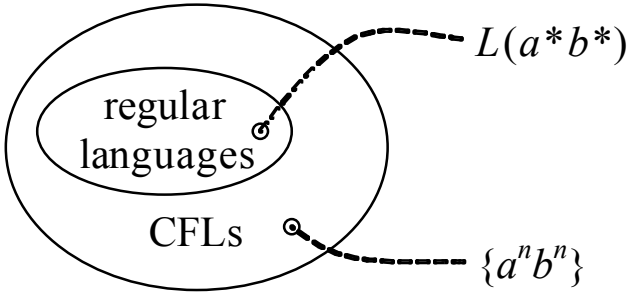
Chapter Fourteen:
The Context-Free Frontier

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. If regular languages are a safe and settled territory, context-free languages are more like frontier towns. Some operations like union get you safely to another context-free language; others like complement and intersection just leave you in the wilderness.

Outline

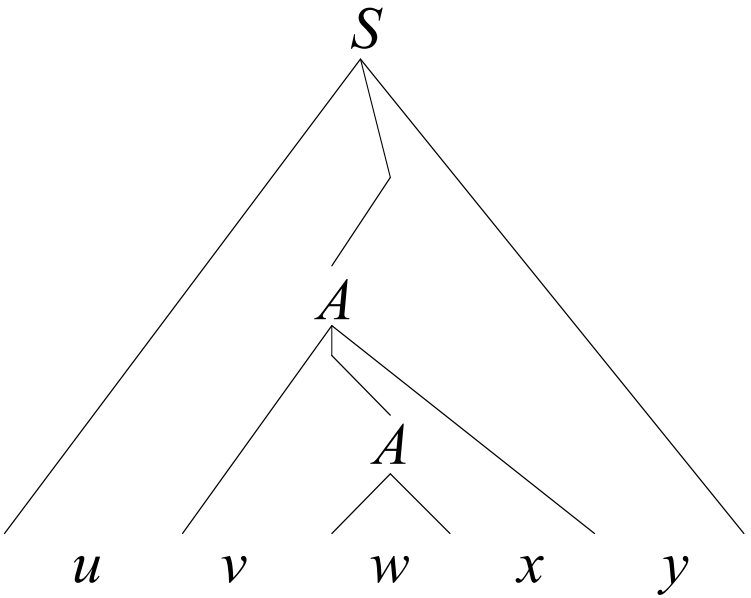
- 14.1 Pumping Parse Trees
- 14.2 The Language $\{a^n b^n c^n\}$
- 14.3 Closure Properties For CFLs
- 14.4 Non-Closure Properties
- 14.5 A Pumping Lemma
- 14.6 Pumping-Lemma Proofs
- 14.7 The Languages $\{xx\}$

Pumping Parse Trees

- 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
  – 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…

