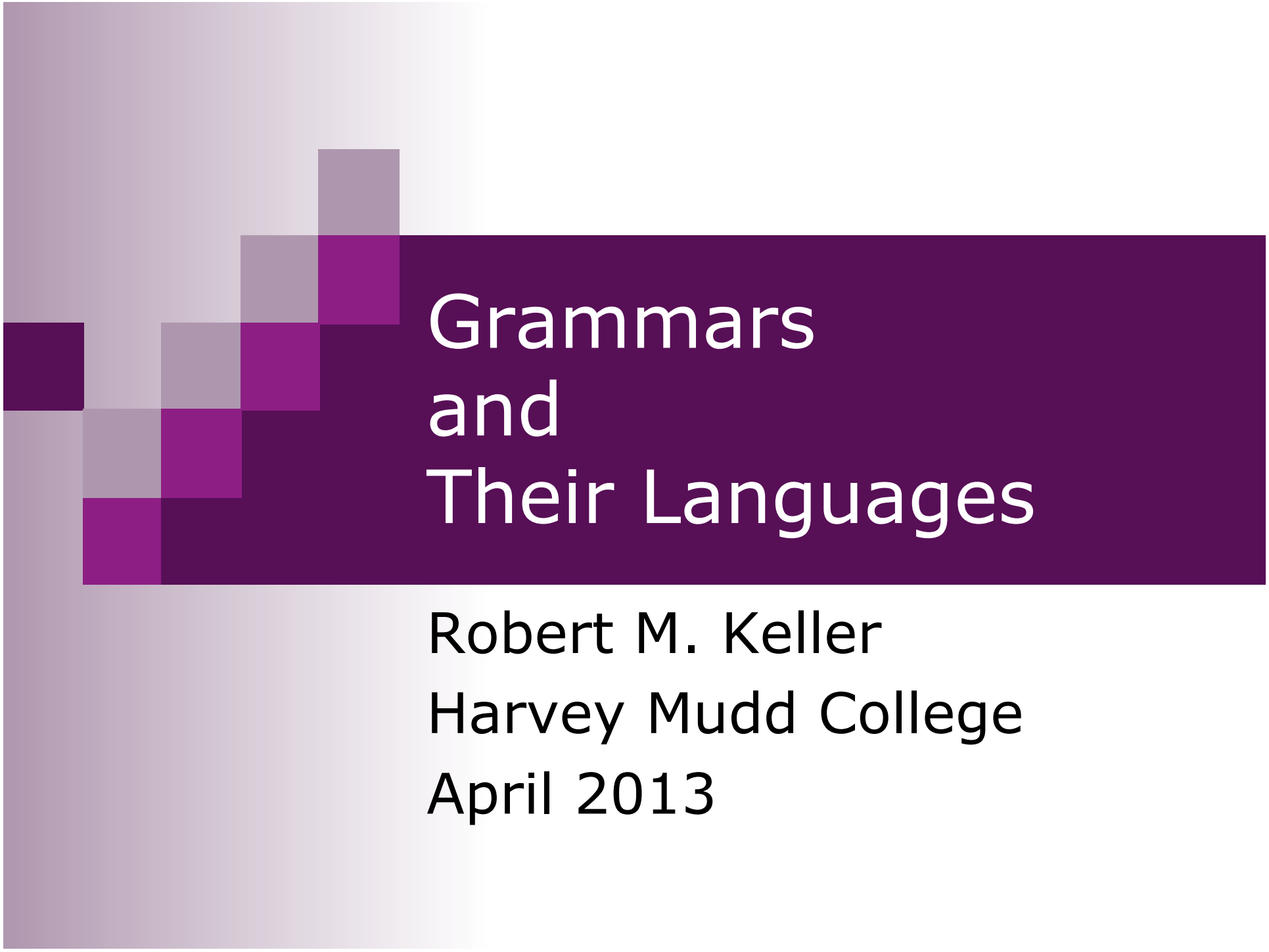


Robert M. Keller
Harvey Mudd College
April 2013

Review of Grammars

- In CS 60, the grammars studied their 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 the family of “L systems”.

Definition of Grammars

- A grammar consists of 4 parts:
  - Terminal alphabet Σ
  - Auxiliary (aka “non-terminal” or “variables”) alphabet N
  - Productions (to be defined)
  - Start symbol S $\in \Sigma$

Definition of Grammars

- A grammar consists of 4 parts:
  - Terminal alphabet Σ
  - Auxiliary alphabet N (such that $N \cap \Sigma = \varnothing$)
  - A finite set of productions $\rightarrow$
  - Start symbol $S \in N$
- Each production has the form $x \rightarrow y$, where $x \in (\Sigma \cup A)^+$, $y \in (\Sigma \cup A)^*$.
- A production enables **rewriting** a substring x of a derived string as the string y.
- x cannot be ε because there is nothing to rewrite in that case.

Derivation in a Grammar

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

- Thus, strings in the language generated do not include any auxiliaries. The latter must have been replaced by rewriting according to productions first.

