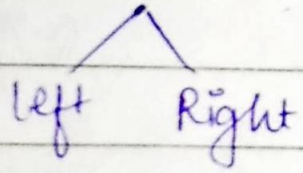
14 Oct '21

Parse Tree and PDA.

Parse Tree

↳ one way to show derivation.



1. leaf node → Terminal / ε
2. Internal node - Non terminal
3. Root node - String symbol
4. → (A) → $B_1 B_2 \ldots B_n$ left to Right

eg.

$E \rightarrow I$
$E \rightarrow E+E$
$E \rightarrow E*E$
$E \rightarrow (E)$

$I \rightarrow a$
$I \rightarrow b$
$I \rightarrow I0$
$I \rightarrow I1$
$I \rightarrow Ia$
$I \rightarrow Ib$.

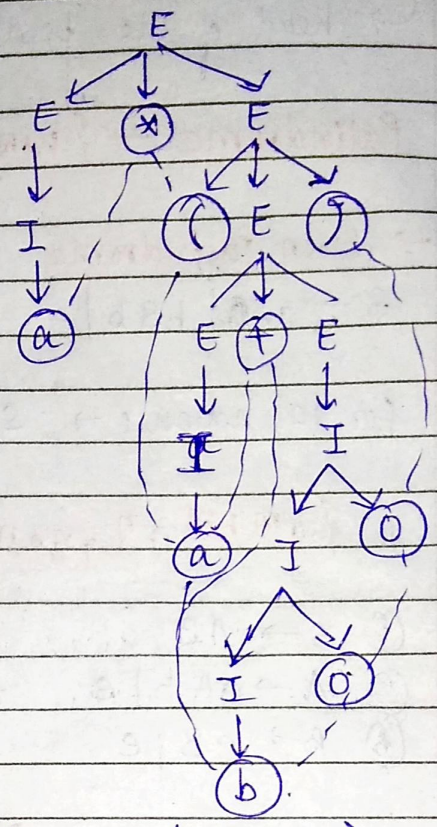
Yield - Concatenation of all leaf nodes from left to right.

Date / /
Page No.

for string a * (a + b00)

$E \rightarrow E * E$
$\rightarrow I * E$
$\rightarrow a * E$
$\rightarrow a * (E)$
$\rightarrow a * (E + E)$
$\rightarrow a * (I + E)$
$\rightarrow a * (a + E)$
$\rightarrow a * (a + I)$
$\rightarrow a * (a + I0)$
$\rightarrow a * (a + I00)$
$\rightarrow a * (a + b00)$

Parse Tree.



yield → a * (a + b00)

→ Application of CFG

1) Balanced Parenthesis –

$B \rightarrow BB \mid (B) \mid \epsilon$
$G = (\{B\}, \{(,)\}, P, B)$

2) if E else S
if E_1 else E_2 else S.

as defined previously.

$S \rightarrow E \mid SS \mid iS \mid iSe$
$G = (\{E, S\}, \{i, e\}, P, S)$

[i represent if
e represent else] → terminal

Date / /
Page No.

YACC (Yet Another compiler compiler) which is a parser generater and check acceptance of a string.
→ syntax analyzer.

- It takes input as token and Grammar (CFG) and we get a parse tree.

Q What's the problem YACC face?
→ Ambiguity - Ambiguous Grammar leads to conflicts

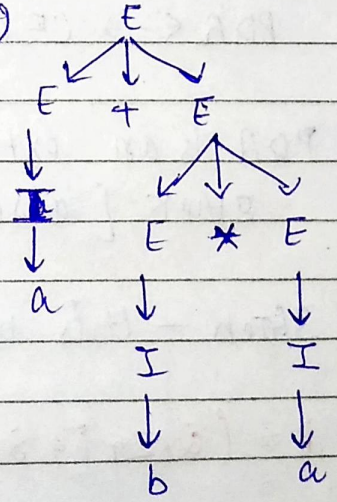
Conflicts

Shift Reduce (SR) [whether to push or pop]
Reduce Reduce (RR) [which Production Rule to choose]

eg → string → a+b*a.

① Ist parse tree.

E → E*E	Derivation – (left most)
E → E+E	E ⇒ E+E
E → I	⇒ I+E
I → a	⇒ a+E
I → b	⇒ a+E*E
I → Ia	⇒ a+I*E
I → Ib	⇒ a+b*E
I → I0	⇒ a+b*I
I → Ib.	⇒ a+b*a

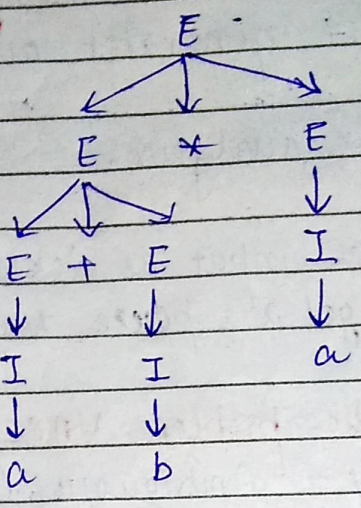


Date / /
Page No.

② 2nd Parse Tree

Derivation (left most)

$E \Rightarrow E * E$
$\Rightarrow E + E * E$
$\Rightarrow I + E * E$
$\Rightarrow a + E * E$
$\Rightarrow a + I * E$
$\Rightarrow a + b * E$
$\Rightarrow a + b * I$
$\Rightarrow a + b * a$



* If we get two different parse tree or two different left most or two different right most derivations, then the grammar given is ambiguous.

