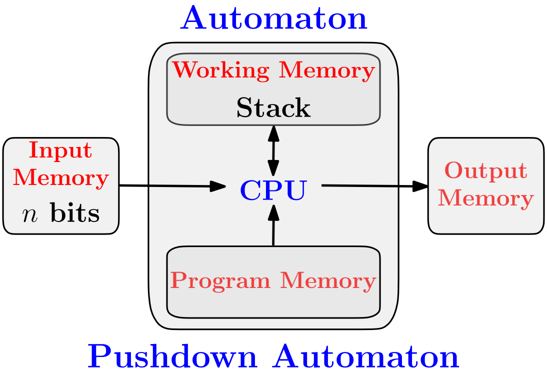
Push Down Automata

Imdad ullah Khan

Push Down Automata

Automata are distinguished by type/amount of working memory

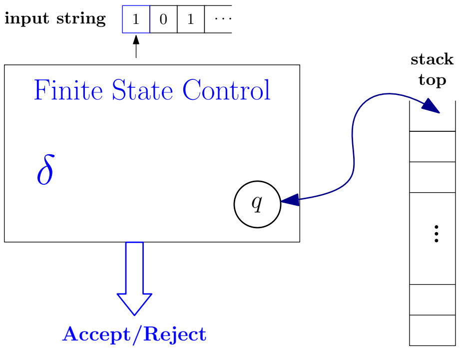
A Push Down Automata has LIFO (stack) working memory



A push down automaton (PDA) is a finite automaton with a stack

The stack can store an infinite number of symbols from a stack alphabet Γ

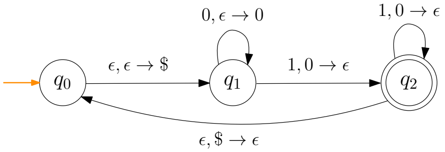
The PDA reads an input symbol and stack top, changes state, and pushes onto the stack in one transition



Anatomy of PDA

A PDA over alphabet Σ and stack alphabet Γ is depicted as a directed graph with self-loop

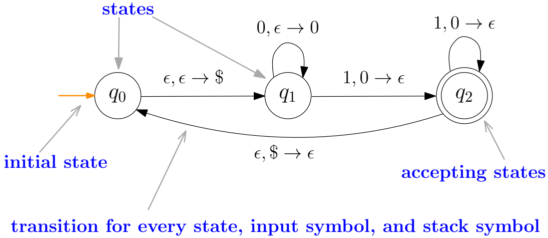
$\triangleright$ called state diagram of the PDA



Anatomy of PDA

A PDA over alphabet Σ and tape alphabet Γ is depicted as a directed graph with self-loop

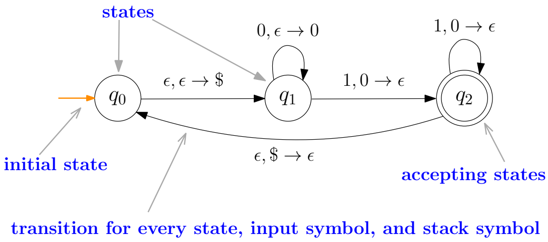
▷ called state diagram of the PDA

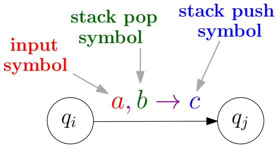


Anatomy of PDA

A PDA over alphabet Σ and stack alphabet Γ is depicted as a directed graph with self-loop

