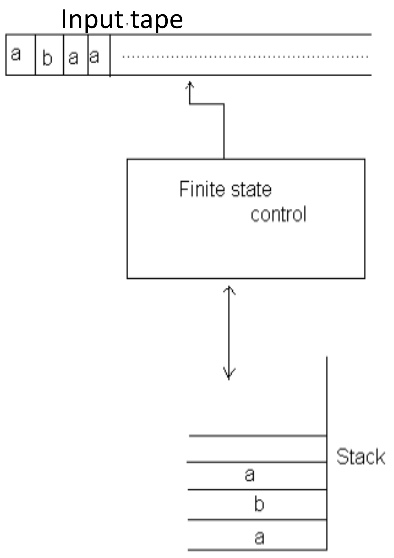
Pushdown Automata (PDA)

Informal Introduction

The context free languages have a type of automaton that defines them. This automaton is called "pushdown automaton" which can be thought as a Є-NFA with the addition of stack. The presence of a stack means that, the pushdown automata can remember infinity amount of information. However, the pushdown automaton can only access the information on its stack in a Last-in-first-out way.

We can define PDA informally as device shown in figure:



PDA is an abstract machine determined by following three things:

- Input table
- Finite stack control
- A stack

Each moves of the machine is determined by three things;

- The current state
- Next input symbol
- Symbol on the top of stack

The moves consist of;

- Changing state | staying on same state
- Replacing the stack top by string of zero or more symbols.

Popping the top symbol off the stack means replacing it by Є.
Pushing Y on the stack means replacing stack's top, say X, by YX.
of stack corresponds to stack's top. The single move of the machine contains only one stack operation either push or pop.

Formal Definition:

A PDA is defined by seven tuples (Q, Σ, Γ, δ, q0, z0, F)
Where,

Q- Finite set of states.
Σ- Finite set of input symbols | alphabets.
Γ- Finite set of stack alphabet
q0- Start state of PDA q0 ε Q
z0- Initial stack symbol; z0 ε Γ
F- Set of final states; F is subset of Q
δ- Transition function that maps Q × (Σ U {Є}) × Γ → Q × Γ*

as for finite automata, δ governs the behavior of PDA. Formally, δ takes as argument a triple δ (q,a,X) where

- q is a state.
- 'a' is either an input symbol in Σ or Є. (note Є does not belongs to Σ)
- X is a stack symbol.

The output of δ is a finite set of pairs (p, γ), where p is the new state and γ is the string on the stack after transition.

i.e the moves of PDA can be interpreted as;

$\delta(q, a, z) = \{(P_1, \gamma_1)\ (p_2, \gamma_2)\ \ldots\ldots\ldots\ldots\ldots(p_m, \gamma_m)\}$ here $q, p_i \varepsilon Q$, $a \varepsilon \Sigma \cup Є$ & $z \varepsilon \Gamma$, $\gamma_i \varepsilon \Gamma^*$

Graphical notation of PDA

We can use transition diagram to represent a PDA, where

- Any state is represented by a node in graph (diagram)
- Any arc labeled with "start" indicates the start state and doubly circled states are accepting / final states.
- The arc corresponds to transition of PDA as:

  Arc labeled as **<u>a, x | α</u>** means the transition δ(q, a, x) = (p,α) for arc from state p to q.

Example: A PDA accepting a string over {a, b} such that number of a's and b's are equal.
i.e. L={w | w ε {a, b}* and a's and b's are equal}.

The PDA that accepts the above language can be constructed using the idea that the PDA should push the input symbol if the top of the stack symbol is same as it otherwise Pop the stack.
For this, let us construct a PDA as;

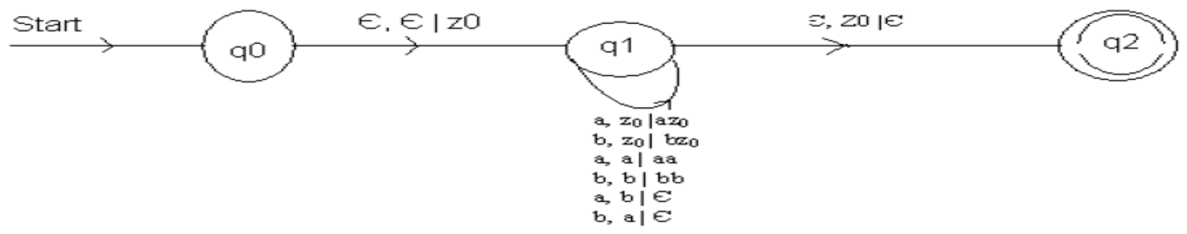
P = {Q, Σ, Γ, δ, q0, z0, F} be the PDA recognizing the given language. where, let us suppose

