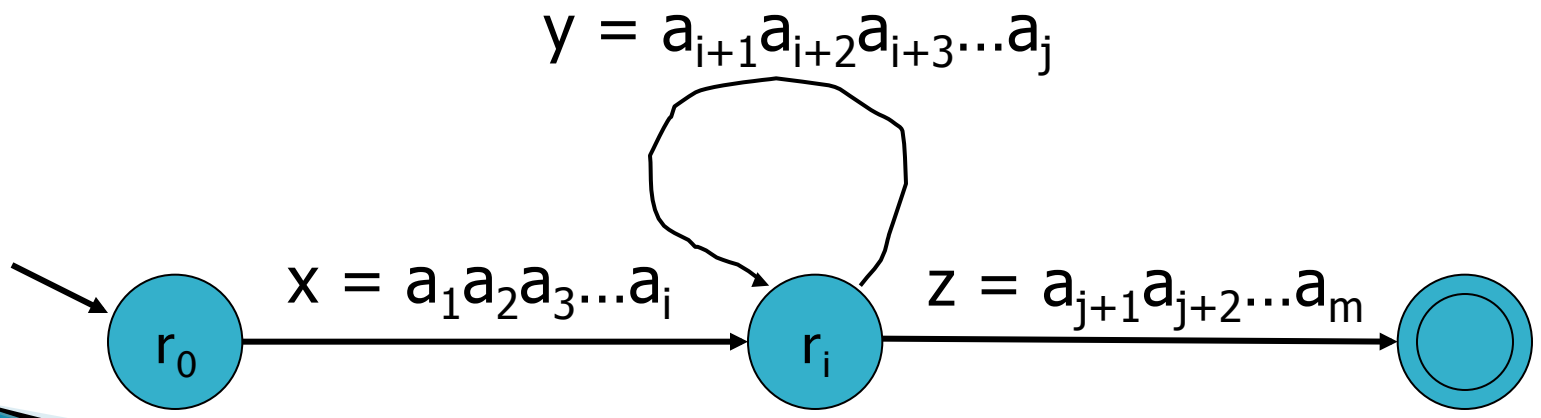
Non-Context Free Languages

The Pumping Lemma for Regular Languages

- Let L be a regular language
- Then there exists a constant p (which varies for different languages), such that for every string $s \in L$ with $|s| \geq p$, s can be expressed as $s = xyz$ such that:
  - $|y| > 0$
  - $|xy| \leq p$
  - For all $k \geq 0$, the string xy^kz is also in L.

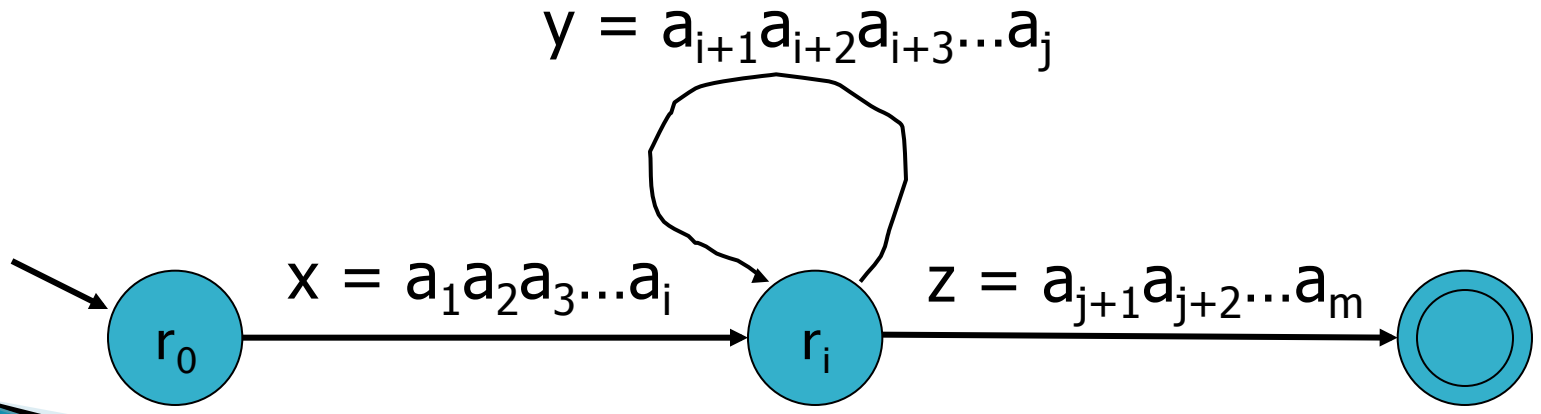
The Pumping Lemma for Regular Languages

- Consider a string $s = a_1a_2a_3\ldots a_m \in L$
- Suppose s is a long string – comprising at least as many symbols as the DFA for language L has states
  - At least one state must be visited twice



The Pumping Lemma for Regular Languages

- Why can't we use this same argument for CFLs?
  - Because PDAs involve the stack as well, and the stack may be different when you return to the repeated state