▷ called state diagram of the PDA

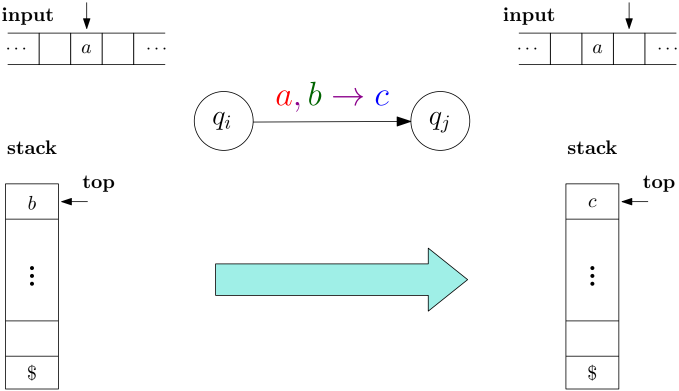




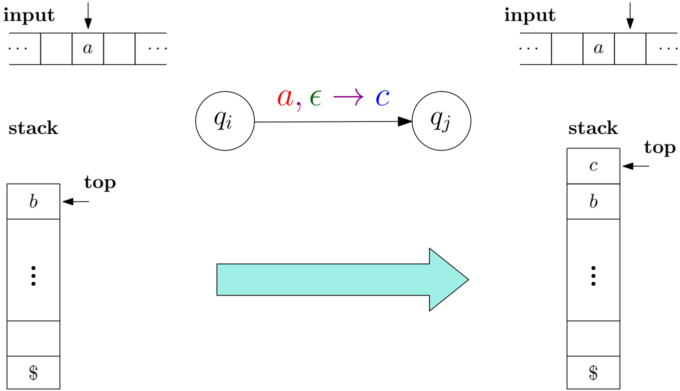
$\$ \in \Gamma$: Initial stack symbol that is helpful to check if the stack is empty. By convention, it is pushed in the first transition in a PDA

ϵ: Empty string indicates read nothing, pop nothing or push nothing

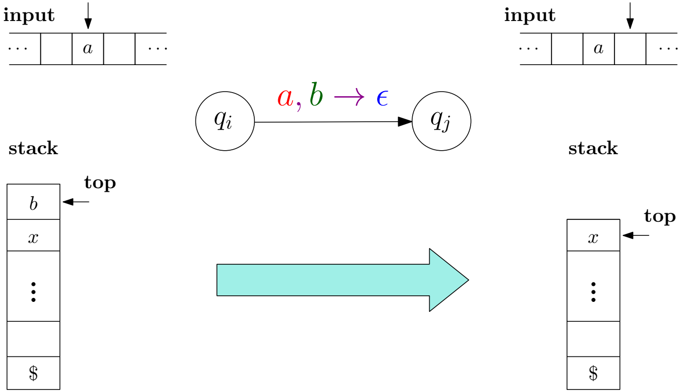
Working of PDA



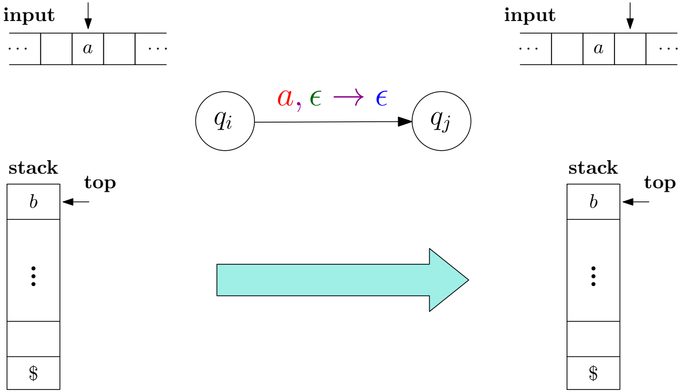
Working of PDA



Working of PDA

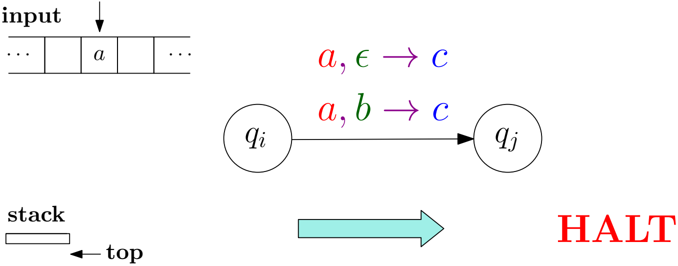


Working of PDA



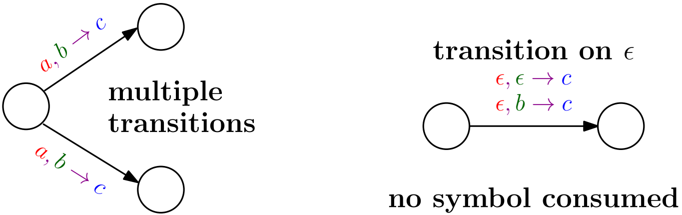
Working of PDA

No transition is allowed to be followed when stack is empty (except pushing $)

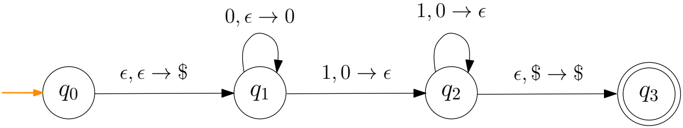


The PDA halts in q_i and rejects the input string

Non-Deterministic PDA



Working of PDA: Example



A string is accepted if there is a computation from start state (with non-deterministic choices) such that all input is consumed and the last state is final state

