Exercise 2: SOLUTIONS

Collaborators and Resources: SOLUTION SET, for use by CS3102 at UVA students in Spring 2020 only. Anyone reading this outside of that course for that semester is in violation of the honor code.

Exercise 2-2: Maximum number of Inputs

The *depth* of a circuit is the length of the longest path (in the number of gates) from the an input to an output in the circuit. Prove using induction that the maximum number of inputs for a Boolean circuit (as defined by Definition 3.5 in the book) that produces one output that depends on all of its inputs with depth d is 2^d for all $d \geq 0$. (Note: there are ways to prove this without using induction, but the purpose of this problem is to provide induction practice, so only solutions that are well constructed proofs using the induction principle will be worth full credit.

Base case:

for $d = 1$, we have a depth-1 circuit. In this case, this means a single gate, which can have at most $2 = 2^1$ inputs.

Inductive Hypothesis:

Assume that the maximum number of inputs for a depth k circuit (whose output depends on all inputs) is 2^k.

Inductive Step:

Consider a circuit of depth $k + 1$. We can build a circuit of depth $k + 1$ from a circuit of depth k by "replacing" each of the 2^k inputs of a circuit of depth k (which is maximum by the inductive hypothesis). The doubles the number of inputs from the previous step (as each input becomes a gate, which has 2 inputs), and so now this depth $k + 1$ circuit has $2 \cdot 2^k = 2^{k+1}$ inputs.

Exercise 2-3: Compare n bit numbers (Exercise 3.2 in TCS book)

Prove that there exists a constant c such that for every n there is a Boolean circuit (using only *AND*, *OR*, and *NOT* gates) C of at most $c \cdot n$ gates that computes the function $CMP_{2n} : \{0,1\}^{2n} \rightarrow \{0,1\}$ such that $CMP_{2n}(a_0 \cdots a_{n-1} b_0 \cdots b_{n-1}) = 1$ if and only if the number represented by $a_0 \cdots a_{n-1}$ is larger than the number represented by $b_0 \cdots b_{n-1}$.

I will prove this using a proof by induction.

Base Case

We can compute CMP_2 using a constant number of gates. In this case, comparing two 1-bit strings can be implemented by the following program:

```
def CMP2(a, b):
    diff = XOR(a,b)
    return IF(diff, a, 0)
```

$XOR(a,b)$ will be 1 when $a \neq b$. The IF function will therefore return 0 whenever $a = b$. When $a \neq b$ we want to return the value of a (as when a is greater it will be 1, and when it is lesser it will be 0).

XOR requires 5 gates, and IF requires 4, so we can say that the CMP_2 function requires no more than $10 \cdot 1$ gates, so we'll call our constant $c = 10$

Inductive Hypothesis

Assume that we can compute CMP_{2k} using at most $10k$ gates.

Inductive Step

we can compute $CMP_{2(k+1)}$ using CMP_{2k} as follows:

```
def CMP2k+1(a0, a1, ..., ak, b0, b1, ..., bk):
    diff0 = XOR(a0, b0)
    cmpRest = CMP2k(a1, ..., ak, b1, ..., bk)
    return IF(diff0, a, cmpRest)
```

In this case, whenever the most significant bits of a and b are different, the return value should be the value of $a0$ (as when a is greater it will be 1, and when it is lesser it will be 0). If $a0 = b0$ then the return value will be the result of comparing the rest of the bits.

In this case, the number of gates used for CMP_{2k+1} is the number required by the XOR, the IF, and CPM_{2k}. The XOR requires 5 gates, the IF requires 4 gates, and by our inductive hypothesis CMP_{2k} requires no more than $10k$ gates. This means that CMP_{2k+1} requires no more than $5+9+10k < 10(k+1)$ gates.

Exercise 2-4: *NOR* is universal (Exercise 3.7 in TCS book)

Let $NOR : \{0,1\}^2 \rightarrow \{0,1\}$ defined as $NOR(a,b) = NOT(OR(a,b))$. Prove that $\{NOR\}$ is a universal set of gates (i.e., anything that can be computed using *AND,OR,NOT* can also be computed using just *NOR*).

proof We can show that NOR is universal by showing how to construct a universal set of gates from NOR.

Since we have that $NOR(a,b) = NOT(OR(a,b))$, if we can construct $NOT(a)$ by: $NOR(a,a) = NOT(OR(a,a)) = NOT(a)$

Also, if we apply deMorgan's law we have that $NOR(a,b) = NOT(OR(a,b)) = AND(NOT(a), NOT(b))$. Since showed above that we could implement $NOT(a)$, this implies we can also implement *AND*.

With *NOT* and *AND* we can build *NAND*, which is universal.

Exercise 2-5: XOR is not universal (based on Exercise 3.5 in TCS book)

Prove that the set $\{XOR, 0, 1\}$ is not universal. (You can use any strategy you want to prove this; see the book for one hint of a possible strategy, but we think you may be able to find easier ways to prove this, and it is not necessary to follow the strategy given in the book.

To demonstrate this, we will show that there is at least one function that cannot be computed by $\{XOR, 0, 1\}$. Any 2- input function that we can compute with $\{XOR, 0, 1\}$ must be the result of performing an XOR on 0, 1, an input bit a, an input bit b, or the result of some other chain of *XOR* operations. In this case can can compute each of the following 2-input functions **(with justification)**:

– 0 **(given)**

– 1 **(given)**

– $XOR(a, b)$ **(given)**

– $a = XOR(a, 0)$

– $b = XOR(b, 0)$

– $NOT(a) = XOR(a, 1)$

– $NOT(b) = XOR(b, 1)$

– $XOR(NOT(a), b)$

– $XOR(a, NOT(b))$

No other functions besides those listed can be computed. Note that all combinations of an *XOR* with all pairs from $a, b, 0, 1$ is already enumerated. If we did an $XOR(a, b)$ on any other pair of functions, that would only get another function on the list:

– $XOR(NOT(a), NOT(a)) = XOR(a, b)$

– for any x, y we have that $XOR(x, XOR(x, y)) = y$

Since this list does not include $AND(a, b)$ (among other functions), *XOR* is not universal.