CS202 - AUTOMATA THEORY
ASSIGNMENT-2

DOMS | Page No.
Date / /

...e the grammar given as follows:

$S \rightarrow aB/bA$) → Rule 1, (R1)

$A \rightarrow a/aS/bAA$ → Rule 2, (R2)

$B \rightarrow b/bS/aBB$ → Rule 3, (R3)

(a) For the string $w = aaabbabbba$, find:-

(i) a leftmost derivation. ($S \Rightarrow^{L*} w$)

Soln:

$S \xrightarrow{R1} aB \xrightarrow{R3} aaBB \xrightarrow{R3} aaaBBB \xrightarrow{R3} aaabBB \xrightarrow{R3} aaabbBB \xrightarrow{R3} aaabbaBB \xrightarrow{R3} aaabbabB \xrightarrow{R3} aaabbabbS \xrightarrow{R1} aaabbabbbA \xrightarrow{R2} \boxed{aaabbabbba} = w$

(A leftmost derivation involves replacing leftmost non-terminal of the string, starting from S, and ending at 'w', which is made up of terminals)

(ii) a rightmost derivation. ($S \Rightarrow^{R*} w$)

Soln:

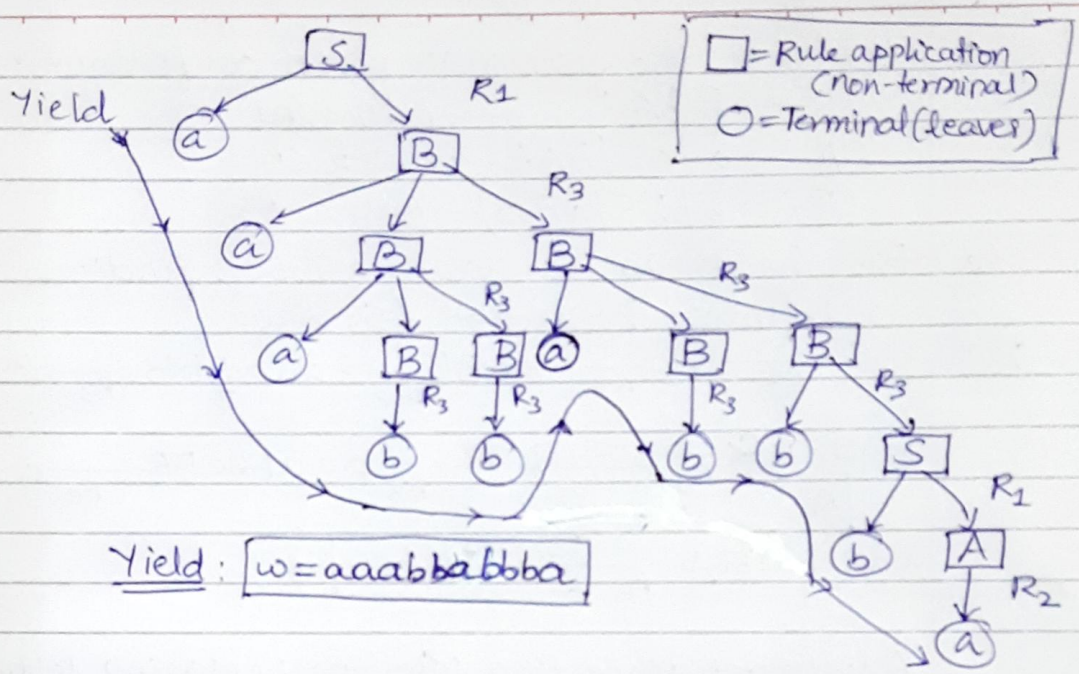
$S \xrightarrow{R1} aB \xrightarrow{R3} aaBB \xrightarrow{R3} aaBaBB \xrightarrow{R3} aaBaBbS \xrightarrow{R1} aaBaBbbA \xrightarrow{R2} aaBaBbba \xrightarrow{R3} aaBabbba \xrightarrow{R3} aaaBbabbba \xrightarrow{R3} \boxed{aaabbabbba} = w$

(A rightmost derivation involves replacing rightmost nonterminal of the string, starting from S, and ending at 'w', which is made up of terminals.

(iii) a parse tree.

