Lecture 7
Pushdown Automata

COSE215: Theory of Computation

Seunghoon Woo

Fall 2023

Contents

- **Deterministic Pushdown Automata**

Deterministic Pushdown Automata

- **Deterministic Pushdown Automata**
  - A pushdown automaton that never has a choice in its move

Deterministic Pushdown Automata

- **(Deterministic) Pushdown Automata: Formal definition**
  - A pushdown automaton (PDA) is a 7-tuple: $\boldsymbol{M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)}$
    - ❖ Q is a finite set of **internal states**
    - ❖ Σ is a finite set of **symbols**
    - ❖ Γ is a finite set of **symbols called stack alphabets**
    - ❖ δ is a set of **transition functions**
      - $\delta: Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow 2^{(Q \times \Gamma^*)}$
      - $\delta(q, a, b)$ contains at most one element ($q \in Q,\ a \in \Sigma \cup \{\lambda\},\ b \in \Gamma$)
      - If $\delta(q, \lambda, b)$ is not empty, then $\delta(q, c, b)$ must be empty for every $c \in \Sigma$
    - ❖ $q_0 \in Q$ is **the initial state**
    - ❖ $z \in \Gamma$ is **the initial stack alphabet**
    - ❖ $F \subseteq Q$ is a set of **final states**

Deterministic Pushdown Automata

- **(Deterministic) Pushdown Automata: Formal definition**
  - A pushdown automaton (PDA) is a 7-tuple: $\boldsymbol{M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)}$
    - Q is a finite set of **internal states**
    - Σ is a finite set of **symbols**
    - Γ is a finite set of **symbols called stack alphabets**
    - δ is a set of **transition functions**

For any given input symbol and any stack top, at most one move can be made

- $\delta(q, a, b)$ contains at most one element ($q \in Q,\ a \in \Sigma \cup \{\lambda\},\ b \in \Gamma$)
- If $\delta(q, \lambda, b)$ is not empty, then $\delta(q, c, b)$ must be empty for every $c \in \Sigma$

    - $q_0 \in Q$ is **the initial state**
    - $z \in \Gamma$ is **the initial stack alphabet**
    - $F \subseteq Q$ is a set of **final states**

Deterministic Pushdown Automata

- **(Deterministic) Pushdown Automata: Formal definition**
  - A pushdown automaton (PDA) is a 7-tuple: $M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$
    - Q is a finite set of **internal states**
    - Σ is a finite set of **symbols**
    - Γ is a finite set of **symbols called stack alphabets**
    - δ is a set of **transition functions**
      - $\delta: Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to 2^{(Q \times \Gamma^*)}$

When a λ-move is possible for some configuration, no input-consuming alternatives is available

      - If $\delta(q, \lambda, b)$ is not empty, then $\delta(q, c, b)$ must be empty for every $c \in \Sigma$
    - $q_0 \in Q$ is **the initial state**
    - $z \in \Gamma$ is **the initial stack alphabet**
    - $F \subseteq Q$ is a set of **final states**

