

Problem Set 5:
Will be posted this week

World's most boring fourth grade class		
Day 1	Today we'll learn to multiply 1 digit numbers 2×5	
Day 2	Today we'll learn to multiply 2 digit numbers 23×57	
Day 3	Today we'll learn to multiply 3 digit numbers 234×576	
Day 4	Today we'll learn to multiply 4 digit numbers 2345×5767	
	...	
Day 365	Today we'll learn to multiply 365 digit numbers $23456767... \times 57673432...$	

Class 11:

Universal Circuits

Deterministic Finite Automata.

University of Virginia
cs3120: DMT2
Wei-Kai Lin

Recap: Size Hierarchy Theorem

Theorem 5.5 (Size Hierarchy Theorem)

For every sufficiently large n and $10n < s < 0.1 \cdot 2^n / n$,

$$SIZE_n(s) \subsetneq SIZE_n(s + 10n) .$$



All n -bit functions, $\{0, 1\}^n \rightarrow \{0, 1\}$

$SIZE_n(\frac{2^n}{10n})$, many f.

$SIZE(s + 10n)$

Exists function here

$SIZE(s)$

$10n < s < 0.1 \cdot 2^n / n$

Exists function here

$SIZE(s + 20n)$

Exists function here, ...

$SIZE(s + 30n)$

Is it surprising?

- Functions are often $O(n)$, $O(n \log n)$, $O(n^2)$, ...

- Number of (n-bit input) functions?

$$2^{2^n}$$

- Number of “layers” = different sizes?

$$2^n / n$$

$$2^{2^n} \gg 2^n / n$$

Universal Circuits

Recap: Representing Circuits as Bits

```
def CIRCUIT(X[0],X[1]):
    temp2 = NAND(X[0],X[1])
    temp3 = NAND(X[0],temp2)
    temp4 = NAND(X[1],temp2)
    temp5 = NAND(temp3,temp4)
    return temp5
```

```
2, 1    // 2-bit in, 1-bit out
0, 1    // 1st line. 0 and 1 are input
0, 2    // 2nd line. 2, 3, ... are temp
1, 2    // 3rd line (and so on
3, 4
5        // return, one line per bit
```

1 line:

input & output lengths

0: n, m

ℓ lines:

one NAND per line

n: $v_{n,1}$, $v_{n,2}$

n+1: $v_{n+1,1}$, $v_{n+1,2}$

...

m lines:

