



BLM2502

Theory of

Computation

Spring 2016

BLM2502 Theory of Computation

» Course Outline

- | » Week | Content |
|-------------|---|
| » 1 | Introduction to Course |
| » 2 | Computability Theory, Complexity Theory, Automata Theory, Set Theory, Relations, Proofs, Pigeonhole Principle |
| » 3 | Regular Expressions |
| » 4 | Finite Automata |
| » 5 | Deterministic and Nondeterministic Finite Automata |
| » 6 | Epsilon Transition, Equivalence of Automata |
| » 7 | Pumping Theorem |
| » 8 | April 10 - 14 week is the first midterm week |
| » 9 | Context Free Grammars |
| » 10 | Parse Tree, Ambiguity, |
| » 11 | Pumping Theorem |
| » 12 | Turing Machines, Recognition and Computation, Church-Turing Hypothesis |
| » 13 | Turing Machines, Recognition and Computation, Church-Turing Hypothesis |
| » 14 | May 22 – 27 week is the second midterm week |
| » 15 | Review |
| » 16 | Final Exam date will be announced |



The Pumping Lemma for CFL's

Pumping Lemma

- » Recall the pumping lemma for regular languages.
- » It told us that if there was a string long enough to cause a cycle in the DFA for the language, then we could "pump" the cycle and discover an infinite sequence of strings that had to be in the language.
- » For CFL's the situation is a little more complicated.
- » We can always find two pieces of any sufficiently long string to "pump" in tandem.
 - > That is: if we repeat each of the two pieces the same number of times, we get another string in the language.



Statement of the CFL Pumping Lemma

- » For every context-free language L There is an integer n , such that For every string z in L of length $> n$ There exists $z = uvwxy$ such that:
1. $|vwx| < n$.
 2. $|vx| > 0$.
 3. For all $i > 0$, uv^iwx^iy is in L .



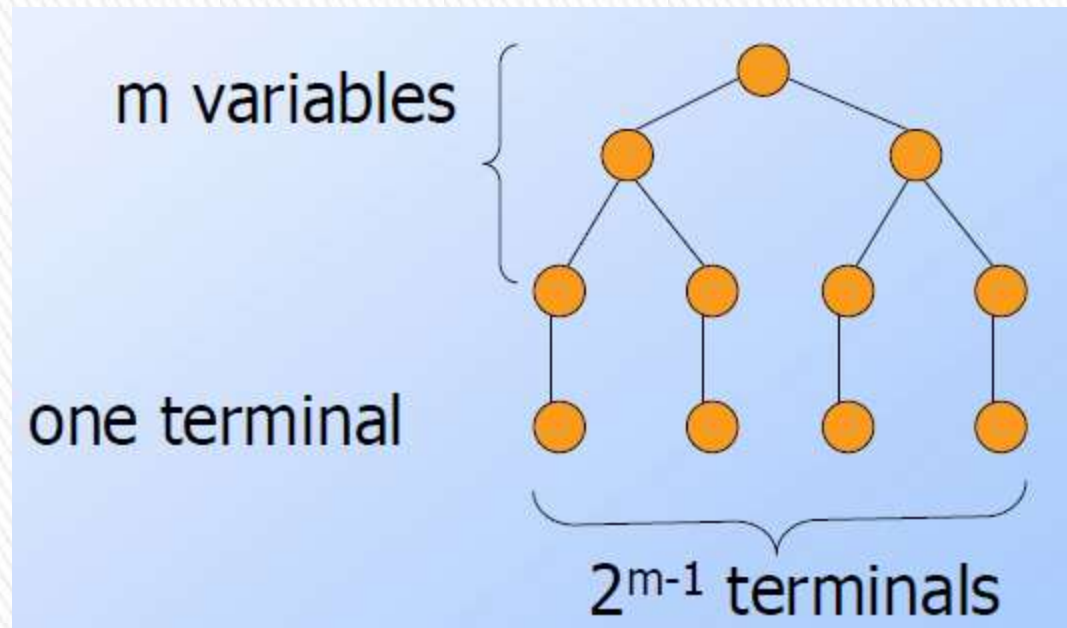
Proof of the Pumping Lemma

- » Start with a CNF grammar for $L - \{\epsilon\}$.
- » Let the grammar have m variables.
 - > Pick $n = 2^m$.
 - > Let $|z| > n$.
- » We claim ("Lemma 1") that a parse tree with yield z must have a path of length $m+2$ or more.



