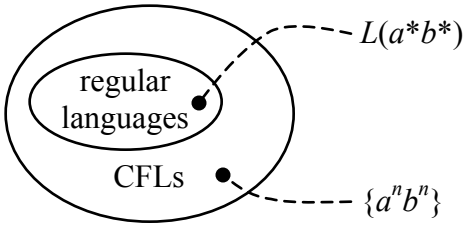
Chapter Fourteen:
The Context-Free Frontier

At this point we have two major language categories, the regular languages and the context-free languages, and we have seen that the CFLs include the regular languages, like this:



Are there languages outside of the CFLs? In this chapter we will see that the answer is *yes*, and we will see some simple examples of languages that are not CFLs.

- We have already seen that there are many *closure properties* for regular languages.
  - Given any two regular languages, there are many ways to combine them—intersections, unions, and so on—that are guaranteed to produce another regular language.
- The context-free languages also have some closure properties, though not as many as the regular languages.

Outline

- 14.1 Pumping Parse Trees
- 14.2 The Language $\{a^n b^n c^n\}$
- 14.3 Closure Properties For CFLs
- 14.4 Non-Closure Properties
- 14.5 A Pumping Lemma
- 14.6 Pumping-Lemma Proofs
- 14.7 The Languages $\{xx\}$

Non-context-free Languages

- We have seen that one of the hallmarks of context-free languages is some notion of symmetry:
  - Matching a's and b's
  - Balanced parentheses
  - Etc.
- It is precisely this symmetry that we use to show that there are non-context-free languages.

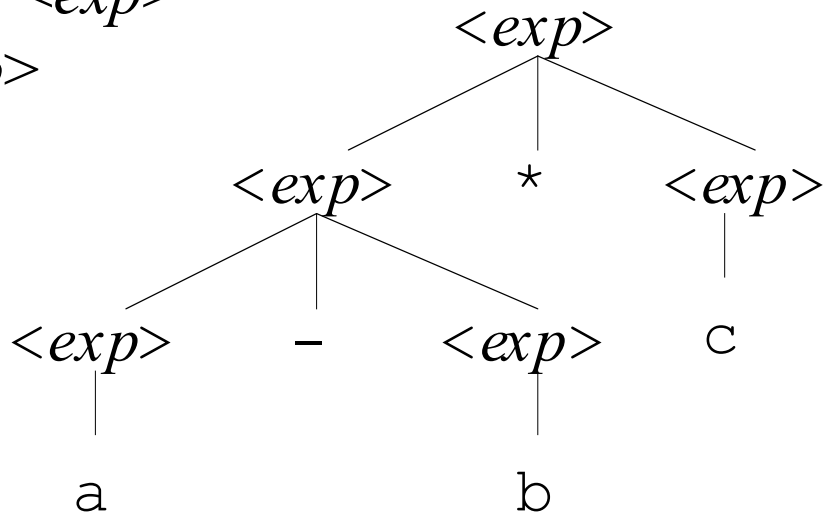
Parse Trees

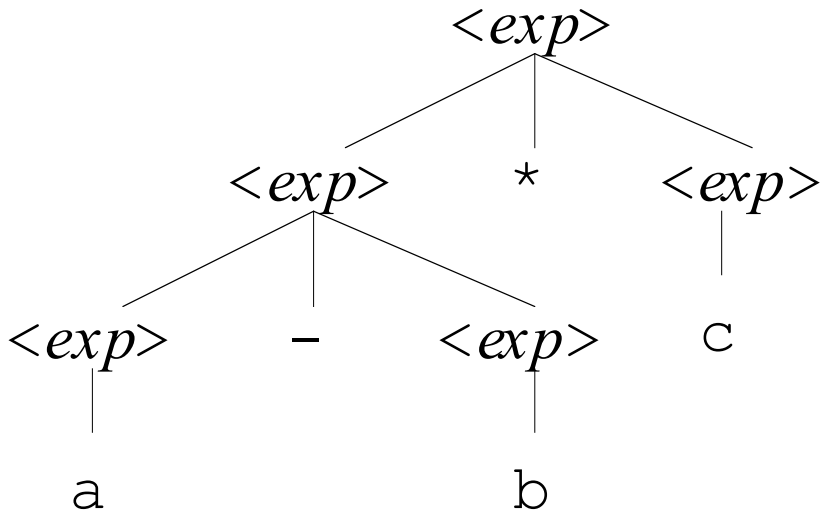
- We've treated productions as rules for building strings
- Now think of them as rules for building *trees:*
  - Start with *S* at the root
  - Add children to the nodes, always following the rules of the grammar: $R \rightarrow x$ says that the symbols in *x* may be added as children of the nonterminal symbol *R*
  - Stop only when all the leaves are terminal symbols
- The result is a *parse tree*

Example

<exp> ::= <exp> - <exp> | <exp> * <exp> | <exp> = <exp>
| <exp> < <exp> | (<exp>) | a | b | c

<exp> ⇒ <exp> * <exp>
⇒ <exp> - <exp> * <exp>
⇒ a- <exp> * <exp>
⇒ a-b* <exp>
⇒ a-b*c





- The parse tree specifies:
  - Syntax: it demonstrates that `a-b*c` is in the language
  - Also, it is a plan for evaluating the expression when the program is run
  - First evaluate `a-b`, then multiply that result by `c`
- It specifies how the parts of the program fit together
- And that says something about what happens when the program runs

