MA/CSSE 474
Theory of Computation

Bottom-up parsing
Pumping Theorem for CFLs

Recap: Going One Way

Lemma: Each context-free language is accepted by some PDA.

Proof (by construction):

The idea: Let the stack do the work.

Two approaches:

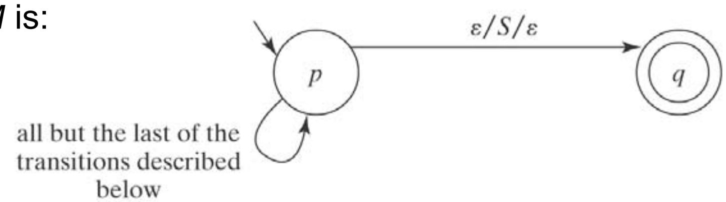
- Top down
- Bottom up

Top-down VS Bottom-up

Approach	Top-down	Bottom-up
Read the input string	left-to-right	left-to-right
Derivation	leftmost	rightmost
Order of derivation discovery	forward	backward

Bottom-Up PDA

The outline of M is:



$M = (\{p, q\}, \Sigma, V, \Delta, p, \{q\})$, where Δ contains:

- The shift transitions: $((p, c, \varepsilon), (p, c))$, for each $c \in \Sigma$.
- The reduce transitions: $((p, \varepsilon, (s_1 s_2 \ldots s_n.)^R), (p, X))$, for each rule $X \to s_1 s_2 \ldots s_n.$ in G. **Undoes an application of this rule.**
- The finish-up transition: $((p, \varepsilon, S), (q, \varepsilon))$.

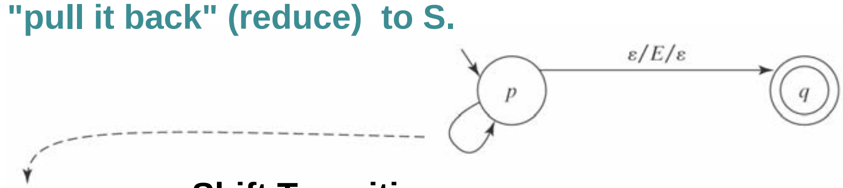
Top-down parser discovers a leftmost derivation of the input string (If any).
Bottom-up parser discovers a rightmost derivation (in reverse order)

Bottom-Up PDA

The idea: Let the stack keep track of what has been found.

(1) $E \rightarrow E + T$
(2) $E \rightarrow T$
(3) $T \rightarrow T * F$
(4) $T \rightarrow F$
(5) $F \rightarrow (E)$
(6) $F \rightarrow id$

Discover a rightmost derivation in reverse order. Start with the string of terminals and attempt to "pull it back" (reduce) to S.



Shift Transitions:

(7) $(p, \text{id}, \varepsilon), (p, \text{id})$
(8) $(p, (, \varepsilon), (p, ()$
(9) $(p,), \varepsilon), (p,))$
(10) $(p, +, \varepsilon), (p, +)$
(11) $(p, *, \varepsilon), (p, *)$

Reduce Transitions:

(1) $(p, \varepsilon, T + E), (p, E)$
(2) $(p, \varepsilon, T), (p, E)$
(3) $(p, \varepsilon, F * T), (p, T)$
(4) $(p, \varepsilon, F), (p, T)$
(5) $(p, \varepsilon,)E( ), (p, F)$
(6) $(p, \varepsilon, \text{id}), (p, F)$

Example:
id + id * id

When the right side of a production is on the top of the stack, we can replace it by the left side of that production…

…or not! That's where the nondeterminism comes in: choice between shift and reduce; choice between two reductions.

Hidden during class, revealed later: Solution to bottom-up example

A bottom-up parser is sometimes called a shift-reduce parser. Show how it works on id + id * id

State	stack	remaining input	transition to use
p	ε	id + id * id	7
p	id	+ id * id	6
p	F	+ id * id	4
p	T	+ id * id	2
p	E	+ id * id	10
p	+E	id * id	7
p	id+E	* id	6
p	F+E	* id	4
p	T+E	* id	11
p	*T+E	id	7
p	id*T+E	ε	6
p	F*T+E	ε	3
p	T+E	ε	1
p	E	ε	0
q	ε	ε	

Note that the top of the stack is on the left. This is what I should have done in the class for sections 1 and 2 (and I did do it for section 3).

