Pushdown Automata for CFLs

Prof. Shrisha Rao
IIIT Bangalore

srao@iiitb.ac.in

2024-09-05

Exercises

Give a CFG for each of the following languages:

(1) The language $\{0^n1^m \mid n \neq m\}$. *Hint: This requires two cases; $n < m$ and $n > m$.*

(2) The language of all UNBALANCED brackets, on the alphabet $\{(,)\}$.

(3) The language of all strings in $(0+1)^*$ that have twice as many 1s as 0s.

Answers

(1) $\{0^n1^m \mid n \neq m\}$:
$S \longrightarrow A \mid B$
$A \longrightarrow 0A1 \mid A1 \mid 1$ (for $n < m$)
$B \longrightarrow 0B1 \mid 0B \mid 0$ (for $n > m$)

(2) Unbalanced brackets, on $\{(,)\}$.

$$S \rightarrow U \mid O$$

$$U \rightarrow U\,U \mid (\,U\,) \mid \,)\, \mid \,)\,U \mid (\,U$$

$$O \rightarrow O\,O \mid (\,O\,) \mid (\, \mid (\,O \mid O\,($$

(3) All strings in $(0+1)^*$ that have twice as many 1s as 0s.
$S \longrightarrow \epsilon \mid 1\,S\,1\,S\,0\,S \mid 1\,S\,0\,S\,1\,S \mid 0\,S\,1\,S\,1\,S$

Pushdown Automata

- We have seen that a DFA/NFA cannot accept context-free languages such as $\{0^n 1^n \mid n \geq 0\}$, and a reason for this is that they have no storage or memory.
- Thus, an obvious way to derive a machine that can accept CFLs is to enhance a finite automaton with storage.
- A *pushdown automaton* (PDA) is a finite automaton with a pushdown stack added. A stack is a LIFO structure, so stack operations only access the top element in the stack.

Formal Definition of a Pushdown Automaton

A *pushdown automaton* is a 6-tuple $\langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ where Q is a finite set of states, Σ is a finite set called the *input alphabet*, Γ is a finite set called the *stack alphabet*, $q_0 \in Q$ is a distinguished start state, $F \subseteq Q$ a set of final states, and δ is a transition function, with

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\lambda\}) \rightarrow 2^{Q \times (\Gamma \cup \{\lambda\})}$$

PDA ALPHABETS

- The input alphabet Σ is similar to the corresponding alphabet of a DFA/NFA. It is denoted by lowercase letters $a, b, c, \ldots$ or numerical symbols 0 and 1.
- The *stack alphabet* Γ (which is not the same as the input alphabet) contains elements that may be stored on the stack. Elements of Γ are denoted by uppercase letters $A, B, C, \ldots$.
- Similar to ϵ which represents the empty string, we use λ which represents the empty stack symbol (reflecting the lack of a stack push or pop).

PDA OPERATIONS

- The computation of a PDA always starts with the automaton in state q_0, the input to be given, and the stack empty.
- A PDA considers the current state, the input symbol, and the symbol on the top of the stack, to reach the next state. During a transition, a PDA may also push a symbol onto the stack.
- A PDA is said to accept a string if it ends on an accept state *with the stack empty*. If a PDA is in an accept state and there is no further symbol to process, but the stack is NOT empty, then the PDA is not considered to accept the string.

PDA TRANSITIONS

The transition function δ gives all possible transitions for a given state, input symbol, and stack top value. For instance, the value

$$\delta(q_i, a, A) = \{(q_j, B), (q_k, C)\}$$

indicates that when the PDA is in state q_i, has an input symbol a, and the stack top symbol is $A \in \Gamma \cup \{\lambda\}$, there are two possible transitions: to state q_j with B pushed onto the stack, or q_k with C pushed onto the stack.

PDA transitions—cont'd

Alternatively,

$$(q_j, B) \in \delta(q_i, a, A)$$

means that q_i is the current state, a is the current input symbol, and A is the current stack top value. The transition may cause the PDA to accept the input symbol a, change the state from q_i to q_j, pop A from the stack, and push B onto the stack.

State Diagrams for PDAs

$$\delta(q_i, a, A) = (q_j, B)$$

which means that in state q_i, on reading input symbol a and popping the stack top A, the PDA moves to state q_j and pushes B onto the stack, is represented as:

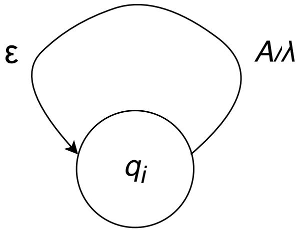
State Diagrams for PDAs—cont'd

$$(q_i, \lambda) \in \delta(q_i, \epsilon, A)$$

State Diagrams for PDAs—cont'd

$$(q_i, \lambda) \in \delta(q_i, \epsilon, A)$$

In state q_i, on reading ϵ (i.e., no input symbol), pop A from the stack, stay in q_i, and push λ (i.e., nothing) onto the stack.