Proof of Lemma 1

- » If all paths in the parse tree of a CNF grammar are of length $< m+1$, then the longest yield has length 2^{m-1} , as in figure:

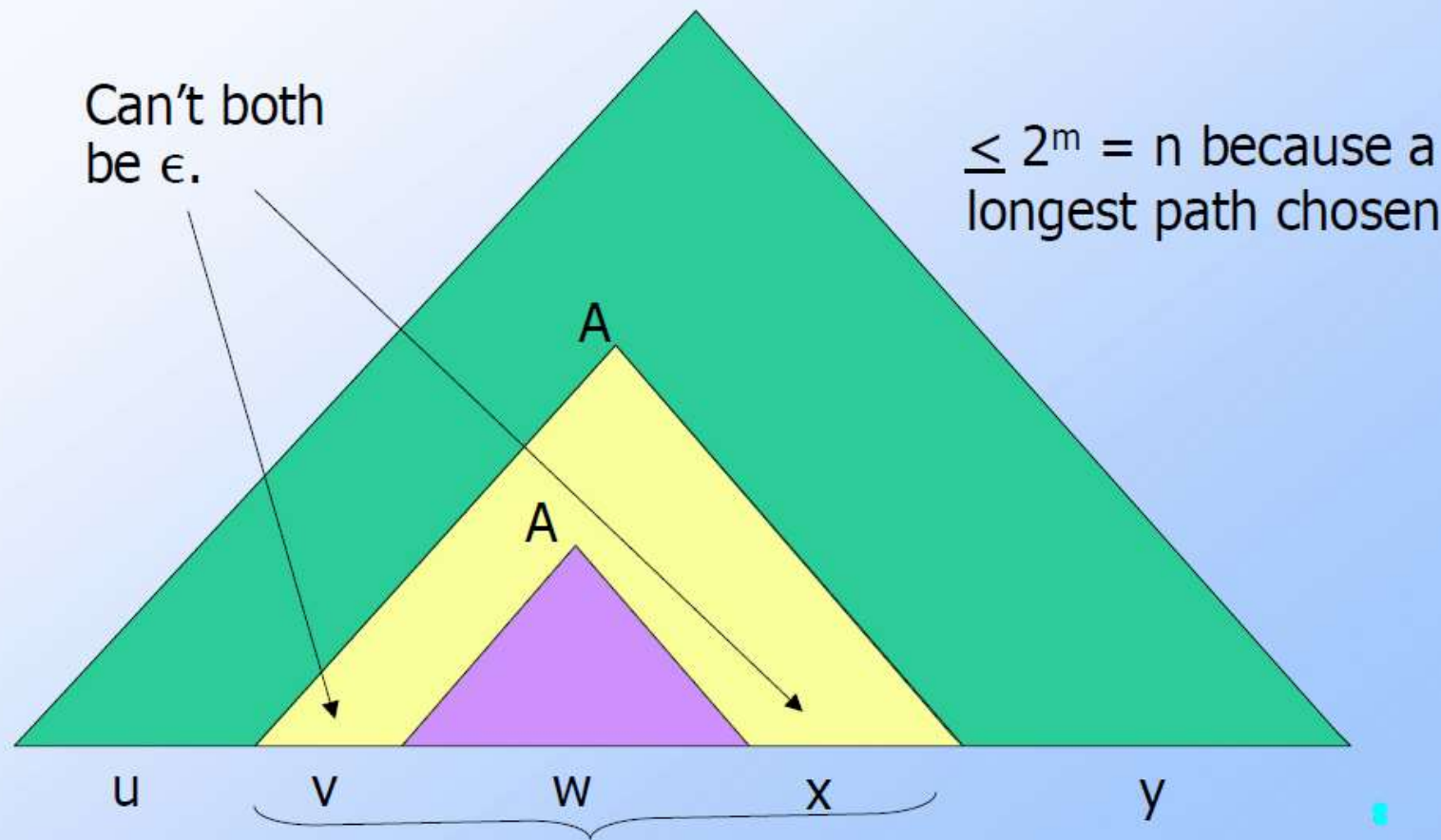


Proof of the Pumping Lemma

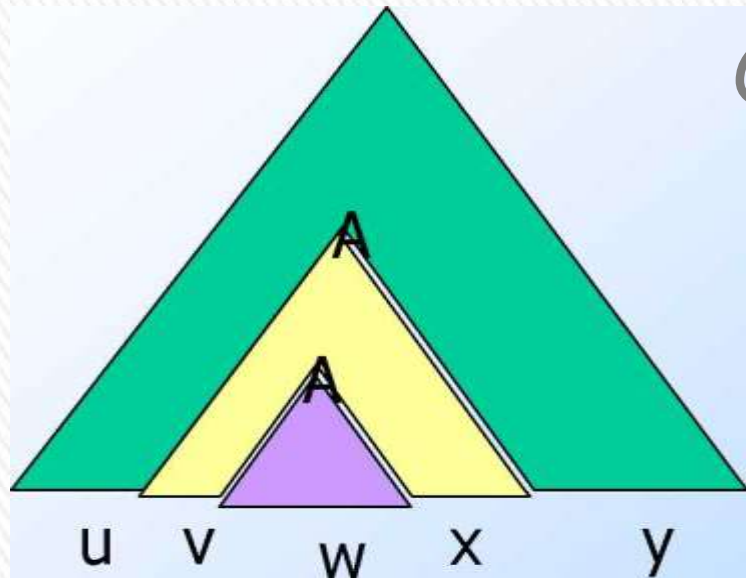
- » Now we know that the parse tree for z has a path with at least $m+1$ variables.
- » Consider some longest path.
- » There are only m different variables, so among the lowest $m+1$ we can find two nodes with the same label, say A .
- » The parse tree thus looks like:



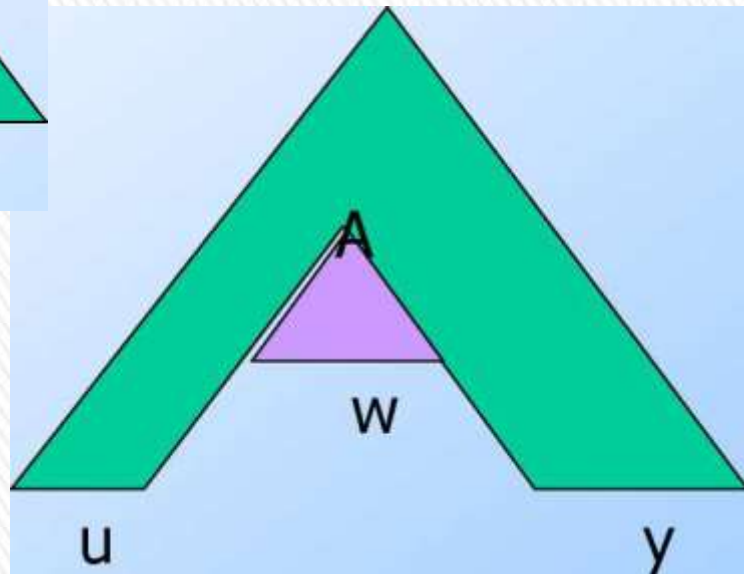
Parse Tree in the Pumping-Lemma



Pumping



Once

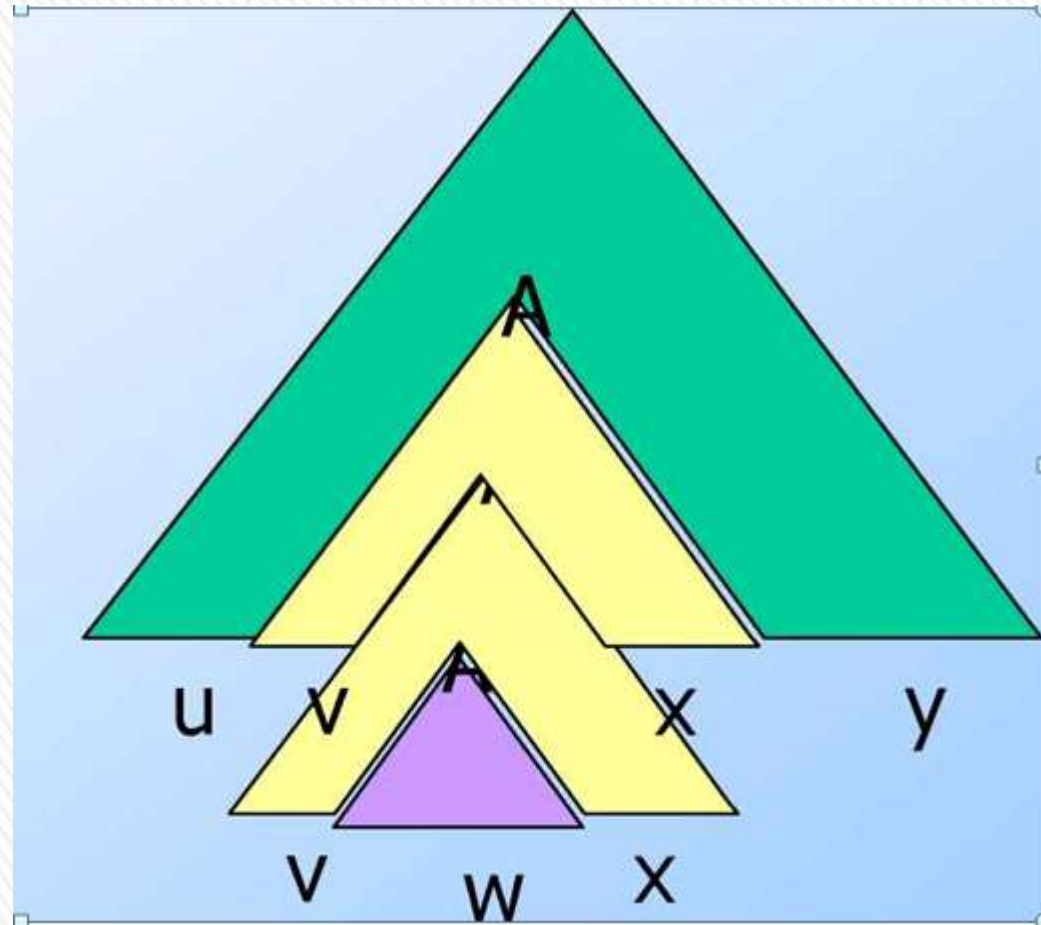


Zero times



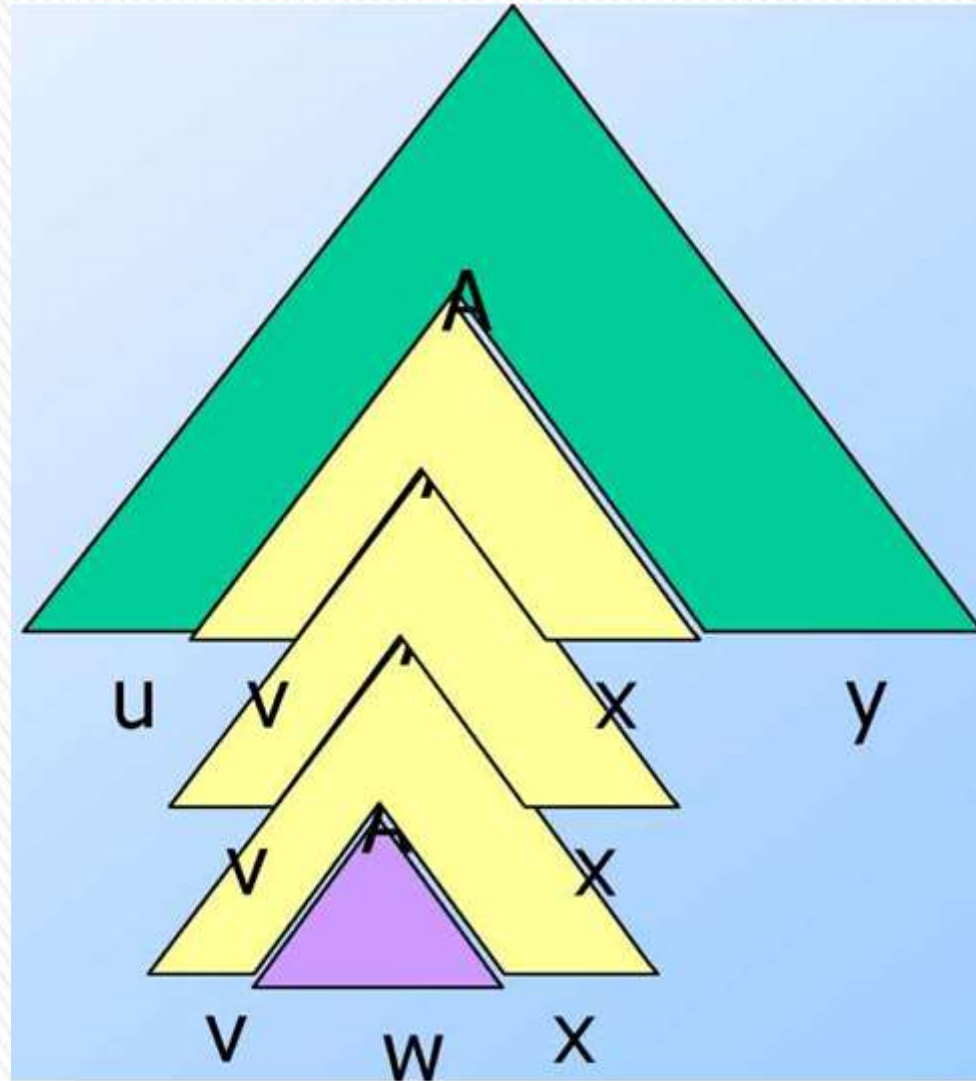
Pumping

Twice



Pumping

Thrice, ...



Using the Pumping Lemma

- » Non-CFL's typically involve trying to match two pairs of counts or match two strings.
- » Example: pumping lemma can be used to show that $L = \{ww \mid w \in (0+1)^*\}$ is not a CFL.
- » $\{0^i 1 0^i \mid i > 1\}$ is a CFL.
 - > We can match one pair of counts.



Using the Pumping Lemma

- » $L = \{0^i10^i10^i \mid i > 1\}$ is not a CFL
 - > We can't match two pairs, or three counts as a group.
 - > Proof using the pumping lemma.
 - > Suppose L were a CFL.
 - > Let n be L 's pumping-lemma constant.
 - > Consider $z = 0^n10^n10^n$.
 - > We can write $z = uvwxy$, where $|vwx| < n$, and $|vx| > 1$.
 - > Case 1: vx has no 0's.
