CS340: Theory of Computation Sem I 2017-18

Lecture Notes 11: Pushdown Automata

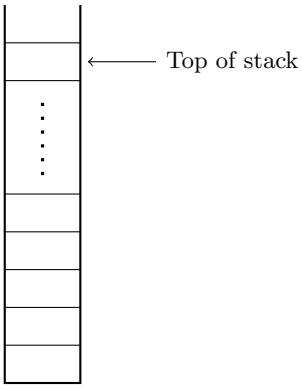
Raghunath Tewari IIT Kanpur

1 Pushdown Automata

It is an ϵ-NFA appended with a stack.

1.1 Reviewing a Stack

A stack is a data structure that allows addition/deletion/access of an element only at the top of a stack.



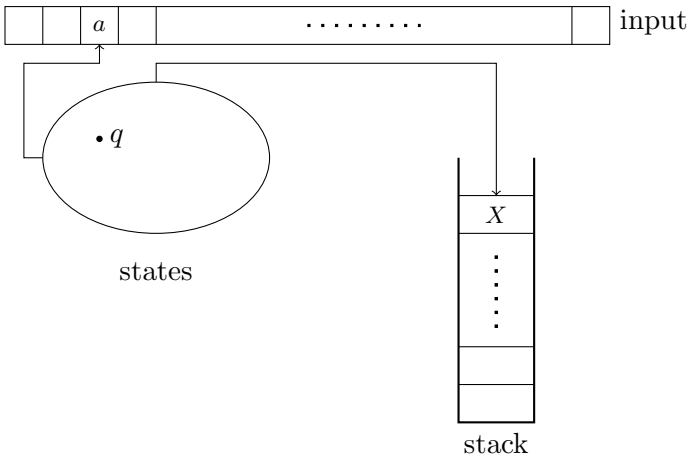
A stack

Name	Description	Position
Push	Adding an element	Top of stack
Pop	Deleting an element	Top of stack

The above table summarizes the operations that can be performed on a stack.

- The length of the stack is unbounded.
- Contents of the entire stack is not visible to the automaton at any instant. Only the top element is accessible.
- To get to an element the automaton has to "pop" its way down. Elements that are popped are forgotten.

1.2 Informal Description of a Pushdown Automaton



- We allow the pushdown automaton to have a potentially different alphabet for its stack. Usually Γ denotes the stack alphabet.

- In case of a pushdown automaton, the transition function depends on,

  (i) the current state,

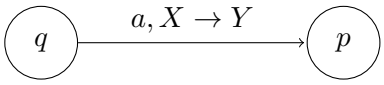
  (ii) the currently read input symbol (can be ϵ), and

  (iii) the currently read stack symbol at the top of the stack (again it can be ϵ).

- Each transition is of the form

$$(q, a, X) \longrightarrow (p, Y).$$

  This is represented as follows:



  In this case, the automaton,

  (i) reads the input bit a (can be ϵ),

  (ii) moves from state q to state p,

  (iii) replaces the symbol X at the top of the stack with the symbol Y (both can be ϵ).

  Observe that if $X = \epsilon$ then it corresponds to pushing Y on the stack and if $Y = \epsilon$ then it corresponds to popping X from the stack.

- Since a pushdown automaton is non-deterministic, multiple possibilities can exist.

1.3 Formal Definition

Definition 1.1. A *pushdown automaton* (PDA in short) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where, Q is the finite set of states, Σ and Γ are the input and stack alphabet respectively, q_0 is the start state, F is the set of accept states, and

$$\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \longrightarrow 2^{Q \times \Gamma_\epsilon}$$

is the transition function of the automaton.