Pumping Parse Trees

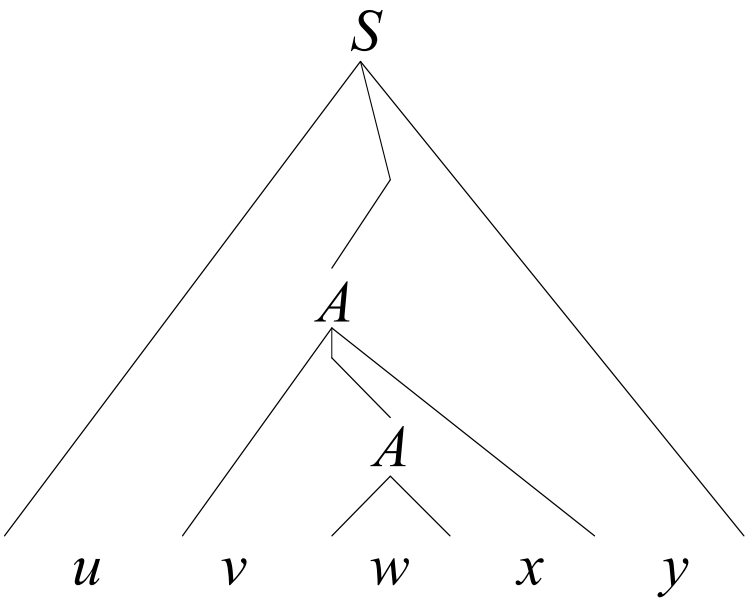
- A *pumping parse tree* for a CFG $G = (V, \Sigma, S, P)$ is a parse tree with two properties:
  1. There is a node for some nonterminal symbol *A*, which has that same nonterminal symbol *A* as one of its descendants
  2. *The terminal string generated from the ancestor A is longer than the terminal string generated from the descendant A*
- Like every parse tree, a pumping parse tree shows that a certain string is in the language
- Unlike other parse trees, it identifies an infinite set of other strings that must also be in the language…
- *In other words: a grammar that produces an infinite set of strings has to be recursive in its non-terminals.*

Lemma 14.1.1

Consider the grammar G:
S → u A y
A → v A x
A → w
with A,S ∈ V and u,v,w,x,y ∈ Σ*

Recursive Rule!

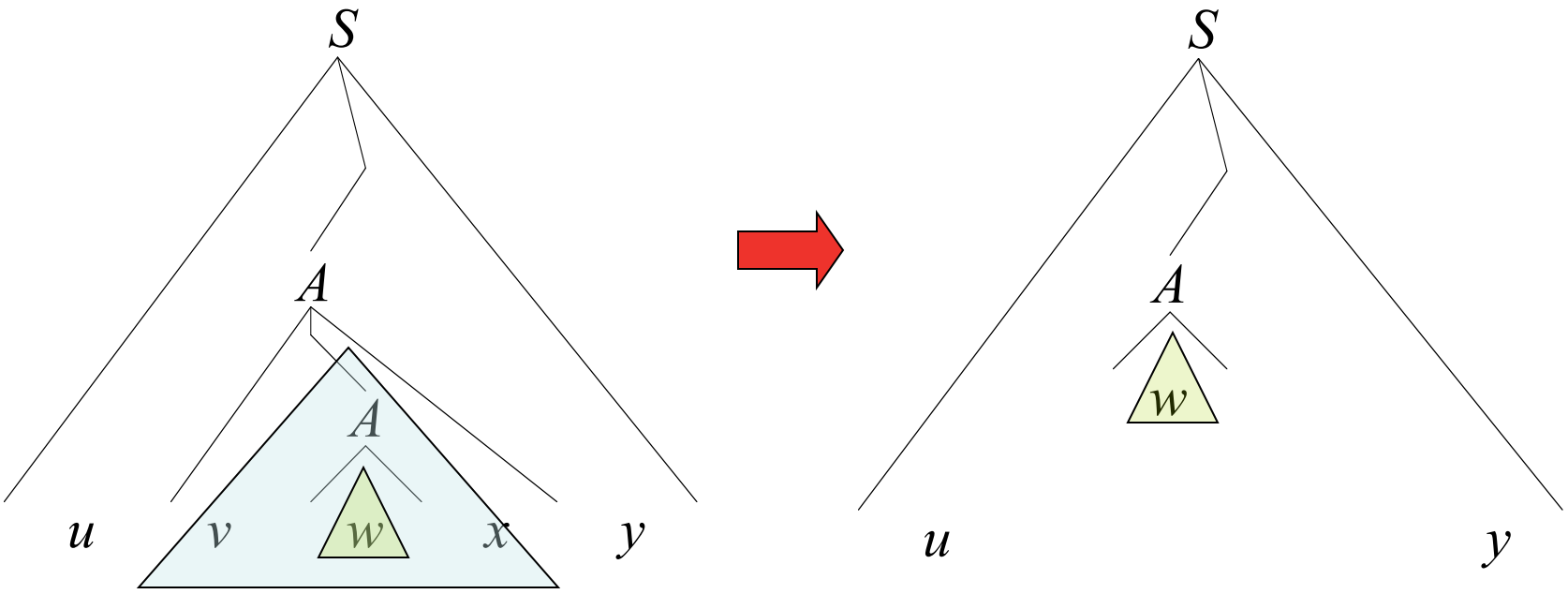
The grammar *G* generates a pumping parse tree with yield as shown, then *L*(*G*) includes uv^iwx^iy for all *i*.



