Pushdown Automata

Pushdown Automata

A **pushdown automaton** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where Q, Σ, Γ, F are finite sets, and

1. Q is the **set of states**,
2. Σ is the **input alphabet**,
3. Γ is the **stack alphabet**,
4. $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma_\epsilon)$ is the **transition function**,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the **set of accept states**.

Pushdown Automata (cont'd)

At each computational step, a PDA does the following:

1. nondeterministically **decides whether to read the next input symbol and the current stack symbol**; for input symbol w_i and stack symbol s_j the choices are $(w_i, s_j), (\epsilon, s_j), (w_i, \epsilon), (\epsilon, \epsilon)$; (if the input or stack are empty, there are fewer than 4 choices);
2. if the choice is (u, v), then if $u \neq \epsilon$ it **reads (moves past)** u and if $v \neq \epsilon$ it **pops** v **from the stack**; and
3. it nondeterministically **selects the next state and a symbol to be put on the stack** according to the transition function.

It halts when **either (i)** there is **no next move**, **or (ii) no input symbols are left** and **the current state is an accept state.**

Pushdown Automata (cont'd)

$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ **accepts** a word $w \in \Sigma^*$ if w can be represented as $w_1 \cdots w_m$ with $w_1, \ldots, w_m \in \Sigma_\epsilon$ and there exist $r_0, \ldots, r_m \in Q$ and $s_0, \ldots, s_m \in \Gamma^*$ satisfying the following conditions:

1. $r_0 = q_0$ and $s_0 = \epsilon$.
2. For every i, $1 \le i \le m$, $(r_i, b) \in \delta(r_{i-1}, w_i, a)$, where $s_{i-1} = at$ and $s_i = bt$ for some $a, b \in \Gamma_\epsilon$ and $t \in \Gamma^*$.
3. $r_m \in F$.

Here the stack is read from top to bottom (or left to right).

PDA, Example 1

$L = \{0^n 1^n \mid n \geq 0\}$.

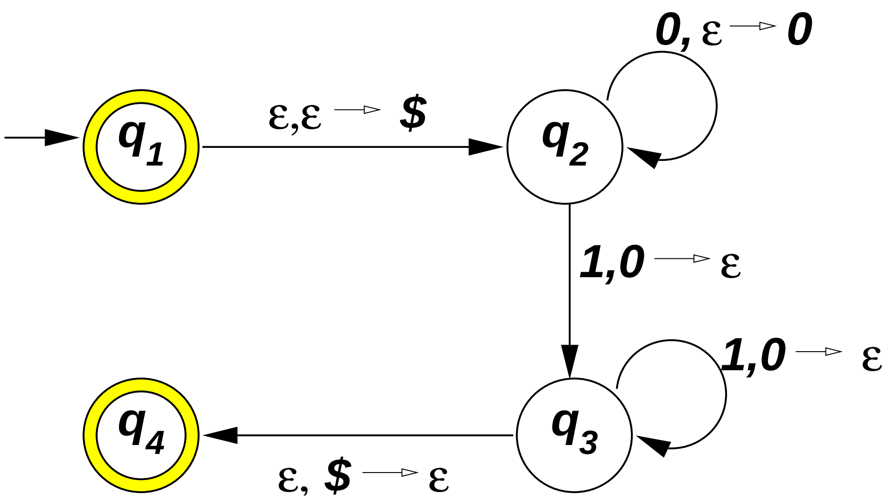
Design Idea

- Use a special symbol $\$$ to **mark the bottom of the stack.** (Using ϵ wouldn't work because ϵ is *always* "there".)
- First put onto the stack all the **0s preceding the 1s.**
- Then try to **match the stacked 0s with the 1s.**
- The input string is in L if, and only if, **the bottom $\$$ is the top symbol when all the input symbols have been read.**

Example 1 (cont'd)

$\Gamma = \{0, 1, \$\}$, $Q = \{q_1, q_2, q_3, q_4\}$, $F = \{q_1, q_4\}$, q_1 is the initial state

We use q_1 to place the $ on the stack, q_2 to read 0s and push them onto the stack (and react to the first 1), and q_3 to read 1s and pop 0s from the stack (and react to the $ by going to accept state q_4). As a diagram:



Example 1 (cont'd)

As a transition table (where blank means empty set):

