TOC - Assignment 2

Nirjhar Nath · BMC202239

① Given that L is a context-free language (CFL).

We need to show that there is a PDA M that accepts L such that M has just two states and doesn't make ϵ-moves.

Since ~~M is~~ L is a CFL, let $G = (V, T, P, S)$ be a CFG in Greibach normal form (GNF) that produces L.

If $\epsilon \in L$, then we can construct a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ that ~~acce~~ accepts L, as follows:
(let $\alpha_1 \alpha_2 \cdots \alpha_n = \epsilon$ if $n = 0$ throughout)

$Q = \{q_0, q_1\}$ having just two states,

$\Sigma = T$, $\Gamma = V \cup \hat{V}$ where $\hat{V} = \{\hat{A} \mid A \in V\}$;

δ is defined as follows:

If $S \to a \alpha_1 \alpha_2 \cdots \alpha_n$ exists in P, then

$$\delta(q_0, a, Z_0) = \{(q_1, \alpha_1 \alpha_2 \cdots \hat{\alpha}_n Z_0)\}$$

If $S \to a$ exists in P, then

$$\delta(q_0, a, Z_0) = (q_0, \epsilon)$$

If $A \to a \alpha_1 \alpha_2 \cdots \alpha_n$ exists in P, then

$$\delta(q_1, a, A) = \{(q_1, \alpha_1 \alpha_2 \cdots \alpha_n)\}$$

~~If $\hat{A} \to a \alpha_1 \cdots \alpha_n$ exists then~~

and $\delta(q_1, a, \hat{A}) = \{(q_1, \alpha_1 \alpha_2 \cdots \hat{\alpha}_n)\}$

If $A \to a$ exists in P, then

$$\delta(q_1, a, \hat{A}) = (q_0, \epsilon)$$

$F = \{q_1\}$

~~We have, $\alpha_1 \alpha_2 \cdots \alpha_n = \epsilon$ if $n = 0$ throughout.~~

If $\epsilon \notin L$, then we can construct a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, \phi)$ that accepts L, as follows;

$Q = \{q_0, q_1\}$ having just two states

$\Sigma = T$, $\Gamma = V \cup \hat{V}$ where $\hat{V} = \{\hat{A} \mid A \in V\}$

δ is defined as follows:

If $S \to a\alpha_1\alpha_2 \cdots \alpha_n$ exists in P, then

$$\delta(q_0, a, Z_0) = (q_0, \alpha_1\alpha_2 \cdots \hat{\alpha}_n)$$

If $S \to a$ exists in P, then

$$\delta(q_0, a, Z_0) = (q_1, \epsilon)$$

If $A \to a\alpha_1\alpha_2 \cdots \alpha_n$ exists in P, then

$$\delta(q_0, a, A) = (q_0, \alpha_1\alpha_2 \cdots \alpha_n)$$

and $\delta(q_0, a, \hat{A}) \ni (q_0, \alpha_1\alpha_2 \cdots \hat{\alpha}_n)$

If $A \to a$ exists in P, then

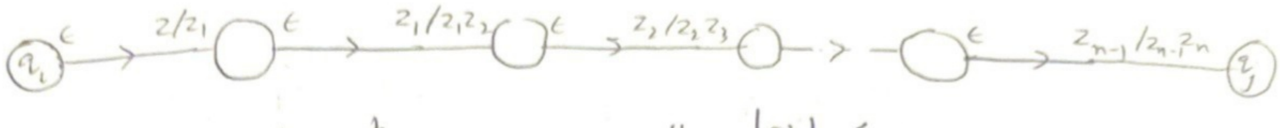
$$\delta(q_0, a, \hat{A}) \ni (q_1, \epsilon)$$

② Given that L is a CFG. We need to show that $L = L(M)$ for a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ such that $(p, \gamma) \in \delta(q, a, X) \Rightarrow |\gamma| \leq 2$.

$\because$ L is a CFG, so $\exists$ a PDA M' that accepts L. If there is a transition of the form

$$q_i \xrightarrow{a \qquad Z/\gamma} q_j$$

with $|\gamma| > 2$. Then if $\gamma = Z_1 Z_2 \cdots Z_n$, we can change the above transition to the following form



This also gives the same result, $|\gamma| \leq 2$.

