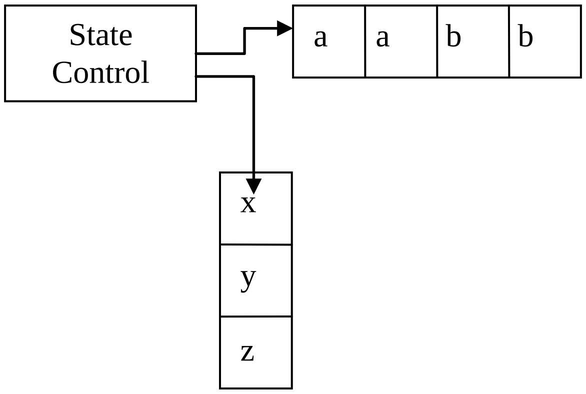
Pushdown Automata

Pushdown automata are like non-deterministic finite automata, but have an extra component called a *stack*.

A stack provides additional memory beyond the finite amount available. The stack allows pushdown automata to recognize some nonregular languages.



A pushdown automaton (PDA) can write symbol on the stack and read them back later.
-Writing a symbol "pushes down" all the other symbols on the stack. At any time, the symbol on the top can be read and removed. The remaining symbols then move back up.
-Writing a symbol on the stack is referred to as *pushing* the symbol,
-Removing the symbol is referred to as *popping* it.

A stack is a "last in, first out" storage device and read/write access can only be done at the top.

Formal definition of a pushdown automaton:

The formal definition of a pushdown automaton is similar to that of a finite automaton, except for the stack. The stack is a device containing symbols drawn from some alphabet. The machine may use different alphabets for its input and its stack. So we specify both an input alphabet Σ and a stack alphabet Γ.
The transition function δ which describes the automaton behavior is defined as $\delta \rightarrow Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P(Q \times \Gamma_\varepsilon)$

A pushdown automaton is a 6-tuple P(Q, $\sum$, Γ, δ, q_0, F), where Q, $\sum$, , Γ, F are all finite sets:

1. Q is the set of states.
2. $\sum$ is the set of input symbols
3. Γ is the stack alphabet
4. δ: Q x $\sum_\varepsilon$ x $\Gamma_\varepsilon \rightarrow$ *P*(Q x Γ_ε) is the transition function
5. $q_0 \in$ Q is the start state and
6. F $\subseteq$ Q is the set of accept states.

<u>Computation:</u>

A pushdown automaton M =(Q, $\sum$, Γ, δ, q_0, F) computes as follows:
It accepts input w if w can be written as w= $w_1 w_2 \dots w_m$, where each $w_i \in \sum_\varepsilon$ and sequences of states $r_0, r_1, \dots r_m \in$ Q and strings $s_0, s_1, \dots, s_m \in \Gamma^*$ exist that satisfy the following 3 conditions. The strings s_i represent the sequence of stack contents that M has on the accepting branch of the computation:

1. $r_0=q_0$ and $s_0=\varepsilon$. This condition signifies that M starts on the start state and that the stack is empty.
2. for i=0,…., m-1, we have $(r_{i+1},b) \in \delta(r_i, w_{i+1},a)$, where s_i=at and s_{i+1} = bt for some a, b $\in \Gamma_\varepsilon$ and t $\in \Gamma^*$. This condition states that M moves properly according to the state stack, and next input symbol.
3. $r_m \in$F. This condition states that an accept state occurs at the input end.

The following is the formal description of the PDA that recognizes the language $\{0^n1^n \mid n \geq 0\}$.
Let M_1= (Q, $\sum$, Γ, δ, q1, F) where
Q={q1, q2, q3, q4},
$\sum$ = {0, 1},
Γ = {0, \$},
F= {q1, q4}, and
δ is given by the following table, wherein blank entries signify
δ can also be defined as such:

((q1, 0, ε), (q2, 0\$))
((q2, 0, 0\$), (q2, 0))
((q2, 0, 0), (q2, 0))
((q2, 1, 0), (q3, ε))

((q3, 1, 0), (q3, ε))
((q3, ε, $), (q4, ε))

Input	**0**			**1**			**ε**		
Stack	**0**	**$**	**ε**	**0**	**$**	**ε**	**0**	**$**	**ε**
q1			**(q2,0$)**						
q2	**(q2,0)**			**(q3,ε)**					
q3				**(q3,ε)**				**(q4,ε)**	
q4									

Consider the following language defined by the expression:
$\{a^n b^n c^{2n} | \ n \geq 1\}$
(Assignment: Using Jflap design the corresponding push down automaton)

Theorem:

A language is context free if and only if some pushdown automaton recognizes it.

