The Pumping Lemma for Context Free Grammars

Chomsky Normal Form

- Chomsky Normal Form (CNF) is a simple and useful form of a CFG
- Every rule of a CNF grammar is in the form
  $A \rightarrow BC$
  $A \rightarrow a$
- Where "a" is any terminal and A,B,C are any variables except B and C may not be the start variable
  - There are two and only two variables on the right hand side of the rule
  - Exception: $S \rightarrow \varepsilon$ is permitted where S is the start variable

Theorem

- Any context free language may be generated by a context free grammar in Chomsky Normal Form
- To show how this is possible we must be able to convert any CFG into CNF
  1. Eliminate all ε rules of the form $A \rightarrow \varepsilon$
  2. Eliminate all unit rules of the form $A \rightarrow B$
  3. Convert any remaining rules into the form $A \rightarrow BC$

Proof

- First add a new start symbols S_0 and the rule $S_0 \rightarrow S$ where S was the original start symbol
  - This guarantees the new start symbol is not on the RHS of any rule
- Remove all ε rules.
  - Remove a rule $A \rightarrow \varepsilon$ where A is not the start symbol. For each occurrence of A on the RHS of a rule, add a new rule with that occurrence of A deleted
  - Ex:

    $R \rightarrow uAv$ becomes $R \rightarrow uv$
  - This must be done for each occurrence of A, e.g.:

    $R \rightarrow uAvAw$ becomes $R \rightarrow uvAw \mid uAvw \mid uvw$

  Repeat until all ε rules are removed, not including the start

Proof

- Next remove all unit rules of the form $A \rightarrow B$
  - Whenever a rule $B \rightarrow u$ appears, add the rule $A \rightarrow u$.
  - u may be a string of variables and terminals
  - Repeat until all unit rules are eliminated
- Convert all remaining rules into the form with two variables on the right
  - The rule $A \rightarrow u_1u_2u_3\ldots u_k$ becomes
  - $A \rightarrow u_1A_1 \quad A_1 \rightarrow u_2A_2 \quad \ldots \quad A_{k-2} \rightarrow u_{k-1}u_k$
  - Where the A_i's are new variables. u may be a variable or a terminal (and in fact a terminal must be converted to a variable since CNF does not allow a mixture of variables and terminals on the right hand side)

Example

- Convert the following grammar into CNF

  $S \rightarrow ASA \mid aB$

  $A \rightarrow B \mid S$

  $B \rightarrow b \mid \varepsilon$

First add a new start symbol S_0:

  $S_0 \rightarrow S$

  $S \rightarrow ASA \mid aB$

  $A \rightarrow B \mid S$

  $B \rightarrow b \mid \varepsilon$

Example

- Next remove the epsilon transition from rule B

  $S_0 \rightarrow S$
  $S \rightarrow ASA \mid aB \mid a$
  $A \rightarrow B \mid S \mid \varepsilon$
  $B \rightarrow b$

- We must repeat this for rule A:

  $S_0 \rightarrow S$
  $S \rightarrow ASA \mid aB \mid a \mid AS \mid SA \mid S$
  $A \rightarrow B \mid S$
  $B \rightarrow b$

Example

- Next remove unit rules, starting with $S_0 \rightarrow S$ and $S \rightarrow S$ can also be removed

  $S_0 \rightarrow ASA \mid aB \mid a \mid AS \mid SA$
  $S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$
  $A \rightarrow B \mid S$
  $B \rightarrow b$

- Next remove the rule for $A \rightarrow B$

  $S_0 \rightarrow ASA \mid aB \mid a \mid AS \mid SA$
  $S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$
  $A \rightarrow b \mid S$
  $B \rightarrow b$

- Next remove the rule for $A \rightarrow S$

  $S_0 \rightarrow ASA \mid aB \mid a \mid AS \mid SA$
  $S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$
  $A \rightarrow b \mid ASA \mid aB \mid a \mid AS \mid SA$
  $B \rightarrow b$

Example

- Finally convert the remaining rules to the proper form by adding variables and rules when we have more than three things on the RHS
  - $S_0 \rightarrow$ ASA | aB | a | AS | SA
  - $S \rightarrow$ ASA | aB | a |AS | SA
  - $A \rightarrow$ b| ASA | aB | a |AS | SA
  - $B \rightarrow$ b
- Becomes
  - $S_0 \rightarrow AA_1$ | A_2B | a | AS | SA
  - $A_1 \rightarrow$ SA
  - $A_2 \rightarrow$ a
  - $S \rightarrow AA_1$ | A_2B | a |AS | SA
  - $A \rightarrow$ b| AA_1 | A_2B | a |AS | SA
  - $B \rightarrow$ b

