



Grammars and Their Languages

Robert M. Keller
Harvey Mudd College
22 September 2003



Review of Grammars

- We assume some familiarity with grammars from CS 60.
- The grammars studied there were of a specialized type, known as “context-free” grammars.
- Here we will present a more general form of grammar, of which the context-free will be a special case.



Generality of Grammars

- For now, we will concentrate on string grammars, grammars for generating languages that are sets of strings.
- Other kinds of grammars can be used to generate graphs, trees, . . .
- There are other similar formal systems for generating languages that are not grammars. An example is “L systems”.



Definition of Grammars

- A grammar consists of 4 parts:
 - Terminal alphabet Σ
 - Auxiliary alphabet
 - Productions
 - Start symbol S $\in \Sigma$



Definition of Grammars

- A grammar consists of 4 parts:
 - Terminal alphabet Σ
 - Auxiliary alphabet A (such that $A \cap \Sigma = \emptyset$)
 - A finite set of productions P
 - Start symbol $S \in A$
- Each production has the form $x \rightarrow y$,
where $x \in (\Sigma \cup A)^+$, $y \in (\Sigma \cup A)^*$.
- Note: The book calls auxiliaries “variables”.
Elsewhere, they are also called “non-terminals”.



Derivation in a Grammar

- A derivation in of a grammar is a sequence of strings $x_0 \sqsupset x_1 \sqsupset x_2 \sqsupset \dots$ each in $(\sqsupset \sqsupset A)^*$ where:
 - $x_0 = S$, the start symbol
 - For each i , $x_i \sqsupset x_{i+1}$ provided that there are string $u, v, x_i = u, v, v', w$ such that
 - $x_i = uvw$,
 - $x_{i+1} = uv'w$,
 - $v \sqsupset v'$ is a production
- The **language generated by a grammar** is the set of strings $x \sqsupset \sqsupset^*$ such that there is a derivation that ends with x .



Example of a Grammar

- Productions:

- $S \rightarrow ab$
- $S \rightarrow aSb$
- $S \rightarrow SS$

- Example derivations of strings in the language:

- $S \rightarrow ab$
- $S \rightarrow aSb \rightarrow aabb$
- $S \rightarrow aSb \rightarrow aaSbb \rightarrow aaabbb$
- $S \rightarrow SS \rightarrow abS \rightarrow abab$
- $S \rightarrow SS \rightarrow SSS \rightarrow ababab$
- $S \rightarrow SS \rightarrow aSbS \rightarrow aabbS \rightarrow aabbaSb \rightarrow aabbaabb$



Example:

Grammar for Additive Arithmetic Expressions

- The start symbol is A .
- The terminals are $\{a, b, c, +\}$.
- The productions are:
 - $A \rightarrow V$
 - $A \rightarrow V + A$
 - $V \rightarrow a$
 - $V \rightarrow b$
 - $V \rightarrow c$
- Sample derivations:
 1. $A \rightarrow V \rightarrow a$
 2. $A \rightarrow V \rightarrow c$
 3. $A \rightarrow V + A \rightarrow c + A \rightarrow c + V \rightarrow c + a$
 4. $A \rightarrow V + A \rightarrow c + A \rightarrow c + V + A \rightarrow c + b + A \rightarrow c + b + V \rightarrow c + b + a$

Another Example of a Grammar for the Language $\{a^n b^n c^n \mid n \geq 0, n > 0\}$

□ The grammar:

- Terminal alphabet $\Sigma = \{a, b, c\}$
- Auxiliary alphabet $\{S, A, B, C, F\}$
- Productions Π :

- $S \rightarrow FE$ $E \rightarrow ABCE$ $E \rightarrow \epsilon$
- $BA \rightarrow AB$ $CA \rightarrow AC$ $CB \rightarrow BC$
- $FA \rightarrow a$ $aA \rightarrow aa$ $aB \rightarrow ab$
- $bB \rightarrow bb$ $bC \rightarrow bc$ $cC \rightarrow cc$

- Start symbol S

- A derivation (underlines show symbols replaced):
 $S \rightarrow \underline{F}E \rightarrow \underline{F}A\underline{BCE} \rightarrow \underline{F}A\underline{B}C\underline{A}B\underline{C}E \rightarrow \underline{F}A\underline{B}C\underline{A}B\underline{C} \rightarrow$
 $\underline{F}A\underline{B}A\underline{C}B\underline{C} \rightarrow \underline{F}A\underline{A}B\underline{C}B\underline{C} \rightarrow \underline{F}A\underline{A}B\underline{B}C\underline{C} \rightarrow \underline{a}A\underline{B}B\underline{C}C \rightarrow$
 $\underline{a}A\underline{B}B\underline{C}C \rightarrow \underline{a}a\underline{B}B\underline{C}C \rightarrow \underline{a}a\underline{b}B\underline{C}C \rightarrow \underline{a}a\underline{b}b\underline{C}C \rightarrow \underline{a}a\underline{b}b\underline{c}C \rightarrow \underline{a}a\underline{b}b\underline{c}c$



Types of Grammars

- Grammars are classified by the kinds of productions they allow:
 - Type 0: no restriction
 - Type 1: length of LHS \leq length of RHS
 - Type 2: LHS is a single auxiliary only
 - Type 3: LHS is a single auxiliary, and RHS is either ϵ , or ϵA where A is an auxiliary and $\epsilon \in \Sigma$.
- Note: Our definition differs from the book's slightly. However, the basic ideas are the same.




