Computational Models - Lecture 6

Iftach Haitner.
Tel Aviv University.

November 19, 2018
Outline

- Closure under homomorphism and inverse homomorphism
- Algorithmic questions regrading CFLs
- Equivalence of CFGs and PDAs
- Sipser’s book, 2.1, 2.2&2.3
Pushdown Automata, reminder
Part I

Closure Under Homomorphism and Inverse homomorphism
Homomorphism, reminder

**Definition 1 (Homomorphism)**

An **homomorphism** from alphabet $\Delta$ to words over alphabet $\Sigma$, is a function $h: \Delta \mapsto \Sigma^*$.

For $w \in \Delta^*$, let $h(w = w_1, \ldots, w_n) = h(w_1) \cdots h(w_n)$.

For $L \subseteq \Delta^*$, let $h(L) = \{h(w): w \in L\}$.

By definition, $h(\varepsilon) = \varepsilon$ and $h(\emptyset) = \emptyset$.

**Examples:**

- Let $h: \{0, 1\} \mapsto \{a, b\}^*$ be defined by $h(1) = aba$ and $h(0) = aa$.
  
  $h(010) = aa \, aba \, aa$. For $L_1 = (01)^*$, $h(L_1) = (aaaba)^*$.

- Let $h(0) = a$, $h(1) = a$. For $L_2 = \{0^n1^n: n \geq 0\}$, $h(L_2) = \{a^{2n}: n \geq 0\}$. 
Closure under homomorphism

Theorem 2

Context free languages are closed under homomorphism.

Namely, $L$ is a CFL $\implies h(L)$ is a CFL.
Closure under homomorphism

Theorem 2

*Context free languages are closed under homomorphism.*

Namely, $L$ is a CFL $\implies h(L)$ is a CFL.

Proof:
Closure under homomorphism

Theorem 2

Context free languages are closed under homomorphism.

Namely, $\mathcal{L}$ is a CFL $\implies h(\mathcal{L})$ is a CFL.

Proof: use the grammar ...
Definition 3 (Inverse homomorphism)

For homomorphism $h: \Delta \rightarrow \Sigma^*$, define its inverse homomorphism $h^{-1}: \Sigma^* \rightarrow P(\Delta^*)$, by $h^{-1}(w) = \{ x \in \Delta^*: h(x) = w \}$.

For $\mathcal{L} \subseteq \Sigma^*$, let $h^{-1}(\mathcal{L}) = \bigcup_{x \in \mathcal{L}} h^{-1}(x) = \{ x \in \Delta^*: h(x) \in \mathcal{L} \}$

Example: $h(0) = a$, $h(1) = b$ and $h(2) = a$. Then $h^{-1}(\{ a^n b a^n : n \geq 0 \}) = \{ \{0, 2\}^n \{0, 2\}^n : n \geq 0 \}$. 
Closure under inverse homomorphism

Theorem 4

Context free languages are closed under inverse homomorphism.

Namely, $\mathcal{L}$ is a CFL $\implies h^{-1}(\mathcal{L})$ is a CFL.
Closure under inverse homomorphism

**Theorem 4**

*Context free languages are closed under inverse homomorphism.*

Namely, $\mathcal{L}$ is a CFL $\implies h^{-1}(\mathcal{L})$ is a CFL.

**Proof idea:** Let $\mathcal{L}$ be a CFL, let $P$ be a PDA for $\mathcal{L}$ and let $h: \Delta \mapsto \Sigma^*$. 

How do we implement $Buff$?

Iftach Haitner (TAU)  Computational Models, Lecture 6  November 19, 2018  8 / 35
**Closure under inverse homomorphism**

**Theorem 4**

*Context free languages are closed under inverse homomorphism.*

Namely, \( \mathcal{L} \) is a CFL \( \Rightarrow h^{-1}(\mathcal{L}) \) is a CFL.

**Proof idea:** Let \( \mathcal{L} \) be a CFL, let \( P \) be a PDA for \( \mathcal{L} \) and let \( h: \Delta \mapsto \Sigma^* \).

\[ w \in h^{-1}(\mathcal{L}) \iff h(w) \in \mathcal{L}(P). \]
Closure under inverse homomorphism

**Theorem 4**

*Context free languages are closed under inverse homomorphism.*

Namely, \( \mathcal{L} \) is a CFL \( \implies h^{-1}(\mathcal{L}) \) is a CFL.

**Proof idea:** Let \( \mathcal{L} \) be a CFL, let \( P \) be a PDA for \( \mathcal{L} \) and let \( h: \Delta \mapsto \Sigma^* \).

- \( w \in h^{-1}(\mathcal{L}) \iff h(w) \in \mathcal{L}(P) \).
- Hence, to decide \( w \) simply emulate \( P(h(w)) \)...

Algorithm 5 (PDA for \( h^{-1}(\mathcal{L}) \), informal)
---

**On input** \( w \):

1. Initiate a "buffer" Buff to \( h(a) \), where \( a \) is the first letter of \( w \).
2. Emulate a running of \( P \) with Buff as its input string. Each time Buff is fully read by \( P \), set Buff = \( h(a) \), where \( a \) is the next letter in \( w \) (if exists).
3. Accept iff \( P \) does...
Closure under inverse homomorphism

Theorem 4

*Context free languages are closed under inverse homomorphism.*

Namely, \( \mathcal{L} \) is a CFL \( \implies h^{-1}(\mathcal{L}) \) is a CFL.

**Proof idea:** Let \( \mathcal{L} \) be a CFL, let \( P \) be a PDA for \( \mathcal{L} \) and let \( h: \Delta \mapsto \Sigma^* \).

1. \( w \in h^{-1}(\mathcal{L}) \iff h(w) \in \mathcal{L}(P) \).
2. Hence, to decide \( w \) simply emulate \( P(h(w)) \)...
Closure under inverse homomorphism

**Theorem 4**

*Context free languages are closed under inverse homomorphism.*

Namely, \( \mathcal{L} \) is a CFL \( \implies h^{-1}(\mathcal{L}) \) is a CFL.

**Proof idea:** Let \( \mathcal{L} \) be a CFL, let \( P \) be a PDA for \( \mathcal{L} \) and let \( h: \Delta \mapsto \Sigma^* \).

- \( w \in h^{-1}(\mathcal{L}) \iff h(w) \in \mathcal{L}(P) \).
- Hence, to decide \( w \) simply emulate \( P(h(w)) \)....

**Algorithm 5 (PDA for \( h^{-1}(\mathcal{L}) \), informal)**

On input \( w \):

1. Initialized a “buffer” Buff to \( h(a) \), where \( a \) is the first letter of \( w \)

2. Emulate a running of \( P \) with Buff as its input string.

   Each time Buff is fully read by \( P \), set Buff = \( h(a) \), where \( a \) is the next letter in \( w \) (if exists)

3. Accept iff \( P \) does
Closure under inverse homomorphism

**Theorem 4**

*Context free languages are closed under inverse homomorphism.*

Namely, $\mathcal{L}$ is a CFL $\implies h^{-1}(\mathcal{L})$ is a CFL.

**Proof idea:** Let $\mathcal{L}$ be a CFL, let $P$ be a PDA for $\mathcal{L}$ and let $h: \Delta \mapsto \Sigma^*$.

- $w \in h^{-1}(\mathcal{L}) \iff h(w) \in \mathcal{L}(P)$.
- Hence, to decide $w$ simply emulate $P(h(w))$.

**Algorithm 5 (PDA for $h^{-1}(\mathcal{L})$, informal)**

On input $w$:

1. Initialized a “buffer” Buff to $h(a)$, where $a$ is the first letter of $w$.
2. Emulate a running of $P$ with Buff as its input string. Each time Buff is fully read by $P$, set Buff $\leftarrow h(a)$, where $a$ is the next letter in $w$ (if exists).
3. Accept iff $P$ does.

How do we implement Buff?
Closure under inverse homomorphism, the PDA definition

Given PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ and homomorphism $h: \Delta \rightarrow \Sigma^*$, define a new PDA $P' = (Q', \Delta, \Gamma, \delta', q'_0, F')$. 

Iftach Haitner (TAU)  Computational Models, Lecture 6  November 19, 2018  9 / 35
Closure under inverse homomorphism, the PDA definition

Given PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ and homomorphism $h: \Delta \rightarrow \Sigma^*$, define a new PDA $P' = (Q', \Delta, \Gamma, \delta', q'_0, F')$.

* Idea: we would like to have $\tilde{\delta}_{P'}(q, a, s) = \tilde{\delta}_P(q, h(a), s)$ (similar to what we did with DFAs)
Closure under inverse homomorphism, the PDA definition

Given PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ and homomorphism $h: \Delta \rightarrow \Sigma^*$, define a new PDA $P' = (Q', \Delta, \Gamma, \delta', q'_0, F')$.

- Idea: we would like to have $\tilde{\delta}_{P'}(q, a, s) = \tilde{\delta}_P(q, h(a), s)$ (similar to what we did with DFAs)

- Let $k = \max_{a \in \Delta} |h(a)|$ and $\tilde{\Sigma} = \bigcup_{0 \leq i \leq k} \Sigma^i$
Closure under inverse homomorphism, the PDA definition

Given PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ and homomorphism $h: \Delta \rightarrow \Sigma^*$, define a new PDA $P' = (Q', \Delta, \Gamma, \delta', q'_0, F')$.

* Idea: we would like to have $\hat{\delta}_{P'}(q, a, s) = \hat{\delta}_P(q, h(a), s)$ (similar to what we did with DFAs)

- Let $k = \max_{a \in \Delta} |h(a)|$ and $\tilde{\Sigma} = \cup_{0 \leq i \leq k} \Sigma^i$
- $Q' = Q \times \tilde{\Sigma}$
Closure under inverse homomorphism, the PDA definition

Given PDA \( P = (Q, \Sigma, \Gamma, \delta, q_0, F) \) and homomorphism \( h: \Delta \to \Sigma^* \), define a new PDA \( P' = (Q', \Delta, \Gamma, \delta', q'_0, F') \).

- Idea: we would like to have \( \hat{\delta}_{P'}(q, a, s) = \hat{\delta}_P(q, h(a), s) \) (similar to what we did with DFAs)

  - Let \( k = \max_{a \in \Delta} |h(a)| \) and \( \tilde{\Sigma} = \bigcup_{0 \leq i \leq k} \Sigma^i \)
  - \( Q' = Q \times \tilde{\Sigma} \)
  - \( q'_0 = [q_0, \varepsilon] \)
Closure under inverse homomorphism, the PDA definition

Given PDA \( P = (Q, \Sigma, \Gamma, \delta, q_0, F) \) and homomorphism \( h: \Delta \rightarrow \Sigma^* \), define a new PDA \( P' = (Q', \Delta, \Gamma, \delta', q'_0, F') \).