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

Now δ is defined as;

1. δ(q0, Є, Є) = (q1, z0) //initialize the stack to indicate the bottom of stack.
2. δ(q1, a, z0) = (q1, az0)
3. δ(q1, b, z0) = (q1, bz0)
4. δ(q1, a, a) = (q1, aa)
5. δ(q1, b, b) = (q1, bb)
6. δ(q1, a, b) = (q1, Є)
7. δ(q1, b, a) = (q1, Є)
8. δ(q1, Є, z0) = (q2, Є) // indicate the acceptance of string.

So the graphical notation for the PDA constructed in example 1 can be constructed as;



Let us trace for w= aabbbaab

The execution of PDA on the given string can be traced as shown on the table;

S.N	State	unread string	Stack	Transition Used
1	q_0	aabbbaab	Ɛ	--
2	q_1	aabbbaab	z_0	1
3	q_1	abbbaab	az_0	2
4	q_1	bbbaab	aaz_0	4
5	q_1	bbaab	az_0	7
6	q_1	baab	z_0	7
7	q_1	aab	bz_0	3
8	q_1	ab	z_0	6
9	q_1	b	az_0	2
10	q_1	Ɛ	z_0	7
11	q_2	Ɛ	Ɛ	8

Finally we have q2 state. Hence accepted.

Exercise

1) Trace as above for string aababb
2) Trace as above for string ababa (note this string will be not accepted.i.e. PDA will not reach to final state)

Instantaneous Description of PDA (ID)

Any configuration of a PDA can be described by a triplet (q, w, r).

Where,

q- is the state.

w- is the remaining input.

γ- is the stack contents.

Such a description by triple is called an instantaneous description of PDA (ID). For finite automaton, the $\hat{\delta}$ notation was sufficient to represent sequences of instantaneous description through which a finite automaton moved, since the ID for a finite automaton is just its state. However, for PDA's we need a notation that describes changes in the state, the input and stack. Thus, we adopt the notation of instantaneous description.

Let P = {Q, Σ, Γ, δ, q0, z0, F} be a PDA. Then, we define a relation ├, "yields as (q, aw, zα) ├ (p, w, βα) if δ(q, a, z) contains (p, β, and may be Є). This move reflects the idea that, by consuming "a" from the input, and replacing z on the top of stack by β, we can go from state q to state p.

Note that what remains on the input w, and what is below the top of stack, β, do not influence the action of the PDA.

For the PDA described earlier accepting language of equal a's and b's, [see example 1] the accepting sequence of ID's for string 1001 can be shown as;

(q0, 1001, Є) ├ (q1, 1001, z_0)
├ (q1, 001, 1z_0)
├ (q1, 01, z_0)
├ (q1, 1, 0z_0)
├ (q1, Є, z_0)
|- (q2, Є , Є) Accepted

Input String

POP if there is mismatch between Input & z0 content
ie. 0 and 1 mismatch here

Therefore (q0, 1001, z0) ├* (q2, Ɛ, Ɛ)

Language of a PDA

We can define acceptance of any string by a PDA in two ways;
1) Acceptance by final state:
- Given a PDA P, the language accepted by final state, L(P) is;
{w | (q, w, z0) ├* (P, Є, γ)} where P ε F and γ ε Γ*.

2) Acceptance by empty stack:
- Given a PDA P, the language accepted by empty stack, L(P), is
{w | (q, w, z0) ├* (P, Є, Є)} where P ε Q.

Deterministic Pushdown Automata (DPDA)

While the PDAs are by definition, allowed to be Non-deterministic, the deterministic PDA is quite important. In practice the parsers generally behave like Deterministic PDA, so the class of language accepted by these PDAs are practically important. DPDA are suitable for use in programming language.