Example of a Grammar

- Productions:
  - $S \rightarrow ab$
  - $S \rightarrow aSb$
  - $S \rightarrow SS$
- Example derivations of strings in the language:
  1. $S \Rightarrow ab$
  2. $S \Rightarrow aSb \Rightarrow aabb$
  3. $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$
  4. $S \Rightarrow SS \Rightarrow abS \Rightarrow abab$
  5. $S \Rightarrow SS \Rightarrow SSS \Rightarrow ababab$
  6. $S \Rightarrow SS \Rightarrow aSbS \Rightarrow aabbS \Rightarrow aabbaSb \Rightarrow aabbaabb$
- Examples of strings not in the language?

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 $\{a^n b^n c^n \mid n > 0\}$

- The grammar:
  - Terminal alphabet $\Sigma = \{a, b, c\}$
  - Auxiliary alphabet {S, A, B, C, E, F}
  - Productions $\rightarrow$:
    - $S \rightarrow FE$ $\quad E \rightarrow ABCE$ $\quad E \rightarrow \varepsilon$
    - $BA \rightarrow AB$ $\quad CA \rightarrow AC$ $\quad CB \rightarrow BC$
    - $FA \rightarrow a$ $\quad aA \rightarrow aa$ $\quad aB \rightarrow ab$
    - $bB \rightarrow bb$ $\quad bC \rightarrow bc$ $\quad cC \rightarrow cc$
  - Start symbol S

- A derivation (underlines show symbols replaced):
  $S \Rightarrow F\underline{E} \Rightarrow FABC\underline{E} \Rightarrow FABCABC\underline{E} \Rightarrow FAB\underline{CA}BC \Rightarrow FA\underline{BA}CBC \Rightarrow FAAB\underline{CB}C \Rightarrow \underline{FA}ABBCC \Rightarrow \underline{aA}BBCC \Rightarrow a\underline{aB}BCC \Rightarrow aa\underline{bB}CC \Rightarrow aab\underline{bC}C \Rightarrow aabb\underline{cC} \Rightarrow aabbcc$

Types of Grammars

- Grammars are classified by the kinds of productions they allow, from least restrictive to most restrictive:
  - Type 0: no restriction on productions
  - 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 σA where A is an auxiliary and $\sigma \in \Sigma$

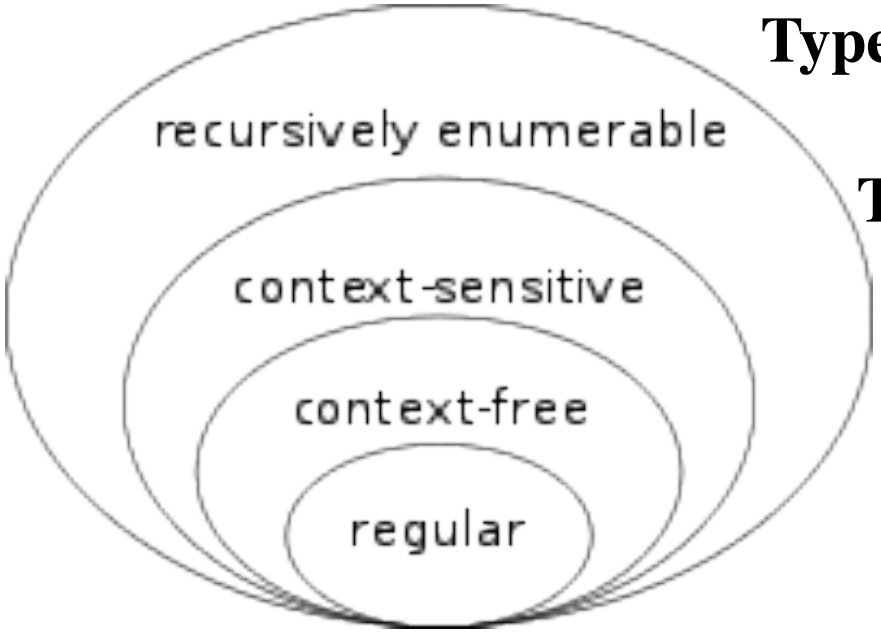
Names for Types of Grammars

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

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

Chomsky, Noam (1959). "On certain formal properties of grammars". Information and Control 2 (2): 137–167.