Soln Let us illustrate the parse tree from which the above derivations (i) and (ii) are obtained/constructed.

→



(b) Is the grammar given ~~an~~ ambiguous? Justify.

Sol^n: A grammar is said to be ambiguous if there exists a word 'w' that has multiple, distinct ~~derivatio~~ parse trees (atleast 2) in that grammar.

This means that the word has atleast two leftmost derivations.

Consider w = aabbab.

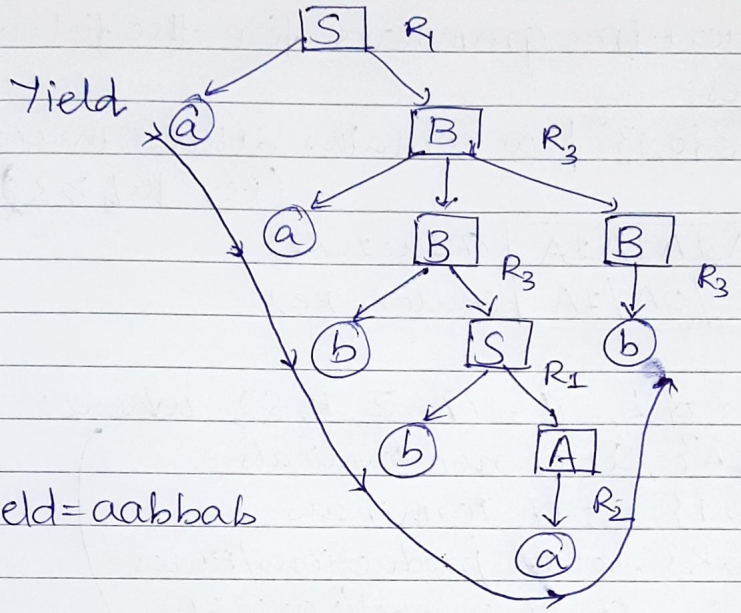
$S \xrightarrow[R_1]{} aB \xrightarrow[R_3]{} aaBB$

From aaBB:

- $aaBB \xrightarrow{R_3} aabSB \xrightarrow{R_1} aabbABB \xrightarrow{R_2} aabbaBB \xrightarrow{R_3} aabbab$
- $aaBB \xrightarrow{R_3} aabBB \xrightarrow{R_3} aabbS \xrightarrow{R_1} aabbaB \xrightarrow{R_3} aabbab$

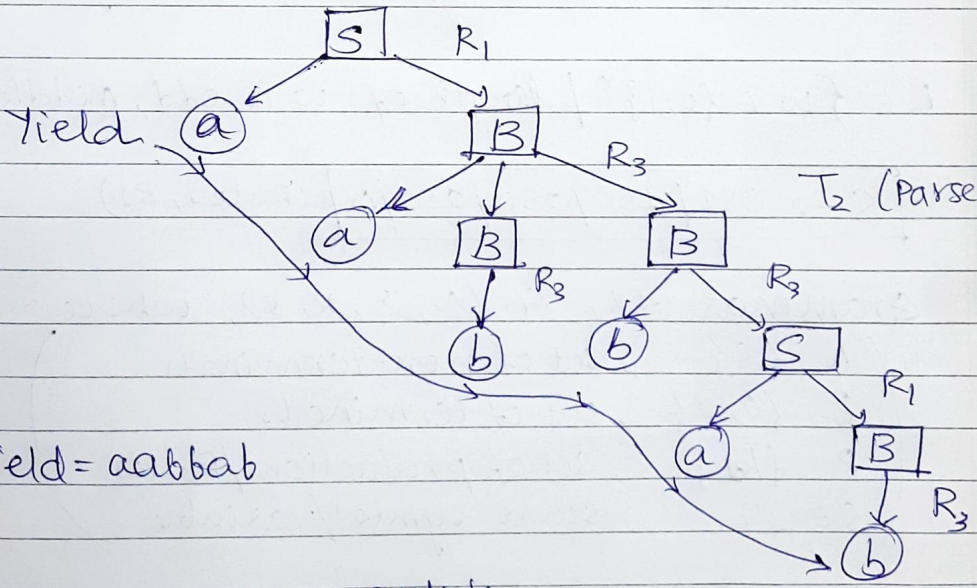
These are two distinct leftmost derivations of w = aabbab

Let us construct their parse trees.



