

# Properties of Context-free Languages



---

Reading: Chapter 7



# Topics

---

- Normal forms
- Pumping lemma for CFLs
- Closure and decision properties of CFLs



# How to “simplify” CFGs?

---



# Three ways to simplify/clean a CFG

---

*(clean)*

1. Eliminate *useless symbols*

*(simplify)*

2. Eliminate  $\epsilon$ -productions

$$A \not\Rightarrow \epsilon$$

3. Eliminate *unit productions*

$$A \not\Rightarrow B$$



# Eliminating useless symbols

---

Grammar cleanup



# Eliminating *useless symbols*

---

A symbol  $X$  is reachable if there exists:

- $S \rightarrow^* \alpha X \beta$

A symbol  $X$  is generating if there exists:

- $X \rightarrow^* w$ ,
  - for some  $w \in T^*$

For a symbol  $X$  to be “useful”, it has to be both reachable *and* generating

- $S \rightarrow^* \alpha X \beta \rightarrow^* w'$ , for some  $w' \in T^*$

reachable      generating



# Algorithm to detect useless symbols

---

1. First, eliminate all symbols that are *not* generating
1. Next, eliminate all symbols that are *not* reachable

Is the order of these steps important,  
or can we switch?

# Example: Useless symbols

- $S \rightarrow AB \mid a$
- $A \rightarrow b$

1.  $A, S$  are generating
2.  $B$  is *not generating* (and therefore is useless)
3.  $\implies$  Eliminating  $B$ ... (i.e., remove any production involving  $B$ )
  1.  $S \rightarrow a$
  2.  $A \rightarrow b$
4. Now,  $A$  is *not reachable* and therefore is useless

5. Simplified G:

1.  $S \rightarrow a$

What would happen if you reverse the order:  
i.e., test reachability before generating?

Will fail to remove:  
 $A \rightarrow b$

$$X \rightarrow^* w$$

## Algorithm to find all generating symbols

- Given:  $G=(V,T,P,S)$
- Basis:
  - Every symbol in  $T$  is obviously generating.
- Induction:
  - Suppose for a production  $A \rightarrow \alpha$ , where  $\alpha$  is generating
  - Then,  $A$  is also generating

$$S \rightarrow^* \alpha X \beta$$

## Algorithm to find all reachable symbols

- Given:  $G=(V,T,P,S)$
- Basis:
  - $S$  is obviously reachable (from itself)
- Induction:
  - Suppose for a production  $A \rightarrow \alpha_1 \alpha_2 \dots \alpha_k$ , where  $A$  is reachable
  - Then, all symbols on the right hand side,  $\{\alpha_1, \alpha_2, \dots, \alpha_k\}$  are also reachable.



# Eliminating $\epsilon$ -productions

---

$A \Rightarrow \epsilon$

~~X~~

What's the point of removing  $\epsilon$  transitions ?

$$A \rightarrow \epsilon$$

# Eliminating $\epsilon$ -productions

Caveat: It is *not* possible to eliminate  $\epsilon$ -productions for languages which include  $\epsilon$  in their word set

So we will target the grammar for the *rest of the language*

Theorem: If  $G=(V,T,P,S)$  is a CFG for a language  $L$ , then  $L-\{\epsilon\}$  has a CFG without  $\epsilon$ -productions

Definition:  $A$  is “nullable” if  $A \rightarrow^* \epsilon$

- If  $A$  is nullable, then any production of the form “ $B \rightarrow CAD$ ” can be simulated by:
  - $B \rightarrow CD \mid CAD$ 
    - This can allow us to remove  $\epsilon$  transitions for  $A$



# Algorithm to detect all nullable variables

---

- Basis:
  - If  $A \rightarrow \varepsilon$  is a production in  $G$ , then  $A$  is nullable  
(note:  $A$  can still have other productions)
- Induction:
  - If there is a production  $B \rightarrow C_1 C_2 \dots C_k$ , where *every*  $C_i$  is nullable, then  $B$  is also nullable



# Eliminating $\epsilon$ -productions

---

Given:  $G=(V,T,P,S)$

Algorithm:

1. Detect all nullable variables in  $G$
2. Then construct  $G_1=(V,T,P_1,S)$  as follows:
  - i. For each production of the form:  $A \rightarrow X_1 X_2 \dots X_k$ , where  $k \geq 1$ , suppose  $m$  out of the  $k$   $X_i$ 's are nullable symbols
  - ii. Then  $G_1$  will have  $2^m$  versions for this production
    - i. i.e, all combinations where each  $X_i$  is either present or absent
  - iii. Alternatively, if a production is of the form:  $A \rightarrow \epsilon$ , then remove it

# Example: Eliminating $\epsilon$ -productions

- Let  $L$  be the language represented by the following CFG  $G$ :
  - $S \rightarrow AB$
  - $A \rightarrow aAA \mid \epsilon$
  - $B \rightarrow bBB \mid \epsilon$

