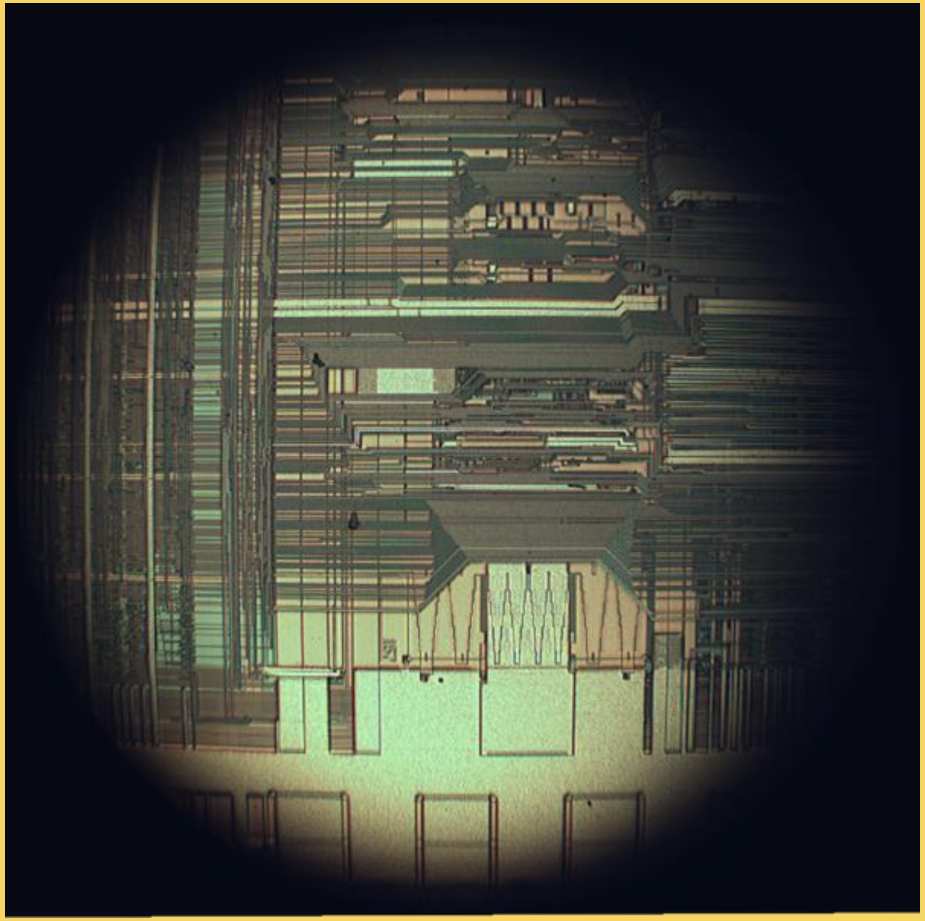


https://en.wikipedia.org/wiki/Integrated_circuit

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

Quiz 5 due Sunday, 10am

Class 11: Non-universal, Circuit size class

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

Plan

Midterm 1 review

Non-Universal Gates

Circuit-Size Class

Textbook [TCS] Section 3 and 4

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

Circuits and universal

Midterm 1 Review:
Regular Expressions and DFA

Regular Expressions and DFA

DFA:
easier to think as graphs

RE: We use the notation frequently, eg,

$1^40 = 11110$

1^n = repeat 1 for n times

$(3120)^*$

Definition 6.2 (Deterministic Finite Automaton)

A deterministic finite automaton (DFA) with C states over $\{0,1\}$ is a pair $(T, \mathcal{S})$ with $T : [C] \times \{0,1\} \to [C]$ and $\mathcal{S} \subseteq [C]$. The finite function T is known as the *transition function* of the DFA. The set $\mathcal{S}$ is known as the set of *accepting states*.