Lemma:
If a language is context free, then some pushdown automaton recognizes it.

Let A be a Context free language; then there is a CFG G that generates it.

PDA as a program:
We draw the program of a PDA as a flowchart. There are 5 components to this:

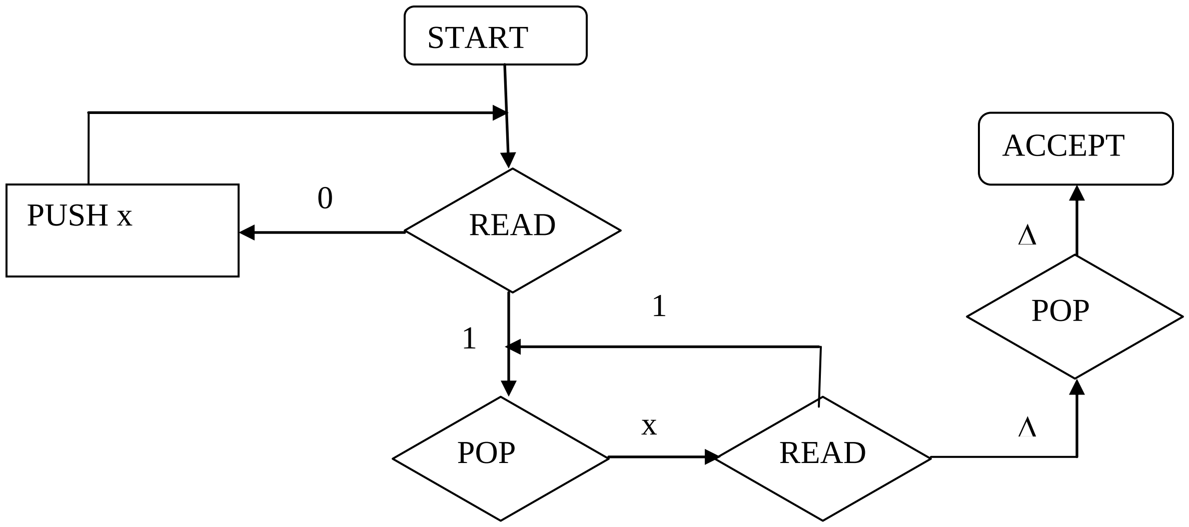
- A single start state
- A single halt-and-accept state
- A reader box: this reads one symbol from the input and, depending on the symbol read, goes to a new state

- a pop box: this pops one symbol from the stack and, depending on the symbol , goes to a new state.
- A push box: this adds a specific symbol to the stack.

The flowchart has no reject state, if the machine gets to a state and there are no legal continuation, then we assume that *the machine halts and rejects the input*.

We also need a special symbol, we will use Δ to indicate the end of the input string. We will also use Δ to indicate the result of popping when the stack is empty- we can think of this as the stack starting with the special symbol Δ on it.

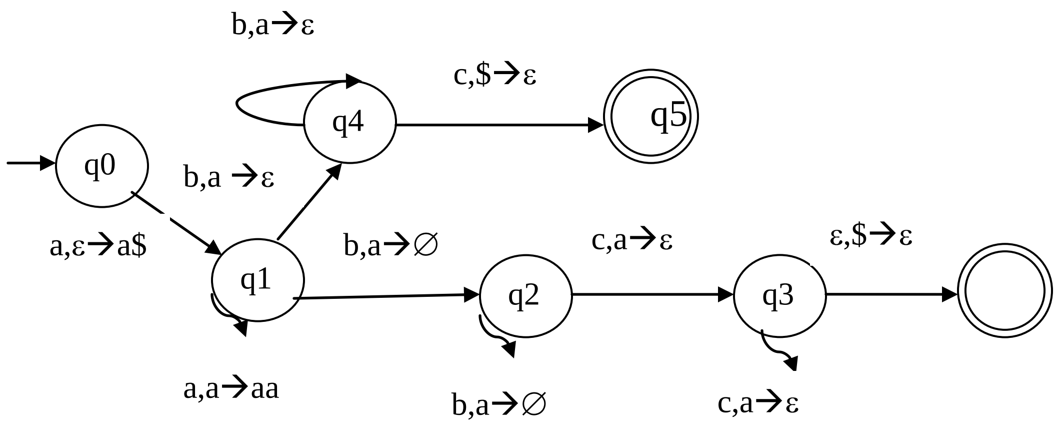
The language $\{0^n 1^n : n > 0\}$



Balanced brackets:
We consider strings consisting entirely of left and right brackets. Such strings are called balanced if (a) reading from left to right, the number of left brackets is always at least the number of right brackets; and (b) the total number of left brackets equals the total number of right brackets.
A grammar for such strings is:
$S \rightarrow (S) | SS | \varepsilon$
Draw the flowchart for this grammar.

