CS 121: Lecture 5
Completeness: Computing every (finite) function

Adam Hesterberg

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

Administrative

Outline

- Every function $f: \{0,1\}^n \to \{0,1\}^m$ can be computed by basic operations.
  - NAND-universality is really universal!
- Not every function $f: \{0,1\}^n \to \{0,1\}^m$ can be computed by, e.g., {NOT}
  - Not every gate is universal.
- Every function f can be computed by circuit with $O(nm2^n)$ gates. (complexity upper bound)
  - Aside: Syntactic Sugar
- Tomorrow: Some (most!) functions require $\Omega\left(\frac{2^n}{n}\right)$ gates. (Complexity lower bound. Limitations of circuits!)

Universality

AON-CIRC program / NAND circuit / NAND-CIRC program / NOR circuit / etc…

Theorem (4.12): $\forall f: \{0,1\}^n \to \{0,1\}^m$ there is a Boolean circuit C computing f.

- Suffices to consider functions $f: \{0,1\}^n \to \{0,1\}$

- Arbitrary functions have truth tables. Example:

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

Universality: Proof

Let δ_{001}: $\{0,1\}^3 \to \{0,1\}$ be defined as $\delta_{001}(x) = \begin{cases} 1 \; if \; x = 001 \\ 0 \; otherwise \end{cases}$

Q: Give Boolean circuit to compute δ_{001}.

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

Q: Give Boolean circuit to compute f

Non-universality: Why NAND?

- Is $\{NOT\} = \{NAND\}$?
- Is $\{EVEN_3\} = \{NAND\}$?
  - $EVEN_3: \{0,1\}^3 \to \{0,1\}$ is 1 iff an even number of inputs are 1

$EVEN_3 - CIRC$ example straightline program:

X[0], X[1], X[2] inputs

T ← $EVEN_3(X[0], X[1], X[2])$

U ← $EVEN_3(T, X[1], X[1])$

…

Y = $AND(X[0], X[1])$?

Exercise 1: Universality

1. Is $\{EVEN_3, NOT\}$ universal?

2. Is $\{EVEN_3, AND\}$ universal?

3. Is $\{ODD_3, AND\}$ universal? (Hint: instead of self-duality, prove a different invariant. What if all inputs are 0?)

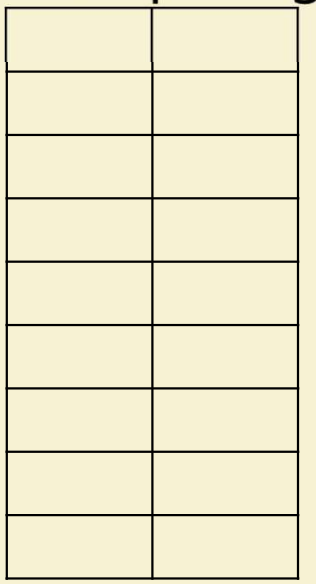
Universality: Size

Theorem (4.12): $\forall f: \{0,1\}^n \to \{0,1\}^m$ there is a Boolean circuit C computing f. *Moreover,* $|C| \leq O(n \cdot 2^n \cdot m)$

Proof: Let $f_i: \{0,1\}^n \to \{0,1\}$ be i^{th} bit of f ($f_i(x) = f(x)_i$)

Enough to prove Thm for $m = 1$!

Computing $f_0, \ldots, f_{m-1} \Rightarrow$ Computing f

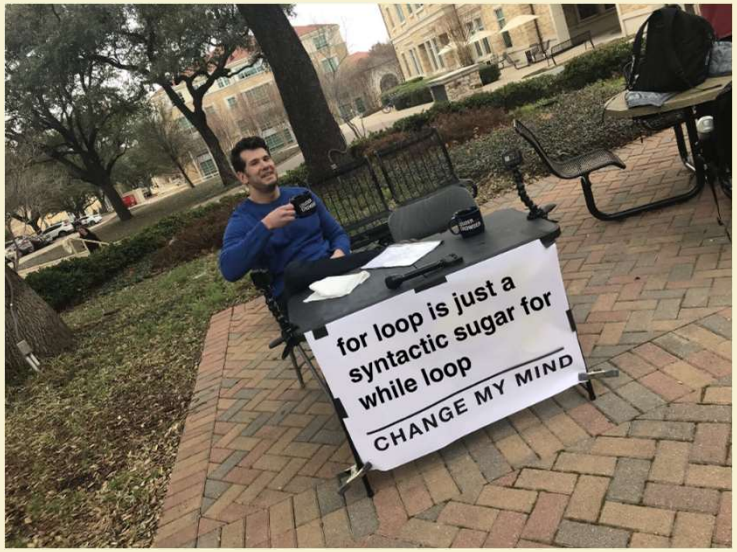


$$f(x) = \delta_{0^n}(x) \vee \delta_{0^{n-2}10}(x) \vee \quad \ldots \quad \vee \delta_{1^n}(x)$$

