Parsing with PDA

Parsing Techniques

Top-Down Parsing

- Starts with the highest level of the parse tree and works down
- Prefers Left Most Derivation
- It's main decision is to select what production rule to use? in order to construct the string

Bottom Up Parsing

- Starts with the lowest level of the parse tree and works up
- Prefers Right Most derivation
- It's main decision is to select when to use a production rule? to reduce the string to get the starting symbol

If the PDA = (Q, ∑, S, δ, q_0,z0, F)

For top-down parsing, a PDA has the following four types of transitions –

- Push the start symbol ‘S’ into the stack
- Pop the non-terminal on the left hand side of the production at the top of the stack and push its right-hand side string
- If the top symbol of the stack matches with the input symbol being read, pop it
- If the input string is fully read then go to the final state, accept the string (F#Φ)

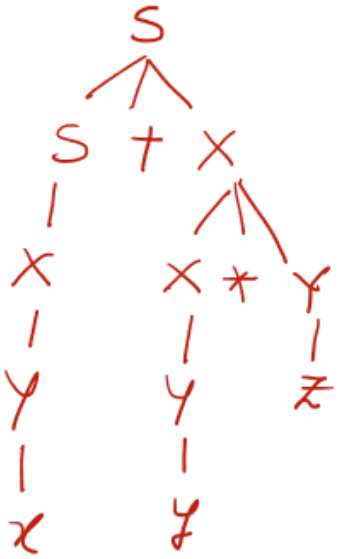
  OR
- If the input string is fully read and the stack is empty, accept the string (F=Φ)

Example1:

Design a top-down parser for the expression "x+y*z" for the grammar G with the following production rules –

P: S → S+X | X, X → X*Y | Y, Y → (S) | x |y |z

Solution:



(q0, x+y*z, z0) ⊢(q0, x+y*z, Sz0)

⊢(q0, x+y*z, S+Xz0)

⊢(q0, x+y*z, X+Xz0)

⊢(q0, x+y*z, Y+X z0)

⊢(q0, x+y*z, x+Xz0) ⊢(q0,+y*z, +Xz0)

⊢ (q0,y*z, Xz0)⊢(q0,y*z, X*Yz0)

⊢(q0,y*z, Y*Yz0) ⊢(q0,y*z, y*Yz0) ⊢(q0, *z,*Yz0)

⊢(q0, z, Yz0) ⊢(q0, z, zz0) ⊢(q0, ^, z0) ⊢(q0, ^, ^) →Accept

Example2:

Generate a Grammar fir following Language:

L1= {x ∈ {a,b}* | na(x) > nb(x)}

G(L1) = a|aS| bSS |SSb| SbS

Example2:

PDA M = (Q, ∑, Γ, δ, q_0,z0, F) ; ∑ ={a,b}

State	Input	Stack Symbol	Move(s)
q0	^	z0	{(q1, Sz0)}
q1	^	S	{(q1, a), (q1, aS), (q1, bSS), (q1, SSb), (q1, SbS)}
q1	a	a	{(q1, ^)}
q1	b	b	{(q1, ^)}
q1	^	z0	{(q2, z0)}

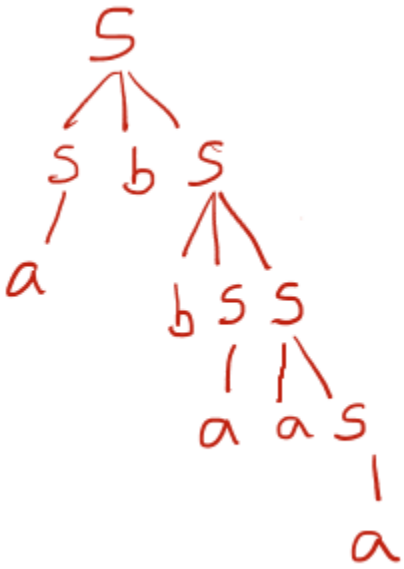
δ(q0,^,z0)= {(q1, Sz0)} and so on….

Q={q0, q1, q2}; Γ = {S, a, b, z0}; F={q2}

Example2:

Obtain Top-Down parsing for string w= ab^2a^3

$(q0, ab^2a^3, z0) \vdash (q1, ab^2a^3, Sz0)$
$\vdash (q1, ab^2a^3, SbSz0) \vdash (q1, ab^2a^3, abSz0)$
$\vdash (q1, b^2a^3, bSz0) \vdash (q1, ba^3, Sz0)$
$\vdash (q1, ba^3, bSSz0) \vdash (q1, a^3, SSz0)$
$\vdash (q1, a^3, aSz0) \vdash (q1, a^2, Sz0)$
$\vdash (q1, a^2, aSz0) \vdash (q1, a, Sz0) \vdash (q1, a, az0)$
$\vdash (q1, ^, z0) \vdash (q2, ^, z0)$ → Accept



If the PDA = (Q, ∑, S, δ, q_0,z0, F)

