CS 3313
Foundations of Computing:

Properties of Context Free Languages

http://gw-cs3313-2021.github.io

Properties of Context Free Languages

- What are the properties of CFLs ? Why is this interesting ?
  - What types of languages are CFL ?
    - Can all properties/semantics of a programming language be captured by a CFL ?
    - Can natural languages be described by CFGs ?
      - Can we determine ambiguity and remove ambiguity ?
      - Can we parse natural languages using a CFG for the syntax ?
  - If we combine CFLs using set operations, is the resulting language CFL ?
- How do we prove if a language is not context free ?
  - Pumping lemma for CFLs !!

Why bother with Properties/limits of CFLs

- Exercise in abstraction:
- Scenario: In a program, we have function declaration and then a function call.
  - The actual and formal parameters need to match
  - Ex: int foo(int x, char y)…. and main has: z= foo(a,b)
    - a must be an int, b must be a char
- Question: Can this property be described/specified by a context free grammar ?
- Abstraction: the property can be captured by $\{a^n b^m c^n d^m\}$
  - a^n,b^m are formal parameters – n of type a (int), m of type b (char)

Pumping Lemma: Intuition

- Informally: DFAs don't have external memory, so languages that require "storing" counts, strings, etc. are likely to not be regular
  - Ex: {equal number of a's and b's}, { ww^R},….
- 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 transition graph, then we could "pump" the cycle and discover an infinite sequence of strings that had to be in the language.
  - Apply it using the 2-person game:
    - You pick the string after adversary picks n (i.e., you cannot specify a value for n)

Intuition – (2)

- For CFL's the situation is a little more complicated.
- PDAs have external memory – a stack
  - But stack is limited in its capabilities
    - One "counter"
    - If you store something in the stack then when you check storage (i.e., pop the stack) the reverse pattern is popped.
  - Informal limits:
    - Languages that require multiple counters { $a^n b^n c^n$}
    - Languages that require exact patterns {ww}
  - *If you push a pattern into the stack in the "first part" of the string, then that pattern repeats in "second part"*
- 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.

Properties of Parse Trees

- Lemma 1: Let G in Chomsky Normal Form (CNF), then for any parse tree with yield w (string w generated by grammar) if n is the length of the longest path in the tree then $|w| \leq 2^{n-1}$.
- Proof: What type of tree is a parse tree for a CNF grammar ? – binary tree
- Recall CS1311 !!!
- Or prove by induction on length of the path
  - Basis: n=1 derivation must be $S \rightarrow a$
  - Ind.Step: Since G is in CNF, $S \rightarrow AB$ and $A \Rightarrow^* w_2$ and $B \Rightarrow^* w_2$
    - A derives substring w_1 with path $\leq$ n-1
    - B derives substring w_2 with path $\leq$ n-1
    - From IH: $|w_1| \leq 2^{n-2}$ and $|w_2| \leq 2^{n-2}$
    - $|w| = |w_1| + |w_2| \leq 2^{n-1}$

Properties of parse trees for arbitrarily long strings

- From previous theorems, if L is a CFL then there exists CNF $G=(V,T,P,S)$ such that $L=L(G)$
  - L is generated by a CNF grammar G
  - $|V| = m$ finite set of variables – m variables