Let $F : \{0,1\}^* \to \{0,1\}$ be a Boolean function with the infinite domain $\{0,1\}^*$. We say that $(T, \mathcal{S})$ *computes* a function $F : \{0,1\}^* \to \{0,1\}$ if for every $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, if we define $s_0 = 0$ and $s_{i+1} = T(s_i, x_i)$ for every $i \in [n]$, then

$$s_n \in \mathcal{S} \Leftrightarrow F(x) = 1$$

Definition 6.6 (Regular expression)

A *regular expression* e over an alphabet Σ is a string over $\Sigma \cup \{(,), |, *, \emptyset, ""\}$ that has one of the following forms:

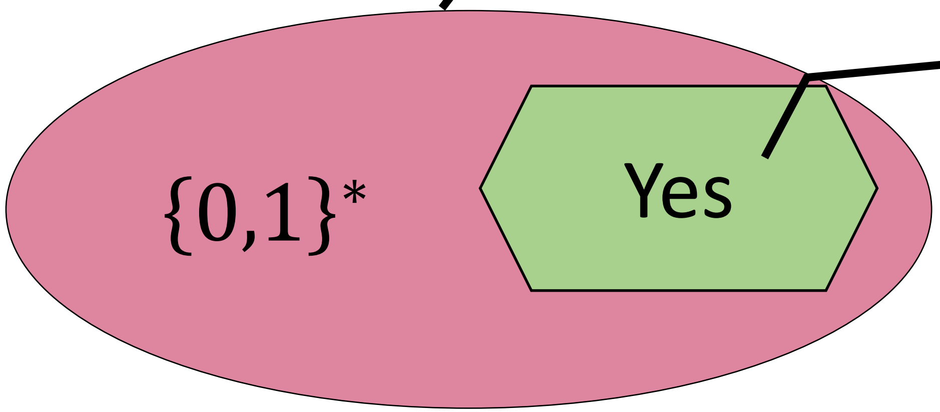
1. $e = \sigma$ where $\sigma \in \Sigma$
2. $e = (e'|e'')$ where e', e'' are regular expressions.
3. $e = (e')(e'')$ where e', e'' are regular expressions. (We often drop the parentheses when there is no danger of confusion and so write this as $e'\ e''$.)
4. $e = (e')^*$ where e' is a regular expression.

Finally we also allow the following "edge cases": $e = \emptyset$ and $e = ""$. These are the regular expressions corresponding to accepting no strings, and accepting only the empty string respectively.

Problems, Functions, Languages

Problem = *Language* = Binary Function

Any binary string $x \in \{0,1\}^*$ is an *instance*.



The subset of Yes instances:
The *Problem*, or the *Language*,
or the Boolean function
that outputs 1

Reg-Fun = DFA-Comp

Theorem 6.17 (DFA and regular expression equivalency)
Let $F : \{0,1\}^* \rightarrow \{0,1\}$. Then F is regular if and only if there exists a DFA $(T, \mathcal{S})$ that computes F.

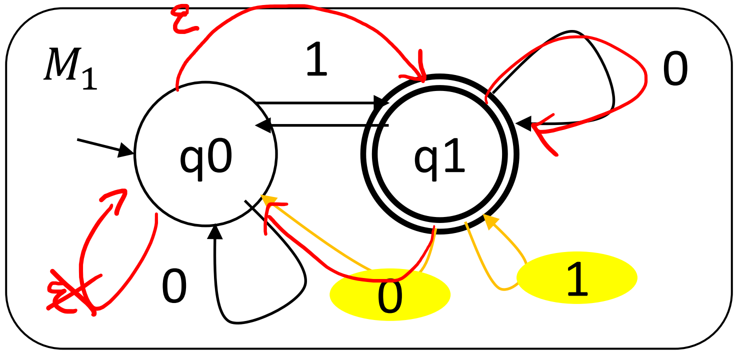
Can build DFA if and only if
Can build regular expression (with constructions).

Implication: Closure:

Theorem 6.19 (Closure of regular expressions)
Let $f : \{0,1\}^k \rightarrow \{0,1\}$ be any finite Boolean function, and let $F_0, \ldots, F_{k-1} : \{0,1\}^* \rightarrow \{0,1\}$ be regular functions. Then the function $G(x) = f(F_0(x), F_1(x), \ldots, F_{k-1}(x))$ is regular.

