

*week 4:* **Properties and Normal Forms of Context-free Languages**

This Lecture Covers Chapter 7 of HMU: Properties of Context-free Languages

*slides created by:* Dirk Pattinson, based on material by  
Peter Hoefner and Rob van Glabbeek; with improvements by Pascal Bercher

*convenor & lecturer:* Pascal Bercher

**The Australian National University**

Semester 1, 2025

## Content of this Chapter

- Chomsky Normal Form
- Pumping Lemma for Context-free Languages (CFLs)
- Closure Properties of CFLs
- Decision Properties of CFLs

Additional Reading: Chapter 7 of HMU.

# Chomsky Normal Form (CNF) for CFG

# 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.
- › **Motivation:** Such normal forms can be exploited by algorithms (don't have to deal with all possible cases) and by proofs (same reason: can exploit this structure).
- › 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:
  - ›  $A \longrightarrow BC$  for  $A, B, C \in V$ , or
  - ›  $A \longrightarrow a$  for  $A \in V$  and  $a \in T$ .

and every variable in  $V$  is useful, i.e. appears in the derivation of at least one terminal string: for all  $X \in V$  there is  $\alpha, \beta, w$  such that  $S \xRightarrow{*}_G \alpha X \beta \xRightarrow{*}_G w$ .

- › CNF disallows:
  - ›  ~~$A \longrightarrow \epsilon$~~  [ **$\epsilon$ -productions**].
  - ›  ~~$A \longrightarrow B$~~  for  $A, B \in V$ . [**Unit productions**].
  - ›  ~~$A \longrightarrow B_1 \cdots B_k$~~ ,  $A \in V$ ,  $B_i \in V \cup T$  for  $k \geq 2$  [**Complex productions**].

- › Note that CNF can also be provided if  $\epsilon \in L$ . We only need a few additional steps.

Towards CNF [Step 1: Remove  $\epsilon$ -Productions]

The goal is to eliminate all  $\epsilon$ -productions (see next slide for a definition).

Example: Grammar with  $\epsilon$ -productions

Suppose  $G = (\{A, B, C\}, \{0, 1\}, \mathcal{P}, A)$  with  $\mathcal{P}$ :

- >  $A \longrightarrow BC$
- >  $B \longrightarrow 0B \mid \epsilon$
- >  $C \longrightarrow C11 \mid \epsilon$

How could an equivalent grammar look like without  $\epsilon$ -productions?

Example: Grammar without  $\epsilon$ -productions (with same language as above)

Now,  $G' = (\{A, B, C\}, \{0, 1\}, \mathcal{P}', A)$  with  $\mathcal{P}'$ :

- >  $A \longrightarrow BC \mid \textcolor{red}{B} \mid \textcolor{red}{C} \mid \cancel{\epsilon}$
- >  $B \longrightarrow 0B \mid \textcolor{red}{0} \mid \cancel{\epsilon}$
- >  $C \longrightarrow C11 \mid \textcolor{red}{11} \mid \cancel{\epsilon}$

Note that the  $\epsilon$  is in the first language, but not in the second.

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

Procedure to Eliminate  $\epsilon$ -Productions

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

Examples: Suppose that in a given grammar,  $B, D$  are nullable and  $C$  is not.

- › If  $A \rightarrow BCD$  is a rule in  $\mathcal{P}$ , then  $A \rightarrow BCD|CD|BC|C$  are rules in  $\mathcal{P}_{\text{no-}\epsilon}$ .
- › Similarly, if  $A \rightarrow BD$  is a rule in  $\mathcal{P}$ , then  $A \rightarrow BD|B|D$  are rules in  $\mathcal{P}_{\text{no-}\epsilon}$ .

Towards CNF [Step 1: Remove  $\epsilon$ -Productions]

## Examples

- › The one from Slide 5. (Eliminates  $\epsilon$  from language.)
- › The two from Slide 6. (Languages stay equivalent.)