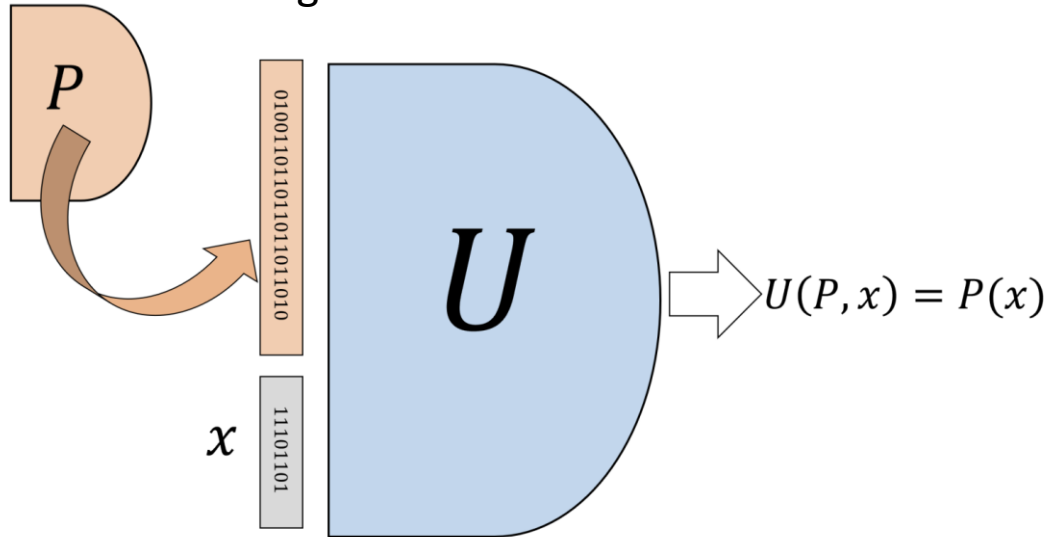
one output bit per line

last: $r_1, r_2, \dots r_m$

Universal Circuit/Program

Circuit

Figure 5.6 from TCS Book



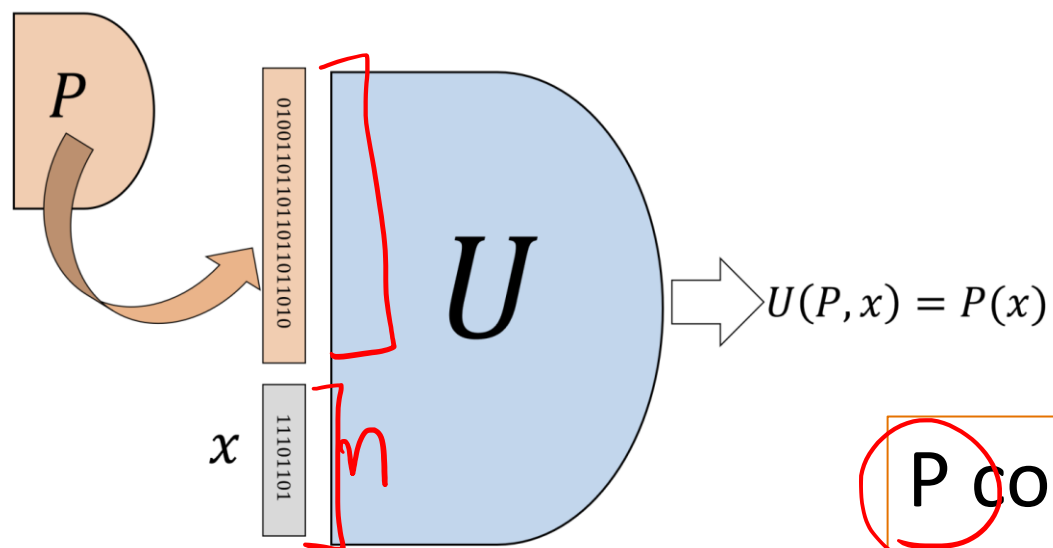
program U takes a program description P and input x as its input, and “simulates” running P on x :

$$U(P, x) = P(x)$$

Theorem 5.9 (Bounded Universality of NAND-CIRC programs)

For every $s, n, m \in \mathbb{N}$ with $s \geq m$ there is a NAND-CIRC program $U_{s,n,m}$ that computes the function $EVAL_{s,n,m}$.

Parameters of $U_{s,n}$



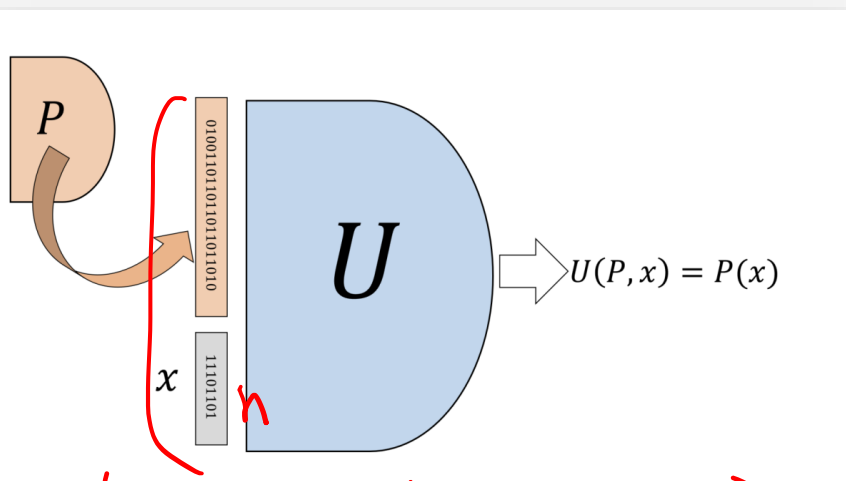
P computes $f: \{0,1\}^n \rightarrow \{0,1\}^m$

n : P computes n -bit input function

s : Circuit size (num lines) of P

(m : P outputs m bits; $m = 1$ this class)

Circuit Size of $U_{s,n}$



- Input size of $U_{s,n}$?