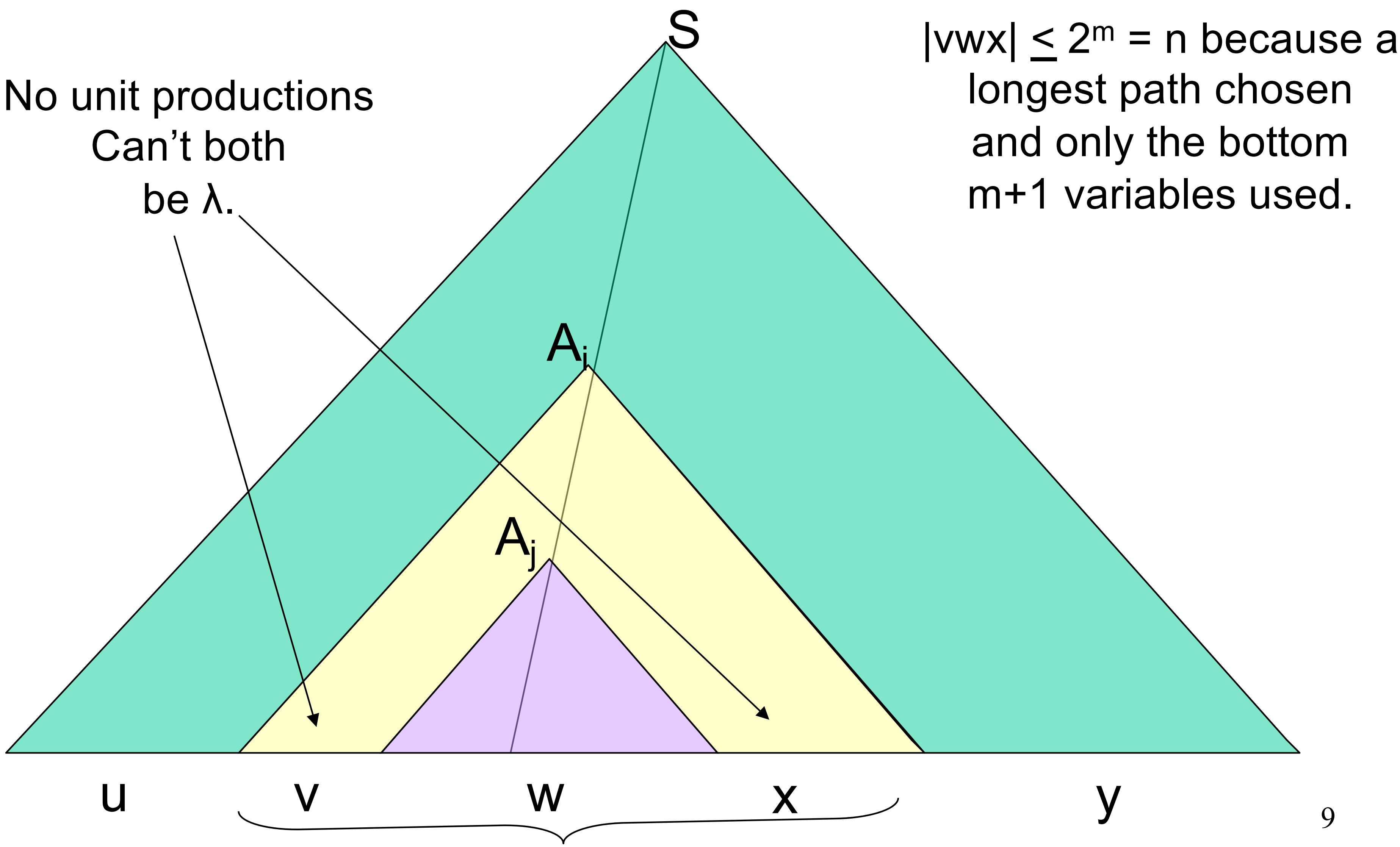
- We are implicitly discussing infinite languages
  - If a language is finite then it is a regular language
    - Implies regular grammar (subset of CFLs)

- Suppose we have $z \in L(G)$ and $|z| \geq n = 2^m$
- What can we say about parse tree for z ?
  - From lemma 1, parse tree for z must have a path of length at least m+1
    - Yield of the tree is $\leq 2^m$