* Idea: we would like to have \( \hat{\delta}_{P'}(q, a, s) = \hat{\delta}_P(q, h(a), s) \) (similar to what we did with DFAs)

- Let \( k = \max_{a \in \Delta} |h(a)| \) and \( \tilde{\Sigma} = \bigcup_{0 \leq i \leq k} \Sigma^i \)
- \( Q' = Q \times \tilde{\Sigma} \)
- \( q'_0 = [q_0, \varepsilon] \)
- \( F' = F \times \{\varepsilon\} \)
Closure under inverse homomorphism, the PDA definition

Given PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ and homomorphism $h: \Delta \rightarrow \Sigma^*$, define a new PDA $P' = (Q', \Delta, \Gamma, \delta', q'_0, F')$.

* Idea: we would like to have $\tilde{\delta}_{P'}(q, a, s) = \tilde{\delta}_P(q, h(a), s)$ (similar to what we did with DFAs)

- Let $k = \max_{a \in \Delta} |h(a)|$ and $\tilde{\Sigma} = \bigcup_{0 \leq i \leq k} \Sigma^i$
- $Q' = Q \times \tilde{\Sigma}$
- $q'_0 = [q_0, \varepsilon]$
- $F' = F \times \{\varepsilon\}$
- $\delta'$ is defined as follows:

...
Closure under inverse homomorphism, the PDA definition

Given PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ and homomorphism $h: \Delta \rightarrow \Sigma^*$, define a new PDA $P' = (Q', \Delta, \Gamma, \delta', q'_0, F')$.

* Idea: we would like to have $\tilde{\delta}_{P'}(q, a, s) = \tilde{\delta}_P(q, h(a), s)$ (similar to what we did with DFAs)

► Let $k = \max_{a \in \Delta} |h(a)|$ and $\tilde{\Sigma} = \bigcup_{0 \leq i \leq k} \Sigma^i$

► $Q' = Q \times \tilde{\Sigma}$

► $q'_0 = [q_0, \varepsilon]$

► $F' = F \times \{\varepsilon\}$

► $\delta'$ is define as follows:
  ► $\delta'([q, \varepsilon], a, \varepsilon) = \{([q, h(a)], \varepsilon)\}$ (for $a \in \Delta$)
Closure under inverse homomorphism, the PDA definition

Given PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ and homomorphism $h: \Delta \rightarrow \Sigma^*$, define a new PDA $P' = (Q', \Delta, \Gamma, \delta', q'_0, F')$.

* Idea: we would like to have $\hat{\delta}_{P'}(q, a, s) = \hat{\delta}_P(q, h(a), s)$ (similar to what we did with DFAs)

  - Let $k = \max_{a \in \Delta} |h(a)|$ and $\tilde{\Sigma} = \bigcup_{0 \leq i \leq k} \Sigma^i$
  - $Q' = Q \times \tilde{\Sigma}$
  - $q'_0 = [q_0, \varepsilon]$
  - $F' = F \times \{\varepsilon\}$
  - $\delta'$ is defined as follows:
    - $\delta'([q, \varepsilon], a, \varepsilon) = \{([q, h(a)], \varepsilon)\}$ \hspace{1cm} \((a \in \Delta)\)
    - If $(p, \gamma) \in \delta(q, \varepsilon, \xi)$, then $(p, x, \gamma) \in \delta'([q, x], \varepsilon, \xi)$ \hspace{1cm} \((x \in \tilde{\Sigma})\)
Closure under inverse homomorphism, the PDA definition

Given PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ and homomorphism $h: \Delta \to \Sigma^*$, define a new PDA $P' = (Q', \Delta, \Gamma, \delta', q'_0, F')$.

* Idea: we would like to have $\hat{\delta}_{P'}(q, a, s) = \hat{\delta}_{P}(q, h(a), s)$ (similar to what we did with DFAs)

- Let $k = \max_{a \in \Delta} |h(a)|$ and $\tilde{\Sigma} = \cup_{0 \leq i \leq k} \Sigma^i$
- $Q' = Q \times \tilde{\Sigma}$
- $q'_0 = [q_0, \varepsilon]$
- $F' = F \times \{\varepsilon\}$
- $\delta'$ is define as follows:
  - $\delta'([q, \varepsilon], a, \varepsilon) = \{([q, h(a)], \varepsilon)\}$
  - If $(p, \gamma) \in \delta(q, \varepsilon, \xi)$, then $([p, x], \gamma) \in \delta'(q, x, \varepsilon, \xi)$ ($x \in \tilde{\Sigma}$)
  - If $(p, \gamma) \in \delta(q, a, \xi)$, then $([p, x], \gamma) \in \delta'(q, ax, \varepsilon, \xi)$ ($a \in \Sigma, x \in \tilde{\Sigma}$)
Part II

Algorithmic Questions
Emptiness of CFGs

Question 6
Given a CFG, $G$, is $\mathcal{L}(G) = \emptyset$?
**Question 6**

Given a CFG, $G$, is $L(G) = \emptyset$?

In other words, is there a string generated by $G$?
Emptiness of CFGs

**Question 6**
Given a CFG, $G$, is $L(G) = \emptyset$?

In other words, is there a string generated by $G$?

**Theorem 7**

*There is an algorithm that solves this problem (and always halts).*
**Emptiness of CFGs**

**Question 6**
Given a CFG, $G$, is $\mathcal{L}(G) = \emptyset$?

In other words, is there a string generated by $G$?

**Theorem 7**
*There is an algorithm that solves this problem (and always halts).*

Possible approaches for a proof:

- **Bad Idea**: We know how to test whether $w \in \mathcal{L}(G)$ for any string $w$, so just try it for each $w$...
Question 6
Given a CFG, $G$, is $L(G) = \emptyset$?

In other words, is there a string generated by $G$?

Theorem 7
There is an algorithm that solves this problem (and always halts).

Possible approaches for a proof:

- **Bad Idea**: We know how to test whether $w \in L(G)$ for any string $w$, so just try it for each $w$...

- **Better Idea**: Can the start variable generate a string of terminals?
Emptiness of CFGs

**Question 6**
Given a CFG, $G$, is $\mathcal{L}(G) = \emptyset$?

In other words, is there a string generated by $G$?

**Theorem 7**
*There is an algorithm that solves this problem (and always halts).*

Possible approaches for a proof:

- **Bad Idea:** We know how to test whether $w \in \mathcal{L}(G)$ for any string $w$, so just try it for each $w$...

- **Better Idea:** Can the start variable generate a string of terminals?

- **A more holistic approach:** Can a particular variable generate a string of terminals?
Checking emptiness

Idea: Mark variables that can produce a string of terminals

**Algorithm 8 (Deciding $\mathcal{L}(G) = \emptyset$)**

1. Mark all terminal symbols in $G$.
2. Repeat until no new variable become marked:
   - Mark any $A$ where $A \rightarrow U_1 U_2 \ldots U_k$ and all $U_i$ have already been marked.
3. Remove all unmarked variables, and any rule they appear in.
4. If $S$ is removed, then $\mathcal{L}(G) = \emptyset$. 
Checking emptiness

Idea: Mark variables that can produce a string of terminals

**Algorithm 8 (Deciding \( \mathcal{L}(G) = \emptyset \))**

1. Mark all terminal symbols in \( G \).
2. Repeat until no new variable become marked:
   - Mark any \( A \) where \( A \rightarrow U_1 U_2 \ldots U_k \) and all \( U_i \) have already been marked.
3. Remove all unmarked variables, and any rule they appear in.
4. If \( S \) is removed, then \( \mathcal{L}(G) = \emptyset \).

▶ Termination?

Iftach Haitner (TAU)  Computational Models, Lecture 6  November 19, 2018  12 / 35
Checking emptiness

Idea: Mark variables that can produce a string of terminals

Algorithm 8 (Deciding $\mathcal{L}(G) = \emptyset$)

1. Mark all terminal symbols in $G$.
2. Repeat until no new variable become marked:
   - Mark any $A$ where $A \rightarrow U_1 U_2 \ldots U_k$ and all $U_i$ have already been marked.
3. Remove all unmarked variables, and any rule they appear in.
4. If $S$ is removed, then $\mathcal{L}(G) = \emptyset$.

- Termination?
- Correctness?
Question 9
Given a CFG $G$, is $L(G) = \Sigma^*$?
Question 9

Given a CFG $G$, is $\mathcal{L}(G) = \Sigma^*$?

- We just saw an algorithm to determine, given a CFG $G$, whether $\mathcal{L}(G) = \emptyset$.
Question 9

Given a CFG $G$, is $\mathcal{L}(G) = \Sigma^*$?

- We just saw an algorithm to determine, given a CFG $G$, whether $\mathcal{L}(G) = \emptyset$.

- $\mathcal{L}(G) = \Sigma^*$ iff $\overline{\mathcal{L}(G)} = \emptyset$. Why not modify the algorithm so it determines emptiness of the complement?
Question 9

Given a CFG $G$, is $L(G) = \Sigma^*$?

- We just saw an algorithm to determine, given a CFG $G$, whether $L(G) = \emptyset$.
- $L(G) = \Sigma^*$ iff $\overline{L(G)} = \emptyset$. Why not modify the algorithm so it determines emptiness of the complement?
- Unfortunately, CFGs are not closed under complement.
Question 9

Given a CFG $G$, is $\mathcal{L}(G) = \Sigma^*$?

- We just saw an algorithm to determine, given a CFG $G$, whether $\mathcal{L}(G) = \emptyset$

- $\mathcal{L}(G) = \Sigma^*$ iff $\overline{\mathcal{L}(G)} = \emptyset$. Why not modify the algorithm so it determines emptiness of the complement?

- Unfortunately, CFGs are not closed under complement.
Question 9
Given a CFG $G$, is $L(G) = \Sigma^*$?

- We just saw an algorithm to determine, given a CFG $G$, whether $L(G) = \emptyset$.
- $L(G) = \Sigma^*$ iff $\overline{L(G)} = \emptyset$. Why not modify the algorithm so it determines emptiness of the complement?
- Unfortunately, CFGs are not closed under complement.

Fact 10
There is no algorithm to solve CFG fullness.
Question 9

Given a CFG $G$, is $\mathcal{L}(G) = \Sigma^*$?

- We just saw an algorithm to determine, given a CFG $G$, whether $\mathcal{L}(G) = \emptyset$.

- $\mathcal{L}(G) = \Sigma^*$ iff $\overline{\mathcal{L}(G)} = \emptyset$. Why not modify the algorithm so it determines emptiness of the complement?