T_1 (Parse Tree 1)

Yield = aabbab



T_2 (Parse Tree 2)

Yield = aabbab

A simple way to ~~tell~~ conclude that the two derivations are distinct: R_2 is applied once in T_1, but is never applied in T_2.

Hence, w has two leftmost derivations in G.
Thus, the grammar given is NOT unambiguous

DOMS | Page No.
Date / /

2. Write context-free grammars for the following languages:

(a) $L = \{ w \in \{0,1\}^* \mid w \text{ contains atleast three 1's} \}$ (i.e., $|w|_1 \geq 3$)

Soln:

$S \rightarrow A1A1A1A$ (Rule 1, R1)
$A \rightarrow \in / 0A / 1A$ (Rule 2, R2)

Grammar of L, $G = (N, \Sigma, R, S)$, where:-
- $N = \{S, A\}$, Set of non-terminals
- $\Sigma = \{0, 1\}$, Set of Terminals
- $R = \{R_1, R_2\}$, set of productions/Rules
- $S = S$, start symbol/variable.

(b) $L = \{ w \in \{0,1\}^* \mid$ length of w is odd, middle symbol is zero $\}$

Soln:

$S \rightarrow 0S1 / 1S0 / 0S0 / 1S1 / 0$ (Rule 1, R1)

Grammar of L, $G = (N, \Sigma, R, S)$, where:-
- $N = \{S\}$, set of non-terminals
- $\Sigma = \{0, 1\}$, set of terminals
- $R = \{R_1\}$, set of productions/Rules
- $S = S$, start symbol/variable

(c) $L = \{ a^i b^j c^k \mid i, j, k \geq 0 ;\ i+j = k \}$

Soln:

$S \rightarrow aSc / S_1$ (Rule 1, R1)
$S_1 \rightarrow \in / bS_1c$ (Rule 2, R2)

Grammar of L, $G = (N, \Sigma, R, S)$, where:-
- $N = \{S, S_1\}$, set of non-terminals
- $\Sigma = \{a, b, c\}$, set of terminals
- $R = \{R_1, R_2\}$, Set of productions/rules
- $S = S$, start Symbol/variable

DOMS | Page No.
Date / /

3. Construct a PDA equivalent to the following grammar:

$(S \to aAA) \longrightarrow$ Rule 1, R1

$(A \to aS/bS/a) \longrightarrow$ Rule 2, R2

Sol^n: We are given a context-free grammar, G, such that

$G = (N, \Sigma, R, S)$, where,

- $N = \{S, A\}$, set of non-terminals
- $\Sigma = \{a, b\}$, set of terminals
- $R = \{R_1, R_2\}$, set of productions/rules
- $S = S$, start symbol/variable

We want to convert it to a PDA,

$M = (Q, \Sigma, \Gamma, \Delta, q_0, F)$.

Let us construct M as:-

- $Q = \{p, q\}$, set of states in PDA
- $\Sigma = \{a, b\}$, set of letters in alphabet
- $\Gamma = \{a, b, S, A\}$, stack alphabet ($\Gamma = N \cup \Sigma$)
- $q_0 = \{p\}$, initial state of PDA
- $F = \{q\}$, final/accepting state of PDA

and, transition function, Δ is defined as follows:

$\Delta(p, \epsilon, \perp) = \{(q, push(S))\}$, [Initialization step] (Type 1)

Processing of non-terminals (Type 3):

$\Delta(q, \epsilon, S) = \{(q, pop(S), push(aAA))\}$

$\Delta(q, \epsilon, A) = \{(q, pop(A), push(aS)), (q, pop(A), push(bS)), (q, pop(A), push(a))\}$

Matching and popping (Type 2):

$\Delta(q, a, a) = \{(q, pop(a))\}$

$\Delta(q, b, b) = \{(q, pop(b))\}$

$(q, \$, \perp) \longrightarrow$ (condition for acceptance is empty stack and end of word!)

Final state

Hence, we have defined a PDA for the given CFG.

Idea for pseudocode (optional part) is attached at end.