Intenlh, $t = |P| + |x| = O(s \log s) + n$

- Implement U via LOOKUP?

func t to 1 $\Rightarrow 2^t$

$\text{Size}(U) = 2^t$ (bad)

Efficient Universal Circuit?

- We want $U_{s,n}$ of smaller circuit size
Ideally, polynomial in $s = O(s^3)$
- Is that possible?
- Idea: write the “algorithm” in Python

Universal Circuit, Python version

```
def  $U_{s,n}(P = (g_0, g_1, \dots, g_s), x)$ :
```

```
# len(x) is  $n$ 
```

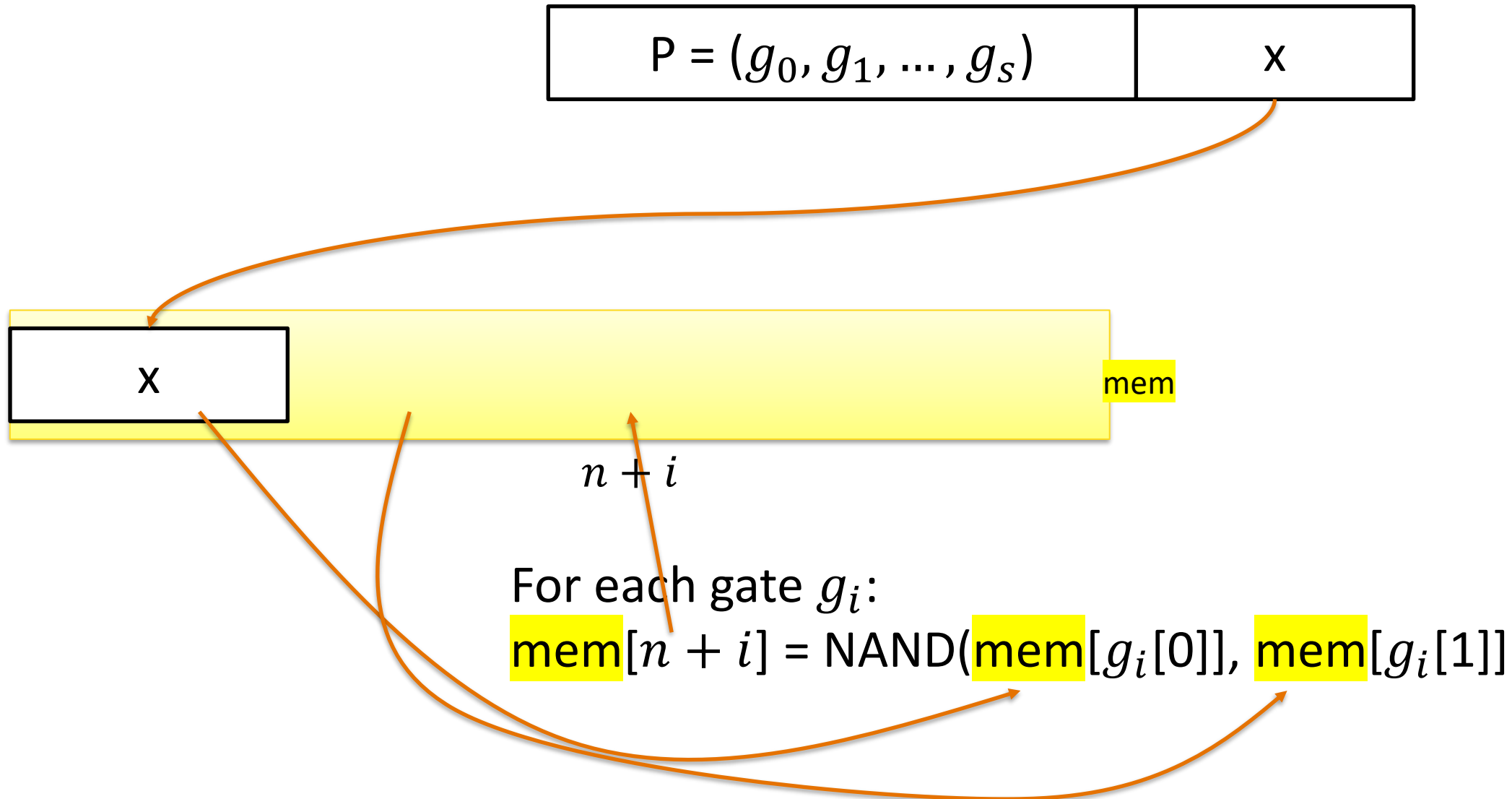
```
mem = x.append([0] *  $s$ )
```

```
for  $i$  in range( $s$ ):
```