Using this, we can construct a ~~DDD~~ PDA M such that

$$(p,\gamma) \in \delta(q,a,X) \Rightarrow |\gamma| \leq 2 \text{ and } L(M) = L.$$ ∎

③ Let $L = \{0^n 1^n \mid n \geq 0\} \cup \{0^n 1^{2n} \mid n \geq 0\}$.

We need to show that L cannot be accepted by ~~a~~ a deterministic PDA.

Clearly, L does not have prefix property; i.e., $\exists\, w_1, w_2$ ~~such~~ such that w_1 is a proper prefix of w_2, ~~and~~ $w_1 \in L$ and $w_2 \in L$.

$\therefore \nexists\, M$ such that $N(M) = L$.

~~Suppose~~

Assume, to the contrary that $\exists$ a deterministic PDA M such that $L(M) = L$.

Now, since the PDA has finitely many states and since there are infinitely many strings of the form $0^n 1^n$, so by PHP, $\exists$ two strings such that after them, we get them to be in the same state.

$\therefore \exists\, n_1, n_2$ with $n_1 \neq n_2$ such that

$$(q_0, 0^{n_1}1^{n_1}, Z_0) \vdash^* (p, \epsilon, \gamma) \text{ and } (q_0, 0^{n_2}1^{n_2}, Z_0) \vdash^* (p, \epsilon, \gamma)$$

and $p \in F$;

since after reading $0^{n_1}1^{n_1}$ and $0^{n_2}1^{n_2}$, both will be in identical accepting states.

For $0^{n_1}1^{n_1}1^{n_2}$ and $0^{n_2}1^{n_2}1^{n_2}$, we have

$$(q_0, 0^{n_1}1^{n_1}1^{n_1}, z_0) \vdash^* (p, 1^{n_1}, \gamma) \vdash^* (p', \epsilon, \gamma')$$

$$(q_0, 0^{n_2}1^{n_2}1^{n_2}, z_0) \vdash^* (p, 1^{n_2}, \gamma) \vdash^* (p'', \epsilon, \gamma'')$$

and $p', p'' \in F$

But then, reading the string $0^{n_1}1^{n_1}1^{n_2}$, we get

$$(q_0, 0^{n_1}1^{n_1}1^{n_2}, z_0) \vdash^* (p, 1^{n_2}, \gamma) \vdash^* (p'', \epsilon, \gamma'')$$

and $p'' \in F$.

$\therefore$ $0^{n_1}1^{n_1}1^{n_2}$ is accepted by the PDA but $0^{n_1}1^{n_1}1^{n_2} \notin L$ since $n_1 \neq n_2$, a contradiction.

$\therefore$ L cannot be accepted by a deterministic PDA.

(4) We need to construct a PDA that accepts $w \in \{a,b\}^*$ such that $\# a = 2 \# b$ in w.

~~Let M be a PDA that accepts all such strings so that M =~~

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a PDA that accepts all such strings, ~~where~~ which is constructed as follows:

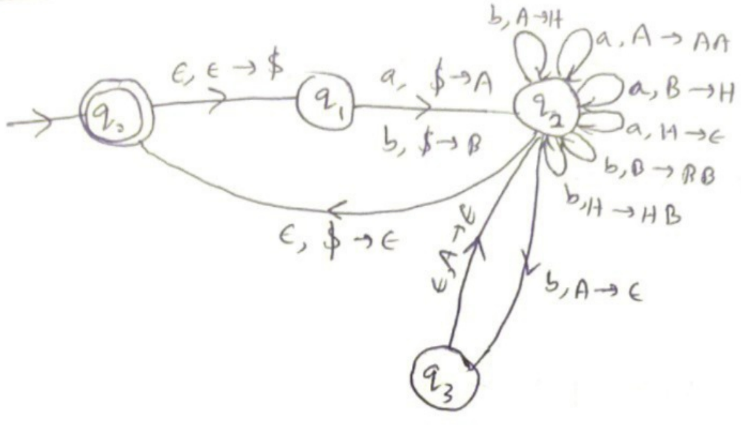
$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{a, b\}$