- Unfortunately, CFGs are not closed under complement.

Fact 10

There is no algorithm to solve CFG fullness.

- We are not prepared to prove this remarkable fact (yet).
Finiteness of CFGs

Question 11
Given a CFG $G$, is $|\mathcal{L}(G)|$ finite?
Finiteness of CFGs

Question 11
Given a CFG $G$, is $|\mathcal{L}(G)|$ finite?

First, a useful subroutine.

Algorithm 12 (Removing redundant variables and terminals)

1. Mark all terminal symbols in $G$.
2. Repeat until no new variable become marked:
   - Mark any $A$ where $A \rightarrow U_1 U_2 \ldots U_k$ and all $U_i$ have already been marked.
3. Remove all unmarked variables, and any rule they appear in.
4. If $S$ is removed, then $\mathcal{L}(G) = \emptyset$.
5. Remove any variable $A$ not reachable from $S$.
6. Remove any terminal which does not appear in some rule.
### Question 13

Given a CFG $G$, is $|\mathcal{L}(G)|$ finite?

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Remove redundant variables and terminals.</td>
</tr>
<tr>
<td>2.</td>
<td>Turn into a CNF.</td>
</tr>
</tbody>
</table>
| 3.   | Create a graph $C$:  
  - Add node $v_A$ for each variable $A \in V$.  
  - Add directed edges $(v_A, v_B)$ and $(v_A, v_C)$, for each rule $(A \rightarrow BC) \in R$. |
| 4.   | Return TRUE iff $C$ has no cycles. |

**Correctness?**

Less efficient algorithm "using" the pumping lemma:  

- $|\mathcal{L}(G)| = \infty$ iff $\exists w \in \mathcal{L}(G)$ with $\ell \leq |w| \leq 2\ell$ (\(\ell\) is the pumping length)

- Hence, it suffices to check whether $\mathcal{L}(G)$ has a word on such length.
Given a CFG $G$, is $|\mathcal{L}(G)|$ finite?

Algorithm 14

1. Remove redundant variables and terminals.
2. Turn into a CNF.
3. Create a graph $C$:
   - Add node $v_A$ for each variable $A \in V$.
   - Add directed edges $(v_A, v_B)$ and $(v_A, v_C)$, for each rule $(A \rightarrow BC) \in R$.
4. Return $\text{TRUE}$ iff $C$ has no cycles.
Back to finiteness of CFGs

Question 13
Given a CFG $G$, is $|\mathcal{L}(G)|$ finite?

Algorithm 14
1. Remove redundant variables and terminals.
2. Turn into a CNF.
3. Create a graph $C$:
   - Add node $v_A$ for each variable $A \in V$.
   - Add directed edges $(v_A, v_B)$ and $(v_A, v_C)$, for each rule $(A \rightarrow BC) \in R$.
4. Return TRUE iff $C$ has no cycles.

Correctness?
Back to finiteness of CFGs

Question 13
Given a CFG $G$, is $|L(G)|$ finite?

Algorithm 14
1. Remove redundant variables and terminals.
2. Turn into a CNF.
3. Create a graph $C$:
   - Add node $v_A$ for each variable $A \in V$.
   - Add directed edges $(v_A, v_B)$ and $(v_A, v_C)$, for each rule $(A \rightarrow BC) \in R$.
4. Return TRUE iff $C$ has no cycles.

Correctness?
Less efficient algorithm “using” the pumping lemma:
Question 13
Given a CFG $G$, is $|L(G)|$ finite?

Algorithm 14
1. Remove redundant variables and terminals.
2. Turn into a CNF.
3. Create a graph $C$:
   - Add node $v_A$ for each variable $A \in V$.
   - Add directed edges $(v_A, v_B)$ and $(v_A, v_C)$, for each rule $(A \rightarrow BC) \in R$.
4. Return TRUE iff $C$ has no cycles.

Correctness?
Less efficient algorithm “using” the pumping lemma:
- $|L(G)| = \infty$ iff $\exists w \in L(G)$ with $\ell \leq |w| \leq 2\ell$ (\(\ell\) is the pumping length)
Back to finiteness of CFGs

**Question 13**
Given a CFG $G$, is $|\mathcal{L}(G)|$ finite?

**Algorithm 14**

1. Remove redundant variables and terminals.
2. Turn into a CNF.
3. Create a graph $C$:
   - Add node $v_A$ for each variable $A \in V$.
   - Add directed edges $(v_A, v_B)$ and $(v_A, v_C)$, for each rule $(A \to BC) \in R$.
4. Return TRUE iff $C$ has no cycles.

**Correctness?**

Less efficient algorithm “using” the pumping lemma:
- $|\mathcal{L}(G)| = \infty$ iff $\exists w \in \mathcal{L}(G)$ with $l \leq |w| \leq 2l$ (?) ($l$ is the pumping length)
- Hence, it suffices to check whether $\mathcal{L}(G)$ has a word on such length
Inherent ambiguity

**Question 15**

Given a CFG $G$, is $L(G)$ inherently ambiguous?

(i.e., for any CFG generating $L(G)$ exists $w \in L$ with two different parse trees).
Inherent ambiguity

**Question 15**

Given a CFG $G$, is $\mathcal{L}(G)$ inherently ambiguous?

(i.e., for any CFG generating $\mathcal{L}(G)$ exists $w \in \mathcal{L}$ with two different parse trees).

**Fact 16**

There is no algorithm to solve CFG inherent ambiguity.
Inherent ambiguity

Question 15
Given a CFG $G$, is $\mathcal{L}(G)$ inherently ambiguous?

(i.e., for any CFG generating $\mathcal{L}(G)$ exists $w \in \mathcal{L}$ with two different parse trees).

Fact 16
There is no algorithm to solve CFG inherent ambiguity.

We will not prove this fact, yet you want to know it to put things in context.
When are two CFGs equivalent?

Question 17

Given two CFG $G_1$ and $G_2$, test if $L(G_1) = L(G_2)$.
Is there an algorithm to solve this problem?
Part III

Equivalence Theorem
The CFG–PDA equivalence theorem

**Theorem 18**

\[ L_{\text{PDA}} = L_{\text{CFG}} \]  (a language is context free if and only if some pushdown automata accepts it).
The CFG–PDA equivalence theorem

**Theorem 18**

\[ \mathcal{L}_{\text{PDA}} = \mathcal{L}_{\text{CFG}} \] (a language is context free if and only if some pushdown automata accepts it).

This time (unlike the regular expression vs. regular languages theorem), both the proof “if” part and of the “only if” part are non trivial.
The CFG–PDA equivalence theorem

**Theorem 18**

$\mathcal{L}_{\text{PDA}} = \mathcal{L}_{\text{CFG}}$ (a language is context free if and only if some pushdown automata accepts it).

This time (unlike the regular expression vs. regular languages theorem), both the proof “if” part and of the “only if” part are non trivial.

Proof sketch follows.
Lemma 19

\[ \mathcal{L}_{\text{CFG}} \subseteq \mathcal{L}_{\text{PDA}} \] (any CFL has a PDA that accepts it).
Lemma 19

\[ \mathcal{L}_{\text{CFG}} \subseteq \mathcal{L}_{\text{PDA}} \] (any CFL has a PDA that accepts it).

- Let \( \mathcal{L} \) be a CFL, and let \( G = (V, \Sigma, R, S) \) be a CFG for \( \mathcal{L} \).
Lemma 19

\( \mathcal{L}_{\text{CFG}} \subseteq \mathcal{L}_{\text{PDA}} \) (any CFL has a PDA that accepts it).

- Let \( \mathcal{L} \) be a CFL, and let \( G = (V, \Sigma, R, S) \) be a CFG for \( \mathcal{L} \).
- We build a PDA \( P = (Q, \Sigma, \Gamma, \delta, q_0, F) \), such that on input \( w \) it “figures out” if there is a derivation of \( w \) using \( G \).
Lemma 19

\( \mathcal{L}_{\text{CFG}} \subseteq \mathcal{L}_{\text{PDA}} \) (any CFL has a PDA that accepts it).

- Let \( \mathcal{L} \) be a CFL, and let \( G = (V, \Sigma, R, S) \) be a CFG for \( \mathcal{L} \).
- We build a PDA \( P = (Q, \Sigma, \Gamma, \delta, q_0, F) \), such that on input \( w \) it "figures out" if there is a derivation of \( w \) using \( G \).
Lemma 19

\( \mathcal{L}_{\text{CFG}} \subseteq \mathcal{L}_{\text{PDA}} \) (any CFL has a PDA that accepts it).

- Let \( \mathcal{L} \) be a CFL, and let \( G = (V, \Sigma, R, S) \) be a CFG for \( \mathcal{L} \).
- We build a PDA \( P = (Q, \Sigma, \Gamma, \delta, q_0, F) \), such that on input \( w \) it “figures out” if there is a derivation of \( w \) using \( G \).

Question 20

How does \( P \) figure out which substitution to make?
Lemma 19

\[ \mathcal{L}_{\text{CFG}} \subseteq \mathcal{L}_{\text{PDA}} \] (any CFL has a PDA that accepts it).

- Let \( \mathcal{L} \) be a CFL, and let \( G = (V, \Sigma, R, S) \) be a CFG for \( \mathcal{L} \)
- We build a PDA \( P = (Q, \Sigma, \Gamma, \delta, q_0, F) \), such that on input \( w \) it “figures out” if there is a derivation of \( w \) using \( G \).

Question 20

How does \( P \) figure out which substitution to make?

Answer: It guesses.
Simplifying assumptions

1. In a single move, a PDA can push a whole word (from some fixed set) into the stack (first letter at the top)
Simplifying assumptions

1. In a single move, a PDA can push a whole word (from some fixed set) into the stack (first letter at the top)

   Can we justify it?
Simplifying assumptions

1. In a single move, a PDA can push a whole word (from some fixed set) into the stack (first letter at the top)
   
   Can we justify it?

2. When deriving a word from a CFL, we always substitute the left most variable
Simplifying assumptions

1. In a *single* move, a PDA can push a *whole* word (from some fixed set) into the stack (first letter at the top)

   Can we justify it?

