EC 380 Homework B

45
45

Name: Onkar V Apte
OSU ID: [redacted]

3.7

Exercise 3.7: (Halting oracle) Suppose a *black box* is made available to us which takes a non-negative integer x as input, and then outputs the value of $h(x)$, where $h(\cdot)$ is the halting function defined in Box 3.2 on page 130. This type of black box is sometimes known as an *oracle* for the halting problem. Suppose we have a regular Turing machine which is augmented by the power to call the oracle. One way of accomplishing this is to use a two-tape Turing machine, and add an extra program instruction to the Turing machine which results in the oracle being called, and the value of $h(x)$ being printed on the second tape, where x is the current contents of the second tape. It is clear that this model for computation is more powerful than the conventional Turing machine model, since it can be used to compute the halting function. Is the halting problem for this model of computation undecidable? That is, can a Turing machine aided by an oracle for the halting problem decide whether a program for the Turing machine with oracle will halt on a particular input?

Let this new model of turing machine, i.e one w/ oracle be newTM(x).

Now, consider another function T(x) such that:

```
T(x) :
    y = halt(x)
    if (!y == 0) :
            loop forever.
    else :
      halt.
```

Let the turing number for this machine be θ.

$\therefore$ $T(\theta)$ will only halt if halt $(\theta) = 0$, which is the condition for turing machine w/ turing number θ to loop / run forever i.e. not halt.

$\therefore$ It is a contradiction.

$\therefore$ newTM(x) will fail if turing number of the same machine is passed to the function.

$\therefore$ Turing machine w/ halting oracle can decide the halting problem for a particular i/p as long as i/p $\neq$ value of it's own machine.

$\therefore$ Halting problem for new model of TM is still undecidable.

3.9

Exercise 3.9: Prove that $f(n)$ is $O(g(n))$ if and only if $g(n)$ is $\Omega(f(n))$. Deduce that $f(n)$ is $\Theta(g(n))$ if and only if $g(n)$ is $\Theta(f(n))$.

by defination, we know that $f(n) = O(g(n))$ means:

for some $c > 0$ & constant n_0 & all $n > n_0$: $\underline{f(n) \leq c \cdot g(n)}$

$\therefore$ for $n > n_0$, rearranging the equation : $\underline{\frac{1}{c} f(n) \leq g(n)}$

$\therefore$ for all $n > n_0$, & constant $k = \frac{1}{c}$, we can write:

$$\underline{g(n) \geq k \cdot f(n)}$$

$\therefore$ for all $n > n_0$, $c > 0$, $k > 0$, above equation shows

$$\boxed{g(n) = \Omega(f(n))}$$

Now, we know that $f(n) = \Theta(g(n))$ means:

for some $c_1, c_2 > 0$ & $n > n_0$, $\underline{c_1 g(n) \leq f(n) \leq c_2 g(n)}$

Now, simillar to how we re-arranged above,

this implies : ① $g(n) \leq \frac{1}{c_1} f(n)$

② $g(n) \geq \frac{1}{c_2} f(n)$

for some n & $\frac{1}{c_1}, \frac{1}{c_2} > 0$.

$\therefore$ Say $k_1 = \frac{1}{c_1}$ & $k_2 = \frac{1}{c_2}$

$\therefore$ Combining ① & ② gives us: $k_2 f(n) \leq g(n) \leq k_1 f(n)$

$$\therefore \boxed{g(n) = \Theta(f(n))}$$

3.10

Exercise 3.10: Suppose $g(n)$ is a polynomial of degree k. Show that $g(n)$ is $O(n^l)$ for any $l \geq k$.

$\therefore$ we can say $g(n) = c_1 + c_2 n + c_3 n^2 + c_4 n^3 \dots + c_k n^k$

$g_k(n) = \sum_{i=0}^{k} c_i n^i$ —— (c_i are constants)

For $k=1$, $g_1(n) = c_0 + c_1 n$ which is $\leq n^2$

$\therefore g_1(n) = O(n^l)$ for $l \geq 1$

Lets assume $g_k(n) = O(n^l)$, $l \geq k$ is true.