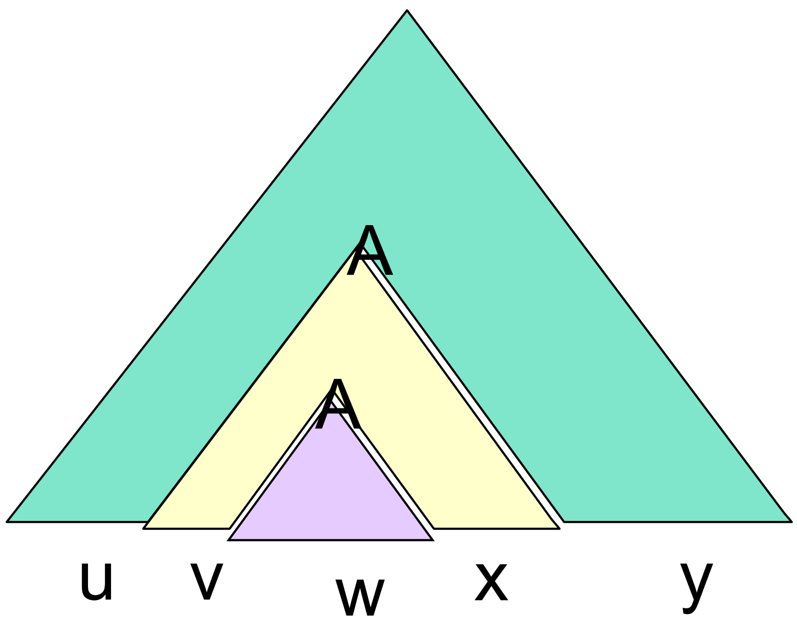
Parse tree properties

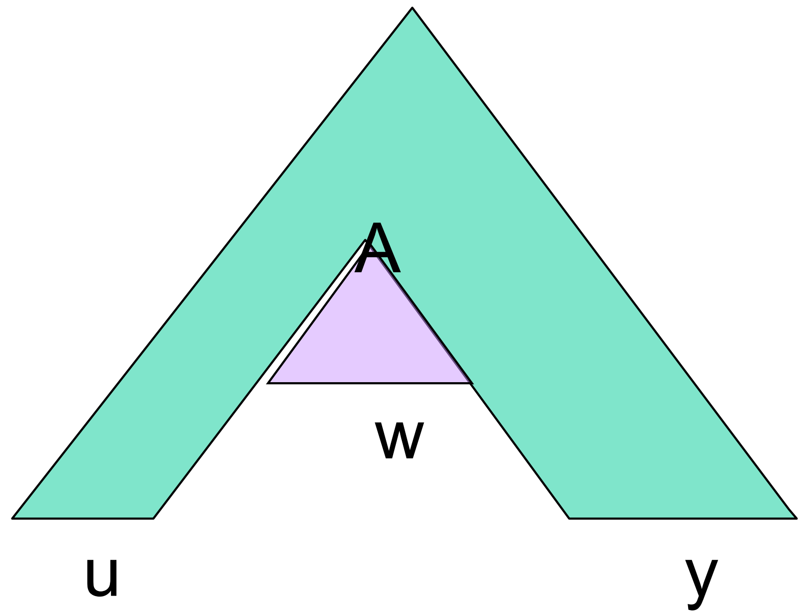
- If path has length $k \geq m+1$, then it has $k+1$ vertices/nodes in the path
  - Last vertex is labelled with a terminal
- Therefore path has k internal nodes labelled with variables of the grammar
  - These are $A_1, A_2 \ldots A_i, \ldots A_j \ldots A_k$
  - A_1 is the start symbol S
- We have m distinct variables => from pigeon hole principle, at least two of the vertices A_i and A_j are the same variable
  - In fact, from the leaf, these two occur within path of length $m+1$
- So what does this tell us about the parse tree for z ?

