This lecture covers Chapter 7 of HMU: Properties of CFLs

- Chomsky Normal Form
- Pumping Lemma for CFGs
- Closure Properties of CFLs
- Decision Properties of CFLs

Additional Reading: Chapter 7 of HMU.
Chomsky Normal Forms

> A normal or canonical form (be it in algebra, matrices, or languages) is a standardized way of presenting the object (in this case, languages).

> A normal form for CFGs provides a prescribed structure to the grammar without compromising on its power to define all context-free languages.

> Every non-empty language \( L \) with \( \epsilon \notin L \) has Chomsky Normal Form grammar \( G = (V, T, \mathcal{P}, S) \) where every production rule is of the form:
  
  > \( A \rightarrow BC \) for \( A, B, C \in V \), or
  
  > \( A \rightarrow a \) for \( A \in V \) and \( a \in T \).

> CNF disallows:
  
  > \( A \rightarrow \epsilon \) [\( \epsilon \)-productions].
  
  > \( A \rightarrow B \) for \( A, B \in V \). [Unit productions].
  
  > \( A \rightarrow B_1 \cdots B_k \), \( A \in V \), \( B_i \in V \cup T \) for \( k \geq 2 \) [Complex productions].
Towards CNF [Step 1: Remove $\epsilon$-Productions]

- $\epsilon$-production: $A \rightarrow \epsilon$ for some $A \in V$.

- Let us call a variable $A \in V$ as **nullable** if $A \Rightarrow^* \epsilon$.

- We can identify nullable variables as follows:
  - **Basis:** $A \in V$ is nullable if $A \rightarrow \epsilon$ is a production rule in $P$.
  - **Induction:** $B \in V$ is nullable if $B \rightarrow A_1 \cdots A_k$ is in $P$, and each $A_i$ is nullable.

**Procedure to Eliminate $\epsilon$-Productions**

- Given $G = (V, T, P, S)$ define $G_{\text{no-} \epsilon} = (V, T, P_{\text{no-} \epsilon}, S)$ as follows:
  1. Start with $P_{\text{no-} \epsilon} = P$. Find all nullable variables of $G$.
  2. For each production rule in $P$ do the following:
     - If the body contains $k > 0$ nullable variables, add $2^k$ productions to $P_{\text{no-} \epsilon}$ obtained by choosing a subset of nullable variables and replacing each by $\epsilon$.
  3. Delete any production in $P_{\text{no-} \epsilon}$ of the form $Y \rightarrow \epsilon$ for any $Y \in V$.

For example, suppose that in a given grammar, $B, D$ are nullable and $C$ is not. If $A \rightarrow BCD$ is a rule in $P$, then $A \rightarrow BCD|CD|BC|C$ are rules in $P_{\text{no-} \epsilon}$. Similarly, if $A \rightarrow BD$ is a rule in $P$, then $A \rightarrow BD|B|D$ are rules in $P_{\text{no-} \epsilon}$.
Towards CNF [Step 1: Remove $\epsilon$-Productions]

An Example

Suppose $G = (\{A, B, C\}, \{0, 1\}, \mathcal{P}, A)$ with $\mathcal{P}$: $A \rightarrow BC$; $B \rightarrow 0B|\epsilon$; $C \rightarrow C11|\epsilon$.

- $B$ and $C$ are nullable since $B \rightarrow \epsilon$ and $C \rightarrow \epsilon$. Then, $A$ is also nullable.

- Define $G_{no-\epsilon} = (\{A, B, C\}, \{0, 1\}, \mathcal{P}_{no-\epsilon}, A)$ with $\mathcal{P}_{no-\epsilon}$ containing
  - $A \rightarrow BC|B|C|\epsilon$
  - $B \rightarrow 0A|\epsilon$
  - $C \rightarrow C11|\epsilon$

Theorem 7.1.1

The above induction procedure described in Slide 1 identifies all nullable variables.