Names for Types of Grammars

- ❑ Type 0: phrase-structure grammar
- ❑ Type 1: context-sensitive grammar
- ❑ Type 2: context-free grammar
- ❑ Type 3: right-linear grammar

Type 2 was the type of grammar studied in CS 60.

These types are called the “Chomsky Hierarchy”, after linguist Noam Chomsky, who first named them.

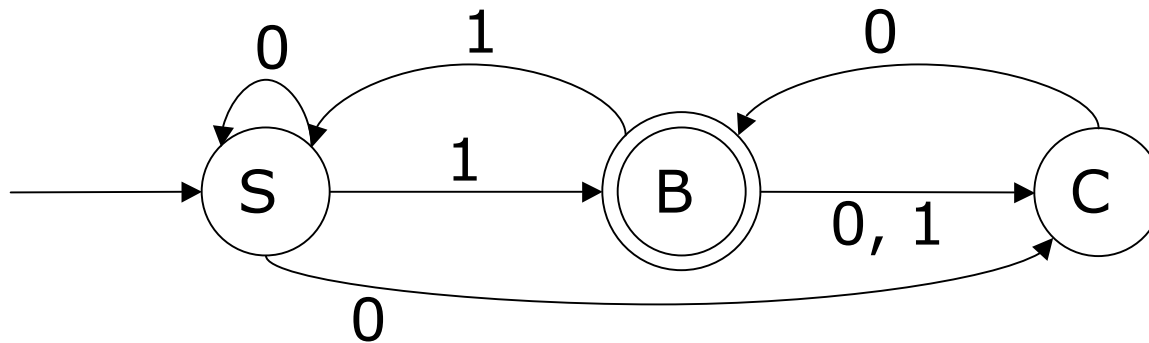


A language is regular iff it is generated by some type 3 grammar.

- Type 3 productions are of one of two types:
 - $B \rightarrow \alpha C$, where $B \in A$, $\alpha \in \Sigma^*$
 - $B \rightarrow \epsilon$
- To prove this result, identify the states of a NFA with auxiliaries in the grammar. Assume a single start state and no ϵ -transitions (WLOG!).
 - $B \rightarrow \alpha C$ is a production if state B goes to state C via symbol α .
 - $B \rightarrow \epsilon$ is a production iff B is an accepting state in the NFA.
- The language generated by the grammar is the language generated by the NFA.
- The only way to get rid of an auxiliary in the derived string is to use the production $B \rightarrow \epsilon$, which corresponds to the NFA being in an accepting state.

Example: NFA vs. Grammar

NFA:



Grammar:

- Start symbol is S
- Productions:

$S \rightarrow 0S$
 $S \rightarrow 0C$
 $S \rightarrow 1B$

$B \rightarrow 1S$
 $B \rightarrow 0C$
 $B \rightarrow 1C$
 $B \rightarrow \epsilon$

$C \rightarrow 0B$



There are languages of type 2 that are **not regular**.

- $\{0^n 1^n \mid n \geq 0\}$ is known to be non-regular.
- The following type 2 grammar generates it:
 - $S \rightarrow 0S1$
 - $S \rightarrow \epsilon$



Grammars vs. Regular Expressions

- Every regular expression corresponds to a **type 2** grammar in a natural way. (The connection to a type 3 grammar is through Kleene's theorem.)
- Each sub-expression is identifiable with an auxiliary or a terminal symbol. The productions are:
 - $R \rightarrow ST$ if R is a product of sub-expressions S and T
 - $R \rightarrow S$ and $R \rightarrow T$ if R is a union of sub-expressions S and T
 - $R \rightarrow SR$ and $R \rightarrow \epsilon$ if R is S^*
 - $R \rightarrow \epsilon$ if $\epsilon \in R$
 - $R \rightarrow \epsilon$ if R is ϵ
 - none if R is \emptyset

Example

- Regular expression: $0((10)^* \sqcup 01)^*$
 - $R \sqcup ST$ // $R = 0((10)^* \sqcup 01)^* = ST$
 - $S \sqcup 0$ // $S = 0$
 - $T \sqcup VT$ // $T = ((10)^* \sqcup 01)^* = V^*$
 - $T \sqcup \sqcup$
 - $V \sqcup W$ // $V = (10)^* \sqcup 01 = W \sqcup X$
 - $V \sqcup X$
 - $W \sqcup YW$ // $W = (10)^* = Y^*$
 - $W \sqcup \sqcup$
 - $Y \sqcup 10$ // $Y = 10$
 - $X \sqcup 01$ // $X = 01$

- Note the connection with solving language equations.