 - > Then at least one of them is a 1, and uwv has at most one 1, which no string in L does.



Using the Pumping Lemma

- » Still considering $z = 0^n 10^n 10^n$.
- » Case 2: vx has at least one 0.
- » vwx is too short ($\text{length} < n$) to extend to all three blocks of 0's in $0^n 10^n 10^n$.
- » Thus, uwy has at least one block of n 0's, and at least one block with fewer than n 0's.
- » Thus, uwy is not in L .





Simplifications of Context-Free Grammars

A Substitution Rule

$$S \rightarrow aB$$

$$A \rightarrow aaA$$

$$A \rightarrow abBc$$

$$B \rightarrow aA$$

$$B \rightarrow b$$

Substitute

$$B \rightarrow b$$

Equivalent
grammar

$$S \rightarrow aB \mid ab$$

$$A \rightarrow aaA$$

$$A \rightarrow abBc \mid abbc$$

$$B \rightarrow aA$$



$$S \rightarrow aB \mid ab$$

$$A \rightarrow aaA$$

$$A \rightarrow abBc \mid abbc$$

$$B \rightarrow aA$$

Substitute

$$B \rightarrow aA$$

$$S \rightarrow \cancel{aB} \mid ab \mid aaA$$

$$A \rightarrow aaA$$

$$A \rightarrow \cancel{abBc} \mid abbc \mid abaAc$$

Equivalent
grammar ➤

In general: $A \rightarrow xBz$

$$B \rightarrow y_1$$

Substitute

$$B \rightarrow y_1$$

$$A \rightarrow xBz \mid xy_1z$$

equivalent
grammar >

Nullable Variables

ε – production :

$$X \rightarrow \varepsilon$$

Nullable Variable:

$$Y \Rightarrow \dots \Rightarrow \varepsilon$$

Example:

$$S \rightarrow aMb$$

$$M \rightarrow aMb$$

$$M \rightarrow \varepsilon$$

Nullable variable

ε – production



Removing ε – productions

$$S \rightarrow aMb$$

$$M \rightarrow aMb$$

~~$$M \rightarrow \varepsilon$$~~

Substitute

$$M \rightarrow \varepsilon$$

$$S \rightarrow aMb \mid ab$$

$$M \rightarrow aMb \mid ab$$

After we remove all the ε – productions
all the nullable variables disappear
(except for the start variable)