Theorem 7.1.2

$L(G_{no-\epsilon}) = L(G) \setminus \{\epsilon\}$.\(^a\)

\(^a\)Proof in the Additional Proofs Section at the end
Towards CNF [Step 2: Remove Unit Productions]

> Given a grammar $G$ and variables $A, B \in V$, we say $(A, B)$ form a **unit pair** if $A \xrightarrow{G}^* B$ using unit productions alone.

> We can identify unit pairs as follows:

  > Basis: For each $A \in V$, $(A, A)$ is a unit pair (since $A \xrightarrow{G}^* A$).

  > Induction: If $(A, B)$ is a unit pair, and $B \rightarrow C$ is a production in $\mathcal{P}$, then $(A, C)$ is a unit pair.

> Suppose $A \rightarrow BC$ and $C \rightarrow \epsilon$ are productions then $A \xrightarrow{G}^* B$, but $(A, B)$ is **not** a unit pair.

**Procedure to Eliminate Unit Productions**

> Given $G = (V, T, \mathcal{P}, S)$ define $G_{\text{no-unit}} = (V, T, \mathcal{P}_{\text{no-unit}}, S)$ as follows:

  1. Start with $\mathcal{P}_{\text{no-unit}} = \mathcal{P}$. Find all unit pairs of $G$.

  2. For every unit pair $(A, B)$ and non-unit production rule $B \rightarrow \alpha$, add rule $A \rightarrow \alpha$ to $\mathcal{P}_{\text{no-unit}}$.

  3. Delete **all** unit production rules in $\mathcal{P}_{\text{no-unit}}$. 
Towards CNF [Step 2: Remove Unit Productions]

An Example

Suppose $G = (\{A, B, C, D\}, \{a, b\}, \mathcal{P}, A)$ with $\mathcal{P}$:

$A \rightarrow B|aC$; $B \rightarrow A|bD$; $C \rightarrow aC|\epsilon$; $D \rightarrow bD|\epsilon$.

$\triangleright$ $(A, B)$ and $(B, A)$ are the only two pairs of unit variables.

$\triangleright$ Define $G_{\text{no-unit}} = (\{A, B, C, D\}, \{a, b\}, \mathcal{P}_{\text{no-unit}}, A)$ with $\mathcal{P}_{\text{no-unit}}$ containing

- $A \rightarrow aC|bD$  
- $B \rightarrow bD|aC$
- $C \rightarrow aC|\epsilon$
- $D \rightarrow bD|\epsilon$

$\triangleright$ Note: Rules with $B$ being the head can never be used.

Theorem 7.1.3

The induction procedure on Slide 5 identifies all unit pairs.

Theorem 7.1.4

$L(G_{\text{no-unit}}) = L(G)$.\(^b\)

\(^b\)Outline of the proof is given in the Additional Proofs Section at the end
Towards CNF [Step 3: Remove Useless Variables]

- A symbol \( X \in V \cup T \) is said to be
  - **generating** if \( X \xrightarrow{\star}_G w \) for some \( w \in T^\star \);
  - **reachable** if \( S \xrightarrow{\star}_G \alpha X \beta \) for some \( \alpha, \beta \in (V \cup T)^\star \); and
  - **useful** if \( S \xrightarrow{\star}_G \alpha X \beta \xrightarrow{\star}_G w \) for some \( w \in T^\star \) and \( \alpha, \beta \in (V \cup T)^\star \).
    (Useful \( \Rightarrow \) Reachable + Generating)

- Given a grammar \( G \), we can identify generating variables as follows:
  - **Basis**: For each \( s \in T \), \( s \xrightarrow{\star}_G s \). So \( s \) is generating.
  - **Induction**: If \( A \xrightarrow{\star}_G \alpha \), and every symbol of \( \alpha \) is generating, so is \( A \).

- Given a grammar \( G \), we can identify reachable variables as follows:
  - **Basis**: \( S \xrightarrow{\star}_G S \) so \( S \) is reachable.
  - **Induction**: If \( A \xrightarrow{\star}_G \alpha \), and \( A \) is reachable, so is every symbol of \( \alpha \).
Towards CNF [Step 3: Remove Useless Variables]

Procedure to Eliminate Useless Variables

> Given $G = (V, T, \mathcal{P}, S)$ define $G_G = (V_G, T, \mathcal{P}_G, S)$ as follows:
  > Find all generating symbols of $G$
  > $V_G$ is the set of all generating variables.
  > $P_G$ is the set of production rules involving only generating symbols.