Closure Properties

- ❑ From the previous discussion, it can easily be seen that context free languages are closed under:
 - ❑ union
 - ❑ product (concatenation)
 - ❑ star operator
- ❑ Similarly, we can easily see that context free languages are closed under reversal, prefix, suffix, etc.
- ❑ We will eventually see that, unlike regular languages, they are *not* closed under intersection.



Closure Under Substitution (Homomorphism)

- Suppose that L is a language over Σ .
- By a **substitution map**, we mean a function that assigns to each element of a string from an alphabet Σ .
- Example: $\Sigma = \{0, 1\}$, $\Delta = \{a, b, c\}$,
 $s(0) = ab$, $s(1) = cbaba$.
- We can “extend” s to map any language over Σ by simply applying s to the letters in each string in the language and concatenating the results for that string.
- Example: $L = \{1\}^*\{0\}$
 $s(L) = \{cbaba\}^*\{ab\}$



Both regular and context-free languages are closed under substitution mapping.

Grammar Shorthand

- Suppose that productions are:
 - $A \rightarrow V$
 - $A \rightarrow A + V$
 - $V \rightarrow a$
 - $V \rightarrow b$
 - $V \rightarrow c$
- *Group* by common left-hand sides
- Use | (read “or”) to represent alternatives:
 - $A \rightarrow V \mid A + V$
 - $V \rightarrow a \mid b \mid c$
- Note: | “binds more loosely” than other symbols.
- Same grammar, just a briefer notation.
- | is like union in regular expressions.
 - $A \rightarrow V \mid A + V$ has a solution: $A = V (+ V)^*$



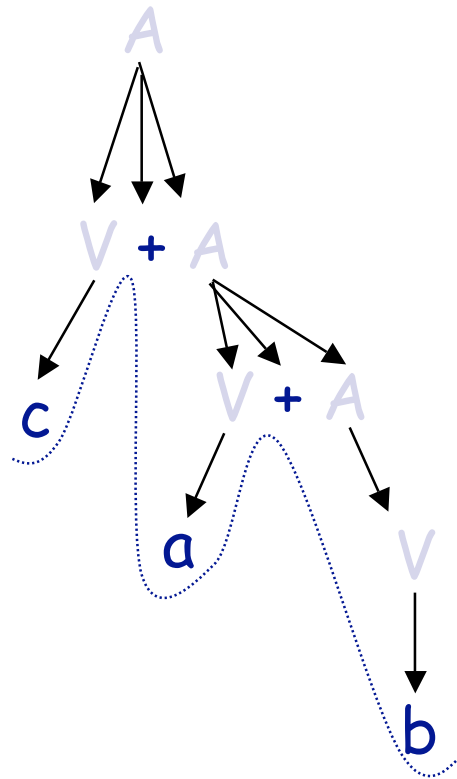
There are languages that are type 1 but not type 2.

- $\{a^k b^k c^k \mid k \in \mathbb{N}, k > 0\}$ can be shown to be type 1. However, there is no type 2 grammar that generates it.
- This is due to the ***pumping lemma for context-free languages.***
- Before presenting this, we need to review **derivation trees.**

Derivation Tree Visualization

$A \rightarrow V \mid V + A$
 $V \rightarrow a \mid b \mid c$

lines indicate
that a production
is being applied



Terminal string = "fringe" of tree = "c + a + b"



Derivation Tree Advantage

- ❑ The derivation tree has the advantage over linear derivations using \square .
- ❑ Many different derivations can be shown using a single tree.
- ❑ These derivations are, in some sense, equivalent.

- ❑ Exercise: List all derivations corresponding to the tree on the previous page.



Ambiguity

- ❑ Derivation trees are often used, e.g. in compilers, to assign **meaning** to generated strings.
- ❑ If a string has more than one derivation tree, it is called **ambiguous**.
- ❑ An ambiguous grammar is one that generates at least one ambiguous string.



Ambiguity

- Consider the grammar
 - $A \rightarrow V \mid A * A \mid A + A$
 - $V \rightarrow a \mid b \mid c$
- Show that this grammar is ambiguous.



Inherent Ambiguity

- For a given language, there may be both ambiguous and unambiguous grammars.
- A language that has no unambiguous grammar is called **inherently ambiguous**.
- An example, which we don't prove here, of such a language is

$$\{a^n b^n c^m d^m \mid n, m > 0\} \not\equiv \{a^n b^m c^m d^n \mid n, m > 0\}$$



Pumping Lemma for Context-Free Languages

- Let L be an infinite context-free language. Then there is a number n such that if $u \in L$ and $|u| > n$ then there are strings v, w, x, y, z , such that
 - $u = vwxyz$
 - $|wy| > 0$ (at least one of w or y is non-empty)
 - $|wxy| \leq n$
 - $(\forall m \geq 0) \ v w^m x y^m z \in L$



Proof that $\{a^k b^k c^k \mid k \in \mathbb{N}, k > 0\}$ is not context-free using the pumping lemma

- Suppose $\{a^k b^k c^k \mid k \in \mathbb{N}, k > 0\}$ were context-free. Let n be the integer that exists according to the pumping lemma. Consider $u = a^n b^n c^n$ and decompose into $vwxyz$.
- One of w and y is not ϵ . Suppose it's w . The other case is symmetric. By the PL, vw^2xy^2z is in L .
- Analyzing the cases for w as to whether it consists of all of one letter or of two letters, in all cases we get a contradiction.