For bottom-up parsing, a PDA has the following four types of transitions –

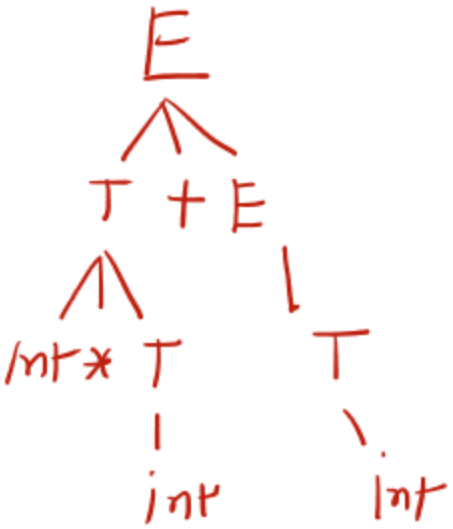
- Push the current input symbol into the stack
- Replace the right-hand side of a production at the top of the stack with its left-hand side
- If the top of the stack element matches with the current input symbol, pop it
- If the input string is fully read and only if the start symbol ‘S’ remains in the stack, pop it and go to the final state ‘F’ , accept the string (F#Φ)

  OR
- If the input string is fully read and the stack is empty (start symbol ‘S’ can be popped), accept the string (F=Φ)

Bottom-up parsing uses only two kinds of actions:
i) Shift ii) Reduce

e.g. E→ T + E | T
T→ int | int*T |(E)

The Example with Shift-Reduce Parsing



Stack-Top

|int * int + int	shift
int | * int + int	shift
int * | int + int	shift
int * int | + int	reduce T → int
int * T | + int	reduce T → int * T
T | + int	shift
T + | int	shift
T + int |	reduce T → int
T + T |	reduce E → T
T + E |	reduce E → T + E
E |	

Example 3:

Design a bottom–up parser for the expression "x+y*z" for the grammar G with the following production rules –

P: S → S+X | X, X → X*Y | Y, Y → (S) | x |y |z

Solution:

(q0, x+y*z, z0) ⊢ (q0, +y*z, xz0)

⊢ (q0, +y*z, Yz0)

⊢ (q0, +y*z, Xz0) ⊢ (q0, +y*z, Sz0)

⊢(q0, y*z, +Sz0) ⊢ (q0, *z, y+Sz0)

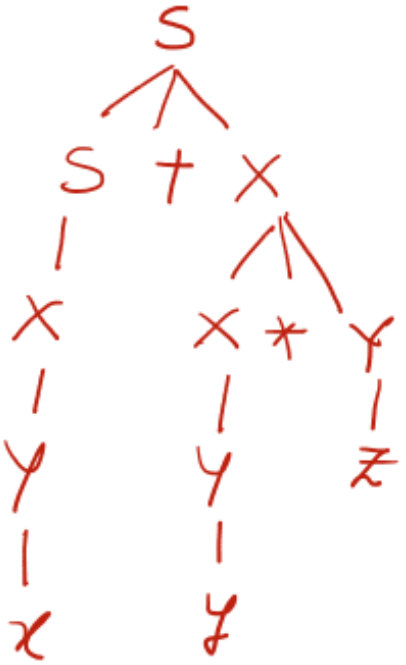
⊢ (q0, *z, Y+Sz0)

⊢ (q0, *z, X+Sz0) ⊢ (q0, z, *X+Sz0)

⊢ (q0, ^, z*X+Sz0) ⊢ (q0, ^, Y*X+Sz0)

⊢ (q0, ^, X+Sz0) ⊢ (q0, ^, Sz0)

⊢ (q0, ^, z0) (POP S) → Accept



Example 4:

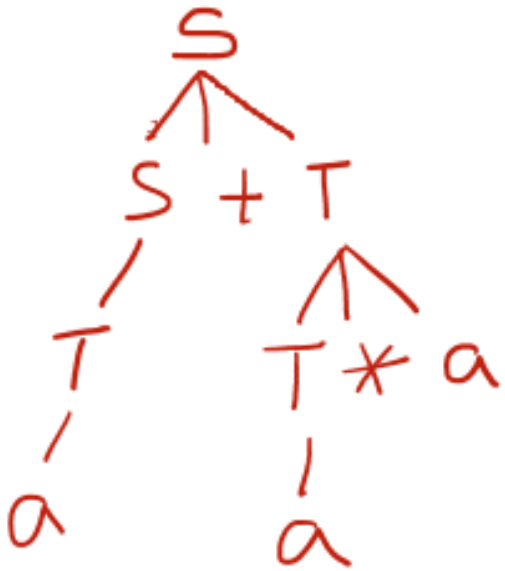
Consider following productions for grammar G:

S→ S+T | T

T→ T*a | a

-Demonstrate Bottom –up parsing for string ***w= a+a*a***

-Demonstrate stack manipulation



Move	Production	Stack Contents	Unread Input
-	-	z0	a+a*a
Shift	-	az0	+a*a
Reduce	T→a	Tz0	+a*a
Reduce	S→T	Sz0	+a*a
Shift	-	+Sz0	a*a
Shift	-	a+Sz0	*a
Reduce	T→a	T+Sz0	*a
Shift	-	*T+Sz0	a
Shift	-	a*T+Sz0	^
Reduce	T→T*a	T+Sz0	^
Reduce	S→S+T	Sz0	^
PoP S	-	z0	^ →Accept