> Now, define $G_{GR} = (V_{GR}, T_{GR}, \mathcal{P}_{GR}, S)$ as follows:
  > Find all reachable symbols of $G_G$
  > $V_{GR}$ is the set of all reachable variables.
  > $P_{GR}$ is the set of production rules involving only reachable symbols.

The Order of Eliminating Variables is Important!

> Consider $G = (\{A, B, S\}, \{0, 1\}, \mathcal{P}, S)$ with $\mathcal{P} : S \rightarrow AB|0; A \rightarrow 1A; B \rightarrow 1$.

> $A$ is not generating. Removing $A$ and the rules $S \rightarrow AB$ and $A \rightarrow 1A$ results in $B$ being unreachable. Removing $B$ and $B \rightarrow 1$ yields $G_{GR} = (\{S\}, \{0\}, S \rightarrow 0, S)$.

> Reversing the order, we first see that all symbols are reachable; removing then the non-generating symbol $A$ and production rules $S \rightarrow AB$ and $A \rightarrow 1A$ yields $G_{RG} = (\{B, S\}, \{0\}, S \rightarrow 0$ and $B \rightarrow 0, S)$. But $B$ is unreachable now!
Towards CNF [Step 3: Remove Useless Variables]

Theorem 7.1.5

The induction procedure on Slide7 identifies all generating variables.

Theorem 7.1.6

The induction procedure on Slide7 identifies all reachable variables.

Theorem 7.1.7

(1) $L(G) = L(G_{GR})$; and (2) Every symbol in $G_{GR}$ is useful.\(^c\)

\(^c\)Proof in the Additional Proofs Section at the end
Towards CNF [Step 4: Remove Complex Productions]

Procedure to Eliminate Complex Productions

- Given \( G = (V, T, P, S) \), define \( \hat{G} = (\hat{V}, T, \hat{P}, S) \) as follows:
  - Start with \( \hat{G} = G \) and do the following operations.
  - For every terminal \( a \in T \) that appears in the body of length 2 or more, introduce a new variable \( A \) and a new production rule \( A \rightarrow a \).
  - Replace the occurrence all such terminals in the body of length 2 or more by the introduced variables.
  - Replace every rule \( A \rightarrow B_1 \cdots B_k \) for \( k > 2 \), by introducing \( k - 2 \) variables \( D_1, \ldots, k - 2 \), and by replacing the rule by the following \( k - 1 \) rules:
    
    \[
    A \rightarrow B_1 D_1, \quad D_2 \rightarrow B_3 D_3, \quad \cdots, \quad D_{k-2} \rightarrow B_{k-1} B_k \\
    D_1 \rightarrow B_2 D_2, \quad \cdots, \quad D_{k-3} \rightarrow B_{k-2} D_{k-2}
    \]

- Note: Each introduced variable appears in the head exactly once.

Theorem 7.1.8

\[ L(G) = L(\hat{G}). \]

\( ^d \) Outline of the proof is given in the Additional Proofs Section at the end.
The Chomsky Normal Form

Theorem 7.1.9

For every context-free language $L$ containing a non-empty string, there exists a grammar $G$ in Chomsky Normal Form such that $L \setminus \{\epsilon\} = L(G)$.

Proof

> Since $L$ is a CFL, it must correspond to some CFG $G$.
> Eliminate $\epsilon$ productions (Step 1) to derive a grammar $G_1$ from $G$ such that $L(G_1) = L(G) \setminus \{\epsilon\}$.
> Eliminate unit productions (Step 2) to derive a grammar $G_2$ from $G_1$ such that $L(G_2) = L(G_1)$.
> Eliminate useless variables (Step 3) to derive a grammar $G_3$ from $G_2$ such that $L(G_3) = L(G_2)$.
> Eliminate complex productions (Step 4) to derive a grammar $G_4$ from $G_3$ such that $L(G_4) = L(G_3)$.
> $G_4$ contains no $\epsilon$-productions, no unit productions, no useless variables, and no productions with body consisting of 3 or more symbols; Hence $G_4$ is in CNF.
Pumping Lemma