DOMS | Page No. | Date / /

4. Give a context-free grammar for generating regular ~~operations~~ expressions defined over alphabet $\Sigma = \{a, b\}$

Soln: To describe a context-free grammar for generating regular expressions defined over Σ, we must consider all possible operations, and the general syntax of a regular expression.

$$S \to a \mid b \mid SS \mid S+S \mid S^* \mid (S) \mid \phi \quad \text{(Rule 1, R1.)}$$

Here,
- $S \to a/b$, handles termination.
- $S \to \phi$, handles empty language (which is regular)
- $S \to SS$, handles concatenation
- $S \to S+S$, handles "union" of languages of regular expressions
- $S \to S^*$, handles Kleene star
- $S \to (S)$, handles parentheses application.

The grammar G is defined as, $G = (N, \Sigma, R, S)$, where,

- $N = \{S\}$, set of nonterminals
- $\Sigma = \{a, b, +, *, (,), \phi\}$, set of terminals. (↳ The symbol ϕ, not the empty set)
- $R = \{R_1\}$, ↳ R_1 is described above
- $S = S$, ↳ start symbol/variable

Important

Note: If we use the symbol 'ϵ' to denote the language containing empty string, then $(S \to \epsilon)$ must be appended as a rule to the grammar above.

HOWEVER: For ϵ, we can have a zero-step ~~run of~~ derivation as well.

In the event that 'ϵ' is the symbol ~~for~~ such that $lang(\epsilon) = \{\epsilon\}$

Then, the rule R_1 will be:

$$S \to a \mid b \mid SS \mid S+S \mid S^* \mid (S) \mid \phi \mid \epsilon$$

[Also, append ϵ to Σ of G]

DOMS | Page No.
Date / /

5. Show that if all productions of a CFG G' are of the form $A \to wB$ or $A \to w$, then L(G) is a regular set.

Soln: The given description is that of a "right linear grammar". Any language having a right linear grammar, will be regular. (This is to be proven.)

From the given format, all rules are of the form $A \to wB$ or $A \to w$.
$A \to wB$ is the only type of rule that provides for a "relation" between non-terminals, and any cases of $A \to w$, where $w = \epsilon$, can easily be cleared up. This can be done by replacing A with ϵ, in all the other rules that it appears in.

Cases: (When $A \to \epsilon$ is a rule)

$B \to wA$, $A \to \epsilon$ $\Rightarrow$ $B \to w$

$B \to wA$, $A \to w'$, $A \to \epsilon$ $\Rightarrow$ $B \to ww' | w$

$B \to wA$, $A \to w'C$, $A \to \epsilon$ $\Rightarrow$ $B \to w | ww'C$

$B \to wA$, $A \to w'C$, $A \to w''$, $A \to \epsilon$ $\Rightarrow$ $B \to w | ww'' | ww'C$

(As well as eliminating A in general)

The only case of ϵ-rules that are left unaccounted for is if $(S \to \epsilon) \in R$, (where R is set of rules)

Consider a right linear grammar G, that has been reduced to contain no ϵ-rules, as described above. ($S \to \epsilon$ may persist)

The grammar G is such that, G = (N, Σ, R, S),
where, N = Set of nonterminals
Σ = Alphabet (set of terminals)
R = Set of rules (all of the form A → wB or A → w)
S = Start symbol/variable

We will construct an NFA 'M' using this grammar, G.
Since we are only required to prove that all regular languages having right linear grammars are regular (and not that all regular languages have a right linear grammar), we will just construct a machine 'M' from G, and argue that L(M) = L(G).

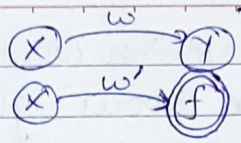
↓

(i.e.) It is not necessary to establish that all NFAs can be converted to right linear grammars. (though that is possible)

The derivation of a word 'w' would be of the form: $S \vdash w_1 A_1 \vdash w_1 w_2 A_2 \vdash^* w_1 \dots w_{n-1} A_{n-1} \vdash w_1 w_2 \dots w_n$
such that,

$$\left(S \to w_1 A_1,\ A_1 \to w_2 A_2, \dots, A_{n-2} \to w_{n-1} A_{n-1},\ A_{n-1} \to w_n\right)$$