2. When deriving a word from a CFL, we always substitute the *left most* variable

   Does it change the derived language?
Informal description of $P$

**Algorithm 21 ($P$)**

1. Push $S\$ \text{ on stack}$

2. While top of the stack $t$ is not $\$:
   
   2.1 If $t$ is variable $A$, (non-deterministically) select rule $A \rightarrow \alpha$ and substitute $t$ with $\alpha$.
   
   2.2 If $t$ is a terminal $a$, read next input and compare; Reject if different.

3. Accept if end of input and stack is empty.
Informal description of $P$

**Algorithm 21 ($P$)**

1. Push $S\$\$ on stack

2. While top of the stack $t$ is not $\$$:
   
   2.1 If $t$ is variable $A$, (non-deterministically) select rule $A \rightarrow \alpha$ and substitute $t$ with $\alpha$.
   
   2.2 If $t$ is a terminal $a$, read next input and compare; Reject if different.

3. **Accept** if end of input and stack is empty.

Only non-deterministic choice is in 2.1.
State diagram for $P$

- **$q_{start}$**: Transition to $S$ on $\epsilon,\epsilon$.
- **$q_{loop}$**: Transition to $\alpha$ on $\epsilon,A$ for a rule $A \rightarrow \alpha$.
  - Transition to $\epsilon$ on $a,a$ for a terminal $a$.
  - Transition to $\epsilon$ on $\epsilon,\$.
- **$q_{accept}$**: Final state.
Example

For CFG $S \rightarrow 0S1|\varepsilon$, the related PDA is

$q_{\text{start}} \rightarrow \epsilon,\epsilon \rightarrow S \$ 

$q_{\text{loop}} \rightarrow 0,0 \rightarrow \epsilon \\
1,1 \rightarrow \epsilon \\
\epsilon, S \rightarrow 0S1 \\
\epsilon, S \rightarrow \epsilon \\
\epsilon, \$ \rightarrow \epsilon \\
$q_{\text{accept}} \rightarrow \epsilon$
Claim: $\mathcal{L}(P) = \mathcal{L}(G)$
Claim: $\mathcal{L}(P) = \mathcal{L}(G)$

Claim 22

$S \overset{*}{\to} \alpha$ iff $\exists \alpha_1, \alpha_2 \in (\Sigma \cup V)^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{\text{loop}}, \alpha_2\$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$.

Note that $\alpha_1 \in \Sigma^*$. 
Claim: $\mathcal{L}(P) = \mathcal{L}(G)$

Claim 22

$S \xrightarrow{*} \alpha$ iff $\exists \alpha_1, \alpha_2 \in (\Sigma \cup V)^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{loop}, \alpha_2\$) \in \hat{\delta}(q_{start}, \alpha_1, \varepsilon)$.

Note that $\alpha_1 \in \Sigma^*$.

Does the above yields that $\mathcal{L}(P) = \mathcal{L}(G)$?
Claim: $\mathcal{L}(P) = \mathcal{L}(G)$

Claim 22

$S \xrightarrow{*} \alpha$ iff $\exists \alpha_1, \alpha_2 \in (\Sigma \cup V)^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{\text{loop}}, \alpha_2\$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$.

Note that $\alpha_1 \in \Sigma^*$.

Does the above yields that $\mathcal{L}(P) = \mathcal{L}(G)$?

- $\alpha \in \mathcal{L}(G)$
Claim: $\mathcal{L}(P) = \mathcal{L}(G)$

Claim 22

$S \xrightarrow{*} \alpha$ iff $\exists \alpha_1, \alpha_2 \in (\Sigma \cup V)^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{\text{loop}}, \alpha_2\dollar) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$.

Note that $\alpha_1 \in \Sigma^*$.

Does the above yields that $\mathcal{L}(P) = \mathcal{L}(G)$?

- $\alpha \in \mathcal{L}(G)$
Claim: $\mathcal{L}(P) = \mathcal{L}(G)$

Claim 22

$S \xrightarrow{*} \alpha$ iff $\exists \alpha_1, \alpha_2 \in (\Sigma \cup V)^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{\text{loop}}, \alpha_2\$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$.

Note that $\alpha_1 \in \Sigma^*$.

Does the above yields that $\mathcal{L}(P) = \mathcal{L}(G)$?

- $\alpha \in \mathcal{L}(G) \implies S \xrightarrow{*} \alpha$
Claim: $\mathcal{L}(P) = \mathcal{L}(G)$

Claim 22

$S \xrightarrow{*} \alpha$ iff $\exists \alpha_1, \alpha_2 \in (\Sigma \cup V)^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{\text{loop}}, \alpha_2\varepsilon) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$.

Note that $\alpha_1 \in \Sigma^*$.

Does the above yields that $\mathcal{L}(P) = \mathcal{L}(G)$?

- $\alpha \in \mathcal{L}(G) \implies S \xrightarrow{*} \alpha \implies \exists \alpha_1, \alpha_2 \in \Sigma^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{\text{loop}}, \alpha_2\varepsilon) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$
Claim: $\mathcal{L}(P) = \mathcal{L}(G)$

Claim 22

$S \xrightarrow{*} \alpha$ iff $\exists \alpha_1, \alpha_2 \in (\Sigma \cup V)^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{\text{loop}}, \alpha_2\$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$.

Note that $\alpha_1 \in \Sigma^*$.

Does the above yields that $\mathcal{L}(P) = \mathcal{L}(G)$?

$\alpha \in \mathcal{L}(G) \implies S \xrightarrow{*} \alpha \implies \exists \alpha_1, \alpha_2 \in \Sigma^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{\text{loop}}, \alpha_2\$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \implies (q_{\text{accept}}, \varepsilon) \in \hat{\delta}(q_{\text{start}}, \alpha, \varepsilon)$. 
Claim: $\mathcal{L}(P) = \mathcal{L}(G)$

Claim 22

$S \rightarrow^* \alpha$ iff $\exists \alpha_1, \alpha_2 \in (\Sigma \cup V)^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{loop}, \alpha_2\$) $\in \hat{\delta}(q_{start}, \alpha_1, \varepsilon)$.

Note that $\alpha_1 \in \Sigma^*$.

Does the above yields that $\mathcal{L}(P) = \mathcal{L}(G)$?

- $\alpha \in \mathcal{L}(G) \Longrightarrow S \rightarrow^* \alpha \Longrightarrow \exists \alpha_1, \alpha_2 \in \Sigma^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{loop}, \alpha_2\$) $\in \hat{\delta}(q_{start}, \alpha_1, \varepsilon)$ $\Longrightarrow (q_{accept}, \varepsilon)$ $\in \hat{\delta}(q_{start}, \alpha, \varepsilon)$.

- $\alpha \in \mathcal{L}(P)$
Claim: $\mathcal{L}(P) = \mathcal{L}(G)$

Claim 22

$S \to^* \alpha$ iff $\exists \alpha_1, \alpha_2 \in (\Sigma \cup V)^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{\text{loop}}, \alpha_2\$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$.

Note that $\alpha_1 \in \Sigma^*$.

Does the above yields that $\mathcal{L}(P) = \mathcal{L}(G)$?

1. $\alpha \in \mathcal{L}(G) \implies S \to^* \alpha \implies \exists \alpha_1, \alpha_2 \in \Sigma^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{\text{loop}}, \alpha_2\$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \implies (q_{\text{accept}}, \varepsilon) \in \hat{\delta}(q_{\text{start}}, \alpha, \varepsilon)$.
2. $\alpha \in \mathcal{L}(P)$
Claim: $\mathcal{L}(P) = \mathcal{L}(G)$

Claim 22

$S \xrightarrow{*} \alpha$ iff $\exists \alpha_1, \alpha_2 \in (\Sigma \cup V)^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{\text{loop}}, \alpha_2\$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$.

Note that $\alpha_1 \in \Sigma^*$.

Does the above yields that $\mathcal{L}(P) = \mathcal{L}(G)$?

- $\alpha \in \mathcal{L}(G) \implies S \xrightarrow{*} \alpha \implies \exists \alpha_1, \alpha_2 \in \Sigma^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{\text{loop}}, \alpha_2\$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \implies (q_{\text{accept}}, \varepsilon) \in \hat{\delta}(q_{\text{start}}, \alpha, \varepsilon)$.
- $\alpha \in \mathcal{L}(P) \implies (q_{\text{loop}}, \$) \in \hat{\delta}(q_{\text{start}}, \alpha, \varepsilon)$
Claim: $\mathcal{L}(P) = \mathcal{L}(G)$

Claim 22

$S \xrightarrow{*} \alpha$ iff $\exists \alpha_1, \alpha_2 \in (\Sigma \cup V)^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{\text{loop}}, \alpha_2\$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$.

Note that $\alpha_1 \in \Sigma^*$.

Does the above yields that $\mathcal{L}(P) = \mathcal{L}(G)$?

- $\alpha \in \mathcal{L}(G) \implies S \xrightarrow{*} \alpha \implies \exists \alpha_1, \alpha_2 \in \Sigma^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{\text{loop}}, \alpha_2\$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$.

- $\alpha \in \mathcal{L}(P) \implies (q_{\text{loop}}, \$) \in \hat{\delta}(q_{\text{start}}, \alpha, \varepsilon) \implies S \xrightarrow{*} \alpha\varepsilon = \alpha$
Claim: $\mathcal{L}(P) = \mathcal{L}(G)$

Claim 22

$S \xrightarrow{*} \alpha$ iff $\exists \alpha_1, \alpha_2 \in (\Sigma \cup V)^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{\text{loop}}, \alpha_2\$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$.

Note that $\alpha_1 \in \Sigma^*$.

Does the above yields that $\mathcal{L}(P) = \mathcal{L}(G)$?

$\alpha \in \mathcal{L}(G) \implies S \xrightarrow{*} \alpha \implies \exists \alpha_1, \alpha_2 \in \Sigma^*$ with $\alpha = \alpha_1 \alpha_2$ and $(q_{\text{loop}}, \alpha_2\$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$.

$\alpha \in \mathcal{L}(P) \implies (q_{\text{loop}}, \$) \in \hat{\delta}(q_{\text{start}}, \alpha, \varepsilon)$.

$\implies S \xrightarrow{*} \alpha\varepsilon = \alpha \implies \alpha \in \mathcal{L}(G)$.
$S \rightarrow^* \alpha \iff \alpha = \alpha_1 \alpha_2$ such that $(q_{\text{loop}}, \alpha_2) \in \tilde{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$

Proof:
$S \rightarrow^{*} \alpha \implies \alpha = \alpha_1\alpha_2$ such that $(q_{\text{loop}}, \alpha_2)$ $\in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$

Proof: by induction on the number of derivations steps used to yield $\alpha$ from $S$. 
\[ S \rightarrow^* \alpha \iff \alpha = \alpha_1 \alpha_2 \text{ such that } (q_{\text{loop}}, \alpha_2) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \]

Proof: by induction on the number of derivations steps used to yield \( \alpha \) from \( S \).

