CIS 770: Formal Language Theory

Pavithra Prabhakar

Kansas State University

Spring 2015

Restricted Infinite Memory: The Stack

- So far we considered automata with finite memory or machines with infinite memory
- Today: automata with access to an infinite stack — infinite memory but restricted access
- The stack can contain an unlimited number of characters. But
  - can read/erase only the top of the stack: pop
  - can add to only the top of the stack: push
- On longer inputs, automaton may have more items in the stack

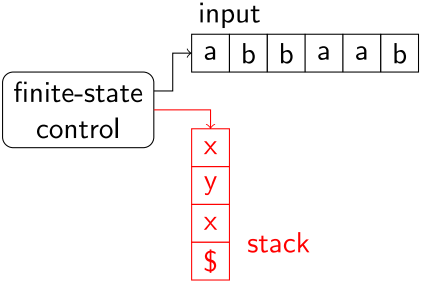
Keeping Count Using the Stack

- An automaton can use the stack to recognize $\{0^n1^n\}$
  - On reading a 0, push it onto the stack
  - After the 0s, on reading each 1, pop a 0
  - (If a 0 comes after a 1, reject)
  - If attempt to pop an empty stack, reject
  - If stack not empty at the end, reject
  - Else accept

Matching Parenthesis Using the Stack

- An automaton can use the stack to recognize balanced parenthesis
- e.g. (())() is balanced, but ())() and (() are not
  - On seeing a (push it on the stack
  - On seeing a) pop a (from the stack
  - If attempt to pop an empty stack, reject
  - If stack not empty at the end, reject
  - Else accept

Pushdown Automata (PDA)

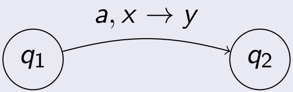


A Pushdown Automaton

- Like an NFA with ϵ-transitions, but with a stack
  - Stack depth unlimited: not a finite-state machine
  - Non-deterministic: accepts if any thread of execution accepts

Pushdown Automata (PDA)

- Has a non-deterministic finite-state control
- At every step:
  - Consume next input symbol (or none) and pop the top symbol on stack (or none)
  - Based on current state, consumed input symbol and popped stack symbol, do (non-deterministically):
    1. push a symbol onto stack (or push none)
    2. change to a new state



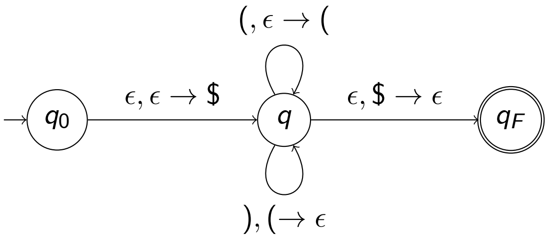
If at q_1, with next input symbol a and top of stack x, then can consume a, pop x, push y onto stack and move to q_2 (any of a, x, y may be ϵ)

Pushdown Automata (PDA): Formal Definition

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where

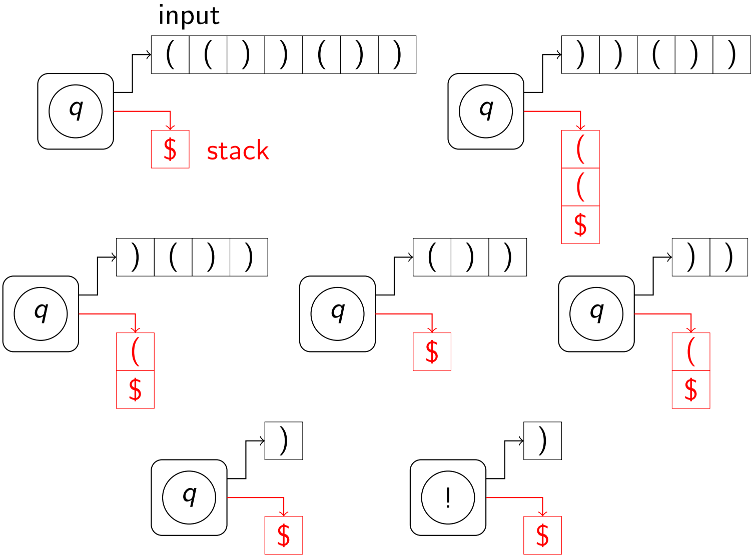
- Q = Finite set of states
- Σ = Finite input alphabet
- Γ = Finite stack alphabet
- q_0 = Start state
- $F \subseteq Q$ = Accepting/final states
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q \times (\Gamma \cup \{\epsilon\}))$