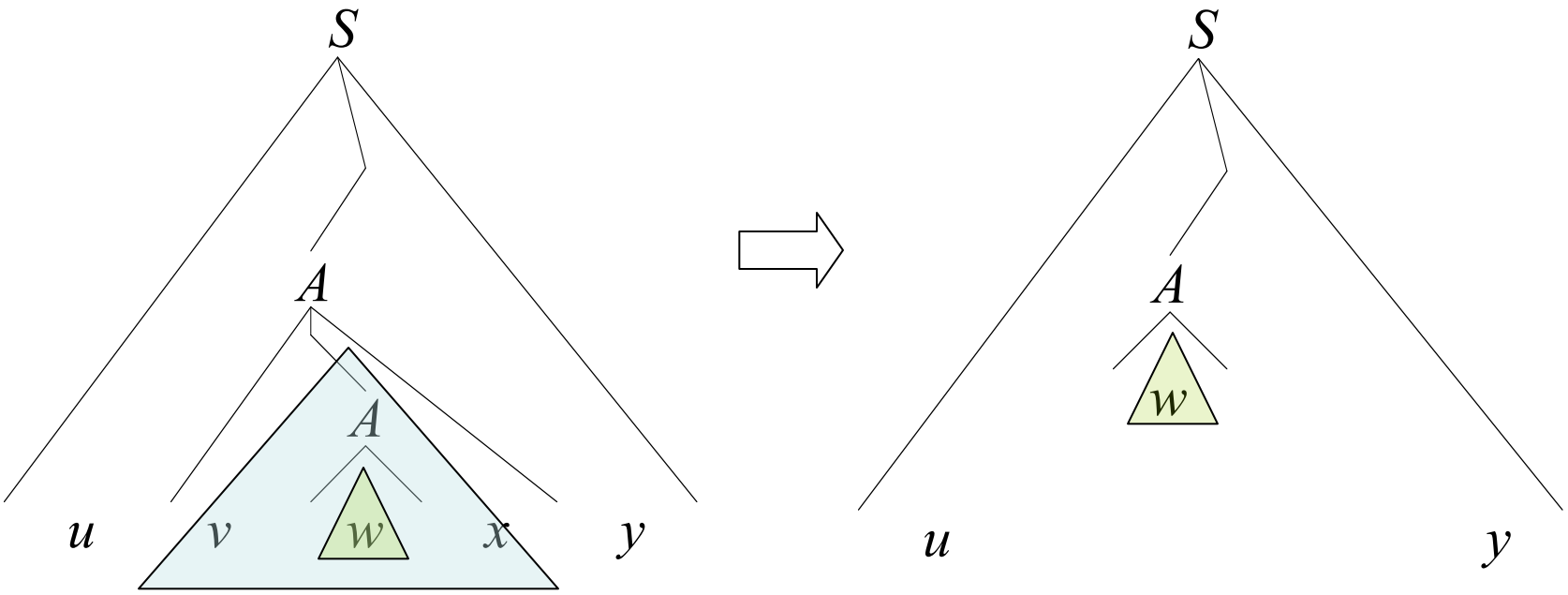
Lemma 14.1.1

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



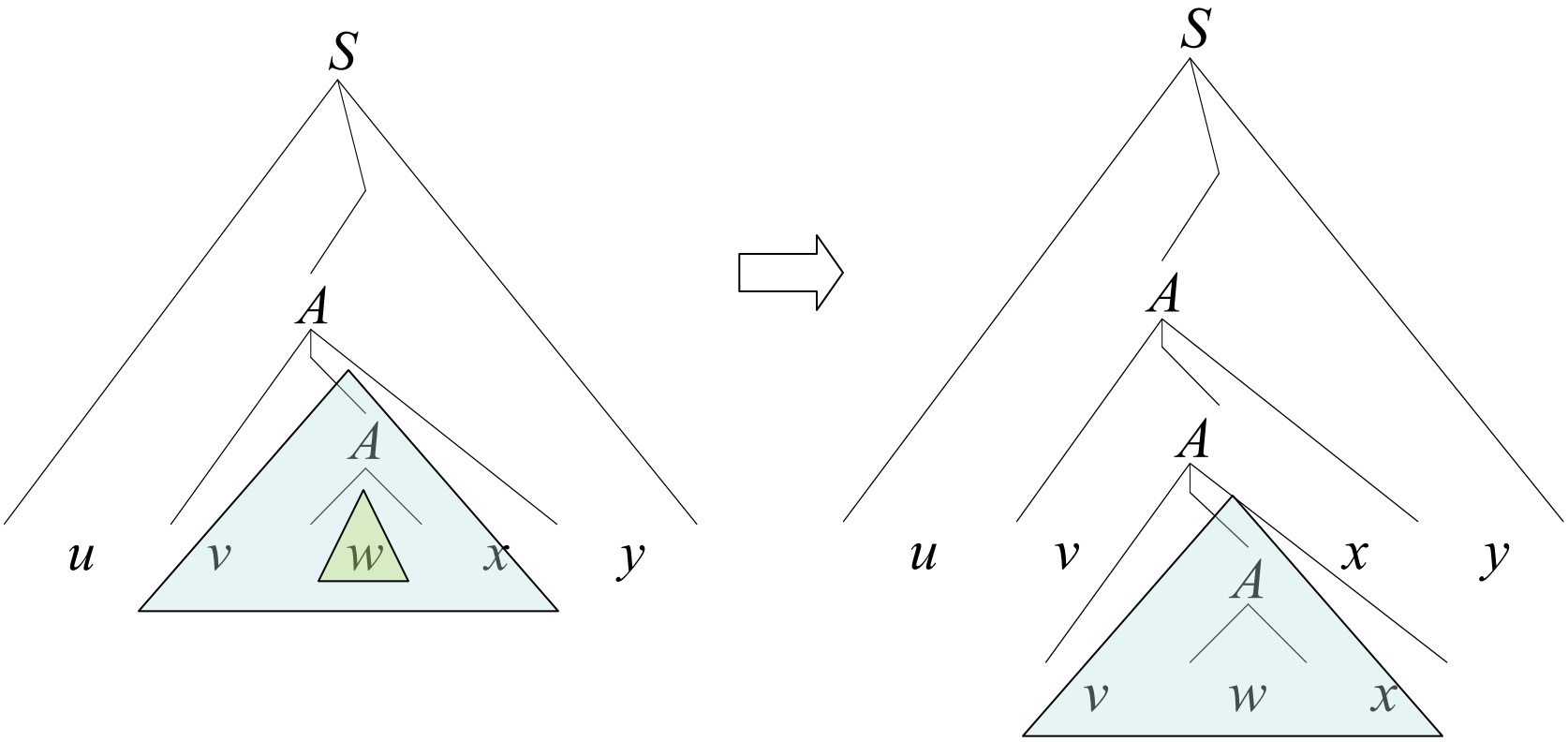
- As shown:
  - *uvwxy* is the whole derived string
  - *A* is the nonterminal that is its own descendant
  - *vwx* is the string derived from the ancestor *A*
  - *w* is the string derived from the descendant
  - |*vwx*| > |*w*|, so *v* and *x* are not both ε
- There are two subtrees rooted at *A*
-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

Cut And Paste, $i = 0$



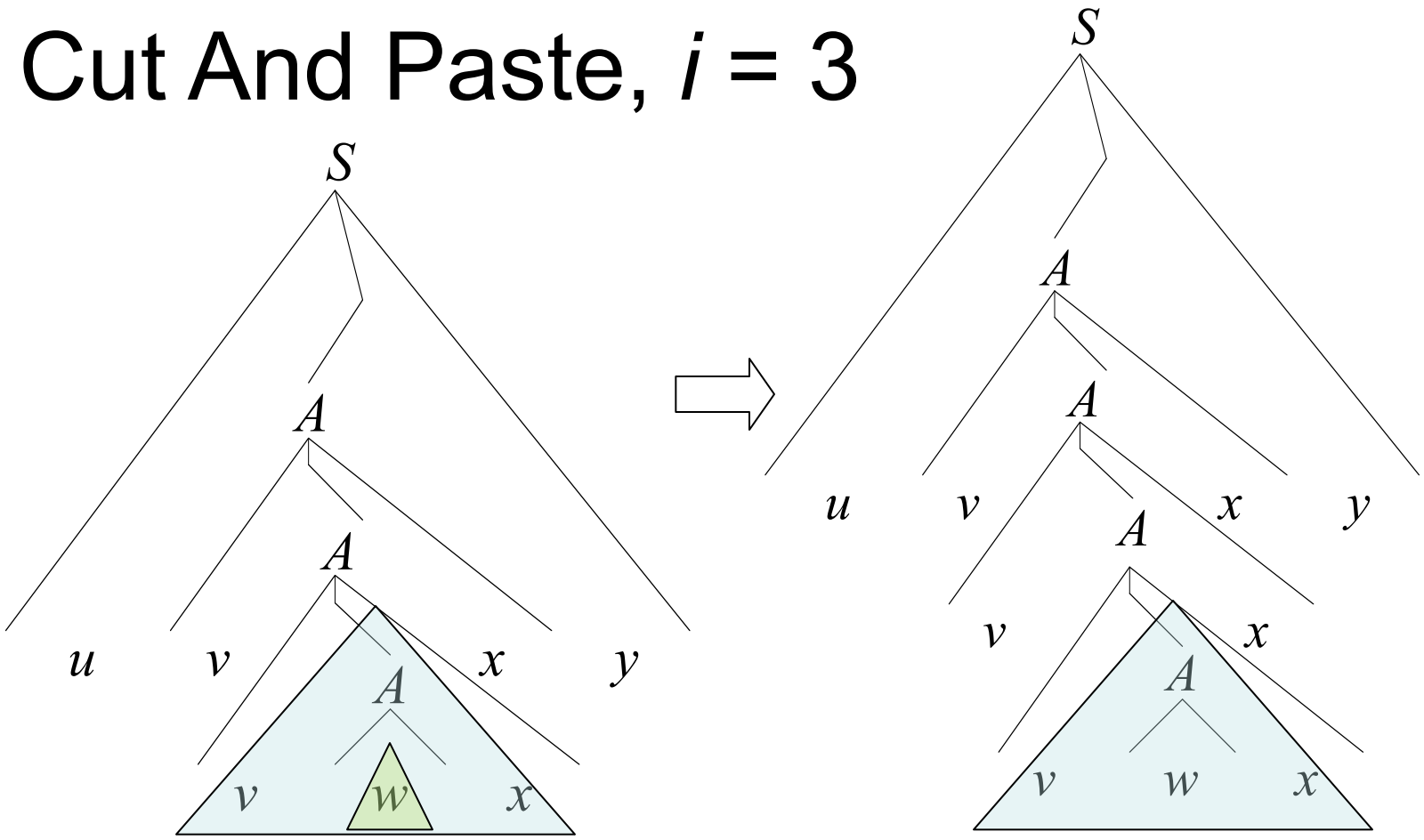
- We can replace the *vwx* subtree with the *w* subtree
- That makes a parse tree for *uwy*
- That is, uv^iwx^iy for $i = 0$