S ⇒ uAy ⇒ uvAxy ⇒ uvwxy

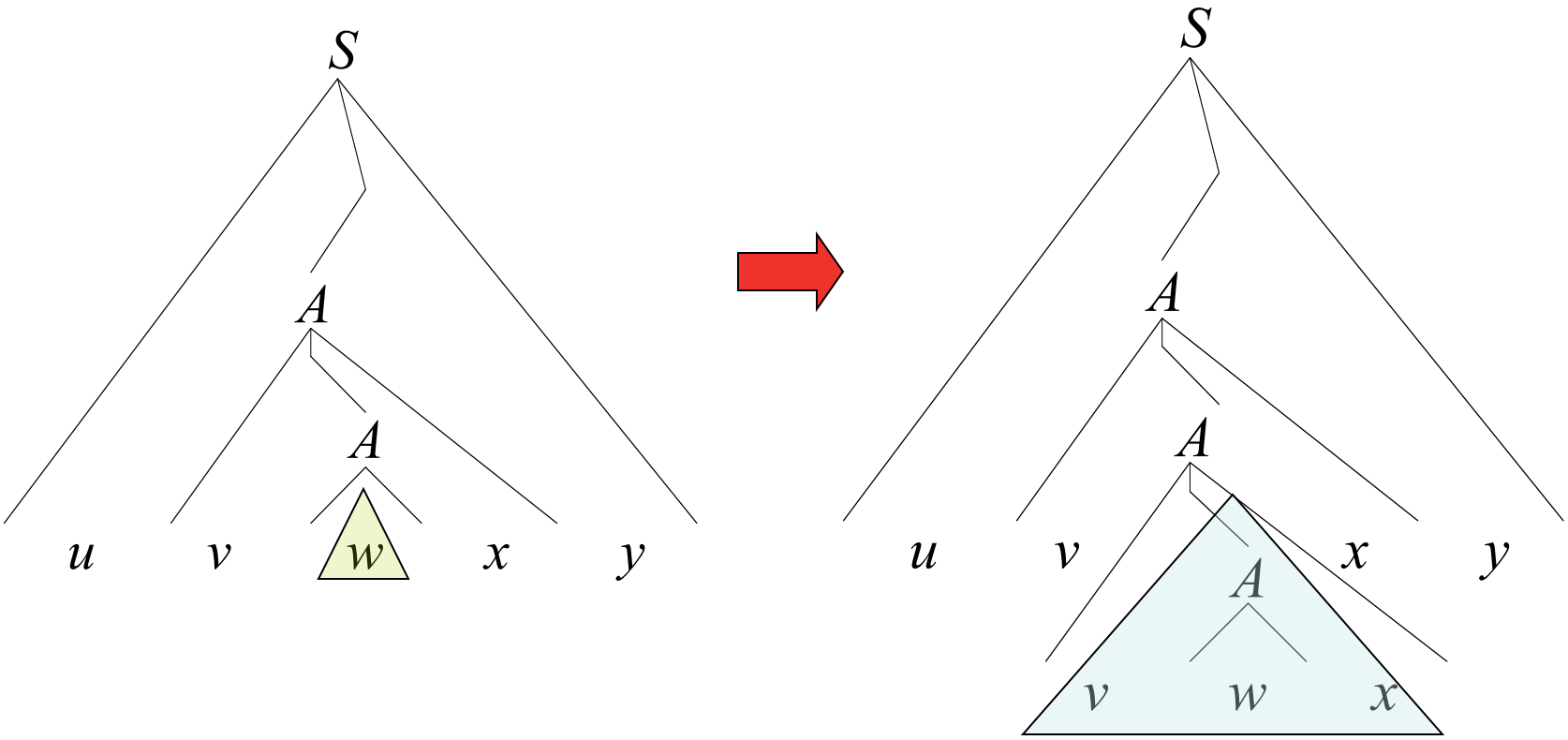
- As shown:
  - *uvwxy* is the whole derived string
  - *A* is the nonterminal that is its own descendant
  - *vwx* is the string derived from the ancestor *A*
  - *w* is the string derived from the descendant
  - |*vwx*| > |*w*|, so *v* and *x* are not both ε
- There are two subtrees rooted at *A*
- We can make other legal parse trees by substitution using the recursive rule…

