CS340A: Assignment 2

Submission By: Alphabets

---

1. **Solution:**
(a)
($\implies$) If input x is accepted by F, then there exists an accepting configuration sequence that starts with $< q_0, x >$
Lets say $x = x_0x_1x_2..x_n$
Initially, $x = zy$ where $z = x$ and y is empty, hence initial configuration $= < q_0, x >$ as F will be in state q_0 (initial state) after reading ϵ.
In case of DFA we will track the only sequence of state transitions and in case of NFA, we will track the sequence of state transitions which finally land up in accepting state to build our configuration sequence.

$$q_0 \xrightarrow{x_0} q_1 \xrightarrow{x_1} q_2 \xrightarrow{x_0} q_3... \xrightarrow{x_1} q_m$$

After reading $x_0, x = zy$ where $z = x_0$ and $y = x_1x_2...x_n$
Hence the configuration will be $< q_1, x_1x_2x...x_n >$
Similarly all the configurations can be captured and finally the last configuration after reading x_n will be $< q_m, \epsilon >$ where $q_m \in$ accepting state

So we have this configuration sequence i.e. $< q_0, x >, < q_1, x_1x_2..x_n >, < q_2, x_2x_3..x_n >, ... < q_m, \epsilon >$

It is indeed an accepting configuration sequence as

(a) Each $< q_i, y_i >$ is configuration (All $q_i \in$ Q as we have used those q_i which form the transitions in DFA and y_i is always suffix of x. Also F lands in q_i after reading z_i where $x = z_iy_i$)

(b) F moves from $< q_i, y_i >$ to $< q_{i+1}, y_{i+1} >$ in 1 step (From our construction).

(c) $q_m \in$ Accepting state

(d) $y_0 =$ x.

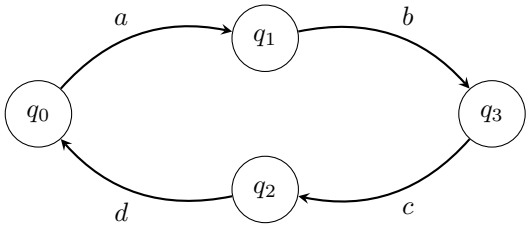
($\impliedby$) If there exists an accepting configuration sequence starting with $< q_0, x >$, then x is accepted
Say that configuration sequence is $< q_0, x >< q_1, y1 >< q_2, y2 > ... < q_m, \epsilon >$ where $q_m \in F$.
Final configuration is $< q_m, >$ which means that time $x = zy$ where y $= \epsilon$ hence $z = x$.
Thus after reading z, Automata is in q_m where $z = x$ and q_m is accepting state.
Hence x is accepted by automata.

(b)
Without loss of generalization, we can take number of final states in F to be 1.
Say there are m states, $q_0, q_1, q_2, ...q_m$ where q_0 is start state and q_m is final state. We will do so by moving to the sets in opposite fashion to that of transition.
Idea: We will store the set of configuration sequences for every state which takes F to q_m from that state. Represent this set as Q_i.
Computation: Start from the final state q_m, store $< q_m, >$ in Q_m.
Now move to all the states one by one which have state transition to q_m. For eg, $q_t \xrightarrow{a} q_m$, then store $< q_t, a >$ in Q_t. Similarly move to previous step. say it is $q_p \xrightarrow{b} q_t$, then store $< q_p, ba >$ in Q_p.
These steps will satisfy our invariant that set Q_i has those configuration sequences that takes F from q_i to q_m.
We can prove this by induction:
Base case:$< q_m, > \in Q_m$ and ϵ takes q_m to q_m. Hence base case is satisfied.
Induction hypothesis: Invariant holds for Q_i, where there is a path from q_i to q_m.
Induction proof: Now, we will proof that for all q_j s.t. q_j has a transition to q_i, the invariant holds for Q_j.