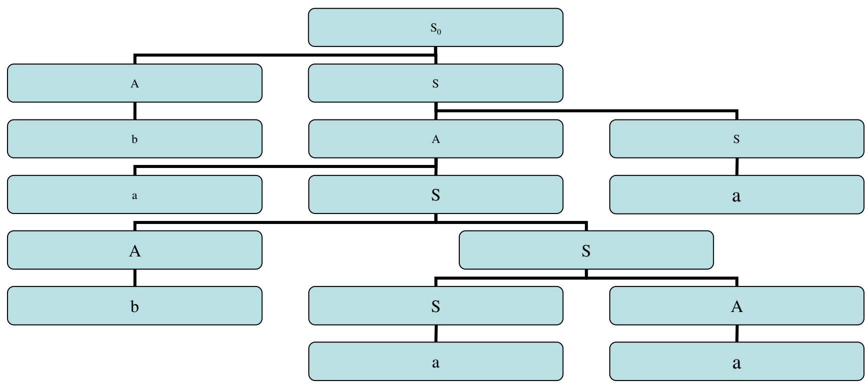
We are done!

CNF and Parse Trees

- Chomsky Normal Form is useful to interpret a grammar as a parse tree
  - CNF forms a binary tree!
  - Consider the string babaaa on the previous grammar

    $S_0 \rightarrow$ AS $\rightarrow$ bS $\rightarrow$ bAS $\rightarrow$ bASS $\rightarrow$ baSS $\rightarrow$ baASS $\rightarrow$ babSS $\rightarrow$ babSAS $\rightarrow$ babaAS $\rightarrow$ babaaS $\rightarrow$ babaaa

Grammar as a Parse Tree



Why is this useful?

- Because we know lots of things about binary trees
- We can now apply these things to context-free grammars since any CFG can be placed into the CNF format
- For example
  – If yield of the tree is a terminal string w
  – If n is the height of the longest path in the tree
  – Then $|w| \leq 2^{n-1}$
  – How is this so? (Next slide)

Yield of a CNF Parse Tree

- Yield of a CNF parse tree is $|w| \leq 2^{n-1}$
- Base Case: n = 1
  - If the longest path is of length 1, we must be using the rule $A \rightarrow t$ so $|w|$ is 1 and $2^{1-1} = 1$
- Induction
  - Longest path has length n, where n>1. The root uses a production that must be of the form $A \rightarrow BC$ since we can't have a terminal from the root
  - By induction, the subtrees from B and C have yields of length at most 2^{n-2} since we used one of the edges from the root to these subtrees
  - The yield of the entire tree is the concatenation of these two yields, which is $2^{n-2} + 2^{n-2}$ which equals $2*2^{n-2} = 2^{n-2+1}=2^{n-1}$

The Pumping Lemma for CFL's

- The result from the previous slide ($|w| \leq 2^{n-1}$) lets us define the pumping lemma for CFL's
- The pumping lemma gives us a technique to show that certain languages are not context free
  - Just like we used the pumping lemma to show certain languages are not regular
  - But the pumping lemma for CFL's is a bit more complicated than the pumping lemma for regular languages
- Informally
  - The pumping lemma for CFL's states that for sufficiently long strings in a CFL, we can find two, short, nearby substrings that we can "pump" in tandem and the resulting string must also be in the language.

The Pumping Lemma for CFL's

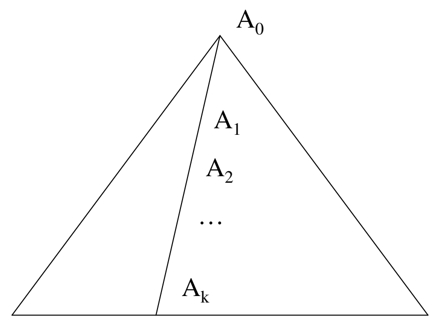
- Let L be a CFL. Then there exists a constant *p* such that if z is any string in L where $|z| \geq p$, then we can write z = uvwxy subject to the following conditions:
  1. $|vwx| \leq p$. This says the middle portion is not larger than p.
  2. $vx \neq \varepsilon$. We'll pump v and x. One may be empty, but both may not be empty.
  3. For all $i \geq 0$, uv^iwx^iy is also in L. That is, we pump both v and x.

Why does the Pumping Lemma Hold?