Input:	0			1			ϵ		
Stack:	0	\$	ϵ	0	\$	ϵ	0	\$	ϵ
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \epsilon)\}$					
q_3				$\{(q_3, \epsilon)\}$				$\{(q_4, \epsilon)\}$	

Let $(q, u, v), q \in Q, u \in \Sigma^*, v \in \Gamma^*$ denote the configuration in which **the state is q, the remaining input is u, and the stack word is v.**

000111 is accepted by a path: $(q_1, 000111, \epsilon) \Rightarrow (q_2, 000111, \$) \Rightarrow (q_2, 00111, 0\$) \Rightarrow (q_2, 0111, 00\$) \Rightarrow (q_2, 111, 000\$) \Rightarrow (q_3, 11, 00\$) \Rightarrow (q_3, 1, 0\$) \Rightarrow (q_3, \epsilon, \$) \Rightarrow (q_4, \epsilon, \epsilon)$.

PDA, Example 2

$L = \{u \in \{0,1\}^* \mid u$ has an equal number of 0s and 1s $\}$.

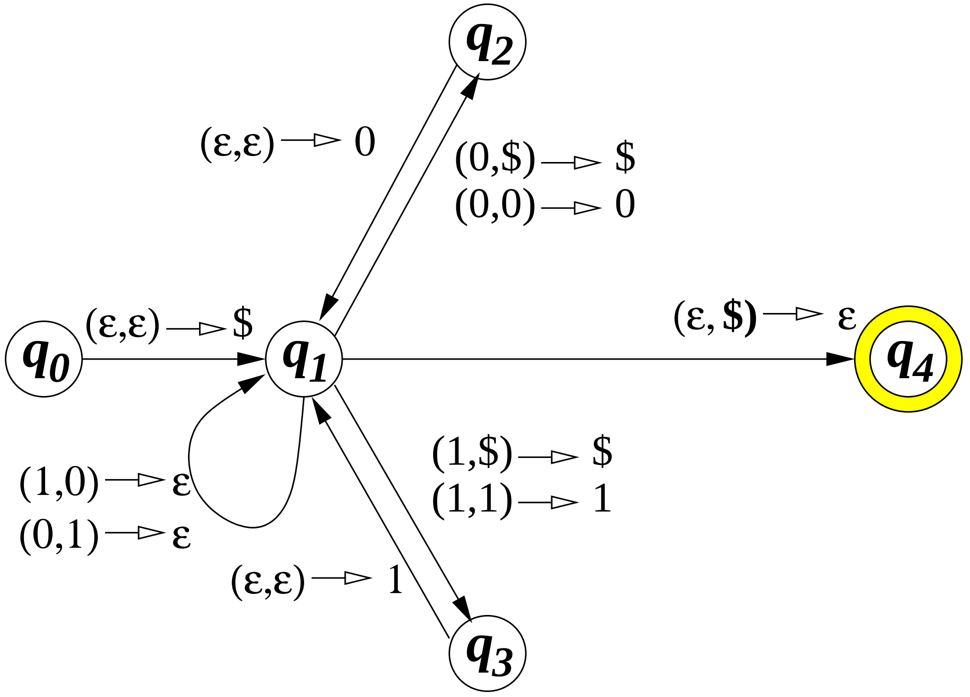
Design Idea

- We note that for any initial portion of a binary string, if that portion is "unbalanced", there's either an excess of 0s or an excess of 1s. The idea is to always save this excess on the stack. The stack will never have both 0s and 1s on it.
- So when an input symbol is processed that is the same as the top stack symbol (or the top stack sybol is the bottom-of-stack marker), this symbol adds to the excess and should be pushed onto the stack.

Note: Pushing an excess symbol onto the stack takes *two* steps, since "looking at" the top of the stack consumes the top symbol, and we then have to put it back on.

- If an input symbol is processed that is complementary to the top stack symbol, these two symbols "cancel", i.e., we should pop the stack.
- If we run out of input symbols with the stack empty, we clearly have a string with an equal number of 0s and 1s.