Here, all productions except the last one are of the form A → wB. The last rule is of the form A → w
This can be modelled into an NFA, where the states denote non-terminals of the grammar. Since the final state would contain no non-terminals, we add an extra state, f, to handle this. (i.e., to be a final state.)
All rules of the form X → wY are simulated in the transition function of M such that, $\hat{\delta}(X, w) = Y$. ($\hat{\delta}$ is a function from $Q \times \Sigma^* \to Q$, where $Q = N \cup \{f\}$)

$X \to^* wY \equiv$ 

$X \to^* w' \equiv$

i.e., iff $X \to^* wY$, iff $Y \in \hat{\delta}(X, w)$

$X \to^* w$, iff $f \in \hat{\delta}(X, w)$

$S \in F$ iff $(S \to \epsilon) \in R$ (iff $\hat{\delta}(S, \epsilon) \in F$)

Hence, $\hat{\delta}$ is defined as above.

So, NFA, $M = (Q, \Sigma, \delta, q_0, F)$

$Q = N \cup \{f\}$

$\Sigma = \Sigma$

$\delta \Rightarrow$ defined using $\hat{\delta}$, (or a homomorphism instead) ~~as explained later~~.

$q_0 = S$

$F = \{f\}$ (If $S \to \epsilon \in R$, then $S \in F$)

[A better definition for δ would be:
$\delta(X, a) = \{Y \mid X \to aY \in R\} \cup \{f \mid X \to a \in R\}$]

When we defined $\hat{\delta}(X, w) = Y$, we could consider several dummy states $\alpha_1, \alpha_2, \ldots \alpha_n$, such that,

($\delta(X, w_1) = \alpha_1$, $\delta(\alpha_1, w_2) = \alpha_2$, $\delta(\alpha_2, w_3) = \alpha_3$, ... $\delta(\alpha_{n-1}, w_n) = Y$)

(where, $w = w_1 w_2 \cdots w_n$.

→ To ensure that transitions occur due to letters (and the definition of transition function is not diluted).

★ Another way to define δ, without dummy states would be to construct a homomorphism from words 'w' to some alphabet 'σ', such that every word appearing in a rule $A \to wB$ or $A \to w$ would be mapped to ONE letter in σ.

The homomorphism would be $g: \Sigma^* \to \sigma$.
One way to think about $g(w)$ would be to consider an alphabet where the words in Σ^* are put as subscripts.

(i.e.) If $\hat{\delta}(X, w) \to Y$, we transform it to $\boxed{\delta(X, a_w) \to Y}$ to ensure integrity of δ ($\delta: Q \times \sigma \to Q$)

$\left(\begin{array}{l} \sigma = \{a_w \mid w \in \Sigma^*\} \\ Q = N \cup \{f\} \end{array}\right)$ ← definition of σ and Q

~~To argue correctness~~

Here, NFA, $M = (Q, \sigma, \delta, q_0, F)$, such that,

$Q = N \cup \{f\}$
$\sigma = \{a_w \mid w \in \Sigma^*\}$,
$\delta: Q \times \sigma \to Q$
$q_0 = S$
$F = \{f\}$

Description of δ:
If $(X \to wY) \in R$
then $\delta(X, a_w) = Y$
If $(X \to w) \in R$
then $\delta(X, a_w) = f$
If $(S \to \epsilon) \in R$
then $S \in F$.

or
If $\hat{\delta}(S \to \epsilon) \in F$,
then $S \in F$.

More concisely, if "ϵ-closure"(S) contains 'ϵ', then $S \in F$.

→ To argue correctness of this construction:

If $w \in L(G)$, then, there exists a derivation based on the rules of G, such that, $S \vdash^* w$,
i.e., for $w = w_1 w_2 \cdots w_n$, (and rules $S \to w_1 A_1$, $A_1 \to w_2 A_2$, etc.)
$S \vdash w_1 A_1 \vdash w_1 w_2 A_2 \vdash^* w_1 w_2 \cdots w_{n-1} A_{n-1} \vdash w$ (and $A_{n-1} \to w_n$)
$(w = w_1 w_2 \cdots w_n)$

For these productions, there exist corresponding rules in the NFA that lead to final state f.,
the run is: $S \xrightarrow{a_{w_1}} A_1 \xrightarrow{a_{w_2}} A_2 \xrightarrow{a_{w_3}} A_3 \to \cdots \xrightarrow{a_{w_{n-1}}} A_{n-1} \xrightarrow{a_{w_n}} f$

These are accounted for in our description of $\delta : Q \times \Sigma \to Q$.