A pushdown automata P = (Q, Σ, Γ, δ, q0, z0, F) is deterministic pushdown automata if there is no configuration for which P has a choice of more than one moves. i.e. P is deterministic if following two conditions are satisfied;

1) For any q ε Q, x ε Γ, If δ(q, a, x) ≠Φ for some a ε Σ then δ(q, Є, x) = Φ
i.e. if δ(q, a, x) is non-empty for some a, then δ(q, Є, x) must be empty.
2) For any q ε Q, a ε Σ U {Є} and x ε Γ, δ(q, a, x) has at most one element.

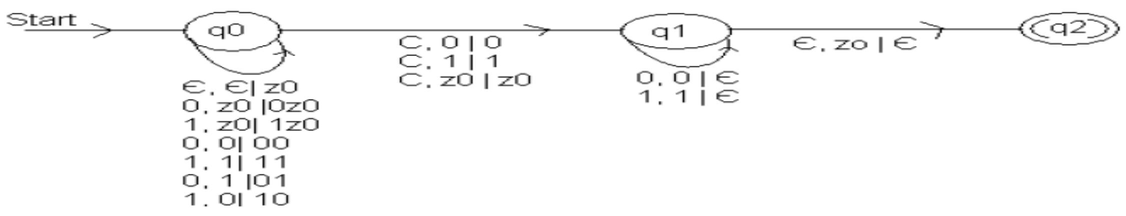
Example: A DPDA accepting language L = {w c w^R | w ε (0+1)*} is constructed as;

P = ({q0, q1, q2}, {0, 1}, {0, 1, z0}, δ, q0, z0, {q2}) where δ is defined as;

1. δ(q0, Є, Є) = (q0, z0) //initialize the stack
2. δ(q0, 0, z0) = (q0, 0z0)
3. δ(q0, 1, z0) = (q0, 1z0)
4. δ(q0, 0, 0) = (q0, 00)
5. δ(q0, 1, 1) = (q0, 11)
6. δ(q0, 0, 1) = (q0, 01)

7. δ(q0, 1, 0) = (q0, 10)
8. δ(q0, C, 0) = (q1, 0) //change the state when centre is reached.
9. δ(q0, C, 1) = (q1, 1) // change the state when centre is reached
10. δ(q0, C, z0) = (q1, z0)// change the state when centre is reached
11. δ(q1, 0, 0) = (q1, Є)
12. δ(q1, 1, 1) = (q1, Є)
13. δ(q1, Є, z0) = (q2, Є)

The graphical notation for the above PDA is:



Example: A DPDA accepting strings of balanced ordered parenthesis P;
Q = (q0, q1, q2), Σ = {{,}, (,), [,]}, Γ = Σ U {z0}, δ, q0, z0, {q2}

Where δ is defined as:
1. δ(q0, Є, Є) = (q0, z0)
2. δ(q0, {, z0) = (q1, {z0)
3. δ(q0, [, z0) = (q1, [z0)
4. δ(q0, (, z0) = (q1, (z0)
5. δ(q1, {, {) = (q1, {{)
6. δ(q1, [, [ ) = (q1, [[ )
7. δ(q1, (, ( ) = (q1, (( )
8. δ(q1, }, {) = (q1, Є)
9. δ(q1,], [ ) = (q1, Є)
10. δ(q1,), ( ) = (q1, Є)
11. δ(q1,), {) = (q1, ({)
12. δ(q1, {, [ ) = (q1, {[ )
13. δ(q1, Є, z0) = (q2, Є)

Evaluate or trace for the string [{ }]
(q0, [{ }], z0) ├ (q1, { }], [z0)
├ (q1,}], {[z0)
├ (q1,], [z0)
├ (q1, Є, z0)
├ (q2, Є, Є) Accepted

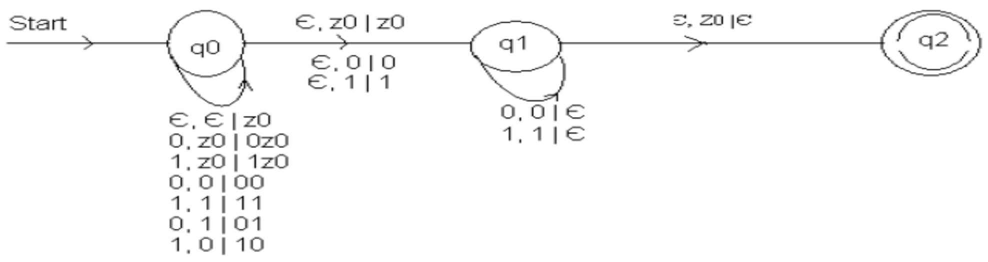
1.Try for the string [{( ) }].