- Given any context free grammar G, we can convert it to CNF. The parse tree creates a binary tree.
- Let G have *m* variables. Choose this as the value for the longest path in the tree.
  - The constant p can then be selected where $p = 2^m$.
  - Suppose a string z = uvwxy where $|z| \geq p$ is in L(G)
    - We showed previously that a string in L of length m or less must have a yield of 2^{m-1} or less.
    - Since $p = 2^m$, then 2^{m-1} is equal to p/2.
    - This means that z is too long to be yielded from a parse tree of length m.
  - What about a parse tree of length m+1?
    - Choose longest path to be m+1, yield must then be 2^m or less
    - Given $p=2^m$ and $|z| \leq p$ this works out
    - Any parse tree that yields z must have a path of length at least m+1. This is illustrated in the following figure:

Parse Tree

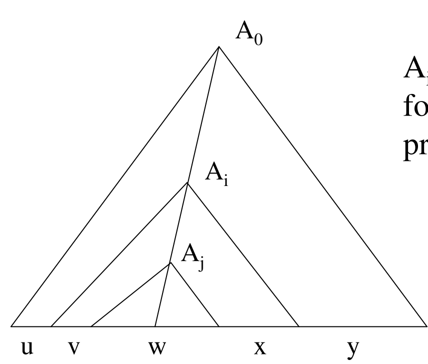
- z=uvwxy where |z| ≥ p



- Variables $A_0, A_1, \ldots A_k$
- If k≥m then at least two of these variables must be the same, since only m unique variables

Parse Tree

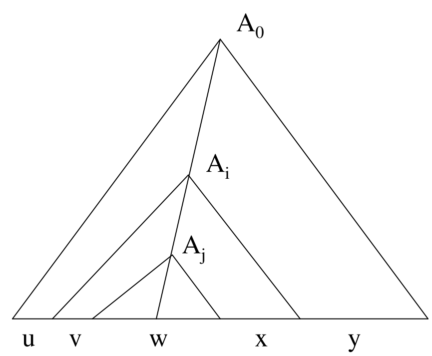
- Suppose the variables are the same at $A_i = A_j$ where $k-m \leq i < j \leq k$



$A_i = A_j$ although we may follow different production rules for each

Pumping Lemma

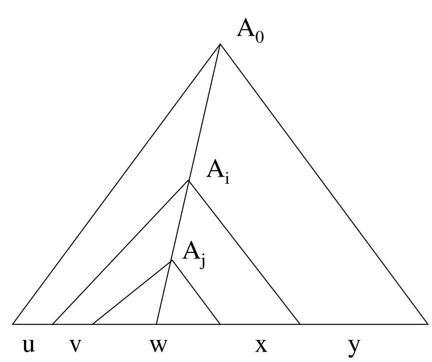
- Condition 2: $vx \neq \varepsilon$
- Follows since we must use a production from A_i to A_j and can't be a terminal or there would be no A_j.
- Therefore we must have two variables; one of these must lead to A_j and the other must lead to v or x or both.
- This means v and x cannot both be empty but one might be empty.



Pumping Lemma

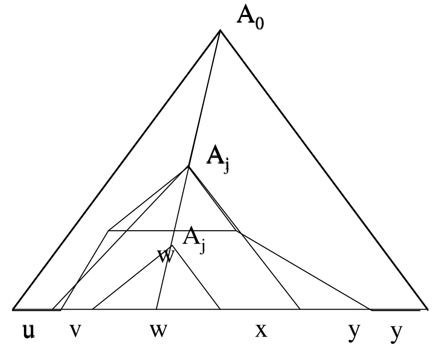
- Condition 1 stated that $|vwx| \leq p$
- This says the yield of the subtree rooted at A_i is $\leq p$
- We picked the tree so the longest path was m+1, so it easily follows that
  $|vwx| \leq p \leq 2^{m+1-1}$

(A_i could be A_0 so vwx is the entire tree)



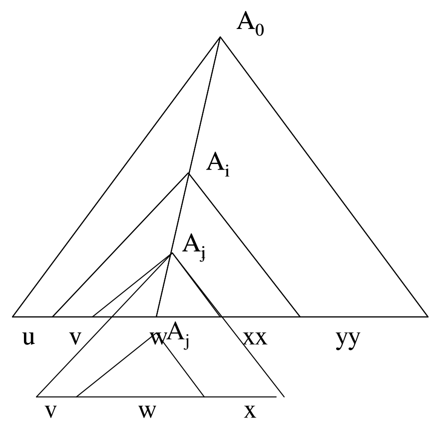
Pumping Lemma

- Condition 3 stated that for all $i \geq 0$, uv^iwx^iy is also in L
- We can show this by noting that the symbol $A_i = A_j$
- This means we can substitute different production rules for each other
- Substituting A_j for A_i the resulting string must be in L



Pumping Lemma

- Substituting A_i for A_j
- Result:
- uv^1wx^1y, uv^2wx^2y, etc.