Hence, the derivation can be translated into an accepting run of the NFA. So, $L(G) \subseteq L(M)$ ⟶ (1)

If some word $a_{w_1} a_{w_2} \ldots a_{w_n}$ has an accepting run in the NFA 'M', then, similar to the proof above, we can construct the corresponding derivation,
$(S \vdash w_1 A_1 \vdash^* w_1 w_2 \ldots w_n)$

Also, note that, $a_{w_1} a_{w_2} \ldots a_{w_n}$ is the word accepted by the machine. Using the inverse homomorphism g^{-1}, we can recover the word. $(g^{-1}(a_{w_i}) = w_i)$.

Thus, ~~technically~~ $L(M) \subseteq L(G)$ ⟶ (2)

(1) & (2) $\Rightarrow$ $\boxed{L(M) = L(G)}$

Hence, the construction is correct.
Keep in mind that the word segments $w_1, w_2, \ldots w_n$ have been pushed to subscripts (via homomorphism) to ensure that each transition in the NFA consumes ONE word segment.

↳ Considering the segments themselves, we would be required to deal with dummy states that appear during processing of letters in a word segment.

Both intuitively and construction-wise, we have, hence, proved that every language that has a right linear grammar is regular.

DOMS | Page No.
Date / /

6. Construct a CFG equivalent to the NPDA shown in figure 1, using the algorithm done in the class.

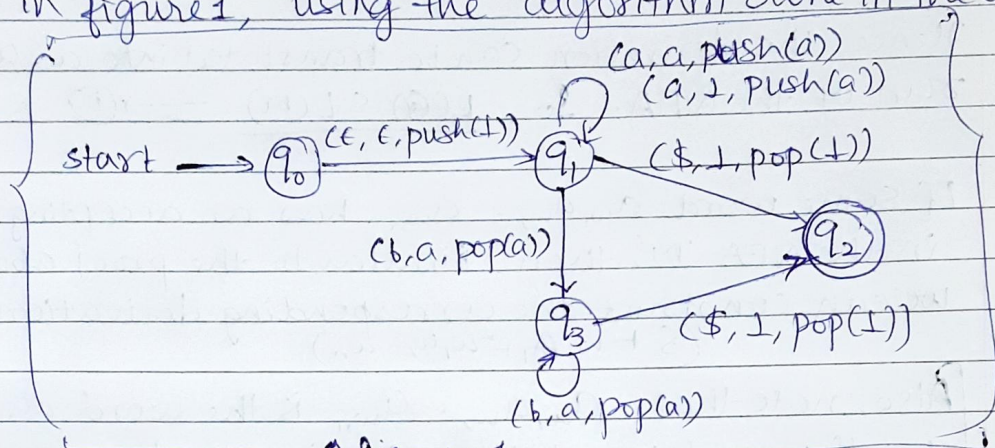


figure 1

Soln: Firstly, let us state the algorithm done in class:

Given a PDA, $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$,
we construct a grammar, $G = (N, \Sigma, R, S)$, such that:
$S = S$, $\Sigma = \Sigma$, $N = \{A_{p,q} \mid p, q \in Q\} \cup \{S\}$
R contains some types of rules, including:

Type 1: $S \longrightarrow A_{q_0,f_1} \mid A_{q_0,f_2} \mid \cdots \mid A_{q_0,f_l}$, where $F = \{f_1, f_2, \cdots, f_l\}$

Type 2: $A_{p,q} \longrightarrow A_{p,r} \cdot A_{r,q} \quad \forall\ p, r, q \in Q$

Type 3: $A_{p,q} \longrightarrow a A_{p',q'} b \quad \forall\ p, q, p', q' \in Q$, such that $(p, a, X) \vdash (p', push(x))$ and $(q', b, X) \vdash (q, pop(x))$ (any in top)

Type 4: $A_{p,p} \longrightarrow \epsilon \quad \forall\ p \in Q$

So, Applying this directly to the PDA given to us,

Type 1 rule: $S \longrightarrow A_{02}$ Rule 1 (R_1)