## Theorem 7.1.1

*The induction procedure described in Slide 6 identifies **all** nullable variables.*

## Theorem 7.1.2

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

---

<sup>a</sup>Proof in the Additional Proofs Section at the end

**Recall:** We could extend the procedure to keep  $\epsilon \in L(G)$ .

**Procedure:** Add a new start symbol with two rules:

- › One that goes into  $\epsilon$  (only if  $\epsilon \in L(G)$ ),
- › one that goes into the original start symbol.

## Towards CNF [Step 2: Remove Unit Productions]

Example: Grammar with Unit Productions

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

>  $A \longrightarrow aC \mid B$

>  $B \longrightarrow bD \mid A$

>  $C \longrightarrow aC \mid \epsilon$

>  $D \longrightarrow bD \mid \epsilon$

How could an equivalent grammar look like without unit productions?

Example: Grammar without Unit Productions

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

>  $A \longrightarrow aC \mid \textcolor{red}{bD} \textcolor{red}{\cancel{\mid B}}$

>  $B \longrightarrow bD \mid \textcolor{red}{aC} \textcolor{red}{\cancel{\mid A}}$

>  $C \longrightarrow aC \mid \epsilon$

>  $D \longrightarrow bD \mid \epsilon$

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

## 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.
- › Note: Suppose  $A \rightarrow BC$  and  $C \rightarrow \epsilon$  are productions then  $A \xRightarrow{*}_G B$ , but  $(A, B)$  is **not** a unit pair. (Though we are going to use this step after the first anyway.)

## 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]

## Example

See Slide 8.

## Theorem 7.1.3

*The induction procedure on Slide 9 identifies **all** unit pairs.*

## Theorem 7.1.4

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

---

<sup>b</sup>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  $X \xRightarrow[G]{*} w$  for some  $w \in T^*$ ;

› **reachable** if  $S \xRightarrow[G]{*} \alpha X \beta$  for some  $\alpha, \beta \in (V \cup T)^*$ ; and

› **useful** if  $S \xRightarrow[G]{*} \alpha X \beta \xRightarrow[G]{*} w$  for some  $w \in T^*$  and  $\alpha, \beta \in (V \cup T)^*$ .

(Useful  $\Rightarrow$  Reachable + Generating, but not necessarily vice versa!)

Suppose  $X \xRightarrow[G]{*} a$ , so  $X$  is generating. Assume  $S \xRightarrow[G]{*} \alpha X \beta$ , so  $X$  is reachable.

Now assume each rule  $A \rightarrow \alpha$  with  $X \in \alpha$  has another variable  $B \in \alpha$  with empty language. So we can't turn  $X$  into a terminal word, although  $X$  is generating!)

› Given a grammar  $G$ , we can identify generating variables as follows:

› Basis: For each  $a \in T$ ,  $a \xRightarrow[G]{*} a$ . So  $a$  is generating.

› Induction: If  $A \rightarrow \alpha$ , and every symbol of  $\alpha$  is generating, so is  $A$ .

› Given a grammar  $G$ , we can identify reachable variables as follows:

› Basis:  $S \xRightarrow[G]{*} S$  so  $S$  is reachable.

› Induction: If  $A \rightarrow \alpha$ , 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.
  - >  $\mathcal{P}_G$  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.
  - >  $\mathcal{P}_{GR}$  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  $\mathcal{P} : S \rightarrow AB \mid 0; A \rightarrow 1A; B \rightarrow 1$ .
- >  $A$  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  $G_{RG} = (\{B, S\}, \{0\}, S \rightarrow 0 \text{ and } B \rightarrow 0, S)$ . But  $B$  is unreachable now!