Acceptance by PDA → derived from CFG

- Much more complex than the other direction.
- Nonterminals in the grammar that we build from the PDA M are based on a combination of M's states and stack symbols.
- It gets very messy.
- Takes 9½ dense pages in the textbook (265-274).
- I think we can use our limited course time better.

How Many Context-Free Languages Are There?

(we had a slide just like this for regular languages)

Theorem: For any finite input alphabet Σ, there is a countably infinite number of CFLs over Σ.

Proof:

- Upper bound: we can lexicographically enumerate all the CFGs.
- Lower bound: Each of {a}, {aa}, {aaa}, … is a CFL.

The number of languages over Σ is uncountable.

Thus there are more languages than there are context-free languages.

So there must be some languages that are not context-free.

Languages That Are and Are Not Context-Free

a*b* is regular.

$A^nB^n = \{a^n b^n : n \geq 0\}$ is context-free but not regular.

$A^nB^nC^n = \{a^n b^n c^n : n \geq 0\}$ is not context-free.
We will show this soon.

Is *every* regular language also context-free?

Showing that L is Context-Free

Techniques for showing that a language *L* is context-free:

1. Exhibit a **CFG** for *L*.

2. Exhibit a **PDA** for *L*.

3. Use the **closure properties** of context-free languages.

Unfortunately, these are weaker than they are for regular languages.
union, reverse, concatenation, Kleene star
intersection of a CFL with a regular language

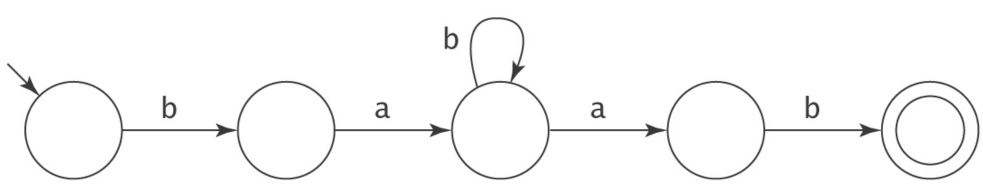
NOT intersection, complement, set difference

CFL Pumping Theorem

Show that *L* is Not Context-Free

Recall the basis for the pumping theorem for regular languages: A DFSM M.

If a string is longer than the number of M's states…

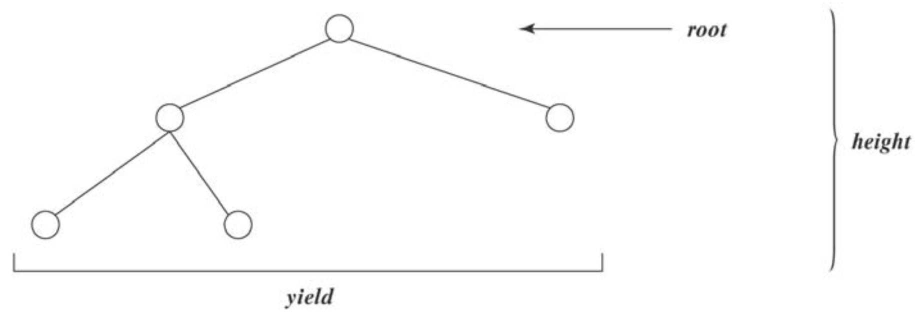


Why would it be hard to use a PDA to show that long strings from a CFL can be pumped?

Some Tree Geometry Basics

The ***height*** h of a tree is the length of the longest path from the root to any leaf.

The ***branching factor*** b of a tree is the largest number of children associated with any node in the tree.



Theorem: The length of the ***yield*** (concatenation of leaf nodes) of any tree T with height h and branching factor b is $\leq b^h$. **Shown in CSSE 230.**

A Review of Parse Trees

A ***parse tree***, (a.k.a. ***derivation tree***) derived from a grammar $G = (V, \Sigma, R, S)$, is a rooted, ordered tree in which:

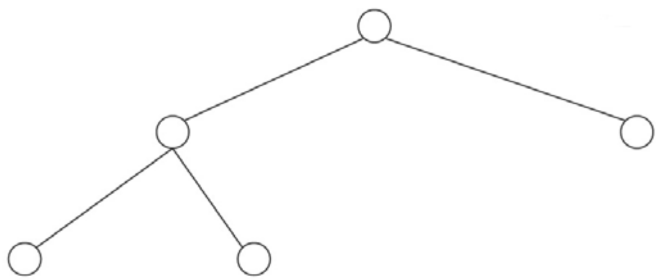
- Every leaf node is labeled with an element of $\Sigma \cup \{\varepsilon\}$,
- The root node is labeled S,
- Every interior node is labeled with an element of N (i.e., $V - \Sigma$),
- If m is a non-leaf node labeled X and the children of m (left-to-right on the tree) are labeled $x_1, x_2, \ldots, x_n$, then the rule $X \rightarrow x_1\ x_2 \ldots x_n$ is in R.