$\Gamma = \{A, B, H, \$\}$

$F = \{q_0\}$

The transition diagram below defines δ:



~~If the PDA is in state q_3, then~~

~~The stack~~ In this stack, the invariant will be:
if the PDA is in the state q_3,

$$\#a - \#b = (\#A - \#H - \#B)$$

So, for every a, we push A or remove H and replace B with H, and for every b, we push B or ~~[illegible]~~ replace A with H or delete AA from the state non-deterministically. ▨

(5) Let L be the language that consists of the set of primes encoded in binary; i.e., $L = \{ p \mid p \in \{0,1\}^* \text{ and } p \text{ written in decimal is a prime} \}$. We shall use pumping lemma to prove that L is not context-free.
Assume, to the contrary, that L is context-free.
Let p_{bin} be a binary string in L and P_{dec} be the corresponding decimal number.

We know that $p_{bin} \in L \implies P_{dec}$ is a prime
Let $n = 2^k$ be the pumping ~~length~~ lemma constant where k is the number of variables (here digits) and $p_{bin} = uvwxy$ such that $|vx| \geq 1$, $|vwx| \leq n$ and $|u| = a$, $|v| = b$, $|w| = c$, $|x| = d$, $|y| = e$.

Then, $p_{dec} = y + x\cdot 2^e + w\cdot 2^{d+e} + v\cdot 2^{c+d+e} + u\cdot 2^{b+c+d+e}$

Then by pumping lemma, we have,

$\forall m \geq 0,\ p_{bin}^{m} = u v^m w x^m y \in L.$

So, $p_{bin}^{m} = y + x\cdot 2^e(1+2^d+\dots+2^{(m-1)d}) + w\cdot 2^{md+e}$
$+ v\cdot 2^{c+md+e}(1+2^b+\dots+2^{(m-1)d})$
$+ u\cdot 2^{mb+c+md+e}$,

$$p_{bin}^{d} = y + x\cdot 2^e\left(\frac{2^{pd}-1}{2^d-1}\right) + w\cdot 2^{pd+e} + v\cdot 2^{c+pd+e}\left(\frac{2^{pb}-1}{2^b-1}\right) + u\cdot 2^{pb+c+pd+e}$$

By Fermat's little theorem, we have

$a^p \equiv a \pmod p$

$\Rightarrow (a^p)^k \equiv a^k \pmod p$

i.e., $a^{pk} \equiv a^k \pmod p$

If $b \mid a$, then $\left(\frac{a}{b}\right) \pmod p \equiv (ab^{-1}) \pmod p$

$\equiv \frac{a \pmod p}{b \pmod p}$

$\therefore 2^{pd-1} \equiv 2^{d-1} \pmod p$

$\Rightarrow 2^{pd} \equiv 2^d \pmod p$

and hence, $p_{bin}^{p} \equiv y + x\cdot 2^e + w\cdot 2^{d+c} + v\cdot 2^{c+d+e} + u\cdot 2^{b+c+d+e}$

$\equiv p \pmod p \equiv 0 \pmod p$

Let dec(x) denote the decimal representation of x.

Since $dec(p_{bin}^{p}) > p$, we get that $p \mid dec(p_{bin}^{p})$

$\therefore$ $dec(p_{bin}^{p})$ is not prime, i.e., $p_{bin}^{p} \notin L$, a contradiction to the pumping lemma.

$\therefore$ L is not context-free.

(6) Given, $G = (V, T, P, S)$ is a CFG. An algorithm that takes two strings $\alpha, \beta \in (V \cup T)^*$ as input and checks if $\alpha \underset{G}{\Rightarrow} \beta$ is given below:

We shall first create a CYK table for β for strings of all lengths as follows:

for $i = 1$ to n do
  if $\beta_{i1} \in \Sigma$
    then $V_{i1} = \{A \mid A \to \beta_{i1}$ is a production$\}$
  else
    $V_{i1} = \beta_{i1}$

Now we can check if some variable A can produce β_{ik} in constant time.