Unit-Productions

Unit Production: $X \rightarrow Y$

(a single variable in both sides)

Example: $S \rightarrow aA$

$A \rightarrow a$

$A \rightarrow B$

$B \rightarrow A$

$B \rightarrow bb$

Unit Productions



Removal of unit productions:

$$S \rightarrow aA$$

$$A \rightarrow a$$

~~$$A \rightarrow B$$~~

$$B \rightarrow A$$

$$B \rightarrow bb$$

Substitute

$$A \rightarrow B$$

$$S \rightarrow aA \mid aB$$

$$A \rightarrow a$$

$$B \rightarrow A \mid B$$

$$B \rightarrow bb$$



Unit productions of form $X \rightarrow X$
can be removed immediately

$$S \rightarrow aA \mid aB$$

$$A \rightarrow a$$

$$B \rightarrow A \mid \cancel{B}$$

$$B \rightarrow bb$$

Remove

$$B \rightarrow B$$

$$S \rightarrow aA \mid aB$$

$$A \rightarrow a$$

$$B \rightarrow A$$

$$B \rightarrow bb$$



$$S \rightarrow aA \mid aB$$

$$A \rightarrow a$$

~~$$B \rightarrow A$$~~

$$B \rightarrow bb$$

Substitute

$$B \rightarrow A$$

$$S \rightarrow aA \mid aB \mid aA$$

$$A \rightarrow a$$

$$B \rightarrow bb$$



Remove repeated productions

$$S \rightarrow \textcircled{aA} \mid aB \mid \cancel{aA}$$

$$A \rightarrow a$$

$$B \rightarrow bb$$



Final grammar

$$S \rightarrow aA \mid aB$$

$$A \rightarrow a$$

$$B \rightarrow bb$$



Useless Productions

$$S \rightarrow aSb$$

$$S \rightarrow \lambda$$

$$S \rightarrow A$$

$$A \rightarrow aA \text{ Useless Production}$$

Some derivations never terminate...

$$S \Rightarrow A \Rightarrow aA \Rightarrow aaA \Rightarrow \dots \Rightarrow aa \dots aA \Rightarrow \dots$$

Another grammar:

$$S \rightarrow A$$

$$A \rightarrow aA$$

$$A \rightarrow \lambda$$

$$B \rightarrow bA$$

Useless Production

Not reachable from S



In general:

If there is a derivation

$$S \Rightarrow \dots \Rightarrow xAy \Rightarrow \dots \Rightarrow w \in L(G)$$



consists of
terminals

Then variable A is useful

Otherwise, variable A is useless



A production $A \rightarrow x$ is useless
if any of its variables is useless

$$S \rightarrow aSb$$

$$S \rightarrow \varepsilon$$

Productions

Variables

$$S \rightarrow A$$

useless

useless

$$A \rightarrow aA$$

useless

useless

$$B \rightarrow C$$

useless

useless

$$C \rightarrow D$$

useless



Removing Useless Variables and Productions

Example Grammar:

$$S \rightarrow aS \mid A \mid C$$

$$A \rightarrow a$$

$$B \rightarrow aa$$

$$C \rightarrow aCb$$



First: find all variables that can produce strings with only terminals or ε (possible useful variables)

$$S \rightarrow aS \mid \textcircled{A} \mid C$$

$$\textcircled{A \rightarrow a}$$

$$\textcircled{B \rightarrow aa}$$

$$C \rightarrow aCb$$

Round 1: $\{A, B\}$

(the right hand side of production that has only terminals)

Round 2: $\{A, B, S\}$

(the right hand side of a production has terminals and variables of previous round)

This process can be generalized >

Then, remove productions that use variables other than $\{A, B, S\}$

$$S \rightarrow aS \mid A \mid \cancel{C}$$

$$A \rightarrow a$$

$$B \rightarrow aa$$

$$\cancel{C \rightarrow aCb}$$



$$S \rightarrow aS \mid A$$

$$A \rightarrow a$$

$$B \rightarrow aa$$



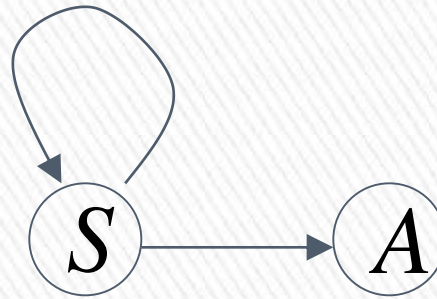
Second: Find all variables
reachable from S