Language classes corresponding to grammars



Type 0: phrase-structure grammar

Type 1: context-sensitive grammar

Type 2: context-free grammar

Type 3: right-linear grammar

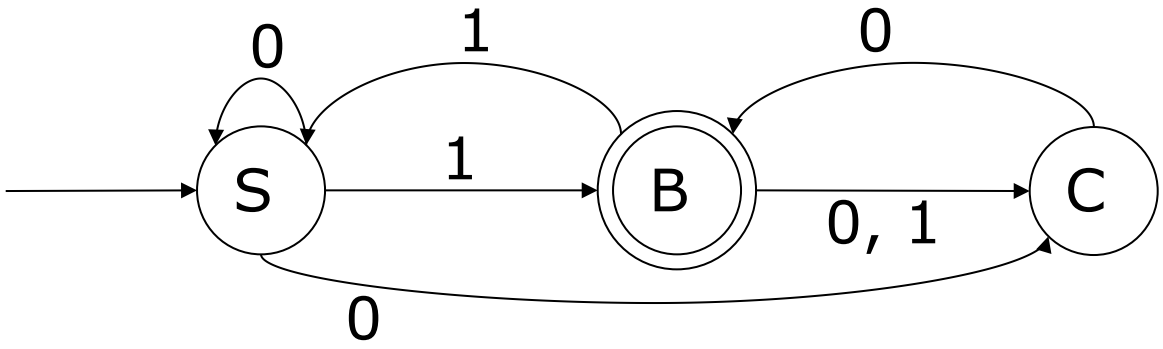
http://en.wikipedia.org/wiki/Chomsky_hierarchy

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

- Type 3 productions are of one of two types:
  - $B \to \sigma C$, where $B, C \in A$, $\sigma \in \Sigma$
  - $B \to \varepsilon$
- 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!).
  - $B \to \sigma C$ is a production if state B goes to state C via symbol σ.
  - $B \to \varepsilon$ 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 $B \to \varepsilon$, which corresponds to the NFA being in an accepting state.

Example: NFA vs. Grammar

NFA:



Grammar:

- Start symbol is S
- Productions:

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

- # of productions = # of arcs + # of accepting states

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

- $\{0^n1^n \mid n \geq 0\}$ is known to be non-regular.
- The following type 2 grammar generates it:
  - $S \rightarrow 0S1$
  - $S \rightarrow \varepsilon$

Summary

- Type 2 grammars are strictly more powerful than Type 3 grammars and regular expressions.

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 $R \rightarrow \varepsilon$ if R is S*
  - $R \rightarrow \sigma$ if $\sigma \in \Sigma$
  - $R \rightarrow \varepsilon$ if R is ε
  - none if R is $\varnothing$

Example

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

- Note the connection with solving language equations using Arden's rule.

Closure Properties

- From the previous discussion, it can easily be seen that context-free (type 2) 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 Σ a string from an alphabet Δ.

- 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 → V
  - A → A + V
  - V → a
  - V → b
  - V → c
- *Group* by common left-hand sides
- Use | (read "or") to represent alternatives:
  - A → V | A + V
  - V → a | b | c
- Note: | "binds more loosely" than other symbols.
- Same grammar, just a briefer notation.
- | is like union in regular expressions.
  - A → V | A + V has a solution: A = V (+ V)*

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

- $\{a^k b^k c^k \mid k > 0\}$ can be shown to be type 1. However, there is no type 2 grammar that generates it.

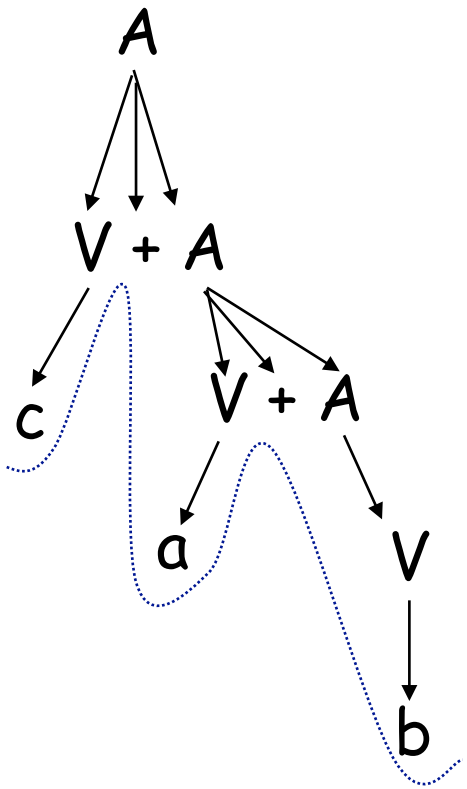
- This is due to the ***pumping lemma for context-free languages***.

- 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 an advantage over linear derivations using $\Rightarrow$.
- 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 is undesirable when we later get to assigning a meaning to strings in the language.

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 context-free grammars.
- A language that has *no* unambiguous context-free grammar is called **inherently ambiguous**.
- An example, which we don't prove here, of such a language is

  $\{a^n b^n c^m d^m \mid n, m > 0\} \cup \{a^n b^m c^m d^n \mid n, m > 0\}$

Pumping Lemma for Context-Free Languages

- Let L be a context-free language. Then there is a number p such that if $s \in L$ and $|s| > p$ then there are strings u, v, x, y, z, such that
  - $s = uvxyz$
  - $|vy| > 0$ (i.e. at least one of w or y is non-empty)
  - $|vxy| \leq p$
  - $(\forall m \geq 0)\ \ u\ v^m\ x\ y^m\ z \in L$

- We can use a Skolem function to give p. We name it pump(L), the **pumping length** of L.