Cut And Paste, *i* = 0



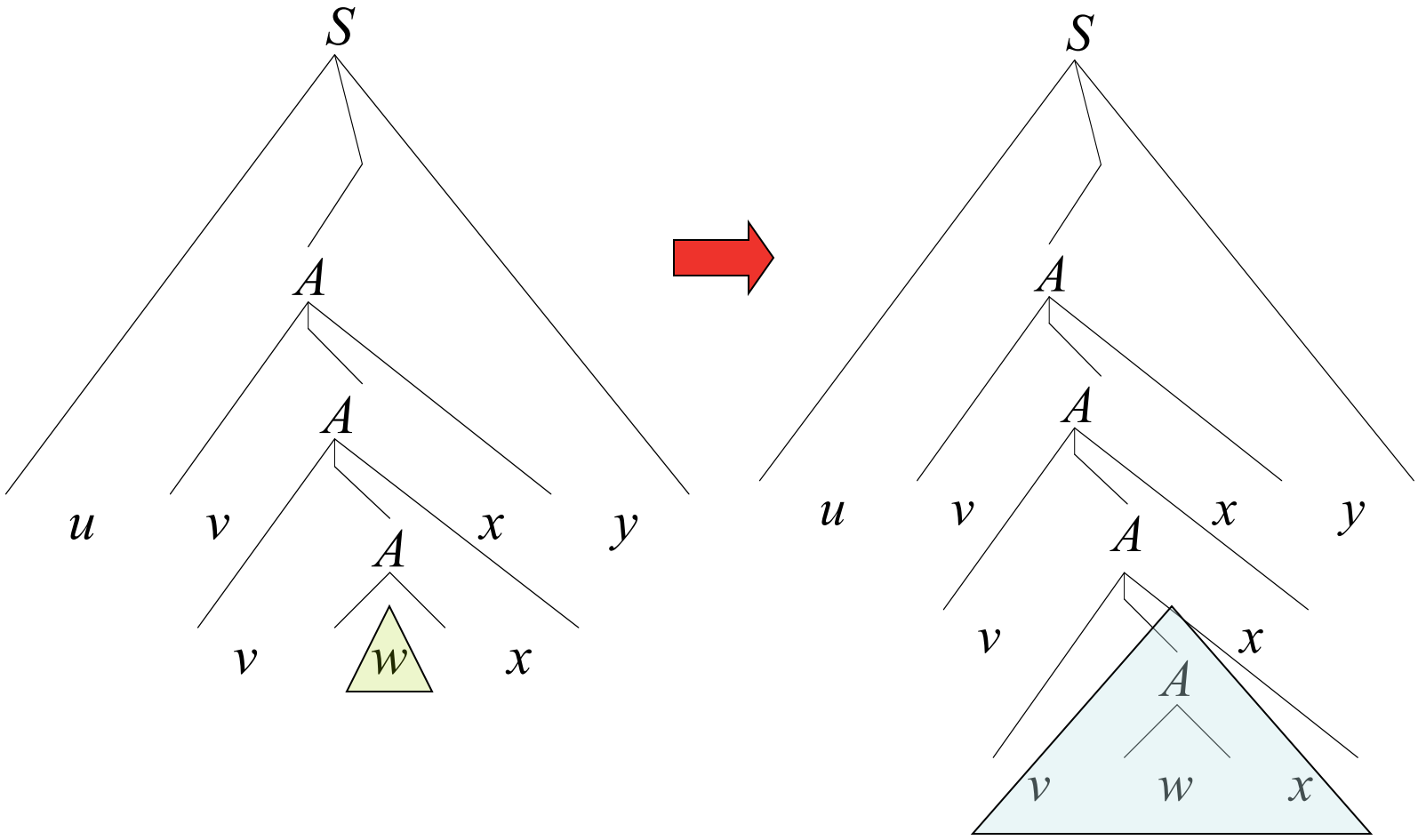
- We can replace the *vwx* subtree with the *w* subtree
- That makes a parse tree for *uwy*
- That is, uv^iwx^iy for $i = 0$
- Corresponds to the derivation: S ⇒ u A y ⇒ u w y
- We used the recursive rule *zero* times.

Cut And Paste, $i = 2$



- We can replace the *w* subtree with the *vwx* subtree
- That makes a parse tree for *uvvwxxy*
- That is, uv^iwx^iy for $i = 2$
- Corresponds to the derivation:
  S ⇒ uAy ⇒ uvAxy ⇒ uvvAxxy ⇒ uvvwxxy
- We used the recursive rule *twice*.

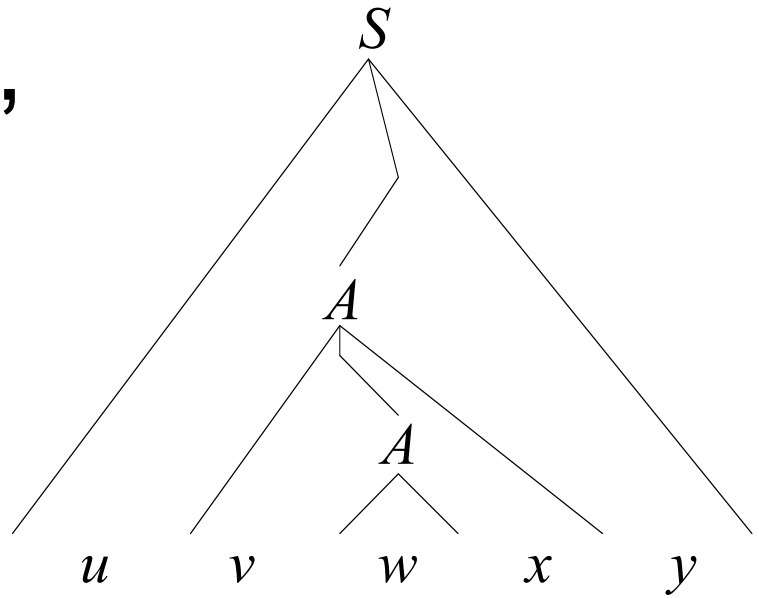
Cut And Paste, $i = 3$



- We can replace the *w* subtree with the *vwx*, again
- That makes a parse tree for *uvvvwxxxy*
- That is, uv^iwx^iy for $i = 3$ (you get the idea….)