Non-deterministic Finite Automata

Similar to DFA,
but any num of out arrows



Accept: traverse to any accepting state

Skipped: ϵ transitions

A **Nondeterministic *Finite Automaton*** over alphabet {0,1} is a tuple (C, T, S) where:

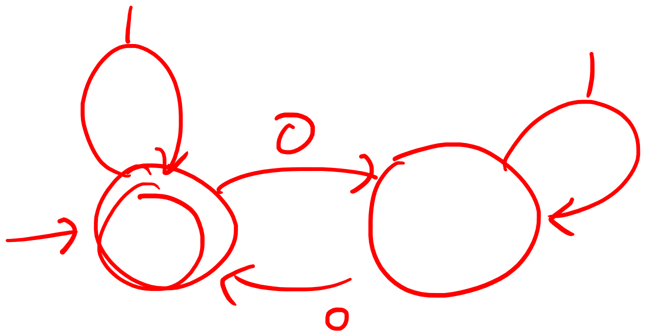
1. C --- the number of *states*
2. $T: [C] \times \{0,1\} \to \boldsymbol{pow([C])}$
   a transition function
3. $S \subseteq [C]$ --- the set of accept states

Execution:
The string $x = b_0 b_1 \ldots b_n$ is matched by the NFA $M = (C, T, S)$ iff there are states $s_0, s_1, s_2, \ldots, s_n \in Q$ such that $\boldsymbol{s_{i+1} \in T(s_i, b_i)}$ for all $i = 0, \ldots, n-1$ and $s_0 = 0$ and $s_n \in S$.

Is a Language / Function Regular?

(a) $L = \{w \in \{0,1\}^* \mid w \text{ contains an even number of 0s}\}$

Yes, but why?



Sometimes hard?

Is a Language / Function Regular?

(d) $L = \{w \in \{0,1\}^* \mid \sum_{i=0}^{k-1} w[i] \leq k/4$ where $k = |w|$ and $w[i]$ denotes the ith bit of w $\}$

$0/1$

No, but why?

$\forall\ n_0\ \exists\ w,\ |w| > n_0,\ w \in L$

$w = 1^{n_0} 0^{3n_0}$

$\forall\ x, y, z,\ xyz = w,\ |y| \geq 1,\ |xy| \leq n_0$

~~x y z~~, $x = 1^{l_0}$, $y = 1^{l_1}$, $l_0 + l_1 \leq n_0$, $l_1 \geq 1$

$\exists\ k = 2$

$x y^2 z = 1^{n_0} 1^{l_1} 0^{3n_0} \notin L$

Sometimes hard?

Lemma: e is regular expression, then
There is some number n_0 such that
for every $w \in \{0,1\}^*$ with $|w| > n_0$ and $\Phi_e(w) = 1$,
there exist strings $x, y, z \in \{0,1\}^*$, $w = xyz$,
satisfying the following conditions:

- $|y| \geq 1$
- $|xy| \leq n_0$
- $\Phi_e(x\, y^k\, z) = 1$ for every natural number k

Non-regular Language but Pumping

Let $L = \{\underline{10^k1^k \mid k \geq 1}\} \cup \{1^i0^j1^k \mid j,k \geq 1, i \neq 1\}$.

Ex: 1000111, 001, 11011, but not 1001 $\in L$

Pumping conditions:

Exist n_0= ~~1~~ 2

For all w = 10^k1^k $\qquad 110^j1^k \in L$

Exist x, y, z = "", "", "1", ... $\qquad$ x "", y "11", z ...

For all k = 0^k1^k, 10^k1^k, 110^k1^k ... $\qquad$ 0^j1^k, 110^j1^k, 1110^j1^k, ...

Regular?

Lemma: e is regular expression, then
There is some number n_0 such that
for every $w \in \{0,1\}^*$ with $|w| > n_0$ and $\Phi_e(w) = 1$,
there exist strings $x, y, z \in \{0,1\}^*$, $w = xyz$,
satisfying the following conditions:

- $|y| \geq 1$
- $|xy| \leq n_0$
- $\Phi_e(x\, y^k\, z) = 1$ for every natural number k

Non-regular Language but Pumping

Let $L = \underbrace{\{10^k1^k \mid k \geq 1\}}_{}$ …

Let $L = \overbrace{\{10^k1^k \mid k \geq 1\}}^{L_1} \cup \overbrace{\{1^i0^j1^k \mid j,k \geq 1, i \neq 1\}}^{L_2}$.