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 $i = 2$

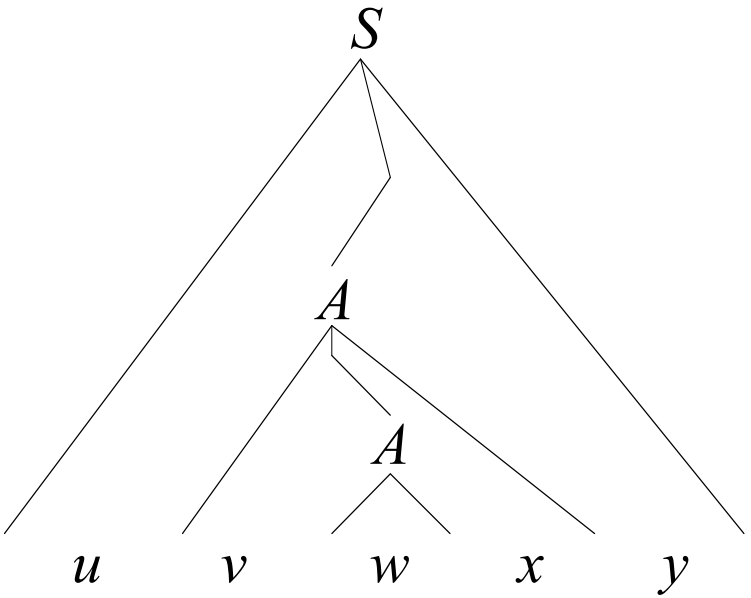
Cut And Paste, *i* = 3



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

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.



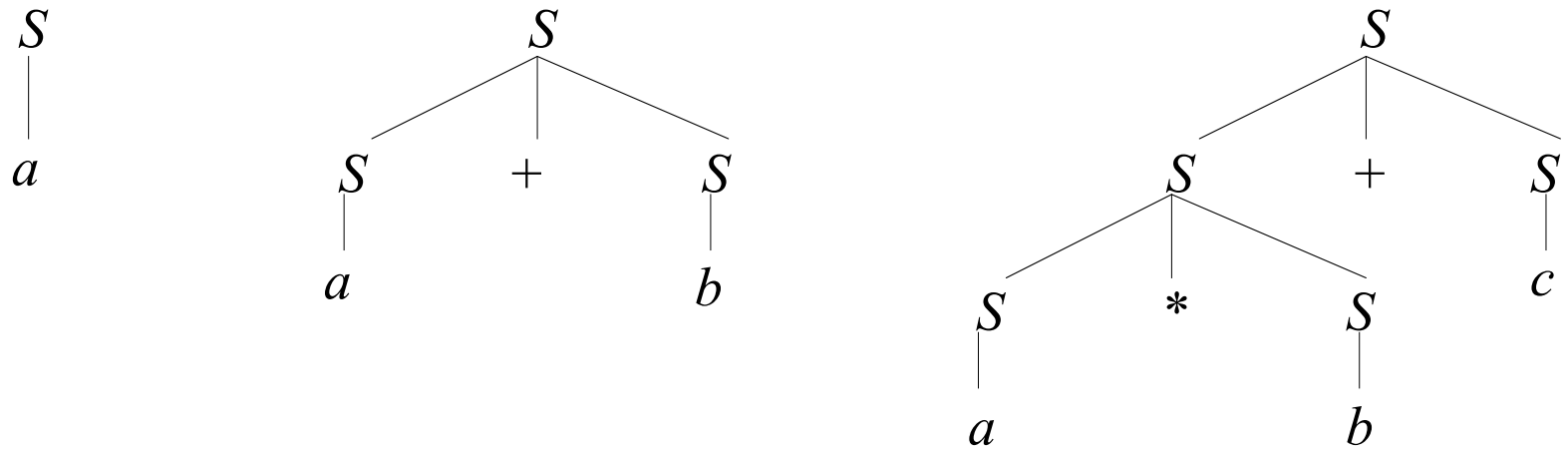
- We can substitute one A subtree for the other, any number of times
- That generates a parse tree for uv^iwx^iy for any i
- Therefore, for all i, $uv^iwx^iy \in L(G)$

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 are not both ε
- The next lemma shows that pumping parse trees are not at all hard to find

