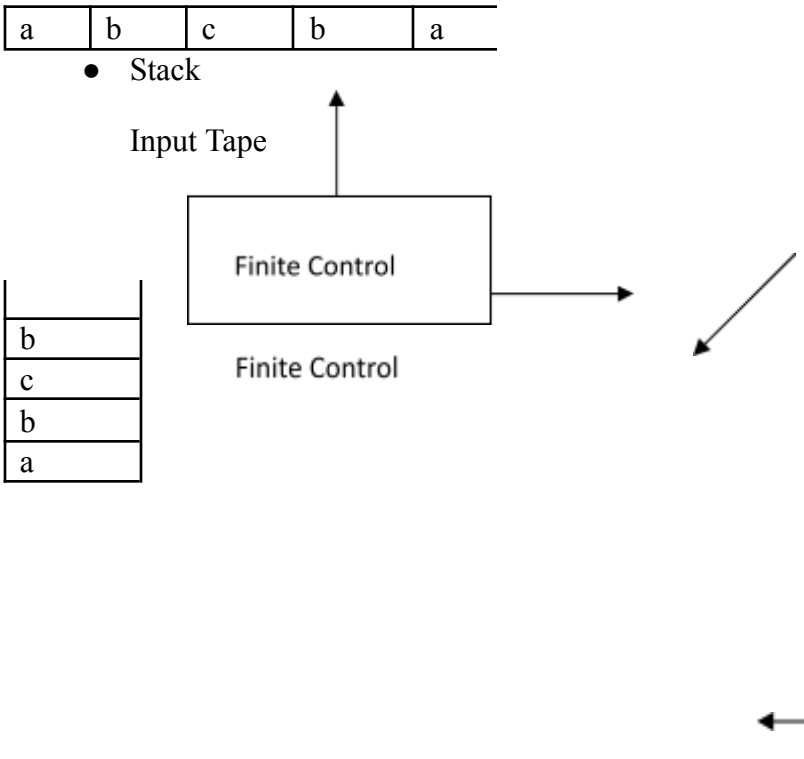
Push down Automata

Definition, Language of PDA, Equivalence of PDA‘s and; CFG‘s, Deterministic Pushdown Automata.

Pushdown Automata(PDA)

Consider the language L1 L1={ WCW^R : W ∈ $\sum^*$ or L2= { a^nb^n : n≥ 1 }, these languages are generated by CFG but can't be accepted by FA. So all CFG's are not accepted by FA. here we need to remember the strings before it goes to next part of the string and compare it with first half but FA can't do this because in FA don’t have capability of remember anything. PDA is essentially a FA with control of both an input tape and a stack to store what it has read. A stack is a FIFO. PDA can accept all CFGs. PDA have three things

- input tape
- Finite control



- Stack

Definition of PDA:

A pushdown automata is a system which is mathematically defined by sixtuple M= (K, ∑, Γ ,Δ , S , F)

where ,

K: Finite set of states

∑: is an alphabets (input symbols)

Γ : stack symbols

S ∈ K ,initial state

F⊆ K :set of final states

Δ : transition relation is a finite subset of Kx(∑ ⋃ {e} x Γ *) x (K x Γ *)

i.e

Transition :

if (p, a, β) , (q , γ) ∈ Δ , then M whenever it is in state p with ' β ' at the top of stack may read ' γ' from the input tape ,replace ' β' by ' γ' on the top of the stack and enter state q. such pair (p , a, β) , (q, γ) is called a transition of M.

Push/pop

Push : symbol input is to add to the top of stack . e.g (p, u, e) , (q, a) pushes a to the top of stack

Pop: symbol is to remove it from the top of the stack . e.g. (P, u, e) , (q , e) pops a.

Configuration of PDA

Configuration of a pushdown automaton is defined to be a member of K X ∑* x Γ * . The first component is the state of the machine (K), the second is the portion of the input yet to be read (∑*) , and the third is the contents of the pushdown store, read top-down (Γ *).