We shall create a dp table of ~~size~~ size $(n+1)(m+1)$ where $dp[0][0] = Y$ as $\alpha_{1,0} = \epsilon = \beta_{1,0}$ and $\alpha_{1,0} \Rightarrow \beta_{1,0}$

$dp[i][0] = N$ $\forall i \neq 0$ as the grammar is in CNF so no variable can generate ϵ.

$dp[0][j] = N$ $\forall j \neq 0$ as the empty string ϵ cannot generate anything.

$$dp[i][j] = \begin{cases} Y, & \text{iff } \alpha_{1,i} \Rightarrow \beta_{ij} \\ N, & \text{otherwise} \end{cases}$$

where $\alpha_{i,j}$:= substring of α of length j starting from i.

Initialize everything with N.

Now, for all pairs (i, j) with $i \leq n, j \leq m$ in ascending order, if $dp[i][j] = Y$, we shall try to match $\alpha_{i+1,1}$ with $\beta_{j+1,1}, \beta_{j+1,2}, \ldots, \beta_{j+1, m-j-1}$

~~If $\alpha_{i+1,1} \in T$ then else. if $\alpha_{i+1,1} \in \beta_{j+1}$~~

Thus,

if $\alpha_{i+1,1} \in T$ then

  if $\alpha_{i+1,1} = \beta_{j+1,1}$ then

   $dp[i+1][j+1] = Y$

else if $\alpha_{i+1,1} \in$ Variables

  for $k = 1$ to $m-j$

   if $\alpha_{i+1,1} \in CYK[j][k]$ then

    $dp[i+1][j+k] = Y$

$dp[i][j]$ was Y and $\alpha_{i+1,1} \underset{G}{\Rightarrow} \beta_{j+1,k}$.

So, $\alpha_{1,i+1} \Rightarrow \beta_{1,j+k}$ and hence $dp[i+1][j+k] = Y$.

Now we shall check if $dp[n][m] = Y$

if Y then $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$

$\alpha \overset{*}{\underset{G}{\not\Rightarrow}} \beta$

The time taken for CYK table making is $O(|\beta|^3)$. The time for dp table making is equal to that taken for accessing each cell and matching next character of α with all possible $\beta_{j+1,1}, \beta_{j+1,2}, \ldots, \beta_{j+1,m-j-1}$ with CYK checking in constant time.

$\therefore$ The time taken is $O(|\alpha||\beta||\beta|) = O(|\alpha||\beta|^2)$

$\therefore$ In terms of $|\alpha| + |\beta|$,

$$T(|\alpha| + |\beta|) = O(|\beta|^3) + O(|\alpha||\beta|^2)$$
$$= O((|\alpha| + |\beta|)^3).$$

The naive approach can be using the following algorithm:

```
Match(α, β):
    if α = " " and β = " " then
        return True
    if α = " " or β = " " then
        return False
    ans = False
    α = a α'
    for all possible partitions β = β1 β2 :
        if a derives β1 then
            ans = Match(α', β2) or ans
    return ans
```

~~In the above algorithm, O(|β|^3)~~

In the above algorithm, ~~O(|β|^3)~~

the time taken for the CYK table is $O(|\beta|^3)$.

Assuming $|\alpha| + |\beta| = k$, we get

$$T(2k) = T(2k-1) + T(2k-2) + \cdots + T(k)$$

$$\geq 2\,T(2k-2), \quad \because T(2k-1) = T(2k-2) + \cdots + T(k) \geq T(2k-2)$$

$$\Rightarrow T(2k) = O(2^{2k})$$

$$\Rightarrow T(n) = O(2^n)$$, which is in exponential time.

$\therefore$ Using dp makes it more efficient i.e., in polynomial time of the form $O(n^3)$.

(7) Given that $G=(V,T,P,S)$ is a CFG in CNF.
Let w be a string/word along with G as input. Let S be the ~~starts~~ start variable.
We shall associate a cost function $c : P \to \mathbb{R}^+$ to each production in the algorithm given below.
The modified CYK algorithm that takes a string $w \in T^*$ and computes a minimum cost parse tree for w if $w \in L(G)$.