PDA, Example 2 (cont'd)



Example: $(q_0, 011100, \epsilon) \Rightarrow (q_1, 011100, \$) \Rightarrow (q_2, 11100, \$) \Rightarrow (q_1, 11100, 0\$) \Rightarrow (q_1, 1100, \$) \Rightarrow (q_3, 100, \$) \Rightarrow (q_1, 100, 1\$) \Rightarrow (q_3, 00, 1\$) \Rightarrow (q_1, 00, 11\$) \Rightarrow (q_1, 0, 1\$) \Rightarrow (q_1, \epsilon, \$) \Rightarrow (q_4, \epsilon, \epsilon)$

PDA, Example 2 (concluded)

$L = \{u \in \{0,1\}^* \mid u$ has an equal number of 0s and 1s $\}$.
$Q = \{q_0, q_1, q_2, q_3, q_4\}$, $F = \{q_4\}$, $\Gamma = \{0, 1, \$\}$, q_0 is the initial state

	0			1			ϵ	
	0	1	$\$$	0	1	$\$$	ϵ	$\$$
q_0							$(q_1, \$)$	
q_1	$(q_2, 0)$	(q_1, ϵ)	$(q_2, \$)$	(q_1, ϵ)	$(q_3, 1)$	$(q_3, \$)$		(q_4, ϵ)
q_2							$(q_1, 0)$	
q_3							$(q_1, 1)$	

Here $\{$ and $\}$ are omitted. State q_0 is used to place \$ on the stack, q_1 is used to pop stack symbols "matching" a complementary input symbol (and to recognize when we're done), q_2 is used to push unmatched 0s onto the stack, and q_3 to push unmatched 1's onto the stack.

PDAs Recognize CFLs

Theorem. *Each context-free language is recognized by a PDA.*

Proof Let L be produced by a Chomsky normal form grammar $G = (V, \Sigma, R, S)$.

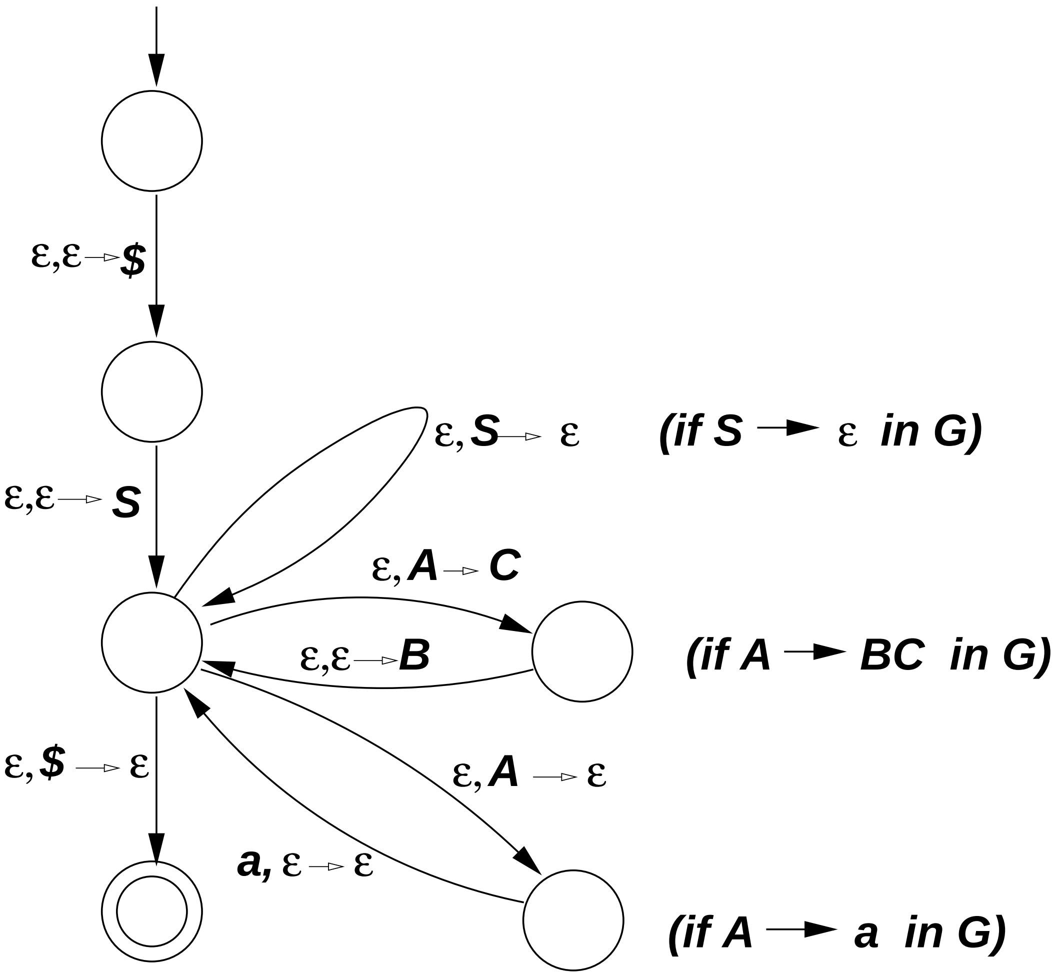
IDEA Construct a PDA that performs **leftmost derivations**. A step of a derivation using $A \rightarrow BC$ is implemented by **popping the left-hand side and pushing the right-hand side of the rule onto the stack**. A step using $A \rightarrow a$ is implemented by **popping the left-hand side and comparing the right-hand side to the input symbol (expending that symbol)**.