Single derivation step:

- Exists a rule \( S \rightarrow \alpha \).
\[ S \rightarrow^* \alpha \implies \alpha = \alpha_1 \alpha_2 \text{ such that } (q_{\text{loop}}, \alpha_2$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \]

Proof: by induction on the number of derivations steps used to yield \( \alpha \) from \( S \).

Single derivation step:

- Exists a rule \( S \rightarrow \alpha \).
- Thus \( (q_{\text{loop}}, \alpha$) \in \hat{\delta}(q_{\text{start}}, \varepsilon, \varepsilon), \)

\( \text{and proof follows for } \alpha_1 = \varepsilon \text{ and } \alpha_2 = \alpha \).

Assume \( S \rightarrow^* \alpha \) in \( k > 1 \) derivation steps.

- Let \( \alpha' \) be the string derived by the first \( (k-1) \) steps.
- By i.h \( \alpha' = \alpha'_1 \alpha'_2 \) such that \( (q_{\text{loop}}, \alpha'_2$) \in \hat{\delta}(q_{\text{start}}, \alpha'_1, \varepsilon) \).

Write \( \alpha'_2 = w_1 A w_2 \) where \( A \) is the left most variable in \( \alpha'_2 \).

\( \hat{\delta}(q_{\text{loop}}, w_1, \alpha'_2$) \in \hat{\delta}(q_{\text{start}}, \alpha'_1 w_1, \varepsilon) \).

\( k \text{th derivation step replaces this occurrence of } A \) with a string \( s \).

\( \hat{\delta}(q_{\text{loop}}, sw_2$) \in \hat{\delta}(q_{\text{start}}, \alpha'_1 w_1, \varepsilon) \).

Thus, \( (q_{\text{loop}}, sw_2$) \in \hat{\delta}(q_{\text{start}}, \alpha'_1 w_1, \varepsilon) \).

To complete the proof take \( \alpha_1 = \alpha'_1 w_1 \) and \( \alpha_2 = sw_2 \).
$S \xrightarrow{*} \alpha \implies \alpha = \alpha_1 \alpha_2$ such that $(q_{\text{loop}}, \alpha_2\$$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$

Proof: by induction on the number of derivations steps used to yield $\alpha$ from $S$.

Single derivation step:

- Exists a rule $S \rightarrow \alpha$.
- Thus $(q_{\text{loop}}, \alpha\$$) \in \hat{\delta}(q_{\text{start}}, \varepsilon, \varepsilon)$,
$S \to^* \alpha \iff \alpha = \alpha_1 \alpha_2$ such that $(q_{\text{loop}}, \alpha_2) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$

Proof: by induction on the number of derivations steps used to yield $\alpha$ from $S$.

Single derivation step:

- Exists a rule $S \to \alpha$.
- Thus $(q_{\text{loop}}, \alpha) \in \hat{\delta}(q_{\text{start}}, \varepsilon, \varepsilon)$, and proof follows for $\alpha_1 = \varepsilon$ and $\alpha_2 = \alpha$. 

$Iftach Haitner (TAU)$  
Computational Models, Lecture 6  
November 19, 2018  26 / 35
$S \xrightarrow{*} \alpha \iff \alpha = \alpha_1\alpha_2$ such that $(q_{\text{loop}}, \alpha_2\$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$

Proof: by induction on the number of derivations steps used to yield $\alpha$ from $S$.

Single derivation step:

- Exists a rule $S \rightarrow \alpha$.
- Thus $(q_{\text{loop}}, \alpha\$) \in \hat{\delta}(q_{\text{start}}, \varepsilon, \varepsilon)$, and proof follows for $\alpha_1 = \varepsilon$ and $\alpha_2 = \alpha$.

Assume $S \xrightarrow{*} \alpha$ in $k > 1$ derivation steps.
$S \rightarrow^* \alpha \implies \alpha = \alpha_1 \alpha_2$ such that $(q_{\text{loop}}, \alpha_2) \in \tilde{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$

Proof: by induction on the number of derivations steps used to yield $\alpha$ from $S$.

Single derivation step:

- Exists a rule $S \rightarrow \alpha$.
- Thus $(q_{\text{loop}}, \alpha) \in \tilde{\delta}(q_{\text{start}}, \varepsilon, \varepsilon)$, and proof follows for $\alpha_1 = \varepsilon$ and $\alpha_2 = \alpha$.

Assume $S \rightarrow^* \alpha$ in $k > 1$ derivation steps.

- Let $\alpha'$ be the string derived by the first $(k - 1)$ steps.
\( S \xrightarrow{*} \alpha \implies \alpha = \alpha_1 \alpha_2 \) such that \((q_{\text{loop}}, \alpha_2\$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)\)

Proof: by induction on the number of derivations steps used to yield \( \alpha \) from \( S \).

Single derivation step:

- Exists a rule \( S \rightarrow \alpha \).
- Thus \((q_{\text{loop}}, \alpha\$) \in \hat{\delta}(q_{\text{start}}, \varepsilon, \varepsilon)\), and proof follows for \( \alpha_1 = \varepsilon \) and \( \alpha_2 = \alpha \).

Assume \( S \xrightarrow{*} \alpha \) in \( k > 1 \) derivation steps.

- Let \( \alpha' \) be the string derived by the first \((k - 1)\) steps.
- By i.h \( \alpha' = \alpha'_1 \alpha'_2 \) such that \((q_{\text{loop}}, \alpha'_2\$) \in \hat{\delta}(q_{\text{start}}, \alpha'_1, \varepsilon)\)
\[ S \xrightarrow{*} \alpha \implies \alpha = \alpha_1 \alpha_2 \text{ such that } (q_{\text{loop}}, \alpha_2) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \]

Proof: by induction on the number of derivations steps used to yield \( \alpha \) from \( S \).

Single derivation step:

- Exists a rule \( S \rightarrow \alpha \).
- Thus \( (q_{\text{loop}}, \alpha) \in \hat{\delta}(q_{\text{start}}, \varepsilon, \varepsilon) \), and proof follows for \( \alpha_1 = \varepsilon \) and \( \alpha_2 = \alpha \).

Assume \( S \xrightarrow{*} \alpha \) in \( k > 1 \) derivation steps.

- Let \( \alpha' \) be the string derived by the first \( (k - 1) \) steps.
- By i.h \( \alpha' = \alpha_1' \alpha_2' \) such that \( (q_{\text{loop}}, \alpha_2') \in \hat{\delta}(q_{\text{start}}, \alpha_1', \varepsilon) \)
- Write \( \alpha_2' = w_1 A w_2 \) where \( A \) is the left most variable in \( \alpha_2' \). (?)
\( S \overset{*}{\rightarrow} \alpha \implies \alpha = \alpha_1 \alpha_2 \text{ such that } (q_{\text{loop}}, \alpha_2 \$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \)

Proof: by induction on the number of derivations steps used to yield \( \alpha \) from \( S \).

Single derivation step:

- Exists a rule \( S \rightarrow \alpha \).
- Thus \( (q_{\text{loop}}, \alpha \$) \in \hat{\delta}(q_{\text{start}}, \varepsilon, \varepsilon) \), and proof follows for \( \alpha_1 = \varepsilon \) and \( \alpha_2 = \alpha \).

Assume \( S \overset{*}{\rightarrow} \alpha \) in \( k > 1 \) derivation steps.

- Let \( \alpha' \) be the string derived by the first \( (k - 1) \) steps.
- By i.h \( \alpha' = \alpha'_1 \alpha'_2 \) such that \( (q_{\text{loop}}, \alpha'_2 \$) \in \hat{\delta}(q_{\text{start}}, \alpha'_1, \varepsilon) \)
- Write \( \alpha'_2 = w_1 Aw_2 \) where \( A \) is the left most variable in \( \alpha'_2 \). (?)

\((*) \) \( \hat{\delta}(q_{\text{loop}}, w_1, \alpha'_2 \$) \in \hat{\delta}(q_{\text{start}}, \alpha'_1 w_1, \varepsilon) \)
$S \xrightarrow{*} \alpha \implies \alpha = \alpha_1 \alpha_2$ such that $(q_{loop}, \alpha_2 \varepsilon) \in \hat{\delta}(q_{start}, \alpha_1, \varepsilon)$

Proof: by induction on the number of derivations steps used to yield $\alpha$ from $S$.

Single derivation step:

- Exists a rule $S \rightarrow \alpha$.
- Thus $(q_{loop}, \alpha \varepsilon) \in \hat{\delta}(q_{start}, \varepsilon, \varepsilon)$, and proof follows for $\alpha_1 = \varepsilon$ and $\alpha_2 = \alpha$.

Assume $S \xrightarrow{*} \alpha$ in $k > 1$ derivation steps.

- Let $\alpha'$ be the string derived by the first $(k - 1)$ steps.
- By i.h $\alpha' = \alpha_1' \alpha_2'$ such that $(q_{loop}, \alpha_2' \varepsilon) \in \hat{\delta}(q_{start}, \alpha_1', \varepsilon)$
- Write $\alpha_2' = w_1 A w_2$ where $A$ is the left most variable in $\alpha_2'$. (?)

$(\ast) \hat{\delta}(q_{loop}, w_1, \alpha_2' \varepsilon) \in \hat{\delta}(q_{start}, \alpha_1' w_1, \varepsilon)$

- $k$'th derivation step replaces this occurrence of $A$ with a string $s$ (?)
\[ S \rightarrow^* \alpha \quad \implies \quad \alpha = \alpha_1 \alpha_2 \text{ such that } (q_{\text{loop}}, \alpha_2) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \]

Proof: by induction on the number of derivations steps used to yield \( \alpha \) from \( S \).

Single derivation step:

- Exists a rule \( S \rightarrow \alpha \).
- Thus \( (q_{\text{loop}}, \alpha) \in \hat{\delta}(q_{\text{start}}, \varepsilon, \varepsilon) \), and proof follows for \( \alpha_1 = \varepsilon \) and \( \alpha_2 = \alpha \).

Assume \( S \rightarrow^* \alpha \) in \( k > 1 \) derivation steps.

- Let \( \alpha' \) be the string derived by the first \( (k - 1) \) steps.
- By i.h \( \alpha' = \alpha'_1 \alpha'_2 \) such that \( (q_{\text{loop}}, \alpha'_2) \in \hat{\delta}(q_{\text{start}}, \alpha'_1, \varepsilon) \)
- Write \( \alpha'_2 = w_1 Aw_2 \) where \( A \) is the left most variable in \( \alpha'_2 \). (\( ? \))

\((*)\) \( \hat{\delta}(q_{\text{loop}}, w_1, \alpha'_2) \in \hat{\delta}(q_{\text{start}}, \alpha'_1 w_1, \varepsilon) \)

- \( k' \)th derivation step replaces this occurrence of \( A \) with a string \( s \) (\( ? \))

\((***)\) (since \( A \implies S \)): \( (q_{\text{loop}}, sw_2) \in \hat{\delta}(q_{\text{loop}}, w_1, \alpha'_2) \).
\( S \xrightarrow{*} \alpha \implies \alpha = \alpha_1 \alpha_2 \text{ such that } (q_{\text{loop}}, \alpha_2\$) \in \tilde{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \)

Proof: by induction on the number of derivations steps used to yield \( \alpha \) from \( S \).