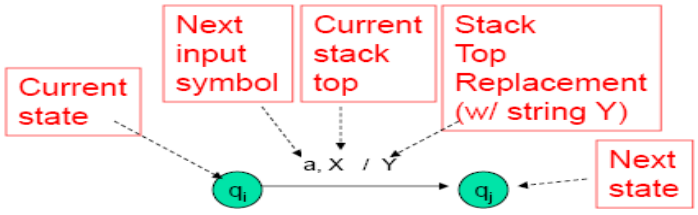
For example, if the configuration were (q, W, abc), the ' a ' would be on the top of the stack and the 'c' on the bottom. If (p, x, α) and (q, y, β) are configurations of M, we say that (p, x, α) yields in one step (q, y, β) . Thus a configuration of the machine is a triple (q, string, stack).

- When pushing symbol x, the configuration changes from (q, a.w, ε.t) to (qi, w, x.t).
- When popping symbol x, the configuration changes from(q, a.w, x.t) to (qi,w, t).
- When we don't wish the stack to change at all in a computation step, the machine moves from a configuration (q, a·w, ε·t) to (q_i,w, ε.t).
- Finally, on the occasion that we actually do wish to change the symbol c at the top of stack with symbol d, the configuration (q, a.w, c.t) changes to (qi,w, d.t).

An execution starts with the configuration (q_0, s, ε), i.e., the machine is in the start state, the input string s is on the tape, and the stack is empty. A successful execution is one which finishes in a configuration where s has been completely read, the final state of the machine is an accept state, and the stack is empty.

PDA as a state diagram:

$\delta(q_i, a, X)=\{(q_j, Y)\}$



Example 1

Design a pushdown automaton M to accept the language L = {WcW^R : w ∈{a, b } *}.

Solution : According to given language , ababcbaba ∈ L but abcab ϵ L. Let required PDA, M= (K, ∑, Γ ,Δ , S , F) where

K = {s , f}

∑= {a, b, c}

Γ= {a, b}

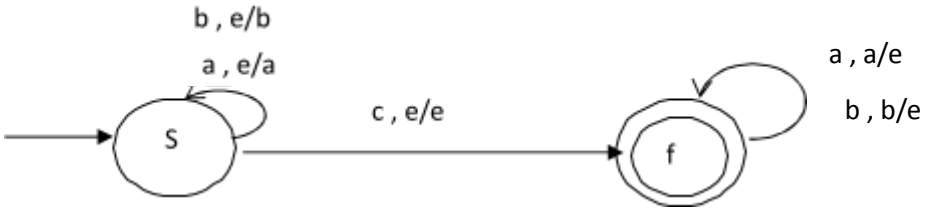
F = {f} and

Δ is given by following five transitions

1. ((s, a, e),(s, a) //push a
2. ((s, b, e),(s, b)) //push b
3. ((s, c, e), (f, e)) // change the state

4. ((f, a, a),(f, e)) // pop a on reading of input symbol a
5. ((f, b, b),(f, e)) // pop b on reading of input symbol b

State diagram



e.g Lets check for the string *abbcbba*

State	Unread input	stack content	Transition used
s	abbcbba	e	-
s	bbcbba	a	1
s	bcbba	ba	2
s	cbba	bba	2
f	bba	bba	3
f	ba	ba	5
f	a	a	5
f	e	e	4

- Here when machine sees a 'c' in input string , it switches from state s to f without operating on its stack.
- If the input symbol does not match the top stack symbol, no further operation is possible
- IF an automaton M reaches in configuration (f, e, e) , final state, end of input , empty stack , then the input was indeed of the form WcW^R

Example 2

Design a PDA M that accept a language given by L= { a^nb^n : $n \geq 1$ } .