Matching Parenthesis: PDA construction

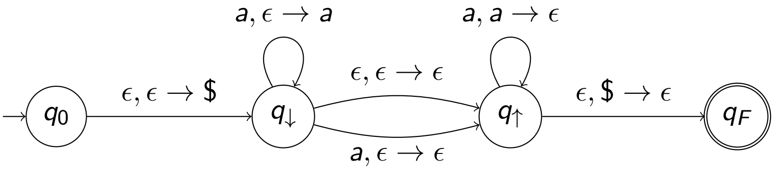


- First push a “bottom-of-the-stack” symbol \$ and move to q
- On seeing a (push it onto the stack
- On seeing a) pop if a (is in the stack
- Pop \$ and move to final state q_F

Matching Parenthesis: PDA execution

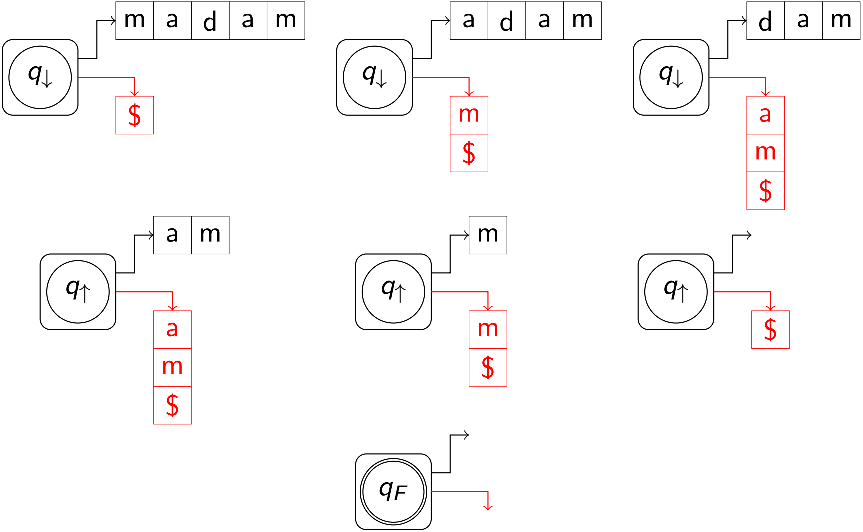


Palindrome: PDA construction


$a, \epsilon \to a$
$a, a \to \epsilon$
$\epsilon, \epsilon \to \$$
$\epsilon, \epsilon \to \epsilon$
$\epsilon, \$ \to \epsilon$
q_0
$q_\downarrow$
$q_\uparrow$
q_F
$a, \epsilon \to \epsilon$

- First push a “bottom-of-the-stack” symbol \$ and move to a pushing state
- Push input symbols onto the stack
- Non-deterministically move to a popping state (with or without consuming a single input symbol)
- If next input symbol is same as top of stack, pop
- If \$ on top of stack move to accept state

Palindrome: PDA execution



Instantaneous Description

In order to describe a machine's execution, we need to capture a "snapshot" of the machine that completely determines future behavior

- In the case of an NFA (or DFA), it is the state
- In the case of a PDA, it is the state + stack contents

Definition

An instantaneous description of a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is a pair $\langle q, \sigma \rangle$, where $q \in Q$ and $\sigma \in \Gamma^*$