CYK-table (G, w):

$n =$ Length of w
initialize all V_{ij} as in table

for $i = 1$ to n:

$V_{i,1} = \{ (A, \ldots, c, 1) \mid A \to w_{1,1}$ is a production with cost $c \}$

for $j = 2$ to n

for $i = 1$ to $n-j+1$

$V_{ij} = \emptyset$

for $k = 1$ to $j-1$

for productions $A \to BC$ and V_{ik} containing a tuple T_1 starting with B and $V_{i+k,\, j-k}$ containing a tuple T_2 starting with C,

tuple $= (A, B, C,$ min-cost(T_1) + min-cost(T_2) + cost$(A \to BC)$, $k)$

where cost$(A \to BC)$ is the cost of the production $A \to BC$.

if min-cost (tuple) < min-cost (T)

$T = tuple$

else

$V_{i,j} = V_{i,j} \cup \{tuple\}$

return ~~tuple~~ table

Min-Cost-Parse-tree (G, S, w, i, j):

if $T \notin V_{i,j}$ such that $T.S = S$

then return Null

else

$T = (S, A_1, A_2, \text{min-cost}, P)$

where S, A_1, A_2 are the variable and P is the partition.

return

S — Min-cost-Parse-tree (G, A_1, w, i, k)

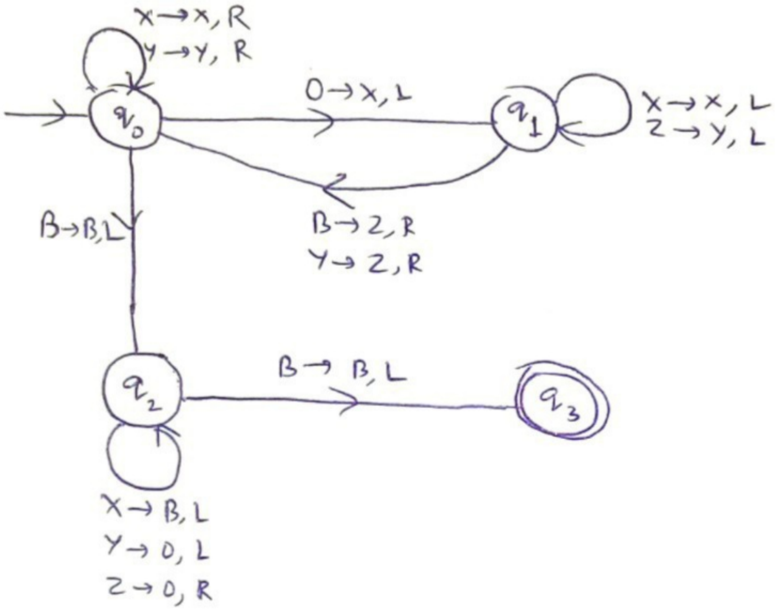
S — Min-cost-Parse-tree $(G, A_2, w, i+k, j-k)$

(8) We need to describe a turing machine M that converts 0^n as input to the binary encoding bin(n) for any $n \in \mathbb{N}$.

Let M be a turing machine such that

$$M = \left(\{q_0, q_1, q_2, q_3\}, \{0\}, \{0, 1, X, Y, Z, B\}, \delta, q_0, B, \{q_3\}\right)$$

and the transition function δ is defined as follows:



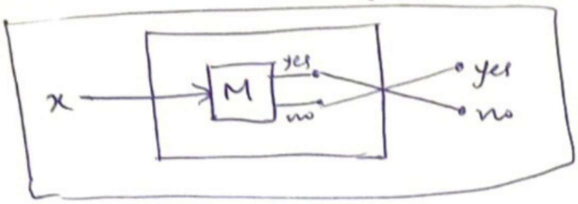
This turing machine M converts 0^n to bin(n) for any $n \in \mathbb{N}$.

(9) $\Rightarrow$: Let L be recursive.

Then let M ~~be~~ be a turing machine that accepts L and halts on all input.