PDA Recognize CFLs (cont'd)

N's input alphabet = Σ, stack alphabet = $V \cup \{\$\}$. It has 1 accept state.

N pushes a \$ and then an S on the stack and repeats the following until it halts:

1. N pops a stack symbol A. If $A =$ \$, N **enters the accept state and halts**.
2. N **nondeterministically chooses a production rule $A \rightarrow w$.**
   - If $w = \epsilon$, N takes no action (back to step 1).
   - If $w = BC$ for some variables B, C, then N **pushes a C then a B onto the stack.**
   - If $w = a$, N reads the next input symbol b; **if $a \neq b$ then N halts.**


ε,ε → $
ε,ε → S
ε,S → ε
(if S → ε in G)
ε,A → C
ε,ε → B
(if A → BC in G)
ε,$ → ε
ε,A → ε
a, ε → ε
(if A → a in G)

CFLs Capture PDA

Theorem. *Every language recognized by PDA is context free.*

Proof Let L be recognized by a PDA M. We can assume

(*) **M has a unique final state and, when it enters the state, the stack is empty.**

(**) **In a single move M either pops or pushes, but not both.**

Idea for ensuring (*): Modify M to create a new PDA N, which has

- a new stack symbol $\perp$ (to mark bottom of stack),
- a new initial state I (for putting $\perp$ on the stack),
- a clean-up state C (for clearing the stack),
- a new (unique) accepting state A.

CFLs Capture PDA (cont'd)

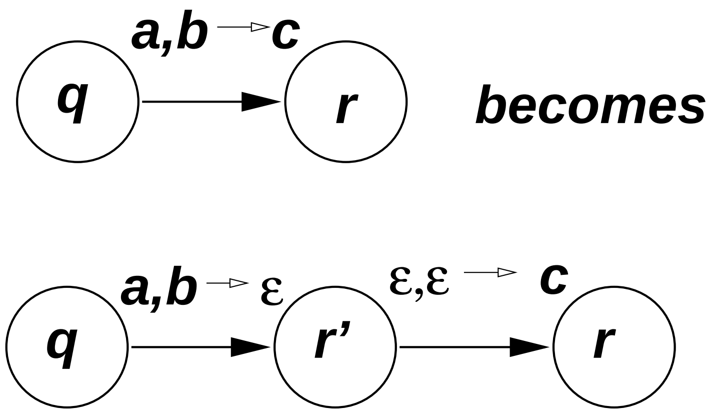
- The only permissible action in I is to **put one $\perp$ on the stack without reading an input symbol and go to the old initial state.**
- In each state in Q the action of N is the same except that **there is an (ϵ, ϵ)-transition from each former accepting state to C.**
- The goal in state C is to **remove stack symbols one after another** and to **enter A upon observing a $\perp$.**

CFLs Capture PDA (cont'd)

As to (**), **divide each state involving push and pop into two states, one for push only and the other for pop only:**

$(r, c) \in \delta(q, a, b)$ becomes a pair of moves:

(r', ϵ) in $\delta(q, a, b)$ followed by (r, c) in $\delta(r', \epsilon, \epsilon)$



CFLs Capture PDA (cont'd)

Now suppose M satisfies (*) and (**).

Construct a CFG (V, Σ, P, S): $V = \{A_{pq} \mid p, q \in Q\}$ and $S = A_{q_0 f}$, where q_0 is the initial state of M and f is the unique final state of M.