Now, $g_{k+1}(n) = \sum_{i=0}^{k+1} c_i n^i = \sum_{i=0}^{k} c_i n^i + c_{k+1} n^{k+1}$

$= g_k(n) + c_{k+1} n^{k+1}$ — (1)

& according to our assumption, $g_k(n) = O(n^l)$.

$\therefore g_k(n) \leq a n^l$ for some constant a & $l \geq k$

$\therefore$ our equation (2) $\Rightarrow$ $g_{k+1}(n) \leq a n^l + c_{k+1} n^{k+1}$

$\leq a n^l$ —— (but $l \geq k+1$, not k)

$\therefore g_{k+1}(n) = O(n^l)$ For $l \geq k+1$.

Hence, our assumption was right.

Hence proved $\Rightarrow$

$\therefore$ $\boxed{g_k(n) = O(n^l), \quad l \geq k}$

3.11

Exercise 3.11: Show that $\log n$ is $O(n^k)$ for any $k > 0$.

$\because \quad \log(n) = O(n^k)$,

$\log(n) \leq c \cdot n^k$ where c is some constant and $k > 0$.

For $n = 1$, $\log(n) = \log(1)$
$= 0 \leq c \cdot n^k$ where $k > 0$, $n > n_0$

Now, assume that $\log n = O(n^k)$ for $k > 0$ is true.

$\therefore \quad \log(n) \leq c \cdot n^k$

Now, we know that $n+1 \leq 2n$ for $n \geq 0$.

$\therefore$ taking log on both sides,

$$\log(n+1) \leq \log(2n)$$
$$\log(n+1) \leq \log 2 + \log(n)$$
$$\log(n+1) \leq 1 + \underline{\log(n)} \quad \text{——— (i)}$$

From our assumption this is less than or equal to $(c \cdot n^k)$

$\therefore$ equation (i) becomes $\Rightarrow$ $\log(n+1) \leq 1 + c \cdot n^k$
$\log(n+1) \leq c\, n^k$
$\therefore \quad \log(n+1) = O(n^k)$ for $k > 0$

Hence, by inductive reasoning, $\boxed{\log(n) = O(n^k) \text{ for } k > 0}$

3-12

Exercise 3.12: ($n^{\log n}$ **is super-polynomial)** Show that n^k is $O(n^{\log n})$ for any k, but that $n^{\log n}$ is never $O(n^k)$.

Say : $f(n) = n^k$ → Now consider $\frac{f(n)}{g(n)} = \frac{n^k}{n^{\log(n)}}$

$g(n) = n^{\log(n)}$

$$= \frac{1}{n^{\log(n)} \cdot n^{-k}}$$

$$= \frac{1}{n^{\log(n) - k}}$$

Now, taking limit as $n \to \infty$ on both sides,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{1}{n^{\log(n) - k}}$$

as $n \to \infty$, $\log(n) \longrightarrow \infty$

$$\therefore n^{\log(n) - k} \longrightarrow \infty$$

$$\therefore \frac{1}{n^{\log(n) - k}} \longrightarrow 0$$

$\therefore \lim_{n\to\infty} \frac{n^k}{n^{\log(n)}} = 0$ $\therefore$ limit belongs to $[0, \infty]$.

$$\therefore n^k = O\left(n^{\log(n)}\right)$$

Now, consider $\frac{g(n)}{f(n)} = \frac{n^{\log(n)}}{n^k} = \frac{1}{n^k \cdot n^{-\log(n)}}$

$$= \frac{1}{n^{k - \log(n)}}$$

taking $\lim \to \infty$ on both sides,

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = \lim_{n \to \infty} \frac{1}{n^{k - \log(n)}}$$

as $n \longrightarrow \infty$, $\log(n) \longrightarrow \infty$

$\therefore$ $k - \log(n) \longrightarrow -\infty$

$\therefore$ $n^{k - \log(n)} \longrightarrow -\infty$.