From Grammars to Trees

Given a context-free grammar *G*:

- Let *n* be the number of nonterminal symbols in *G*.
- Let *b* be the branching factor of *G*

Suppose that a tree *T* is generated by *G* and no nonterminal appears more than once on any path from the root:



The maximum height of *T* is:

The maximum length of *T*'s yield is:

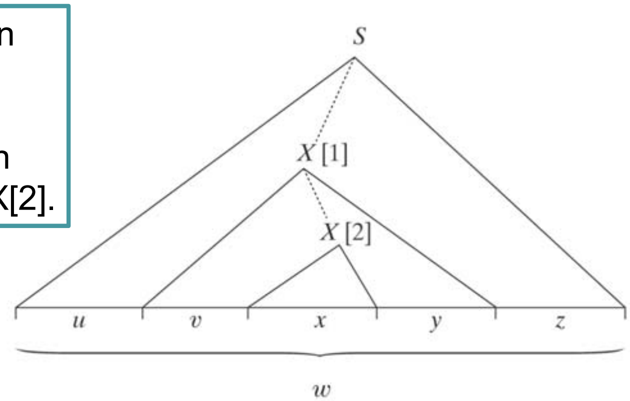
The Context-Free Pumping Theorem

We use parse trees, not machines, as the basis for our argument.
Let L = L(G), and let w$\in$L.

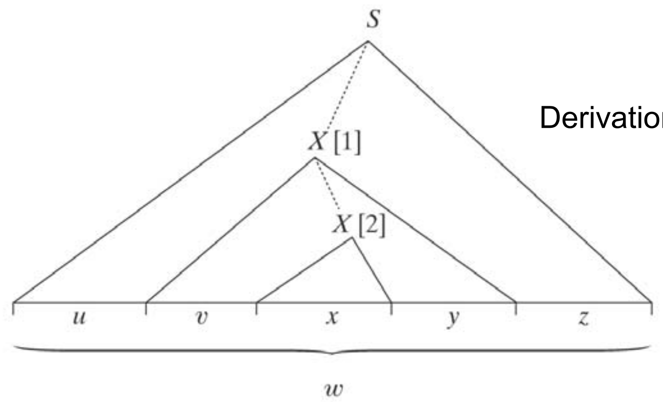
Let T be a parse tree for w such that has the smallest possible number of nodes among all trees based on a derivation of w from G.

Suppose L(G) contains a string w such that |*w*| is greater than b^n.
Then its parse tree must look like (for some nonterminal X):

X[1] is the lowest place in the tree for which this happens.
I.e., there is no other X in the derivation of x from X[2].



The Context-Free Pumping Theorem



Derivation of w

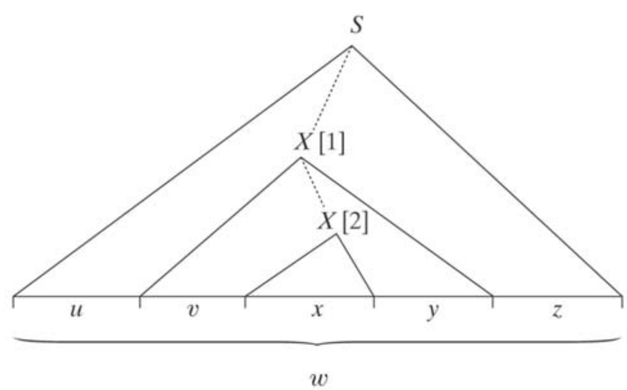
There is another derivation in *G*:

$S \Rightarrow^* uXz \Rightarrow^* uxz$,

in which, at X[1], the nonrecursive rule that leads to x is used instead of the recursive one that leads to vXy.

So ***uxz*** is also in *L*(*G*).