- A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ accepts a string $w \in \Sigma^*$ if

  - there exists $a_1, a_2, \ldots a_m \in \Sigma_\epsilon$,
  - there exists states $r_0, r_1, \ldots r_m \in Q$, and
  - there exists strings $s_0, s_1, \ldots s_m \in \Gamma^*$,

  such that the following is true

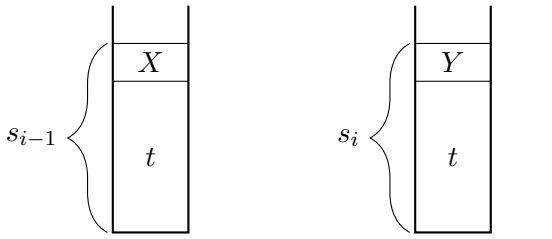
  (i) $w = a_1 a_2 \ldots a_m$ (input is concatenation of the symbols being read)

  (ii) $r_0 = q_0$ and $s_0 = \epsilon$, (initial condition)

  (iii) $\forall i \in \{1, 2, \ldots m\}$, if $(r_i, Y) \in \delta(r_{i-1}, a_i, X)$
  then $s_{i-1} = Xt$ and $s_i = Yt$, for some $X, Y \in \Gamma_\epsilon$ and $t \in \Gamma^*$. (transition condition)

(iv) $r_m \in F$ (acceptance condition)

The figure below shows how the stack changes in one transition step.



Before ith step After ith step

- The language of M,

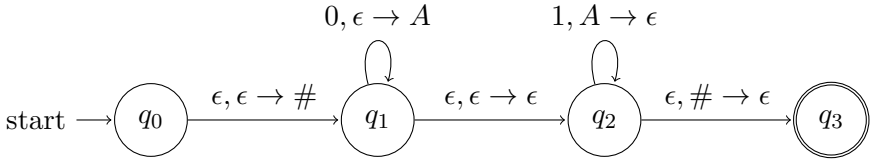
$$L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}.$$

1.4 Example of a PDA

Consider the non-regular language

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

Idea: Push a symbol (say 'A') when reading a 0 from the input and pop the symbol when reading a 1. To check if the stack becomes empty exactly after all the 1's have been read, we initially push a 2nd symbol (say '#') on the stack without reading any input bit. Now after the 1's have been read if the stack contains the symbol # at its top then we accept the input else not. Below is the state diagram of the PDA for L.



We will now see the computation of some strings on this automaton.

Input string: $0^2 1^2$

Symbol read	**Transition used**	**Current state**	**Stack contents**
–	–	q_0	ϵ
ϵ	$(q_0, \epsilon, \epsilon) \longrightarrow (q_1, \#)$	q_1	$\#$
0	$(q_1, 0, \epsilon) \longrightarrow (q_1, A)$	q_1	$A\#$
0	$(q_1, 0, \epsilon) \longrightarrow (q_1, A)$	q_1	$AA\#$
ϵ	$(q_1, \epsilon, \epsilon) \longrightarrow (q_2, \epsilon)$	q_2	$AA\#$
1	$(q_2, 1, A) \longrightarrow (q_2, \epsilon)$	q_2	$A\#$
1	$(q_2, 1, A) \longrightarrow (q_2, \epsilon)$	q_2	$\#$
ϵ	$(q_2, \epsilon, \#) \longrightarrow (q_3, \epsilon)$	$\mathbf{q_3}$	ϵ

The input is accepted since the final state is an accept state and the input is completely read.

Input string: $0^2 1$

Symbol read	Transition used	Current state	Stack contents
$-$	$-$	q_0	ϵ
ϵ	$(q_0, \epsilon, \epsilon) \longrightarrow (q_1, \#)$	q_1	$\#$
0	$(q_1, 0, \epsilon) \longrightarrow (q_1, A)$	q_1	$A\#$
0	$(q_1, 0, \epsilon) \longrightarrow (q_1, A)$	q_1	$AA\#$
ϵ	$(q_1, \epsilon, \epsilon) \longrightarrow (q_2, \epsilon)$	q_2	$AA\#$
1	$(q_2, 1, A) \longrightarrow (q_2, \epsilon)$	q_2	$A\#$

Observe that no more transitions can be applied at this point and the current state of the automaton is not an accept state. Hence the input is not accepted.