Deterministic Pushdown Automata

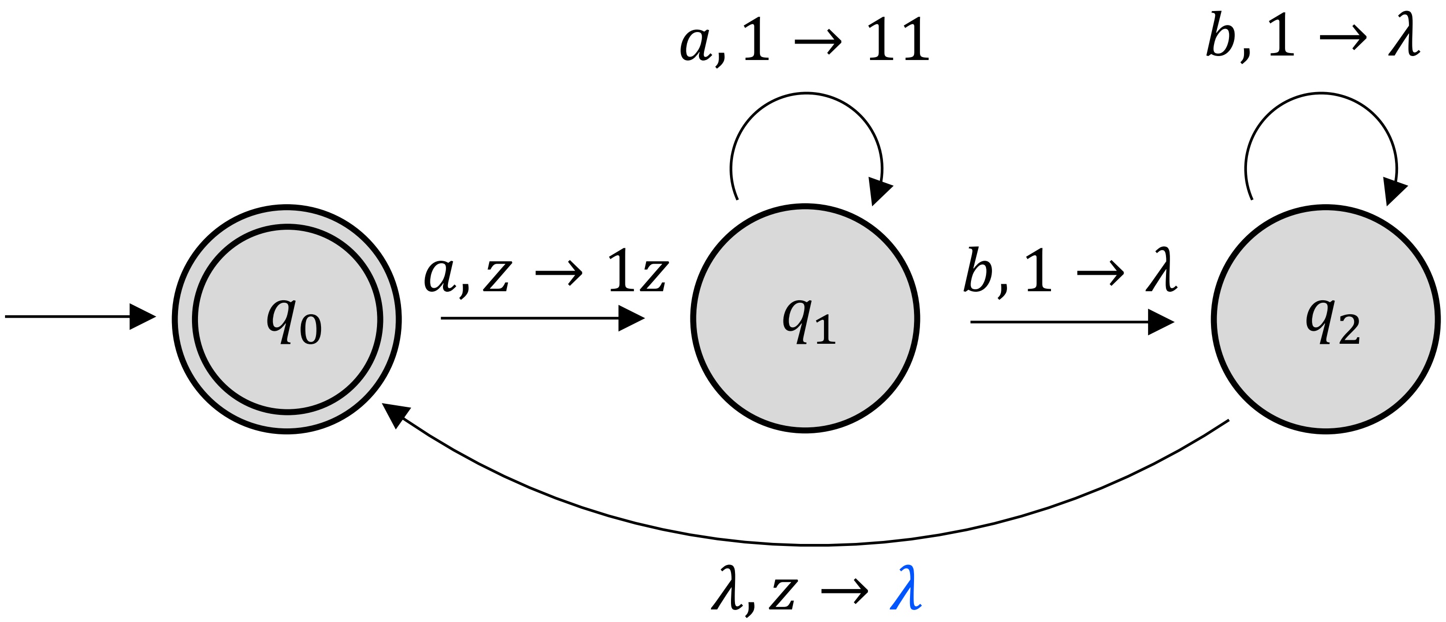
- **Difference between finite automata**
  - DFA
    - No λ-transition is allowed
    - No dead configuration
    - A DFA is equivalent in expressive power to an NFA
  - DPDA
    - λ-transition is possible
      - The top of the stack plays a role in determining the next move
      - The presence of λ-transition does not imply nondeterminism
    - Some transitions of a DPDA may be to the empty set
      - Dead configuration may occur
      - The only criterion for determinism is that at all times at most one possible move exists
    - DPDA and NPDA may not be equivalent

Deterministic Pushdown Automata

- **A language L is said to be a deterministic context-free language if and only if there exists a DPDA M such that $L = L(M)$**