Solution

Let required PDA, M= (K, ∑, Γ ,Δ , S , F) where

K = {s , q, f}

∑=

{a,b} Γ=

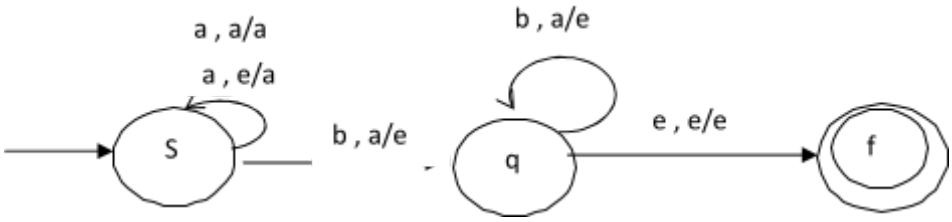
{a}

F = {f} and

Δ is given by following five transitions

1. ((s,a,e),(s,a) *//push a*
2. ((s,a,a),(s,a) *//push a*
3. ((s,b,a),(q,e)) *//pop a, when first b is encountered and also switch state from s to q*
4. ((q, b, a), (q, e)) *// continue pop a*
5. ((q,e,e),(f,e)) *// go to final state*

State diagram



e.g Lets check for the string *aabb*

State	Unread input	stack content	Transition used
s	aabb	e	-
s	abb	a	1
s	bb	aa	2
s	b	a	3
q	e	e	4
f	e	e	5

Example 3: Design a PDA that accept the following language L = {WW^R : w ∈{a, b } * }.
Solution: By analysis of language the machine must guess when it has reached the middle of the input string and change from state s to state f in a non deterministic fashion.

Let required PDA, M= (K, ∑, Γ ,Δ , S , F) where

K = {s , q, f}

∑= {a, b}

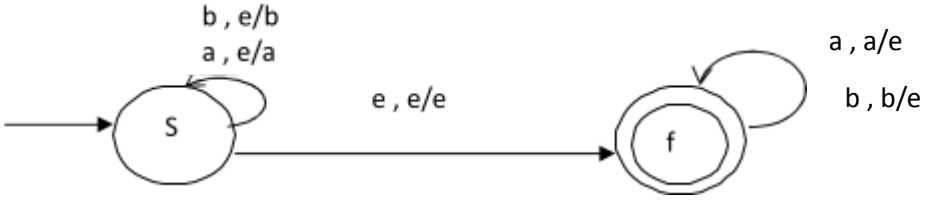
Γ= {a, b}

F = {f} and

Δ is given by following five transitions

1. ((s, a, e),(s, a) *//push a*
2. ((s, b, e),(s, b) *//push b*
3. ((s, e, e),(f, e)) *//change the state non deterministically (middle of the input string is guessed by machine M itself)*
4. ((f, a, a), (f, e)) *// pop a*
5. ((f, b, b),(f, e)) *// pop b*

Here, ((s, e, e), (f, e)) means change to state f without reading /consuming any symbol. Clearly whenever the M is in state S it can non- deterministically choose either to push the next input symbol on the stack or to switch to state f without consuming any input.