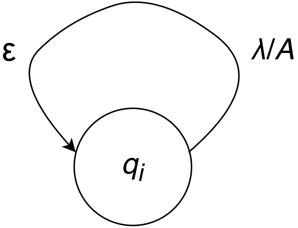
State Diagrams for PDAs—cont'd

$$(q_i, A) \in \delta(q_i, \epsilon, \lambda)$$

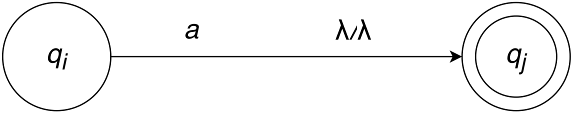
State Diagrams for PDAs—cont'd

$$(q_i, A) \in \delta(q_i, \epsilon, \lambda)$$

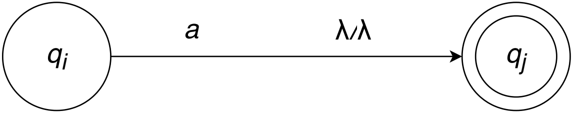
In state q_i on reading ϵ (i.e., no input symbol) and popping λ from the stack (i.e., not popping the stack), stay in q_i and push A onto the stack.



State Diagrams for PDAs—cont'd



State Diagrams for PDAs—cont'd



This is the PDA version of a standard DFA transition; in state q_i, with input symbol a, λ is popped from the stack (i.e., the stack is not popped), the state is changed to an accept state q_j, and λ is pushed onto the stack (i.e., there is no stack push).

$$\delta(q_i, a, \lambda) = (q_j, \lambda)$$

A PDA TO ACCEPT THE CFL $\{a^n b^n \mid n \geq 0\}$

- As always, the PDA has to process the input string (which may be of the form $a^n b^n$), and starts with the stack empty.
- The PDA must end in an accept state *and have an empty stack*, to be considered to accept the input string. (If the stack is non-empty, the string is not considered to be accepted even if the PDA is an accept state.)
- The obvious solution is to push a symbol A onto the stack each time a is read, and then pop A from the stack each time b is read.

A PDA to accept $\{a^n b^n \mid n \geq 0\}$

$Q = \{q_0, q_1\}$

$\Sigma = \{a, b\}$

$\Gamma = \{A\}$

$F = ?$

$\delta = ?$

A PDA to accept $\{a^n b^n \mid n \geq 0\}$—cont'd

$Q = \{q_0, q_1\}$

$\Sigma = \{a, b\}$

$\Gamma = \{A\}$

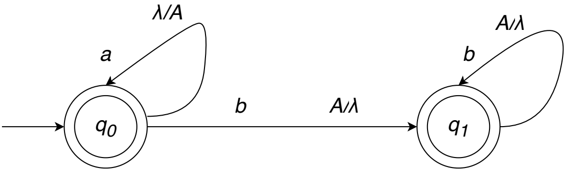
$F = \{q_0, q_1\}$

$\delta(q_0, a, \lambda) = (q_0, A)$

$\delta(q_0, b, A) = (q_1, \lambda)$

$\delta(q_1, b, A) = (q_1, \lambda)$

A PDA TO ACCEPT $\{a^n b^n \mid n \geq 0\}$—CONT'D

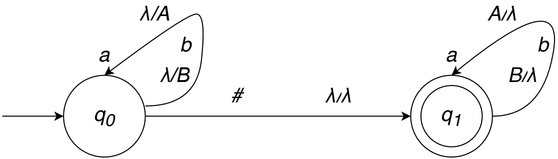


Exercise: show the computation of this PDA for the strings *aaabbb* (which is accepted) and *aabbb* (which is not).

A PDA TO ACCEPT $\{w\#w^R \mid w \in (a+b)^*\}$

- In this case we need to copy the string w onto the stack, and read it off in the reverse order after reaching $\#$, to compare with w^R.
- We can still do it with just two states, but we need three input symbols $a, b, \#$, and two stack symbols (e.g., A and B).

A PDA TO ACCEPT $\{w\#w^R \mid w \in (a+b)^*\}$—CONT'D



Exercise: write down the full specification $(Q, \Sigma, \Gamma, \delta, F)$ of the PDA, and show its working for the string $aabb\#bbaa$ (which is accepted) and $abba\#abb$ (which is not).

Practice Problems for PDAs

Give a PDA for each the following languages:

4. $\emptyset$
5. $\{\epsilon\}$
6. $\{w \mid w$ contains an unequal number of 1s and 0s$\}$
7. $\{w \mid$ the length of w is even$\}$
8. $\{w \mid$ the length of w is at least 3 and its final symbol is a $0\}$
9. $\{w \mid w \in (0+1)^* \setminus \{0^i 1^i\}, i \in \mathbb{N}\ \}$
10. $\{0^m 1^n \mid m = n \text{ or } m = 2n\}$.
11. $\{0^n \mid n \geq 0\} \cup \{0^n 1^n \mid n \geq 0\}$