Computation

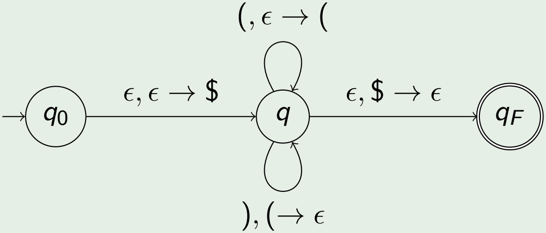
Definition

For a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, string $w \in \Sigma^*$, and instantaneous descriptions $\langle q_1, \sigma_1 \rangle$ and $\langle q_2, \sigma_2 \rangle$, we say $\langle q_1, \sigma_1 \rangle \xrightarrow{w}_P \langle q_2, \sigma_2 \rangle$ iff there is a sequence of instantaneous descriptions $\langle r_0, s_0 \rangle, \langle r_1, s_1 \rangle, \ldots \langle r_k, s_k \rangle$ and a sequence $x_1, x_2, \ldots x_k$, where for each i, $x_i \in \Sigma \cup \{\epsilon\}$, such that

- $w = x_1 x_2 \cdots x_k$,
- $r_0 = q_1$, and $s_0 = \sigma_1$,
- $r_k = q_2$, and $s_k = \sigma_2$,
- for every i, $(r_{i+1}, b) \in \delta(r_i, x_{i+1}, a)$ such that $s_i = as$ and $s_{i+1} = bs$, where $a, b \in \Gamma \cup \{\epsilon\}$ and $s \in \Gamma^*$

Example of Computation

Example



$\langle q_0, \epsilon \rangle \xrightarrow{(()(} \langle q, ((\$ \rangle$ because

$$\langle q_0, \epsilon \rangle \xrightarrow{x_1=\epsilon} \langle q, \$ \rangle \xrightarrow{x_2=(} \langle q, (\$ \rangle \xrightarrow{x_3=(} \langle q, ((\$ \rangle \xrightarrow{x_4=)} \langle q, (\$ \rangle \xrightarrow{x_5=(} \langle q, ((\$ \rangle$$

Acceptance/Recognition

Definition

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ accepts a string $w \in \Sigma^*$ iff for some $q \in F$ and $\sigma \in \Gamma^*$, $\langle q_0, \epsilon \rangle \xrightarrow{w}_P \langle q, \sigma \rangle$

Definition

The language recognized/accepted by a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is $L(P) = \{w \in \Sigma^* \mid P \text{ accepts } w\}$. A language L is said to be accepted/recognized by P if $L = L(P)$.

Expressive Power of CFGs and PDAs

CFGs and PDAs have equivalent expressive powers. More formally, . . .

Theorem

For every CFG G, there is a PDA P such that $L(G) = L(P)$. In addition, for every PDA P, there is a CFG G such that $L(P) = L(G)$. Thus, L is context-free iff there is a PDA P such that $L = L(P)$.

From CFG to PDA

Problem

Given grammar $G = (V, \Sigma, R, S)$ need to design PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that $L(P) = L(G)$. In other words, given $w \in \Sigma^*$, the PDA P needs to figure out whether $S \overset{*}{\Rightarrow}_G w$.

Intuition

The PDA P will try to construct a derivation of w from S, by "applying" one rule at a time starting from S.

Simulating Derivations

Challenge I: Choosing rule to apply

How do you choose the rule of grammar to apply to get the next step of derivation?

Use non-determinism to guess this choice!

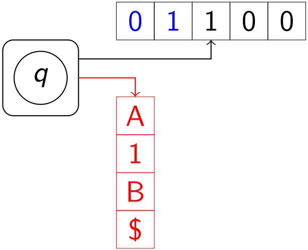
Simulating Derivations

Challenge II: Storing Intermediate strings

In order to construct the derivation, we need to know what the current intermediate string is, so that we know what rules/steps can be applied next. How do we store this intermediate string?