Pumping Lemma

- We have now shown all conditions of the pumping lemma for context free languages
- To show a language is not context free we
  - Pick a language L to show that it is not a CFL
  - Then some *p* must exist, indicating the maximum yield and length of the parse tree
  - We pick the string z, and may use *p* as a parameter
  - Break z into uvwxy subject to the pumping lemma constraints
    - $|vwx| \leq p$, $|vx| \neq \varepsilon$
  - We win by picking i and showing that uv^iwx^iy is not in L, therefore L is not context free

Example 1

- Let L be the language $\{ 0^n1^n2^n \mid n \geq 1 \}$. Show that this language is not a CFL.
- Suppose that L is a CFL. Then some integer p exists and we pick $z = 0^p1^p2^p$.
- Since z=uvwxy and $|vwx| \leq p$, we know that the string vwx must consist of either:
  - all zeros
  - all ones
  - all twos
  - a combination of 0's and 1's
  - a combination of 1's and 2's
- The string vwx cannot contain 0's, 1's, and 2's because the string is not large enough to span all three symbols.
- Now "pump down" where i=0. This results in the string uwy and can no longer contain an equal number of 0's, 1's, and 2's because the strings v and x contains at most two of these three symbols. Therefore the result is not in L and therefore L is not a CFL.

Example 2

- Let L be the language $\{ a^i b^j c^k \mid 0 \le i \le j \le k \}$. Show that this language is not a CFL. This language is similar to the previous one, except proving that it is not context free requires the examination of more cases.
- Suppose that L is a CFL.
- Pick $z = a^p b^p c^p$ as we did with the previous language.
- As before, the string vwx cannot contain a's, b's, and c's. We then pump the string depending on the string vwx as follows:
  - There are no a's. Then we try pumping down to obtain the string uv^0wx^0y to get uwy. This contains the same number of a's, but fewer b'c or c's. Therefore it is not in L.
  - There are no b's but there are a's. Then we pump up to obtain the string uv^2wx^2y to give us more a's than b's and this is not in L.
  - There are no b's but there are c's. Then we pump down to obtain the string uwy. This string contains the same number of b's but fewer c's, therefore this is not in C.
  - There are no c's. Then we pump up to obtain the string uv^2wx^2y to give us more b's or more a's than there are c's, so this is not in C.
- Since we can come up with a contradiction for any case, this language is not a CFL language.

Example 3

- Let L be the language $\{ww \mid w \in \{0,1\}^*\}$. Show that this language is not a CFL.
- As before, assume that L is context-free and let p be the pumping length.
- This time choosing the string z is less obvious. One possibility is the string: 0^p10^p1. It is in L and has length greater than p, so it appears to be a good candidate.
- But this string can be pumped as follows so it is not adequate for our purposes:

$$\overbrace{\underbrace{000\ldots000}_{u}\ \ \underbrace{0}_{v}\ \ \underbrace{1}_{w}}^{0^p1}\ \ \overbrace{\underbrace{0}_{x}\ \ \underbrace{000\ldots0001}_{y}}^{0^p1}$$

Example 3

- This time lets try $z=0^p1^p0^p1^p$ instead. We can show that this string cannot be pumped.
- We know that $|vwx| \leq p$.
  - Let's say that the string |vwx| consists of the first p 0's. If so, then if we pump this string to uv^2wx^2y then we'll have introduced more 0's in the first half and this is not in L.
  - We get a similar result if |vwx| consists of all 0's or all 1's in either the first or second half.
  - If the string |vwx| matches some sequence of 0's and 1's in the first half of z, then if we pump this string to uv^2wx^2y then we will have introduced more 1's on the left that move into the second half, so it cannot be of the form ww and be in L. Similarly, if |vwx| occurs in the second half of z, them pumping z to uv^2wx^2y moves a 0 into the last position of the first half, so it cannot be of the form ww either.
  - This only leaves the possibility that |vwx| somehow straddles the midpoint of z. But if this is the case, we can now try pumping the string down. uv^0wx^0y = uwy has the form of $0^p1^i0^j1^p$ where i and j cannot both equal p. This string is not of the form ww and therefore the string cannot be pumped and L is therefore not a CFL.