Goal: To construct  $G_1$ , which is the grammar for  $L - \{\epsilon\}$

- Nullable symbols:  $\{A, B\}$
- $G_1$  can be constructed from  $G$  as follows:
  - $B \rightarrow b \mid bB \mid bB \mid bBB$ 
    - $\implies B \rightarrow b \mid bB \mid bBB$
  - Similarly,  $A \rightarrow a \mid aA \mid aAA$
  - Similarly,  $S \rightarrow A \mid B \mid AB$

$G_1$ :

- $S \rightarrow A \mid B \mid AB$
- $A \rightarrow a \mid aA \mid aAA$
- $B \rightarrow b \mid bB \mid bBB$

- Note:  $L(G) = L(G_1) \cup \{\epsilon\}$

# Eliminating unit productions

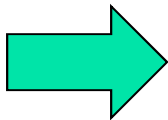
$A \Rightarrow B$  ← B has to be a variable  
X

What's the point of removing unit transitions ?

Will save #substitutions

E.g.,

$A \Rightarrow B \mid \dots$   
 $B \Rightarrow C \mid \dots$   
 $C \Rightarrow D \mid \dots$   
 $D \Rightarrow xxx \mid yyy \mid zzz$



$A \Rightarrow xxx \mid yyy \mid zzz \mid \dots$   
 $B \Rightarrow xxx \mid yyy \mid zzz \mid \dots$   
 $C \Rightarrow xxx \mid yyy \mid zzz \mid \dots$   
 $D \Rightarrow xxx \mid yyy \mid zzz$

*before*

*after*

$$A \rightarrow B$$

# Eliminating unit productions

- Unit production is one which is of the form  $A \rightarrow B$ , where both A & B are variables
- E.g.,
  1.  $E \rightarrow T \mid E+T$
  2.  $T \rightarrow F \mid T^*F$
  3.  $F \rightarrow I \mid (E)$
  4.  $I \rightarrow a \mid b \mid la \mid lb \mid l0 \mid l1$
- How to eliminate unit productions?
  - Replace  $E \rightarrow T$  with  $E \rightarrow F \mid T^*F$
  - Then, upon recursive application wherever there is a unit production:
    - $E \rightarrow F \mid T^*F \mid E+T$  (substituting for T)
    - $E \rightarrow I \mid (E) \mid T^*F \mid E+T$  (substituting for F)
    - $E \rightarrow a \mid b \mid la \mid lb \mid l0 \mid l1 \mid (E) \mid T^*F \mid E+T$  (substituting for I)
    - Now, E has no unit productions
  - Similarly, eliminate for the remainder of the unit productions



# The Unit Pair Algorithm: to remove unit productions

- Suppose  $A \rightarrow B_1 \rightarrow B_2 \rightarrow \dots \rightarrow B_n \rightarrow \alpha$
- Action: Replace all intermediate productions to produce  $\alpha$  directly
  - i.e.,  $A \rightarrow \alpha; B_1 \rightarrow \alpha; \dots B_n \rightarrow \alpha;$

Definition:  $(A,B)$  to be a “*unit pair*” if  $A \rightarrow^* B$

- We can find all unit pairs inductively:
  - Basis: Every pair  $(A,A)$  is a unit pair (by definition). Similarly, if  $A \rightarrow B$  is a production, then  $(A,B)$  is a unit pair.
  - Induction: If  $(A,B)$  and  $(B,C)$  are unit pairs, and  $A \rightarrow C$  is also a unit pair.



# The Unit Pair Algorithm: to remove unit productions

---

Input:  $G=(V,T,P,S)$

Goal: to build  $G_1=(V,T,P_1,S)$  devoid of unit productions

Algorithm:

1. Find all unit pairs in  $G$
2. For each unit pair  $(A,B)$  in  $G$ :
  1. Add to  $P_1$  a new production  $A \rightarrow \alpha$ , for every  $B \rightarrow \alpha$  which is a *non-unit* production
  2. If a resulting production is already there in  $P$ , then there is no need to add it.