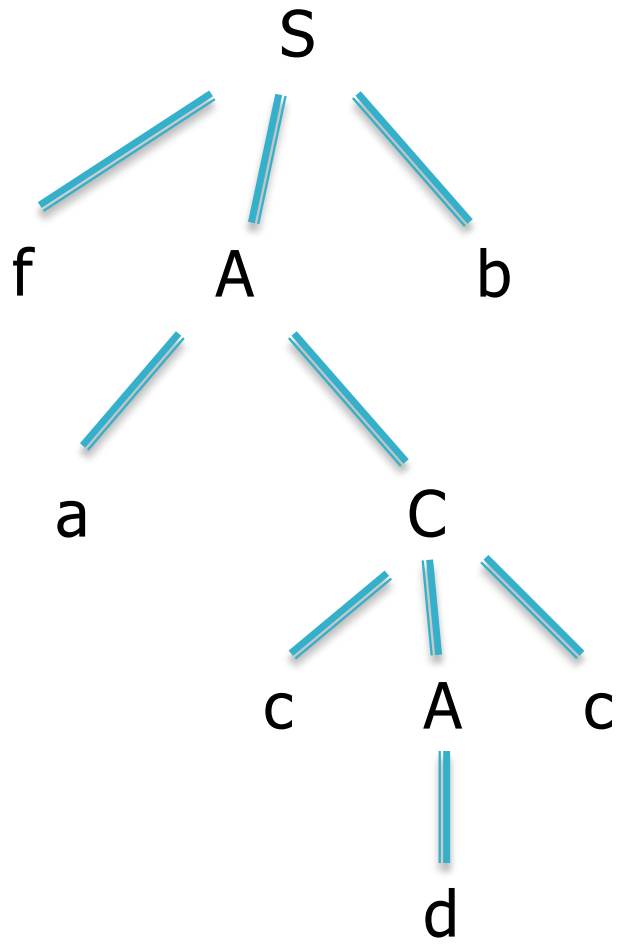


The Pumping Lemma for CFLs

- With CFLs
  - A string is represented by its derivation (or parse tree) based on the productions (rules) of a CFG.
  - The idea behind the Pumping Lemma for CFLs:
    - If a string is long enough, then at least one variable in its derivation will have to be repeated along one of its parse tree paths (from root node to leaf node).
    - We can repeatedly reapply productions for the repeated variable and the resultant string will also be in the language

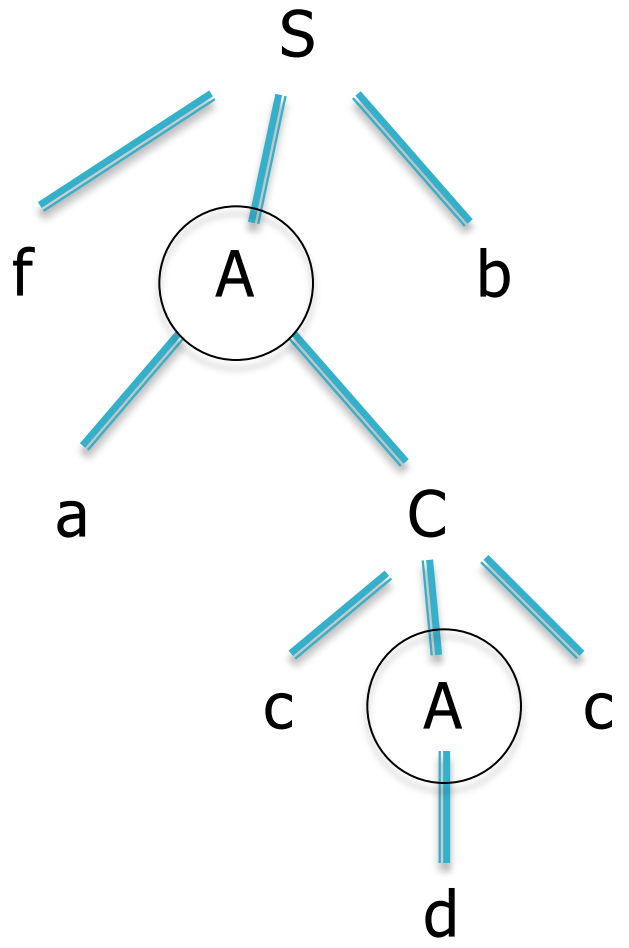
The Pumping Lemma for CFLs

- Example:
  - S → fAb
  - A → aC | d | ε
  - C → cAc
- Example string:
  - facdcb



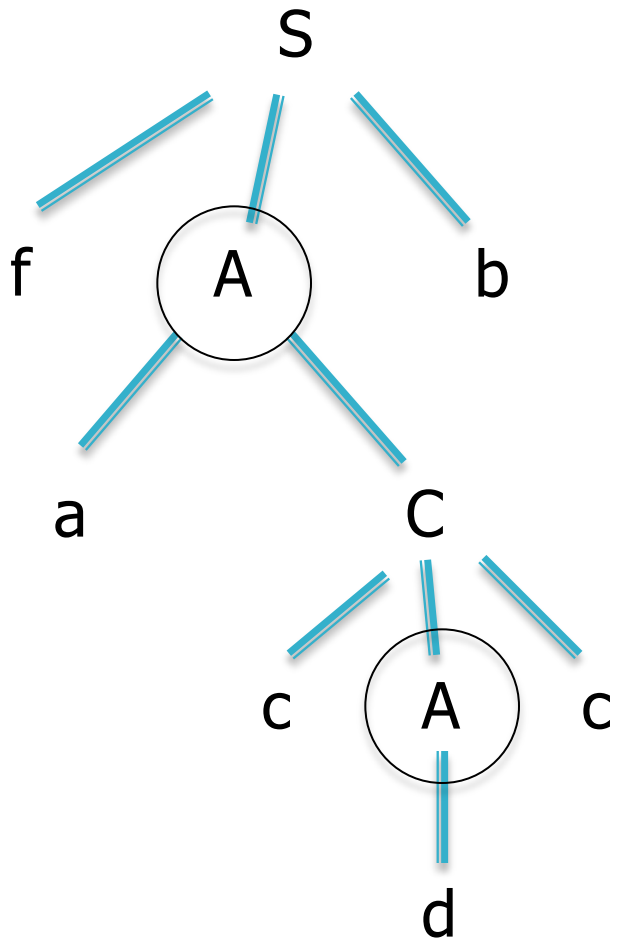
The Pumping Lemma for CFLs

- Idea – for a long enough string, eventually one of the variables has to repeat along one of the paths in the derivation of that string