Parse Tree in the Pumping-Lemma Proof

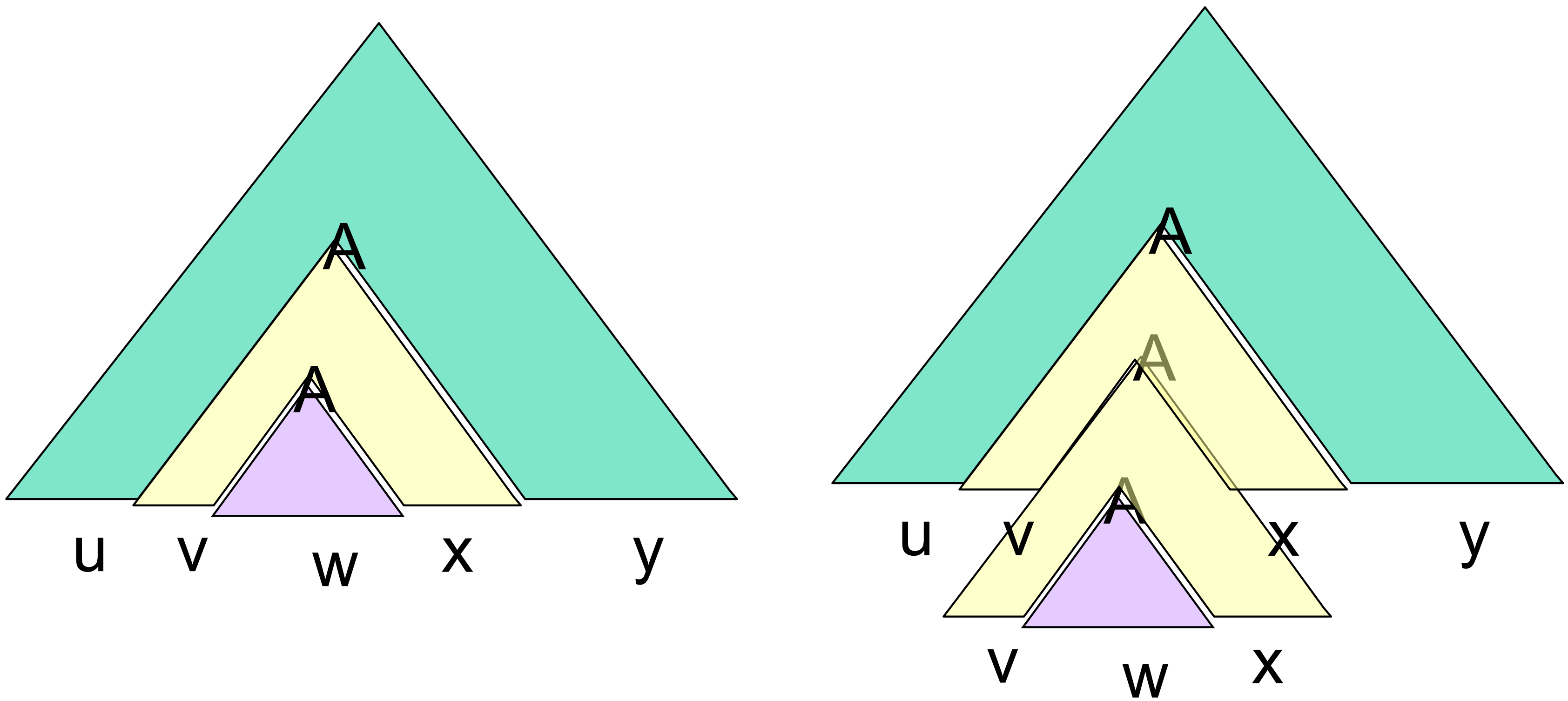


Pump Zero Times



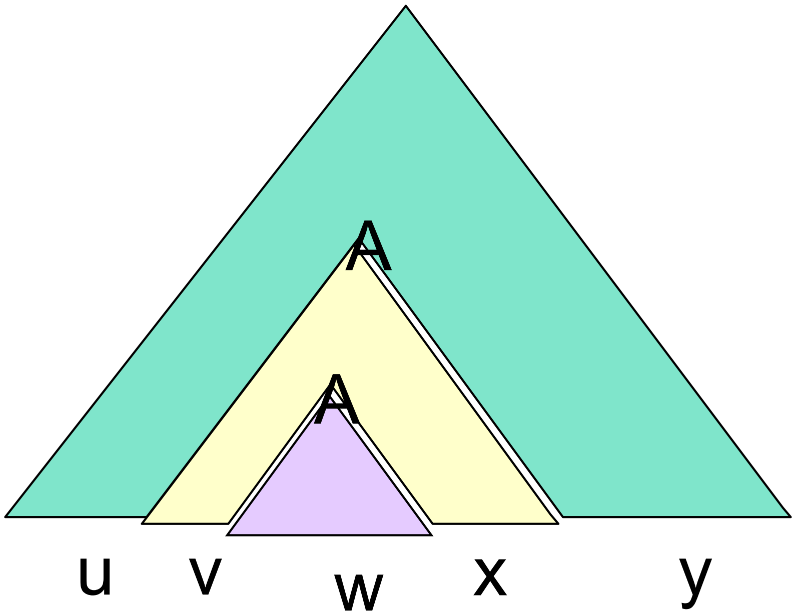


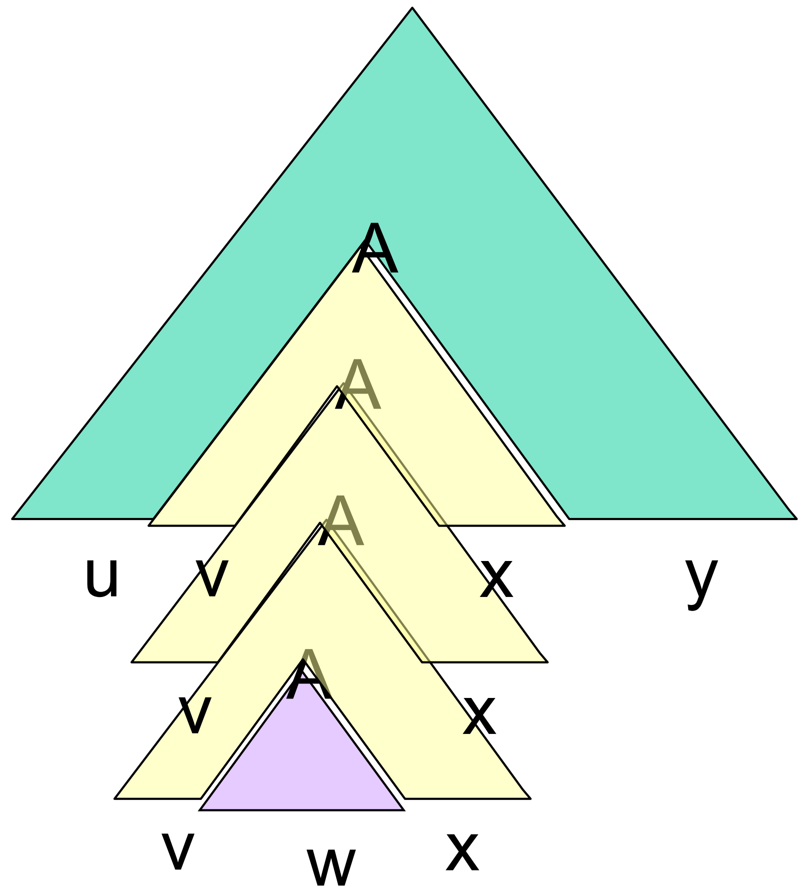
Pump Twice



Pump Thrice

Etc., Etc.





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 $\geq$ n

There exists z = uvwxy such that:

1. $|vwx| \leq n$.
2. $|vx| > 0$.
3. For all $i \geq 0$, uv^iwx^iy is in L.

How do use the pumping lemma: recall 2 person adversarial game

- **For all** context free languages L, **there exists** *n*…**for all z** in L

….**there exists** uvw*xy*….

- Logical statements/assertions that have several alternations of for all and there exists quantifiers can be thought of as a game between two players

- Application of the pumping lemma can be seen as a two player game (of 5 steps)

Pumping Lemma as Adversarial Game

1. Player 1 (we) picks language we want to show is not a CFL
2. Player 2 “adversary” gets to pick n
   - We do not know the value of n, and must plan for all values of n
3. We get to pick z, and may use n as a parameter
   - Can express z using the parameter n
4. Adversary gets to break z into $uvwxy$ subject only to the constraints that $|vwx| \leq n$ *and* $|vx| \geq 1$.
5. We “win” the game, if we can, by picking i and showing uv^iwx^iy *is not in* L
   - We have to show this for all cases of how adversary breaks z into $uvwxy$

Example: $L = \{ a^i b^i c^i \}$

- Informally: CFL (PDA) can count & match two groups of symbols but not three (since we have one counter)
- Apply pumping lemma to prove L is not CFL
- Assume L is CFL
- Let n be the constant of the lemma.
- Pick $z = a^n b^n c^n$
- Big difference from pumping lemma for regular languages