At most 2^n copies of δ_{x_i}, each computable by circuit of $n-1$ ANDs and $\leq n$ NOTs

$\Rightarrow$ Size $\leq O(n \cdot 2^n)$ ■

"Syntactic Sugar"

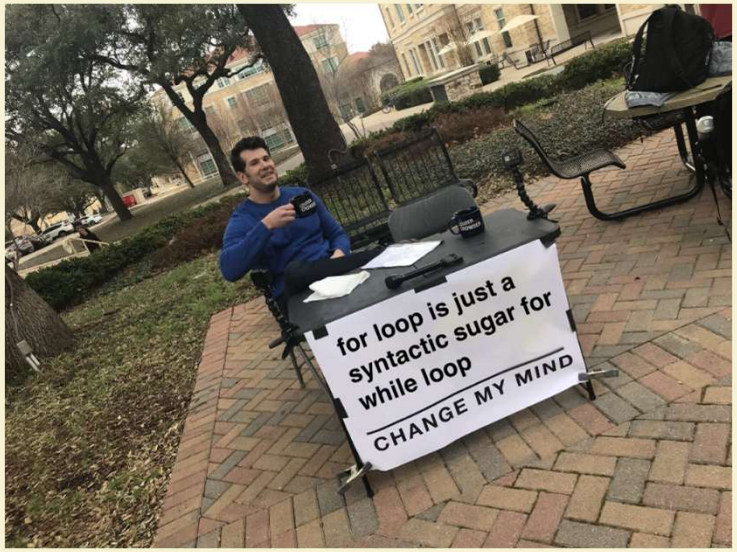


Take programming language "P"
and make it into "P++" by:

- Adding extra features to P++ on top of P
- Write a "transpiler" that takes P++ program and maps it to a P program that has equivalent functionality.

Example 1: C++ was initially developed by Bjarne Stroustrup who wrote the CFront compiler to compile C++ programs into C programs.

"Syntactic Sugar"

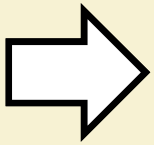


Take programming language "P"
and make it into "P++" by:

- Adding extra features to P++ on top of P
- Write a "transpiler" that takes P++ program and maps it to a P program that has equivalent functionality.

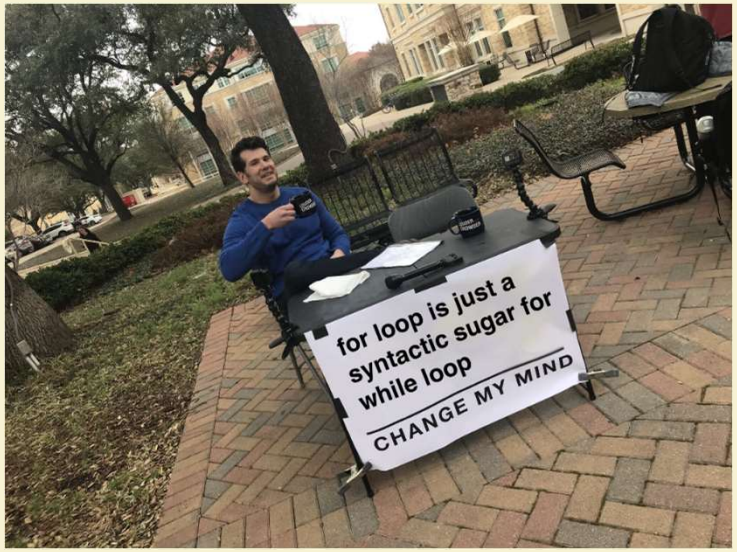
Example 2: `for` is syntactic sugar for `while`. In *C*:

```
for (init ; condition ; iterate)
    do_something
```



```
init;
while (condition) {
    do_something;
    iterate;
}
```

"Syntactic Sugar"

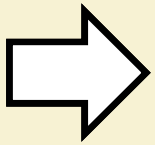


Take programming language "P"
and make it into "P++" by:

- Adding extra features to P++ on top of P
- Write a "transpiler" that takes P++ program and maps it to a P program that has equivalent functionality.