Single derivation step:

- Exists a rule \( S \rightarrow \alpha \).
- Thus \( (q_{\text{loop}}, \alpha\$) \in \tilde{\delta}(q_{\text{start}}, \varepsilon, \varepsilon) \), and proof follows for \( \alpha_1 = \varepsilon \) and \( \alpha_2 = \alpha \).

Assume \( S \xrightarrow{*} \alpha \) in \( k > 1 \) derivation steps.

- Let \( \alpha' \) be the string derived by the first \( (k - 1) \) steps.
- By i.h \( \alpha' = \alpha'_1 \alpha'_2 \) such that \( (q_{\text{loop}}, \alpha'_2\$) \in \tilde{\delta}(q_{\text{start}}, \alpha'_1, \varepsilon) \)
- Write \( \alpha'_2 = w_1 Aw_2 \) where \( A \) is the left most variable in \( \alpha'_2 \). (?)

\[
(*) \quad \tilde{\delta}(q_{\text{loop}}, w_1, \alpha'_2\$) \in \tilde{\delta}(q_{\text{start}}, \alpha'_1 w_1, \varepsilon)
\]

- \( k' \)th derivation step replaces this occurrence of \( A \) with a string \( s \) (?)

\[
(**) \quad (\text{since } A \implies S): \quad (q_{\text{loop}}, sw_2\$) \in \tilde{\delta}(q_{\text{loop}}, w_1, \alpha'_2\$).
\]

- Thus, \( (q_{\text{loop}}, sw_2\$) \in \tilde{\delta}(q_{\text{start}}, \alpha'_1 w_1, \varepsilon) \).
$S \xrightarrow{\star} \alpha \implies \alpha = \alpha_1 \alpha_2$ such that $(q_{\text{loop}}, \alpha_2) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon)$

Proof: by induction on the number of derivations steps used to yield $\alpha$ from $S$.

Single derivation step:

- Exists a rule $S \rightarrow \alpha$.
- Thus $(q_{\text{loop}}, \alpha) \in \hat{\delta}(q_{\text{start}}, \varepsilon, \varepsilon)$, and proof follows for $\alpha_1 = \varepsilon$ and $\alpha_2 = \alpha$.

Assume $S \xrightarrow{\star} \alpha$ in $k > 1$ derivation steps.

- Let $\alpha'$ be the string derived by the first $(k - 1)$ steps.
- By i.h $\alpha' = \alpha_1' \alpha_2'$ such that $(q_{\text{loop}}, \alpha_2') \in \hat{\delta}(q_{\text{start}}, \alpha_1', \varepsilon)$
- Write $\alpha_2' = w_1 A w_2$ where $A$ is the left most variable in $\alpha_2'$. (?)

$(\ast)$ $\hat{\delta}(q_{\text{loop}}, w_1, \alpha_2') \in \hat{\delta}(q_{\text{start}}, \alpha_1' w_1, \varepsilon)$

- $k$'th derivation step replaces this occurrence of $A$ with a string $s$ (?)

$(\ast\ast)$ (since $A \implies S$): $(q_{\text{loop}}, sw_2) \in \hat{\delta}(q_{\text{loop}}, w_1, \alpha_2')$.

- Thus, $(q_{\text{loop}}, sw_2) \in \hat{\delta}(q_{\text{start}}, \alpha_1' w_1, \varepsilon)$.
- To complete the proof take $\alpha_1 = \alpha_1' w_1$ and $\alpha_2 = sw_2$. 

Iftach Haitner (TAU) 
Computational Models, Lecture 6 
November 19, 2018 26 / 35
\(\alpha = \alpha_1\alpha_2\) such that \((q_{\text{loop}}, \alpha_2\$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \implies S \xrightarrow{*} \alpha\)

Proof:
\[ \alpha = \alpha_1 \alpha_2 \text{ such that } (q_{\text{loop}}, \alpha_2 \$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \implies S \xrightarrow{\ast} \alpha \]

Proof: by induction on the number of steps used by \( P \) to process \( \alpha_1 \).
\[ \alpha = \alpha_1 \alpha_2 \text{ such that } (q_{\text{loop}}, \alpha_2) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \implies S \rightarrow^* \alpha \]

Proof: by induction on the number of steps used by \( P \) to process \( \alpha_1 \).

Single step:

\[ \begin{align*}
\text{\textbullet{} } \alpha_1 &= \varepsilon \text{ and } \alpha_2 = S,
\end{align*} \]
\[ \alpha = \alpha_1 \alpha_2 \text{ such that } (q_{\text{loop}}, \alpha_2) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \implies S \xrightarrow{*} \alpha \]

Proof: by induction on the number of steps used by \( P \) to process \( \alpha_1 \).

Single step:

\[ \bullet \quad \alpha_1 = \varepsilon \text{ and } \alpha_2 = S, \]
\[ \alpha = \alpha_1 \alpha_2 \text{ such that } (q_{\text{loop}}, \alpha_2 \$) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \implies S \to^* \alpha \]

Proof: by induction on the number of steps used by \( P \) to process \( \alpha_1 \).

Single step:

- \( \alpha_1 = \varepsilon \) and \( \alpha_2 = S \), and the proof follows since \( S \to^* S = \alpha_1 \alpha_2 \).
\( \alpha = \alpha_1 \alpha_2 \) such that \((q_{\text{loop}}, \alpha_2) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \implies S \xrightarrow{*} \alpha \)

Proof: by induction on the number of steps used by \( P \) to process \( \alpha_1 \).

Single step:

- \( \alpha_1 = \varepsilon \) and \( \alpha_2 = S \), and the proof follows since \( S \xrightarrow{*} S = \alpha_1 \alpha_2 \).
\[ \alpha = \alpha_1 \alpha_2 \text{ such that } (q_{\text{loop}}, \alpha_2) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \implies S \xrightarrow{*} \alpha \]

Proof: by induction on the number of steps used by \( P \) to process \( \alpha_1 \).

Single step:

- \( \alpha_1 = \varepsilon \) and \( \alpha_2 = S \), and the proof follows since \( S \xrightarrow{*} S = \alpha_1 \alpha_2 \).

Assume \( \alpha_1 \) was processed in \( k > 1 \) steps.
\[\alpha = \alpha_1 \alpha_2 \text{ such that } (q_{\text{loop}}, \alpha_2) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \implies S \vdash^* \alpha\]

Proof: by induction on the number of steps used by \(P\) to process \(\alpha_1\).

Single step:

- \(\alpha_1 = \varepsilon\) and \(\alpha_2 = S\), and the proof follows since \(S \vdash^* S = \alpha_1 \alpha_2\).

Assume \(\alpha_1\) was processed in \(k > 1\) steps.

- Let \(\alpha'\) and \(\alpha'\$\) be input string read and stack value before last step:
\( \alpha = \alpha_1 \alpha_2 \) such that \((q_{loop}, \alpha_2) \in \delta(q_{start}, \alpha_1, \varepsilon) \implies S \xrightarrow{*} \alpha \)

Proof: by induction on the number of steps used by \( P \) to process \( \alpha_1 \).

Single step:

▷ \( \alpha_1 = \varepsilon \) and \( \alpha_2 = S \), and the proof follows since \( S \xrightarrow{*} S = \alpha_1 \alpha_2 \).

Assume \( \alpha_1 \) was processed in \( k > 1 \) steps.

▷ Let \( \alpha'_1 \) and \( \alpha'_2 \) be input string read and stack value before last step:

▷ \((q_{loop}, \alpha'_2) \in \delta(q_{start}, \alpha'_1, \varepsilon) \)
\[ \alpha = \alpha_1 \alpha_2 \text{ such that } (q_{\text{loop}}, \alpha_2) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \implies S \Rightarrow^* \alpha \]

Proof: by induction on the number of steps used by \( P \) to process \( \alpha_1 \).

Single step:

- \( \alpha_1 = \varepsilon \) and \( \alpha_2 = S \), and the proof follows since \( S \Rightarrow^* S = \alpha_1 \alpha_2 \).

Assume \( \alpha_1 \) was processed in \( k > 1 \) steps.

- Let \( \alpha'_1 \) and \( \alpha'_2 \) be input string read and stack value \textit{before} last step:
  - \( (q_{\text{loop}}, \alpha'_2) \in \hat{\delta}(q_{\text{start}}, \alpha'_1, \varepsilon) \)
  - \( (q_{\text{loop}}, \alpha_2) \in \hat{\delta}(q_{\text{loop}}, \sigma, \alpha'_2) \) (for some \( \sigma \in \Sigma_{\varepsilon} \))
\[ \alpha = \alpha_1 \alpha_2 \text{ such that } (q_{\text{loop}}, \alpha_2) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \implies S \xrightarrow{*} \alpha \]

Proof: by induction on the number of steps used by \( P \) to process \( \alpha_1 \).

Single step:

\[ \alpha_1 = \varepsilon \text{ and } \alpha_2 = S, \text{ and the proof follows since } S \xrightarrow{*} S = \alpha_1 \alpha_2. \]

Assume \( \alpha_1 \) was processed in \( k > 1 \) steps.

\[ \begin{align*}
& \text{Let } \alpha_1' \text{ and } \alpha_2 \text{ be input string read and stack value before last step:} \\
& \quad (q_{\text{loop}}, \alpha_2) \in \hat{\delta}(q_{\text{start}}, \alpha_1', \varepsilon) \\
& \quad (q_{\text{loop}}, \alpha_2) \in \tilde{\delta}(q_{\text{loop}}, \sigma, \alpha_2') \text{ (for some } \sigma \in \Sigma) \\
& \quad \text{By i.h } S \xrightarrow{*} \alpha' = \alpha_1' \alpha_2'.
\end{align*} \]
\( \alpha = \alpha_1 \alpha_2 \text{ such that } (q_{\text{loop}}, \alpha_2) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \implies S \xrightarrow{*} \alpha \)

Proof: by induction on the number of steps used by \( P \) to process \( \alpha_1 \).

Single step:

- \( \alpha_1 = \varepsilon \) and \( \alpha_2 = S \), and the proof follows since \( S \xrightarrow{*} S = \alpha_1 \alpha_2 \).