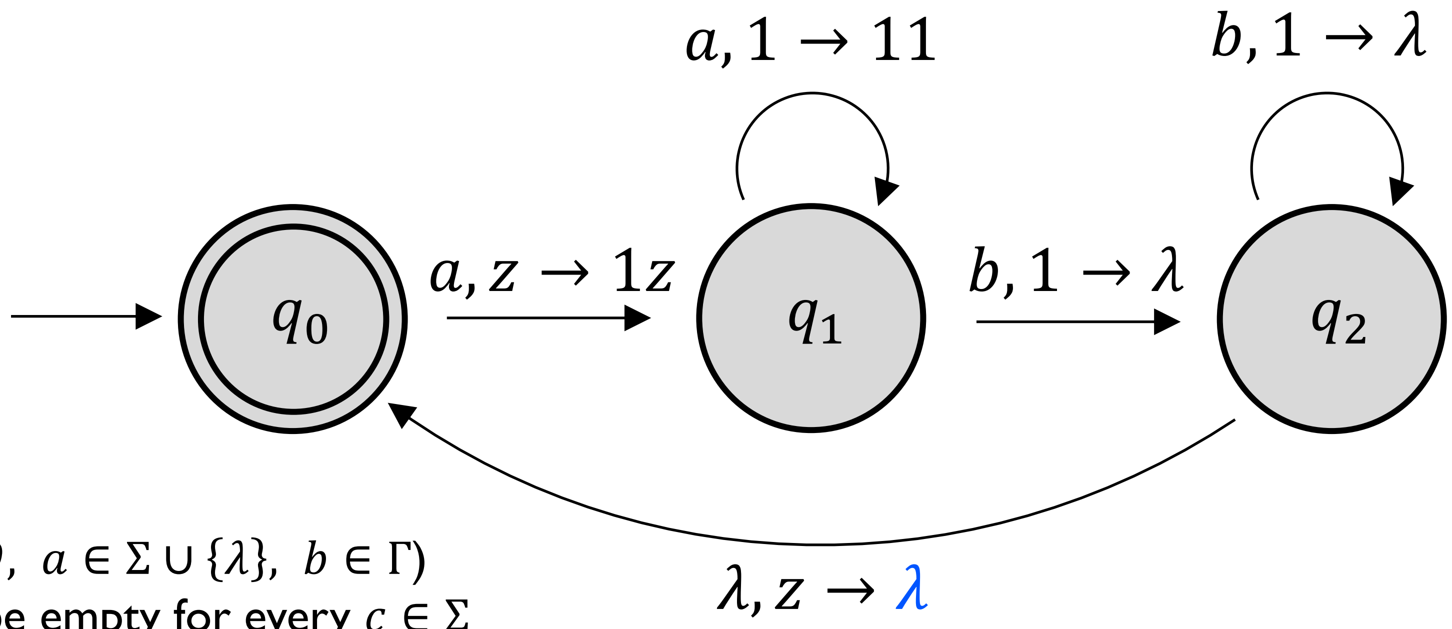
Deterministic Pushdown Automata

- **DPDA: example**
  - $\boldsymbol{L = \{a^n b^n \mid n \geq 0\}}$
  - $M = (\{q_0, q_1, q_2\}, \{a, b\}, \{0, 1\}, \delta, q_0, z, \{q_0\})$
    - ❖ $\delta(q_0, a, z) = \{(q_1, 1z)\}$
    - ❖ $\delta(q_1, a, 1) = \{(q_1, 11)\}$
    - ❖ $\delta(q_1, b, 1) = \{(q_2, \lambda)\}$
    - ❖ $\delta(q_2, b, 1) = \{(q_2, \lambda)\}$
    - ❖ $\delta(q_2, \lambda, z) = \{(q_0, \lambda)\}$



Deterministic Pushdown Automata

- **DPDA: example**
  - $\boldsymbol{L = \{a^n b^n \mid n \geq 0\}}$
  - $M = (\{q_0, q_1, q_2\}, \{a, b\}, \{0, 1\}, \delta, q_0, z, \{q_0\})$
    - ❖ $\delta(q_0, a, z) = \{(q_1, 1z)\}$
    - ❖ $\delta(q_1, a, 1) = \{(q_1, 11)\}$
    - ❖ $\delta(q_1, b, 1) = \{(q_2, \lambda)\}$
    - ❖ $\delta(q_2, b, 1) = \{(q_2, \lambda)\}$
    - ❖ $\delta(q_2, \lambda, z) = \{(q_0, \lambda)\}$



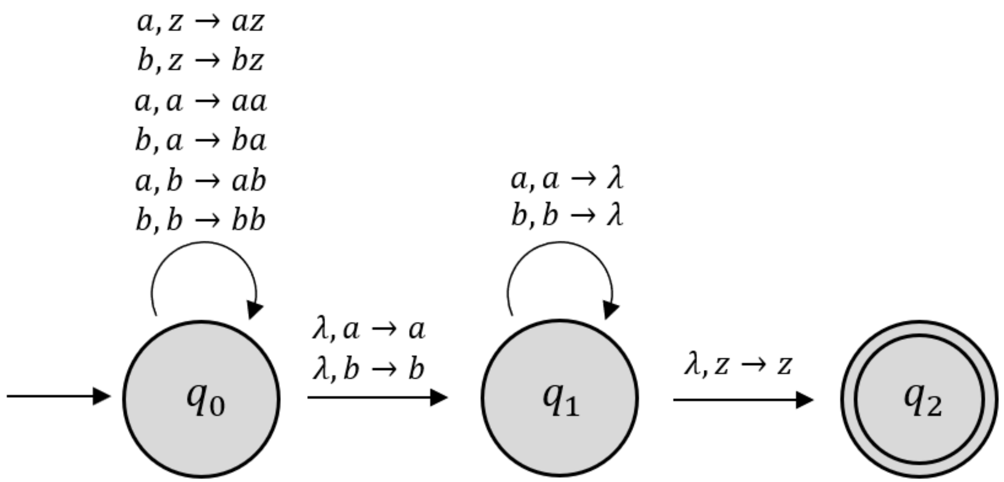
- $\delta(q, a, b)$ contains at most one element ($q \in Q,\ a \in \Sigma \cup \{\lambda\},\ b \in \Gamma$)
- If $\delta(q, \lambda, b)$ is not empty, then $\delta(q, c, b)$ must be empty for every $c \in \Sigma$