Key idea: For any states p, q, A_{pq} is a variable generating **the strings that can take M from p to q while "preserving the stack"**. Then by taking p as the start state and q as the accept state, we have a variable that generates **precisely the strings accepted by M**, and thus can serve as start symbol.

CFLs Capture PDA (cont'd)

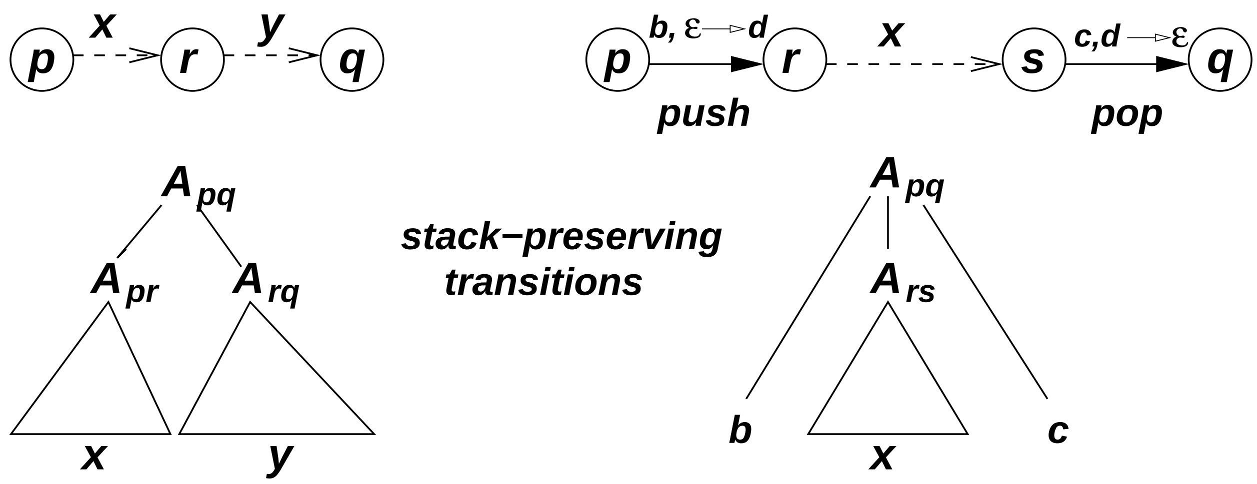
More precisely, the variable A_{pq} corresponds to the set of all strings expended by M under the following conditions:

- **M starts with p and ends with state q.**
- For some $k \geq 0$, the stack height is:
  - **always at least k** and
  - **precisely k at the start as well as at the end**.

Production rules:

- For every $p \in Q$, $A_{pp} \rightarrow \epsilon$.
- For every $p, q, r \in Q$, $A_{pq} \rightarrow A_{pr}A_{rq}$.
- For every $p, q, r, s \in Q$, $b, c \in \Sigma_\epsilon$, and $d \in \Gamma_\epsilon$,

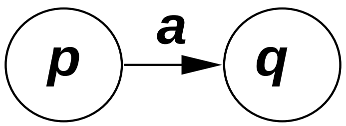
  if $(r, d) \in \delta(p, b, \epsilon)$ and $(q, \epsilon) \in \delta(s, c, d)$, then $A_{pq} \rightarrow bA_{rs}c$.



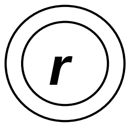
Regular Languages $\subset$ CFLs

First proof Any FA can be viewed as a PDA that never pops or pushes the stack.

Second proof Build a CFG for a FA, where

for 

we use $P \rightarrow aQ$

for 

we use $R \rightarrow \varepsilon$

Properties of Context-Free Languages

Theorem. *The context-free languages are closed under union, concatenation, and star.*

Proof Let S_1 and S_2 be the start symbols of two CFGs. Let S be the new start symbol of the new CFG we are creating.

Adding $S \rightarrow S_1 \mid S_2$ works for union.

Adding $S \rightarrow S_1 S_2$ works for concatenation.

Adding $S \rightarrow \epsilon \mid S S_1$ works for star. ∎

This also shows how regular expressions can be converted to equivalent CFGs, providing a 3rd proof that regular languages are CFLs.