- Store the intermediate string on the stack!
  - Doesn't work! The PDA can only read the top of the stack (which is say the leftmost symbol of the intermediate string), but it needs to know some variable of the string to know which rule to apply.
- Store only part of the string on the stack; the part that immediately follows the first (leftmost) variable of the intermediate string
- Portion before the first variable in the intermediate string will be "matched" with the input

Example



The above PDA snapshot depicts intermediate string $01A1B$ as follows:

- 01 has been read from the input
- $A1B$ is on the stack; “bottom of stack” is $

PDA Algorithm

1. Push \$ and the start symbol onto the stack.
2. Repeat the following steps
    1. If top of the stack is a variable A, pick (nondeterministically) a rule for A, and push onto the stack the RHS of the rule; don't read any input symbol.
    2. If top of the stack is a terminal a, then read the next input symbol. If the input symbol is not a, then this branch dies. Otherwise, continue.
    3. If top of the stack is \$ then pop the symbol and go to accepting state. There are no transition from accept state, and so if input is accepted only if there are no more input symbols.

Pushing Multiple Symbols

PDA Transitions

Formally, $\delta(q, x, a)$ contains pairs (q', b) which means that the PDA reads at most one symbol from input (if $x \neq \epsilon$), pops the top of the stack (if $a \neq \epsilon$) and pushes at most one symbol onto the stack (if $b \neq \epsilon$).

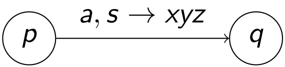
To simplify the construction, we will assume that we can push as many symbols as we want in one step. Thus, $\delta(q, x, a)$ contains pairs (q', σ), where $\sigma = b_1 b_2 \cdots b_k \in \Gamma^*$

- Can easily be carried out by additional states that push the symbols of σ one by one. Formally, we have new states $q_1, \ldots q_{k-1}$ and transitions

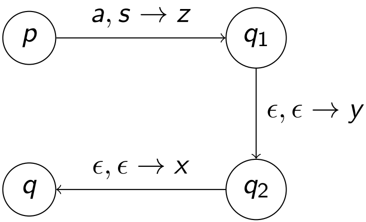
$$\begin{array}{l}(q_1, b_k) \in \delta(q, x, a),\ \delta(q_1, \epsilon, \epsilon) = \{(q_2, b_{k-1})\}, \\ \delta(q_2, \epsilon, \epsilon) = \{(q_3, b_{k-2})\},\ \ldots\ \delta(q_{k-1}, \epsilon, \epsilon) = \{(q', b_1)\}\end{array}$$

Pushing Multiple Symbols

An Example



Transition pushing multiple symbols


p
$a, s \rightarrow z$
q_1
$\epsilon, \epsilon \rightarrow y$
q_2
$\epsilon, \epsilon \rightarrow x$
q

Multiple pushes implemented by single pushes

Formal Definition of PDA

Let $G = (V, \Sigma, R, S)$ be a CFG. Define $P_G = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that

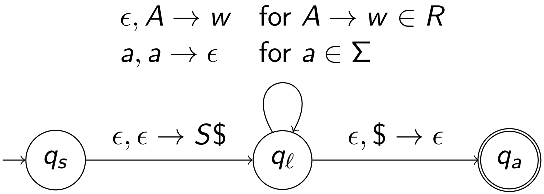
- $Q = \{q_s, q_\ell, q_a\}$
- $\Gamma = V \cup \Sigma \cup \{\$\}$, where $\$ \notin V \cup \Sigma$
- $q_0 = q_s$
- $F = \{q_a\}$
- Assuming $a \in \Sigma$ and $A \in V$, δ is given by

$$\begin{array}{ll} \delta(q_s, \epsilon, \epsilon) = \{(q_\ell, S\$)\} & \delta(q_\ell, a, a) = \{(q_\ell, \epsilon)\} \\ \delta(q_\ell, \epsilon, A) = \{(q_\ell, w) \mid A \rightarrow w \in R\} & \delta(q_\ell, \epsilon, \$) = \{(q_a, \epsilon)\} \end{array}$$

In all other cases, $\delta(\cdot) = \emptyset$

To remove steps that push more than one symbol, we will add more states.