The Pumping Lemma for CFLs

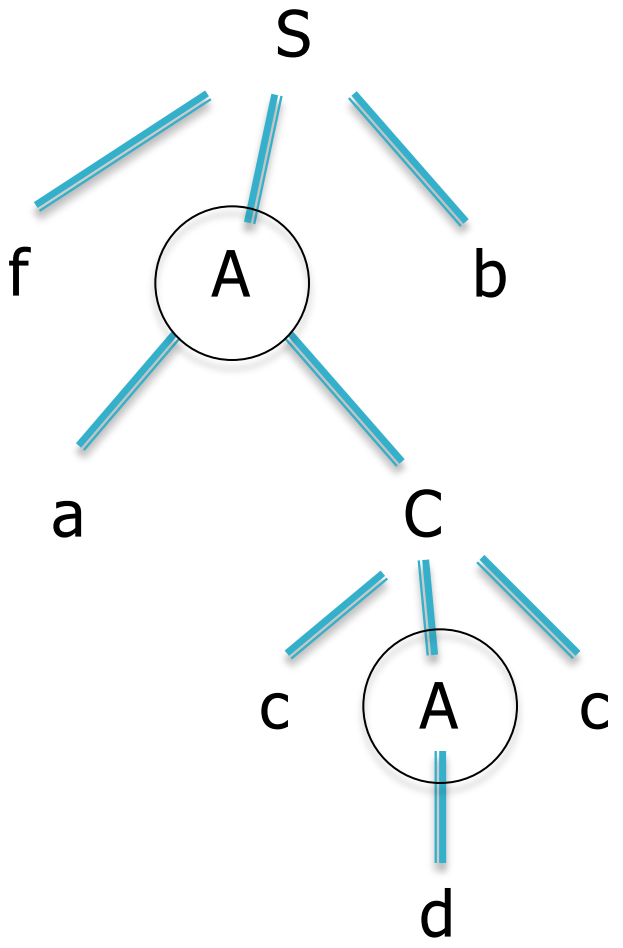
- Why is this true?
  - Suppose the grammar has V variables (it must have some finite number of variables)
  - Identify the longest path in a string's parse tree derivation
  - If this path is at least V+2 symbols long including the terminal leaf node (i.e. the depth of the tree is at least V+1) then one of the variables must repeat along its path



(Pigeonhole property)

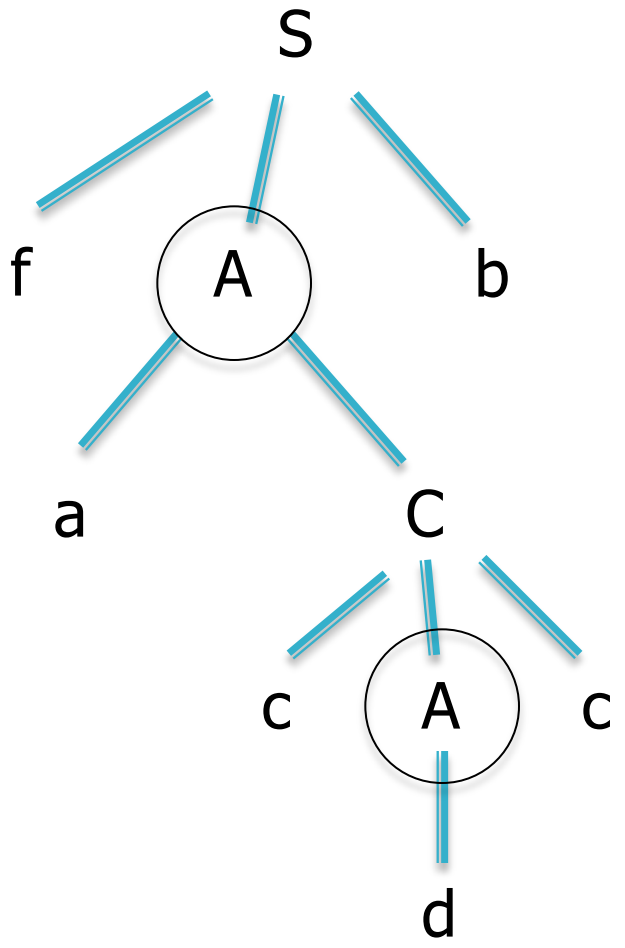
The Pumping Lemma for CFLs

- OK, so how long does the string need to be to guarantee that there is a path in its derivation of depth at least V+1?
  - Let K be the maximum length of any string (variables + terminals) on the right-hand side of any of the grammar rules for the language
    - That means the parse tree can branch into at most K children at each level
    - After V levels, the tree could have as many as K^V leaf nodes (terminals)
    - If there are more leaf nodes than K^V (i.e. if the string is longer than this many symbols), we know there must be at least one path of depth at least V+1