At the end of computation, we do not care about content of stack

PDA: Formal Definition

A PDA P is a 7-tuple $P = (Q, \Sigma, \Gamma, \delta, q_0, \$, F)$

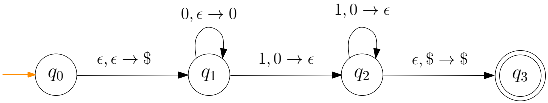
- Q is a finite set of states
- Σ is a finite input alphabet
- Γ is a finite stack alphabet
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^*)$ is the transition function
- $q_0 \in Q$ is the initial state
- $\$ \in \Gamma$ is the initial stack symbol
- $F \subseteq Q$ is the set of final states

NPDA for $L = \{0^n1^n | n \geq 0\}$

Consider the language $L = \{0^n1^n | n \geq 0\}$

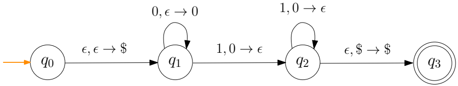
L is not regular, but we can design a PDA to recognize L as follows:

- As each 0 is read, push it onto the stack
- As each 1 is read, pop a 0 from the stack
- If the input is exhausted and the stack is empty, accept
- Otherwise, reject.



Simulating the PDA for $L = \{0^n 1^n | n \geq 0\}$

Configuration of PDA: (q, u, s) denotes the PDA configuration, q is the current state, u is the remaining input, and s is the current stack contents



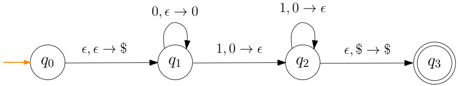
Consider the string $w = 0011 \in L$ The PDA runs as follows:

Pre-transition configuration	Transition	Post-transition configuration
$(q_0, 0011, \epsilon)$	$\epsilon, \epsilon \to \$$	$(q_1, 0011, \$)$
$(q_1, 0011, \$)$	$0, \epsilon \to 0$	$(q_1, 011, 0\$)$
$(q_1, 011, 0\$)$	$0, \epsilon \to 0$	$(q_1, 11, 00\$)$
$(q_1, 11, 00\$)$	$1, 0 \to \epsilon$	$(q_2, 1, 0\$)$
$(q_2, 1, 0\$)$	$1, 0 \to \epsilon$	$(q_2, \epsilon, \$)$
$(q_2, \epsilon, \$)$	$\epsilon, \$ \to \epsilon$	$(q_3, \epsilon, \$)$

The PDA accepts the string w by empty stack

Simulating the PDA for $L = \{0^n 1^n | n \geq 0\}$

Configuration of PDA: (q, u, s) denotes the PDA configuration, q is the current state, u is the remaining input, and s is the current stack contents



Consider the string $w = 0101 \notin L$ The PDA runs as follows:

Pre-transition configuration	Transition	Post-transition configuration
$(q_0, 0101, \epsilon)$	$\epsilon, \epsilon \to \$$	$(q_1, 0101, \$)$
$(q_1, 0101, \$)$	$0, \epsilon \to 0$	$(q_1, 101, 0\$)$
$(q_1, 101, 0\$)$	$1, 0 \to \epsilon$	$(q_2, 01, \$)$
$(q_2, 01, \$)$	No δ	Reject

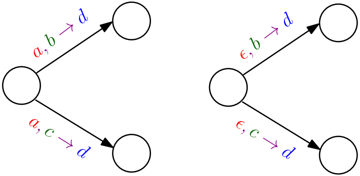
The PDA rejects the string w because there is no transition possible from the configuration $(q_2, 01, \$)$

PDA vs DFA

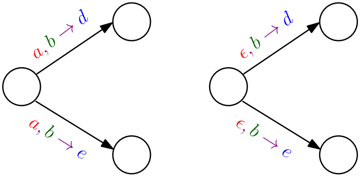
- A PDA is an extension of a DFA with an additional stack memory
- A DFA can only remember a finite amount of information, but a PDA can remember an infinite amount of information using the stack
- A DFA can recognize regular languages, which are a subset of context-free languages
- A PDA can recognize context-free languages, which are more expressive and complex than regular languages
- A DFA can be simulated by a PDA by ignoring the stack and using the same transitions as the DFA
- A PDA cannot be simulated by a DFA in general, because a DFA cannot handle the stack operations and the nondeterminism of the PDA

Deterministic PDA

Allowed Transitions



Not Allowed Transitions

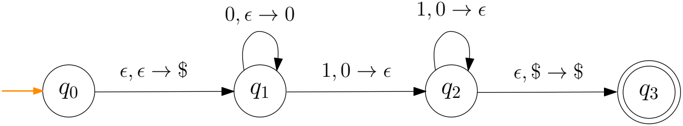


Models of PDA and NPDA

- A PDA can be either deterministic (DPDA) or nondeterministic (NPDA)
- A DPDA is a PDA that has at most one possible transition for any given state, input symbol, and stack symbol
- A NPDA is a PDA that can have more than one possible transition for some state, input symbol, and stack symbol
- A NPDA can also have ϵ-transitions, which do not consume any input symbol but may change the state and the stack
- A NPDA can simulate any DPDA, but not vice versa. Therefore, NPDA is more powerful than DPDA