Deterministic Pushdown Automata

- **DPDA: example**

Pushdown Automata

- **Another example:** Design a PDA for $L = \{ww^R : w \in \{a,b\}^*\}$
  - $M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, b, z\}, \delta, q_0, z, \{q_2\})$

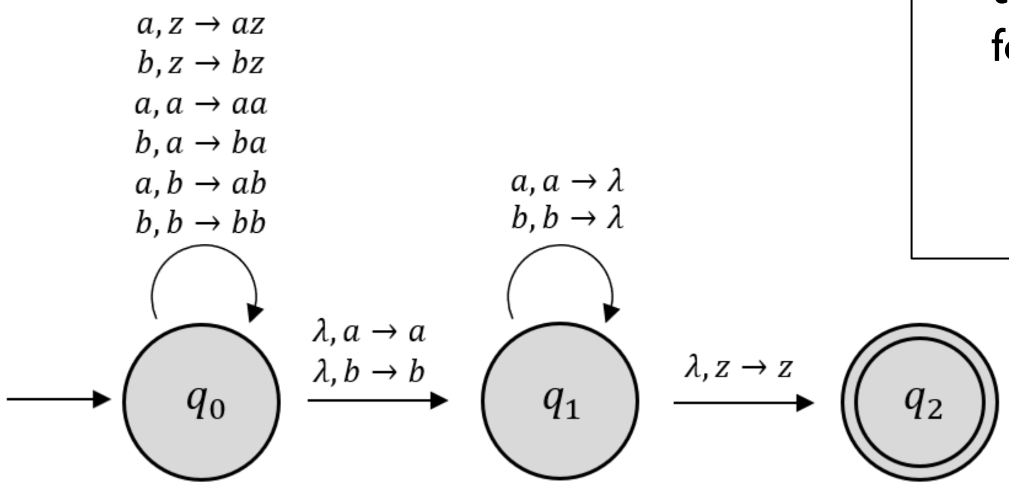


Deterministic Pushdown Automata

- **DPDA: example**

Pushdown Automata

- **Another example:** Design a PDA for $L = \{ww^R : w \in \{a,b\}^*\}$
  - $M = (\{q_0, q_1, q_2\}, \{a,b\}, \{a,b,z\}, \delta, q_0, z, \{q_2\})$



- If $\delta(q, \lambda, b)$ is not empty, then $\delta(q, c, b)$ must be empty for every $c \in \Sigma$
  - $\delta(q_0, a, a) = \{(q_0, aa)\}$
  - $\delta(q_0, \lambda, a) = \{(q_1, a)\}$

This example is not DPDA!

Deterministic Pushdown Automata

- **However, this does not imply that $\{ww^R\}$ is nondeterministic**
  - There may be a DPDA!