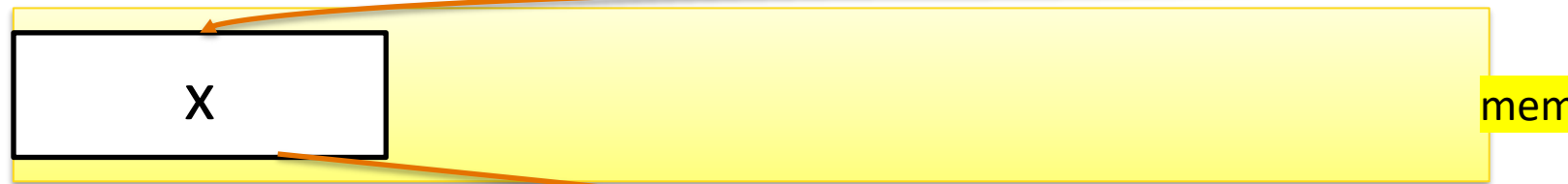
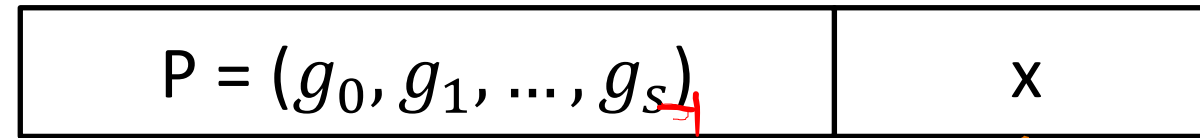
```
    mem[ $n + i$ ] = NAND(mem[ $g_i[0]$ ], mem[ $g_i[1]$ ])
```

```
return mem[ $n + s - 1$ ]
```