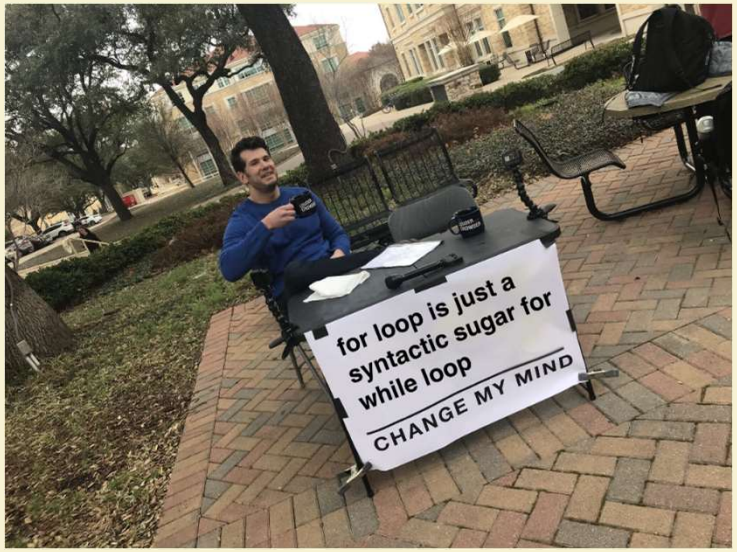
Example 2: `for` is syntactic sugar for `while`. In *Python*:

```
for item in sequence:
    do_something
```



```
itr = iter(sequence)
try:
    while True:
        item = itr.next()
        do_something;
except StopIteration: pass
```

"Syntactic Sugar"



Take programming language "P"
and make it into "P++" by:

- Adding extra features to P++ on top of P
- Write a "transpiler" that takes P++ program and maps it to a P program that has equivalent functionality.

Example 3: Define NAND-CIRC++ to include:

- If statements (If x[0], then x[1], else x[2])
- User-defined procedures
- Variables with non Boolean values (e.g. [256])
- Arrays…

Example Corollary: For every n, there is a circuit of $O(n^{1.6})$ gates to compute the map $a, b \mapsto a \cdot b$ where a, b are n bit numbers.

Universality: Size (Circuit Upper Bounds)

AON-CIRC program / NAND circuit / NAND-CIRC program / NOR circuit / etc…

Theorem (4.12): $\forall f: \{0,1\}^n \to \{0,1\}^m$ there is a Boolean circuit C computing f.

Moreover: $|C| := size(C) \leq \cancel{O(n \cdot 2^n \cdot m)}$ $\cancel{O(2^n \cdot m)}$ $O(2^n \cdot m/n)$ (Thm 4.14)

Exercise 2: Circuit Upper bounds

Theorem: For every f: $\{0,1\}^n \to \{0,1\}^m$ there is a Boolean Circuit computing f with $|C| = O(2^{2^n} 2^n)$ (or $O\left(2^{2^n} 2^n + m\right)$ to specify outputs).

1. How big must m be, in terms of n, for this to be better than the bound of $O(2^n m)$?

2. Prove the theorem. (Hint: How many functions $\{0,1\}^n \to \{0,1\}^1$ exist?)

Q: What's the size of $\{f \mid f: \{0,1\}^3 \rightarrow \{0,1\}\}$?

A: $2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 2^8$

Q: What's the size of $\{f \mid f: \{0,1\}^n \rightarrow \{0,1\}\}$?