PDA (Push Down Automata)

↳ used to define CFG

PDA ⟷ CFG

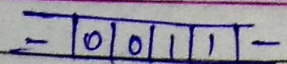
PDA is an extension of a NFA - with ε edge + stack [also called push down stack]

Stack – It is used to remember infinite no. of info

Def. $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

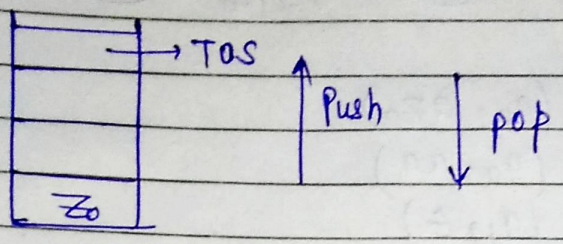
Q - finite set of states
Σ - finite set of symbols / Alphabet
Γ - stack Alphabet (Pushed into stack)
δ - Transition fⁿ $\delta : Q \times (\Sigma \cup \epsilon) \times \Gamma \longrightarrow Q \times \Gamma^*$

Date / /
Page No.



↑ → read head

finite control is used to decide where to go.



for $L = \{0^n 1^n, n > 0\}$

Push 0's into stack and for each 1, we will pop a symbol from stack.

q_0 - Initial state
Z_0 - Initial stack symbol
f - set of final states (0 or more)

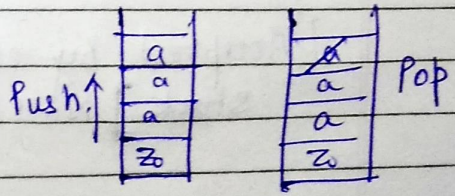
→ Two versions of PDA

① One that accepts by entering an accepting state like in fA

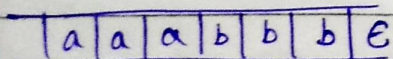
② Another that accepts by emptying the stack regardless of state it is in.

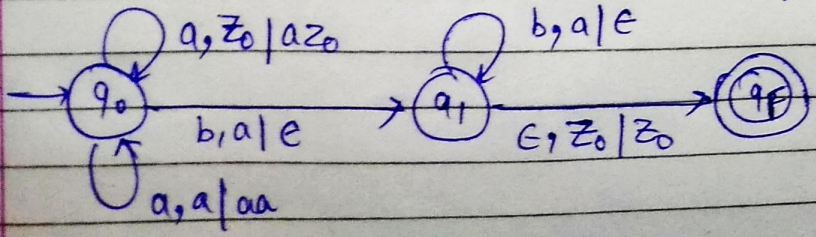
Q Construct a PDA that accepts the language $\{a^n b^n \mid n >= 1\}$ where $\Sigma = \{a, b\}$.

→ $L = \{ab, aabb, aaabbb, \ldots\}$



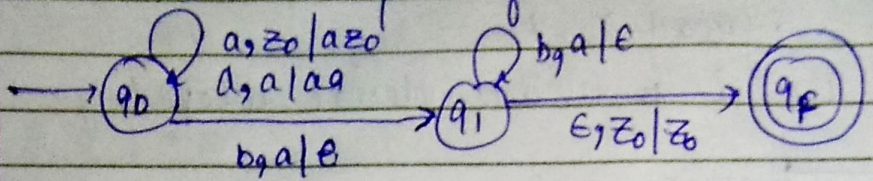
Initial →





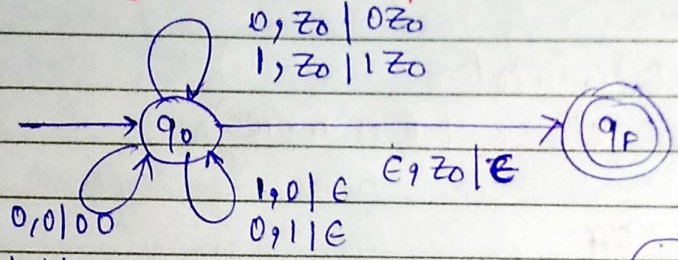
Date / /
Page No.

Instantaneous Description of PDA

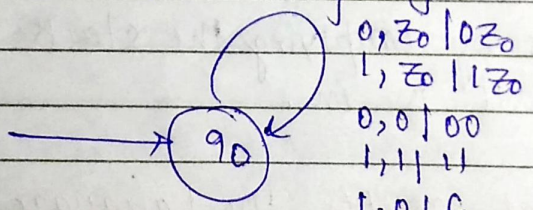


$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, a) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, Z_0) = (q_f, Z_0)$

Q2 $L = \{ w \in \{0,1\}^* \mid n_0(w) = n_1(w) \}$



After fusing edges.



Instantaneous Description

$\delta(q_0, 0, Z_0) = (q_0, 0Z_0)$
$\delta(q_0, 1, Z_0) = (q_0, 1Z_0)$
$\delta(q_0, 1, 0) = (q_0, \epsilon)$
$\delta(q_0, 0, 1) = (q_0, \epsilon)$
$\delta(q_0, 0, 0) = (q_0, 00)$
$\delta(q_0, 1, 1) = (q_0, 11)$
$\delta(q_0, \epsilon, Z_0) = (q_0, \epsilon)$

[Accepted by emptying stack] Null stack]