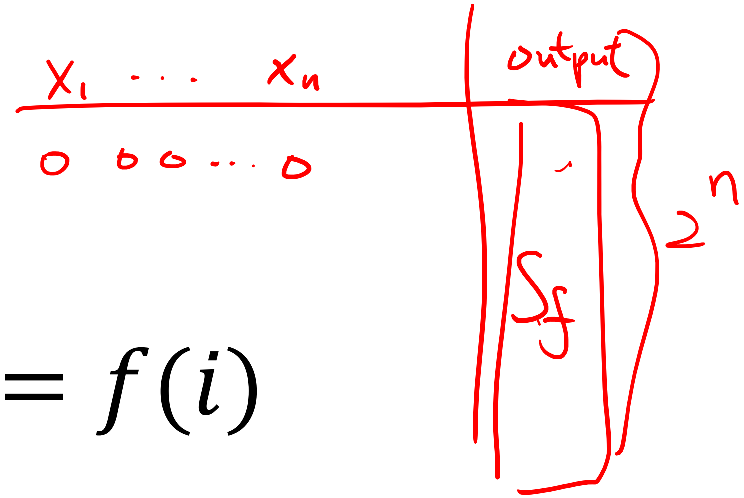
Let $f_1, f_2, \ldots, f_m$ be functions such that

$$f_i: \{0,1\}^n \to \{0,1\}$$

$f_i(x)$ **is the** i**th bit of** $f(x)$



Goal: Compute $f: \{0,1\}^n \to \{0,1\}$



(Also called "Boolean" functions)

Represent f as a string s_f:

$$s_f = b_0 b_1 \dots b_{2^n-1}, \qquad b_i = f(i)$$

We have $f(i) = s_f[i]$

n-bit number in ~~N~~ $[2^n] = \{0, 1, \dots, 2^n - 1\}$

Can we implement "array" using AON gates?

Want: LOOKUP(s, i) outputs s[i]

Array: LOOKUP$_k$($\overbrace{s}^{2^k}$, $\overbrace{i}^{k}$)

k determines len(s) and len(i)
No need comma as input

k: 1,2,3,…

s: 2^k-bit string, s = $b_0 b_1 \ldots b_{2^k-1}$

i: k-bit string, representing $0, 1, \ldots, 2^k - 1$

LOOKUP$_k$(s, i) outputs s[i] = b_i

Tool: IF(cond, a, b)

Output a if cond = 1

Output b if cond = 0

AON circuit implements IF

cond	a	b	IF(cond, a, b)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Circuit: LOOKUP_1(s, i)

k: 1,2,3,…

s = $b_0 b_1 \dots b_{2^k-1}$

i represent $0, 1, \dots, 2^k - 1$

outputs s[i] = b_i

$\text{LOOKUP}_1(\underset{2^1}{\text{s}}, \underset{1}{\text{i}}) = \text{LOOKUP}_1(b_0 b_1, \text{i})$

return IF(i, b_1, b_0)

input var of f

f itself

1(a)

= OR(a, NOT(a))

Circuit: LOOKUP$_2$(s, i)

k: 1,2,3,…

s = $b_0 b_1 \dots b_{2^k-1}$

i represent $0, 1, \dots, 2^k - 1$

outputs s[i] = b_i

LOOKUP$_2$($\underset{2^2}{}$s, $\underset{\text{2-bit}}{\underline{i}}$) = LOOKUP$_2$($\underbrace{b_0 b_1 b_2 b_3}_{s}$, $\underline{i}$) $= \underline{i_1}\ i_2$

IF(i_1, LOOKUP$_1$($b_2 b_3$, i_2),

LOOKUP$_1$($b_0 b_1$, i_2))

Circuit: LOOKUP_k(s, i)

k: 1,2,3,…

s = $b_0 b_1 \ldots b_{2^k-1}$

i represent $0, 1, \ldots, 2^k - 1$

outputs s[i] = b_i

LOOKUP_k(s, i) = $\text{LOOKUP}_k(b_0 b_1 \ldots b_{2^k-1}$, i)

Recurse!