Type 2 rules:
$A_{00} \longrightarrow A_{00}A_{00} \mid A_{01}A_{10} \mid A_{02}A_{20} \mid A_{03}A_{30}$ Rule 2 (R_2)
$A_{01} \longrightarrow A_{00}A_{01} \mid A_{01}A_{11} \mid A_{02}A_{21} \mid A_{03}A_{31}$ Rule 3 (R_3)
$A_{02} \longrightarrow A_{00}A_{02} \mid A_{01}A_{12} \mid A_{02}A_{22} \mid A_{03}A_{32}$ Rule 4 (R_4)
$A_{03} \longrightarrow A_{00}A_{03} \mid A_{01}A_{13} \mid A_{02}A_{23} \mid A_{03}A_{33}$ Rule 5 (R_5)

DOMS | Page No.
Date / /

$A_{10} \rightarrow A_{10}A_{00} \mid A_{11}A_{10} \mid A_{12}A_{20} \mid A_{13}A_{30}$ Rule 6 (R_6)
$A_{11} \rightarrow A_{10}A_{01} \mid A_{11}A_{11} \mid A_{12}A_{21} \mid A_{13}A_{31}$ Rule 7 (R_7)
$A_{12} \rightarrow A_{10}A_{02} \mid A_{11}A_{12} \mid A_{12}A_{22} \mid A_{13}A_{32}$ Rule 8 (R_8)
$A_{13} \rightarrow A_{10}A_{03} \mid A_{11}A_{13} \mid A_{12}A_{23} \mid A_{13}A_{33}$ Rule 9 (R_9)

$A_{20} \rightarrow A_{20}A_{00} \mid A_{21}A_{10} \mid A_{22}A_{20} \mid A_{23}A_{30}$ Rule 10 (R_{10})
$A_{21} \rightarrow A_{20}A_{01} \mid A_{21}A_{11} \mid A_{22}A_{21} \mid A_{23}A_{31}$ Rule 11 (R_{11})
$A_{22} \rightarrow A_{20}A_{02} \mid A_{21}A_{12} \mid A_{22}A_{22} \mid A_{23}A_{32}$ Rule 12 (R_{12})
$A_{23} \rightarrow A_{20}A_{03} \mid A_{21}A_{13} \mid A_{22}A_{23} \mid A_{33}A_{33}$ Rule 13 (R_{13})

$A_{30} \rightarrow A_{30}A_{00} \mid A_{31}A_{10} \mid A_{32}A_{20} \mid A_{33}A_{30}$ Rule 14 (R_{14})
$A_{31} \rightarrow A_{30}A_{01} \mid A_{31}A_{11} \mid A_{32}A_{21} \mid A_{33}A_{31}$ Rule 15 (R_{15})
$A_{32} \rightarrow A_{30}A_{02} \mid A_{31}A_{12} \mid A_{32}A_{22} \mid A_{33}A_{32}$ Rule 16 (R_{16})
$A_{33} \rightarrow A_{30}A_{03} \mid A_{31}A_{13} \mid A_{32}A_{23} \mid A_{33}A_{33}$ Rule 17 (R_{17})

Type 3 rules:

$A_{13} \rightarrow aA_{11}b$ → Rule 18 (R_{18})
$A_{13} \rightarrow aA_{13}b$ → Rule 19 (R_{19})

The only possible push-pop pairs

Since,

$(q_1, a, 1) \vdash (q_1, push(a))$ and $(q_1, b, a) \vdash (q_2, pop(a))$
$(q_1, a, a) \vdash (q_1, push(a))$ and $(q_2, b, a) \vdash (q_2, pop(a))$

~~Hence,~~ (will elaborate on this after type 4 rules)

Type 4 rules:

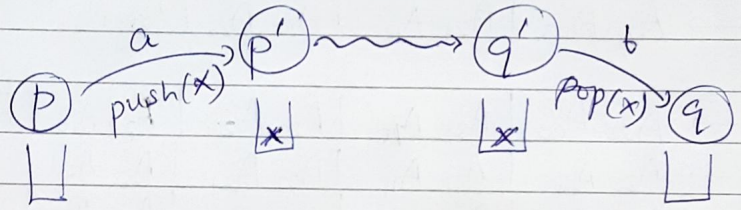
$A_{00} \rightarrow \epsilon$ Rule 20 (R_{20})
$A_{11} \rightarrow \epsilon$ Rule 21 (R_{21})
$A_{22} \rightarrow \epsilon$ Rule 22 (R_{22})
$A_{33} \rightarrow \epsilon$ Rule 23 (R_{23})

Hence, $R = \{R_1, R_2, \ldots, R_{23}\}$ is set of rules
Thus, we have defined a context free grammar for the given PDA.