# Example: eliminating unit productions

G:

1.  $E \rightarrow T \mid E+T$
2.  $T \rightarrow F \mid T^*F$
3.  $F \rightarrow I \mid (E)$
4.  $I \rightarrow a \mid b \mid la \mid lb \mid l0 \mid l1$



G<sub>1</sub>:

1.  $E \rightarrow E+T \mid T^*F \mid (E) \mid a \mid b \mid la \mid lb \mid l0 \mid l1$
2.  $T \rightarrow T^*F \mid (E) \mid a \mid b \mid la \mid lb \mid l0 \mid l1$
3.  $F \rightarrow (E) \mid a \mid b \mid la \mid lb \mid l0 \mid l1$
4.  $I \rightarrow a \mid b \mid la \mid lb \mid l0 \mid l1$



<i>Unit pairs</i>	<i>Productions (to be added to P<sub>1</sub>)</i>
(E,E)	$E \rightarrow E+T$
(E,T)	$E \rightarrow T^*F$
(E,F)	$E \rightarrow (E)$
(E,I)	$E \rightarrow a \mid b \mid la \mid lb \mid l0 \mid l1$
(T,T)	$T \rightarrow T^*F$
(T,F)	$T \rightarrow (E)$
(T,I)	$T \rightarrow a \mid b \mid la \mid lb \mid l0 \mid l1$
(F,F)	$F \rightarrow (E)$
(F,I)	$F \rightarrow a \mid b \mid la \mid lb \mid l0 \mid l1$
(I,I)	$I \rightarrow a \mid b \mid la \mid lb \mid l0 \mid l1$



# Putting all this together...

- Theorem: If  $G$  is a CFG for a language that contains at least one string other than  $\varepsilon$ , then there is another CFG  $G_1$ , such that  $L(G_1) = L(G) - \varepsilon$ , and  $G_1$  has:
  - no  $\varepsilon$ -productions
  - no unit productions
  - no useless symbols

- Algorithm:

- Step 1) eliminate  $\varepsilon$ -productions
- Step 2) eliminate unit productions
- Step 3) eliminate useless symbols

Again,  
the order is  
important!

Why?



# Normal Forms

---



# Why normal forms?

---

- If all productions of the grammar could be expressed in the same form(s), then:
  - a. It becomes easy to design algorithms that use the grammar
  - b. It becomes easy to show proofs and properties



# Chomsky Normal Form (CNF)

Let  $G$  be a CFG for some  $L - \{\epsilon\}$

## Definition:

$G$  is in **CNF** if all its productions are in one of the following two forms:

- i.  $A \rightarrow BC$                       where  $A, B, C$  are variables, or
- ii.  $A \rightarrow a$                               where  $a$  is a terminal

- Further,  $G$  has no useless symbols

By implication from i and ii:

=> a CNF also cannot contain any unit productions  
(also no  $\epsilon$  productions)



# Is this example G in CNF?

G<sub>1</sub>:

1.  $E \rightarrow E+T \mid T^*F \mid (E) \mid Ia \mid Ib \mid I0 \mid I1$
2.  $T \rightarrow T^*F \mid (E) \mid Ia \mid Ib \mid I0 \mid I1$
3.  $F \rightarrow (E) \mid Ia \mid Ib \mid I0 \mid I1$
4.  $I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

## Checklist:

- G has no  $\epsilon$ -productions
- G has no unit productions
- G has no useless symbols
- But...
  - the normal form for productions is violated

# How to convert a G into CNF?

- Assumption: G has no  $\epsilon$ -productions, unit productions or useless symbols
- 1) For every terminal  $a$  that appears in the body of a production:
  - i. create a unique variable, say  $X_a$ , with a production  $X_a \rightarrow a$ , and
  - ii. replace all other instances of  $a$  in G by  $X_a$
- 2) Now, all productions will be in one of the following two forms:
  - $A \rightarrow B_1 B_2 \dots B_k$  ( $k \geq 3$ )      or       $A \rightarrow a$
- 3) Replace each production of the form  $A \rightarrow B_1 B_2 B_3 \dots B_k$  by:
 

$$\begin{array}{c}
 \longleftrightarrow \\
 B_2 \quad C_2 \\
 \longleftarrow \quad \longrightarrow \\
 B_1 \quad C_1
 \end{array}$$

and so on...

  - $A \rightarrow B_1 C_1 \quad C_1 \rightarrow B_2 C_2 \quad \dots \quad C_{k-3} \rightarrow B_{k-2} C_{k-2} \quad C_{k-2} \rightarrow B_{k-1} B_k$

# Example

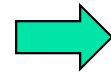
G:

$S \Rightarrow AS \mid BABC$

$A \Rightarrow A1 \mid 0A1 \mid 01$

$B \Rightarrow 0B \mid 0$

$C \Rightarrow 1C \mid 1$



G in CNF:

$X_0 \Rightarrow 0$

$X_1 \Rightarrow 1$

$S \Rightarrow AS \mid BY_1$

$Y_1 \Rightarrow AY_2$

$Y_2 \Rightarrow BC$

$A \Rightarrow AX_1 \mid X_0Y_3 \mid X_0X_1$

$Y_3 \Rightarrow AX_1$

$B \Rightarrow X_0B \mid 0$

$C \Rightarrow X_1C \mid 1$

All productions are of the form:  $A \Rightarrow BC$  or  $A \Rightarrow a$

# Example

G:

1.  $E \rightarrow E+T \mid T^*F \mid (E) \mid Ia \mid Ib \mid I0 \mid I1$
2.  $T \rightarrow T^*F \mid (E) \mid Ia \mid Ib \mid I0 \mid I1$
3.  $F \rightarrow (E) \mid Ia \mid Ib \mid I0 \mid I1$
4.  $I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$



Step (1)

1.  $E \rightarrow EX_+T \mid TX_*F \mid X_{(}EX_{)} \mid IX_a \mid IX_b \mid IX_0 \mid IX_1$
2.  $T \rightarrow TX_*F \mid X_{(}EX_{)} \mid IX_a \mid IX_b \mid IX_0 \mid IX_1$
3.  $F \rightarrow X_{(}EX_{)} \mid IX_a \mid IX_b \mid IX_0 \mid IX_1$
4.  $I \rightarrow X_a \mid X_b \mid IX_a \mid IX_b \mid IX_0 \mid IX_1$
5.  $X_+ \rightarrow +$
6.  $X_* \rightarrow *$
7.  $X_{(} \rightarrow ($
8.  $X_{)} \rightarrow )$
9. ....

Step (2)

1.  $E \rightarrow EC_1 \mid TC_2 \mid X_{(}C_3 \mid IX_a \mid IX_b \mid IX_0 \mid IX_1$
2.  $C_1 \rightarrow X_+T$
3.  $C_2 \rightarrow X_*F$
4.  $C_3 \rightarrow EX_{)}$
5.  $T \rightarrow \dots\dots\dots$
6. ....

# Languages with $\epsilon$

- For languages that include  $\epsilon$ ,
  - Write down the rest of grammar in CNF
  - Then add production “ $S \Rightarrow \epsilon$ ” at the end

E.g., consider:

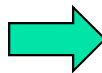
G:

$S \Rightarrow AS \mid BABC$

$A \Rightarrow A1 \mid 0A1 \mid 01 \mid \epsilon$

$B \Rightarrow 0B \mid 0 \mid \epsilon$

$C \Rightarrow 1C \mid 1 \mid \epsilon$



G in CNF:

$X_0 \Rightarrow 0$

$X_1 \Rightarrow 1$

$S \Rightarrow AS \mid BY_1 \mid \epsilon$

$Y_1 \Rightarrow AY_2$

$Y_2 \Rightarrow BC$

$A \Rightarrow AX_1 \mid X_0Y_3 \mid X_0X_1$

$Y_3 \Rightarrow AX_1$

$B \Rightarrow X_0B \mid 0$

$C \Rightarrow X_1C \mid 1$



# Other Normal Forms

---

- Griebach Normal Form (GNF)
  - All productions of the form

$$A \Rightarrow a \alpha$$



# Return of the Pumping Lemma !!

---

Think of languages that cannot be CFL

== think of languages for which a stack will not be enough

e.g., the language of strings of the form  $ww$



# Why pumping lemma?

---

- A result that will be useful in proving languages that *are not* CFLs
  - (just like we did for regular languages)
- But before we prove the pumping lemma for CFLs ....
  - Let us first prove a property about parse trees

Note: any parse tree generated by a CNF will be a binary tree

# The “parse tree theorem”

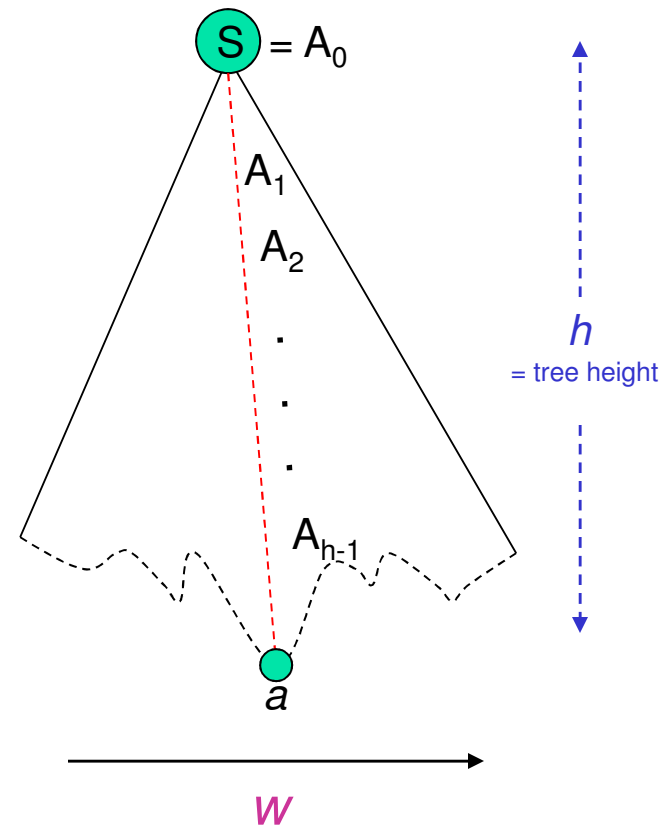
## Given:

- Suppose we have a parse tree for a string  $w$ , according to a CNF grammar,  $G=(V,T,P,S)$
- Let  $h$  be the height of the parse tree

## Implies:

- $|w| \leq 2^{h-1}$

## Parse tree for $w$



# Proof... The size of parse trees

Proof: (using induction on h)

Basis:  $h = 1$

- Derivation will have to be " $S \rightarrow a$ "
- $|w| = 1 = 2^{1-1}$ .

