CS 131 HW 5

Problem 1

① Consider the case where $F: \{0,1\}^* \rightarrow \{0,1\}$ takes 2 Turing Machine P, M as input & $F(P, M) = 1$ iff there is some input x such that P halts on x but M does not halt on x. Prove F is uncomputable.

Proof:

We need to give a direct reduction from HaltOnZero to function F. Let A be a TM that output a tuple of 2 machines such as (E, K) such that machine E halt on input x, but machine K does not halt on input x. We can design it from scratch or take a high level description of it. We now describe the reduction.

The reduction R transforms an input program B for HaltOnZero to the following program:

N(Z):

(a) Run B on 0

(b) Return $A(z) = (E(z), K(z))$

↳ the same input z to machine P, M

```
E(x=z):
    int i = 0;
    i += x;
    return i;

Z(x=z):
    int i = 0;
    while (true)
        i += x;
    return i;
```

We set $R(B) = N$ (for the program above)
The reduction does not run B on 0 but just compute the description of the program N. Clearly, the transformation is computable.

Analysis: We need to argue that for any program B,
HaltOnZero$(B) = F(R(B)) = F(N) = F(E, K)$
There are 2 cases.
If HaltOnZero$(B) = 1$, then N is equivalent to A which output a tuple of machines (E, K) as input to F that make $F = 1$.
Therefore $F(P = E, M = K) = 1$
On the other hand, if HaltOnZero$(B) = 0$, then N would not output a tuple of 2 machines as input to F that make F equal to 1. For example, 3 possible tuples could be (G, H) such that ① $G(z)$ halts & $H(z)$ halts; ② $G(z)$ does not halt & $H(z)$ does not halt; ③ $G(z)$ does not halt & $H(z)$ halt
Therefore N is not equivalent to turing machine A.
So $F(P = G, M = H) = 0$

$\therefore$ F is uncomputable ∎

Problem 2 (Exercise 9.13)

Part 1 :

Let $T_{BB} : \{0,1\}^* \rightarrow \mathbb{N}$ be defined as follows
For every string $P \in \{0,1\}^*$, if P represents a TM program such that when P is executed on the input 0 then it halts within M steps then $T_{BB}(P) = M$.
Otherwise (if P does not represent a TM program, or it is a program that does not halt on zero), $T_{BB}(P) = 0$. Prove T_{BB} is uncomputable.

Proof : We need to give a direct reduction from HaltOnZero.
Let show a reduction from HaltOnZero to T_{BB} (the function that tests whether program P halt on input 0 within M steps)
First, let A be a TM output a program Y that halt on input 0 with non-zero steps M.
Here is the high level description of the reduction.

Let $T_{BB}(P) = M \Rightarrow$ final output $= 1$
This refers to that fact that $T_{BB}(P) =$ non zero steps would result in 1.

The reduction R transforms an input program B for HaltOnZero to the following program:

N(x):

(a) Run B on 0

(b) Return A(Z)

the same input to program P

```
A(z=x):
init i = 0;
if (x == 0)
    for (int j = 0; j < 2; j++)
        i += j;
else while (true)
        i += 1;
return i;
```

We set $R(B) = N$ (ex. the program above)

The reduction does not run B on zero but just computes the description of the program N. Clearly, the transformation is computable.

Analysis: We need to argue that for any program B, HaltOnZero(B) function the same as $T_{BB}(R(B)) = T_{BB}(N)$ which means lead to the same final result either 1 or 0.

There are 2 cases. If HaltOnZero(B) = 1, then clearly N return a program Y that halt on input 0 within M steps (non-zero steps).

So $T_{BB}(N) = M$ = non zero steps, this result in final output = 1.

If HaltOnZero(B) = 0, then N would not halt on input 0, and in particular would not return program Y that halt on input 0 within M steps so that $T_{BB}(N) = 0$. $\therefore$ $T_{BB}(P)$ is uncomputable ∎

Problem 2 (Ex 9.13)

Part 2 Let Tower(n) ≡ the number $2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$ (n times)

Define NBB : $\mathbb{N} \to \mathbb{N}$ to be the function

$NBB(n) = \max_{P \in \{0,1\}^n} T_{BB}(P)$ ~~where~~. Prove that

NBB grows faster than Tower, in the sense that

$Tower(n) = o(NBB(n))$

S for every $\varepsilon > 0$ there is some N_0 such that

$Tower(n) < \varepsilon\, NBB(n)$ for every $n > N_0$

Proof :

According to the reference article, NBB(n) grows faster than any computable function which means NBB(n) grows faster than any function computable by a super turing machine. We need to show Tower(n) is computable by a TM.

First, we need to come up with a program whose description length at most n but takes $\omega(Tower(n))$ steps to stop which means we need to think of a TM that takes longer than Tower(n) steps to halt on 0.

Problem 2 cont.