2.try for the string [( ) ( )]

3.Try for the string {[}]. Note: this should be rejected.

Exercise

2. Construct a PDA accepting Language L = {ww^R | w ε (0+1)* and w^R is reversal of w}



Equivalence of CFG and PDA
Now we will show that the language defined by PDA's are exactly the context free language. The goal is to prove that the CFG and PDA accept the same class of languge. For this we show:

1) From CFG to PDA:
Given a CFG, G = (V, T, P and S), we can construct a push down automaton, M which accepts the language generated by the grammar G, i.e. L(M) = L (G).

The machine can be defined as;
M = ({q}, T, V U T, δ, q, S, Φ)
Where,
Q = {q} is only the state in the PDA.
Σ = T
Γ = V U T (i.e. PDA uses terminals and variables of G as stack symbols)
z0 = S (initial stack symbol is start symbol in grammar)
F = Φ
q0=q
And
δ can be defined as;

i. δ(q, Є, A) = {q, α) / A→ α is a production P of G.
ii. δ(q, a, a) = { (q, Є), for all a ε T}

Here all transition occurs on the only one state q.

Alternatively, we can define push down automata for the CFG with two states p & q, where p being start state. Here idea is that the stack symbol initially is supposed to be Є, and at first PDA starts with state P and reading Є, it inserts start symbol 'S' of CFG into stack and changes the state to q. Then all transitions occur in state q.
i.e. the PDA can be defined as;
M = ({p, q}, T, V U T, δ, p, {q}); stack top is Є.
Then δ can be defined as;

δ(p, Є, Є) = {(q, S) / S is start symbol of grammar G.
δ(q, Є, A) = {(q, α) / A→ α is a production P of G.
δ(q, a, a) = {(q, Є), for all a ε Σ.

Consider an example;

Let G = (V, T, P and S) where,
P is

E→T | E + T
T→ F | T*F
F→ a | (E)

We can define a PDA equivalent to this grammar as;
M = ({q0}, {a, *, +, (,)},{a, *, +, (,), E, T, F}, δ, q0, E, Φ}

Where δ can be defined as;

δ(q0, Є, E) = {(q0, T), (q0,E + T)}
δ(q0, Є, T) = {(q0, F), (q0,T*F)}
δ(q0, Є, F) = {(q0, a), (q0,(E))}
δ(q0, a, a) = {(q0, Є)}
δ(q0, *, *) = {(q0, Є)}
δ(q0, +, +) = {(q0, Є)}
δ(q0, (, ( ) = {(q0, Є)}
δ(q0,),)) = {(q0, Є)}

Now with this PDA, M, let us trace out acceptance of a + (a*a);

(q0, a + (a*a), E) ├ (q0, a + (a*a), E + T)
├ (q0, a + (a*a), T + T)
├ (q0, a + (a*a), F + T)
├ (q0, a + (a*a), a + T)
├ (q0, + (a*a), + T)
├ (q0, (a*a), T)
├ (q0, (a*a), F)
├ (q0, (a*a), (E))
├ (q0, a*a), E))
├ (q0, a*a), T))
├ (q0, a*a), T*F)
├ (q0, a*a), F*F)
├ (q0, a*a), a*F)
├ (q0, *a), *F)
├ (q0, a), F))

├ (q0, a), a))

├ (q0,),))

├ (q0, Є, Є) Accepted (acceptance by empty stack).

In CFG

E→ E + T
→ E + T
→ T + T
→F + T
→a + T
→a + F
→a + (E)
→a + (T*F)
→ a + (F*F)
→a + (a*F)
→a + (a*a)

Exercise
Convert the grammar defined by following production into PDA;
S→0S1 | A
A→ 1S0 | S | . Also we trace acceptance of aaabaaaaa.