Circuit: $\text{LOOKUP}_k(s, i)$

k: 1,2,3,…

s = $b_0 b_1 \ldots b_{2^k-1}$

i represent $0, 1, \ldots, 2^k - 1$

outputs s[i] = b_i

LOOKUP_k(s, i):

first_half = LOOKUP_{k-1}(s[0:2^{k-1}], i[1:k])

second_half = LOOKUP_{k-1}(s[2^{k-1}:2^k], i[1:k])

return IF(i[0], second_half, first_half)

in 10 gates of A/O/N

Size(LOOKUP_k) = $O(2^k)$

$S(k) = 2S(k-1) + O(1)$

$\cancel{S(1) = O(1)}$

Theorem:
AON circuit is universal.

$S(1) = 10$

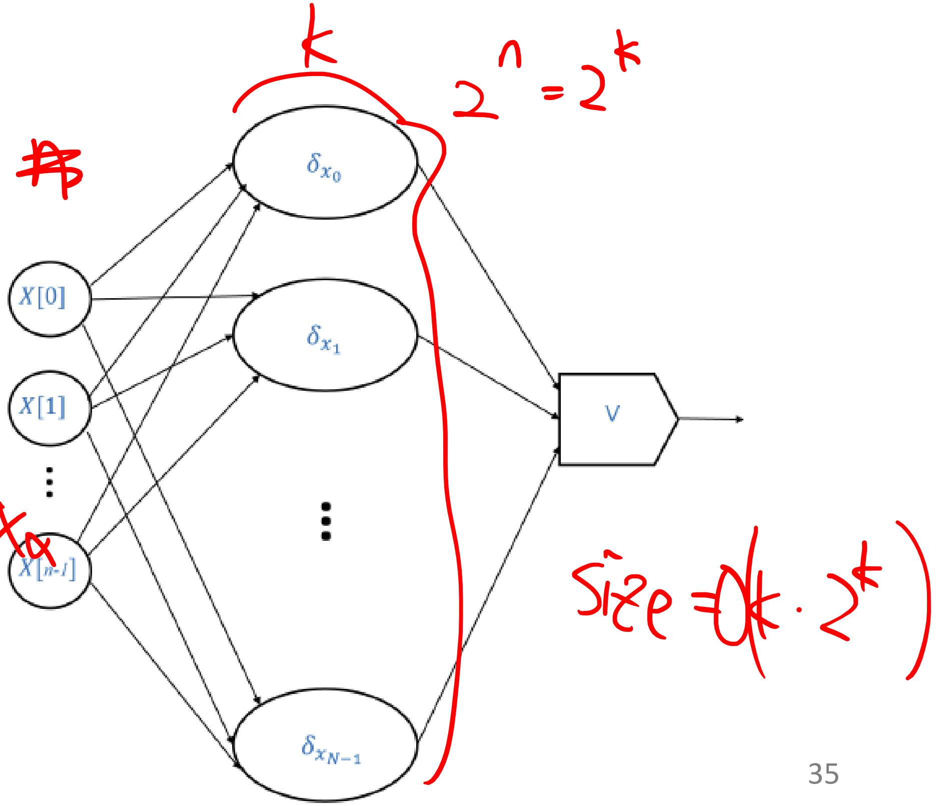
A simpler proof of universality of {AND,OR,NOT}

For $\alpha \in \{0,1\}^n$, the delta function $\delta_\alpha(X) = 1$ iff $x = \alpha$

Function $\delta_\alpha(X)$ can be computed by a circuit of size $O(n)$

To implement function f,
we use the following approach:

$$f(X) = \bigvee_{\alpha \in \{0,1\}^n,\ f(\alpha)=} \delta_{x_\alpha}$$



(Informal) Definition. We say a computation model is ***universal*** if for **any** finite function $f: \{0,1\}^n \to \{0,1\}^m$, there is an “instance” of the model that computes f.

Why do we want Universal Models?

Definition: We say that a gate set $\mathcal{G}$ is a *universal set of operations* (also known as a universal gate set) if there exists a $\mathcal{G}$ program to compute the function NAND.

(Textbook, Definition 3.20)

Plan

Midterm 1 review

Non-universal gates

Circuit size hierarchy

[TCS] Textbook, Section 3 to 4

- https://introtcs.org/public/lec_03_computation.html

HW 4 due this Friday, Feb 20 (10:00pm)
Midterm 1 next Tuesday, 9:30am, 20 min

Quiz 5 ~~next Mon~~ Sun