Input string: 01^2

Symbol read	Transition used	Current state	Stack contents
$-$	$-$	q_0	ϵ
ϵ	$(q_0, \epsilon, \epsilon) \longrightarrow (q_1, \#)$	q_1	$\#$
0	$(q_1, 0, \epsilon) \longrightarrow (q_1, A)$	q_1	$A\#$
ϵ	$(q_1, \epsilon, \epsilon) \longrightarrow (q_2, \epsilon)$	q_2	$A\#$
1	$(q_2, 1, A) \longrightarrow (q_2, \epsilon)$	q_2	$\#$
ϵ	$(q_2, \epsilon, \#) \longrightarrow (q_3, \epsilon)$	q_3	ϵ

In this case although the automaton has reached an accept state, its input will not be accepted since the input is not completely read.

Note 1. Recall that in a non-deterministic automaton (finite or pushdown) an input is accepted if there is *some* sequence of transitions (also known as *computation path*) that leads to acceptance. Conversely an input is not accepted if *all* computation paths lead to rejection. In the above example, for each of the 2nd and 3rd strings, only one non-accepting computation path is shown. You can work out the other computation paths and verify that all of them lead to rejection.

Exercise 1. Construct PDA for the following languages

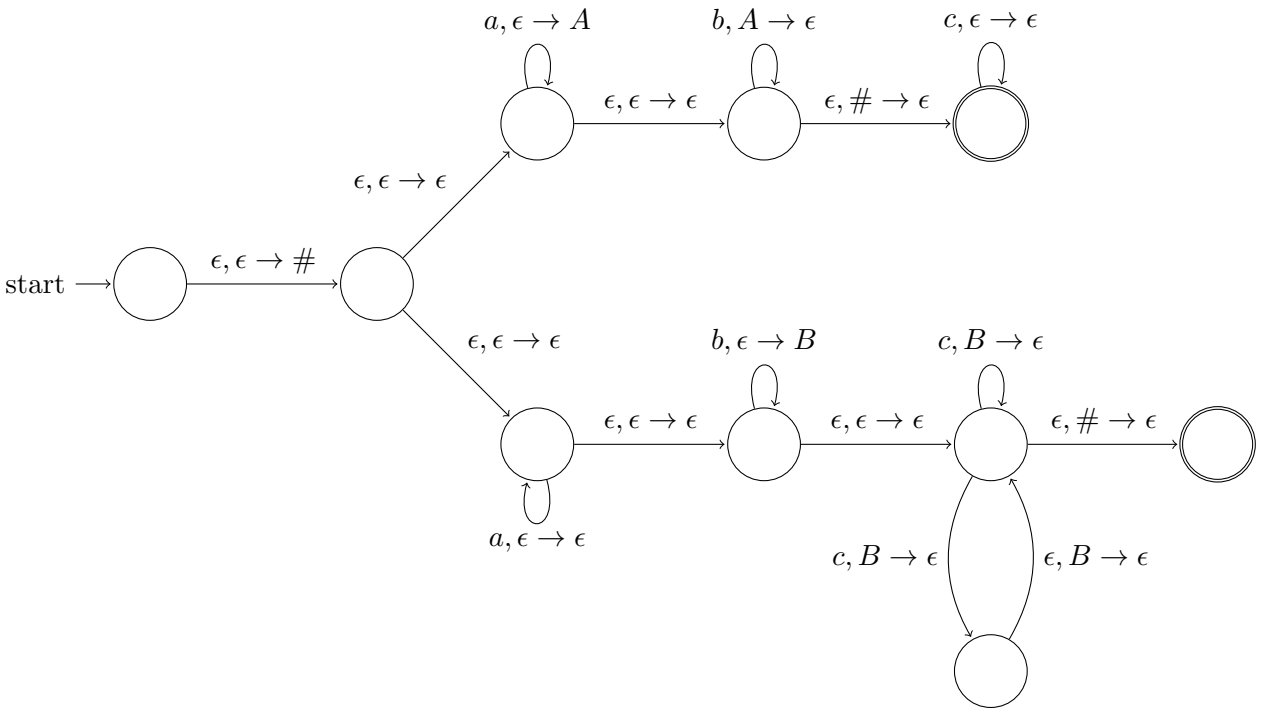
(i) $L_1 = \{w \in \{0,1\}^* \mid \#_0(w) = \#_1(w)\}$

(ii) $L_2 = \{0^{2n}1^{3n} \mid n \geq 0\}$

1.5 More Examples

1. Consider the language
$$L = \{a^i b^j c^k \mid i = j \text{ or } k \leq j \leq 2k\}.$$
We will use nondeterminism to decide which of the two conditions we are going to check and then proceed accordingly.