The Context-Free Pumping Theorem



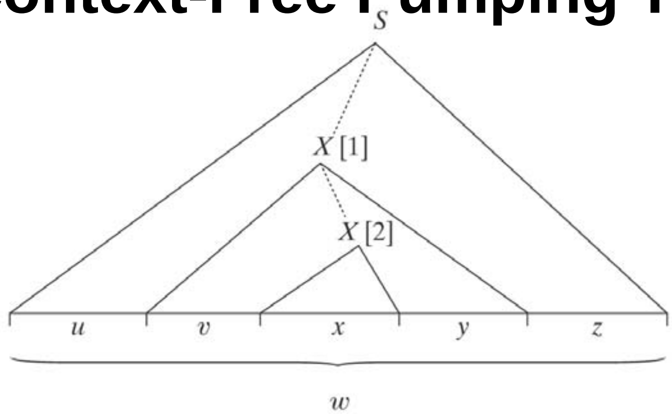
There are infinitely many derivations in *G*, such as:

$S \Rightarrow^* uXz \Rightarrow^* uvXyz \Rightarrow^* uvvXyyz \Rightarrow^* uvvxyyz$

Those derivations produce the strings:
uv^2xy^2z, uv^3xy^3z, uv^4xy^4z, …

So all of those strings are also in *L*(*G*).

The Context-Free Pumping Theorem

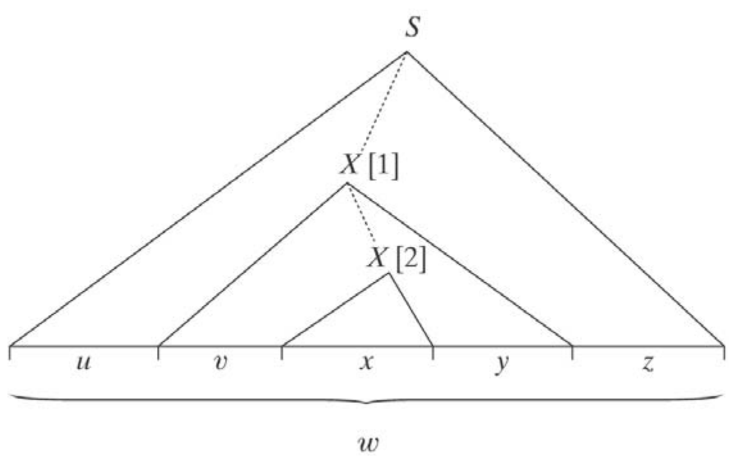


If $rule_1$ is $X \rightarrow X$a, we could have $v = \varepsilon$.

If $rule_1$ is $X \rightarrow$ aX, we could have $y = \varepsilon$.

But it is not possible that ***both*** v and y are ε. If they were, then the derivation $S \Rightarrow^* uXz \Rightarrow^* uxz$ would also yield w and it would create a parse tree with fewer nodes. But that contradicts the assumption that we started with a parse tree for w with the smallest possible number of nodes.

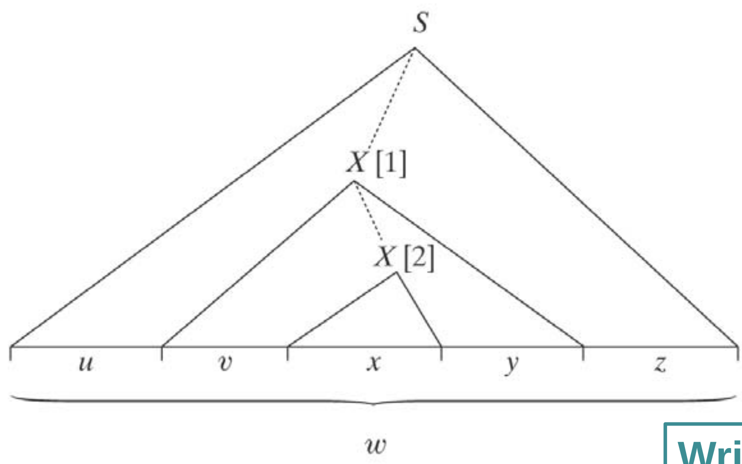
The Context-Free Pumping Theorem



The height of the subtree rooted at [1] is at most:

So $|vxy| \leq$.

The Context-Free Pumping Theorem



If L is a context-free language, then
$\exists k \geq 1$ $(\forall$ strings $w \in L$, where $|w| \geq k$
$(\exists u, v, x, y, z$ $(w = uvxyz$,
$vy \neq \varepsilon$,
$|vxy| \leq k$, and
$\forall q \geq 0$ $(uv^qxy^qz$ is in L)))).

Write it in contrapositive form. Try to do this before going on.