Proof of the CFL Pumping Lemma

- The most direct proof requires a grammar in **Chomsky Normal Form**:
Every production, with one possible exception*, has one of these two forms:
 $A \rightarrow BC$, where B and C are auxiliaries
 $A \rightarrow \epsilon$, where $\epsilon \neq \epsilon$
- For every context-free language not containing ϵ , is generated by some grammar in Chomsky Normal Form.
- Assume this for now, see book for proof.



Observation

- For a Chomsky Normal Form grammar, the derivation tree is **binary**: each auxiliary node has either:
 - two children, both of which are auxiliary
 - one child, which is terminal

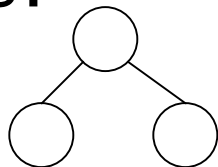
Binary Tree Observation

- ❑ The **height** of a binary tree is defined as the number of nodes from the root to the longest path.
- ❑ A binary tree with height $p+1$ has at most 2^p leaves.
- ❑ A binary tree with at least 2^p leaves has height at least $p+1$.

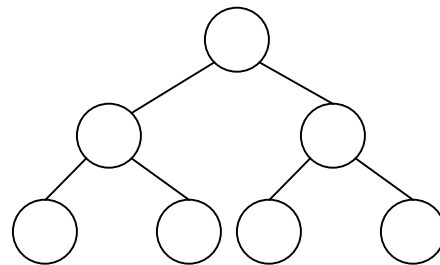
- ❑ Examples:



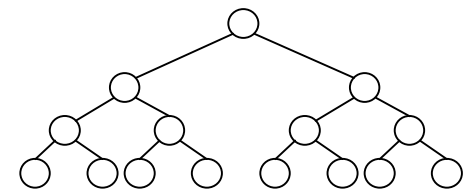
Height 1,
1 leaf



Height 2,
2 leaves



Height 3, 4 leaves



Height 4, 8 leaves

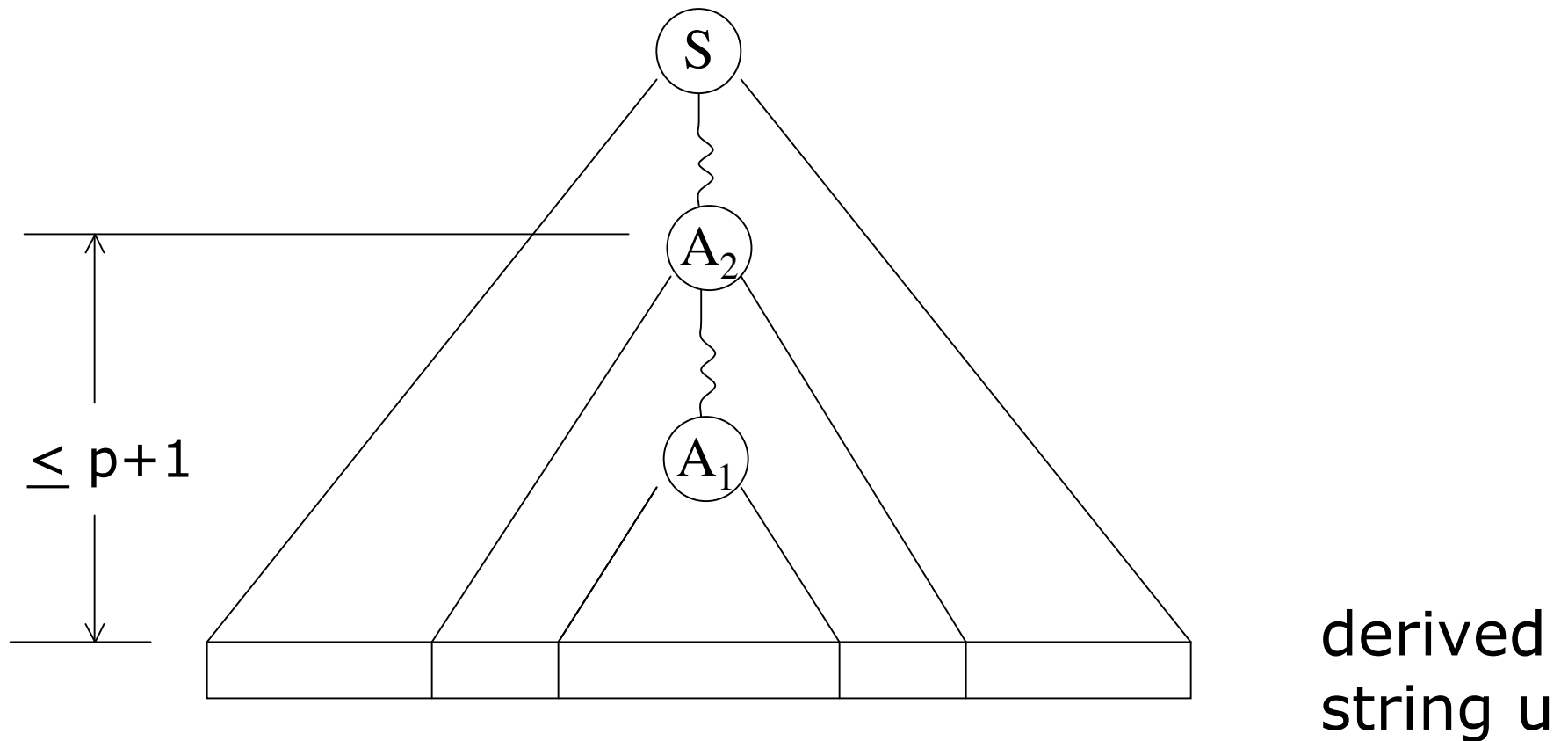


Proof of the Pumping Lemma (1)

- Suppose L is an infinite context-free language, and G is a Chomsky-Normal Form grammar for L .
- **Let p be the number of auxiliary symbols in G , exclusive of the start symbol if the exception is used.**
- We will show that the n that exists in the PL can be satisfied by $n = 2^{p+1}$.
- Let $u \in L$ be such that $|u| \geq n$. Then the derivation tree for u has at least 2^{p+1} leaves, so the height is at least $p+2$.
- Consider a maximum length path from leaf to root in this tree. This path has $> p+1$ auxiliary nodes, therefore some auxiliary must be repeated. Let A_1 be the first instance of a repeated auxiliary on the path and A_2 be the second. Such a repetition must take place in $\leq p+1$ nodes.

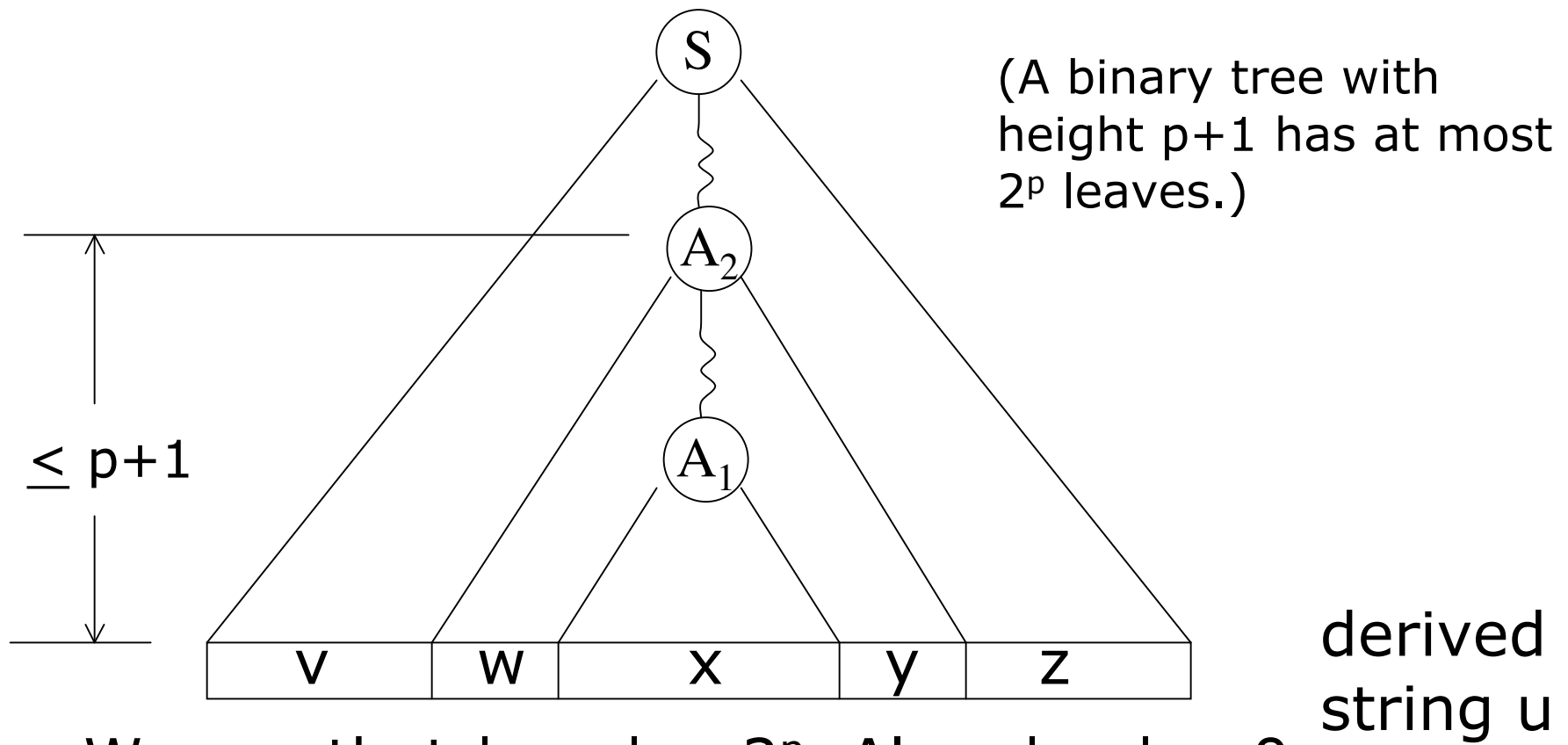
Proof of the Pumping Lemma (2)