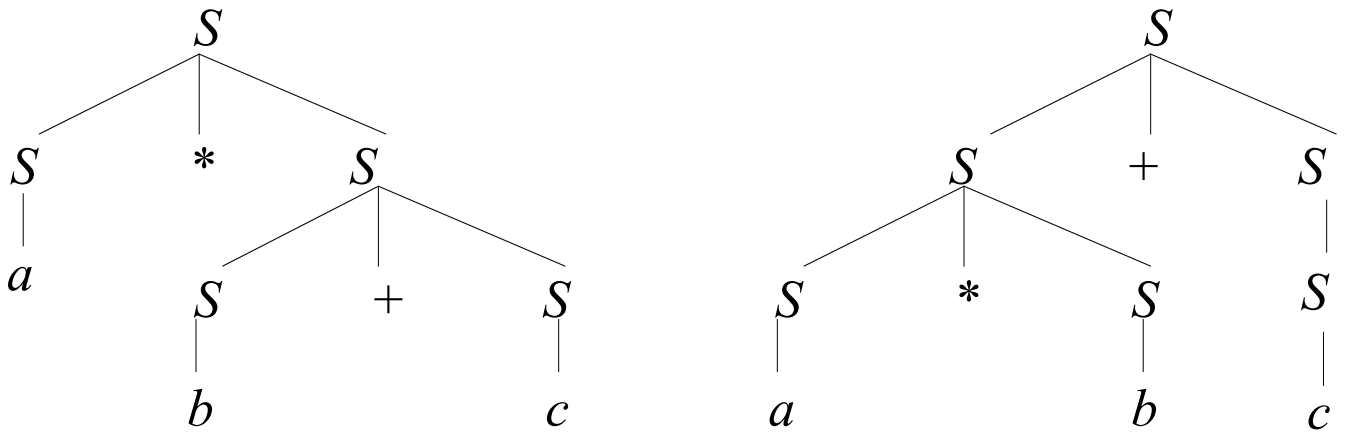
Height Of A Parse Tree

- 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 any CFG, $\Sigma = \{a,b,c\}$
- Suppose by way of contradiction that $L(G) = \{a^n b^n c^n\}$
- 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 or one or two symbols, but only k of the third
- $uv^2wx^2y \notin \{a^n b^n c^n\}$; by contradiction, $L(G) \neq \{a^n b^n c^n\}$

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
  - $V = V_1 \cup V_2 \cup \{S\}$
  - $\Sigma = \Sigma_1 \cup \Sigma_2$
  - $P = P_1 \cup P_2 \cup \{(S \rightarrow S_1S_2)\}$
- $L(G) = L_1L_2$, 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

Non-Closure

- This does *not* mean that every intersection of CFLs fails to be a CFL
- Often, an intersection of CFLs is a CFL
- Just not always!
- Similarly, the complement of a CFL is sometimes, but not always, another CFL…

Theorem 14.4.2

The CFLs are not closed for complement.

- Proof 1: 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 close for intersection
- But this contradicts Theorem 14.4.1
- By contradiction, the CFLs are not closed for complement

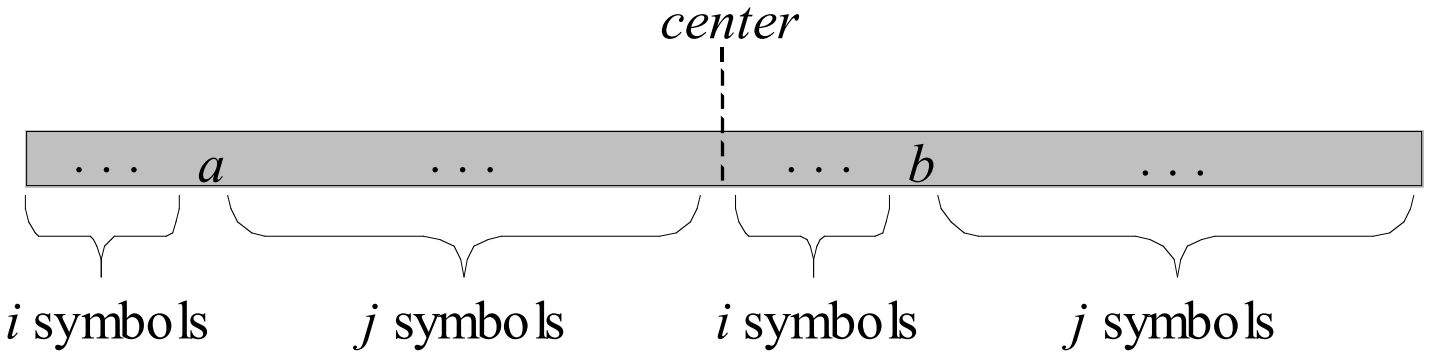
Theorem 14.4.2

The CFLs are not closed for complement.

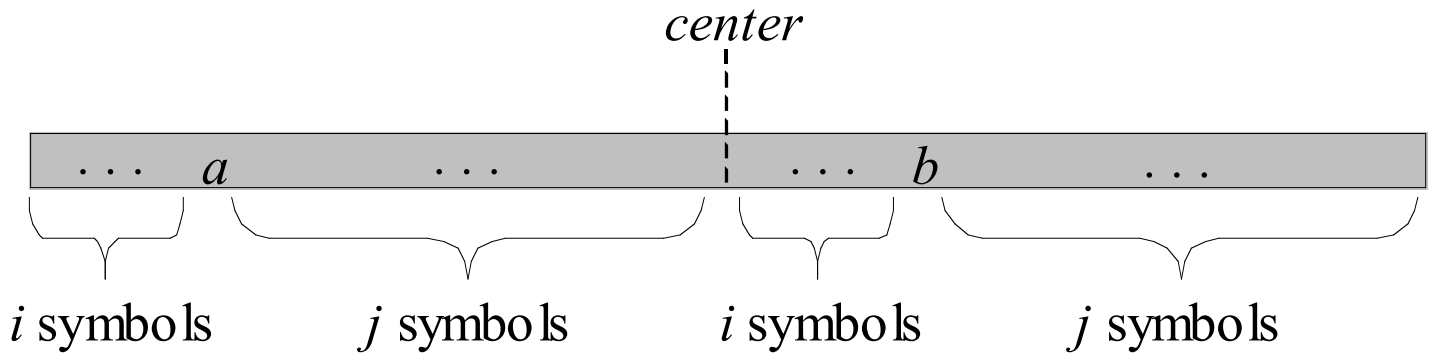
- Proof 2: by counterexample
- Let L be the non-CFL $\{xx \mid x \in \{a,b\}^*\}$
- We will show that $\overline{L} = \{x \in \{a,b\}^* \mid x \notin L\}$ *is* a CFL (next slide)
- Thus we have a language $\overline{L}$ that is a CFL, and its complement $L = \overline{\overline{L}}$ is not a CFL
- So the CFLs are not closed for complement