Say $q_j \xrightarrow{a} q_i$ and $Q_i = \{S_1, S_2, ..S_p\}$ now, Q_j will be $\{aS_1, aS_2, ...aS_p\}$
From q_j, we can go to q_i by reading a and after that we can go to q_m by reading any of the $\{S_1, S_2, ..S_p\}$.
Hence any of the $\{aS_1, aS_2, ...aS_p\}$ will take F from q_j to q_m.— Proved

Now, if we have no cycle in the DFA, i.e. we can never reach q_i from q_i after reading anything other than ϵ., there will be limited number of steps and we will calculate Q_0 which is indeed what we want to calculate.

Handling cycles:
We can see the DFA as a directed graph,
edges are transitions and vertices are states
We will make use of graph algorithm like DFS to find the cycle in DFA and store the string which will invoke the cycle. For eg:



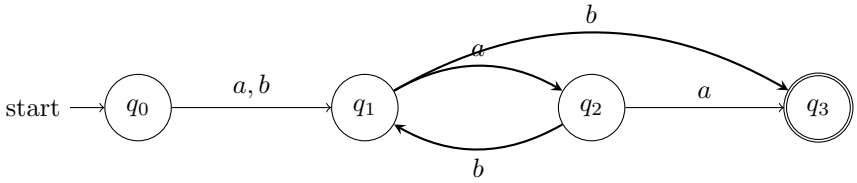
Here after detection of cycle $q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_3 \xrightarrow{c} q_2 \xrightarrow{d} q_0$. We store different strings which invoke cycle for all of these states. i.e. *abcd* for q_0, *bcda* for q_1, *cdba* for q_3 and *dcba* for q_2.
We will store it in different set namely C_i which will store strings that invokes cycle from q_i and leads to q_i after end of string in F.
We will use it like whenever we reach this state while doing our computation, we will do:

(a) $\forall s \in C_i$, append s^* in front of the string t for all $< q_i, t > \in Q_i$. It will also hold the invariant as s will take F from q_i to q_i only.

(b) Remove the transition which takes F into cycle so that the calculation terminates as we have successfuly removed the cycle.

After end of the calculation, Q_0 will store all the accepting configurations.

Proof that it is not CFL
Lets assume the set to be CFL and the language be L.
Without loss of generalization, we can assume that any configuration can be reprsented by single alphabet (For simpler calculations)
So, length of any string in L is same as number of configurations in the corresponding sequence.
We can take any finite automata for our proof, so we will use this one



So, all the accepting configuration sequences of this DFA belongs to L.
Now we will invoke pumping lemma.
Take the input string $a(ab)^p b$, it is indeed accepted by the DFA and we have proved in part a that if x is accepted by F, then there exists an accepting configuration sequence that starts with $< q_0, x >$,
In this case it will be
$< q_0, a(ab)^p b >, < q_1, (ab)^p b >, < q_2, b(ab)^{p-1} b >, < q_1, (ab)^{p-1} b >, ... < q_1, b >, < q_3, >$.
So, it has its equivalent string say S= $\tau_1 \tau_2 .. \tau_{2p+2} \in L$
where τ_1 represents $< q_0, a(ab)^p b >$, τ_2 represents$< q_1, (ab)^p b >$ and so on ..

$|S| > p$ obviously.

Now, S=uvxyz where $|vxy| \leq p$ and $|vy| > 0$

What we we will do is take i=0 in pumping lemma which will say that $s_1 = uv^0 xy^0 z \in$ L.
In s_1, atleast one alphabet will be deleted from the sequence of uvxyz which means the corresponding configuration is missing from the sequence. Now we will look into different cases for that alphabet or sequence of alphabets

(a) τ_1 is not present in s_1, which means the first configuration is not of form $< q_0, x >$ which leads to disqualification of this configuration sequence to be accepting as S never returns to q_0 again (state diagram). (using the fact that first configuration in accepting config. sequence should be $< q_0, x >$.