Pumping Theorem contrapositive

- We want to write it in contrapositive form, so we can use it to show a language is NOT context-free. **Original:**

If L is a context-free language, then
$\exists k \geq 1$ $(\forall$ strings $w \in L$, where $|w| \geq k$
$(\exists u, v, x, y, z$ $(w = uvxyz$,
$vy \neq \varepsilon$,
$|vxy| \leq k$, and
$\forall q \geq 0$ $(uv^qxy^qz$ is in L)))).

Contrapositive: If
$\forall k \geq 1$ $(\exists$ string $w \in L$, where $|w| \geq k$
$(\forall u, v, x, y, z$
$(w = uvxyz$,
$vy \neq \varepsilon$,
$|vxy| \leq k$, and
$\exists q \geq 0$ $(uv^qxy^qz$ is not in L)))),
then L is not a CFL.

Regular *vs*. CF Pumping Theorems

Similarities:

- We don't get to choose k.
- We choose w, the string to be pumped, based on k.
- We don't get to choose how w is broken up (into xyz or uvxyz)
- We choose a value for q that shows that w isn't pumpable.
- We may apply closure theorems before we start.

Things that are different in CFL Pumping Theorem:

- Two regions, v and y, must be pumped in tandem.
- We don't know anything about where in the strings v and y will fall in the string w. All we know is that they are reasonably "close together", i.e., $|vxy| \leq k$.
- Either v or y may be empty, but not both.

Pumping Theorem contrapositive

- We want to write it in contrapositive form, so we can use it to show a language is NOT context-free. **Original:**

If L is a context-free language, then

$$
\begin{aligned}
&\exists k \geq 1 \quad (\forall \text{ strings } w \in L, \text{ where } |w| \geq k \\
&\qquad (\exists u, v, x, y, z \quad (w = uvxyz, \\
&\qquad\qquad vy \neq \varepsilon, \\
&\qquad\qquad |vxy| \leq k, \text{ and} \\
&\qquad\qquad \forall q \geq 0 \ (uv^qxy^qz \text{ is in } L)))).
\end{aligned}
$$

Contrapositive: If

$$
\begin{aligned}
&\forall k \geq 1 \ (\exists \text{ string } w \in L, \text{ where } |w| \geq k \\
&\qquad (\forall u, v, x, y, z \\
&\qquad\quad (w = uvxyz, \\
&\qquad\quad vy \neq \varepsilon, \\
&\qquad\quad |vxy| \leq k, \text{ and} \\
&\qquad\quad \exists q \geq 0 \ (uv^qxy^qz \text{ is not in } L)))),
\end{aligned}
$$

then L is not a CFL.

Example: $A^nB^nC^n$

An Example of Pumping: $A^nB^nC^n$

$A^nB^nC^n = \{a^n b^n c^n, n \geq 0\}$

Choose $w = a^k b^k c^k$ *(we don't get to choose the k)*

1 | 2 | 3 (the regions: all a's, all b's, all c's)

If either *v* or *y* spans two regions, then let *q* = 2 (i.e., pump in once). The resulting string will have letters out of order and thus not be in $A^nB^nC^n$.

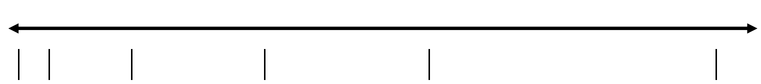
If both *v* and *y* each contain only one distinct character, set *q* to 2. Additional copies of at most two different characters are added, leaving the third unchanged.
We no longer have equal numbers of the three letters, so the resulting string is not in $A^nB^nC^n$.

An Example of Pumping: $\{ a^{n^2}, n \geq 0\}$

$L = \{a^{n^2}, n \geq 0\}$

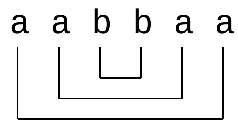
The elements of *L*:

n	*w*
0	ε
1	a^1
2	a^4
3	a^9
4	a^{16}
5	a^{25}
6	a^{36}



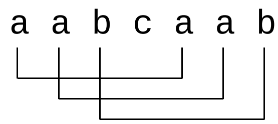
Nested and Cross-Serial Dependencies

PalEven = $\{ww^R : w \in \{a, b\}^*\}$



The dependencies are nested.

WcW = $\{wcw : w \in \{a, b\}^*\}$



Cross-serial dependencies.

Work with another student on these

- **WcW = {wcw : w $\in$ {a, b}*}**
- **$\{(ab)^n a^n b^n : n > 0\}$**
- **$\{x\#y : x, y \in \{0, 1\}^* \text{ and } x \neq y\}$**