$\{x \in \{a,b\}^* \mid x \neq ss \text{ for any } s\}$

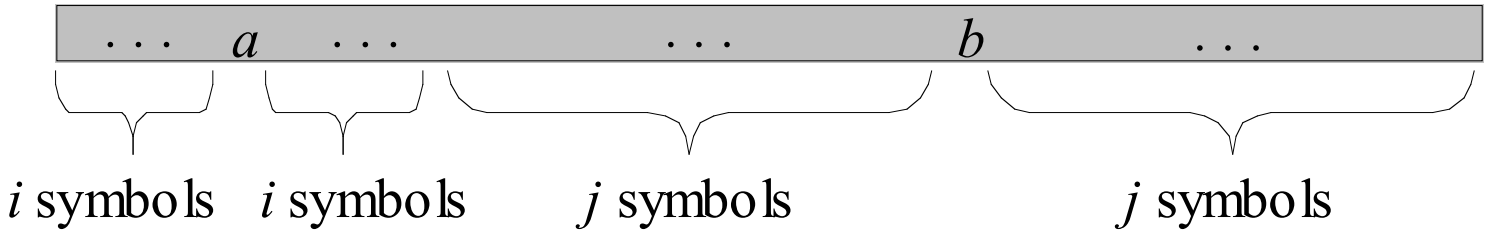
- The language includes:
  - All odd-length strings
  - And all even-length strings with *a* somewhere in the first half, but a corresponding *b* in the second:



  - And all even-length strings with *b* somewhere in the first half, but a corresponding *a* in the second



- *waxybz*, where |*w*| = |*y*| = *i* and |*x*| = |*z*| = *j*
- Since the *x* and *y* parts can be any strings, we can swap them in the picture:



- This is {*way* | |*w*| = |*y*|}, concatenated with {*xbz* | |*x*| = |*z*|}

$\{x \in \{a,b\}^* \mid x \neq ss \text{ for any } s\}$

- So this is a union of three sets:
  - $\{x \in \{a,b\}^* \mid |x| \text{ is odd}\}$
  - $\{way \mid |w| = |y|\}$ concatenated with $\{xbz \mid |x| = |z|\}$
  - $\{xbz \mid |x| = |z|\}$ concatenated with $\{way \mid |w| = |y|\}$
- This CFG generates the language:

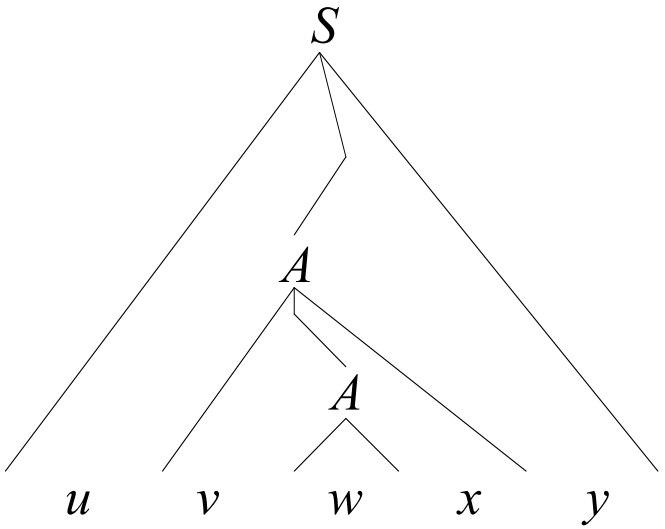
$$
\begin{aligned}
S &\rightarrow O \mid AB \mid BA \\
A &\rightarrow XAX \mid a \\
B &\rightarrow XBX \mid b \\
O &\rightarrow XXO \mid X \\
X &\rightarrow a \mid b
\end{aligned}
$$