3) From PDA to CFG:

Given a PDA M = (Q, Σ, Γ, δ, q0, z0, F); F = Φ, we can obtain an equivalent CFG, G = (V, T, P and S) which generates the same language as accepted by the PDA M.
The set of variables in the grammar consists of;
- The special symbol S, which is start symbol.
- All the symbols of the form [p x q]; p, ε Q and X ε Γ, where p and q are state in Q
i.e. V = {S} U {[p x q]}
The terminal in the grammar T = Σ

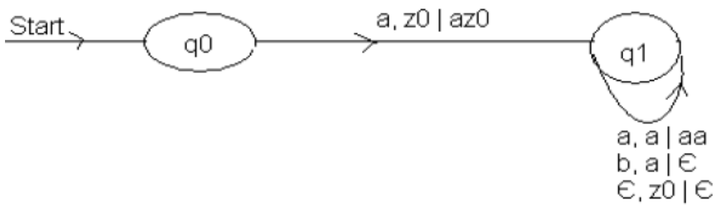
The production of G is as follows;
- For all states q ε Q, S→ [q0, z0, q] is a production of G.
- For any states p, q ε Q, x ε Γ and a ε Σ U {Є} if δ(q, a, x) = (p, Є) then [p × q]→ a
- For any states p, q ε Q, x ε Γ and a ε Σ U {Є}, if δ(q, a, x) = (p, Y1, Y2,……..Yk); where Y1, Y2, ……Yk ε Γ and k ≥ 0, Then for all lists of states P1, P2, …………..Pk, G has the production [p x q] → a [pY_1P_1][$P_1Y_2P_2$]…………..[$P_{k-1}Y_kP_k$].

This last production says that one way to pop x and go from stack q to state P_k is to read "a" (which may be Є), then use some input to pop Y1 off the stack while going from state P to P1, then read some more input that pops Y2 off the stack and goes from P1 t oP2 and so on………

For Example: Let a PDA that recognizes a language
L = {a^nb^n | n >0} be defined as;
1. δ(q0, a, z0) =(q1, az0)

2. δ(q0, a, a) =(q1, aa)
3. δ(q0, b, a) =(q1, Є)
4. δ(q0, Є, z0) =(q1, Є)

The graphical representation is



Let G = (V, T, P and S) be the equivalent CFG for the given PDA where,
V= {S} U {[p × q] / p, q ε Q, x ε Γ}
S= is the start state.
T= Σ
And P is defined by following production;
1. S→[q0 z0 q0] | [q0 z0 q1]
    i.e. S [q0 z0 r2]; for r2 in {q0, q1}
2. From the fact that δ(q0, a, z0) contains (q1, az0), we get production
    [q0 z0 r2]→ a[q1 ar1][r1 z0r2]; for r_i in{q0, q1}
3. From the fact that δ(q1, a, a) contains (q1, aa), we get productions
    [q1 ar2]→ a[q1 ar1][r1 a r2] for ri in {q0, q1}
4. From the fact that δ(q1, b, a) contains (q1, Є), we get
    [q1aq1]→b
5. From the fact that δ(q1, Є, z0) contains (q1, Є), we get
    [q1 z0 q1]→ Є

Now acceptance of aaabbb can be shown as;

S→ [q0 z0 r2]
→a [q1 a r1] [r1 z0 r2]
→ aa [q1 z0 r1] [r1 a r2] [r1 z0 r2]
→ aaa[q1 a r1] [r1 ar2] [r1 ar2] [r1 z0 r2]
→ aaab[r1 ar2] [r1 ar2] [r1 z0 r2]
→ aaabb[r1 ar2] [r1 z0 r2]
→ aaabbb[r1 z0 r2]
→ aaabbb Є
= aaabbb

Exercise

Convert the PDA P = ({p, q}), {0, 1}, {x, z0}, δ, q ,z0) to a CFG if δ is given by

- δ(q, 1, z0) = (q, xz0)
- δ(q, 1, x) = (q, xx)
- δ(q, 0, x) = (q, x)
- δ(q, Є, z0) = (q, Є)
- δ(q, 1, x) = (p, Є)
- δ(q, 0, z0) = (q, z0)