□ Here is a picture of our derivation tree:



Proof of the Pumping Lemma (3)

- Choose v, w, x, y, z as follows:

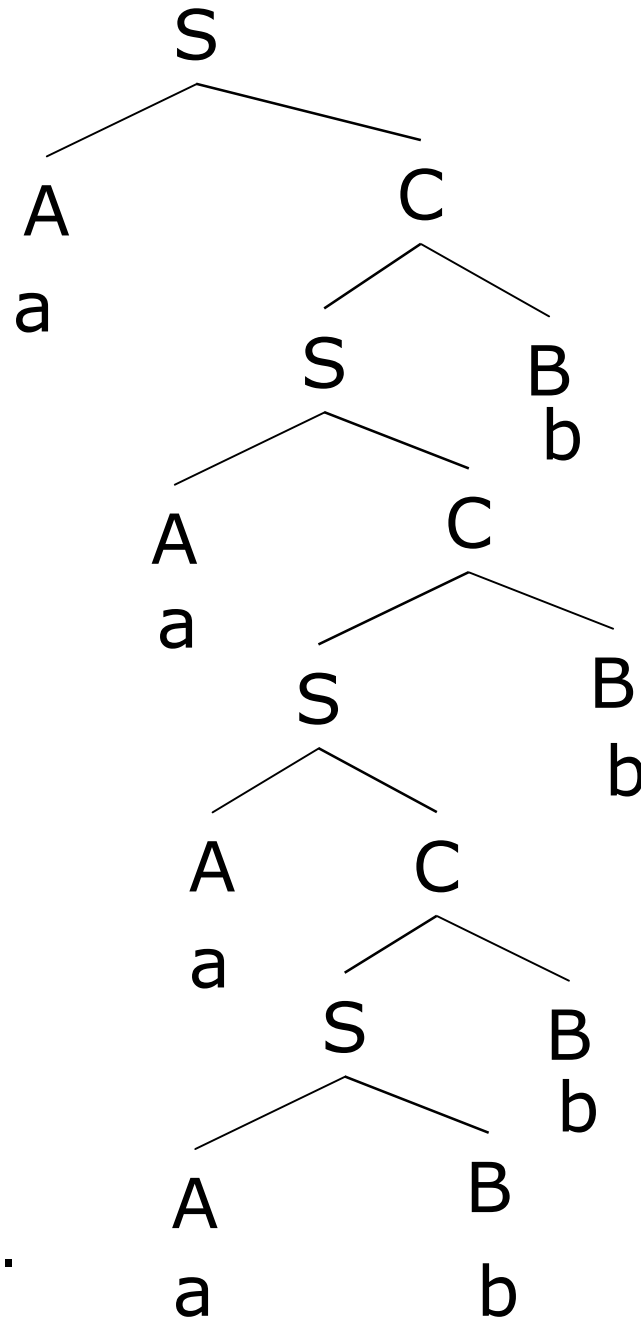


We see that $|wxy| \leq 2^p$. Also, $|wy| > 0$.