Deterministic Pushdown Automata

- **DPDA: practice**
  - $\boldsymbol{L = \{wcw^R \mid w \in \{a, b\}^+\}} \quad (\Gamma = \{\mathbf{0}, \mathbf{1}, \boldsymbol{z}\})$

- $\delta(q, a, b)$ contains at most one element ($q \in Q,\ a \in \Sigma \cup \{\lambda\},\ b \in \Gamma$)
- If $\delta(q, \lambda, b)$ is not empty, then $\delta(q, c, b)$ must be empty for every $c \in \Sigma$

Deterministic Pushdown Automata

- **DPDA: practice**
  - $L = \{wcw^R \mid w \in \{a, b\}^+\}$

Parsing Efficiency

- **The importance of deterministic CFL**
  - They can be parsed efficiently
    - Only one choice
- **Assume that we derive the leftmost derivation of a sentence**
  - If we can determine which production rule to apply at each step, the efficiency of parsing becomes significantly higher

Parsing Efficiency

- **LL grammar**
  - Main characteristic
    - By looking at a limited part of the input, we can predict which production rule must be used
  - The first L indicates that the input is scanned from left to right
  - The second L indicates that leftmost derivations are constructed

Parsing Efficiency

- **LL grammar: example**
  - $S \to aSb \mid ab$

Parsing Efficiency

- **LL grammar: example**
  - $S \rightarrow aSb \mid ab$
  - To determine which production rule to apply, examine the first two symbols of the given input string

Parsing Efficiency

- **LL grammar: example**
  - $S \rightarrow aSb \mid ab$
  - To determine which production rule to apply, examine the first two symbols of the given input string
  - If the second symbol is 'b,' we should apply $S \rightarrow ab$
  - If the second symbol is 'a,' we should apply $S \rightarrow aSb$

Parsing Efficiency

- **LL grammar: example**
  - $S \rightarrow aSb \mid ab$

$$\boldsymbol{aaabbb}$$

Parsing Efficiency

- **LL grammar: example**
  - $S \to aSb \mid ab$

$$\underline{aa}abbb$$

$$S \to aSb$$

Parsing Efficiency

- **LL grammar: example**
  - $S \rightarrow aSb \mid ab$

$$\cancel{a}\underline{aa}bbb$$

$$S \rightarrow aSb$$

Parsing Efficiency

- **LL grammar: example**
  - $S \to aSb \mid ab$

$$\cancel{aa}\underline{ab}bb$$

$$S \to ab$$

Parsing Efficiency

- **LL grammar: example**
  - $S \rightarrow aSb \mid ab$
  - $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$

Parsing Efficiency

- **LL grammar: example**
  - $S \to aSb \mid ab$
  - $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$
  - A grammar is an LL(k) grammar if we can uniquely identify the correct production, given the currently scanned symbol and a "look ahead" of the next k-1 symbols
  - This is an example of an LL(2) grammar

Parsing Efficiency

- **LL grammar: example**
  - $S \rightarrow SS \mid aSb \mid ab$
  - Is this an LL grammar?

Parsing Efficiency

- **LL grammar: example**
  - $S \to SS \mid aSb \mid ab$
  - Is this an LL grammar?
    - NO!
  - If the first symbol is 'a,' we do not know which rule to be used, $S \to SS$ or $S \to aSb$

Parsing Efficiency

- **LL grammar: example**
  - $S \to SS \mid aSb \mid ab$
  - Is this an LL grammar?
    - NO!
  - If the first symbol is 'a,' we do not know which rule to be used, $S \to SS$ or $S \to aSb$
  - Even if we look at first two symbols..
    - $aabb$ $aabbab$
    - We do not know which rule to be used, $S \to SS$ or $S \to aSb$

Parsing Efficiency

- **LL grammar: example**
  - $S \rightarrow SS \mid aSb \mid ab$
  - Is this an LL grammar?
    - NO!
  - But we can generate an LL grammar equivalent to the original grammar
    - $S' \rightarrow aSbS$
    - $S \rightarrow aSbS \mid \lambda$

Parsing Efficiency

- **LL grammar: example**
  - $S \rightarrow SS \mid aSb \mid ab$
  - Is this an LL grammar?
    - NO!
  - But we can generate an LL grammar equivalent to the original grammar
    - $S' \rightarrow aSbS$
    - $S \rightarrow aSbS \mid \lambda$
  - More details about LL grammar will be introduced in the compiler class! (hopefully..)

Next Lecture

- **Properties of Context-free Languages**