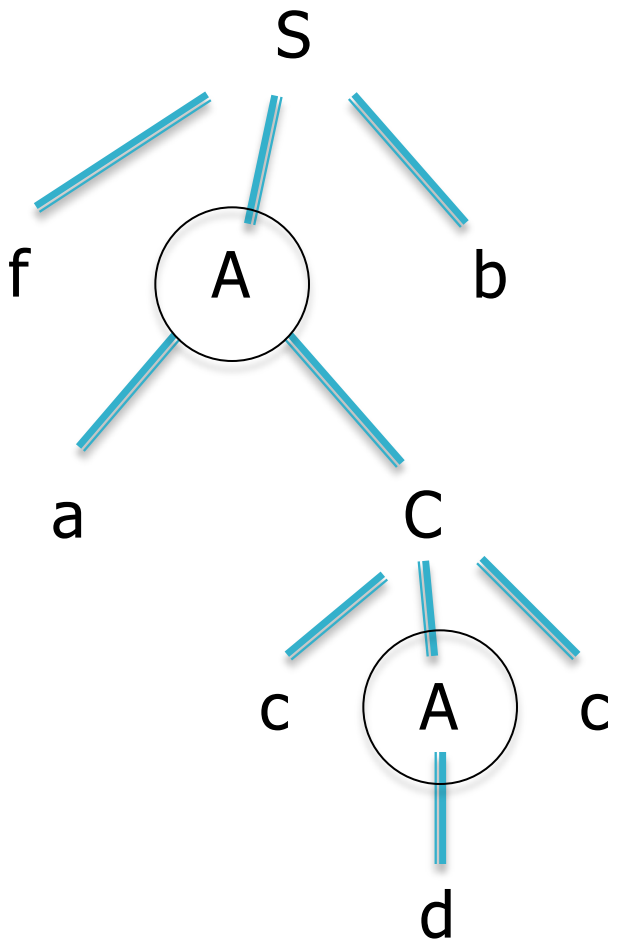
The Pumping Lemma for CFLs

- So what does this mean?
  - There is some length, p, and it may be a large number, but that doesn't matter, such that any string in the grammar with length at least p is guaranteed to have a parse tree with a variable that repeats along one of its paths
  - We'll consider the parse tree with the fewest nodes (in the case of ambiguity and multiple possible derivations)



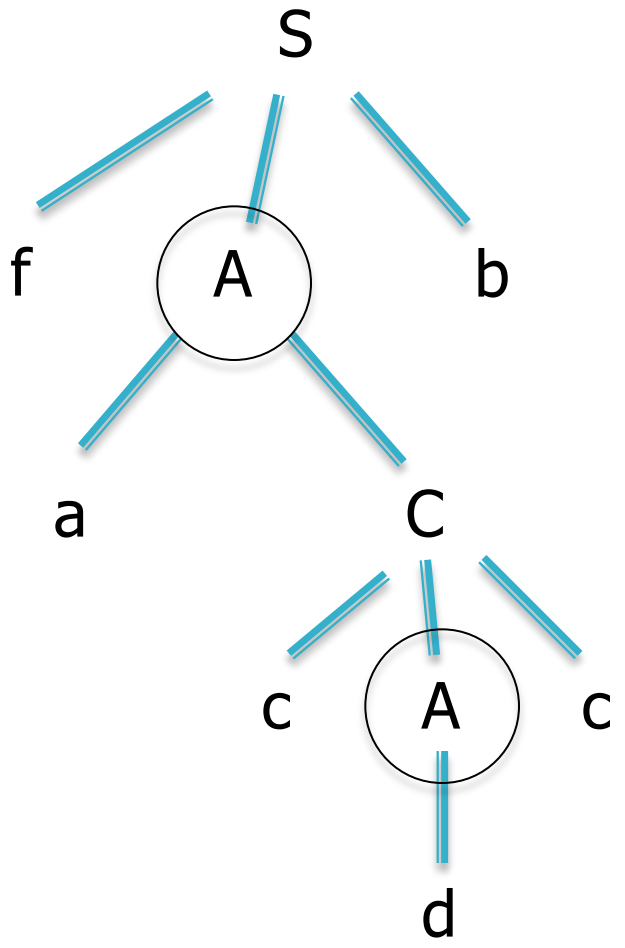
The Pumping Lemma for CFLs

- Now consider dividing our string up into 5 pieces: uvxyz
- u: everything that happens in the parse tree to the left of the the first occurrence of the repeat variable (f)
- v: everything originating from the first occurrence of the repeat variable, but left of the second occurrence (ac)
- x: everything originating from the second occurrence of the repeat variable (d)
- y: everything originating from the first occurrence of the repeat variable, but right of the second occurrence (c)
- z: everything that happens in the parse tree to the right of the first occurrence of the repeat variable (b)