DOMS | Page No.
Date / /

A description of type ③ rules:

If some element has been pushed in one transition $p \to p'$ on reading 'a' from the string, then it must be popped at some transition $q \to q'$ on reading some 'b' from the string. Hence, we can reduce A_{pq} to $aA_{p'q'}b$



Hence, we can pair the pushes and pops, to form the type ③ rules, which were:-

$$\left.\begin{aligned} A_{13} &\to aA_{11}b \\ A_{13} &\to aA_{13}b \end{aligned}\right\} \text{ i.e. } \boxed{A_{13} \to aA_{11}b \mid aA_{13}b}$$

Since many of the type 2 rules will reduce due to repetition/unreachability, the number of rules will reduce greatly.
We obtain:

$\boxed{A_{02} \to aA_{02}b \mid \epsilon}$ as the singular rule in the grammar.

($a^n b^n$, essentially)

Hence, we have established the rules of CFG.

Reference: Resources by Russ Cox on Implementing Regex

DOMS | Page No.
Date / /

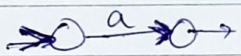
Idea for Question 4 pseudocode:

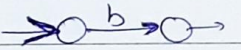
First step: Writing a CFG for regular expressions. (Done in Q4)

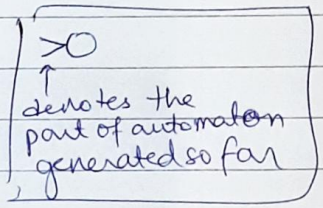
Second step: Parsing CFG
↳ It reads through characters of the regular expression, and as it encounters certain operations like A+B, A^*, AB, it instructs the program on which functions/modules to generate

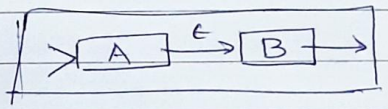
Third step: Automaton generating routines.
↓ forms nodes and patches as required.

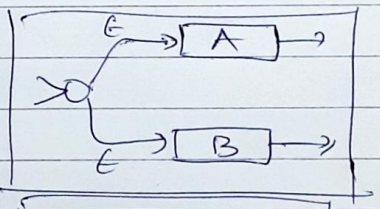
for base cases,

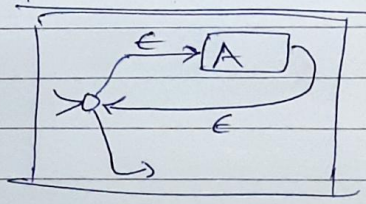
a: 

b: 



A.B: 

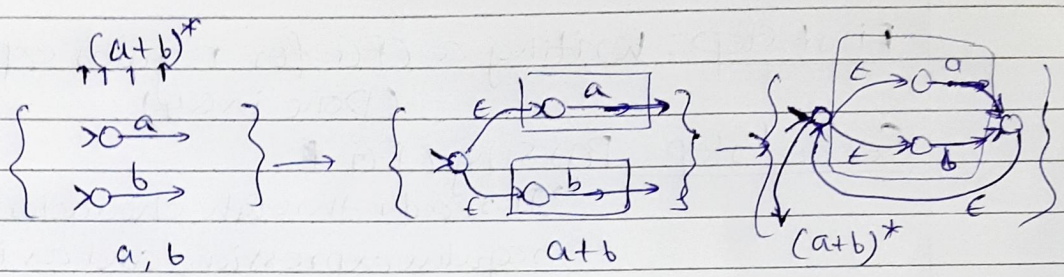
A+B: 

A^*: 

These modules will be "patched" together as the parsing occurs. The system is essentially a directed graph.

DOMS | Page No.
Date / /

Example :



Hence, the automaton (ε-NFA) is constructed.