Proof of non-regular: Assume L is regular for contradiction.

L_2 regular by DFA

$L \setminus L_2 = L_1 = \text{AND}(F_L(x), \text{NOT}(F_{L_2}(x)))$ by closure, L_1 is regular

by Pumping Lemma, L_1 not regular

$\Rightarrow$ L not reg $\square$

Circuits

Recap: {AND, OR, NOT} is universal

{NAND} is 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 $NAND$.

$LOOKUP_n$ computes **any** $f: \{0,1\}^n \to \{0,1\}^m$, using {AND, OR, NOT}

Non-Universality

$\{AND, OR, NOT\}$

$\{XOR, NOT\}$

Proving Universality

$\{NOT\}$

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.

How can we prove some gate set G is universal?

NAND

Proving Non-Universality

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.

*How can we prove some gate set G is **not** universal?*

Non-Universality of $\{OR\}$

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.

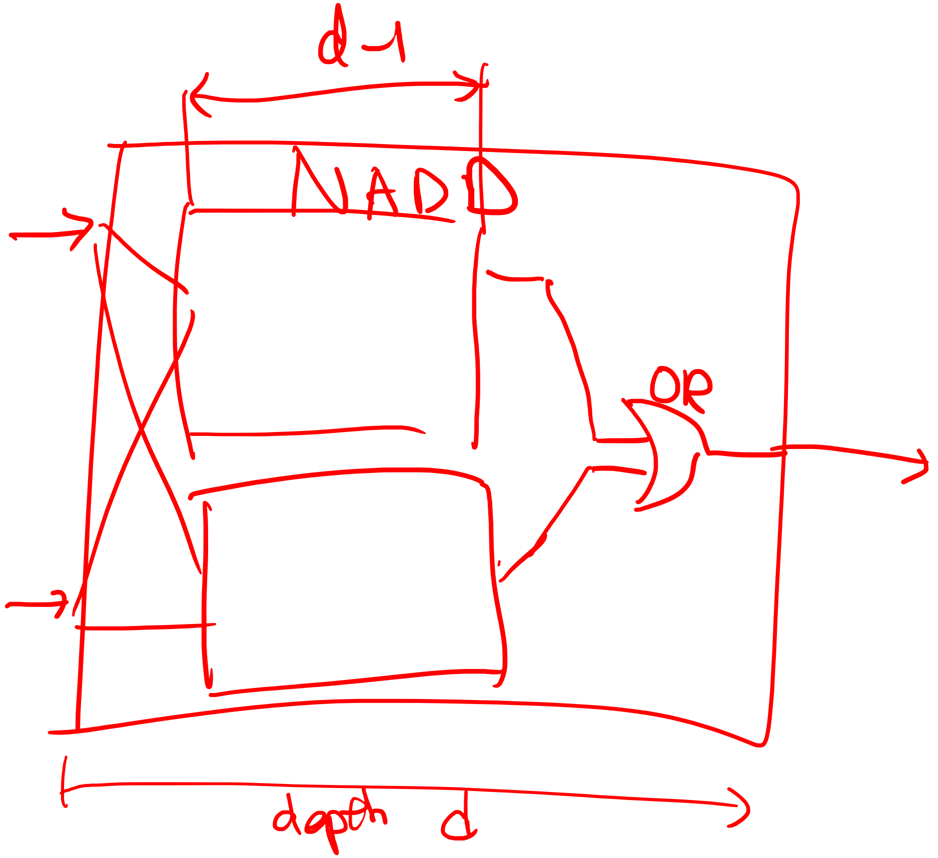
$OR: \{0,1\}^2 \to \{0,1\}$

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

$P(d) :=$ depth d OR-ckt is $\boxed{\text{Monotone}}$

$C(x) \geq C(x')$

$x' \geq x$ in ~~any~~ all bits



Circuit Complexity

How many gates?