Pumping Lemma Expressed in Logic

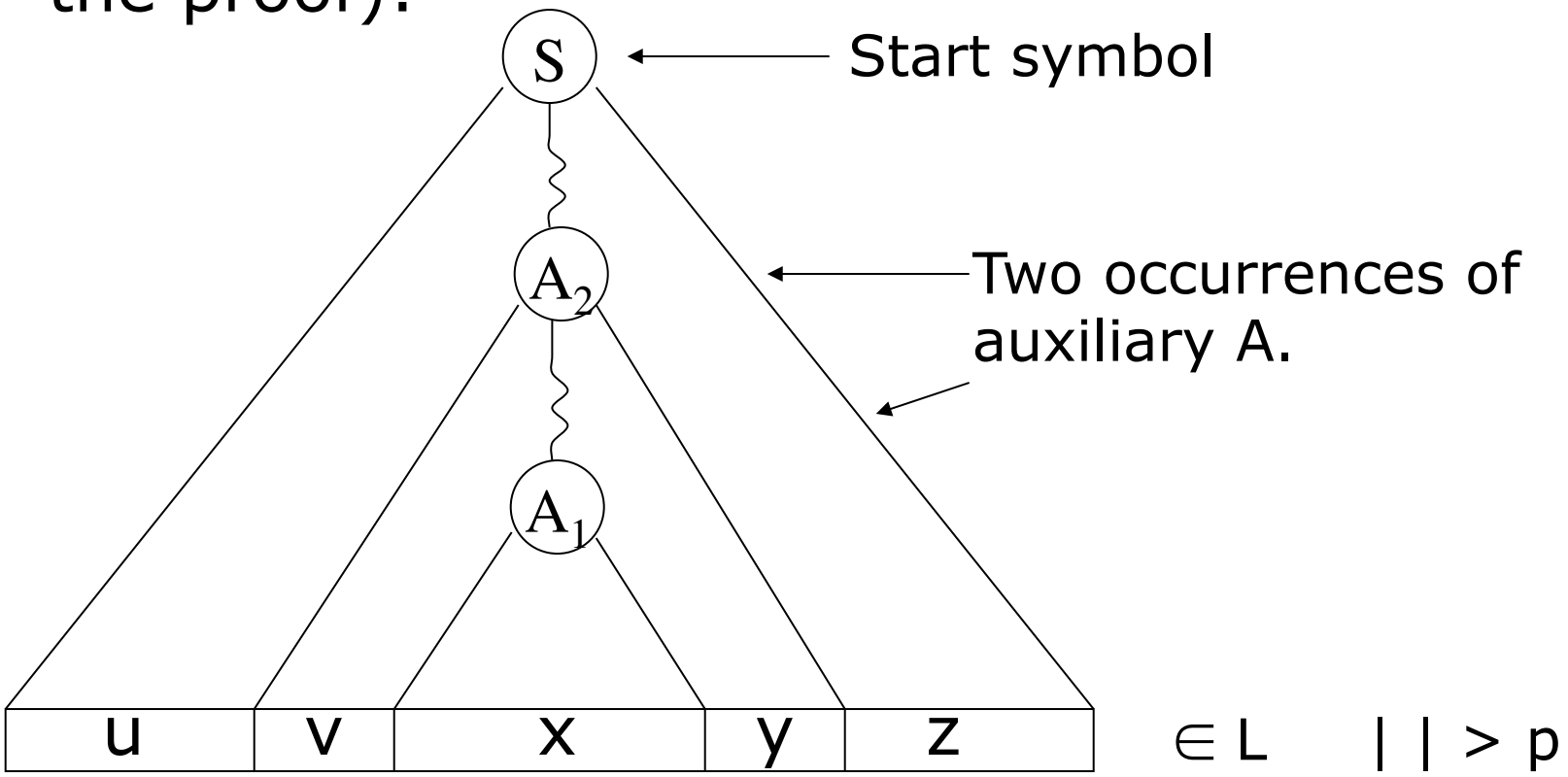
$$
\begin{aligned}
&\forall L\ (\text{context-free}(L) \rightarrow \\
&\quad (\exists p\ \forall s\ (\\
&\qquad (s \in L \wedge |s| > p) \rightarrow \\
&\qquad\quad (\exists u\ \exists v\ \exists x\ \exists y\ \exists z \\
&\qquad\qquad s = uvxyz \\
&\qquad\qquad \wedge\ |vy| > 0 \\
&\qquad\qquad \wedge\ |vxy| \leq p \\
&\qquad\qquad \wedge\ \forall m\ (m \geq 0 \rightarrow u\ v^m\ x\ y^m\ z \in L)))))
\end{aligned}
$$

Proof of the Pumping Lemma

- The proof is analogous to the proof of the PL for regular languages.
- However, we don’t have states to help us. So we use the auxiliary symbols instead.

How to Remember the Pumping Lemma

Think of this "nested wedges" picture (used in the proof):



Impact of the Pumping Lemma

- The PL can be used to show a language is **not context free**, since it provides a *necessary* condition
  (L is CF → L pumpable), thus
  (L not pumpable L → not CF).

- It **cannot** be used to show a language is context-free.

What about finite-languages?

- Finite languages (which are regular and thus context-free) obviously cannot be pumped.