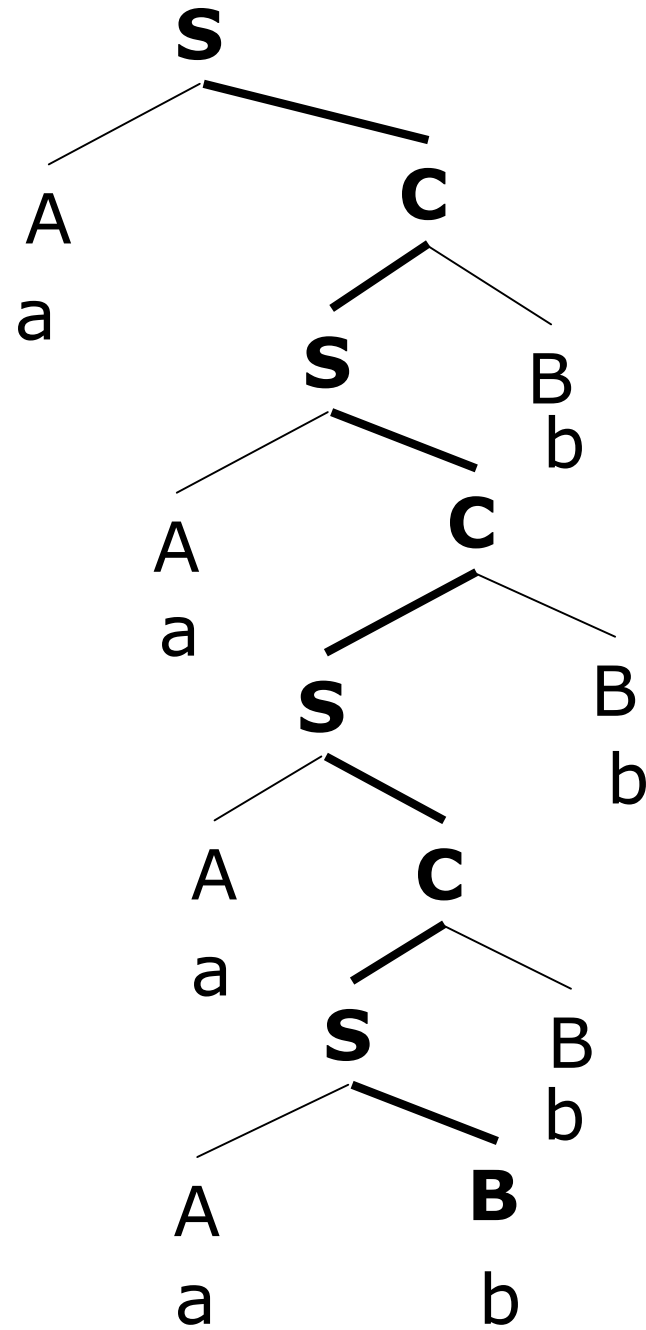
Example

- $S \rightarrow AC$
- $S \rightarrow AB$
- $C \rightarrow SB$
- $A \rightarrow a$
- $B \rightarrow b$
- Derivation tree for
aaaabbbb
- Note: We can illustrate the principle even tho' this string is not length or longer 16.