(b) τ_{2p+2} is not present in s_1, which means the last configuration is not of form $< q_3, >$ which leads to disqualification of this configuration sequence to be accepting as S never goes to q_3 before that (state diagram). (using the fact that last configuration in accepting config. sequence should be $< q_3, >$.

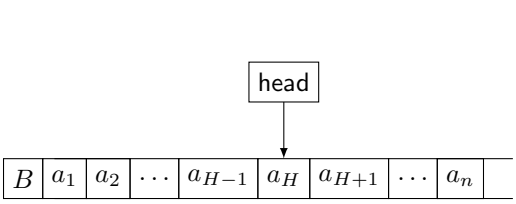
(c) $\tau_k \tau_{k+1} ... \tau_{k+k_1}$, $k_1 \geq 0$ is not present in s_1, which means τ_{k-1} is followed by τ_{k+1+k_1} but we cannot go from τ_{k-1} to τ_{k+1+k_1} in 1 step as number of input alphabets used in this transition is atleast 2. (one from τ_{k-1} to τ_k, another from τ_k to τ_{k+1}) which again disqualifies it to be a accepting configuration sequence.

This proves that L is not CFL.

□

2. **Solution:**
Idea: To prove that any computable set is accepted by 2-PDA, we just need to show that a Turing machine can be simulated using 2-PDA as turing machine accepts all computable sets

Proof: First we need to simulate Tape of turing machine which we will do using 2 stacks.
Say the TM M be $(Q, q_0, \Sigma, b, \delta, F_{acc}, Frej)$
Say the 2 stacks are S_1, S_2
Construction - S_1 will store the contents from left end of tape (After the cells containing B symbol) upto the cell in which head is pointing(excluding it) in the order as Tape[Left $\rightarrow$ *Right*]$\equiv S_1$[bottom $\rightarrow$ *top*]
S_2 will store contents from the cell in which head is pointing upto the right end of input in the order as Tape[Left $\rightarrow$ *Right*]$\equiv S_1$[top $\rightarrow$ *bottom*]



Tape and head of TM

S_1	S_2
a_{H-1}	a_H
...	...
a_3	a_{n-2}
a_2	a_{n-1}
a_1	a_n

We will maitain 2 invariants :

(a) Contents of S_1 + Rev(S_2) = Contents of Tape (left →right)

(b) Top of S_2 will point where head points in tape

Now as we have successfully simulated the static tape using 2 stacks, we move towards the tape operations:

(a) Head moves to left
Pop(S_1) and push same over S_2
Check for invariance:

i. (1): Now S_1 from bottom to top = $a_1a_2...a_{H-2}$
S_2 from bottom to top = $a_na_{n-1}...a_Ha_{H-1}$
S_1 + Rev(S_2) = $a_1a_2...a_{H-1}a_H...a_n$ This is same as Tape content after head moves to left.

ii. Now top of S_2 has a_{H-1} which is exactly same as where head points after moving left.

(b) Head moves to right
Pop from top of S_2 and push the same over S_1 Check for invariance:

i. (1): Now S_1 from bottom to top = $a_1a_2...a_H$
S_2 from bottom to top = $a_na_{n-1}...a_Ha_{H+1}$
S_1 + Rev(S_2) = $a_1a_2...a_{H-1}a_{H+1}...a_n$ This is same as Tape content after head moves to right.

ii. Now top of S_2 has a_{H+1} which is exactly same as where head points after moving right.

(c) Write a_{new} in place of a_H
Pop from top of S_2 and push a_{new} over S_2 Check for invariance:

i. (1): Now S_1 from bottom to top = $a_1a_2...a_{H-1}$
S_2 from bottom to top = $a_na_{n-1}...a_{H+1}a_{new}$
S_1 + Rev(S_2) = $a_1a_2...a_{H-1}a_{new}a_{H+1}...a_n$ This is same as Tape content after writing at head.

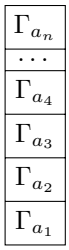
ii. Now top of S_2 has a_{new} which is exactly same as where head points after writing.

