COM3064 Automata Theory

Week 8: Equivalence of PDA's and CFG's and Closure Properties of CFG's

Lecturer: Dr. Sevgi YİĞİT SERT
Spring 2023

Resources: Introduction to The Theory of Computation, M. Sipser,
Introduction to Automata Theory, Languages, and Computation, J.E. Hopcroft, R. Motwani, and J.D. Ullman
BBM401 Automata Theory and Formal Languages, İlyas Çiçekli

Equivalence of PDA's and CFG's

- Context-free grammars and pushdown automata are equivalent in power.
- Both are capable of describing the class of context-free languages.

- We show how to convert any context-free grammar into a pushdown automata that recognize the same language and vice versa.

From Grammars to Pushdown Automata

- Given a CFG G, **we construct a PDA that simulates the leftmost derivations of G**.
- Any left-sentential form can be written as $\boldsymbol{xA\alpha}$, where
  - $\boldsymbol{A}$ is the leftmost variable,
  - $\boldsymbol{x}$ is whatever terminals appear to the left of $\boldsymbol{A}$, and
  - $\boldsymbol{\alpha}$ is the string of terminals and variables that appear to the right of $\boldsymbol{A}$.
  - If a left-sentential form consists of terminals only, then $\boldsymbol{A\alpha}$ is $\boldsymbol{\varepsilon}$**.**

- Suppose the PDA is in an **ID** $(\boldsymbol{q, y, A\alpha})$, representing left-sentential form $\boldsymbol{xA\alpha}$. Let $\boldsymbol{xA\alpha \Rightarrow_{lm} x\beta\alpha}$ be a derivation step from the left-sentential form $\boldsymbol{xA\alpha}$**.**

  - This derivation step says that the production rule is $\boldsymbol{A \rightarrow \beta}$.
  - The move of the PDA is to replace $\boldsymbol{A}$ on the top of the stack by $\boldsymbol{\beta}$, entering ID $(\boldsymbol{q, y, \beta\alpha})$. Note that there is only one state, **q**, for this PDA.

From Grammars to Pushdown Automata

PDA Construction from CFG

PDA Construction from CFG:

- Let $G = (V, T, R, S)$ be a CFG.
- Construct the PDA P that accepts L(G) as follows:

$$P = (\{q\}, T, V \cup T, \delta, q, S),$$

where transition function δ is defined by:

1. For each variable A,

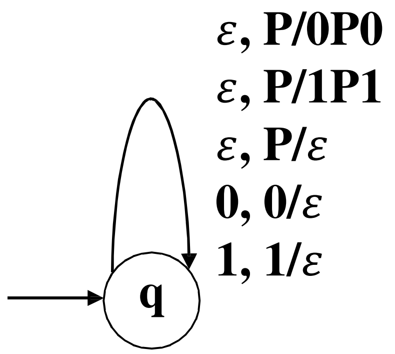
   $\delta(q, \varepsilon, A) = \{ (q, \beta) \mid A \rightarrow \beta$ is a production of G $\}$

2. For each terminal a, $\delta(q, a, a) = \{(q, \varepsilon)\}$

From CFG to PDA - Example

- Let CFG G = ({P}, {0,1}, {P $\rightarrow$ 0P0, P $\rightarrow$ 1P1, P $\rightarrow$ ε}, P)
- L(G) is { ww^r : w $\in$ $\{0,1\}^*$ }

- **We can construct PDA *A* = ({q}, {0,1}, {0,1,P}, δ, q, P) with following transitions.**



$\boldsymbol{\delta}$**(q, ε, P) = { (q,0P0), (q,1P1), (q, ε) }**

$\boldsymbol{\delta}$**(q, 0, 0) = { (q, ε) }**

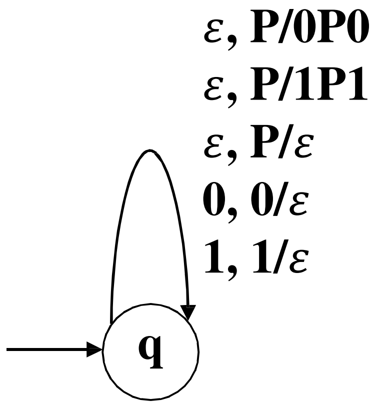
$\boldsymbol{\delta}$**(q, 1, 1) = { (q, ε) }**

From CFG to PDA – Example

$\mathbf{P} \rightarrow \mathbf{0P0}$

$\mathbf{P} \rightarrow \mathbf{1P1}$

$\mathbf{P} \rightarrow \varepsilon$



- A computation sequence for 0110:

  $(q,0110,P) \vdash (q,0110,0P0) \vdash (q,110,P0)$

- A leftmost derivation sequence of 0111:

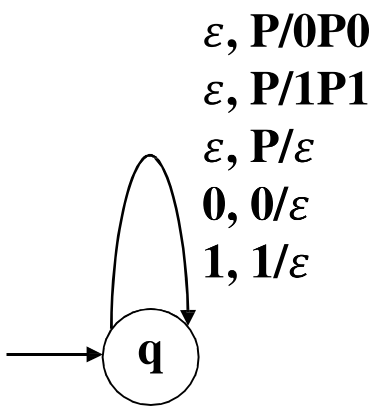
  $P \Rightarrow 0P0$

From CFG to PDA – Example

P → 0P0

P → 1P1

P → ε



- A computation sequence for 0110:

  (q,0110,P) ⊢ (q,0110,0P0) ⊢ (q,110,P0) ⊢ (q,110,1P10) ⊢ (q,10,P10)

- A leftmost derivation sequence of 0111:

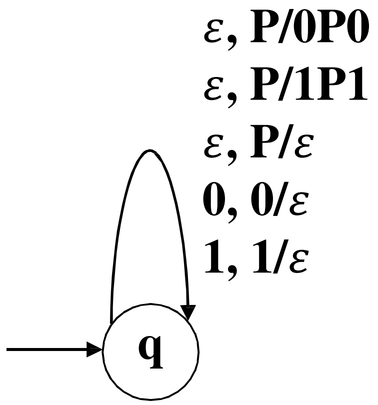
  P ⇒ 0P0 ⇒ 01P10

From CFG to PDA – Example

P → 0P0

P → 1P1

P → ε



- A computation sequence for 0110:

  (q,0110,P) ⊢ (q,0110,0P0) ⊢ (q,110,P0) ⊢ (q,110,1P10) ⊢ (q,10,P10) ⊢ (q,10,10) ⊢ (q,0,0) ⊢ (q, ε, ε)

- A leftmost derivation sequence of 0111:

  $P \Rightarrow 0P0 \Rightarrow 01P10 \Rightarrow 0110$

From a PDA to a CFG

Construct a CFG G=(V,T,R,S) from PDA P=(Q,Σ,Γ,δ,q_0,Z_0) such that L(P) = L(G).

Variables of G:

- G's variables are of the form $[\boldsymbol{pXq}]$ where $\boldsymbol{p \in Q}$, $\boldsymbol{q \in Q}$, $\boldsymbol{X \in \Gamma}$,
  - The variable $[\boldsymbol{pXq}]$ generates all and only the strings **w** such that $(\boldsymbol{p}, \boldsymbol{w}, \boldsymbol{X}) \vdash^* (\boldsymbol{q}, \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon})$.
- In addition, G will also have a start symbol S.
- Thus, $\boldsymbol{V} = \{\ [\boldsymbol{pXq}] : \boldsymbol{p} \in \mathrm{Q}, \boldsymbol{q} \in \mathrm{Q}, \boldsymbol{X} \in \boldsymbol{\Gamma}\} \cup \{\mathrm{S}\}$

From a PDA to a CFG

Construct a CFG $G = (V, T, R, S)$ from PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$, such that L(P) = L(G).

Productions of G:

- Each production for $[pXq]$ comes from a move of P in state p with stack symbol X.

CASE 1: $\delta(p, a, X)$ contains (q, ε).

- G has the production $[pXq] \rightarrow a$ where $a \in \Sigma \cup \{\varepsilon\}$.
  - $[pXq]$ generates a, because reading a is one way to pop X and go from p to q.

CASE 2: $\delta(p, a, X)$ contains (r, Y) for some state **r** and a stack symbol Y.

- For each state $q \in Q$, G has the production $[pXq] \rightarrow a[rYq]$ where $a \in \Sigma \cup \{\varepsilon\}$.
  - We can erase X and go from p to q by reading a (entering state r and replacing X by Y) and then reading some **w** that gets P from r to q while erasing Y.
  - Note: $[pXq] \Rightarrow^* aw$ whenever $[rYq] \Rightarrow^* w$.

From a PDA to a CFG

Productions of G:

GENERAL CASE: $\boldsymbol{\delta(p, a, X)}$ contains $(\boldsymbol{r, Y_1 \ldots Y_k})$ for some state **r** and k≥2.