Ind. Hyp:  $h = k-1$

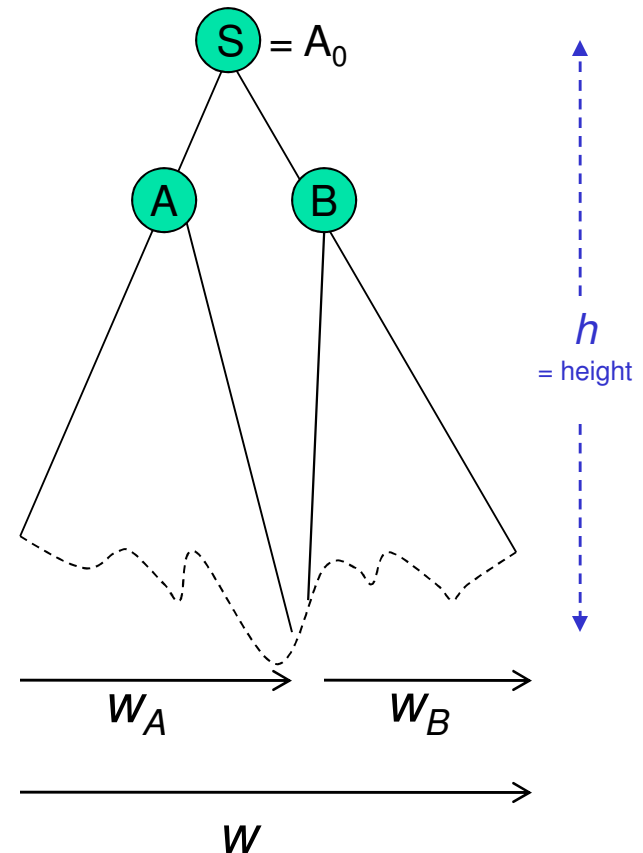
- $|w| \leq 2^{k-2}$

Ind. Step:  $h = k$

S will have exactly two children:  
 $S \rightarrow AB$

- Height of A & B subtrees is at most  $h-1$
- $w = w_A w_B$ , where  $|w_A| \leq 2^{k-2}$   
and  $|w_B| \leq 2^{k-2}$
- $|w| \leq 2^{k-1}$

Parse tree for w





# Implication of the Parse Tree Theorem (assuming CNF)

---

## Fact:

- If the height of a parse tree is  $k$ , then
  - $\implies |w| \leq 2^{k-1}$
- put another way:
  - Assuming CNF, any parse tree of height  $k$  cannot generate a string longer than  $2^{k-1}$

## Implication:

- If  $|w| \geq 2^k$ , then
  - Its parse tree's height is *at least*  $k+1$



# The Pumping Lemma for CFLs

---

Let  $L$  be a CFL.

Then there exists a constant  $N$ , s.t.,

- if  $z \in L$  s.t.  $|z| \geq N$ , then we can write  $z = uvwxy$ , such that:

1.  $|vwx| \leq N$

2.  $vx \neq \varepsilon$

3. For all  $k \geq 0$ :  $uv^kwx^ky \in L$

Note: we are pumping in two places



# Proof: Pumping Lemma for CFL

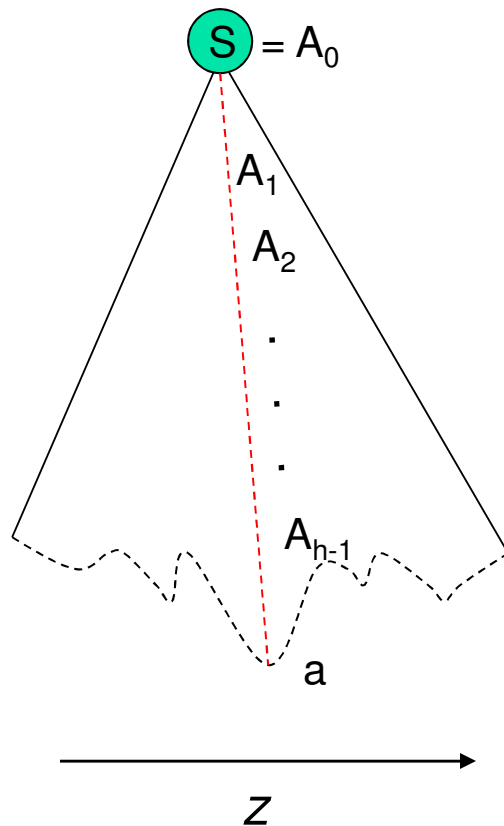
---

- If  $L = \Phi$  or contains only  $\varepsilon$ , then the lemma is trivially satisfied (as it cannot be violated)
- For any other  $L$  which is a CFL:
  - Let  $G$  be a CFG for  $L$  in CNF
  - Let  $m =$  number of variables in  $G$
  - Choose  $N = 2^m$ .
  - Pick some  $z \in L$  s.t.  $|z| \geq N$ 
    - ➔ the parse tree for  $z$  should have height  $\geq m+1$   
(by the parse tree theorem)

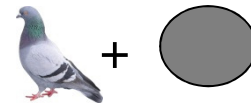
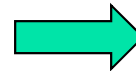
# Parse tree for $z$

Meaning:  
Repetition in the last  $m+1$  variables

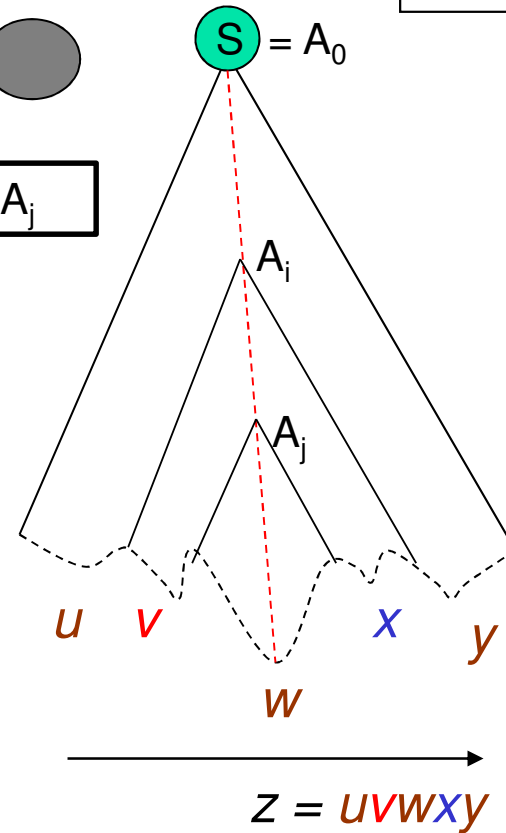
$$h-m \leq i < j \leq h$$



$h \geq m+1$



$$A_i = A_j$$



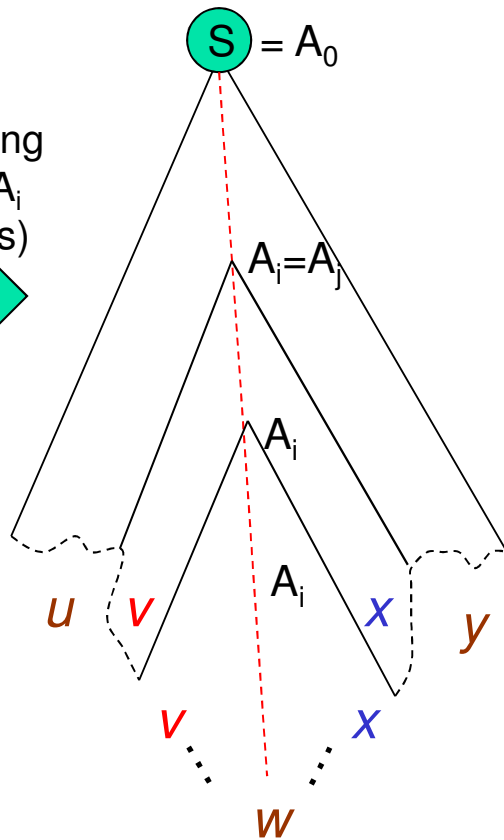
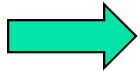
$h \geq m+1$

$m+1$

• Therefore,  $vx \neq \epsilon$

# Extending the parse tree...

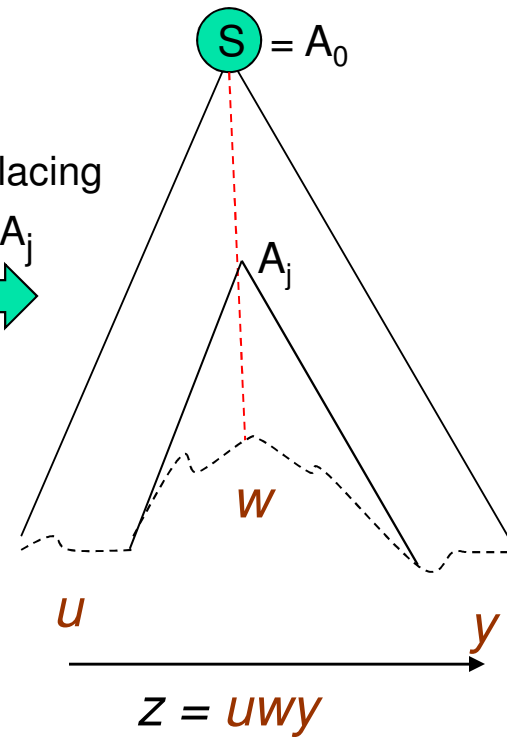
Replacing  $A_j$  with  $A_i$  (k times)