State Diagram of PDA

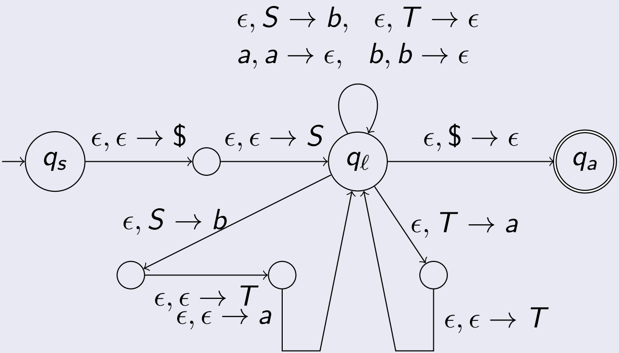

$\epsilon, A \to w$ for $A \to w \in R$
$a, a \to \epsilon$ for $a \in \Sigma$
$\epsilon, \epsilon \to S\$$
$\epsilon, \$ \to \epsilon$
q_s
q_ℓ
q_a

State Diagram of P_G

CFG to PDA

An example

Consider the CFG $G = (\{S, T\}, \{a, b\}, \{S \to aTb \mid b, T \to Ta \mid \epsilon\}, S)$. The PDA P_G is as follows.



$\epsilon, S \to b,\quad \epsilon, T \to \epsilon$
$a, a \to \epsilon,\quad b, b \to \epsilon$
q_s
$\epsilon, \epsilon \to \$$
$\epsilon, \epsilon \to S$
q_ℓ
$\epsilon, \$ \to \epsilon$
q_a
$\epsilon, S \to b$
$\epsilon, T \to a$
$\epsilon, \epsilon \to T$
$\epsilon, \epsilon \to a$
$\epsilon, \epsilon \to T$

Leftmost Derivations

Derivations

At each step, replace *some* variable in the intermediate string by the right-hand side of a rule in the grammar.

Leftmost Derivations

At each step, replace the *leftmost* variable in the intermediate string by the right-hand side of a rule in the grammar.
We will say $\alpha \Rightarrow_{\mathsf{lm}} \beta$ if β is obtained from α by a leftmost derivation step.

Proposition

For any grammar $G = (V, \Sigma, R, S)$, $A \in V$, *and* $\alpha \in (V \cup \Sigma)^*$,
$A \overset{*}{\Rightarrow} \alpha$ *if and only if* $A \overset{*}{\Rightarrow}_{lm} \alpha$.

Head/Tail of Intermediate Strings

Definition

Let $\beta = wA\alpha \in (V \cup \Sigma)^*$ be such that $w \in \Sigma^*$, $A \in V$ and $\beta \in (V \cup \Sigma)^*$. The head of β is w, and the tail is $A\alpha$.

Example

The head of string $01A1B$ is 01 and the tail is $A1B$.

Correctness of Construction

Proposition

Let G be a CFG and let P_G be the PDA constructed. Then $L(P_G) = L(G)$

Proof.

The proof of correctness is based on the following two observations.

- Let $S \overset{*}{\Rightarrow}_{\mathsf{lm}} x\alpha$ be a leftmost derivation, where x is the head and α the tail of the string $x\alpha$. Then, $\langle q_s, \epsilon\rangle \xrightarrow{x}_{P_G} \langle q_\ell, \alpha\$\rangle$. Proved by induction on the number of leftmost derivation steps.
- For every $A \in V$, if $\langle q_\ell, A\rangle \xrightarrow{x}_{P_G} \langle q_\ell, \epsilon\rangle$ then $A \overset{*}{\Rightarrow} x$. Proved by induction on the number of steps taken by P_G.

Proof is skipped. □