- The PL holds vacuously for them:
  The pumping length is 1 longer than the longest string in the language.

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

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

More detailed case analysis

- $a^p b^p c^p = uvxyz \in L$, where $|vxy| \leq p$ and $|vy| > 0$.
- Due to $|vxy| \leq p$, either
  $vxy \in \{a\}^*\{b\}^*$ or $vxy \in \{b\}^*\{c\}^*$.
- If $vxy \in \{a\}^*\{b\}^*$, then uv^2xy^2z has the wrong number of c's to be in L.
- If $vxy \in \{b\}^*\{c\}^*$, then uv^2xy^2z has the wrong number of a's to be in L.
- So in either case we have 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

  $A \rightarrow \sigma$, where $\sigma \in \Sigma$

- Exception: $S \rightarrow \varepsilon$ is allowed, if S is the start symbol and is not on the right-hand side of any production.

- Assume this for now, see book for proof.

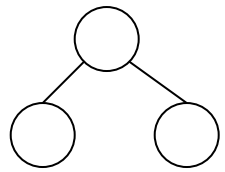
Observation

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

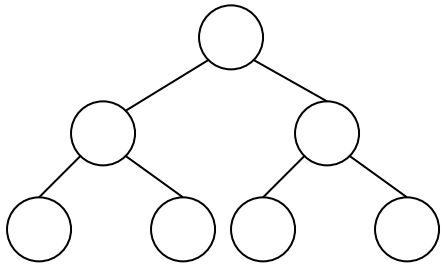
Binary Tree Observation

- 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 n+1 has at most 2^n leaves.
- Thus a binary tree with **at least 2^n leaves** has **height at least n+1**.
- Examples:

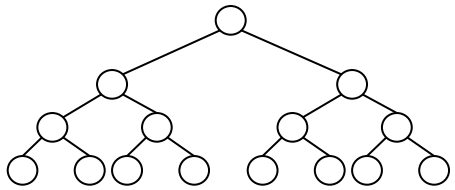
Height 1,
1 leaf
(n = 0)



Height 2,
2 leaves
(n = 1)



Height 3, 4 leaves
(n = 2)



Height 4, 8 leaves
(n = 3)

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 n be the number of auxiliary symbols in G.**
- We will show that the p that exists in the PL can be satisfied by $\mathbf{p = 2^{n+1}}$.

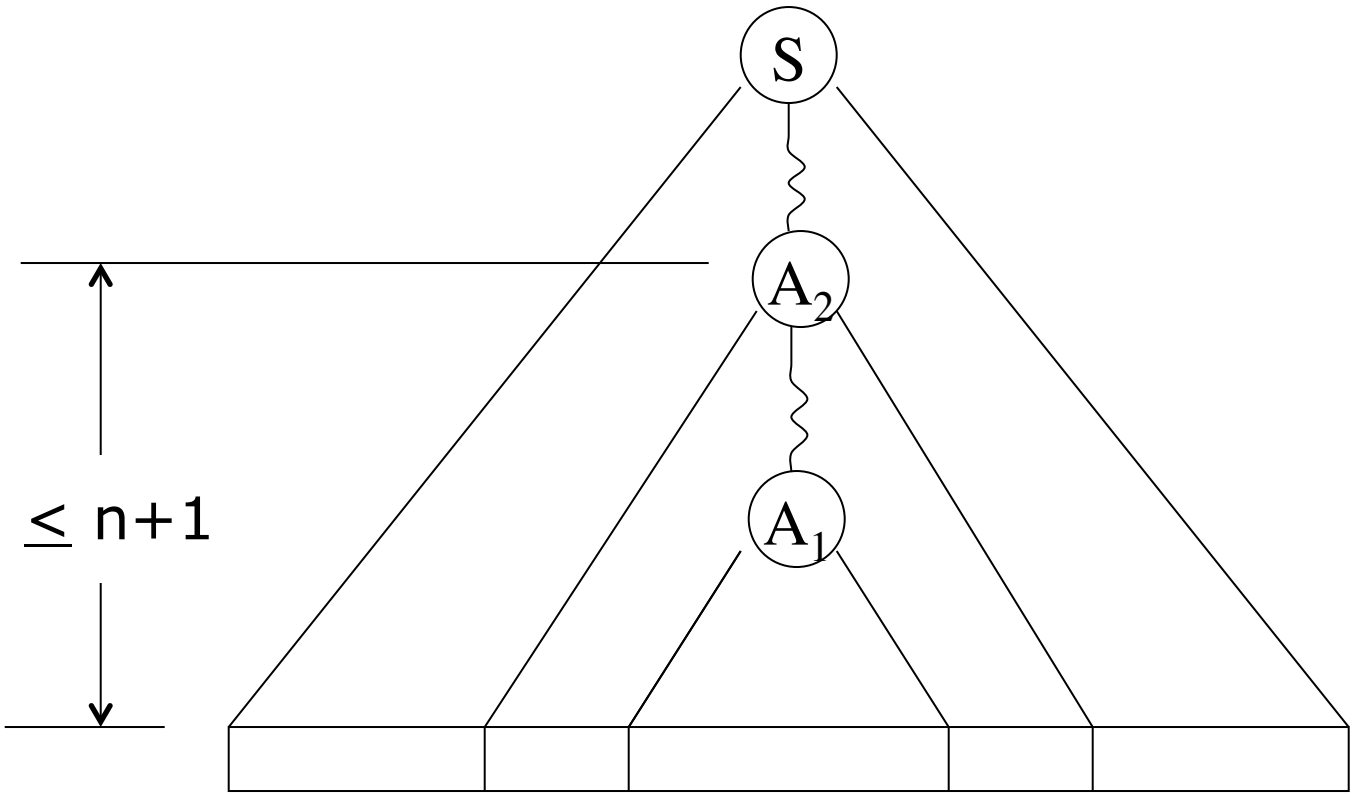
- Let $s \in L$ be such that $|s| \geq p$. Then the derivation tree for s has at least 2^{n+1} leaves, so the height is at least n+2.

