CS 121: Introduction to Theoretical Computer Science	September 17 - 23

Section 2: Finite Computation and Universality

Profs. Madhu Sudan and Adam Hesterberg *Ife Omidiran*

1 Practice Problems

1. Show whether the following gate sets are universal:
   (a) AND and NOT (Hint: Use the definition of universality.)
   (b) AND and OR (Hint: Think about the properties of functions that can be computed using AND/OR. What happens if all of the inputs are 1?)

   Solution:

   (a) NAND(a, b) can be computed by applying the operations NOT(AND(a, b)). The definition of universality of a gate set is that it can compute NAND, thus AND and NOT are universal.

   (b) AND and OR are not universal because they can't compute the constant 0 function. This can be proven by induction on k, the size of the circuit. The base case $k = 1$ is simple: AND(x_0, x_1) = 1 and OR(x_0, x_1) = 1 when $x_0 = x_1 = 1$, but the constant 0 function must output 0 for all inputs. For $k > 1$, consider a circuit that outputs the AND or OR of a previous line or input. Once again, if both inputs are 1, then each of the previous lines must evaluate to 1, so the circuit can only output 1 in this case. However, a NAND circuit can compute the constant 0 function: NAND(a, a) = 0. Thus, AND and OR do not form a universal gate set.

2. Let `IF-CIRC` be the programming language where we have the following operations: `foo = 0`, `foo = 1`, `foo = IF(cond, yes, no)`; that is, we can use the constants 0 and 1, and the $IF : \{0,1\}^3 \to \{0,1\}$ function such that $IF(a,b,c)$ equals b if $a = 1$ and equals c if $a = 0$. Show that AON-CIRC is as powerful as IF-CIRC, and vice versa.[1]

   Solution:
   We show that AON-CIRC is **as powerful** as IF-CIRC in two parts.
   Claim: AON-CIRC is at least as powerful as IF-CIRC.

   Proof: As IF-CIRC is a sequence of statements of the form $d_i = IF(a_i, b_i, c_i)$. To show that AON-CIRC is as powerful as IF-CIRC, we just need to show that we can compute replace every such IF statement with a sequence of AND, OR and NOT statements. Below we show a replacement for the statement $d = \mathrm{IF}(a, b, c)$:

$$\begin{aligned}\text{temp\_0} &= \mathrm{NOT}(a)\\ \text{temp\_1} &= \mathrm{AND}(a, b)\\ \text{temp\_2} &= \mathrm{AND}(\text{temp\_0}, c)\\ d &= \mathrm{OR}(\text{temp\_1}, \text{temp\_2})\end{aligned}$$

[1] Hint: The $LOOKUP_1$ function is closely related to IF.

Once we have this, we can replace every line of an IF-CIRC with the sequence above (taking care to replace the variable names a, b, c, d with the ones in the IF statement).

Claim: IF-CIRC is at least as powerful as AON-CIRC.

Proof: As in the proof above we need to show that each of the lines $b = \text{NOT}(a)$, $c = \text{OR}(a, b)$ and $c = \text{AND}(a, b)$ can be replaced by a sequence of IF statements. The following sentence is equivalent to $b = \text{NOT}(a)$:

$$b = \text{IF}(a, 0, 1).$$

For $c = \text{OR}(a, b)$ we have:

$$c = \text{IF}(a, 1, b).$$

Finallw for $c = \text{AND}(a, b)$ we can use:

$$c = \text{IF}(a, b, 0) \quad .$$

With the above "macros" we can replace every line of an AON-CIRC program with a line of IF-CIRC program.

3. In the proof presented for the universality of NAND, we mentioned, but didn't prove, that the lookup function is in $SIZE(4 \cdot 2^k)$. Prove that $LOOKUP_k$ can indeed be computed using a circuit of at most $4 \cdot 2^k - 1$ gates.

Solution:
We complete the proof using strong induction.

Base cases: We showed in section that $LOOKUP_1$ can be computed using a 4-line AON-CIRC program. An equivalent NAND-CIRC program can be described as follows:

```
def LOOKUP_1(X[0], X[1], X[2]):
    temp_0 = NAND(X[2], X[2])
    temp_1 = NAND(X[0], temp_0)
    temp_2 = NAND(X[1], X[2])
      Y[0] = NAND(temp_1, temp_2)
```

The program is 4 lines, so the base case holds for $l = 1$.
We have the following program for $LOOKUP_2$:

```
def LOOKUP_2(X[0], X[1], X[2], X[3], i[0], i[1]):
    temp_0 = LOOKUP_1(X[2], X[3], i[1])
    temp_1 = LOOKUP_1(X[0], X[1], i[1])
      Y[0] = LOOKUP_1(temp_0, temp_1, i[0])
```

The program uses 3 invocations of LOOKUP_1, each of which can be replaced by 4 NAND lines, resulting in a program that is fewer than $4 \cdot 2^2 - 1$ lines. The base case holds for $l = 2$.

Inductive step: Suppose that we have a program of length k that computes $LOOKUP_n$. We assume as our inductive hypothesis that the length of the program computing $LOOKUP_m$ for all $m < n$ is at most $4 \cdot 2^m - 1$ lines. We need to show that $k \le 4 \cdot 2^n - 1$.
Generalizing the NAND-CIRC program computing $LOOKUP_2$ to the program computing $LOOKUP_n$, the program will require 2 invocations of the LOOKUP_(n-1) procedure, and one invocation of the LOOKUP_1 procedure. By our inductive hypothesis, the LOOKUP_(n-1) procedure is at most $4 \cdot 2^{n-1}$ lines, and we know that the LOOKUP_1 procedure is exactly 4 lines. Thus $k \le 2 \cdot 4 \left(2^{n-1} - 1\right) + 4$, which simplifies to $k \le 4 \left(2^n - 1\right)$.