## Towards CNF [Step 3: Remove Useless Variables]

## Theorem 7.1.5

*The induction procedure on Slide 11 identifies **all** generating variables.*

## Theorem 7.1.6

*The induction procedure on Slide 11 identifies **all** reachable variables.*

## Theorem 7.1.7

- (1)  $L(G) = L(G_{GR})$ ; and
- (2) Every symbol in  $G_{GR}$  is useful.<sup>c</sup>

---

<sup>c</sup>Proof in the Additional Proofs Section at the end

## Towards CNF [Step 4: Remove Complex Productions]

## Procedure to Eliminate Complex Productions

- › Given  $G = (V, T, \mathcal{P}, S)$ , define  $\hat{G} = (\hat{V}, T, \hat{\mathcal{P}}, S)$  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 of 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  $D_1, \dots, D_{k-2}$ , and by replacing the rule by the following  $k - 1$  rules:

$$\begin{array}{ccccccc}
 A \rightarrow B_1 D_1 & & D_2 \rightarrow B_3 D_3 & & \cdots & & D_{k-2} \rightarrow B_{k-1} B_k \\
 D_1 \rightarrow B_2 D_2 & & \cdots & & D_{k-3} \rightarrow B_{k-2} D_{k-2} & & 
 \end{array}$$

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

## Theorem 7.1.8

$$L(G) = L(\hat{G}).^d$$

<sup>d</sup>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 all productions have one terminal or two non-terminals in the body; Hence  $G_4$  is in CNF.

# Pumping Lemma for CFLs

# Pumping Lemma

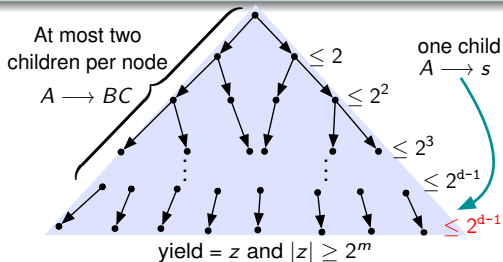
## Theorem 7.2.1

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

(1)  $z = uvwxy$ ; (2)  $vx \neq \epsilon$ ; (3)  $|vwx| \leq n$ ;  $uv^iwx^iy \in L$  for any  $i \geq 0$ .

## Proof

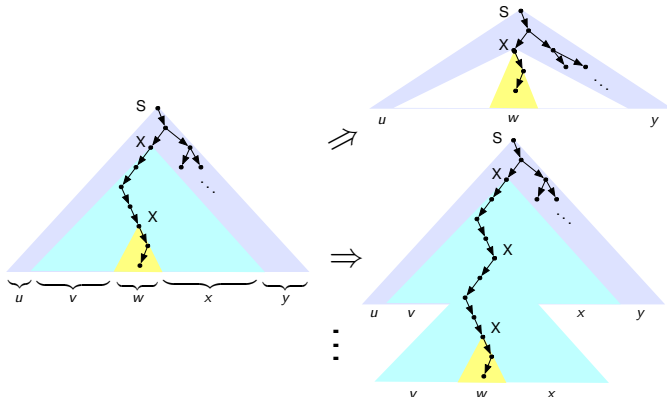
- › Since the claim only pertains to non-empty strings, we can show the claim for  $L \setminus \{\epsilon\}$ .
- › Let CNF grammar  $G$  generate  $L \setminus \{\epsilon\}$ . Choose  $n = 2^m$  where  $m = |V|$  in  $G$ .
- › Pick any  $z$  with  $|z| \geq n$ .
- › Depth  $d \geq m + 1$ .



# Pumping Lemma

## Proof

- › Since depth  $d \geq m + 1$ , there must be a path with at least  $m + 1$  edges.
- › This path has  $m + 2$  nodes, hence  $m + 1$  inner ones, hence  $\geq 2$  must match!
- › Then, the Pumping Lemma claim follows from the following pictorial argument.