Lemma 14.1.1, Continued

If a grammar G generates a pumping parse tree with yield as shown, then $L(G)$ includes uv^iwx^iy for all i.



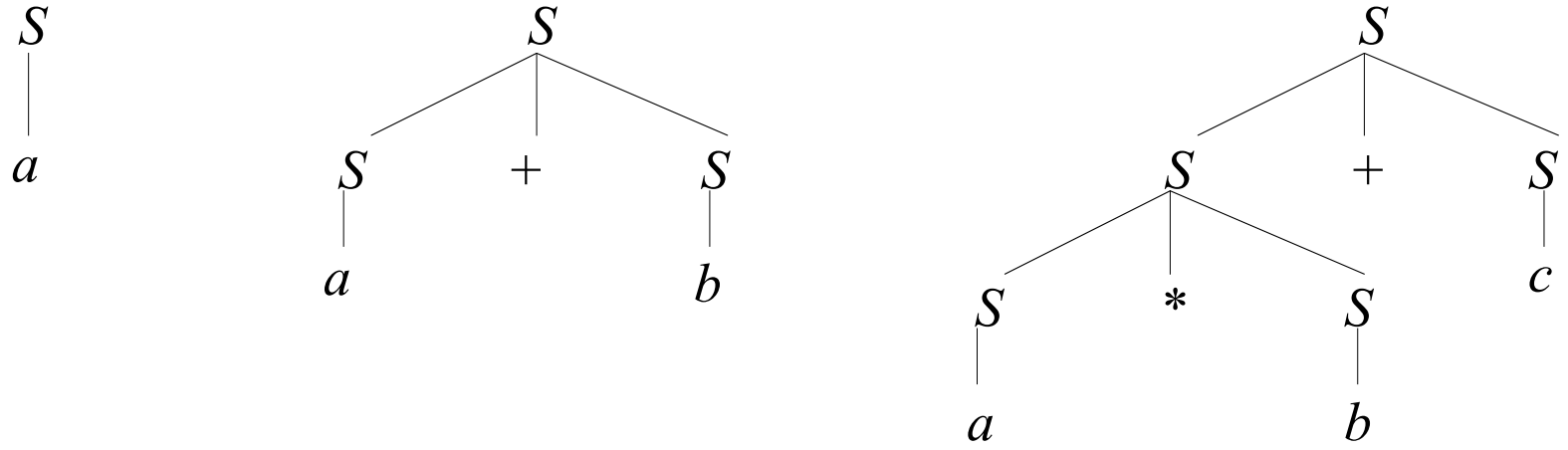
- We can substitute one A subtree for the other, any number of times
- That generates a parse tree for uv^iwx^iy for any i
- Therefore, for all i, $uv^iwx^iy \in L(G)$

Useful Trees

- If we can find a pumping parse tree, we can conclude that for all i, $uv^iwx^iy \in L(G)$
- And note that all these uv^iwx^iy are distinct, because v and x cannot both be ε

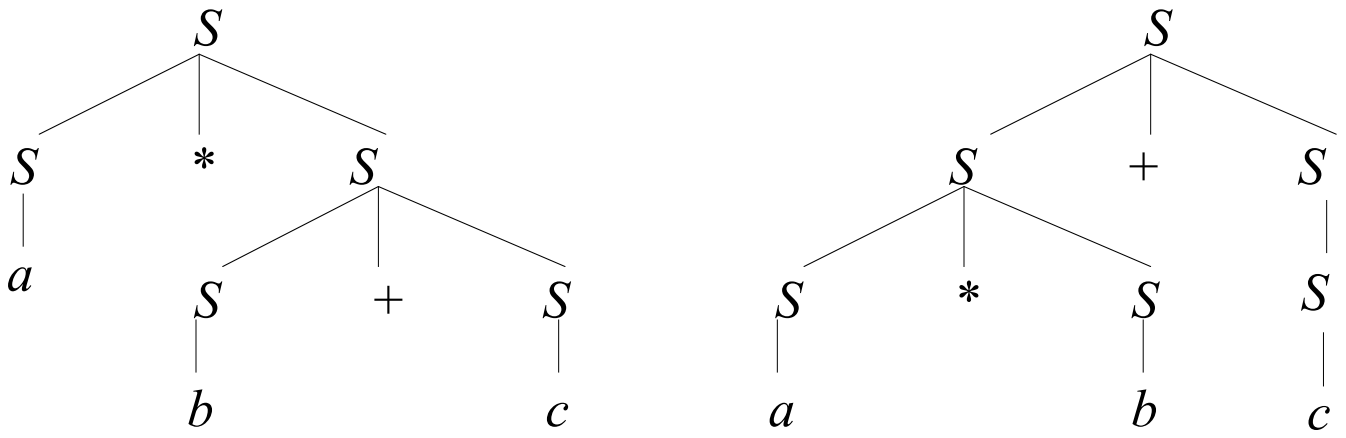
Height Of A Parse Tree

- The height of a parse tree is the number of edges in the *longest* path from the start symbol to any leaf
- For example: $S \rightarrow S \mid S{+}S \mid S{*}S \mid a \mid b \mid c$
- These are parse trees of heights 1, 2, and 3:



Minimum-Size Parse Trees

- A minimum-size parse tree for a string *x* in a grammar *G* is a parse tree that generates *x*, and has no more nodes than any other parse tree in *G* that generates *x*
- For example: $S \rightarrow S \mid S{+}S \mid S{*}S \mid a \mid b \mid c$
- Both these trees generate *a*b+c*, but the second one is not minimum size:



Lemma 14.1.2

Every CFG $G = (V, \Sigma, S, P)$ that generates an infinite language generates a pumping parse tree.

- Proof: let $G = (V, \Sigma, S, P)$ be any CFG, $L(G)$ infinite
- G generates infinitely many minimum-size parse trees, since each string in $L(G)$ has at least one
- Only finitely many can have height $|V|$ or less, so G generates a minimum-size parse tree of height $> |V|$
- Such a tree must be a pumping parse tree:
  - Property 1: it has a path with more than $|V|$ edges; some nonterminal A must occur at least twice on such a path
  - Property 2: replacing the ancestor A with the descendant A makes a tree with fewer nodes; this can't be a tree yielding the same string, because our tree was minimum-size

Outline

- 14.1 Pumping Parse Trees
- 14.2 The Language $\{a^n b^n c^n\}$
- 14.3 Closure Properties For CFLs
- 14.4 Non-Closure Properties
- 14.5 A Pumping Lemma
- 14.6 Pumping-Lemma Proofs
- 14.7 The Languages $\{xx\}$

Theorem 14.2

The language $\{a^n b^n c^n\}$ is not a CFL.

- Proof: let $G = (V, \Sigma, S, P)$ be a CFG with $\Sigma = \{a,b,c\}$
- Suppose by way of contradiction that $L(G) = \{a^n b^n c^n\}$ is context free.
- By Lemma 14.1.2, G generates a pumping parse tree
- By Lemma 14.1.1, for some k, $a^k b^k c^k = uvwxy$, where v and x are not both ε and uv^2wx^2y is in $L(G)$
- v and x must each contain only as, only bs, or only cs; otherwise uv^2wx^2y is not even in $L(a^*b^*c^*)$
- So uv^2wx^2y has more than k copies of one or two symbols, but only k of the third
- $uv^2wx^2y \notin \{a^n b^n c^n\}$; by contradiction, $L(G)$ is not context-free

The Insight

- There must be some string in $L(G)$ with a pumping parse tree: $a^k b^k c^k = uvwxy$
- But no matter how you break up $a^k b^k c^k$ into those substrings $uvwxy$ (where v and x are not both ε) you can show $uv^2wx^2y \notin \{a^n b^n c^n\}$
- Either:
  - v or x has more than one kind of symbol
  - v and x have at most one kind of symbol each

- If *v* or *x* has more than one kind of symbol:
  - uv^2wx^2y would have *a*s after *b*s and/or *b*s after *c*s
  - Not even in $L(a^*b^*c^*)$, so certainly not in $\{a^nb^nc^n\}$
  - Example:

$$\underbrace{a\ a\ a}_{u}\,\underbrace{a\ a\ b\ b}_{v}\,\underbrace{b}_{w}\,\underbrace{b\ b\ c}_{x}\,\underbrace{c\ c\ c\ c}_{y}$$

- If *v* and *x* have at most one kind each:
  - uv^2wx^2y has more of one or two, but not all three
  - Not in $\{a^nb^nc^n\}$
  - Example:

$$\underbrace{a\ a\ a}_{u}\,\underbrace{a\ a}_{v}\,\underbrace{b\ b\ b}_{w}\,\underbrace{b}_{x}\,\underbrace{b\ c\ c\ c\ c\ c}_{y}$$

Outline

- 14.1 Pumping Parse Trees
- 14.2 The Language $\{a^n b^n c^n\}$
- 14.3 Closure Properties For CFLs
- 14.4 Non-Closure Properties
- 14.5 A Pumping Lemma
- 14.6 Pumping-Lemma Proofs
- 14.7 The Languages $\{xx\}$

Closure Properties

- CFLs are closed for some of the same common operations as regular languages:
  - Union
  - Concatenation
  - Kleene star
  - Intersection *with a regular language*
- For the first three, we can make simple proofs using CFGs…

Theorem 14.3.1

If L_1 and L_2 are any context-free languages, $L_1 \cup L_2$ is also context free.

- Proof is by construction using CFGs
- Given $G_1 = (V_1, \Sigma_1, S_1, P_1)$ and $G_2 = (V_2, \Sigma_2, S_2, P_2)$, with $L(G_1) = L_1$ and $L(G_2) = L_2$
- Assume V_1 and V_2 are disjoint (without loss of generality, because symbols could be renamed)
- Construct $G = (V, \Sigma, S, P)$, where
  - $V = V_1 \cup V_2 \cup \{S\}$
  - $\Sigma = \Sigma_1 \cup \Sigma_2$
  - $P = P_1 \cup P_2 \cup \{(S \rightarrow S_1), (S \rightarrow S_2)\}$
- $L(G) = L_1 \cup L_2$, so $L_1 \cup L_2$ is a CFL

almost the same proof!