**Theorem 7.2.1**

Let \( L = \emptyset \) be a CFL. Then there exists \( n > 0 \) such that for any string \( z \in L \) with \( |z| \geq n \),

\[
\begin{align*}
(1) \quad z &= uvwx; \\
(2) \quad vx &\neq \epsilon; \\
(3) \quad |vwx| &\leq n; \\
&uv^iwx^iy \in L \text{ for any } i \geq 0.
\end{align*}
\]

**Proof**

- Since the claim only pertains to non-empty strings, we can show the claim for \( L \setminus \{ \epsilon \} \).
- Let CNF grammar \( G \) generate \( L \setminus \{ \epsilon \} \). Choose \( n = 2^m \) where \( m = |V| \) in \( G \).
- Pick any \( z \) with \( |z| \geq n \).
- Depth \( d \geq m + 1 \).
Proof

\[ \text{Since depth } D = m + 1 \text{ or more, there must be a path with } m + 1 \text{ edges in the tree.} \]
\[ \text{There must be two labels that match!} \]
\[ \text{The claim follows from the following pictorial argument.} \]
Uses of Pumping Lemma

- Pumping lemma can be used to argue that some languages are **not** CFLs.

**Proof that** \( L = \{0^n1^n2^n : n \geq 0\} \) **is Not Context-Free**

- Suppose it were.
- There exists an \( n \) such that for strings \( z \) longer than \( n \) pumping lemma applies.
- Applying pumping lemma to \( z = 0^n1^n2^n \), we see that \( z = uvwxy \) such that \( |vwx| \leq n \).

\[
\begin{array}{ccccccc}
0 & \cdots & 0 & 1 & \cdots & 1 & 2 & \cdots & 2 \\
\hline
\hline
\hline
\end{array}
\]

- \( vwx \) cannot contain both zeros and twos. Two cases arise:
  - Case (a): Suppose \( vwx \) contains no 2s. Then \( uwx \) contains fewer 1s or 0s than 2s. Such a string is not in \( L \).
  - Case (b): Suppose \( vwx \) contains no 1s. Then \( uwx \) contains fewer 1s or 2s than 1s. Such a string is not in \( L \).
Substitution of Symbols with Languages

Let $L$ be a CFL on $\Sigma_1$, and let $h$ be a substitution, i.e., for each $a \in \Sigma_1$, $h(a)$ is a language over some alphabet $\Sigma_a$. We can extend the substitution to words by concatenation, i.e., $h(s_1 \cdots s_k) = h(s_1)h(s_2)\cdots h(s_k)$. One can then extend the substitution to languages by unioning, i.e.,

$$h(L) := \bigcup_{s_1 \cdots s_\ell \in L} h(s_1 \cdots s_\ell) = \bigcup_{s_1 \cdots s_\ell \in L} h(s_1) \cdots h(s_\ell)$$

i.e., $h(L)$ is the language formed by substituting each symbol in a string in the language $L$ by a corresponding language.

An Example

Suppose $L = \{a^n b^n : n \geq 0\}$ and $h(a) = \{0\}$ and $h(b) = \{1, 11\}$. Then,

$$h(L) = \{0^n 1^m : n \leq m \leq 2n\}$$

Theorem 7.3.1

If $L$ is a CFL over $\Sigma_1$ and $h(s)$ is a CFL for every $s \in \Sigma_1$, then $h(L)$ is also a CFL.
Substitution of Symbols with Languages