- It is a CFL

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 $\{xx\}$

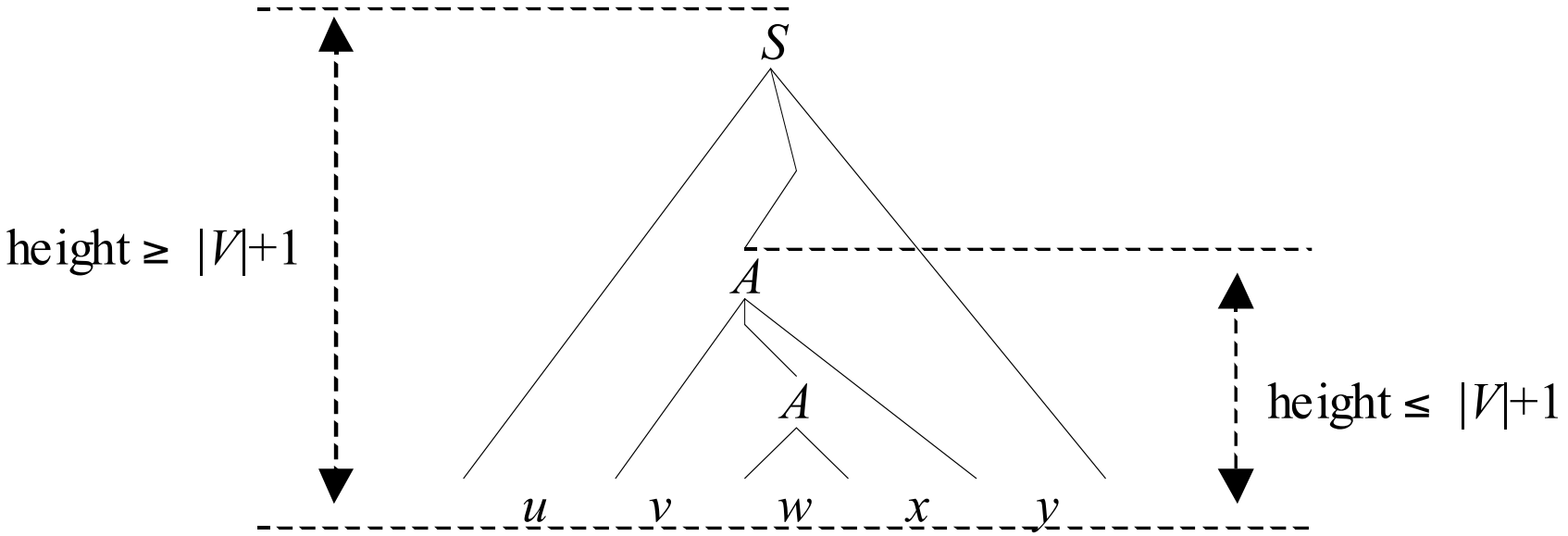
Pumping Parse Trees, Review

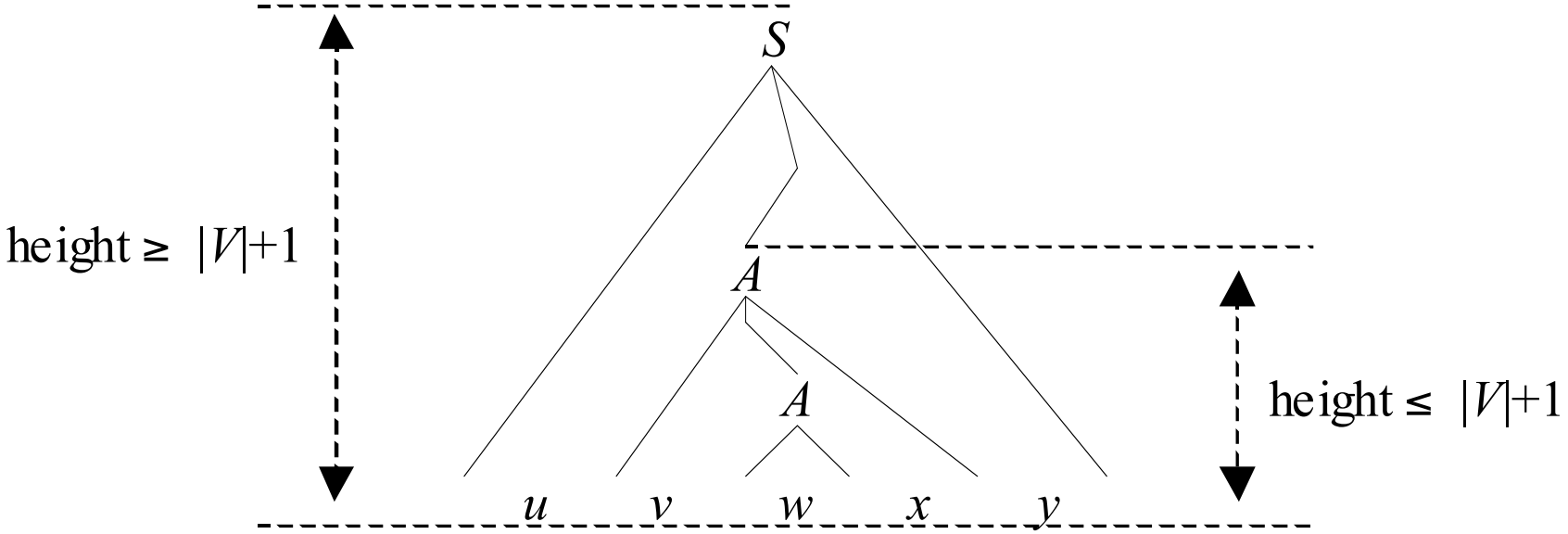


- A *pumping parse tree* for a CFG $G = (V, \Sigma, S, P)$ is a parse tree with two properties:
  1. There is a node for some nonterminal symbol *A,* which has that same nonterminal symbol *A* as one of its descendants
  – The terminal string generated from the ancestor *A* is longer than the terminal string generated from the descendant *A*
- We proved that every grammar for an infinite language generates a pumping parse tree
- To make a general-purpose pumping lemma, we need to be more specific about those *A*s…

Lemma 14.5.1

For every grammar $G = (V, \Sigma, S, P)$, every minimum-size parse tree of height greater than $|V|$ can be expressed as a pumping parse tree with the properties shown:





- Choose any path from root to leaf with > |V| edges
- Working from leaf back to root along that path, choose the first two nodes that repeat some *A*
- As in Lemma 27.1.2, this is a pumping parse tree
- Some nonterminal must have repeated within the first |*V*|+1 edges from the leaf, the height of the subtree generating *vwx* is ≤ |*V*|+1

Bounds

- Previous lemma says that a subtree where some nonterminal A is its own descendant can be found near the fringe
- In other words, we have bounds on the height of that subtree
- That lets us bound the length of the string *vwx* generated by that subtree…

Lemma 14.5.2

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 or subtree of height $|V|+1$ or less.

- Proof 1:
  - There are only finitely many trees of height $|V|+1$ or less
  - Let k be the length of the longest string generated, plus one
- Proof 2:
  - 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*

- Our two proofs gave two different values for *k*
- That doesn't matter
- For any grammar *G* there is a bound *k* on the yield of a tree or subtree of height ≤ |*V*|+1
- 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, $uv^iwx^iy \in A$.

- 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 Lemma 14.5.1 applies
- This is a parse tree for z (property 1), it is a pumping parse tree (properties 2 and 4), and the subtree generating vwx has height ≤ $|V|+1$ (property 3)

Pumping Lemma Structure

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, $uv^iwx^iy \in A$.