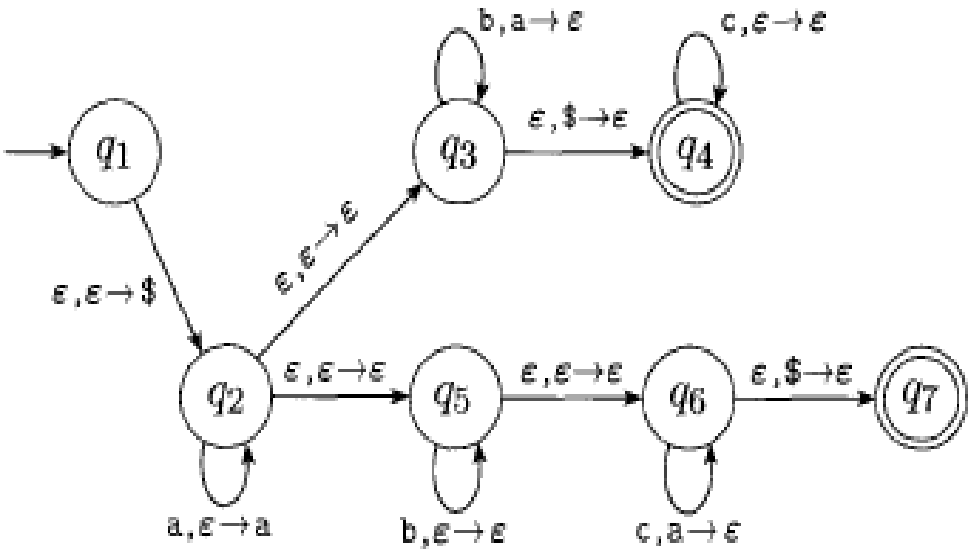


Example 4: Design a PDA that accepts the language
$L = \{a^i b^j c^k : i, j, k \geq O \text{ and } i = j \text{ or } i = k\}$.

Solution :

Let required PDA, $M = (K, \sum, \Gamma, \Delta, S, F)$ where
$K = \{q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$
$\sum = \{a, b, c\}$
$\Gamma = \{a, b\}$
$F = \{q_4, q_7\}$ and
Δ is given by following transition diagram

Informally the PDA for this language works by first reading and pushing the a's. When the a's are done the machine has all of them on the stack so that it can match them with either the b's or the c's. This is a bit tricky because the machine doesn't know in advance whether to match the a's with the b's or the c's. Non-determinism comes in handy here. Using its non-determinism, the PDA can guess whether to match the a's with the b's or with the c's, as shown in the following figure. Think of the machine as having two branches of its non-determinism, one for each possible guess. If either of them match, that branch accepts and the entire machine accepts. In fact we could show, though we do not do so, that non-determinism is essential for recognizing this language with a PDA.



Deterministic PDA (DPDA)

- A PDA is deterministic if there is at most one move for any input symbol, any top stack symbol and at any state
- A no move or move without advancing input string are possible in a deterministic PDA
- A move without advancing input string implies that no other move exists
- The PDA for WCW^R is deterministic
- The set of languages accepted by DPDA's is only a subset of language accepted by non deterministic PDA
- A regular languge is language accepted by a DPDA
- Not all language accepted by DPDA are regular
- All languages accepted by DPDA have an unambiguous VFG
- Not all unambiguous languages are accepted by DPDA.

The PDA is deterministic in the sense that at most one move is possible from any instantaneous description (ID). Formally, we say that a PDA $M = (Q, \sum, \Gamma, \delta, s, F)$, is said to be deterministic if it is an automation as defined in definition of PDA , subject to the restrictions that for each q in Q , b in T and a ϵ $(\sum \cup \{e\})$

1. whenever δ (q, a, b) contains at most one element
2. if δ (q, e, b) is not empty, then δ (q, c, b) must be empty for every c in $\sum$;

The first of these conditions simply requires that for any given input symbol and any stack top ,at most one move can be made. The second condition is that when a e-transitions is possible for some configurations no input consuming alternative is available.

Condition 1 prevents a choice of move for any (q, a, b) or (q, e, b). Note that unlike the finite automaton, a PDA is assumed to be nondeterministic unless we state otherwise.

Condition 2 prevents the possibility of a choice between a move independent of the input symbol (e-move) and a move involving an input symbol.

For finite automata, the deterministic and nondeterministic models were equivalent with respect to the languages accepted. The same is not true for PDA. In fact ww^R is accepted by a nondeterministic PDA, but not by any deterministic PDA.

The Languages of a PDA

We have assumed that a PDA accepts its input by consuming it and entering an accepting state. We call this approach acceptance by final state. We may also define for any PDA the language accepted by empty stack, that is, the set of strings that cause the PDA to empty it stack, starting from the initial ID.