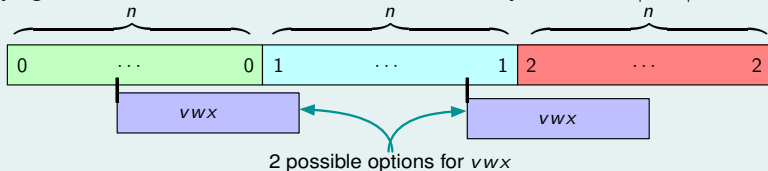


# Uses of Pumping Lemma

- › The Pumping Lemma (PL) can be used to argue that some languages are **not** CFLs.

## Proof that $L = \{0^n 1^n 2^n : n \geq 0\}$ is Not Context-Free

- › Suppose it were.
- › There exists an  $n$  such that for strings  $z$  longer than  $n$  pumping lemma applies.
- › Applying the PL to  $z = 0^n 1^n 2^n$ , we see that  $z = uvwxy$  such that  $|vwx| \leq n$ .



- › There are 3 cases with only one letter. (Show that String is not in  $L$ .)
- ›  $vwx$  cannot contain both zeros and twos. Two cases arise:
  - › Case (a): Suppose  $vwx$  contains no 2s. Then  $uwv$  contains fewer 1s or 0s than 2s. Such a string is not in  $L$ .
  - › Case (b): Suppose  $vwx$  contains no 0s. Then  $uwv$  contains fewer 1s or 2s than 0s. Such a string is not in  $L$ .

# Closure Properties

# 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.,  
 $h(s_1 \cdots s_k) = h(s_1)h(s_2) \cdots h(s_k)$ . // here, we concatenate languages!
- › 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

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

- ›  $h(w) = h(a)h(a)h(b)h(b) = \{0\}\{0\}\{1, 11\}\{1, 11\} = \{00\}\{11, 111, 111, 1111\}$   
 $= \{00\}\{11, 111, 1111\} = \{0011, 00111, 001111\} = \{0^2 1^m : 2 \leq m \leq 2 \cdot 2 = 4\}$
- ›  $h(L) = \{0^n 1^m : n, m \geq 0, \text{ such that } n \leq m \leq 2n\}$

# Substitution of Symbols with Languages

## Theorem 7.3.1

*If  $L$  is a CFL over  $\Sigma_1$  and  $h(a)$  is a CFL for every  $a \in \Sigma_1$ , then  $h(L)$  is also a CFL.*

Let  $G$  be a CFG generating  $L$ .

$$\begin{array}{ccc}
 S \xRightarrow{*} a_1 \cdots a_\ell \quad (\text{in } G) & & \text{For } i = 1, \dots, \ell, \quad S_{a_i} \xRightarrow{*} w_{a_i} \quad (\text{in } G_{a_i}) \\
 \Downarrow & & \Downarrow \\
 S \xRightarrow{*} S_{a_1} \cdots S_{a_\ell} \quad (\text{in } \hat{G} \text{ as well as } G_{\text{sub}}) & & S_{a_i} \xRightarrow{*} w_{a_i} \quad (\text{in } G_{\text{sub}}) \\
 \Downarrow & & \Downarrow \\
 \underbrace{S \xRightarrow{*} S_{a_1} \cdots S_{a_\ell}}_{\Rightarrow} \xRightarrow{*} \underbrace{w_{a_1} S_{a_2} \cdots S_{a_\ell} \xRightarrow{*} w_{a_1} w_{a_2} S_{a_3} \cdots S_{a_\ell} \xRightarrow{*} \cdots \xRightarrow{*} w_{a_1} \cdots w_{a_\ell}}_{\Rightarrow}
 \end{array}$$

# Substitution of Symbols with Languages

## Theorem 7.3.1

*If  $L$  is a CFL over  $\Sigma_1$  and  $h(a)$  is a CFL for every  $a \in \Sigma_1$ , then  $h(L)$  is also a CFL.*

## Proof

- › Let  $G = (V, \Sigma_1, \mathcal{P}, S)$  be a grammar that generates  $L$ .
- › For each  $a \in \Sigma_1$ , 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  $a_1 \cdots a_\ell \in L$ .