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)
```

Recursion:

S(k) $\leq$ 2 · S(k-1) + c’

S(1) $\leq$ c’

Exists c, for all k, S(k) = c · 2^k

How many gates?

How many gates does this construction take?

You can computer any function $f: \{0,1\}^n \to \{0,1\}^m$ with a NAND circuit using no more than $c \cdot m \cdot 2^n$ gates

Note: this can be improved to $c \cdot m \cdot \frac{2^n}{n}$ (theorem 4.16 in TCS)

Where do we go from here?

Is this approach practical?

How many gates need to, say, add or multiply?

$4 \cdot n$ n-bit

$4n^2$, $O(n^{\log_2 3})$

Are there functions that *need* $\Omega(2^n)$ gates? (stay tuned..)

Categorizing Functions by Circuit Size

Fact 1: No functions (n-bit to m-bit) require more than $O(m\,2^n)$ gates

Fact 2: Some functions seem to require much less

So, let's categorize functions by **the # of gates** they need

Circuit Size

Complexity

Given circuit size s, which functions can be computed?

SIZE

$= \{$ ~~circui~~

$SIZE(s)$ The set of all **functions** that can be implemented by a circuit of at most s NAND gates

$SIZE(s) = \{ f : \{0,1\}^n \to \{0,1\}^n \mid \exists$ ckt (size s s.t. $C(x) \equiv f(x)$ $\}$

$SIZE(1000 \cdot m \cdot 2^n)$ Contains all functions $f : \{0,1\}^n \to \{0,1\}^m$

TCS also uses: $SIZE_{n,m}(s)$

The set of all n-input, m-output functions that can be implemented with at most s NAND gates