First, show Tower (n) is a computable by a TM.
We need a n-tape Turing machine such that the output of the previous tape can be used to compute the next tape. The n-tapes are indexed from 1 .... n.
For example :

1 ; compute Tower (1)

2 ; compute Tower (2)

⋮

n ; compute Tower (n)

As a example, we can compute Tower (2) with the output of first tape.
The following process repeat from tape 1 to tape n.
1 Scan the previous tape a
2 The values for the next tape a+1 are doubled for b number of times where b = tape a's output

As a result, Tower (n) is computable function.
If Tower (n) is one of the program from P of

$NBB(n) = \max_{P \in \{0,1\}^n} T_{BB}(P)$, then this means that

→

it takes at least Tower (n) steps to compute NBB(n). P could equal to Tower (n) or some other program that takes more steps to compute.
Therefore, NBB (n) grows faster than Tower (n) ∎

Problem 3 Consider the grammar G with $V = \{R, X, S, T\}$, $\Sigma = \{0, 1\}$, $S = R$;

Rules: $R \to XRX \mid S$

$S \to 0T1 \mid 1T0$

$T \to XTX \mid X \mid \varepsilon$

$X \to 0 \mid 1$

(a) Give 3 strings in the language of G

$R \to XRX \to 0R1 \to 0\underline{S}1 \to 0\underline{0T1}1 \to 00X11$

$\to 00111$

$\boxed{\text{1st string: } 00111}$

$R \to S \to 1\underline{T}0 \to 1\underline{XTX}0 \to 11T00 \to 11\varepsilon 00$

$\boxed{\text{2nd string: } 1100}$

$R \to XRX \to 1\underline{R}0 \to 1\underline{XRX}0 \to 10\underline{R}10$

$\to 10\underline{S}10$

$\to 10\underline{0T1}10$

$\to 100X110$

$\to 1001110$

$\boxed{\text{3rd string: } 1001110}$

(b) $T \xrightarrow{*} 010$

$$R \to S \to 0T1 \quad ✗$$

$$R \to S \to 1T0 \quad ✗$$

$$R \to XRX \to 0R0 \to 0S0 \to 00T10 \quad ✗$$

$$\to 01T00 \quad ✗$$

$$R \to XRX \to 0R0 \to 0XRX0 \to 01R10$$

$$\to 01\underbrace{S}10$$

$$\to 01\underbrace{0T1}10$$

$$\to 010\varepsilon 110 \quad ✗$$

$$R \to XRX \to 0R0 \to 0XRX0 \to 01R10$$

$$\to 01S10$$

$$\to 01\underbrace{1T0}10 \quad ✗$$

$\boxed{\text{False}}$

(c) Give a description of the language of the grammar in english.

The language of the grammar does not accept 4 patterns of strings formed by {0, 1}

① Does not accept repeated patterns of 010 which is *010 (ex. 010/0/0 ....), with overlapping 0s.

② Does not accept repeated patterns of 101 which is *101 (ex. 10/0/0/ ....), with overlapping 1s

③ Does not accept pattern of all 1's (111 ....)

④ Does not accept pattern of all 0's (000 ....)

The grammar accept string as follows:

Let $\mathcal{L}$ be the length of the entire binary string

$$(1|0)^a\, 1\, (1|0)^b\, 0\, (1|0)^a$$

index: 1 ... a-1 a a+1 .... $\mathcal{L}$-a+1 .... $\mathcal{L}$

$$(1|0)^a\, 0\, (1|0)^b\, 1\, (1|0)^a$$

index: 1 ... a-1 a a+1 .... $\mathcal{L}$-a+1 .... $\mathcal{L}$

Problem 4 Design a context free grammar for language $L = \{x \in \{0,1\}^* : x \text{ has more 1's than 0}\}$

$\Sigma = \{0,1\}$

$S = K$

$V = \{A, K\}$

$K \rightarrow AK \mid 1A \mid 1K$

$A \rightarrow AA \mid 0A1 \mid 1A0 \mid \varepsilon$

Note that A generates words that have equal number of 1's & 0's. If a word B have more 1's than 0's then B is under one of the described category.

① $B = 1C$; C have more 1's than 0's

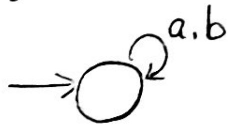
② $B = 1C$; C have equal number of 1's & 0's

③ $B = CD$; C have equal number of 1's & 0's & D have more 1's than 0's

Problem 5

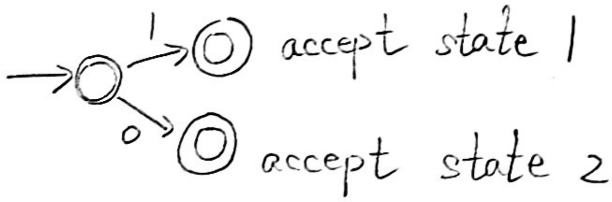
Proof : We need to give a reduction from equivalent to empty which is a transformation from empty to equivalent. Let C be a DFA that does not accept any string illustrated as follows :

C :



DFA C does not accept any string because it has no accepting states. Thus, the language it accept is ϕ which means DFA C output 1.

Let A be a TM that output a tuple of 2 DFAs (B, B') s.t. B & B' are drawn as follows :



B & B' are drawn like this

They are equivalent in a way that output & operations such as union, intersection & complement are the same between B & B'.
We first explain the reduction R from a high level description. The reduction R transform input DFA C for Empty to the following program N :

$N(z)$:

(a) Run DFA C with input ϕ

(b) Return $A(z) = (B, B')$

We set $R(C) = N$ (ex. the program above) The reduction does not run C with input z but just computes the description program N. Clearly, the transformation is computable.

Analysis. We need to argue for any input DFA C,

$\text{Empty}(C) = \text{Equivalent}(R(C)) = \text{Equivalent}(N)$

There are 2 cases.

If $\text{Empty}(C) = 1$, then clearly N is equivalent to $A(z) = (B, B')$ in which B & B' are equivalent DFAs, so $\text{Equivalent}(N) = \text{Equivalent}(B, B') = 1$

On the other hand, if $\text{Empty}(C) = 0$, then N would not refuse any strings, & in particular would not equivalent to $A(z)$ so that $\text{Equivalent}(N) = 0$

■