Ethan Wong
10/9/21
305319001

CS 181 HW1

1. To solve this, we must prove both forwards and backwards cases
"⟹" Forwards
• If there exists a one-to-one mapping from S to T, then there exists an onto mapping from T to S.
One-to-One means that every input has exactly one output. A function is something where an element in the domain (x) matches to a single element in the codomain (y). Using these definitions, we know that if the mapping from S to T is injective, then size(s) ≤ size(T).

Using the Pigeon Hole Principle, we can analyze the above statement.
• size(s) = size(T)
If this is true, then there exists a function $Y: T \to S$ such that G is onto, given that G is one-to-one.
• size(s) < size(T)
If this is true, then there exists a function $Y: T \to S$ such that G is onto, because all elements in T must be matched to an element in S.
These observations show that if there exists a one-to-one mapping from S to T then there also exists an onto mapping from T to S.

"⟸" Backwards
• If there exists an onto mapping from T to S, then there exists a one-to-one mapping from S to T.
Onto means every element in the codomain has at least one element in the codomain that points to it. Knowing that, then we are able to find an element $t \in T$ such that an element $s \in S$ maps to it.
We also know that T to S is onto, which indicates that size(s) ≤ size(T). If this is the case, then the pigeonhole principle isn't even need. In conclusion,

we see that if there exists a surjective mapping from T to S, then there exists an injective mapping from S to T.

Having proved both directions, we see that there is indeed a one-to-one mapping from S to T i.f.f. there is an onto mapping from T to S.

2. String representation of directed graphs with vertex set [n] and degree ≤ 10 that uses at most $1000 n \log n$ bits

- Say that we have the following graph G:

$$A \longrightarrow B \longrightarrow C \longrightarrow D \longrightarrow E$$

The adjacency list for G would be

$A \to B$

$B \to C$

$C \to D$

$D \to E$

$E \to$

To represent these elements you would need at least 3 bits. This is because $n = 5$ and you need 3 bits to represent the number 5.

To represent the graph properly as an adjacency list, we would need these symbols:

$\{A, B, C, D, E, ,, [,], \{, \}\}$ (10 total symbols)

The proper adjacency list would be as follows:

$\{A[B], B[C], C[D], D[E], E\}$

The binary representations for each character is as follows:

{ : 0000

} : 0001

[: 0010

] : 0011

, : 0100

A : 0101

B : 0110

C : 0111

D : 1000

E : 1001

$\hookrightarrow \approx O(\log_2(n))$

If we are to have at most deg (10) for each vertex, then the longest line would look like this:

[A [B, C, D, E, F, G, H, I, J, K]]

There are 24 elements (characters) in this line. There will also be n vertices, or n total lines.

When combining all this together, we see that an encoding for an n-vertex graph with each vertex having $\leq$ deg (10) could be at most $24 n \log n$ bits. This proves that there exists a one-to-one function $E : G_n \to \{0, 1\}^{\lceil 1000 n \log n \rceil}$

OR  NOT 

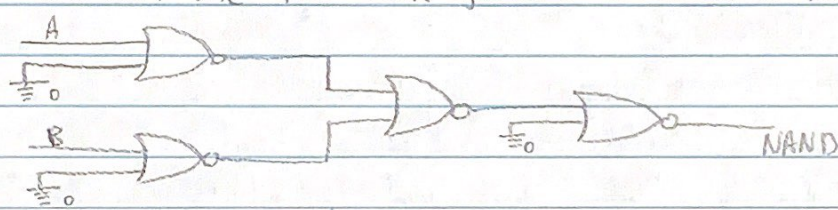
	NAND	
0	0	1
0	1	1
1	0	1
1	1	0

(3.3) pg 155 3. Show that {OR, NOT} can be used to create NAND

- The NOR gate can be created by using an OR gate that is immediately followed by a NOT gate. The truth table for NOR is as follows.

A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

- We can use the NOR gates to construct a NAND gate:



A	B	NAND
0	0	1
0	1	1
1	0	1
1	1	0

- Based on the circuit diagram and truth table above, we see how NOR gates (which are constructed from OR, NOT gates) can be used to create a NAND gate.

4. Show that {AND, OR, 0, 1} is non-universal

Monotonic (increasing) means that if $x_1 > x_0$, then
$f(x_1) > f(x_0)$

First we will show that both OR, AND are monotonic.

AND

0	0	0	$\longrightarrow$				0	1	0
0	1	0	$\longrightarrow$	1	1	1	1	0	0
1	0	0	$\longrightarrow$	1	1	1	1	1	1
1	1	1							

AND is monotonic (increasing)

OR

0	0	0	$\longrightarrow$				0	1	1
0	1	1	$\longrightarrow$	1	1	1	1	0	1
1	0	1	$\longrightarrow$	1	1	1	1	1	1
1	1	1							

OR is monotonic (increasing)

NAND

0	0	1	$\longrightarrow$	0	1	1
0	1	1		1	0	1
1	0	1		1	1	0
1	1	0				

NAND is not monotonic (not increasing)

Two monotonic functions can not be combined to create a non monotonic function. For instance, take NAND. NAND = NOT(A) OR NOT(B). NAND can not be a monotonic function as a result of this.

Proof: Say we have 2 monotonic functions m, n.

- if $x_0 < x_1$, then $m(x_0) < m(x_1)$
  Say that $f(x_0) = y_0$ and $f(x_1) = y_1$
- Since $y_0 < y_1$, then $n(y_1) < n(y_2)$.
- This shows that if for the initial inputs $x_1 < x_2$, then $n(m(x_1)) < n(m(x_2))$

Because NAND is not a monotonic function, there is no way to string monotonic ANDs and ORs together to create it. More specifically, the ANDs and ORs are unable to create a NOT. The NOT is needed in both universal operators: NAND and OR. Since ANDs and ORs can not create these universal operators, we see that the set $\{AND, OR, 0, 1\}$ is not universal. There does exist a function that can't be computed by this set.