So, we have also simulated all the tape operations using 2 stacks and hence we can simulate a Turing machine using 2-PDA which makes it as powerful as Turing machine.

□

3. **Solution:**
We will simulate two stacks using finite number of counters and as we have shown in previous question, two stacks can play the role of tape of turing machine. We will show that counter turing machine can simulate normal turing machine and hence will accept all computable sets.

Proof: Stack alphabet is Γ where $|\Gamma| = k$
We will assign every alphabet of Γ a weight from 1 to k as- $\{\Gamma_1, \Gamma_2, \Gamma_3, ..\Gamma_k\}$
Say the stack content is-



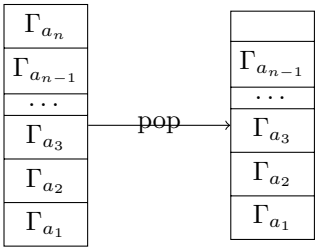
Where $(a_1, a_2, a_3, ...a_n) \in [1, k]$
We can convert this string into a number using $(k+1) - ary$ representation:

$$a_1 * k^n + a_2 * k^{n-1} + a_3 * k^{n-2} + ... + a_n k^0$$

If we store this number in counter. Then, we can store contents of a stack in counter. Also every number will have unique $(k+1) - ary$ representation. Then each number will represent unique stack content.
So, we can see two stacks in two counters(C_1, C_2).

now, for push-pop operations- We would require another counter.

Pop-operation:



$$S => a_1 * k^n + a_2 * k^{n-1} + a_3 * k^{n-2} + ... + (a_{n-1}) * k + a_n * k^0$$
$$S_{pop} => a_1 * k^{n-1} + a_2 * k^{n-2} + a_3 * k^{n-3} + ... + (a_{n-1}) * k^0$$

As per observation,

$$S_{pop} = \frac{S}{k}$$

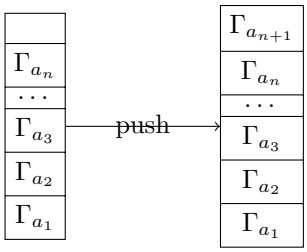
Thus, pop can be simulated by divide operation using third counter(C_3) which is currently at 0. Steps to follow-

(a) decrement k times from C_1 and increment one in C_3.

(b) if $C_1 <$ k then move to (c) else move to (a)

(c) Now, increase or decrease C_1 to make it same as C_3 and then decrement C_3 to 0.

Push-operation:



$$S => a_1 * k^n + a_2 * k^{n-1} + a_3 * k^{n-2} + ... + a_n * k^0$$

$$S_{pop} => a_1 * k^{n+1} + a_2 * k^n + a_3 * k^{n-1} + ... + (a_n) * k + a_{n+1}$$

As per observation,

$$S_{pop} = S * k + a_{n+1}$$

Thus, push can be simulated by multiply operation using C_3. Steps to follow-

(a) decrement 1 from C_1 and increment k times in C_3.

(b) if $C_1 == 0$ then increment a_{n+1} times in C_3 else move to (a)