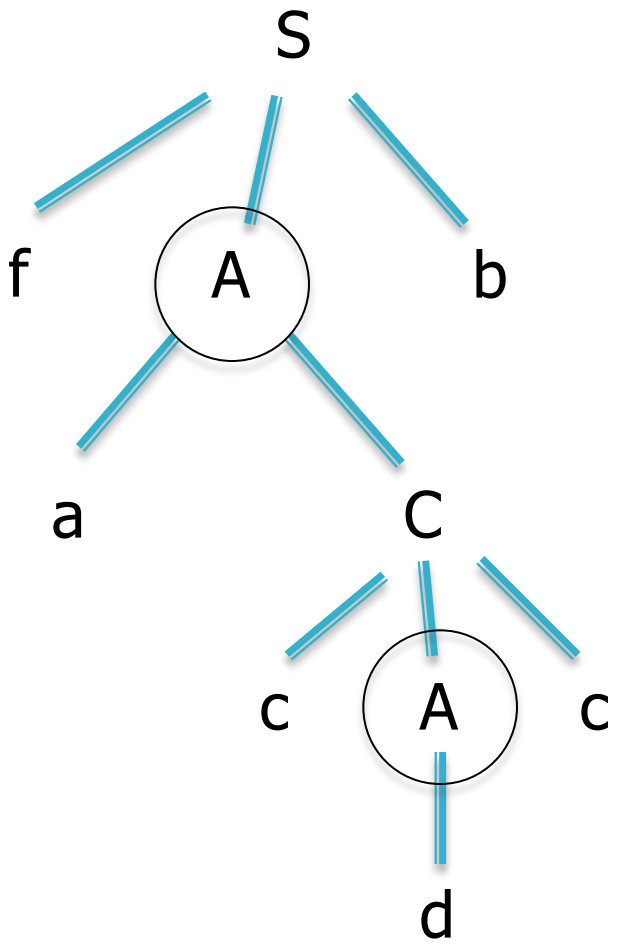
The Pumping Lemma for CFLs

- The string uvxyz is in the grammar
- The string derived starting from the first occurrence of the repeat variable is vxy
- The string derived starting from the second occurrence of the repeat variable is x
- Since a subtree rooted at the repeat variable is a valid derivation based on the grammar, we can swap out any subtree rooted at a given repeat variable for another subtree rooted at a different occurrence of that same repeat variable and the resulting string is still in the grammar
  - (swap out their corresponding strings)



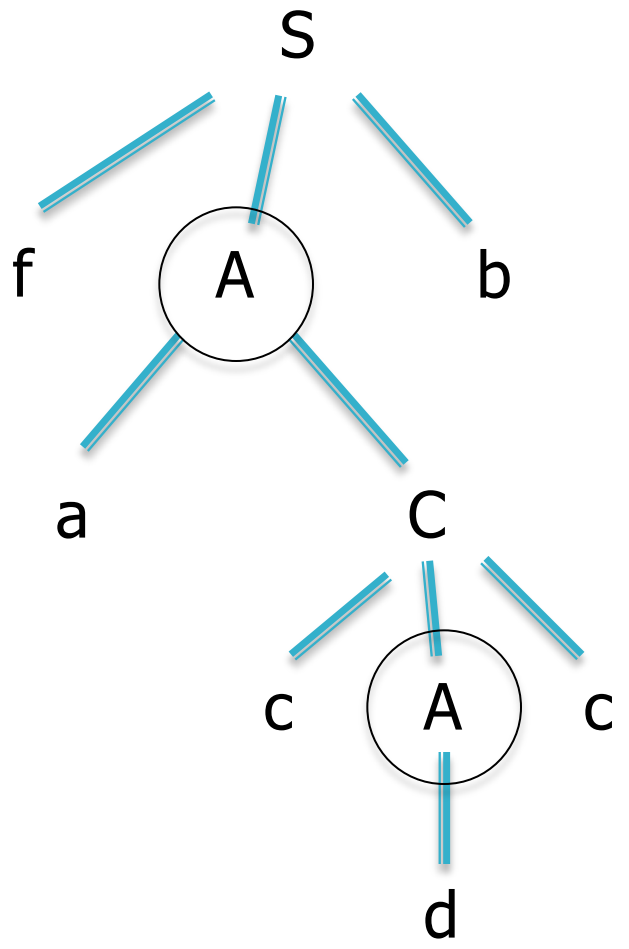
The Pumping Lemma for CFLs

- uvxyz is a valid string
- So is uv (vxy) yz
  - Derived by replacing the subtree rooted at the second occurrence of the repeat variable with the subtree rooted at the first occurrence
- So is uv (v (vxy) y) yz
- So is u (x) z
  - Derived by swapping in the reverse direction