Use a Dependency Graph
where nodes are variables

$$S \rightarrow aS \mid A$$

$$A \rightarrow a$$

$$B \rightarrow aa$$



unreachable



Keep only the variables
reachable from S

$$S \rightarrow aS \mid A$$

$$A \rightarrow a$$

~~$$B \rightarrow aa$$~~



Final Grammar

$$S \rightarrow aS \mid A$$

$$A \rightarrow a$$

Contains only
useful variables

Removing All

- » **Step 1:** Remove Nullable Variables
- » **Step 2:** Remove Unit-Productions
- » **Step 3:** Remove Useless Variables

This sequence guarantees that unwanted variables and productions are removed

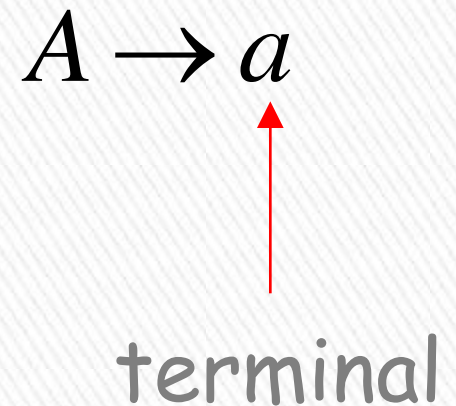
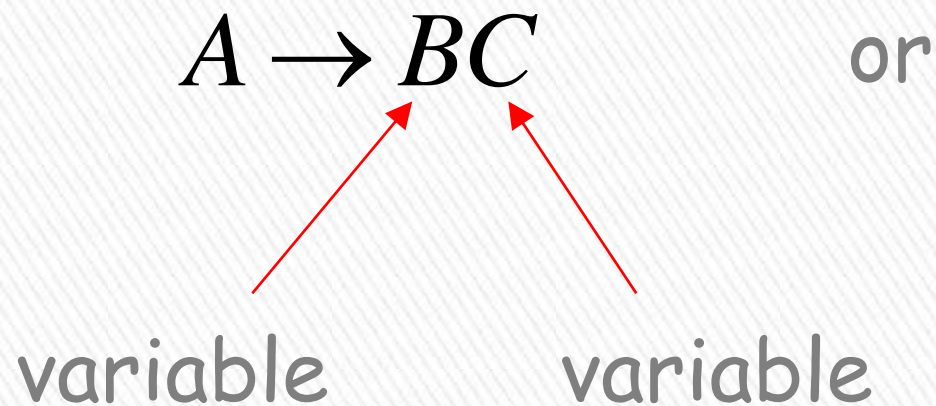




Normal Forms for Context-free Grammars

Chomsky Normal Form

Each productions has form:



Examples:

$$S \rightarrow AS$$

$$S \rightarrow a$$

$$A \rightarrow SA$$

$$A \rightarrow b$$

Chomsky
Normal Form

$$S \rightarrow AS$$

$$S \rightarrow AAS$$

$$A \rightarrow SA$$

$$A \rightarrow aa$$

Not Chomsky
Normal Form



Conversion to Chomsky Normal Form

» Example: $S \rightarrow ABa$

$A \rightarrow aab$

$B \rightarrow Ac$

Not in Chomsky
Normal Form

We will convert it to Chomsky Normal Form



Introduce new variables for the terminals:

$$T_a, T_b, T_c$$

$$S \rightarrow ABa$$

$$A \rightarrow aab$$

$$B \rightarrow Ac$$



$$S \rightarrow ABT_a$$

$$A \rightarrow T_aT_aT_b$$

$$B \rightarrow AT_c$$

$$T_a \rightarrow a$$

$$T_b \rightarrow b$$

$$T_c \rightarrow c$$



Introduce new intermediate variable V_1
to break first production:

$$S \rightarrow ABT_a$$

$$A \rightarrow T_a T_a T_b$$

$$B \rightarrow AT_c$$

$$T_a \rightarrow a$$

$$T_b \rightarrow b$$

$$T_c \rightarrow c$$



$$S \rightarrow AV_1$$

$$V_1 \rightarrow BT_a$$

$$A \rightarrow T_a T_a T_b$$

$$B \rightarrow AT_c$$

$$T_a \rightarrow a$$

$$T_b \rightarrow b$$

$$T_c \rightarrow c$$



Introduce intermediate variable: V_2

$$S \rightarrow AV_1$$

$$V_1 \rightarrow BT_a$$

$$A \rightarrow T_a T_a T_b$$

$$B \rightarrow AT_c$$

$$T_a \rightarrow a$$

$$T_b \rightarrow b$$

$$T_c \rightarrow c$$



$$S \rightarrow AV_1$$

$$V_1 \rightarrow BT_a$$

$$A \rightarrow T_a V_2$$

$$V_2 \rightarrow T_a T_b$$

$$B \rightarrow AT_c$$

$$T_a \rightarrow a$$

$$T_b \rightarrow b$$

$$T_c \rightarrow c$$



Final grammar in Chomsky Normal Form:

$$S \rightarrow AV_1$$

$$V_1 \rightarrow BT_a$$

$$A \rightarrow T_aV_2$$

$$V_2 \rightarrow T_aT_b$$

$$B \rightarrow AT_c$$

$$T_a \rightarrow a$$

$$T_b \rightarrow b$$

$$T_c \rightarrow c$$

Initial grammar

$$S \rightarrow ABa$$

$$A \rightarrow aab$$

$$B \rightarrow Ac$$



In general:

From any context-free grammar
(which doesn't produce ε)
not in Chomsky Normal Form

we can obtain:

an equivalent grammar
in Chomsky Normal Form



The Procedure

First remove:

Nullable variables

Unit productions

(Useless variables optional)



Then, for every symbol a :

New variable: T_a

Add production $T_a \rightarrow a$

In productions with length at least 2
replace a with T_a

Productions of form $A \rightarrow a$
do not need to change!



Replace any production $A \rightarrow C_1 C_2 \cdots C_n$

with $A \rightarrow C_1 V_1$

$V_1 \rightarrow C_2 V_2$

...

$V_{n-2} \rightarrow C_{n-1} C_n$

New intermediate variables: V_1, V_2, \dots, V_{n-2} 

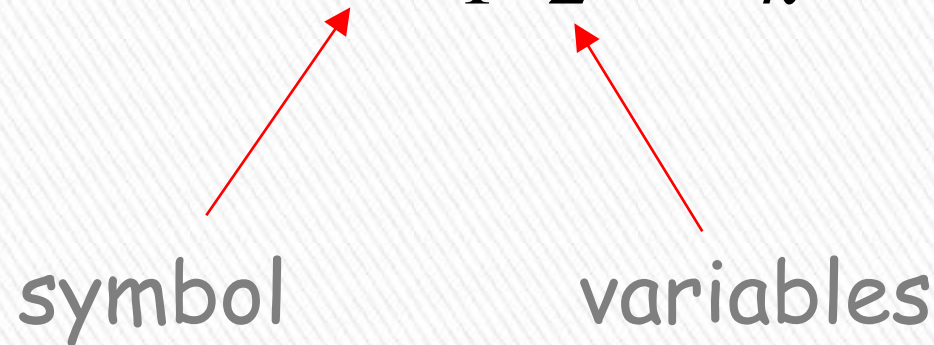
Observations

- Chomsky normal forms are good for parsing and proving theorems
- It is easy to find the Chomsky normal form for any context-free grammar



Greinbach Normal Form

All productions have form:

$$A \rightarrow a V_1 V_2 \cdots V_k \quad k \geq 0$$


symbol

variables



Examples:

$$S \rightarrow cAB$$

$$A \rightarrow aA \mid bB \mid b$$

$$B \rightarrow b$$

Greibach
Normal Form

$$S \rightarrow abSb$$

$$S \rightarrow aa$$

Not Greibach
Normal Form



Conversion to Greinbach Normal Form:

$$S \rightarrow abSb$$

$$S \rightarrow aa$$



$$S \rightarrow aT_bST_b$$

$$S \rightarrow aT_a$$

$$T_a \rightarrow a$$

$$T_b \rightarrow b$$

Greinbach

Normal Form >

Observations

- Greinbach normal forms are very good for parsing strings (better than Chomsky Normal Forms)
- However, it is difficult to find the Greinbach normal of a grammar



BLM2502 Theory of Computation