- Consider a maximum length path from leaf to root in this tree. This path has > n+1 auxiliary nodes, therefore some auxiliary must be repeated.

- Let A_1 be the first instance of a repeated auxiliary on the path from leaf to root and A_2 be the second. Such a repetition must take place in $\leq$ n+1 nodes.

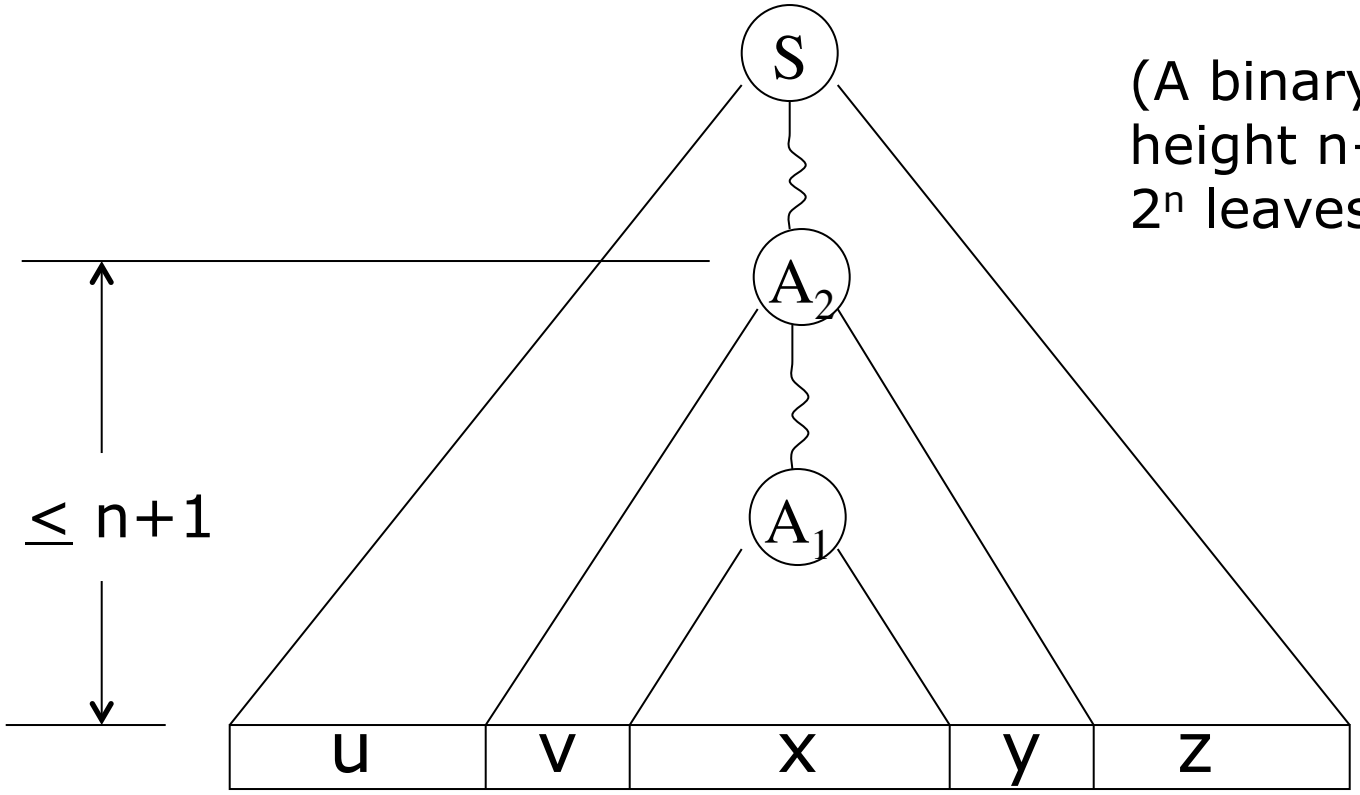
Proof of the Pumping Lemma (2)

- Here is a picture of our derivation tree:



Proof of the Pumping Lemma (3)

- Identify u, v, x, y, z as follows:



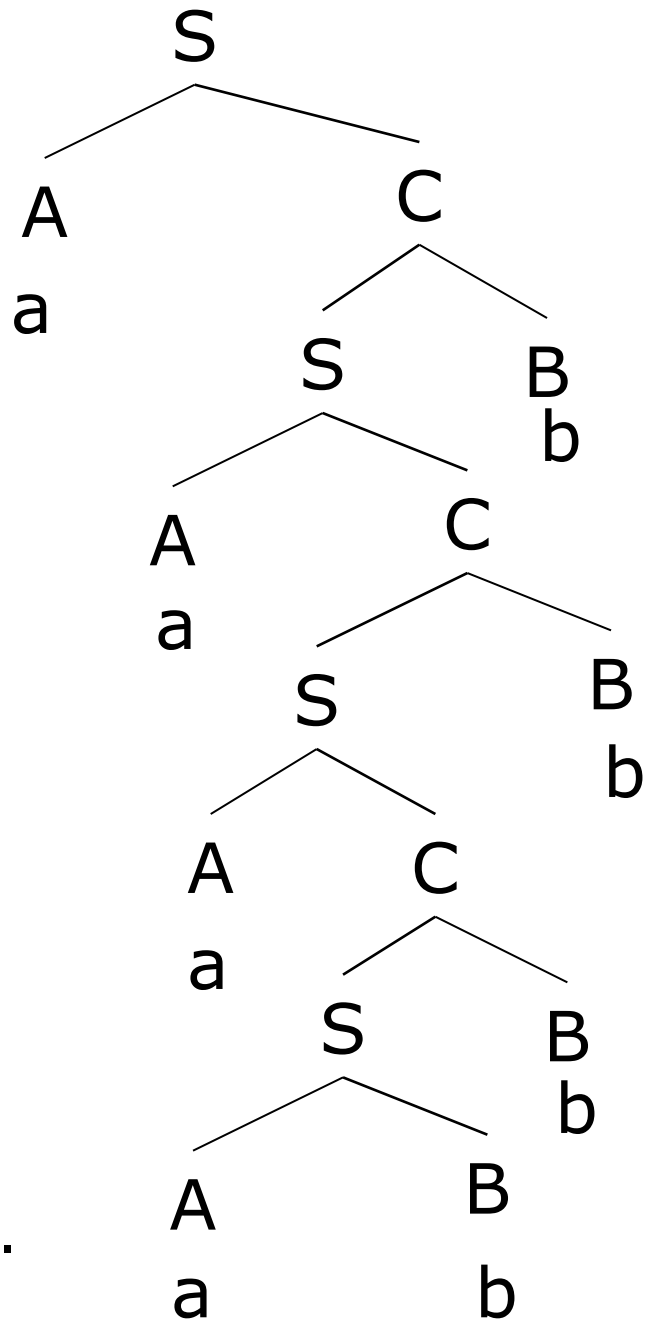
(A binary tree with height n+1 has at most 2^n leaves.)

derived string s

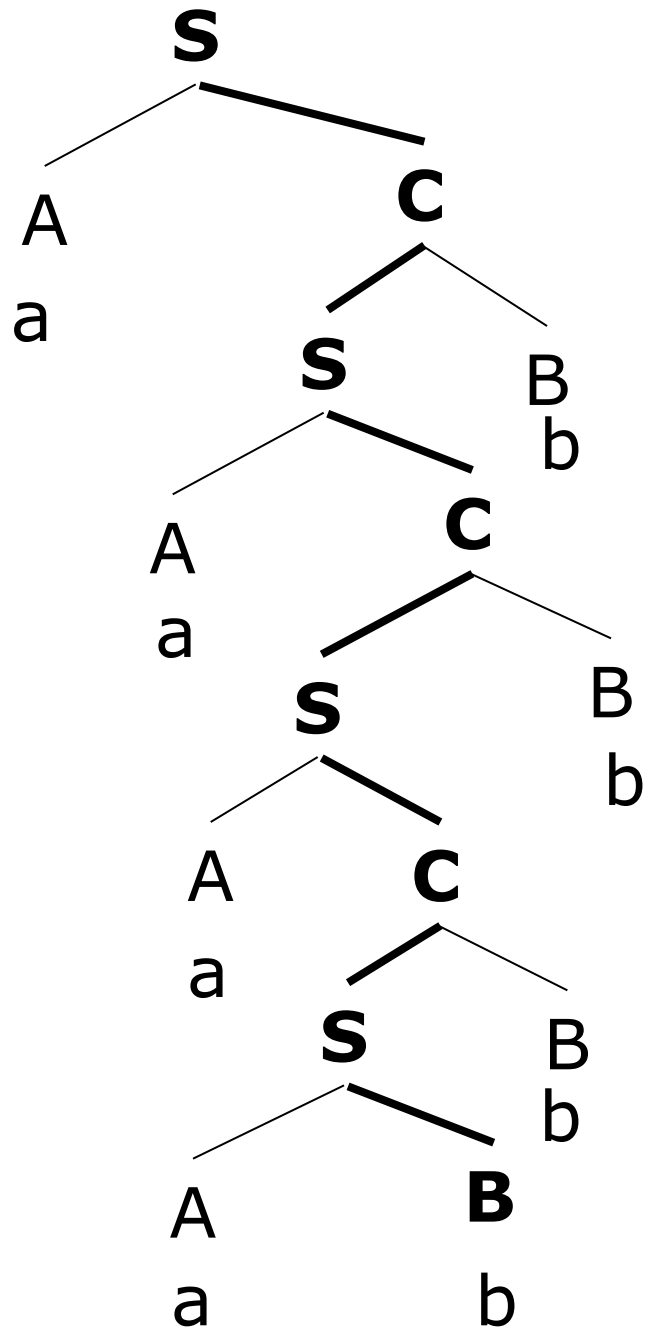
We see that $|vxy| \leq 2^n$. Also, $|vy| > 0$.