$\therefore$ limit DOES NOT belong to $[0, \infty]$

$$\therefore \boxed{n^{\log(n)} = O(n^k) \text{ is ALWAYS FALSE.}}$$

3.13

Exercise 3.13: ($n^{\log n}$ **is sub-exponential**) Show that c^n is $\Omega(n^{\log n})$ for any $c > 1$, but that $n^{\log n}$ is never $\Omega(c^n)$.

Say : $f(n) = c^n$ Consider $\frac{f(n)}{g(n)} = \frac{c^n}{n^{\log(n)}}$

$g(n) = n^{\log(n)}$

taking limit as $n \to \infty$ on both sides,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{c^n}{n^{\log(n)}}$$

$\therefore$ as $n \to \infty$, $\log(n) \to \infty$

$\therefore n^{\log(n)} \to \infty$ $\therefore \frac{c^n}{n^{\log(n)}} \longrightarrow \infty$

& also $c^n \to \infty$

$\therefore \lim_{n\to\infty} \frac{c^n}{n^{\log(n)}} = \infty$ $\therefore$ The limit belongs to $[0, \infty]$

$\therefore \boxed{c^n = \Omega(n^{\log(n)})}$

Now, if $n^{\log n} = \Omega(c^n)$, then

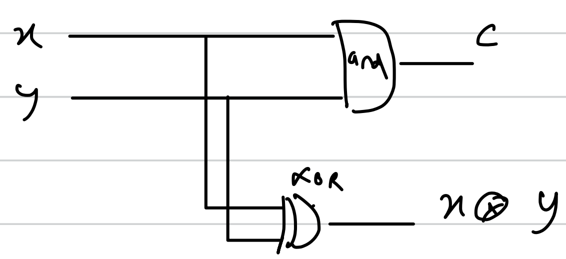
$n^{\log n} \geq c'.c^n$. — for some c' constant.

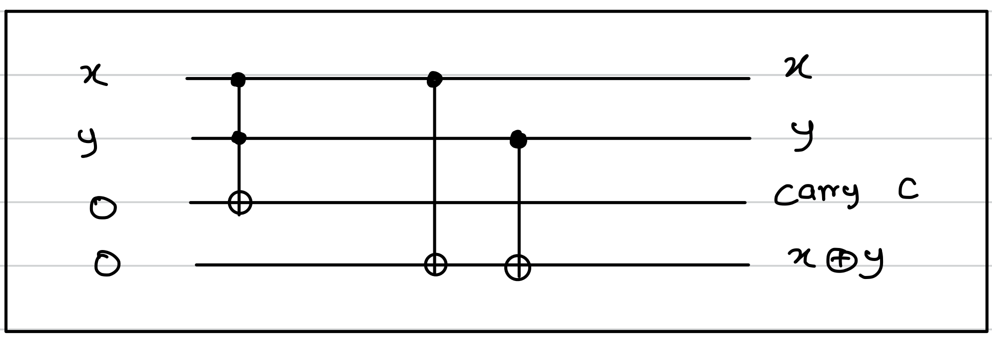
However, it is clear that for any $c > 1$, this is not true. $n^{\log n}$ will grow slower than c^n for any c greater than 1 for some $n > n_0$.

$\therefore \boxed{n^{\log(n)} \text{ is never } \Omega(c^n)}$

3.31

Exercise 3.31: (Reversible half-adder) Construct a reversible circuit which, when two bits x and y are input, outputs $(x, y, c, x \oplus y)$, where c is the carry bit when x and y are added.