- Generate a family of productions (For all states **q, s_1, …, s_{k-1}**)

$$[\boldsymbol{pXq}] \rightarrow \boldsymbol{a}\ [\boldsymbol{rY_1s_1}]\ [\boldsymbol{s_1Y_2s_2}]\ \ldots\ [\boldsymbol{s_{k-2}Y_{k-1}s_{k-2}}]\ [\boldsymbol{s_{k-1}Y_kq}]$$

When k=2: $\boldsymbol{\delta(p, a, X)}$ contains $(\boldsymbol{r, YZ})$ for some state **r**

- Now, $\boldsymbol{P}$ has replaced **X** by $\boldsymbol{YZ}$.
- To have the net effect of erasing $\boldsymbol{X}$, $\boldsymbol{P}$ must erase $\boldsymbol{Y}$, going from state **r** to some state **s**, and then erase $\boldsymbol{Z}$, going from $\boldsymbol{s}$ to $\boldsymbol{q}$.
- Since we do not know state **s**, we must generate a family of production rules:

  $[\boldsymbol{pXq}] \rightarrow \boldsymbol{a}[\boldsymbol{rYs}][\boldsymbol{sZq}]$ for all states $\boldsymbol{s}$.

- Note: $[\boldsymbol{pXq}] \Rightarrow^* \boldsymbol{awx}$ whenever $[\boldsymbol{rYs}] \Rightarrow^* \boldsymbol{w}$ and $[\boldsymbol{sZq}] \Rightarrow^* \boldsymbol{x}$.

From a PDA to a CFG

Productions of G:

- For all states p, G has the production $\boldsymbol{S} \rightarrow [\boldsymbol{q_0 Z_0 p}]$.
- Our intuition that a symbol like $[\boldsymbol{q_0 Z_0 p}]$ is intended to generate all those strings w that cause P to pop $\boldsymbol{Z_0}$ from its stack while going from state $\boldsymbol{q_0}$ to state $\boldsymbol{p}$. That is, $(\boldsymbol{q_0}, \boldsymbol{w}, \boldsymbol{Z_0}) \vdash^* (\boldsymbol{p}, \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon})$.
- If so, then these productions say that start symbol $\boldsymbol{S}$ will generate all strings $\boldsymbol{w}$ that cause P to empty its stack, after starting in its initial ID.
- But state $\boldsymbol{p}$ can be any state.
- Thus, add productions $\boldsymbol{S} \rightarrow [\boldsymbol{q_0 Z_0 p}]$ for each state $\boldsymbol{p}$.

- If there were n states of the PDA, then there would be n production rules of this type, since the last state could be any of the n states.

From a PDA to a CFG: Example

- Convert the PDA $P = (\{p, q\}, \{0,1\}, \{X, Z_0\}, \delta, q, Z_0)$ where $\boldsymbol{\delta}$ is given by:

1. $\delta(q, 1, Z_0) = \{(q, XZ_0)\}$
2. $\delta(q, 1, X) = \{(q, XX)\}$
3. $\delta(q, 0, X) = \{(p, X)\}$
4. $\delta(q, \epsilon, X) = \{(q, \epsilon)\}$
5. $\delta(p, 1, X) = \{(p, \epsilon)\}$
6. $\delta(p, 0, Z_0) = \{(q, Z_0)\}$

From a PDA to a CFG: Example

$$G = (\{[pXp],[pXq],[pZ_0p],[pZ_0q],[qXp],[qXq],[qZ_0p],[qZ_0q],S\},\{0,1\},R,S)$$

and productions in R are:

$S \rightarrow [qZ_0q] | [qZ_0p]$

From 1. $\delta(q,1,Z_0) = \{(q,XZ_0)\}$

$[qZ_0q] \rightarrow 1[qXq][qZ_0q]$
$[qZ_0q] \rightarrow 1[qXp][pZ_0q]$
$[qZ_0p] \rightarrow 1[qXq][qZ_0p]$
$[qZ_0p] \rightarrow 1[qXp][pZ_0p]$

From 2. $\delta(q,1,X) = \{(q,XX)\}$

$[qXq] \rightarrow 1[qXq][qXq]$
$[qXq] \rightarrow 1[qXp][pXq]$
$[qXp] \rightarrow 1[qXq][qXp]$
$[qXp] \rightarrow 1[qXp][pXp]$

From 3. $\delta(q,0,X) = \{(p,X)\}$

$[qXq] \rightarrow 0[pXq]$
$[qXp] \rightarrow 0[pXp]$

From 4. $\delta(q,\epsilon,X) = \{(q,\epsilon)\}$

$[qXq] \rightarrow \epsilon$

From 5. $\delta(p,1,X) = \{(p,\epsilon)\}$

$[pXp] \rightarrow 1$

From 6. $\delta(p,0,Z_0) = \{(q,Z_0)\}$

$[pZ_0q] \rightarrow 0[qZ_0q]$
$[pZ_0p] \rightarrow 0[qZ_0p]$

Deterministic Pushdown Automata (DPDA)

- While PDA's are by definition allowed to be nondeterministic, the deterministic subcase is quite important.
- In particular, parsers generally behave like deterministic PDA's, so the class of languages that can be accepted by these automata is interesting for the insights it gives us into what constructs are suitable for use in programming languages.