Theorem 14.3.2

If L_1 and L_2 are any context-free languages, L_1L_2 is also context free.

- Proof is by construction using CFGs
- Given $G_1 = (V_1, \Sigma_1, S_1, P_1)$ and $G_2 = (V_2, \Sigma_2, S_2, P_2)$, with $L(G_1) = L_1$ and $L(G_2) = L_2$
- Assume V_1 and V_2 are disjoint (without loss of generality, because symbols could be renamed)
- Construct $G = (V, \Sigma, S, P)$, where
  - $V = V_1 \cup V_2 \cup \{S\}$
  - $\Sigma = \Sigma_1 \cup \Sigma_2$
  - $P = P_1 \cup P_2 \cup \{(S \rightarrow S_1 S_2)\}$
- $L(G) = L_1L_2$, so L_1L_2 is a CFL

Kleene Closure

- The Kleene closure of any language L is $L^* = \{x_1 x_2 \ldots x_n \mid n \geq 0, \text{ with all } x_i \in L\}$
- This parallels our use of the Kleene star in regular expressions

Theorem 14.3.3

If L is any context-free language, L^* is also context free.

- Proof is by construction using CFGs
- Given $G = (V, \Sigma, S, P)$ with $L(G) = L$
- Construct $G' = (V', \Sigma, S', P')$, where
  - $V' = V \cup \{S'\}$
  - $P' = P \cup \{(S' \rightarrow SS'), (S' \rightarrow \varepsilon)\}$
- $L(G') = L^*$, so L^* is a CFL

Theorem 14.3.4

If L_1 is any context-free language and L_2 is any *regular* language, then $L_1 \cap L_2$ is context free.

- Proof sketch: by construction of a stack machine
- Given a stack machine M_1 for L_1 and an NFA M_2 for L_2
- Construct a new stack machine for $L_1 \cap L_2$
- A bit like the product construction:
  - If M_1's stack alphabet is Γ, and M_2's state set is Q, the new stack machine uses $\Gamma \times Q$ as its stack alphabet
  - It keeps track of both M_1's current stack and M_2's current state

Outline

- 14.1 Pumping Parse Trees
- 14.2 The Language $\{a^n b^n c^n\}$
- 14.3 Closure Properties For CFLs
- 14.4 Non-Closure Properties
- 14.5 A Pumping Lemma
- 14.6 Pumping-Lemma Proofs
- 14.7 The Languages $\{xx\}$

Non-Closure Properties

- As we just saw, CFLs have some of the same closure properties as regular languages
- But not all
- Not closed for intersection or complement…

Theorem 14.4.1

The CFLs are not closed for intersection.

- Proof: by counterexample
- Consider these CFGs:

$$S_1 \rightarrow A_1B_1 \qquad S_2 \rightarrow A_2B_2$$
$$A_1 \rightarrow aA_1b \mid \varepsilon \qquad A_2 \rightarrow aA_2 \mid \varepsilon$$
$$B_1 \rightarrow cB_1 \mid \varepsilon \qquad B_2 \rightarrow bB_2c \mid \varepsilon$$

- Now $L(G_1) = \{a^n b^n c^m\}$, while $L(G_2) = \{a^m b^n c^n\}$
- The intersection is $\{a^n b^n c^n\}$, which is not a CFL
- So the CFLs are not closed for intersection

Theorem 14.4.2

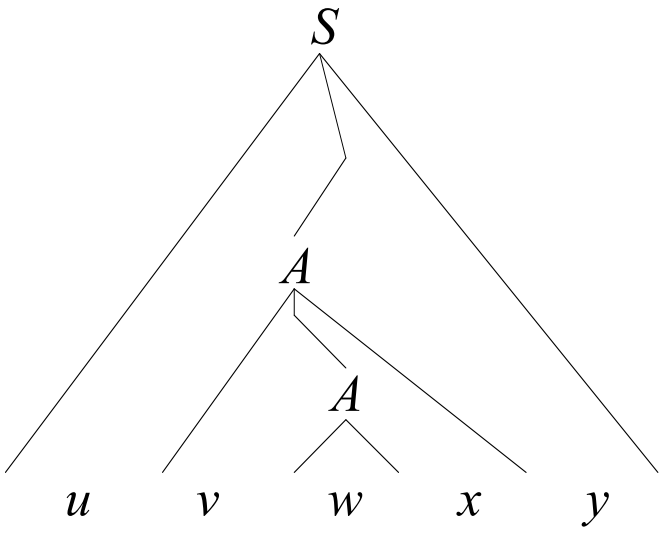
The CFLs are not closed for complement.