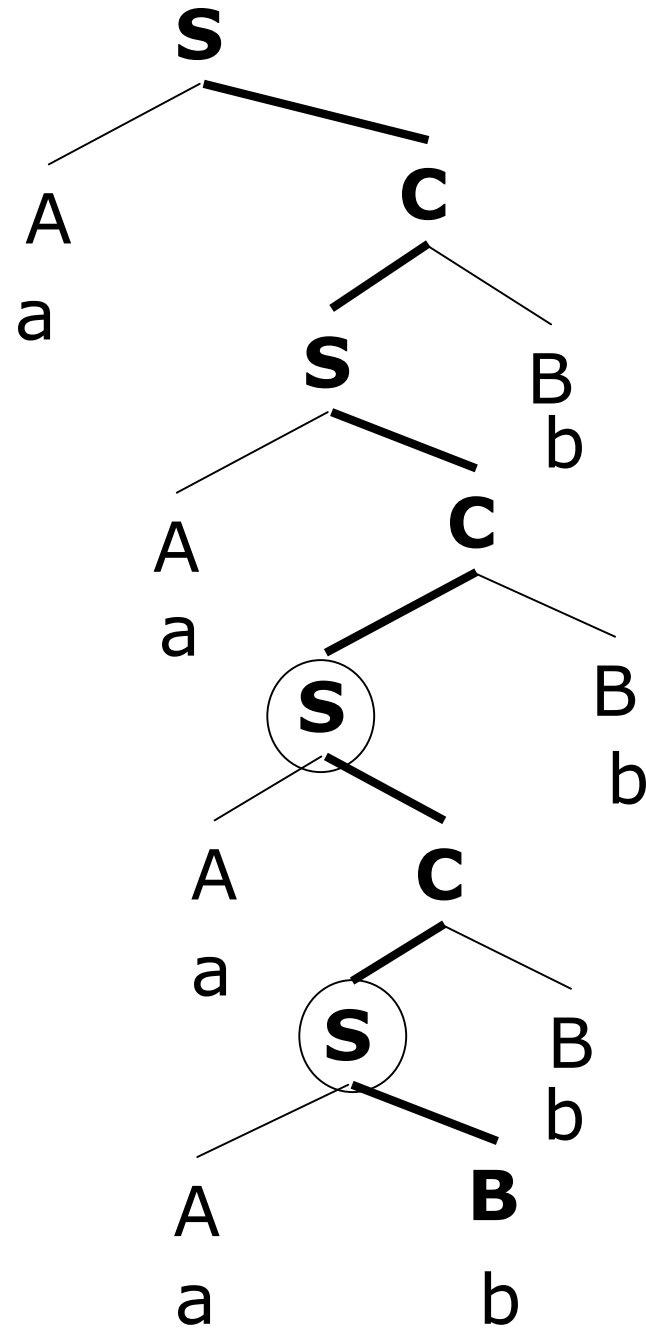


A Long Path





Repeated letters



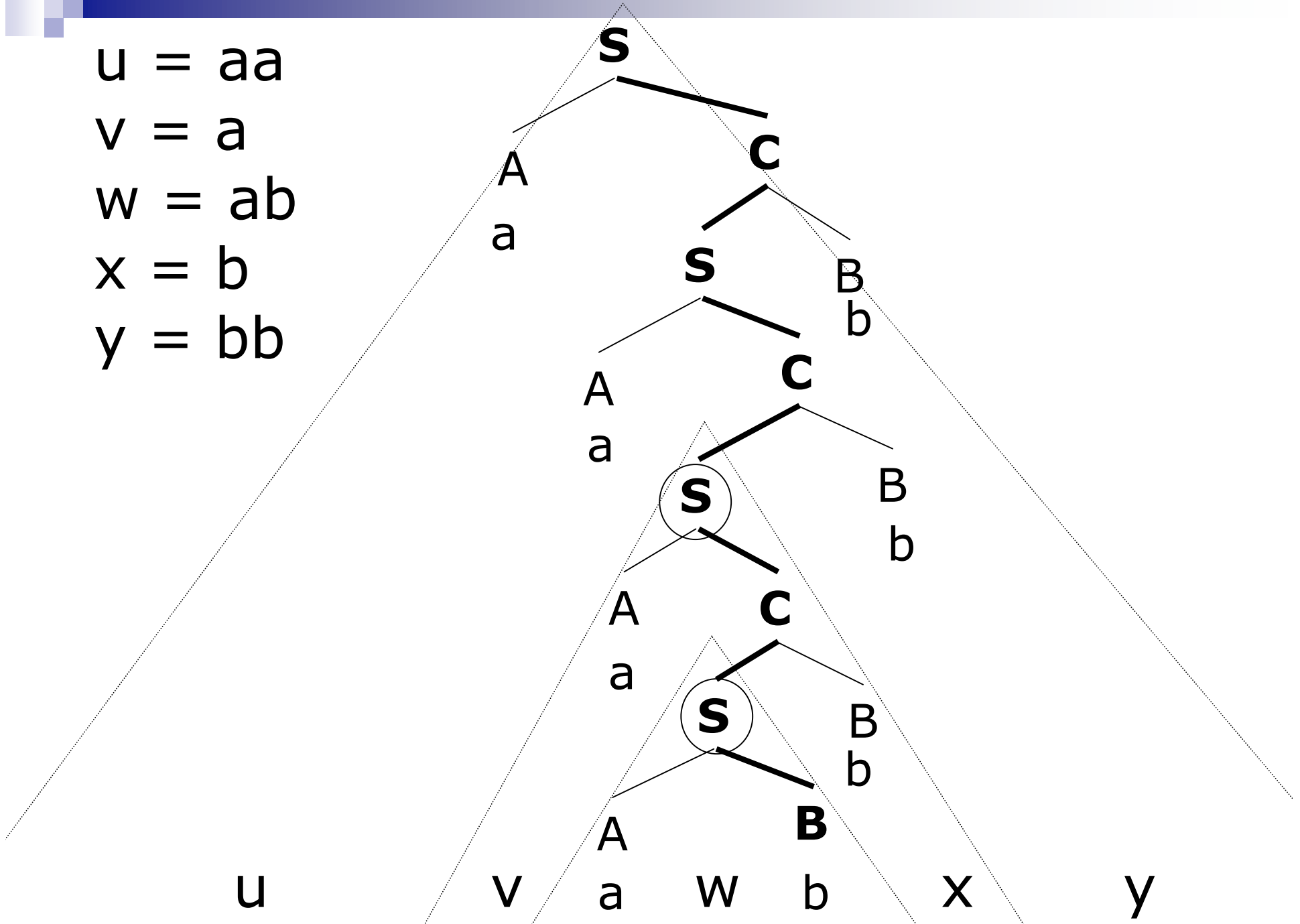
u = aa

v = a

w = ab

x = b

y = bb





Conclusion drawn from pumping

$$u = aa$$

$$v = a$$

$$w = ab$$

$$x = b$$

$$y = bb$$

Conclusion: aaa^kabb^kbb is L for all k .



Non-Closure Under Intersection

- The context-free languages are not closed under intersection.
- These can be shown to be context-free:
 - $\{a^k b^k c^m \mid k, m \geq 0\}$
 - $\{a^m b^k c^k \mid k, m \geq 0\}$
- However, their intersection is:
 - $\{a^k b^k c^k \mid k \geq 0\}$which we know is not context free.



Closure Under Intersection with a Regular Language

- If L is context-free and R is regular, then $L \cap R$ is context-free.
- An easy way to see this is to use a machine characterization of context-free languages, which we discuss subsequently.



Non-Closure Under Complementation


- The context-free languages are not closed under complementation.
- This language can be shown to be not context-free (using the pumping lemma):

$$\{ww \mid w \in \{0, 1\}^*\}$$

- However, the complement:

$$\{0, 1\}^* - \{ww \mid w \in \{0, 1\}^*\}$$

is. A grammar for it is given on the next page.



Grammar for $\{0, 1\}^* - \{ww \mid w \in \{0, 1\}^*\}$


$S \in AB \mid BA \mid A \mid B$

$A \in CAC \mid 0$

$B \in CBC \mid 1$

$C \in 0 \mid 1$

- This remains to be argued.



Proof that $\{ww \mid w \in \{0, 1\}^*\}$ is not context free.

- If this language were context free, so would its intersection with a regular language be.
- If we intersect with the regular language $\{0\}^*\{1\}^*\{0\}^*\{1\}^*$, we get a language where all strings are of the form:

$$0^r 1^s 0^r 1^s$$

Let n be the number that exists by the pumping lemma. Select $u = 0^n 1^n 0^n 1^n$ and decompose u in $uvwxy$ where $|vx| > 0$ and $|vwx| \leq n$.

- Show that uv^2wx^2y cannot be of the form $0^r 1^s 0^r 1^s$.