Nondeterministic Push down automaton:

Find a pushdown automaton that recognizes the language
{ $a^i b^j c^k$| and i=j or i=k}
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 in the stack so it can match them with either the b's or the c's. This is somewhat tricky because the machine does not know in advance whether it should match the a's with the b's or with the c's.



Input	**a**			**b**			**c**			**ε**
Stack	**a**	**$**	**ε**	**a**	**$**	**ε**	**a**	**$**	**ε**	**$**
q0			**(q1,a$)**							
q1	**q1,aa)**			**(q2,∅)**						
q2				**q2,∅)**			**(q3,ε)**			
q3						**(q4,ε)**	**(q3,ε)**			**(q3,ε)**
q4						**q4,ε)**	**(q5, ε)**			
q5							**(q5,ε)**			**(q6,ε)**

Informally, this PDA works by reading an a and pushing it or reading a b and pushing it.

Given that an NPDA is in state p, with symbol A on top of the stack and the input is symbol a, it may do one of the following:

- if δ(p, a,A) then it :
    - "pops" A from the stack

- “pushes” word A1A2…An onto the stack by starting with symbol An, and ending with symbol A1,
- “consumes” a by moving to the next symbol to the right of a
- enters state q

if δ(p, ε,A)= ∅ then M does nothing.

- if $\delta(p, \varepsilon, A) \neq \varnothing$, then, without reading a , it
  1. ``pops'' A off the stack,
  2. ``pushes'' word $A_1...A_n$ onto the stack, and
  3. enters state q,

as long as $(q,A_1...A_n) \in \delta(p,\varepsilon, A)$;

if $\delta(p,\varepsilon, A)=\varnothing$, then M does nothing.

If $(q,\varepsilon) \in \delta(p,a,A)$, then A gets popped off, and nothing gets pushed onto the stack.

Modes of Acceptance

A PDA is a language acceptor. We describe how words are accepted by a PDA M . First, we start with configurations.
A configuration of M is an element of $Q \times \Sigma^* \times \Gamma^*$. For any word u , the configuration $(q_0,u,\$)$ is called the start configuration of u .
A binary relation $\vdash$on the set of configurations is defined as follows: if (p,u,α) and (q,v,β) are configurations of M , then $(p,u,\alpha) \vdash (q,v,\beta)$ provided that $\alpha=A\gamma$ and $\beta=B_1...B_n\gamma$, for some $A,B_1,...,B_n \in \Gamma$, and

- either $u=av$, and $(q,B_1...B_n) \in \delta(p,a,A)$,
- or $u=v$, and $(q,B_1...B_n) \in \delta(p,\varepsilon, A)$.

Now, take the reflexive transitive closure $\vdash^*$of $\vdash$. When $(p,u,\alpha) \vdash^* (q,v,\beta)$, we say that v is derivable from u . A word $u \in \Sigma^*$ is said to be

- accepted on final state by M if $(q_0,u,\$)\vdash^*(q,\varepsilon,\alpha)$ for some final state $q\in F$,
- accepted on empty stack by M if $(q_0,u,\$)\vdash^*(q,\varepsilon,\varepsilon)$,
- accepted on final state and empty stack by M if $(q_0,u,\$)\vdash^*(q,\varepsilon,\varepsilon)$ for some $q\in F$.

Procedure on how to convert G into an equivalent PDA, called P:

1. Place the marker symbol $ and the start variable on the stack
2. Repeat the following steps forever:

a. if the top of stack is a variable (non-terminal) symbol A, non deterministically select one of the rules for A and substitute A by the string on the right-hand side of the rule.

b. If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a. If they match, pop a and repeat step 2. If they do not match, reject on this branch of the non-determinism.

c. if the top of stack is the symbol $, enter the accept state. Doing so accepts the input if it has all been read.

Non-Context Free languages:

The pumping lemma for context free languages states that every context free language has a special value called the pumping length such that all longer strings in the language can be "pumped". This means that the string can be divided into five parts so that the second and the fourth parts may be repeated together.

Theorem:
Pumping lemma for context-free languages: If A is a context free language, then there is a number p(the pumping length) where, if s is any string in A of length at least p, then s may be divided into five pieces s= uvxyz satisfying the conditions:

1. for each $i \geq 0$, $uv^ixy^iz \in A$
2. $|vy| > 0$ and
3. $|vxy| \leq p$

When s is being divided into uvxyz, condition 2 says that either v or y is not the empty string.