# Closure under substitution means...

## Closure under:

- › (Finite) Union: Let  $L_1, \dots, L_k$  CFLs. Then,  $L_1 \cup \dots \cup L_k$  is a CFL.  
Proof: Let  $L = \{1, 2, \dots, k\}$  (which is finite, hence regular, hence a CFL) and  $h(i) = L_i$  for each  $i = 1, \dots, k$ . By Theorem 7.3.1,  $h(L) = L_1 \cup \dots \cup L_k$  is a CFL.
- › (Finite) Concatenation: Let  $L_1, \dots, L_k$  CFLs. Then,  $L_1 \cdots L_k$  is a CFL.  
Proof: Let  $L = \{a_1 a_2 \cdots a_k\}$  (which is finite, hence regular, hence a CFL) and  $h(a_i) = L_i$  for each  $i = 1, \dots, k$ . By Theorem 7.3.1,  $h(L) = L_1 \cdots L_k$  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  $h(a) = \{g(a)\}$ , which is a regular (hence CF) language. Then,  $h(L) = g(L)$  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

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

## 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

- $\triangleright$  Product PDA approach: Run the PDA accepting  $L$  and DFA accepting  $R$  in parallel. Accept input string iff both machines accept.

## Some More Closure Properties – 2

### Theorem 7.3.4

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

### A Coarse Outline of Proof of Theorem 7.3.4, Part 1

#### What we know:

- $h : \Sigma_1 \rightarrow \Sigma_2$  (since homomorphisms can map to another alphabet)
- $L$  is defined over  $\Sigma_2$  and a CFL. Thus there is a PDA  $P$  with  $L(P) = L$ .

#### What we need:

- For  $L' = h^{-1}(L)$  to be a CFL it suffices to show that there is a PDA  $P'$ , such that:  $P'$  accepts  $w$  iff  $h(w) \in L$ , i.e., iff  $P$  accepts  $h(w)$ .

#### Example and Idea:

- Turn each  $w$  into  $h(w)$ , then use  $P$ .
- Let  $\Sigma_1 = \{0, 1\}$ ,  $\Sigma_2 = \{a, b\}$ ,  $h(0) = aa$ ,  $h(1) = bbb$ ,  $w = 011$



## Some More Closure Properties – 2 (cont'd)

### A Coarse Outline of Proof of Theorem 7.3.4, Part 2

#### Problem:

- › A PDA can't manipulate the input string! (Only Turing Machines can do that.)

#### Solution:

- › We store the outcome of  $h$  in the state itself!
- › Recall:  $h(0) = aa$ ,  $h(1) = bbb$ .
- › Let the states of PDA  $P$  be  $q_0, \dots, q_k$ . Then, the PDA  $P'$  that accepts  $h^{-1}(L)$  has  $6k$  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 (e.g.,  $(q_i, aa)$  transitions to  $(q_i, a)$ ). Once the second component is empty, we can move on reading another symbol from  $\{0, 1\}$  and filling the second component again accordingly.

# Some Non-Closure Properties

- > CFLs are not closed under intersection.
  - > Let  $L_1 = \{0^n 1^n 2^m : n, m \geq 0\}$ ,  $L_2 = \{0^n 1^m 2^n : n, m \geq 0\}$ . Both are CFLs. However,  $L_1 \cap L_2 = \{0^n 1^n 2^n : n \geq 0\}$  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 (see slide 23), but it is not. Hence, CFLs cannot be closed under complementation.
  - > Note: There exist CFLs  $L$  such that  $L^c$  is a CFL as well.
- > CFLs are not closed under set difference.
  - > Since CFLs are not closed under complementation, choose a CFL  $L$  such that  $L^c$  is not a CFL. But  $L^c = \Sigma^* \setminus L$  and  $\Sigma^*$  is a CFL. Hence, CFLs are not closed under set difference.
  - > Note: There exist CFLs  $L_1, L_2$  such that  $L_1 \setminus L_2$  is a CFL as well.