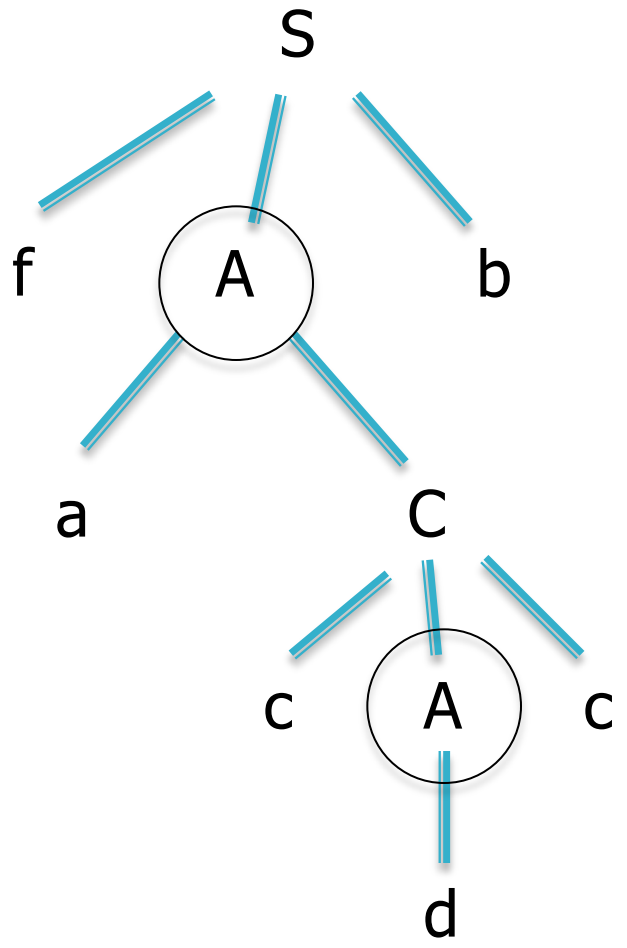
The Pumping Lemma for CFLs

- Any string of the form: uv^kxy^kz is a valid string in the language ($k \geq 0$)
- Also can be shown (technical details in the book, not complicated):
- $|vy| > 0$
- $|vxy| \leq p$



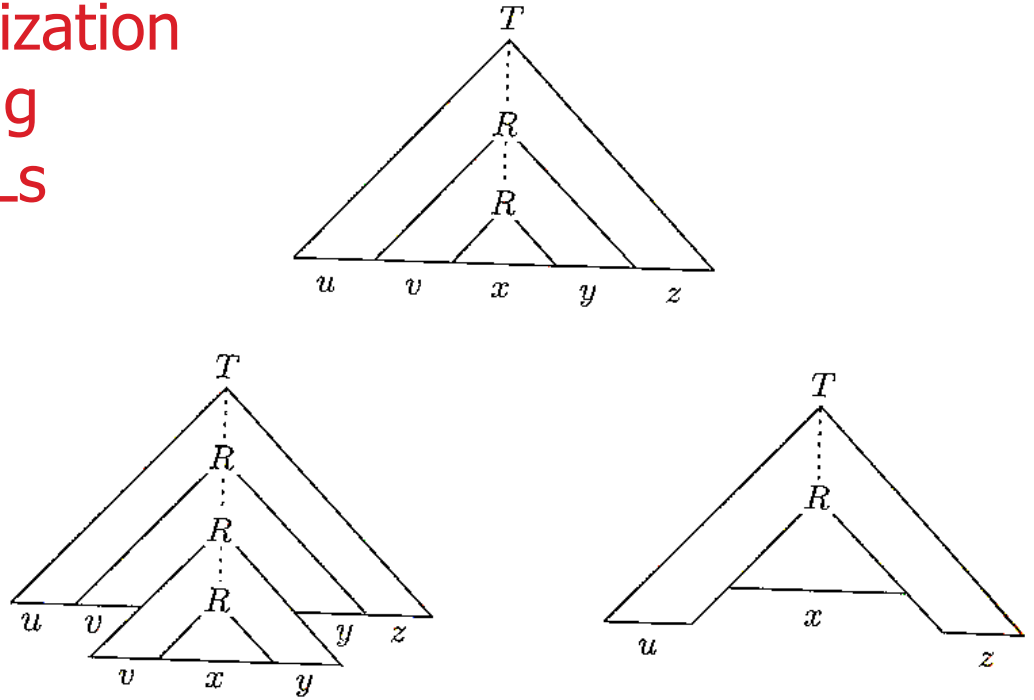
The Pumping Lemma for CFLs

- That is the pumping lemma for CFLs: for any CFL, there exists a pumping length, p, such that any string, s, in the language of length at least p can be broken up into five pieces s = uvxyz such that:
- uv^kxy^kz ($k \geq 0$) is in the language
- $|vy| > 0$
- $|vxy| \leq p$



The Pumping Lemma for CFLs

Sipser's visualization of the pumping lemma for CFLs



The Pumping Lemma for CFLs

- Just like with the pumping lemma for regular languages, the real strength of the pumping lemma for CFLs is proving that languages are not context-free
  - Proof by contradiction
    - Assume that the language to be tested is a CFL
    - Use the pumping lemma to come to a contradiction
    - Original assumption about the language being a CFL is false
- You cannot prove a language to be a CFL using the Pumping Lemma

The Pumping Lemma for CFLs

- Using the Pumping Lemma
  - To show that a language L is not a CFL
    - Assume L is context-free
    - Choose an "appropriate" string s in L
    - Express s = uvxyz following rules of pumping lemma
    - Show that uv^kxy^kz is not in L, for some k
    - The above contradicts the Pumping Lemma
    - Our assumption that L is context-free is wrong
    - L must not be context-free

The Pumping Lemma for CFLs

- Remember the wording of the lemma:
  - FOR ALL strings $s \in L$ that are long enough,
  - THERE EXISTS a decomposition $s = uvxyz$
  - Such that uv^kxy^kz is in the language FOR ALL $k \geq 0$ (and other properties hold too)
- So to show that a language is NOT a CFL by contradiction, we assume that it is a CFL and then show that:
  - THERE EXISTS string $s \in L$ that is long enough (just need 1!)
  - NO MATTER how we decompose $s = uvxyz$ (we have to cover ALL legitimate decompositions!)
  - uv^kxy^kz is NOT in the language FOR SOME $k \geq 0$ (we just have to show 1 case that isn't in the language!)