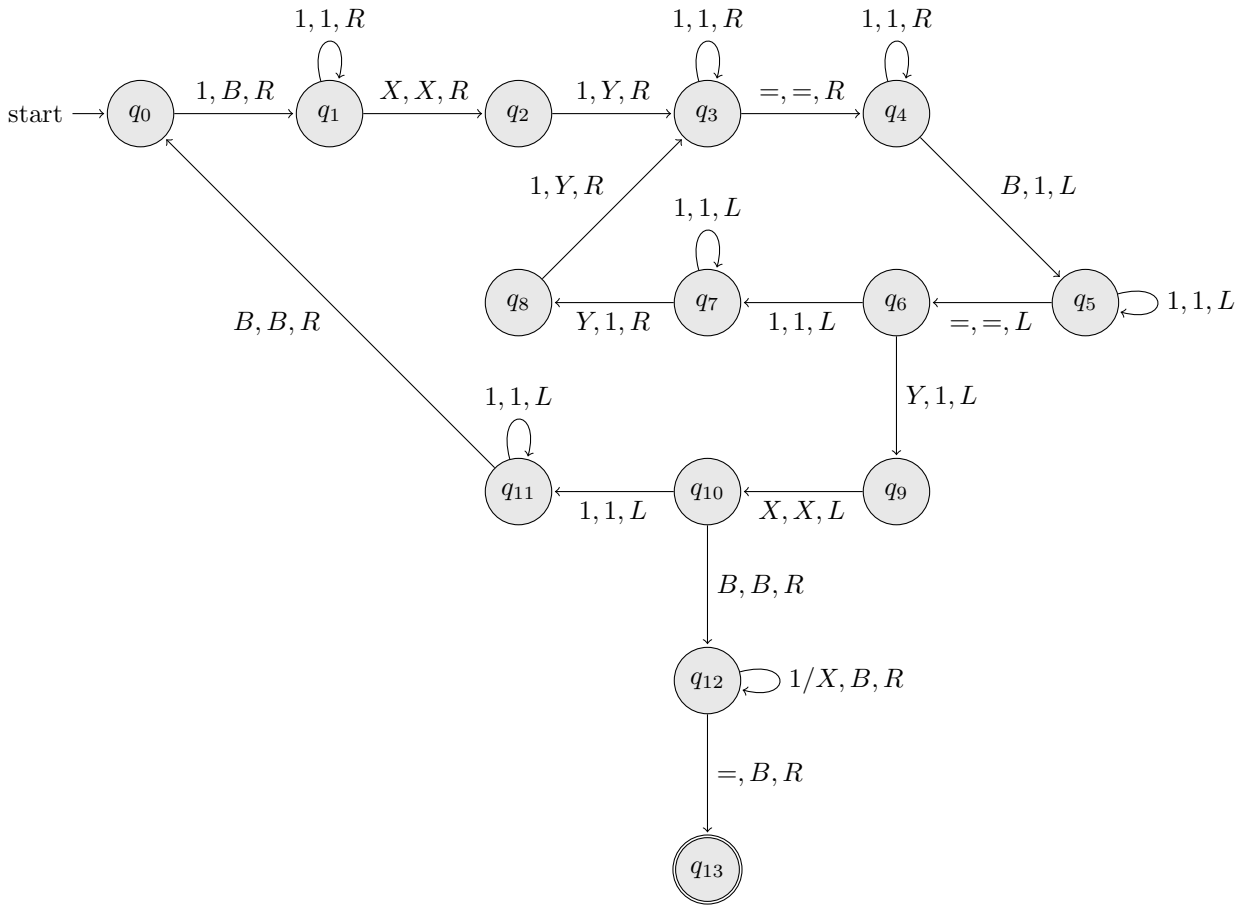
(c) Swap the contents of C_1 and C_3

Therefore, we are able to simulate two stacks using 3 counters which shows that every computable set can be accepted by a counter TM.