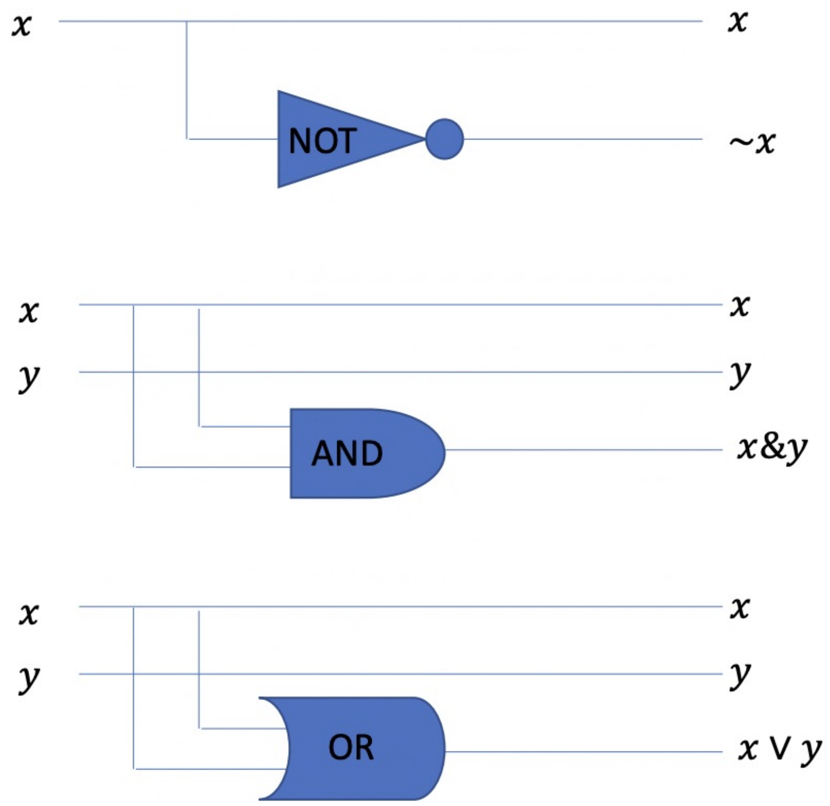


- This is the required circuit.

Problem 1

Problem 1 (Hard Problems Exist)

Argue that there exist Boolean functions that map n input bits to 1 output bit that require at least $2^n/n$ classical logic gates to compute. Assume the universal set of gates AND, OR, NOT and FANOUT (where FANOUT makes one copy of a bit). It is helpful to think of each step of the circuit as an *assignment*, where a new bit value is calculated from one or two existing bit values by applying a gate, and each new gate can draw on any of the current bit values:



After each assignment, the number of bits in the circuit has increased by 1.

∴ the boolean function is $f\ \{0,1\}^n \rightarrow \{0,1\}$

∴ for n bits, there can be 2^n different input.

∴ for 2^n different inputs, there can be 2^{2^n} different mappings of 2^n i/p to either 0 or 1.

∴ Total number of functions $= 2^{2^n}$

- Now, consider a circuit of size q.
- for each circuit, each gate require 2 bit to specify the type of the gate.
- $\therefore$ we will need $\log(q)$ bits to describe wires.

$\therefore$ a circuit of size (q) require $q(O(2) + O(\log(q)))$ bits to describe.

$\therefore$ Total bits required $= q(O(2) + O(\log(q)))$
$= q(O(\log(q)))$

$= O(q \log(q))$

$\therefore$ Total no. of functions computable by q-size circuit is $2^{O(q \log(q))}$

Now, assume $q = \frac{2^n}{n}$

$\therefore$ <u>No. of computable functions by q-size circuit</u> $= 2^{\frac{2^n}{n} \log \frac{2^n}{n}}$

$= 2^{\frac{2^n}{2}(\log 2^n - \log n)}$

$= 2^{\frac{2^n}{2}(n - \log n)}$

but $n - \log(n) = O(n)$ $\because$ $n - \log(n) \leq c \cdot n$ — $(n > n_0)$

$\therefore$ No. of computable functions by q-size circuit becomes

$= 2^{\frac{2^n}{2} O(n)}$

However, $2^{\frac{2^n}{n} O(n)}$ is much much smaller than 2^{2^n}.