The Pumping Lemma for CFLs

- Example:
  - $L = \{ a^n b^n c^n \mid n \geq 0 \}$
  - Strings of the form abc where number of a's, b's and c's are equal
  - Assume that L is context-free. Then by the pumping lemma all strings s in L with $|s| \geq p$ can be expressed as $s = uvxyz$ and
    - $|vy| > 0$
    - $|vxy| \leq p$
    - For any $k \geq 0$, $uv^k xy^k z \in L$

The Pumping Lemma for CFLs

- Example
  - $L = \{ a^n b^n c^n \mid n \geq 0 \}$
  - Choose an appropriate $s = a^p b^p c^p = uvxyz$
  - Since $|vxy| \leq p$ then vxy must consist of
    - All a's or all b's or all c's
    - Some a's and some b's
    - Some b's and some c's
  - Since $|vy| > 0$, it must be contributing something

The Pumping Lemma for CFLs

- In all three cases
  - uv^2xy^2z will not have an equal number of a's b's and c's.
  - Pumping Lemma says $uv^2xy^2z \in L$
  - Contradiction!
  - Our original assumption must be wrong.

  - L is not context-free.

The Pumping Lemma for CFLs

- By the same argument (same choice of s), we can show that:
  - $L = \{ w \in \{ a,b,c\}^* \mid n_a(w) = n_b(w) = n_c(w) \}$
- Is not context-free

The Pumping Lemma for CFLs

- Another Example:
  - $L = \{ a^i b^j c^k \mid i < j \text{ and } i < k \}$
  - Number of a's is less than the number of b's and also less than the number of c's

  - Assume that L is context-free. Then by the pumping lemma all strings s in L with $|s| \geq p$ can be expressed as $s = uvxyz$ and
    - $|vy| > 0$
    - $|vxy| \leq p$
    - For any $k \geq 0$, $uv^k xy^k z \in L$

The Pumping Lemma for CFLs

- Example
  - $L = \{ a^i b^j c^k \mid i < j \text{ and } i < k \}$
  - Choose an appropriate $s = a^p b^{p+1} c^{p+1} = uvxyz$
  - Since $|vxy| \leq p$ then vxy must consist of
    - Case 1: All a's or all b's or all c's
    - Case 2: Some a's and some b's
    - Case 3: Some b's and some c's

The Pumping Lemma for CFLs

- Let's consider each case individually:
  - Case 1: All a's or all b's or all c's
    - If vxy consists of all a's then uv^2xy^2z will contain at least as many a's as b's
    - If vxy consists of all b's then uv^0xy^0z will contain the same number or fewer b's than a's
    - If vxy consists of all c's then uv^0xy^0z will contain the same number or fewer c's than a's

The Pumping Lemma for CFLs

- Let's consider each case individually:
  - Case 2: Some a's and some b's
    - If v and y together contain at least one a, then uv^2xy^2z will contain at least as many a's as c's
    - If v and y together don't contain at least one a, then they contain at least one b, and nothing else, in which case uv^0xy^0z will contain at least as many a's as b's

The Pumping Lemma for CFLs

- Let's consider each case individually:
  - Case 3: Some b's and some c's
    - If vxy consists of only b's and c's then uv^0xy^0z will contain the same number or fewer c's or b's than a's

The Pumping Lemma for CFLs

- In all cases
  - We found a “pumped” (or unpumped) string that the pumping lemma said should be in the language but that did not maintain the relationship of a’s to b’s and c’s as specified in the language.
  - Contradiction!
  - Our original assumption must be wrong.

  - L is not context-free.