Proof of Theorem 7.3.1

> Let $G = (V, \Sigma_1, \mathcal{P}, S)$ be a grammar that generates $L$.
> Let for $a \in \Sigma_1$, let $G_a = (V_a, \Sigma_a, \mathcal{P}_a, S_a)$ be a grammar that generates $h(a)$.
> WLOG, assume that $V \cap V_a = \emptyset$ for each $a \in \Sigma_1$.
> Now define $\hat{G} = (V, \{S_a : a \in \Sigma_1\}, \hat{\mathcal{P}}, S)$ by
  > Every rule of $\hat{\mathcal{P}}$ is a rule of $\mathcal{P}$ obtained by replacing each $a \in \Sigma_1$ by $S_a$.
  > For example, $X \rightarrow aXb$ in $\mathcal{P}$ will correspond to $X \rightarrow S_aXS_b$ in $\hat{\mathcal{P}}$ if $a, b \in \Sigma_1$.
> Let $G_{sub} = (V \cup (\bigcup_{a \in \Sigma_1} V_a), \bigcup_{a \in \Sigma_1} \Sigma_a, \hat{\mathcal{P}} \cup (\bigcup_{a \in \Sigma_1} \mathcal{P}_a), S)$
> Claim: $G_{sub}$ generates $h(L)$.
> Note that $w \in h(L)$ can be written as $w_{a_1} \cdots w_{a_\ell}$ for $w_{a_i} \in h(a_i)$ for each $i$, and for some $a_1 \cdots a_\ell \in L$.

\[
\begin{align*}
S \Rightarrow^* & \ a_1 \cdots a_\ell \quad \text{(in } G) \quad \Downarrow \\
& \quad \Downarrow \\
S \Rightarrow & \ S_{a_1} \cdots S_{a_\ell} \quad \text{(in } \hat{G} \text{ as well as } G_{sub}) \\
& \quad \Downarrow \\
S \Rightarrow & \ S_{a_1} \cdots S_{a_\ell} \Rightarrow \ w_1 \ S_{a_2} \cdots S_{a_\ell} \Rightarrow \ w_{a_1} \ w_{a_2} \ S_{a_3} \cdots S_{a_\ell} \Rightarrow \cdots \Rightarrow \ w_{a_1} \cdots w_{a_\ell}
\end{align*}
\]
Closure under substitution means...

Closure under

- (Finite) Union: Let $L = \{1, 2, \ldots, k\}$ and $h(i) = L_i$ be a CFL for each $i = 1, \ldots, k$. By Theorem 7.3.1, $h(L) = L_1 \cup \cdots \cup L_k$ is a CFL.

- (Finite) Concatenation: Let $L = \{a_1a_2\cdots a_k\}$ and $h(a_i) = L_{a_i}$ be a CFL for each $i = 1, \ldots, k$. By Theorem 7.3.1, $h(L) = L_{a_1} \cdots L_{a_k}$ is a CFL.

- Kleene-\(^*\) closure: Let $L = \{a\}^*$ and $h(a) = L_a$ be a CFL. By Theorem 7.3.1, $h(L) = (L_a)^*$ is a CFL.

- Positive closure: Let $L = \{a\}^+ := \{a^n : n \geq 1\}$ and $h(a) = L_a$ be a CFL. By Theorem 7.3.1, $h(L) = L_a(L_a)^*$ is a CFL.

- Homomorphism: Let $L$ be a CFL and $g$ be a homomorphism (i.e., $h$ maps symbols to strings of symbols over some alphabet). Define $h(a) = \{s(a)\}$, which is a regular/CF language. Then, $h(L) = s(L)$ and by Theorem 7.3.1, it is a CFL.
Some More Closure Properties - 1

Theorem 7.3.2

*If* $L$ *is CFL, then so is* $L^R$.

Proof of Theorem 7.3.2

> If $G = (V, T, \mathcal{P}, S)$ generates $L$, then $G^R = (V, T, \mathcal{P}^R, S)$ generates $L^R$ where $A \xrightarrow{} X_1 \cdots X_\ell$ in $\mathcal{P} \iff A \xrightarrow{} (X_1 \cdots X_\ell)^R = X_\ell X_{\ell-1} \cdots X_1$ in $\mathcal{P}^R$ \hspace{1cm} (1)

Theorem 7.3.3

*If* $L$ *is a CFL, $R$ *is a regular language, then* $L \cap R$ *is a CFL.*

Proof of Theorem 7.3.3

> Product PDA Approach: Run the PDA accepting $L_1$ and DFA accepting $L_2$ in parallel. Accept input string iff both machines are in one of their respective final states.