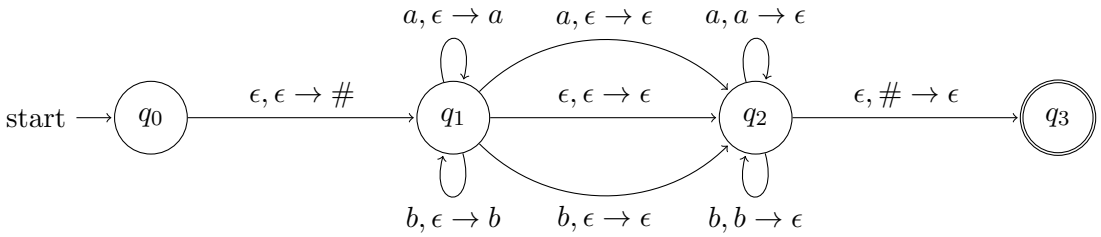
   To check whether $k \leq j \leq 2k$, we push a symbol to the stack for every b that we read and then for every c that we read, we again use non-determinism to pop either one or two of that symbol.



2. Another example

$$\text{PALINDROMES} = \{w = \text{rev}(w) \mid w \in \{a, b\}^*\}.$$

We first push first half of the string in the stack and then compare the second half by popping symbols one at a time. But how do we figure out the middle position of the string? Use nondeterminism!



State q_1 pushes symbols onto the stack as they are being read and state q_2 pops symbols out of the stack as they are being read. If a computation path makes a transition from q_1 to q_2 before reaching the middle point then the stack will become empty before the string is read completely. On the other hand if a computation path makes a transition from q_1 to q_2 after reaching the middle point then the stack will contain some symbols above # even after the string is completely read.

1.6 CFGs and PDAs are equivalent

Theorem 1. *L is a CFL if and only if there exists a PDA M such that $L = L(M)$.*

Proof Idea. To show that if L is a CFL then L is accepted by some PDA start with a CFG and use the stack of the PDA to store the intermediate strings generated in each step of a derivation

of a string. Terminals in the stack are "matched off" with the input of the PDA until a variable is reached. At this point, the PDA uses its non-determinism to replace the variable with one of its substitution rules and push it onto the stack. This process is carried on until the stack becomes empty and all the input symbols have been matched off.
For the other direction, we construct a CFG from a PDA. Without loss of generality assume that the PDA (a) has a single accept state, (b) empties the stack before accepting its input and (c) each transition either does a *push* operation or a *pop* operation but not both. The idea is to have variables of the form A_{pq} for each pair of states p, q. A_{pq} will generate all strings that takes the PDA from state p on an empty stack to state q on an empty stack (of course possibly popping/pushing some symbols at intermediate steps). A_{pq} is then defined inductively by dividing into two cases:

(i) *The first pushed symbol is same as the last popped symbol out of the stack.* Let a and b be the input symbols read at the first and last steps respectively, let r be the states following p and s be the state preceding q. Then we add the rule

$$A_{pq} \longrightarrow aA_{rs}b.$$

(ii) *The first pushed symbol is different from the last popped symbol.* Let r be an intermediate state when the stack becomes empty. In this case we add the rule

$$A_{pq} \longrightarrow A_{pr}A_{rq}.$$

□

Exercise 2 (Reading Exercise)**.** Read the complete proof from your textbook.

Exercise 3. Construct PDA for the following languages

(i) $L_1 = \{a^i b^j c^k \mid j \leq i + k \leq 2j\}$

(ii) $L_2 = \{a^i b^j \mid i \neq j\}$

(iii) $L_3 = L(a^*b^*c^*) \setminus \{a^n b^n c^n \mid n \geq 0\}$

(iv) $L_4 = \overline{L}$, where $L = \{ww \mid w \in \{a, b\}^*\}$