Now, let M' be another turing machine such that on all inputs x, ~~[illegible]~~ run M on x and then if M accepts x, then accept it else reject it.

We have the following logical gate for M':



M'

Then clearly, $L(M') = \overline{L(M)} = \overline{L}$ and M' halts on all inputs.

$\therefore$ ~~[illegible]~~ L and $\overline{L}$ are both recursive and hence, L and $\overline{L}$ are both recursively enumerable.

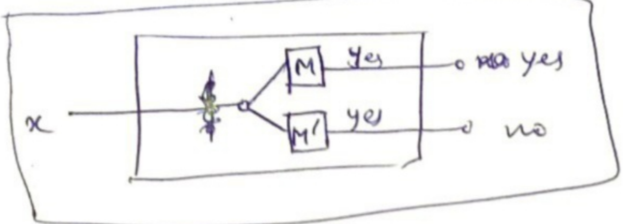
$\Leftarrow$: Let L and $\overline{L}$ be both recursively enumerable.

Then $\exists$ a turing machine M such that ~~[illegible]~~ $L(M) = L$.

Also, $\exists$ a turing machine M' such that $L(M') = \overline{L}$.

~~For any input [illegible]~~

We design a turing machine M'' which runs M and M'' simultaneously for any input as shown below:



M''

$\therefore$ Given an input x, ~~[illegible]~~

- if $x \in L$, then M halts at final state so x is accepted by M''
- if $x \notin L$, i.e., $x \in \overline{L}$, M'' halts, so x is rejected by M''.

$\therefore$ M'' halts on all inputs.

$\therefore$ L is recursive. ∎

(10) First we prove that a turing machine can be used to simulate a queue automaton.
We do it as follows:

Take a ~~two~~ two tape turing machine M such that the first tape contains the input and the second one is used to simulate the queue. We can simulate the push operation ~~as~~ and pop operation as:

<u>Push operation on a symbol S:</u> To do this, scan the 2nd tape of M from the left and place S in the first cell with a blank.

<u>Pop operation on a symbol S:</u> To do this, first we ~~s~~ move the head to the first cell with ~~the~~ $ symbol. Then we can scan the tape ~~from~~ starting from that cell that does not contain a # with a #.

Now we shall prove that a queue automaton can ~~be~~ ~~used to simu~~ simulate a TM.

Consider a TM: M and a PDA Q. Let the queue in Q contain the string in the input tape first and then contain a # symbol.

Let $w_1 w_2 \cdots w_n$ be the ~~input string~~ string in the input tape. If ~~at~~ the head is a cell i in the TM, the the queue is as follows:

$$w_i w_{i+1} \cdots w_n \# w_1 w_2 \cdots w_{i-1}$$

for ~~k~~ $1 \le i \le n$,
$w_i w_{i+1} \cdots w_n$ is indicated by $w_1 w_2 \cdots w_{i-1}$ in the TM.

We can simulate a move in which we go to the left on reading the input, as follows:

An operation like $(Q, w_i) \to (Q', w_i', L)$ makes the queue look like

$$w_{i-1}\, w_i'\, w_{i+1} \cdots w_n \# w_1 w_2 \cdots w_{i-2}$$

if the ~~cell~~ head was at cell i initially.

Now, we pop w_i and push w_i'. Popping and pushing the symbol on the front, we get the queue:

$$w_i'\, w_{i+1} \cdots w_n \# w_1 w_2 \cdots w_{i-1}$$

If the front of the symbol is \$, pop and push both \$ and the symbol that was popped before it.
Then the queue becomes:

$$w_i'\, w_{i+1} \cdots w_n \# w_1 w_2 \cdots w_{i-1} \$ w_{i-1}$$

i.e., w_{i-1} is added to the queue.

We can continue to push and pop the first symbol until we get:

$$\$ w_{i-1}\, w_i' \cdots w_n \# w_1 w_2 \cdots w_{i-2}$$

Finally, popping the \$ symbol we get the queue:

$$w_{i-1}\, w_i' \cdots w_n \# w_1 w_2 \cdots w_{i-2}$$

and hence we are done.

For the left operation, we can proceed ~~sim~~ similarly.