$$z = uv^kwx^ky$$

$h \geq m+1$

Or, replacing  $A_i$  with  $A_j$



$$z = uwy$$

$\implies$  For all  $k \geq 0$ :  $uv^kwx^ky \in L$



## Proof contd..

---

- Also, since  $A_i$ 's subtree no taller than  $m+1$

$\implies$  the string generated under  $A_i$ 's subtree, which is  $vwx$ , cannot be longer than  $2^m (=N)$

But,  $2^m = N$

$\implies |vwx| \leq N$

This completes the proof for the pumping lemma.



# Application of Pumping Lemma for CFLs

---

Example 1:  $L = \{a^m b^m c^m \mid m > 0\}$

Claim: L is not a CFL

Proof:

- Let  $N \leq P/L$  constant
- Pick  $z = a^N b^N c^N$
- Apply pumping lemma to  $z$  and show that there exists at least one other string constructed from  $z$  (obtained by pumping up or down) that is  $\notin L$



# Proof contd...

---

- $z = uvwxy$
- As  $z = a^N b^N c^N$  and  $|vwx| \leq N$  and  $vx \neq \epsilon$ 
  - $\implies v, x$  cannot contain all three symbols (a,b,c)
  - $\implies$  we can pump up or pump down to build another string which is  $\notin L$



## Example #2 for P/L application

---

- $L = \{ ww \mid w \text{ is in } \{0,1\}^* \}$
- Show that  $L$  is not a CFL
  - Try string  $z = 0^N 0^N$ 
    - what happens?
  - Try string  $z = 0^N 1^N 0^N 1^N$ 
    - what happens?



## Example 3

---

- $L = \{ 0^{k^2} \mid k \text{ is any integer} \}$
- Prove  $L$  is not a CFL using Pumping Lemma



## Example 4

---

- $L = \{a^i b^j c^k \mid i < j < k\}$
- Prove that  $L$  is not a CFL



# CFL Closure Properties

---



# Closure Property Results

---

- CFLs are closed under:
  - Union
  - Concatenation
  - Kleene closure operator
  - Substitution
  - Homomorphism, inverse homomorphism

---

- CFLs are *not* closed under:

- Intersection
- Difference
- Complementation

Note: Reg languages  
are closed  
under  
these  
operators



# Strategy for Closure Property Proofs

---

- First prove “closure under **substitution**”
- Using the above result, prove the other closure properties

Note:  $s(L)$  can use a different alphabet

# The *Substitution* operation

For each  $a \in \Sigma$ , then let  $s(a)$  be a language  
If  $w = a_1 a_2 \dots a_n \in L$ , then:

- $s(w) = \{ x_1 x_2 \dots \} \in s(L)$ , s.t.,  $x_i \in s(a_i)$

## Example:

- Let  $\Sigma = \{0,1\}$
- Let:  $s(0) = \{a^n b^n \mid n \geq 1\}$ ,  $s(1) = \{aa, bb\}$
- If  $w = 01$ ,  $s(w) = s(0).s(1)$ 
  - E.g.,  $s(w)$  contains  $a^1 b^1 aa$ ,  $a^1 b^1 bb$ ,  
 $a^2 b^2 aa$ ,  $a^2 b^2 bb$ ,  
... and so on.