- As with the pumping lemma for regular languages, this has alternating "for all" and "there exist" clauses:
  1. ∀ L …
  2. ∃ k …
  3. ∀ z …
  4. ∃ $uvwxy$ …
  5. ∀ i …
- Our proof showed how to construct the ∃ parts
- Now we'll forget about the construction, and only use the ∃

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
- (One may be empty—then the other can be pumped alone, as in the pumping lemma for 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 $\{xx\}$

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 $uv^2wx^2y \notin L$.

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

The Game

- The alternating ∀ and ∃ clauses of the pumping lemma make these proofs a kind of game
- The ∃ parts (*k* and *uvwxy*) are the pumping lemma's moves: these values exist, but are not ours to choose
- The ∀ parts (*L*, *z*, and *i*) are our moves: the lemma holds for all proper values, so we have free choice
- We make our moves strategically, to force a contradiction
- No matter what the pumping lemma does with its moves, we want to end up with some $uv^iwx^iy \notin L$
- We have fewer choices than with the pumping lemma for regular languages, and the opponent has more
- That makes these proofs a little harder

$\{a^n b^n c^n\}$, 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…

Theorem 14.6

The language $\{a^n b^m c^n \mid m \leq n\}$ is not context free.

- Proof: by contradiction using the pumping lemma
- Assume $L = \{a^n b^m c^n \mid m \leq n\}$ is a CFL
- Let k be as given by the pumping lemma
- Choose $z = a^k b^k c^k$, so we have $z \in L$ and $|z| \geq k$
- Let u, v, w, x, and y be as given by the lemma
- Now $uvwxy = a^k b^k c^k$, v and x are not both ε, $|vwx| \leq k$, and for all i, $uv^i wx^i y \in L$
- Now consider pumping with $i = 2$
- v and x cannot contain more than one kind of symbol each; otherwise $uv^2wx^2y \notin L(a^*b^*c^*)$
- That leaves 6 cases…

	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 cases 1-5 have $uv^2wx^2y \notin L$:
  - Case 1 has more *a*s than *c*s
  - Case 2 has more *a*s than *c*s, or more *b*s than *c*s, or both
  - Case 3 has more *b*s than *a*s and more *b*s than *c*s
  - Case 4 has more *b*s than *a*s, or more *c*s than *a*s, or both
  - Case 5 has more *c*s than *a*s and more *c*s than *b*s
- And case 6 contradicts $|vwx| \le k$
- By contradiction, $L = \{a^n b^m c^n \mid m \le n\}$ is not a CFL

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 $\{xx\}$

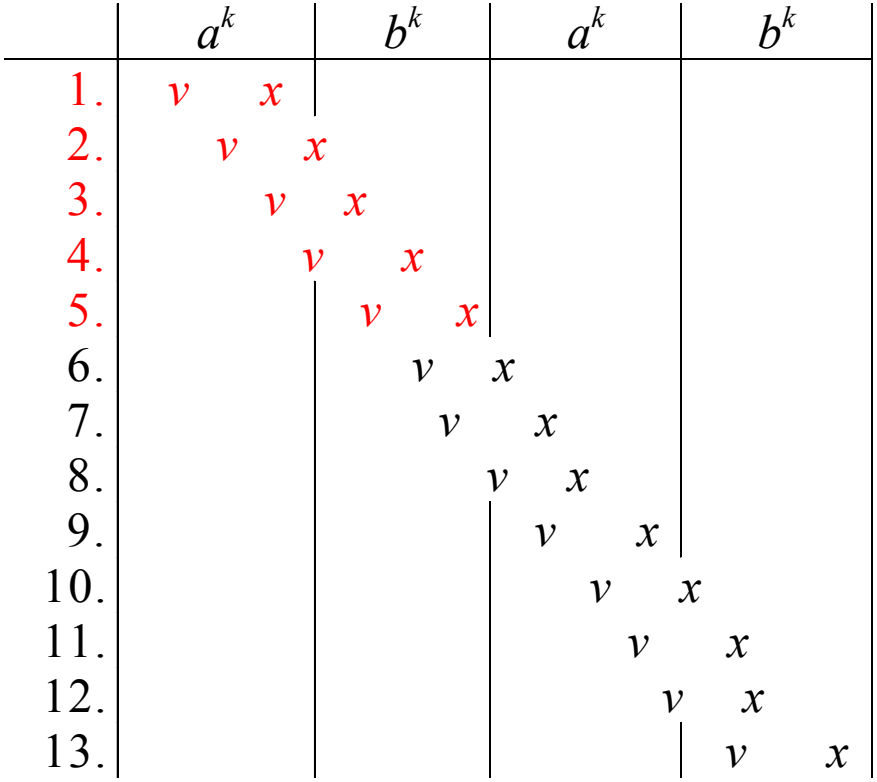
The Languages {*xx*}

- {*xx* | *x* ∈ Σ*}: strings that consist of any string over Σ followed by a copy of the same string
- For Σ = {*a*,*b*}, that includes strings ε, *aa, bb, abab, baba, aaaa, bbbb*, and so on
- We saw that the languages {xx^R} are context free, though not regular for any alphabet with at least two symbols
- Now, about {*xx*}…