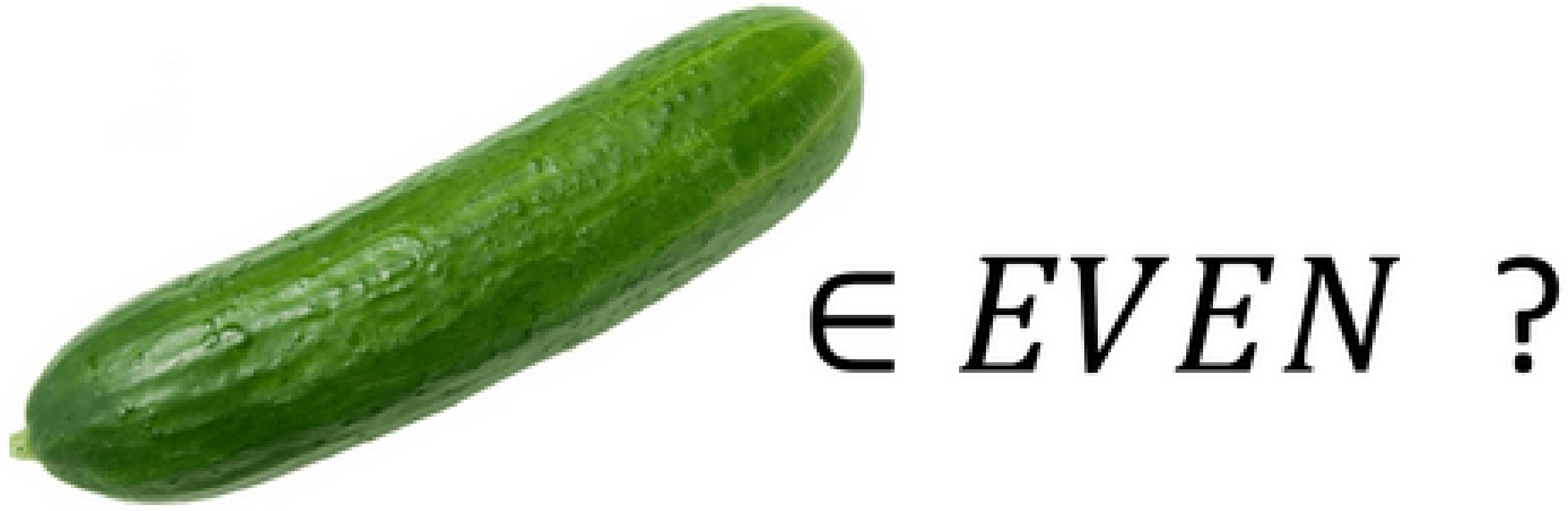
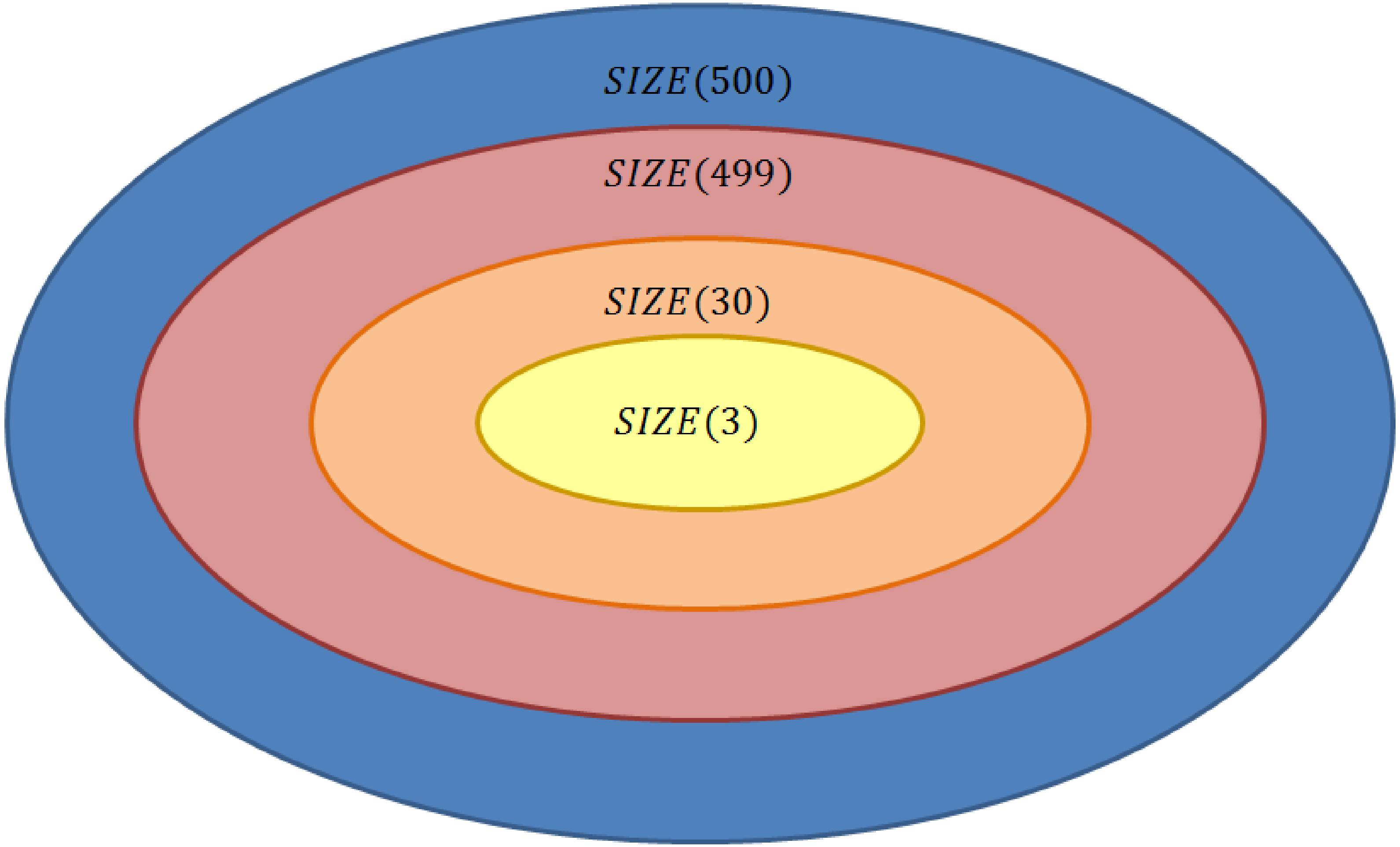


Figure 4.13: A "category error" is a question such as "is a cucumber even or odd?" which does not even make sense. In this book one type of category error you should watch out for is confusing functions and programs (i.e., confusing specifications and implementations). If C is a circuit or program, then asking if $C \in SIZE_{n,1}(s)$ is a category error, since $SIZE_{n,1}(s)$ is a set of functions and not programs or circuits.

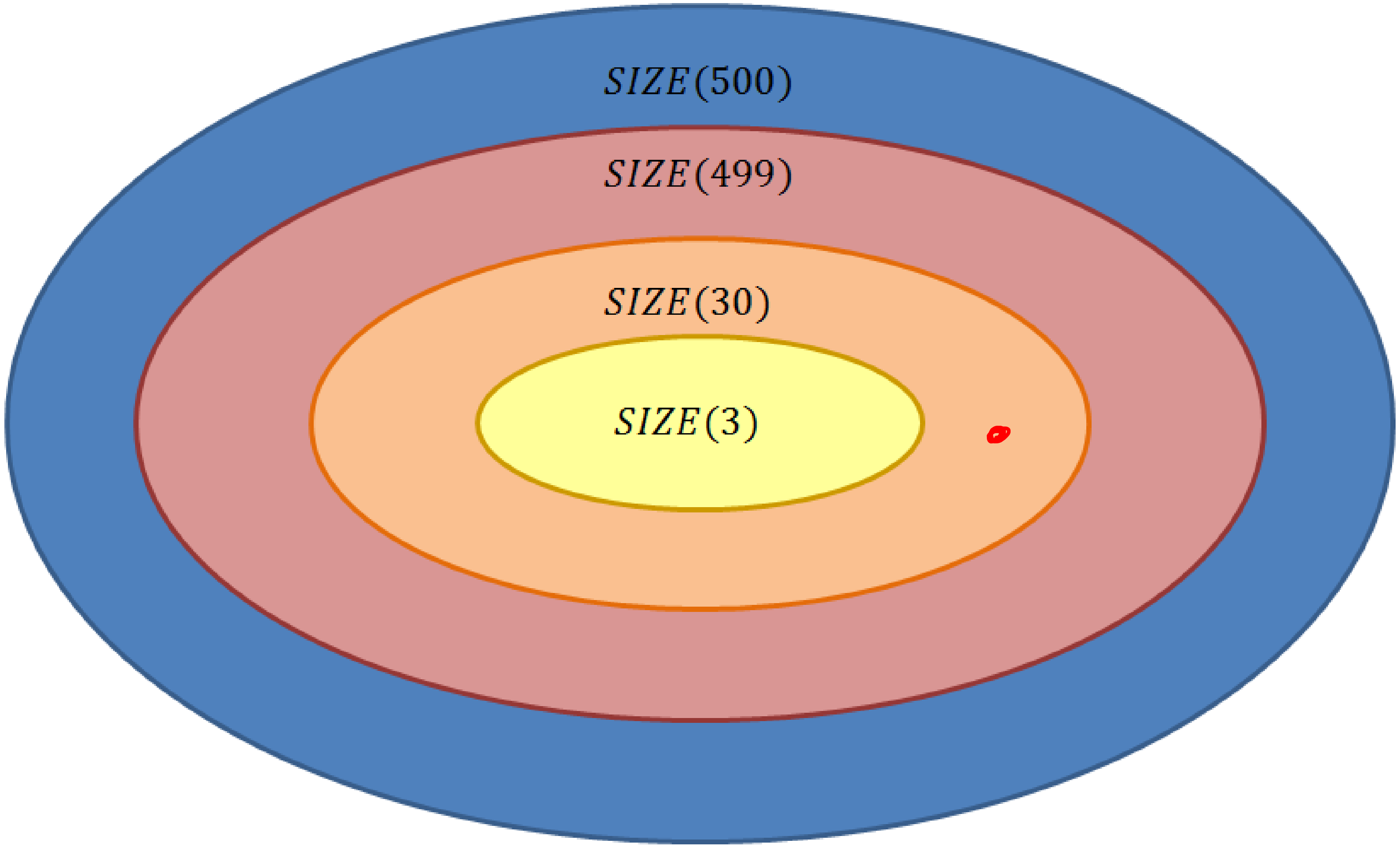
Two Different “Roles”

Function: What we want. Examples:
- Problem (we want to solve)
- Specification (of any module, or what customers want)