□

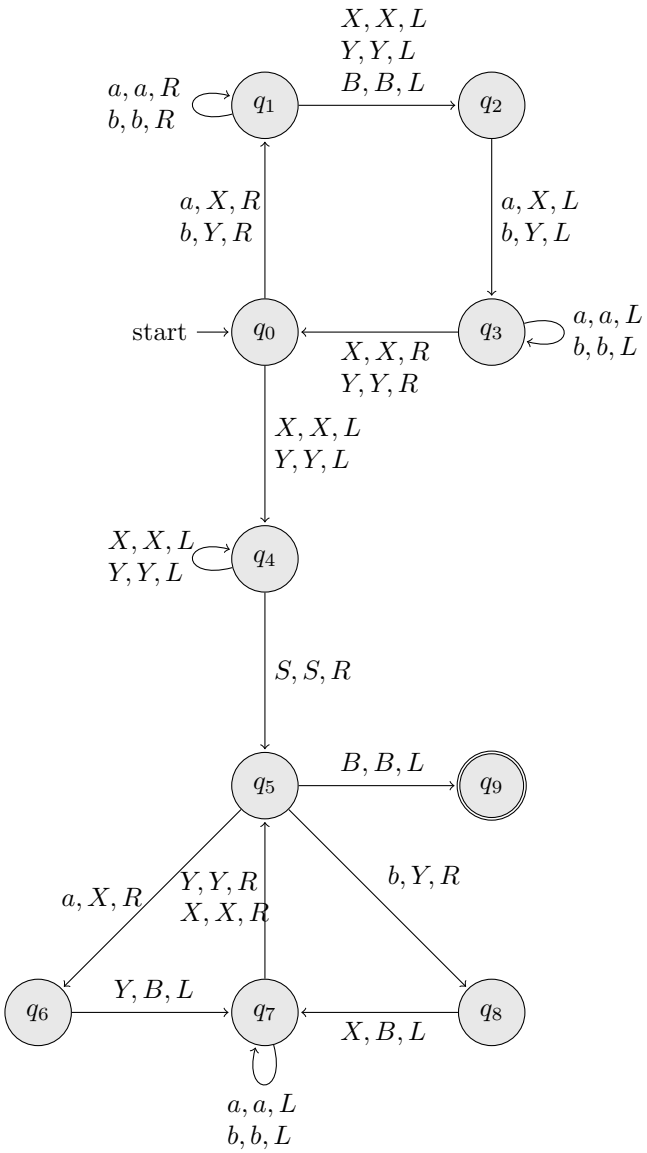
4. **Solution:**

TM M = $\{Q, q_0, \Sigma, B, \delta, F_{acc}, F_{rej}\}$ where
Q = $\{q_0, q_1, ...q_{13}\}$
q_0 is start state
$\Sigma = \{1, X, =\}$
B is blank
Tape alphabet= $\{1, X, =, Y\}$
$Facc = q_{13}$
δ on next page:



□

5. **Solution:**
   TM M = $\{Q, q_0, \Sigma, B, \delta, F_{acc}, F_{rej}\}$ where
   Q = $\{q_0, q_1, ...q_9\}$
   q_0 is start state
   $\Sigma = \{a, b\}$
   B is blank
   Tape alphabet= $\{a, b, X, Y\}$
   $Facc = q_9$
   δ on next page:



□

6. **Solution:**

Say L is computable where
L = $\{(p, q) | \exists\ x$ which is accepted by both p and q$\}$
Without loss of generalization, we can say that L' is also computable the same way as of L where
L'= $\{(p, q) | \exists\ x$ which is rejected by both p and q$\}$
Now define 2 more haulting turing machines as follows:
T_1 be a turing machine which accepts only w and rejects anything else
T_2 be a turing machine which rejects only w and accepts anything else.
where w is a string

As L and L' are computable, we can know whether a pair of turing machines (M_1, M_2) is part of L, L'.

Take any turing machine R,
If $(T_1, R) \in$ L $\implies \exists$ s s.t T_1 accepts s and R accepts s. $\implies$ R accepts w (1)
If $(T_1, R) \notin$ L $\implies \nexists$ s s.t T_1 accepts s and R accepts s. $\implies$ R does not accepts w (2)

If $(T_2, R) \in$ L' $\implies \exists$ s s.t T_2 rejects s and R rejects s. $\implies$ R rejects w (3)
If $(T_2, R) \notin$ L' $\implies \nexists$ s s.t T_1 rejects s and R rejects s. $\implies$ R does not rejects w (4)

If we have answer of above 4 expressions, we can say whether R halts on w or not as follows:
(1) & (3) $\implies$ Contradiction
(1) & (4) $\implies$ R halts on w
(2) & (3) $\implies$ R halts on w
(3) & (4) $\implies$ R does not halts on w.

As Halting problem is undecidable, here we land up in a contradiction. Hence L is uncomputable.

□