$\therefore$ There exists some function which are not computable by a circuit of size q i.e. $\frac{2^n}{n}$.

$\because$ functions computable by circuit w/ size $\frac{2^n}{n}$ is much less than total no. of boolean functions $\left(f\ \{0,1\}^n \to \{0,1\}\right)$

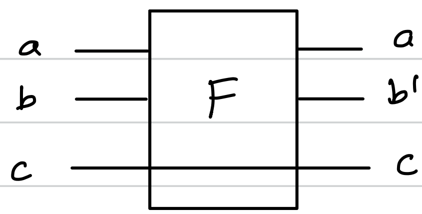
$\therefore$ $\boxed{\text{There exists boolean functions that require atleast } \frac{2^n}{n} \text{ gates to compute. } \left(\therefore \text{ Hard to compute boolean function do exist}\right)}$

Problem 2

Problem 2 (Classical Reversible Computation)

Show that the classical Fredkin gate can perform AND, OR, NOT, and NAND gates reversibly, provided that wires and ancilla bits (initialized in state 0 or 1) are available.

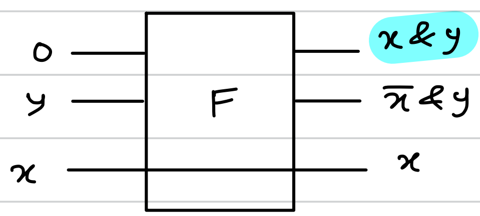
Fredkin gate :



$a' = a \ (c=0)$
$\quad = b \ (c=1)$

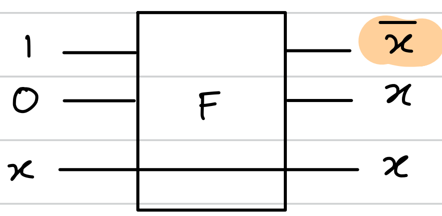
$b' = b \ (c=0)$
$\quad = a \ (c=1)$

① AND : $\boxed{x \,\&\, y}$

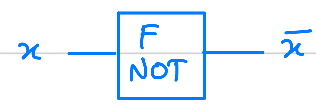


$\longrightarrow$ Let it be

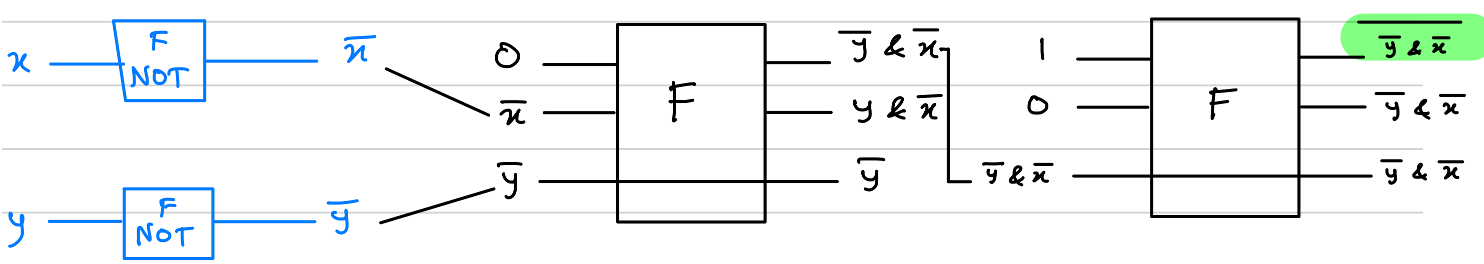
② NOT : $\boxed{\bar{x}}$



$\longrightarrow$ Let it be



③ OR : $\boxed{x + y}$



$\overline{\bar{y} \,\&\, \bar{x}} = \overline{\overline{y + x}} = y + x = x + y$

④ NAND :

$\boxed{\overline{x \,\&\, y}}$