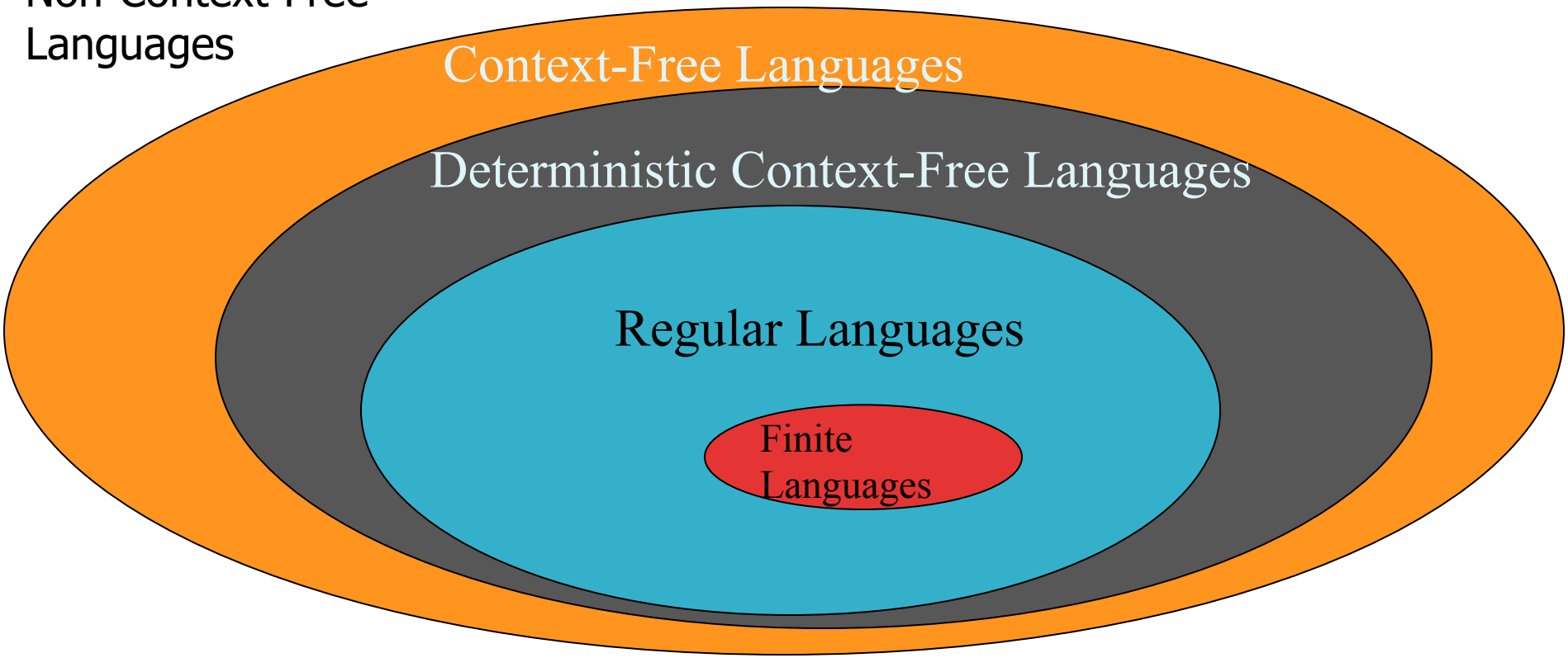
The Pumping Lemma for CFLs

- By the same argument (same choice of s), we can show that:
  - $L = \{ w \in \{ a,b,c\}^* \mid n_a(w) < n_b(w) \text{ and } n_a(w) < n_c(w) \}$
- Is not context-free

Deterministic PDAs

- As mentioned before
  - Our basic PDA is non-deterministic
  - Briefly, a deterministic PDA (DPDA) is one in which there are no “choices” in the transitions
- It turns out that PDAs are not equivalent to DPDAs
  - There are some context free languages that DPDAs can not represent (e.g. palindromes)
    - This is different than the case with regular languages, where DFA and NFA are equivalent

Now Our Picture Looks Like



Closure Properties

- We have already seen that CFLs are closed under:
  - Union
  - Concatenation
  - Kleene Star
- Regular Languages are also closed under
  - Intersection
  - Complementation
  - Difference
- What about Context-Free Languages?

Closure Properties

- CFLs are <u>not</u> closed under intersection
  - If L_1 and L_2 are CFLs then $L_1 \cap L_2$ is not necessarily a CFL.
  - Example:
    - $L_1 = \{a^i b^j c^k \mid i < j \}$
    - $L_2 = \{a^i b^j c^k \mid i < k \}$
    - Are both CFLs

Closure Properties

- CFLs are <u>not</u> closed under intersection

$L_1 = \{a^i b^j c^k \mid i < j\}$	$L_2 = \{a^i b^j c^k \mid i < k\}$
$S \rightarrow ABC$	$S \rightarrow AC$
$A \rightarrow aAb \mid \varepsilon$	$A \rightarrow aAc \mid B$
$B \rightarrow bB \mid b$	$B \rightarrow bB \mid \varepsilon$
$C \rightarrow cC \mid \varepsilon$	$C \rightarrow cC \mid c$

Closure Properties

- CFLs are <u>not</u> closed under intersection
  - $L_1 \cap L_2 = \{a^i b^j c^k \mid i < j \text{ and } i < k \}$
  - Which we just showed to be non-context-free.

Closure Properties

- CFLs are <u>not</u> closed under complement
  - We can express intersection of two CFLs as
    - $A \cap B = ((A \cap B)')' = (A' \cup B')'$
    - If CFLs were closed under complement, they would have to be closed under intersection as well (we know they are closed under union).
    - CONTRADICTION
- CFLs are <u>not</u> closed under difference.
  - We can express intersection of two CFLs as
    - (answer not provided; this is often a homework question)
    - If CFLs were closed under difference, they would have to be closed under intersection as well.
      - CONTRADICTION.

Closure Properties

- What went wrong?
  - Can't we apply the same construction as we did for the **complement** of RLs (Regular Languages)?
    - Reverse the accepting / non-accepting states
    - PDAs can "crash".
      - i.e. Fail by having no place to go.
      - Making non-accepting states accepting will not handle crashes.
        - (It will crash in both machines, so the string will not be accepted in either case.)
        - (we saw this in homework for NFAs)

Closure Properties

- What went wrong?
  - Can't we apply the same construction as we did for the **intersection** of RLs?
    - The states of M are an ordered pair (p, q) where $p \in Q_1$ and $q \in Q_2$
    - Informally, the states of M will represent the current states of M_1 and M_2 at each simultaneous move of the machines.
    - (Even ignoring for now the possibility of moving to multiple states at once)

Closure Properties

- What went wrong?
  - Can't we apply the same construction as we did for the **intersection** of RLs?
    - The problem is the stack.
    - Although we could try the same thing for PDAs and have a combined machine keep track of where both PDAs are at any one time…
    - We can't keep track of what's on both stacks at any given time.

Closure Properties

- However, if one of the CFLs does not use the stack (i.e. it is a DFA), then we can build a PDA that accepts $L_1 \cap L_2$.

- In other words:
  - If L_1 is a context free language and L_2 is a regular language, then $L_1 \cap L_2$ is context free.

  - See Problem 2.18 (and answer) in the textbook

Closure Properties

- Summary
  - CFLs are closed under
    - Union, Concatenation, Kleene Star
  - CFLs are NOT closed under
    - Intersection, Difference, Complement
  - But
    - The intersection of a CFL with a RL is a CFL