A: 2^{2^n}

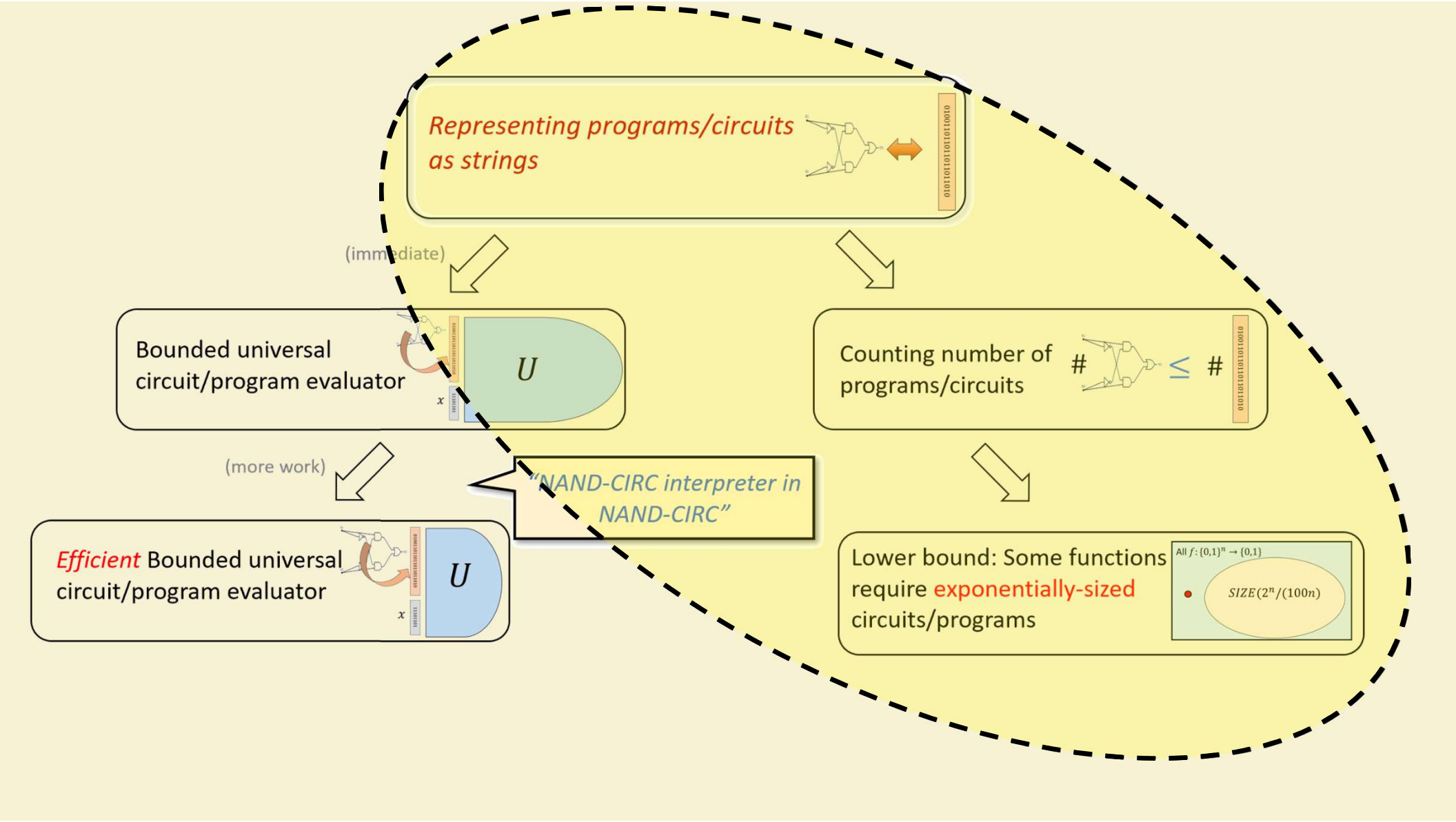
x	$f(x)$
000	y_0
001	y_1
010	y_2
011	y_3
100	y_4
101	y_5
110	y_6
111	y_7

Non-Universality: Size (Circuit Lower Bounds)

AON-CIRC program / NAND circuit / NAND-CIRC program / NOR circuit / etc…

Theorem II: Some functions $f: \{0,1\}^n \to \{0,1\}$ cannot be computed by circuits of size $o(2^n/n)$.

Proof: Recall that if ∃ onto map $A \to B$ then $|A| \geq |B|$


Representing programs/circuits as strings
(immediate)
Bounded universal circuit/program evaluator
U
x
Counting number of programs/circuits
#
≤
#
(more work)
"NAND-CIRC interpreter in NAND-CIRC"
Efficient Bounded universal circuit/program evaluator
U
x
Lower bound: Some functions require exponentially-sized circuits/programs
All $f:\{0,1\}^n \to \{0,1\}$
$SIZE(2^n/(100n))$