# CFLs are closed under Substitution

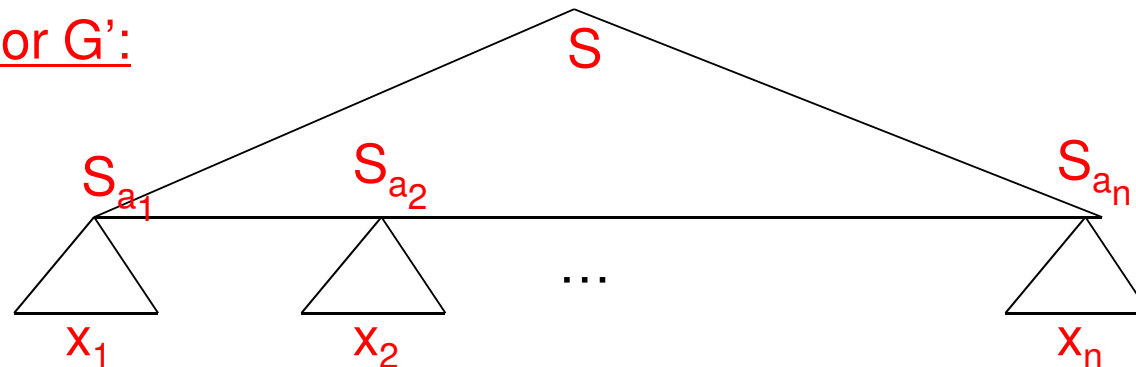
---

- IF  $L$  is a CFL and a substitution defined on  $L$ ,  $s(L)$ , is s.t.,  $s(a)$  is a CFL for every symbol  $a$ , THEN:
  - $s(L)$  is also a CFL

# CFLs are closed under *Substitution*

- $G=(V,T,P,S)$  : CFG for  $L$
- Because each  $s(a)$  is a CFL, there is a CFG for each  $s(a)$ 
  - Let  $G_a = (V_a, T_a, P_a, S_a)$
- Construct  $G'=(V',T',P',S)$  for  $s(L)$
- $P'$  consists of:
  - The productions of  $P$ , but with every occurrence of terminal “ $a$ ” in their bodies replaced by  $S_a$ .
  - All productions in any  $P_a$ , for any  $a \in \Sigma$

Parse tree for  $G'$ :





# CFLs are closed under *union*

---

Let  $L_1$  and  $L_2$  be CFLs

To show:  $L_1 \cup L_2$  is also a CFL

Let us show by using the result of *Substitution*

- Make a new language:

- $L_{\text{new}} = \{a, b\}$  s.t.,  $s(a) = L_1$  and  $s(b) = L_2$

- $\implies s(L_{\text{new}}) = L_1 \cup L_2$



- 
- A more direct, alternative proof

- Let  $S_1$  and  $S_2$  be the starting variables of the grammars for  $L_1$  and  $L_2$

- Then,  $S_{\text{new}} \Rightarrow S_1 \mid S_2$



# CFLs are closed under *concatenation*

---

- Let  $L_1$  and  $L_2$  be CFLs

Let us show by using the result of *Substitution*

- Make  $L_{\text{new}} = \{ab\}$  s.t.,  
 $s(a) = L_1$  and  $s(b) = L_2$   
 $\implies L_1 L_2 = s(L_{\text{new}})$
- 

- A proof without using substitution?



# CFLs are closed under *Kleene Closure*

---

- Let  $L$  be a CFL
- Let  $L_{\text{new}} = \{a\}^*$  and  $s(a) = L_1$ 
  - Then,  $L^* = s(L_{\text{new}})$

# CFLs are closed under *Reversal*

- Let  $L$  be a CFL, with grammar  $G=(V,T,P,S)$
- For  $L^R$ , construct  $G^R=(V,T,P^R,S)$  s.t.,
  - If  $A \Rightarrow \alpha$  is in  $P$ , then:
    - $A \Rightarrow \alpha^R$  is in  $P^R$
    - (that is, reverse every production)



# CFLs are *not* closed under Intersection

---

- Existential proof:
  - $L_1 = \{0^n 1^n 2^i \mid n \geq 1, i \geq 1\}$
  - $L_2 = \{0^i 1^n 2^n \mid n \geq 1, i \geq 1\}$
- Both  $L_1$  and  $L_2$  are CFLs
  - Grammars?
- But  $L_1 \cap L_2$  is *not* a CFL
  - Why?
- We have an example, where intersection is not closed.
- Therefore, CFLs are not closed under intersection



# CFLs are not closed under complementation

---

- Follows from the fact that CFLs are not closed under intersection

- $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$



# CFLs are not closed under difference

---

- Follows from the fact that CFLs are not closed under complementation
- Because, if CFLs are closed under difference, then:
  - $\bar{L} = \Sigma^* - L$
  - So  $\bar{L}$  has to be a CFL too
  - Contradiction



# Decision Properties

---

- Emptiness test
  - Generating test
  - Reachability test
- Membership test
  - PDA acceptance



# “Undecidable” problems for CFL

---

- Is a given CFG  $G$  ambiguous?
- Is a given CFL inherently ambiguous?
- Is the intersection of two CFLs empty?
- Are two CFLs the same?
- Is a given  $L(G)$  equal to  $\Sigma^*$ ?



# Summary

---

- Normal Forms
  - Chomsky Normal Form
  - Griebach Normal Form
  - Useful in proving P/L
- Pumping Lemma for CFLs
  - Main difference:  $z=uv^iwx^iy$
- Closure properties
  - Closed under: union, concatenation, reversal, Kleen closure, homomorphism, substitution
  - Not closed under: intersection, complementation, difference