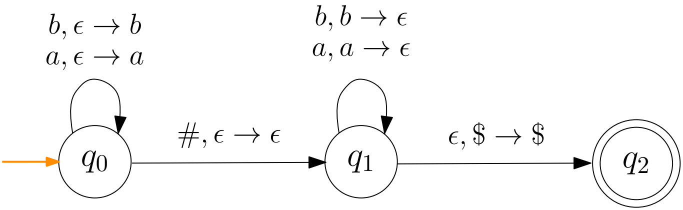
Examples of DPDA

The language $L_1 = \{0^n 1^n | n \geq 0\}$ is recognized by the DPDA



Examples of DPDA

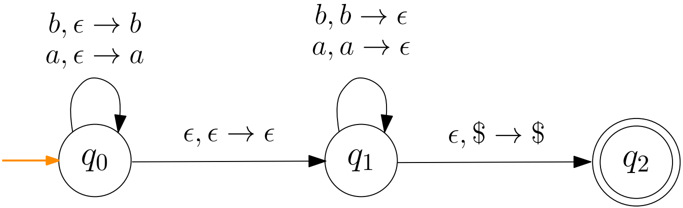
The language $L_2 = \{v\#v^R | v \in \{a, b, \#\}^*\}$ is recognized by the DPDA



Examples of NPDA

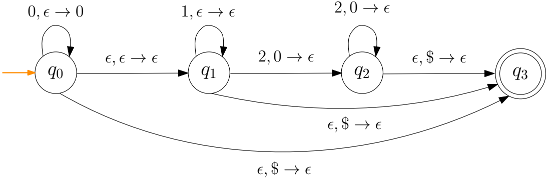
The language $L_3 = \{vv^R | v \in \{a, b\}^*\}$ is recognized by the NPDA

We non-deterministically guess when v ends and v^R starts



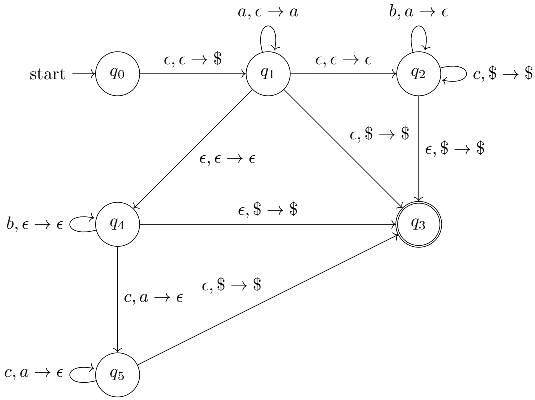
Examples of NPDA

The language $L_4 = \{0^i 1^j 2^k | i, j, k \geq 0 \text{ and } i = k\}$ is recognized by the NPDA



Examples of NPDA

The language $L_5 = \{a^i b^j c^k | i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$ is recognized by the NPDA



Limitation of PDA

- A PDA is a powerful model of computation, but it is not as powerful as a Turing machine
- A PDA can only recognize context-free languages, which are a proper subset of recursively enumerable languages
- A PDA has a limited memory in the form of a stack, which can only be accessed from the top
- A PDA cannot perform arbitrary operations on the stack, such as random access, copying, or reversing
- A PDA cannot handle languages that require more complex memory structures, such as queues, counters, or tapes

Pumping Lemma for Context-Free Languages

- The pumping lemma for context-free languages is a property that all context-free languages share, and it can be used to prove that some languages are not context-free
- The pumping lemma states that if a language L is context-free, then there exists some integer m (called the pumping length) such that every string w in L that has a length of m or more symbols can be written as $w = uvxyz$, where u, v, x, y, and z are substrings of w, such that:
  - $|vxy| \leq m$
  - $|vy| > 0$
  - $uv^kxy^kz \in L$ for all $k \geq 0$
- The intuition behind the pumping lemma is that a long enough string in a context-free language must have a repeated pattern in its derivation tree, and this pattern can be pumped up or down without leaving the language

Examples of Languages that Cannot be Decided by PDA

The following languages that are not context-free, and hence cannot be decided by a PDA

- $L_6 = \{a^n b^n c^n | n \geq 0\}$
- $L_7 = \{a^n b^m c^n d^m | n, m \geq 0\}$
- $L_8 = \{a^{n^2} | n \geq 0\}$