- Proof: by contradiction
- By Theorem 14.3.1, CFLs are closed for union
- Suppose by way of contradiction that they are also closed for complement
- By DeMorgan's laws we have $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$
- This defines intersection in terms of union and complement
- So CFLs are closed for intersection
- But this contradicts Theorem 14.4.1
- By contradiction, the CFLs are not closed for complement

Outline

- 14.1 Pumping Parse Trees
- 14.2 The Language $\{a^n b^n c^n\}$
- 14.3 Closure Properties For CFLs
- 14.4 Non-Closure Properties
- 14.5 A Pumping Lemma
- 14.6 Pumping-Lemma Proofs
- 14.7 The Languages $\{xx\}$

Pumping Parse Trees, Review



- A *pumping parse tree* for a CFG $G = (V, \Sigma, S, P)$ is a parse tree with two properties:
  1. There is a node for some nonterminal symbol *A,* which has that same nonterminal symbol *A* as one of its descendants
  2. The terminal string generated from the ancestor *A* is longer than the terminal string generated from the descendant *A*
- We proved that every grammar for an infinite language generates a pumping parse tree…

Lemma 14.1.2

Every CFG $G = (V, \Sigma, S, P)$ that generates an infinite language generates a pumping parse tree.

- Proof: let $G = (V, \Sigma, S, P)$ be any CFG, $L(G)$ infinite
- G generates infinitely many minimum-size parse trees, since each string in $L(G)$ has at least one
- Only finitely many can have height $|V|$ or less, so G generates a minimum-size parse tree of height $> |V|$
- Such a tree must be a pumping parse tree:
  - Property 1: it has a path with more than $|V|$ edges; some nonterminal A must occur at least twice on such a path
  - Property 2: replacing the ancestor A with the descendant A makes a tree with fewer nodes; this can't be a tree yielding the same string, because our tree was minimum-size

The Value of k

- just as in the case of regular languages we want to characterize a value of k that will force derivations in CFGs to reuse a non-terminal.
- we use the fact the a tree of height h with branching factor b has at most b^h leaf nodes.

lemma

For every CFG $G = (V, \Sigma, S, P)$ there exists some integer k greater than the length of any string generated by any parse tree of height $|V|+1$.

- Proof:
  - Let b be the length of the longest RHS of any production in P
  - Then b is the maximum branching factor in any tree
  - A tree of height $|V|+1$ can have at most $b^{|V|+1}$ leaves
  - Let $k = b^{|V|+1}+1$

The Value Of *k*

- The proof states that we can look at the structure of the grammar and always find a value for k such that a CFG will generate a pumping parsing tree.
- We'll use the fact that such a *k* exists in proofs; we won't need an actual value
- Just like the *k* in the pumping lemma for regular languages

Lemma 14.5.3: The Pumping Lemma for Context-Free Languages

For all context-free languages L there exists some $k \in \mathcal{N}$ such that for all $z \in L$ with $|z| \geq k$, there exist $uvwxy$ such that:

1. $z = uvwxy$,
2. v and x are not both ε,
3. $|vwx| \leq k$, and
4. for all i, $uv^iwx^iy \in L$.

- L is a CFL, so there is some CFG G with $L(G) = L$
- Let k be as given for G by Lemma 14.5.2
- We are then given some $z \in L$ with $|z| \geq k$
- Consider any minimum-size parse tree for z
- It has height > $|V|+1$, so our Lemma applies
- This is a parse tree for z and it is a pumping parse tree

Matching Pairs

- The pumping lemma shows again how matching pairs are fundamental to CFLs
- Every sufficiently long string in a CFL contains a matching pair of substrings (the *v* and *x* of the lemma)
- These can be pumped in tandem, always producing another string uv^iwx^iy in the language

Outline

- 14.1 Pumping Parse Trees
- 14.2 The Language $\{a^n b^n c^n\}$
- 14.3 Closure Properties For CFLs
- 14.4 Non-Closure Properties
- 14.5 A Pumping Lemma
- 14.6 Pumping-Lemma Proofs
- 14.7 The Languages $\{xx\}$

Pumping-Lemma Proofs

- The pumping lemma is very useful for proving that languages are not context free
- For example, $\{a^n b^n c^n\}$…

$\{a^n b^n c^n\}$ Is Not Context Free

1. Proof is by contradiction using the pumping lemma for context-free languages. Assume that $L = \{a^n b^n c^n\}$ is context free, so the pumping lemma holds for L. Let k be as given by the pumping lemma.
2. Choose $z = a^k b^k c^k$. Now $z \in L$ and $|z| \geq k$ as required.
3. Let u, v, w, x, and y be as given by the pumping lemma, so that $uvwxy = a^k b^k c^k$, v and x are not both ε, $|vwx| \leq k$, and for all i, $uv^i wx^i y \in L$.
4. Now consider pumping with $i = 2$. The substrings v and x cannot contain more than one kind of symbol each—otherwise the string uv^2wx^2y would not even be in $L(a^*b^*c^*)$. So the substrings v and x must fall within the string $a^k b^k c^k$ in one of these ways…

$\{a^n b^n c^n\}$, Continued

	a^k	b^k	c^k
1.	v x		
2.	v	x	
3.		v x	
4.		v	x
5.			v x
6.	v		x

But in all these cases, since v and x are not both ε, pumping changes the number of one or two of the symbols, but not all three. So $uv^2wx^2y \notin L$.

5 This contradicts the pumping lemma. By contradiction, $L = \{a^n b^n c^n\}$ is not context free.

$\{a^n b^n c^n\}$, Revisited

	a^k	b^k	c^k
1.	v x		
2.	v	x	
3.		v x	
4.		v	x
5.			v x
6.	v		x

- Case 6 would be a contradiction for another reason: $|vwx| > k$
- We can rule out such cases…