Assume \( \alpha_1 \) was processed in \( k > 1 \) steps.

- Let \( \alpha_1' \) and \( \alpha_2' \) be input string read and stack value before last step:
  - \( (q_{\text{loop}}, \alpha_2') \in \hat{\delta}(q_{\text{start}}, \alpha_1', \varepsilon) \)
  - \( (q_{\text{loop}}, \alpha_2) \in \tilde{\delta}(q_{\text{loop}}, \sigma, \alpha_2') \) (for some \( \sigma \in \Sigma_\varepsilon \))
  - By i.h \( S \xrightarrow{*} \alpha' = \alpha_1' \alpha_2' \).
  - If \( \sigma \neq \varepsilon \) (i.e., \( k \)'th move of \( P \) is reading an input character), then \( \alpha_1 = \alpha_1' \sigma \) and \( \alpha_2 = \sigma \alpha_2 \), and therefore \( S \xrightarrow{*} \alpha_1 \alpha_2 \).
\[ \alpha = \alpha_1 \alpha_2 \text{ such that } (q_{\text{loop}}, \alpha_2) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \implies S \xrightarrow{*} \alpha \]

Proof: by induction on the number of steps used by \( P \) to process \( \alpha_1 \).

Single step:

- \( \alpha_1 = \varepsilon \) and \( \alpha_2 = S \), and the proof follows since \( S \xrightarrow{*} S = \alpha_1 \alpha_2 \).

Assume \( \alpha_1 \) was processed in \( k > 1 \) steps.

- Let \( \alpha'_1 \) and \( \alpha'_2 \) be input string read and stack value before last step:
  - \( (q_{\text{loop}}, \alpha'_2) \in \hat{\delta}(q_{\text{start}}, \alpha'_1, \varepsilon) \)
  - \( (q_{\text{loop}}, \alpha_2) \in \tilde{\delta}(q_{\text{loop}}, \sigma, \alpha'_2) \) (for some \( \sigma \in \Sigma_{\varepsilon} \))

- By i.h. \( S \xrightarrow{*} \alpha' = \alpha'_1 \alpha'_2 \).

- If \( \sigma \neq \varepsilon \) (i.e., \( k \)'th move of \( P \) is reading an input character), then \( \alpha_1 = \alpha'_1 \sigma \) and \( \alpha'_2 = \sigma \alpha_2 \), and therefore \( S \xrightarrow{*} \alpha_1 \alpha_2 \)

Else, \( \alpha'_1 = \alpha_1 \), \( \alpha'_2 = Aw \) and \( \alpha_2 = sw \) for some \( (A \rightarrow s) \in R \) and \( w \in (\Sigma \cup V)^* \)
\( \alpha = \alpha_1 \alpha_2 \) such that \((q_{\text{loop}}, \alpha_2) \in \hat{\delta}(q_{\text{start}}, \alpha_1, \varepsilon) \implies S \xrightarrow{*} \alpha \)

Proof: by induction on the number of steps used by \( P \) to process \( \alpha_1 \).

Single step:

- \( \alpha_1 = \varepsilon \) and \( \alpha_2 = S \), and the proof follows since \( S \xrightarrow{*} S = \alpha_1 \alpha_2 \).

Assume \( \alpha_1 \) was processed in \( k > 1 \) steps.

- Let \( \alpha'_1 \) and \( \alpha'_2 \) be input string read and stack value before last step:
  - \((q_{\text{loop}}, \alpha'_2) \in \hat{\delta}(q_{\text{start}}, \alpha'_1, \varepsilon) \)
  - \((q_{\text{loop}}, \alpha_2) \in \tilde{\delta}(q_{\text{loop}}, \sigma, \alpha'_2) \) (for some \( \sigma \in \Sigma_{\varepsilon} \))
  - By i.h \( S \xrightarrow{*} \alpha' = \alpha'_1 \alpha'_2 \).
  - If \( \sigma \neq \varepsilon \) (i.e., \( k \)'th move of \( P \) is reading an input character), then \( \alpha_1 = \alpha'_1 \sigma \) and \( \alpha_2 = \sigma \alpha_2 \), and therefore \( S \xrightarrow{*} \alpha_1 \alpha_2 \)
  - Else, \( \alpha'_1 = \alpha_1 \), \( \alpha'_2 = Aw \) and \( \alpha_2 = sw \) for some \((A \rightarrow s) \in R \) and \( w \in (\Sigma \cup V)^* \)
  - Hence \( S \xrightarrow{*} \alpha_1 Aw \rightarrow \alpha_1 \alpha_2 \)
Lemma 23

$L_{PDA} \subseteq L_{CFG}$.

If a PDA accepts a language then it is context free.
Lemma 23

$\mathcal{L}_{PDA} \subseteq \mathcal{L}_{CFG}.$

If a PDA accepts a language then it is context free.

We prove the lemma by constructing a CFG $G$ for a language $\mathcal{L}$ accepted by a PDA $P$. 
Lemma 23

\[ L_{PDA} \subseteq L_{CFG}. \]

If a PDA accepts a language then it is context free.

We prove the lemma by constructing a CFG \( G \) for a language \( L \) accepted by a PDA \( P \).

Let \( P = (Q, \Sigma, \Gamma, \delta, q_0, F) \). We assume w.l.o.g. that:

- A single accepting state \( q_a \in F \).
- \( P \) empties the stack before accepting.
- Each transition either pops or pushes.

Can we justify the above?
Lemma 23

$L_{PDA} \subseteq L_{CFG}$.

If a PDA accepts a language then it is context free.

We prove the lemma by constructing a CFG $G$ for a language $L$ accepted by a PDA $P$.

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$. We assume wlg. that:

- A single accepting state $q_a \in F$. 

Lemma 23

\[ \mathcal{L}_{\text{PDA}} \subseteq \mathcal{L}_{\text{CFG}}. \]

If a PDA accepts a language then it is context free.

We prove the lemma by constructing a CFG \( G \) for a language \( \mathcal{L} \) accepted by a PDA \( P \).

Let \( P = (Q, \Sigma, \Gamma, \delta, q_0, F) \). We assume w.l.o.g. that:

- A single accepting state \( q_a \in F \).
- \( P \) empties the stack before accepting.
Lemma 23

\( \mathcal{L}_{\text{PDA}} \subseteq \mathcal{L}_{\text{CFG}} \).

If a PDA accepts a language then it is context free.

We prove the lemma by constructing a CFG \( G \) for a language \( \mathcal{L} \) accepted by a PDA \( P \).

Let \( P = (Q, \Sigma, \Gamma, \delta, q_0, F) \). We assume wlg. that:

- A single accepting state \( q_a \in F \).
- \( P \) empties the stack before accepting
- Each transition either pops or pushes
Lemma 23

$L_{\text{PDA}} \subseteq L_{\text{CFG}}$. If a PDA accepts a language then it is context free.

We prove the lemma by constructing a CFG $G$ for a language $L$ accepted by a PDA $P$

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$. We assume w.l.o.g. that:

- A single accepting state $q_a \in F$.
- $P$ empties the stack before accepting
- Each transition either pops or pushes
Lemma 23

\[ \mathcal{L}_{\text{PDA}} \subseteq \mathcal{L}_{\text{CFG}}. \]

If a PDA accepts a language then it is context free.

We prove the lemma by constructing a CFG \( G \) for a language \( \mathcal{L} \) accepted by a PDA \( P \).

Let \( P = (Q, \Sigma, \Gamma, \delta, q_0, F) \). We assume w.l.o.g. that:

- A single accepting state \( q_a \in F \).
- \( P \) empties the stack before accepting
- Each transition either pops or pushes

Can we justify the above?
Defining $G = (V, \Sigma, R, S)$

(copy to board)

$V = \{ A_{pq} : p, q \in Q \}$
Defining $G = (V, \Sigma, R, S)$

(copy to board)

$V = \{A_{pq} : p, q \in Q\}$

Idea: $A_{pq}$ will generate all strings that take $P$ from $p$ with an empty stack, to $q$ with an empty stack
Defining $G = (V, \Sigma, R, S)$

(copied to board)

- $V = \{A_{pq} : p, q \in Q\}$
  
  Idea: $A_{pq}$ will generate all strings that take $P$ from $p$ with an empty stack, to $q$ with an empty stack

- $S = A_{q_0, q_a}$
Defining $G = (V, \Sigma, R, S)$

(copy to board)

- $V = \{A_{pq} : p, q \in Q\}$
  - Idea: $A_{pq}$ will generate all strings that take $P$ from $p$ with an empty stack, to $q$ with an empty stack
- $S = A_{q_0,q_a}$
- Set $R = R_1 \cup R_2 \cup R_3$, for
Defining $G = (V, \Sigma, R, S)$

(copy to board)

- $V = \{A_{pq} : p, q \in Q\}$
  
  Idea: $A_{pq}$ will generate all strings that take $P$ from $p$ with an empty stack, to $q$ with an empty stack

- $S = A_{q_0, q_a}$

- Set $R = R_1 \cup R_2 \cup R_3$, for
  
  1. $R_1 = \{A_{qq} \rightarrow \varepsilon : q \in Q\}$
Defining $G = (V, \Sigma, R, S)$

(copied to board)

- $V = \{A_{pq}: p, q \in Q\}$
  
  Idea: $A_{pq}$ will generate all strings that take $P$ from $p$ with an empty stack, to $q$ with an empty stack

- $S = A_{q_0, q_a}$

- Set $R = R_1 \cup R_2 \cup R_3$, for
  
  1. $R_1 = \{A_{qq} \rightarrow \varepsilon: q \in Q\}$

  For the “no move” case
Defining $G = (V, \Sigma, R, S)$

(copy to board)

- $V = \{A_{pq} : p, q \in Q\}$
  
  Idea: $A_{pq}$ will generate all strings that take $P$ from $p$ with an empty stack, to $q$ with an empty stack

- $S = A_{q_0, q_a}$

- Set $R = R_1 \cup R_2 \cup R_3$, for
  
  1. $R_1 = \{A_{qq} \rightarrow \varepsilon : q \in Q\}$
     
     For the “no move” case
  
  2. $R_2 = \{A_{pq} \rightarrow A_{p,r}A_{r,q} : p, q, r \in Q\}$
Defining $G = (V, \Sigma, R, S)$

(copy to board)

- $V = \{A_{pq}: p, q \in Q\}$

  Idea: $A_{pq}$ will generate all strings that take $P$ from $p$ with an empty stack, to $q$ with an empty stack

- $S = A_{q_0, q_a}$

- Set $R = R_1 \cup R_2 \cup R_3$, for

  1. $R_1 = \{A_{qq} \rightarrow \varepsilon: q \in Q\}$

     For the “no move” case

  2. $R_2 = \{A_{pq} \rightarrow A_{p,r}A_{r,q}: p, q, r \in Q\}$

     For strings in for which the stack gets empty while processing them
Defining $G = (V, \Sigma, R, S)$

(copy to board)

- $V = \{A_{pq} : p, q \in Q\}$
  
  Idea: $A_{pq}$ will generate all strings that take $P$ from $p$ with an empty stack, to $q$ with an empty stack

- $S = A_{q_0, q_a}$

- Set $R = R_1 \cup R_2 \cup R_3$, for
  
  1. $R_1 = \{A_{qq} \rightarrow \varepsilon : q \in Q\}$
     
     For the “no move” case
  2. $R_2 = \{A_{pq} \rightarrow A_{p, r} A_{r, q} : p, q, r \in Q\}$
     
     For strings in for which the stack gets empty while processing them
  3. $R_3 = \{A_{pq} \rightarrow aA_{r, s} b : p, r, s, q \in Q, a, b \in \Sigma_\varepsilon\}$ for which $\exists \gamma \in \Gamma$ s.t.:
Defining $G = (V, \Sigma, R, S)$

(copied to board)

- $V = \{A_{pq} : p, q \in Q\}$
  
  Idea: $A_{pq}$ will generate all strings that take $P$ from $p$ with an empty stack, to $q$ with an empty stack.