Corollary

*If* $L$ *is a CFL, and* $R$ *is regular, then* $L \setminus R = L \cap R^c$ *is regular*
Some More Closure Properties - 2

**Theorem 7.3.4**

If \( L \) is a CFL, \( h^{-1}(L) = \{w : h(w) \in L\} \) is also a CFL.

**A Coarse Outline of Proof of Theorem 7.3.4**

> Note that editing the input tape is not a valid PDA operation.
> To fix that, we need to alter the state of the PDA \( P \) to store \( h(a) \) in the state itself.

> Let \( L \) be a language over \( \{0, 1\} \) and \( h(0) = aa \) and \( h(1) = bbb \).
> Let the states of PDA \( P \) be \( q_0, \ldots, q_k \). Then, the PDA that accepts \( h^{-1}(L) \) has 6\( k \) states, namely \((q_i, aa), (q_i, a), (q_i, \epsilon), (q_i, bbb), (q_i, bb), \) and \((q_i, b)\).
> The transition between states of \( P' \) is defined as if the second component is the input tape. When the second component is empty, the PDA has the choice to read an input symbol \( a \) and move into a state with \( h(a) \) as the second component.
Some Non-Closure Properties

> CFLs are not closed under intersection.
>  
> Let \( L_1 = \{0^n1^n2^m : n, m \geq 0\}, L_2 = \{0^n1^m2^n : n, m \geq 0\} \). Both are CFLs. However, \( L_1 \cap L_2 = \{0^n1^n2^n : n \geq 0\} \) is not a CFL.

> CFLs are not closed under complementation.
>  
> Suppose CFLs are closed under complementation. Let \( L_1, L_2 \) be the aforementioned CFLs. Then \( L_1 \cap L_2 = (L_1^c \cup L_2^c)^c \) must be a CFL, but it is not. (See Slide 14). Hence, CFLs cannot be closed under complementation.
>  
> Note: There exist CFLs \( L \) such that \( L^c \) is a CFL as well.
Emptiness and Membership

Conversion of a grammar $G$ to a corresponding PDA, PDA to a corresponding grammar $G$, and a grammar to CNF can each be achieved in polynomial time.

Emptiness of a CFL $L$

- Let a grammar $G = (V, T, P, S)$ generating the language $L$ be given. (If PDA is given, convert it to a grammar $G$).
- $G$ is non-empty $\iff$ $S$ is generating.
Emptiness and Membership

Emptiness of a CFL $L$

> Given $G = (V, T, P, S)$ and $w = a_1 \cdots a_\ell$ we identify $\ell(\ell + 1)/2$ sets $E_{i,j}$
  
  $1 \leq i \leq j \leq n$

> $E_{i,j}$ corresponds to all variables that can derive $a_i \cdots a_j$.

> $E_{i,j}$'s are identified from bottom to top, left to right by the following induction.

  > Basis: For each $i = 1, \ldots, \ell$, $E_{i,i}$ contains all variables $X$ such that $X \rightarrow a_i$.

  > Induction: For each $i = 1, \ldots, \ell$ and $j > i$, $E_{ij}$ contains $X$ if:

  1. $X \rightarrow YZ$
  2. $Y \in E_{i,i'}$ and $Z \in E_{i'+1,j}$ for some $i \leq i' < j$.

> $S \in E_{1,\ell} \iff w \in L(G)$. 

---

![Diagram showing the identification of sets $E_{i,j}$ from bottom to top, left to right.](image-url)
Some Undecidable Questions about CFGs and CFLs

- Is a given grammar unambiguous/ambiguous?
- Is a given CFL inherently ambiguous?
- Is the intersection of two CFLs empty?
- Are two CFLs identical?
- Is a given CFL equal to $\Sigma^*$?
Additional Proofs

Proof of Theorem 7.1.2

\(\Leftarrow\) Construct a parse tree with yield \(w \in L(G) \setminus \{\epsilon\}\). Identify a maximal subtree, rooted at say \(X\), whose yield is \(\epsilon\). Delete \(X\) and its subtree. Repeat until no such subtrees remain. In this illustrative example below, suppose that there is only one subtree with \(\epsilon\) yield; let \(B\) be its label and let \(A \rightarrow BCD\) be the production that introduced \(B\). Now, delete \(B\) and its subtree. This new subtree is a parse tree for \(G_{\text{no-}\epsilon}\) with yield \(w\) since \(A \rightarrow CD\) is a valid production rule in \(P_{\text{no-}\epsilon}\) [Why? \(B\) is nullable].