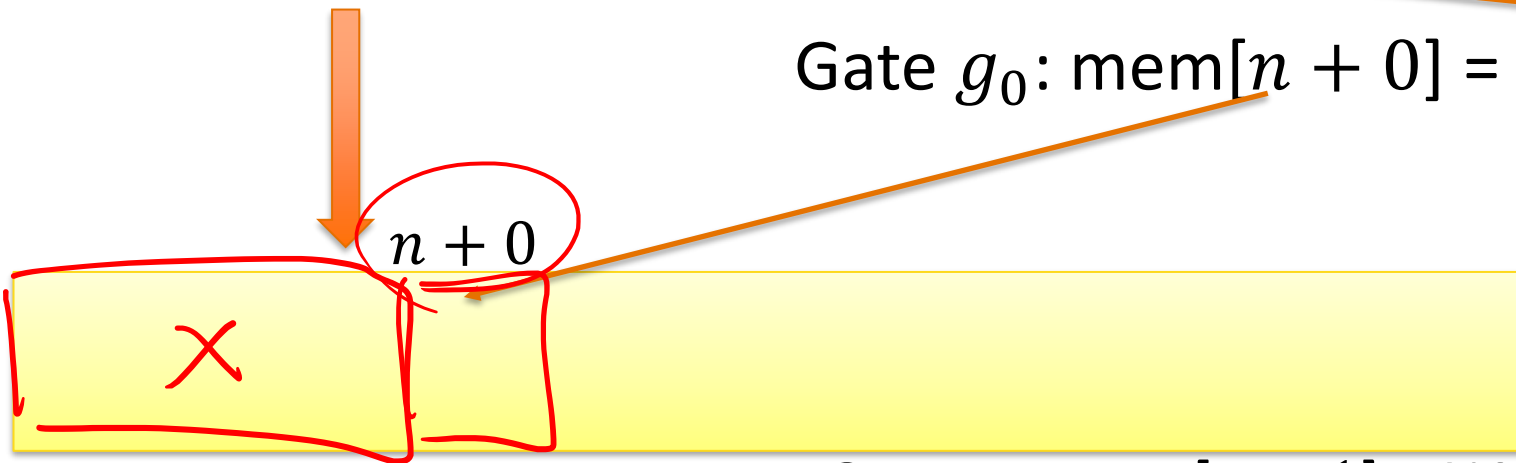
Universal Circuit, Picture version



Universal Circuit, full picture



Gate g_0 : $\text{mem}[n + 0] = \text{NAND}(\text{mem}[g_0[0]], \text{mem}[g_0[1]])$