# Decision Properties

# Language Emptiness

- › 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.

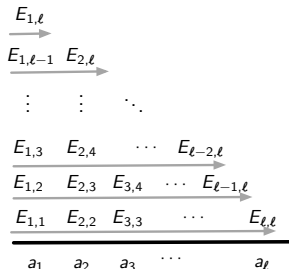
## Emptiness of a CFL $L$

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

# Language Membership – The CYK Algorithm

## Membership of $w$ in a CFL $L$

- › Given CNF  $G = (V, T, \mathcal{P}, S)$  and  $w = a_1 \cdots a_\ell$  we identify  $\sum_{i=1}^{\ell} i = \ell(\ell+1)/2 \in \mathcal{O}(\ell^2)$  sets  $E_{i,j}$ , with  $1 \leq i \leq j \leq \ell$ .
- ›  $E_{i,j}$  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  $i = 1, \dots, \ell$ ,  $E_{i,i}$  contains **all** variables  $X$  such that  $X \rightarrow a_i$ .
  - › Induction: For each  $i = 1, \dots, \ell$  and  $j > i$ ,  $E_{i,j}$  contains  $X$  if:
    - (1)  $X \rightarrow YZ$  (2)  $Y \in E_{i,i'}$  and  $Z \in E_{i'+1,j}$  for some  $i \leq i' \leq j$ .
  - ›  $S \in E_{1,\ell} \iff w \in L(G)$ .



# Some Undecidable Questions about CFGs and CFLs

You might not know yet what “Undecidable” means. Thus:

- › You might want to get back to this in a few weeks!
- › In a nutshell (and quite informally), it implies that there's no algorithm that answers these questions correctly (with yes/no) and always terminates.

Undecidable questions:

- › Is a given grammar unambiguous/ambiguous?
- › Is a given CFL inherently ambiguous?
- › Is the intersection of two CFLs empty?  
(Fun fact: this is used to prove that HTN planning is undecidable.  
We might look into this in week 12!)
- › Are two CFLs identical?
- › Is a given CFL equal to  $\Sigma^*$ ?

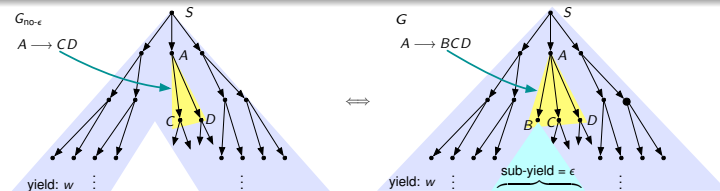
# Additional Proofs

# 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  $\mathcal{P}_{\text{no-}\epsilon}$  [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  $\mathcal{P}$  (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  $w$ .



# Additional Proofs

## 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

$$A \xrightarrow[G]{*} B \text{ and } B \rightarrow \gamma \text{ in } \mathcal{P}.$$

- › Thus, every production rule  $A \rightarrow \gamma$  of  $P_{\text{no-unit}}$  is effectively a derivation  $A \xrightarrow[G]{*} \alpha$  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  $\mathcal{P}$  that are used in this derivation must be followed by a non-unit production rule in  $\mathcal{P}$ .
- › Each such run of unit productions in  $\mathcal{P}$  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  $P_{\text{no-unit}}$ .

# Additional Proofs

## 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  $G$ .

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

$$S \xRightarrow[\text{Rule: } R_1]{G} \gamma_1 \xRightarrow[R_2]{G} \gamma_2 \xRightarrow[R_3]{G} \gamma_3 \cdots \xRightarrow[R_k]{G} \gamma_k = w.$$

‣ Since every variable symbol that appears in this derivation is generating, they and the production rules  $R_1, \dots, R_k$  used in this derivation will be present in  $G_G$ .

‣ Every variable in this derivation is reachable; consequently, the variables that appear and the rules  $R_1, \dots, R_k$  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  $G$ !

# Additional Proofs

## 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  $w \in G$ .