\(\Rightarrow\) Construct a parse tree with yield \(w \in L(G_{\text{no-}\epsilon})\). Identify production rules (used in the tree) that are not in \(P\). For each such rule, find an appropriate rule by appending nullable variables. To the parse tree, add the corresponding nullable variable(s) and a zero-yield subtrees to transform it to a parse tree for \(G\).

In the example, the portion of the parse tree in yellow corresponds to the rule \(A \rightarrow CD\); then there must be some rule in \(P\) (namely \(A \rightarrow BCD\)) such that the added variable(s) (\(B\) in this case) is nullable. So we add a child node with label \(B\) to the node with label \(A\) and append a sub-tree of yield \(\epsilon\) rooted at \(B\). This is now a parse tree for \(G\) with yield \(w\).
Outline of Proof of Theorem 7.1.4

$L(G_{\text{no-unit}}) \subseteq L(G)$: By definition, $A \rightarrow \gamma$ in $P_{\text{no-unit}}$ iff there exists a $B \in V$ such that $A \xrightarrow{\star \hspace{1cm} G} B$ and $B \rightarrow \gamma$ in $P$.

- Thus, every production rule $A \rightarrow \gamma$ of $P_{\text{no-unit}}$ is effectively a derivation $A \xrightarrow{\star \hspace{1cm} G} \alpha$ in $G$.
- Hence, every derivation of $G_{\text{no-unit}}$ is a derivation of $G$.

$L(G) \subseteq L(G_{\text{no-unit}})$: Consider a derivation of $w \in L(G)$ from $S$.

- Argue that every run of unit productions in $P$ that are used in this derivation must be followed by a non-unit production rule in $P$.
- Each such run of unit productions in $P$ followed by a non-unit production can be condensed to a single production in $P_{\text{no-unit}}$. [See definition of $P_{\text{no-unit}}$]
- The condensed derivation is then a derivation of $w$ using rules in $P_{\text{no-unit}}$. 
Proof of Theorem 7.1.7

(1) \( L(G_{GR}) \subseteq L(G) \) since the alphabets and the rule of \( G_{GR} \) are subsets of those of \( G \).

\( \triangleright \) Suppose \( w \in L(G) \). Then, there must be such a derivation of \( w \) from \( S \):

\[
S \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \gamma_3 \cdots \Rightarrow \gamma_k = w.
\]

Rule: \( R_1 \quad R_2 \quad R_3 \quad R_k \)

\( \triangleright \) Since every variable symbol that appears in this derivation is generating, they and the production rules \( R_1, \ldots, R_k \) used in this derivation will be present in \( G_{GR} \).

\( \triangleright \) Every variable in this derivation is reachable; consequently, the variables that appear and the rules \( R_1, \ldots, R_k \) will be present in \( G_{GR} \). Then, \( w \in L(G_{GR}) \).

(2) A straightforward exercise in verifying the definition on Slide 7. Note that the remaining symbols have to be shown to be useful in \( G_{GR} \), and not in \( G \)!
Outline of Proof of Theorem 7.1.8

- \( L(G) \subseteq L(\hat{G}) \) because every production rule of \( \hat{G} \) has a corresponding equivalent derivation of \( \alpha \) from \( A \) in \( \hat{G} \).

- Consider the parse tree of \( w \in L(\hat{G}) \). If there are no introduced variables, then this is also the parse tree of \( w \) in \( G \) and hence \( w \in L(G) \).

- If there are introduced variables, replace them by the complex production in \( G \) that introduced them in the first place (such replacements are always possible). The resultant tree is a parse tree for \( w \) in \( G \), and hence \( w \in G \).