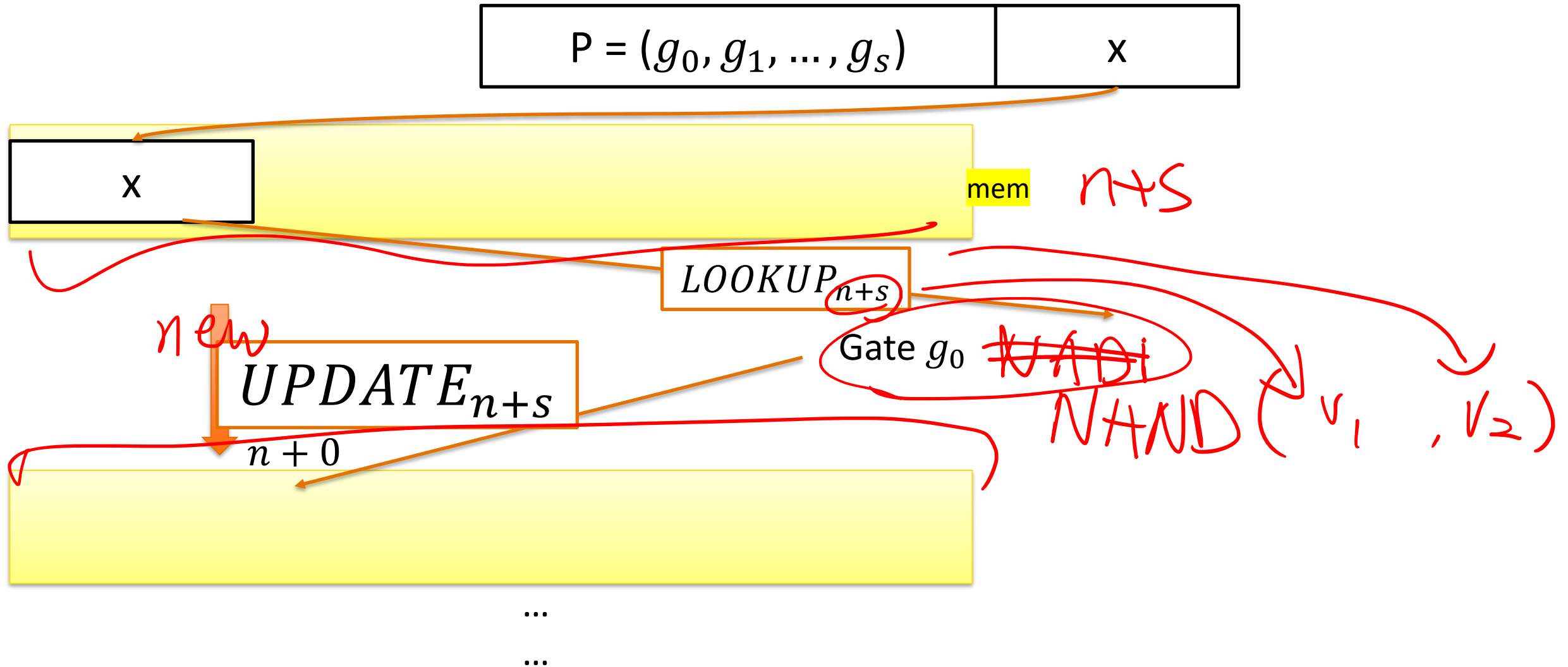


Gate g_1 : $\text{mem}[n + 1] = \text{NAND}(\text{mem}[g_1[0]], \text{mem}[g_1[1]])$

...

...

Universal Circuit, full picture



Size of Universal Circuit

- For each of s gates:
 - 2 LOOKUP, size $O(\ell)$, $\ell = s + n$
 - 1 NAND
 - 1 UPDATE, size $O(\ell \log \ell)$
- Total: $O(s\ell \log \ell) = O(s^3) \ll 2^s$

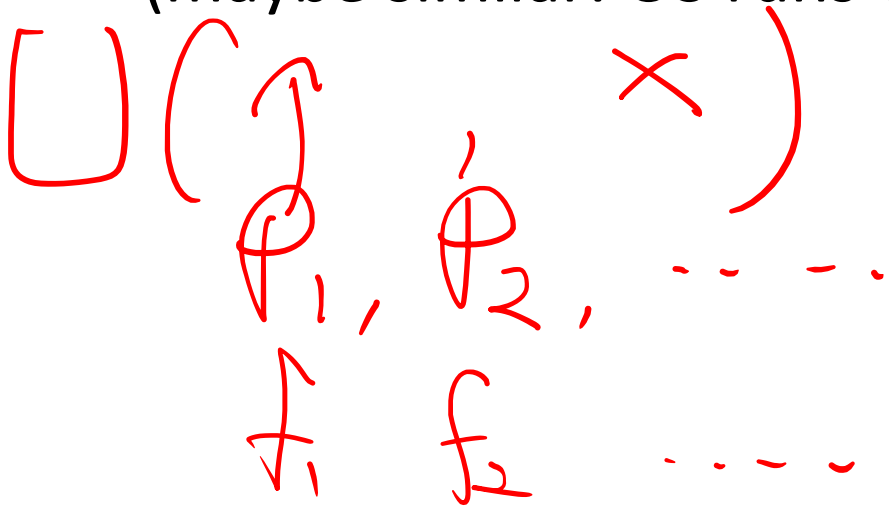
Why (implement) Universal Circuit?

Similar:

- Implement Python interpreter in Python
- Write C/C++ compiler in C/C++ ...

~~optimize~~ / optimize binary string
want search a ckt comp func
P: NULL

(maybe similar: OS runs virtual machine which runs another OS)



another: have many
ckt / func we want to copy

What we are seeing time and again is the notion of *universality* or *self reference* of computation, which is the sense that all reasonably rich models of computation are expressive enough that they can “simulate themselves”. The importance of this phenomenon to both the theory and practice of computing, as well as far beyond it, including the foundations of mathematics and basic questions in science, cannot be overstated.

Functions with Infinite Domains






So far...

We saw how to compute by Boolean circuits that use: AND, OR, NOT, NAND, etc. gates.

A Boolean circuit C_n has a **fixed** number n of input bits. Yet, it can compute **all** functions with fixed n input bits.

What if we do not know the input length at first?

