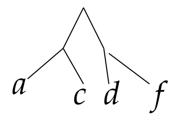
Proof of the Pumping Lemma

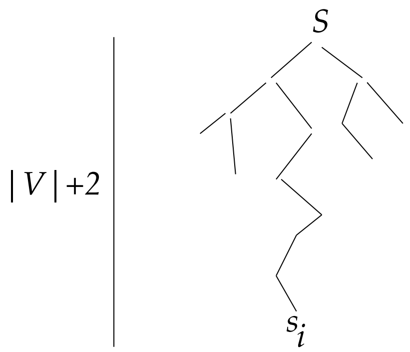
Proof Idea: Let G be a CFG generating A, V the vars. of G.
Suppose G has at most b symbols on the RHS of any rule.
Then parse trees for G have at most b-way branching,
and any tree of height $\leq h$ is the tree for a string of length at most b^h.

Ex. b=2, h=2:



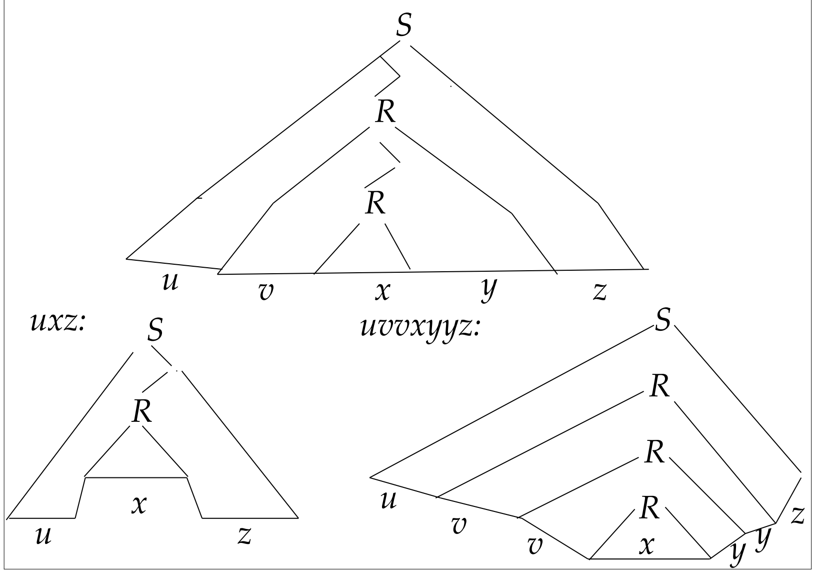
Consider any s in A that's "very long" ($\geq b^{|V|+2}$)
Since s is in A, it has a parse tree, and the parse tree is "very tall" ($\geq |V|+2$) since s is "very long"
Therefore, the parse tree must have a long path ($\geq |V|+2$)

Proof of the Pumping Lemma



Consider the variables along this long path.
There must be a repeat of some variables on the path, since there are |V| variables in G, and there are |V|+1 on the long path (by the pigeonhole principle)
Let R be the last repeated variable from the leaf.
(Take last repeat to satisfy length condition on vxy)

Surgery on Parse Trees



Equivalence of PDA's and CFG's

Th. L is context-free if and only some PDA recognizes L.

$\longrightarrow$ **:** *Proof idea: if L is context-free, we construct a PDA to simulate the grammar of L to see if input w can be generated. The PDA can nondeterministically guess rules of G to apply, and accept only if it generates the input string. The PDA's stack is used to hold intermediate strings of terminals and variables generated so far. As terminals are generated in the beginning of the string, they are checked against the input as its being read.*

$\longleftarrow$ **:**

Use PDA's for Parsing?

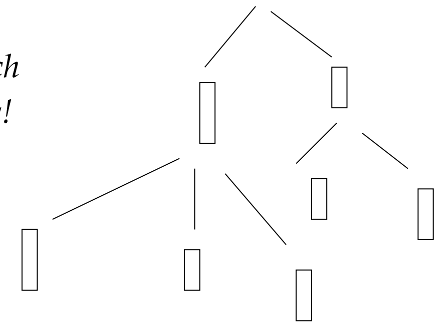
Nondeterministic PDA's not practical

To simulate, have to keep track of

Set of possible states

All possible stack configurations

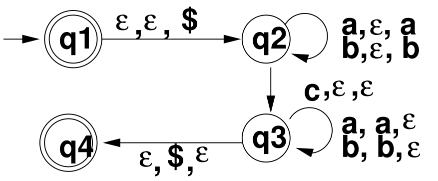
Too much memory!



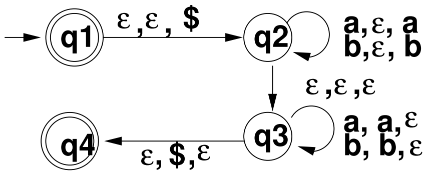
Deterministic PDA's

Subclass of PDA's where at most 1 next configuration
ε moves are allowed.

Ex. DPDA
{ wcw^R|w in {a,b } *}



Ex. Nondet. PDA
{ ww^R|w in {a,b } *}



Deterministic PDA's Closure under Complement

Lemma: The languages accepted by DPDA's are closed under the operation of complement.

Proof idea: Would like to just interchange final and non-final. This doesn't work 1) because the original DPDA M may not get beyond some initial input (because of no next move, or in an infinite loop of ε moves); the new DPDA M′ will have to accept all strings with that initial prefix;

2) due to ε moves after reading the input; some of the states may be final, and some non-final, and so both M and M′ would accept! Idea is to build into the FA the info needed to detect 1. and 2.

HW 2 showed that CFL's are not closed under complement. Therefore, DPDA languages are a proper subset of CFL's.

Parsing Methods

Top-Down	**Bottom-Up**
Construct parse tree	
From Root	*From Leaves*
Choose rule S → ABC *Expand parse tree*	*Reduce: replace RHS of rule by LHS*
S *A B C*	*Successful if reduce input to start symbol*

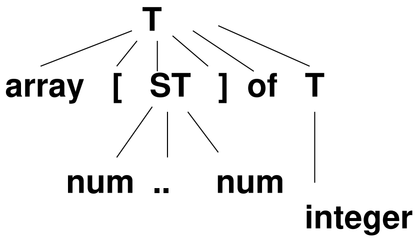
Big Question: How choose rules?

Example

T → id | array [ST] of T | ST

ST → integer | character | num..num

input: array [num..num] of integer



Why Easy?

Example

S → 0S1 | ε

input: 0011

Not-so-easy Examples

E → E + T | T

T → T x F | F

F → (E) | a

a + a vs a x a?

How know when stop applying rule E → E+T??

Add F → id | id(E) *(allows function calls)*

VERYLONGID( ) ???

Don't know till get to (which rule to use!

Rewriting Grammars for Top-down Parsing

Def: G is *immediately left-recursive* if
A → A u is a rule of G.

Ex. E → E + T T → T x F

Not good! Could get into infinite loop when using.

To eliminate:

Replace	By
E → E + T | T	E → TW W → +TW | ε
T → T x F | F	T → F V V → xF V | ε

Left Factoring

Ex. $F \rightarrow id \mid id(E)$ Replace by

$F \rightarrow id\ N$

$N \rightarrow \varepsilon \mid (E)$

Then always choose rule $F \rightarrow id\ N$

when read id

ε Defer choice between identifier and function call

till later!

There are systematic methods for generating top down parsers, if grammar in right form

Left-factoring and eliminating recursion are useful!