- $S = A_{q_0, q_a}$

- Set $R = R_1 \cup R_2 \cup R_3$, for
  1. $R_1 = \{A_{qq} \rightarrow \varepsilon : q \in Q\}$
     For the “no move” case
  2. $R_2 = \{A_{pq} \rightarrow A_{p, r} A_{r, q} : p, q, r \in Q\}$
     For strings in for which the stack gets empty while processing them
  3. $R_3 = \{A_{pq} \rightarrow aA_{r, s} b : p, r, s, q \in Q, a, b \in \Sigma_{\varepsilon}\}$ for which $\exists \gamma \in \Gamma$ s.t.:
     3.1 $(r, \gamma) \in \delta(p, a, \varepsilon)$
Defining $G = (V, \Sigma, R, S)$

(copy to board)

- $V = \{A_{pq} : p, q \in Q\}$
  
  Idea: $A_{pq}$ will generate all strings that take $P$ from $p$ with an empty stack, to $q$ with an empty stack

- $S = A_{q_0, q_a}$

- Set $R = R_1 \cup R_2 \cup R_3$, for
  
  1. $R_1 = \{A_{qq} \rightarrow \varepsilon : q \in Q\}$
     
     For the “no move” case

  2. $R_2 = \{A_{pq} \rightarrow A_p, rA_r, q : p, q, r \in Q\}$
     
     For strings in for which the stack gets empty while processing them

  3. $R_3 = \{A_{pq} \rightarrow aA_r, s b : p, r, s, q \in Q, a, b \in \Sigma_\varepsilon\}$ for which $\exists \gamma \in \Gamma$ s.t.:
     
     3.1 $(r, \gamma) \in \delta(p, a, \varepsilon)$
     
     3.2 $(q, \varepsilon) \in \delta(s, b, \gamma)$
Defining $G = (V, \Sigma, R, S)$

(copied to board)

$V = \{A_{pq} : p, q \in Q\}$

Idea: $A_{pq}$ will generate all strings that take $P$ from $p$ with an empty stack, to $q$ with an empty stack

$S = A_{q_0,q_a}$

Set $R = R_1 \cup R_2 \cup R_3$, for

1. $R_1 = \{A_{qq} \rightarrow \varepsilon : q \in Q\}$
   For the “no move” case

2. $R_2 = \{A_{pq} \rightarrow A_{p,r} A_{r,q} : p, q, r \in Q\}$
   For strings in for which the stack gets empty while processing them

3. $R_3 = \{A_{pq} \rightarrow aA_{r,s}b : p, r, s, q \in Q, a, b \in \Sigma_{\varepsilon}\}$ for which $\exists \gamma \in \Gamma$ s.t.:
   3.1 $(r, \gamma) \in \delta(p, a, \varepsilon)$
   3.2 $(q, \varepsilon) \in \delta(s, b, \gamma)$
   For strings in for which the stack does not get empty while processing them
Example PDA to CFG

\[
\begin{align*}
q_1 & \xrightarrow{\epsilon, \epsilon} q_2 \\
q_2 & \xrightarrow{0, \epsilon} 0 \\
q_1 & \xrightarrow{\epsilon, \$} q_4 \\
q_4 & \xrightarrow{\epsilon, \$} \epsilon \\
q_3 & \xrightarrow{1, 0} q_3 \\
q_3 & \xrightarrow{1, 0} \epsilon
\end{align*}
\]
Some rules in $R$

- $A_{q_i, q_i} \rightarrow \varepsilon$
- $A_{q_1, q_4} \rightarrow A_{q_2, q_3}$  \hspace{1cm} (a = b = \varepsilon, \gamma = $)
- $A_{q_2, q_3} \rightarrow 0A_{q_2, q_3} 1$  \hspace{1cm} (a = 0, b = 1, \gamma = 0)
Proving that $\mathcal{L}(G) = \mathcal{L}(P)$

Claim 24

$$A_{pq} \xrightarrow{*} w \in \Sigma^* \iff (q, \varepsilon) \in \hat{\delta}(p, w, \varepsilon)$$
Proving that $\mathcal{L}(G) = \mathcal{L}(P)$

Claim 24

$A_{pq} \xrightarrow{*} w \in \Sigma^* \iff (q, \varepsilon) \in \hat{\delta}(p, w, \varepsilon)$

This yields that $\mathcal{L}(G) = \mathcal{L}(P)$, by taking $p = q_0$ and $q = q_a$. 
Proving that $L(G) = L(P)$

Claim 24

$$A_{pq} \xrightarrow{*} w \in \Sigma^* \iff (q, \varepsilon) \in \hat{\delta}(p, w, \varepsilon)$$

This yields that $L(G) = L(P)$, by taking $p = q_0$ and $q = q_a$.

- Proving $A_{pq} \xrightarrow{*} w \in \Sigma^* \implies (q, \varepsilon) \in \hat{\delta}(p, w, \varepsilon)$
Proving that $\mathcal{L}(G) = \mathcal{L}(P)$

### Claim 24

$A_{pq} \xrightarrow{*} w \in \Sigma^* \iff (q, \varepsilon) \in \hat{\delta}(p, w, \varepsilon)$

This yields that $\mathcal{L}(G) = \mathcal{L}(P)$, by taking $p = q_0$ and $q = q_a$.

- Proving $A_{pq} \xrightarrow{*} w \in \Sigma^* \implies (q, \varepsilon) \in \hat{\delta}(p, w, \varepsilon)$

  By induction on the (minimal) number of derivation steps used to derive $w$, see next slides
Proving that $\mathcal{L}(G) = \mathcal{L}(P)$

**Claim 24**

$$A_{pq} \xrightarrow{*} w \in \Sigma^* \iff (q, \varepsilon) \in \hat{\delta}(p, w, \varepsilon)$$

This yields that $\mathcal{L}(G) = \mathcal{L}(P)$, by taking $p = q_0$ and $q = q_a$.

- **Proving** $A_{pq} \xrightarrow{*} w \in \Sigma^* \implies (q, \varepsilon) \in \hat{\delta}(p, w, \varepsilon)$

  By induction on the (minimal) number of derivation steps used to derive $w$, see next slides

- **Proving** $(q, \varepsilon) \in \hat{\delta}(p, w, \varepsilon) \implies A_{pq} \xrightarrow{*} w \in \Sigma^*$
Proving that $L(G) = L(P)$

**Claim 24**

$$A_{pq} \xrightarrow{*} w \in \Sigma^* \iff (q, \varepsilon) \in \hat{\delta}(p, w, \varepsilon)$$

This yields that $L(G) = L(P)$, by taking $p = q_0$ and $q = q_a$.

- **Proving** $A_{pq} \xrightarrow{*} w \in \Sigma^* \iff (q, \varepsilon) \in \hat{\delta}(p, w, \varepsilon)$

  By induction on the (minimal) number of derivation steps used to derive $w$, see next slides

- **Proving** $(q, \varepsilon) \in \hat{\delta}(p, w, \varepsilon) \implies A_{pq} \xrightarrow{*} w \in \Sigma^*$

  By induction on the (minimal) number of step it took $P$ to process $w$, DIY
Proving $A_{pq} \rightarrow^* w \in \Sigma^* \iff (q, \varepsilon) \in \hat{\delta}(p, w, \varepsilon)$

Assume first derivation is $A_{pq} \rightarrow A_{p,r} A_{r,q}$. 
Proving $A_{pq} \rightarrow^* w \in \Sigma^* \implies (q, \varepsilon) \in \hat{\delta}(p, w, \varepsilon)$

Assume first derivation is $A_{pq} \rightarrow A_{p,r}A_{r,q}$.
We apply induction on $A_{r,s}$ and $A_{p,r}$ separately.
Proving $A_{pq} \rightarrow^* w \in \Sigma^* \iff (q, \varepsilon) \in \hat{\delta}(p, w, \varepsilon)$, second case

Assume first derivation is $A_{pq} \rightarrow aA_{r,s}b$ (hence, $w = aw'b$).
Proving \( A_{pq} \xrightarrow{*} w \in \Sigma^* \implies (q, \varepsilon) \in \delta(p, w, \varepsilon) \), second case

Assume first derivation is \( A_{pq} \rightarrow aA_{r,s}b \) (hence, \( w = aw'b \)).

- By i.h., \( (s, \varepsilon) \in \delta(r, w', \varepsilon) \)
Proving $A_{pq} \xrightarrow{*} w \in \Sigma^* \implies (q, \varepsilon) \in \hat{\delta}(p, w, \varepsilon)$, second case

Assume first derivation is $A_{pq} \rightarrow aA_{r,s}b$ (hence, $w = aw'b$).

- By i.h., $(s, \varepsilon) \in \hat{\delta}(r, w', \varepsilon)$
- By definition of the grammar (?), it follows that $(q, \varepsilon) \in \hat{\delta}(p, w, \varepsilon)$
A short summary

- Regular Languages $\equiv$ Finite Automata.
- Context Free Languages $\equiv$ Push Down Automata.
- Closure properties of regular languages and of CFLs.
- Most algorithmic problems for finite automata are solvable.
- Some algorithmic problems for finite automata are not solvable.
- Pumping lemmata for both classes of languages.
- There are additional languages out there.
View over the horizon

- Enumerable
- Decidable
- Context free
- Regular