We want *one* “algorithm” for multiplying numbers

World's most boring fourth grade class		
<div>Day 1</div> <div>Today we'll learn to multiply 1 digit numbers</div> <div>$\begin{array}{r} 2 \times \\ 5 \end{array}$</div> <div></div>	<div>Day 2</div> <div>Today we'll learn to multiply 2 digit numbers</div> <div>$\begin{array}{r} 23 \times \\ 57 \end{array}$</div> <div></div>	<div>Day 3</div> <div>Today we'll learn to multiply 3 digit numbers</div> <div>$\begin{array}{r} 234 \times \\ 576 \end{array}$</div> <div></div>
<div>Day 4</div> <div>Today we'll learn to multiply 4 digit numbers</div> <div>$\begin{array}{r} 2345 \times \\ 5767 \end{array}$</div> <div></div>	...	<div>Day 365</div> <div>Today we'll learn to multiply 365 digit numbers</div> <div>$\begin{array}{r} 23456767... \times \\ 57673432... \end{array}$</div> <div></div>

Functions with infinite input space

We would like to compute functions like multiplication

Formally, it would be a function $f : \{0,1\}^* \rightarrow \{0,1\}^*$

$\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \dots\}$ is an **infinite set**, but it **does not** contain any **infinitely long input**.

Satisfactory? $|\mathbb{R}| > |\{0,1\}^*|$
Can not comp MLL on \mathbb{R}

Focusing on Boolean functions

$f : \{0,1\}^* \rightarrow \{0,1\}$ with **one bit** of output is called a **Boolean** function.

In contrast a Boolean *circuit*, could have longer outputs, but it was called Boolean because its internal wires were Boolean.

more than 1 bit

Functions with longer than one output can be “Booleanized”

Example:

$$BMULT(x, y, i) = \begin{cases} i^{th} \text{ bit of } x \cdot y & i < |x \cdot y| \\ 0 & \text{otherwise} \end{cases}$$

Terminology: “Language”

Definition

A Boolean function $f : \{0,1\}^* \rightarrow \{0,1\}$ defines a corresponding set $L_f = \{x \mid f(x) = 1\} \subseteq \{0,1\}^*$

Any $L_f \subseteq \{0,1\}^*$ is also called a **language**.

So, $f(x) = 1 \iff$ “ x belongs to language L_f ”

TCS, 6.1.2: “This name (‘language’) is rooted in *formal language theory* as pursued by linguists such as Noam Chomsky.” It is wildly used.

Deterministic Finite Automata

A new simple computing model

Fixed constant-size memory

Reading input $x = x_1 \dots x_n$ as a stream of bits (once)