These two methods are equivalent, in the sense that a language L has a PDA that accepts it by final state if and only if L has a PDA that accepts it by empty stack.

However for a given PDA P, the languages that P accepts by final state and by empty stack are usually different. We will show conversion of a PDA accepting L by final state into another PDA that accepts L by empty stack, and vice-versa.

Acceptance by Final State

Let M = (K,Σ, δ, s, Γ, F) be a PDA. Then $L_f(M)$, the language accepted by M by final state, is

$L_f(P) = \{ w \in \Sigma^* : (s, w, e) \overset{*}{\vdash} (f, e, \alpha))\}$ for some state $f \in F$ and any stack string α

That is, starting in the initial ID with w waiting on the input, M consumes w from the input and enters an accepting state. The content of the stack at that time is irrelevant.

Acceptance by Empty Stack

Let M = (K,Σ, δ, s, Γ, F) be a PDA. Then $L_e(M)$, the language accepted by M by empty stack , is

$L_e(M) = \{w \in \Sigma^* : (s, w, e) \overset{*}{\vdash} (q, e, e))\}$ for some state $q \in K$

That is, the set of input w that M can consumes and at the same time empty its stack. w from the input and enters an accepting state. The content of the stack at that time is irrelevant.

Two methods are equivalent in the sense that a language L has a PDA that accepts it by final state if and only if L has a PDA that accepts it by empty stack.

- **From Empty Stack PDA to Final State PDA**
- **From Final state PDA to Empty stack PDA**

Proof Read yourself

Equivalence of PDA and CFG

Theorem: The class of languages accepted by PDA is exactly the class of context free languages.

Lemma1: Each CFL is accepted by some PDA.

Lemma2: If a language is accepted by a PDA , it is a CFL.

Proof of Lemma 1:

Construction of PDA equivalent to a CFG.

Let G= (V, ∑, R, S) be a CFG, we must construct a PDA M such that L(M) = L(G) . The machine we construct has only two states p and q and remains permanently in state q after its first move. Also M uses V : set of terminals and ∑ set of non terminals as its stack alphabet.

Let M= (K, ∑, Γ ,Δ , S , F) where

K = {p, q}

∑= ∑

Γ= V ∪ ∑

S=p and

Δ (transition relation) is defined as

1. ((p, e ,e), (q, S)) as S is starting non terminals of CFG.
2. ((q, e, A), (q, x)) for each rule A→ x
3. ((q, a, a) ,(q, e)) for each a ∈∑

Example

Consider the grammar G = (V, ∑, R, S) with V = {S, a, b ,c}, ∑ = {a, b, c}, and R = {S → aSa, S → bSb, S → e), which generates the language $\{WcW^R : w \in \{a, b\}^*\}$.

Design a pushdown automaton.

Solution

The corresponding pushdown automaton, according to the construction above, is

M = (Q, ∑ , Γ, Δ, S, F) where

Q= {p, q}

∑ =∑ = {a, b, c }

Γ ={S, a, b ,c}

S= p

F={ q} and

Δ= {((p, e, e), (q, S)), T1

((q, e, S), (q, aSa)), T2

((q, e, S), (q, bSb)), T3

((q, e, S), (q, e)), T4
((q, a, a), (q, e)), T5
((q, b, b), (q, e)), T6
((q, c, c), (q, e))} T7

The string **abbcbba** is accepted by M through the following sequence of moves.

State	Unread input	stack content	Transition used
p	abbcbba	e	-
q	abbcbba	S	T1
q	abbcbba	aSa	T2
q	bbcbba	Sa	T5
q	bbcbba	bSba	T3
q	bcbba	Sba	T6
q	bcbba	bSbba	T3
q	cbba	Sbba	T6
q	cbba	cbba	T4
q	bba	bba	T7
q	ba	ba	T6
q	a	a	T6
q	e	e	T5