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, is deterministic iff

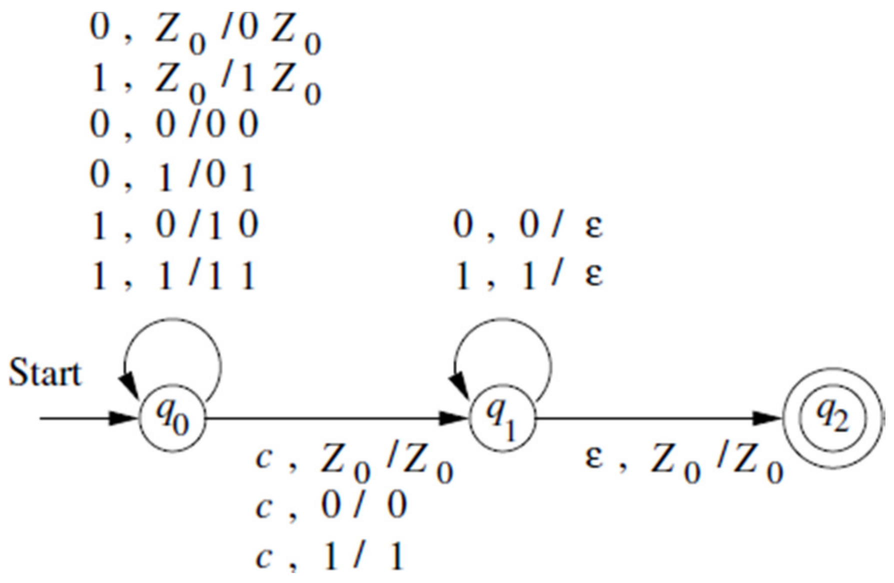
1. (q, a, X) is always empty or a singleton where $a \in \Sigma$, or a is ε.

2. If (q, a, X) is nonempty where $a \in \Sigma$, then (q, ε, X) must be empty.

DPDA - Example

$L = \{\ wcw^R : w \in \{0,1\}^*\ \}$

- L is recognized by the following DPDA:



DPDA Properties

- If L is a regular language, then L = L(P) for some DPDA P.
  - **regular languages ⊂ languages of DPDAs**
- There are languages that are not regular, but there are DPDAs for them.
  - $\{ wcw^R : w \in \{0,1\}^* \}$ is a member of languages of DPDAs but it is not regular.
- There are context free languages which are NOT members of languages of DPDAs.
  - $\{ ww^R : w \in \{0,1\}^* \}$ is NOT a member of languages of DPDAs. i.e. there is NO DPDA for the language $\{ ww^R : w \in \{0,1\}^* \}$.
  - **languages of DPDAs ⊂ context free languages**

Closure Properties of Context-Free Languages

- CFLs are closed under
  - Union
  - Concatenation
  - Kleene Closure
  - Reversal
  - Substitution

- CFLs are NOT closed under
  - Intersection
  - Difference
  - Complement

Closure of CFLs Under Union

- **CFLs are closed under Union.**

- Let L and M be CFL's with grammars G and H, respectively.
- Assume G and H have no variables in common.
  - Names of variables do not affect the language.
- Let S_1 and S_2 be the start symbols of G and H.
- Form a new grammar for L ∪ M by combining all the symbols and productions of G and H.
- Then, add a new start symbol S.
- Add productions $S \to S_1 \mid S_2$.
- In the new grammar, the first step replaces S by either S_1 or S_2.
- In the first case, the result must be a string in L(G) = L, and in the second case a string in L(H) = M.

Closure of CFLs Under Concatenation

- **CFLs are closed under Concatenation.**

- Let L and M be CFL's with grammars G and H, respectively.
- Assume G and H have no variables in common.
- Let S_1 and S_2 be the start symbols of G and H.
- Form a new grammar for LM by starting with all symbols and productions of G and H.
- Add a new start symbol S.
- Add production $S \rightarrow S_1S_2$.
- Every derivation from S results in a string in L followed by one in M.

Closure of CFLs Under Star

- **CFLs are closed under Star.**

- Let L have grammar G, with start symbol S_1.
- Form a new grammar for L^* by introducing to G a new start symbol S and the productions $S \rightarrow S_1S \mid \varepsilon$.
- A rightmost derivation from S generates a sequence of zero or more S_1's, each of which generates some string in L.