Decide if $f(x) = 1$ or not at the end

Example: detecting a particular sub-string

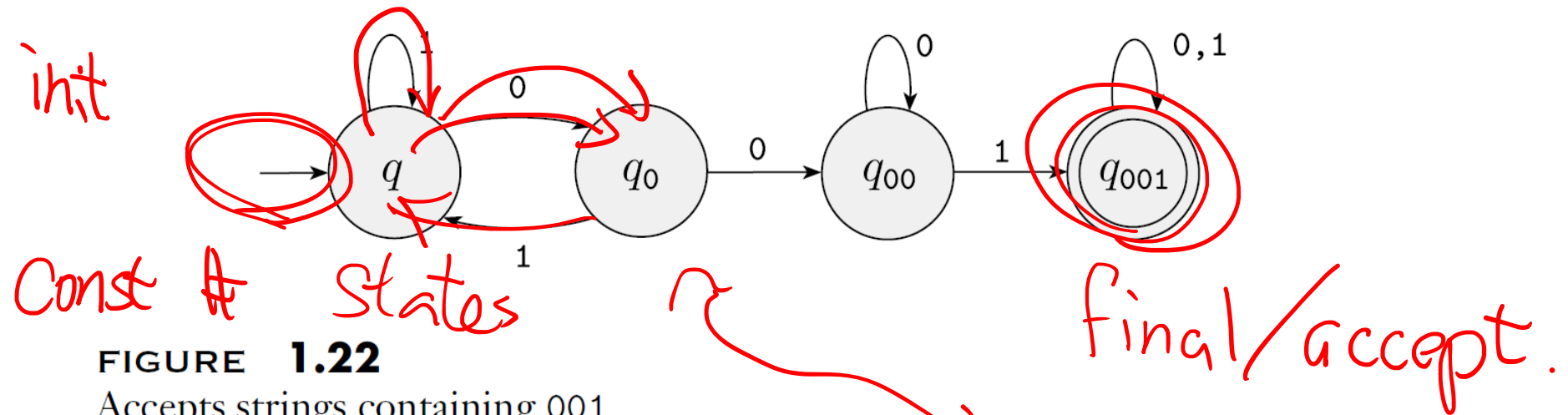


FIGURE 1.22

Accepts strings containing 001

$s = 0110$

$s' = 0010$

Another Example: XOR

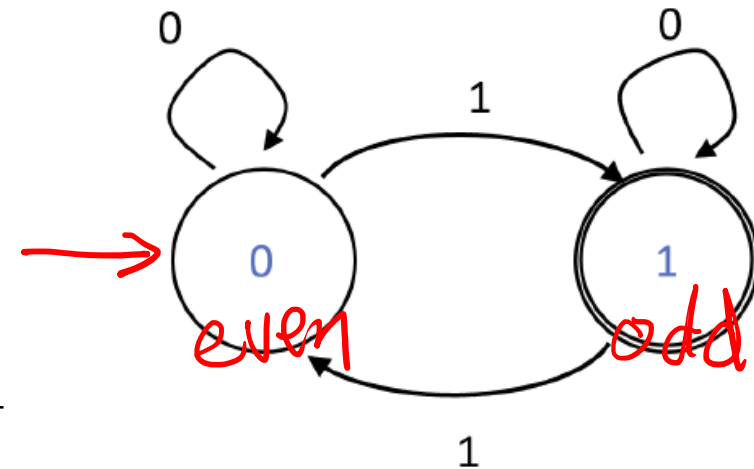
For $x = x_1 \dots x_n$ let $XOR(x) = x_1 \oplus x_2 \dots \oplus x_n$

$XOR(XOR(XOR(x_1, x_2), x_3), x_4) \dots$

As we read through the bits, we only need one bit of memory b that determines the XOR so far.

This can be depicted using a graph

- Two nodes (encoding the bit b)
- We start from $b = 0$
- We change b iff $x_i = 1$.
- We accept (output 1) if we end up at $b = 1$



Formal Definition of FA

A *finite automaton* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the *states*,

2. Σ is a finite set called the *alphabet*, $\{0, 1\}$

3. $\delta: Q \times \Sigma \rightarrow Q$ is the *transition function*,

4. $q_0 \in Q$ is the *start state*, and

5. $F \subseteq Q$ is the *set of accept states*.

diff def in TCS

Example:

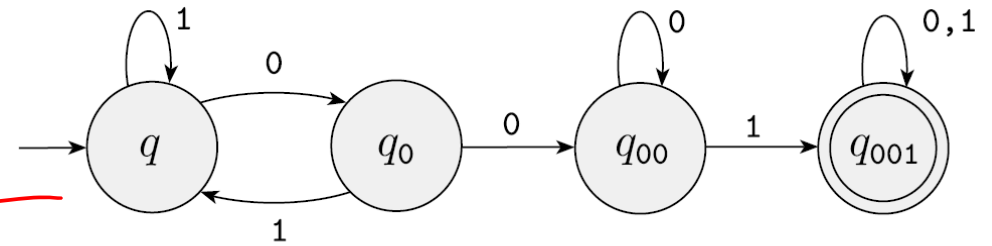


FIGURE 1.22
Accepts strings containing 001

1. $Q =$ q, q_0, q_{00}, q_{001}

2. $\Sigma =$ $\{0,1\}$,

3. δ is described as

		input	
		0	1
State	q	q	q_0
	q_0	q	q_{00}
	q_{00}	q_{00}	q_{001}
	q_{001}	q_{001}	q_{001}
			← accept

4. q is the start state, and

5. $F =$ q_{001}

Formal definition of FAs accepting inputs

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1w_2 \cdots w_n$ be a string where each w_i is a member of the alphabet Σ . Then M *accepts* w if a sequence of states r_0, r_1, \dots, r_n in Q exists with three conditions:

1. $r_0 = q_0$,
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \dots, n - 1$, and
3. $r_n \in F$.

Charge

Universal Circuits

Efficient implementation

Deterministic Finite Automata

Compute on infinite domains

PS5: will be posted this week