  - For regular languages, the pumping lemma allowed us to focus on the first n symbols/locations in the string
  - In CFL, the lemma only states $|vwx| \leq n$
  - This suggests we have to consider different cases where vwx can occur!
  - ***Prove contradiction in every case!***
    - *No matter how adversary breaks up vwx, we prove a contradiction*

Example: Cases for vwx for $L = \{ a^i b^i c^i \}$

1. vwx is entirely within a^n

2. vwx is entirely within b^n

3. vwx is entirely within c^n

4. vwx has two symbols (a and b, or b and c)

Exercise: $L_2 = \{ a^i b^j c^i d^j \}$ a's = c's and b's = d's

- Intuition: L_2 is likely not CFL. If we push a's and b's on the stack (to remember how many), then we pop b's before a's

1. Assume L_2 is CFL *you pick*
2. Let n be the constrant *adversary picks*
3. Consider $z = a^n b^n c^n d^n \in L_2$. *you pick*
4. $z = uvwxy$, $|vwx| \leq n$, *and* $|vx| \geq 1$ *adversary picks*
5. For every $i \geq 0$, $uv^i wx^i y \in L_2$ *you pick I*

- **Question: (a) Find all cases for vwx and then (b) show contradiction for each case**

"weakness" of the Pumping Lemma

- It allows vwx to be anywhere in the string
  - In contrast to pumping lemma for regular languages
- Looking at the proof, we can see the opportunity to limit the 'areas' to pump…..leads to a stronger pumping lemma:

Ogden's lemma: For every context-free language L, there is an integer n (which may in fact be the same as for the pumping lemma), such that if z is any string in L and we mark any n or more positions of z as "distinguished", then z = uvwxy such that:

1. *vwx* has at most n distinguished positions
2. *vx* has at least one distinguished position
3. For all $i \geq 0$, uv^iwx^iy is in L.

Pumping lemma essentially marks all positions as distinguished!