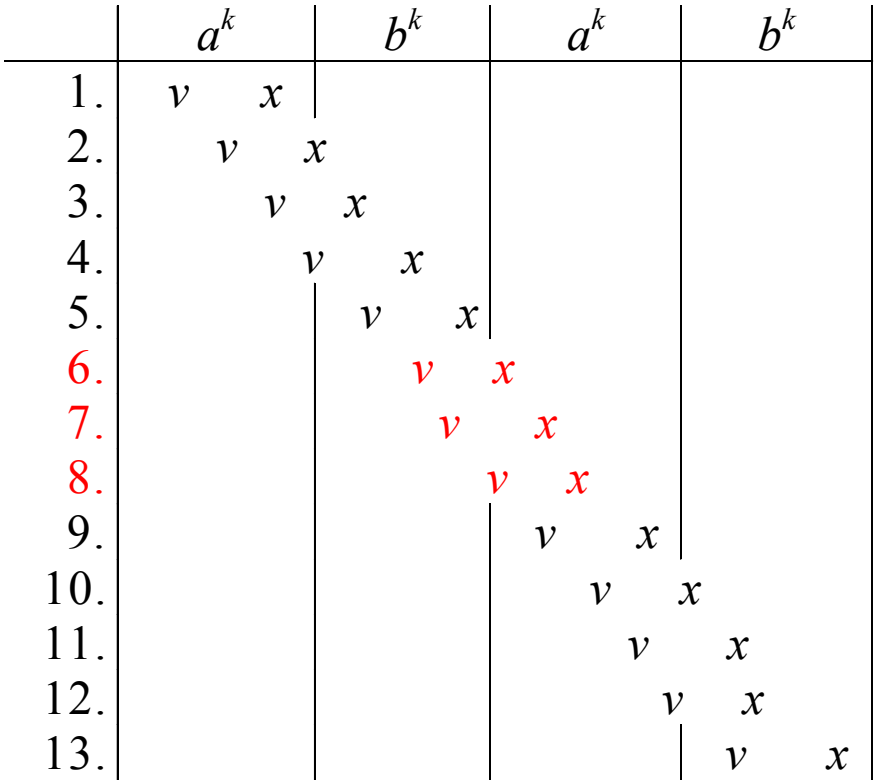
Theorem 14.7

$\{xx \mid x \in \Sigma^*\}$ is not a CFL when $|\Sigma| \geq 2$.

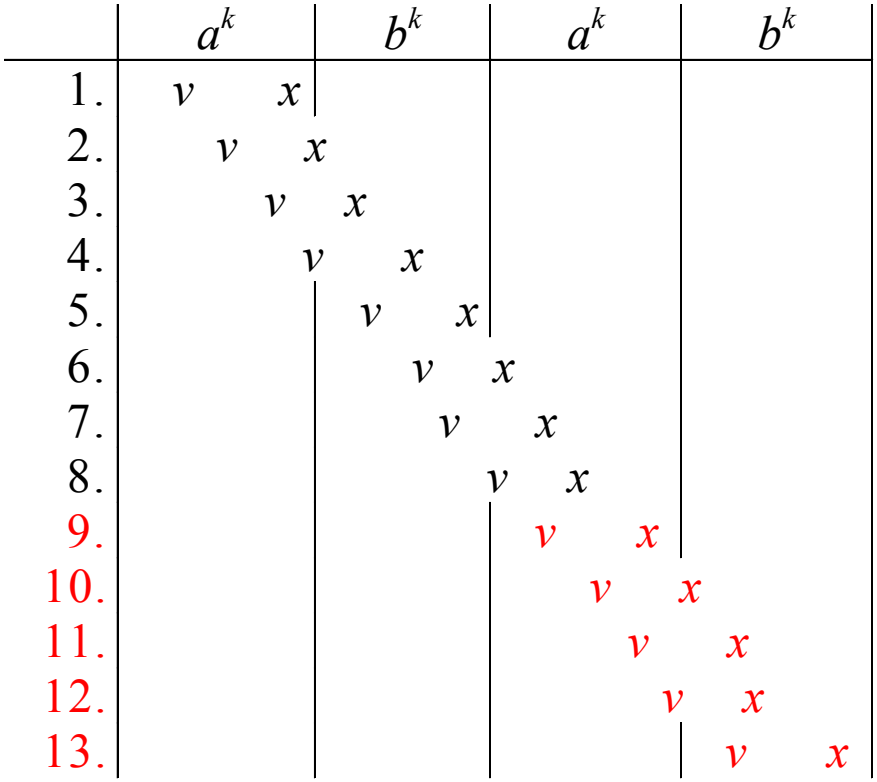
- Proof: by contradiction using the pumping lemma
- Let Σ be any set of at least two symbols, *a* and *b*
- Assume $L = \{xx \mid x \in \Sigma^*\}$ is a CFL
- Let *k* be as given by the pumping lemma
- Choose $z = a^k b^k a^k b^k$, so we have $z \in L$ and $|z| \geq k$
- Let *u*, *v, w, x,* and *y* be as given by the lemma
- Now $uvwxy = a^k b^k a^k b^k$, *v* and *x* are not both ε, $|vwx| \leq k$, and for all *i*, $uv^i wx^i y \in L$
- Consider how the substrings *v* and *x* fall within *z*
- Since $|vwx| \leq k$, *v* and *x* cannot be widely separated
- That leaves 13 cases…



- For cases 1-5, choose i=0
  - Then uv^0wx^0y is some sa^kb^k where $|s| < 2k$
  - The last symbol of the first half is an a, but the last symbol of the second half is a b
  - So $uv^0wx^0y \notin L$



- For cases 6-8, choose i=0
  - Then uv^0wx^0y is some a^ksb^k where $|s| < 2k$
  - This can't be rr for any string r; because if r starts with k as and ends with k bs, we must have $|r| \geq 2k$ and so $|rr| \geq 4k$, while our $|a^ksb^k| < 4k$
  - So $uv^0wx^0y \notin L$



- For cases 9-13, choose i=0
  - Then uv^0wx^0y is some a^kb^ks where $|s| < 2k$
  - The first symbol of the first half is an *a*, but the first symbol of the second half is a *b*
  - So $uv^0wx^0y \notin L$
- We have a contradiction in every case, so *L* is not a CFL

Choice Of *i*

- We ended up using the same value of *i* in each of the 13 cases above
- We could have selected a different value of *i* for each case
- Sometimes, to get a contradiction, your choice of *i* must depend on the *uvwxy* chosen by the lemma
- In the pumping-lemma proof game, your move (choice of *i*) can depend on your opponent's previous move (choice of *uvwxy*)

{*xx*} In Programming Languages

- Many languages require variables to be declared before they are used:

```
int fred = 0;
while (fred==0) {
    ...
}
```

- The same name must occur in two places
- This is a non-context-free construct in the same way that {*xx* | *x* ∈ Σ*} is a non-context-free language
- Can't be wired into a grammar for the language
- Enforced after parsing