Circuit: How do we achieve it. Similar concepts:
- Program (implemented in our favorite PL)
- Implementation (to satisfy the need of customers)



If $x \leq y$, then $SIZE(x) \subseteq SIZE(y)$

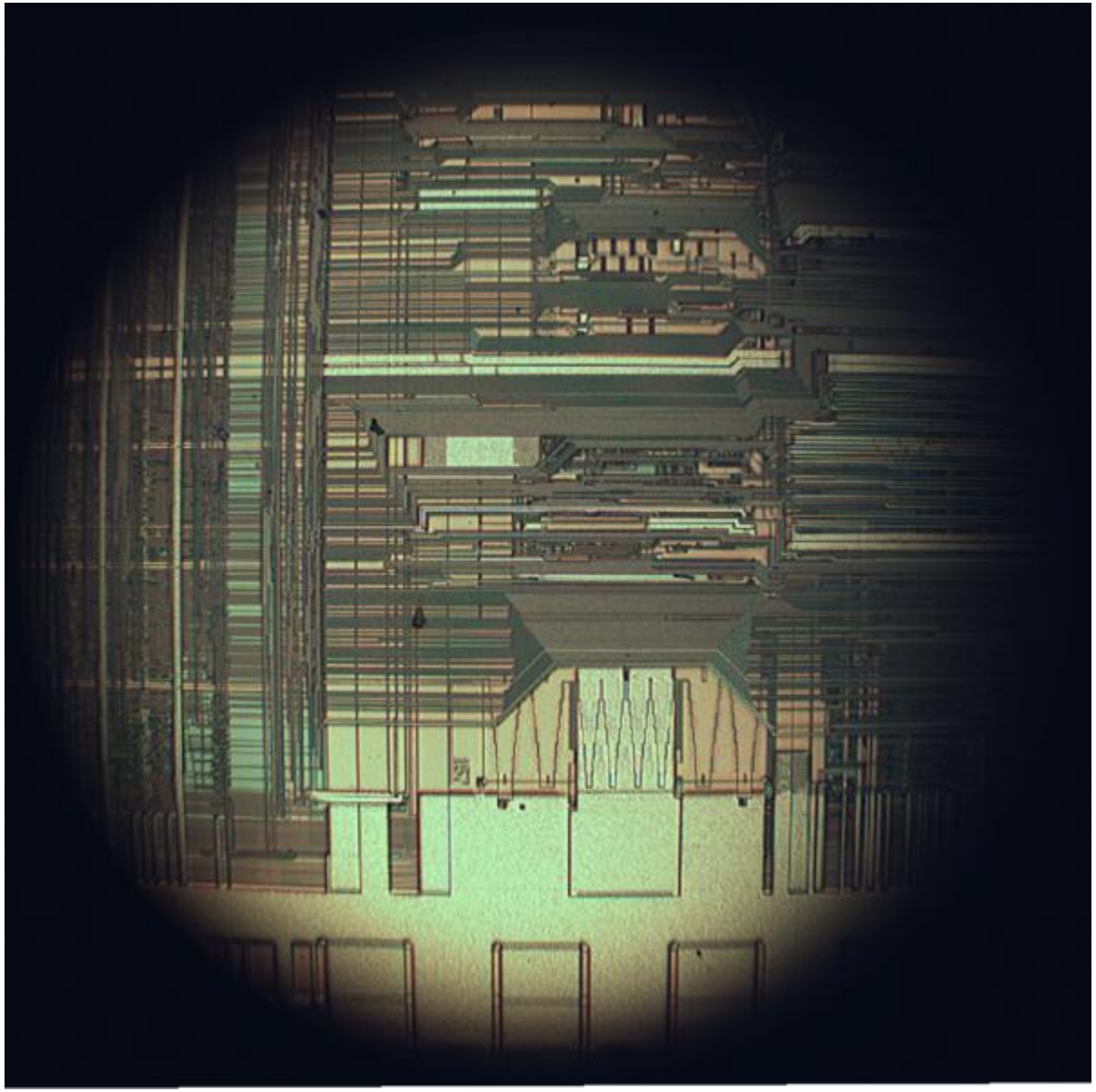


If $x \leq y$, then $SIZE(x) \subseteq SIZE(y)$

But is the inclusion **strict**? Again, stay tuned for that..

Size is the main goal!

“The Intel **486**, officially named **i486** and also known as **80486**, is a microprocessor introduced in 1989.”
Wikipedia: i486



https://en.wikipedia.org/wiki/Integrated_circuit

Plan

Circuit size hierarchy

[TCS] Textbook, Section 4 to 5

- 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