Example

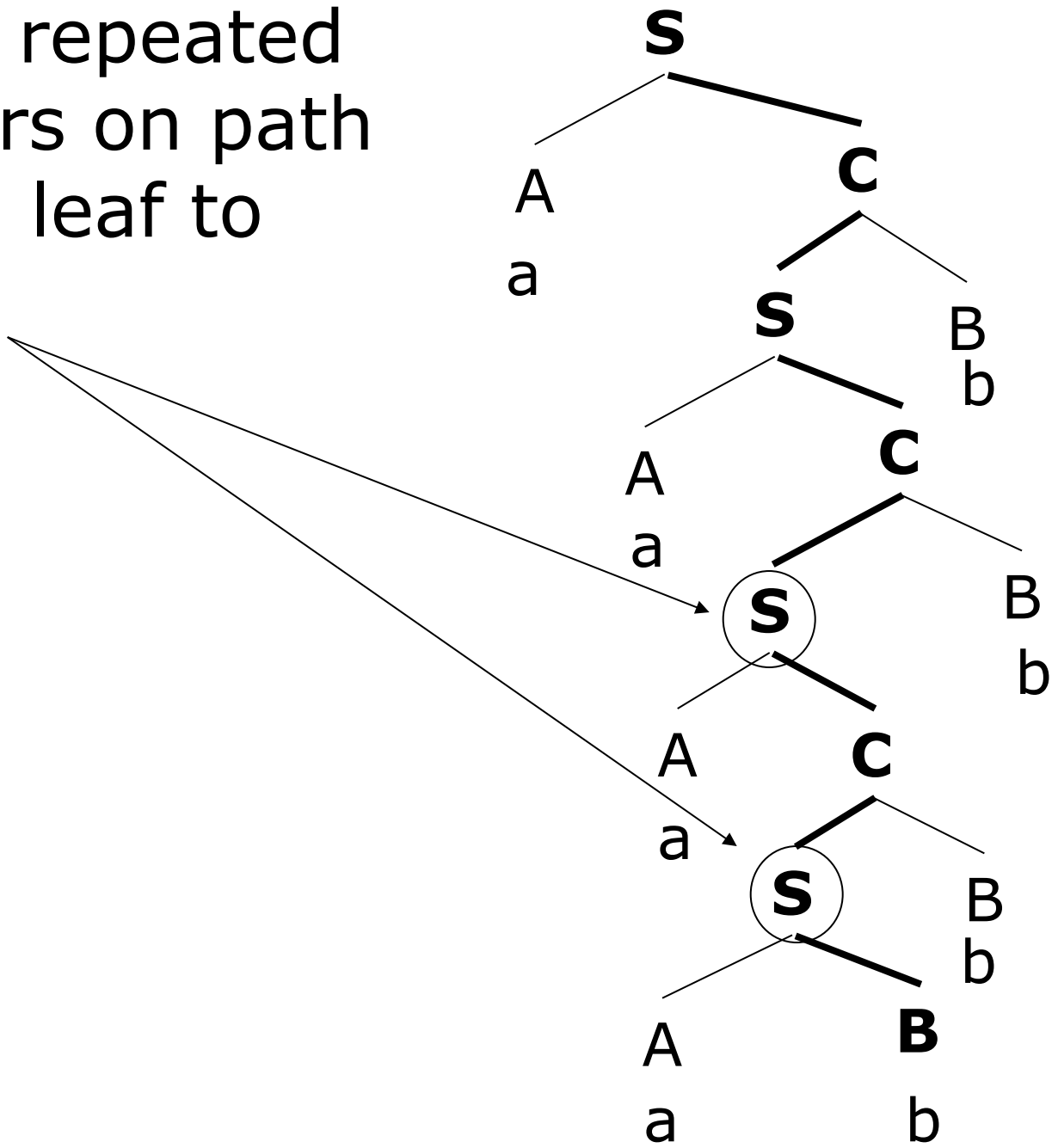
- S → AC
- S → AB
- C → SB
- A → a
- B → b
- Derivation tree for terminal string aaaabbbb
- Note: We can illustrate the principle even tho' this string is not length or longer 16.



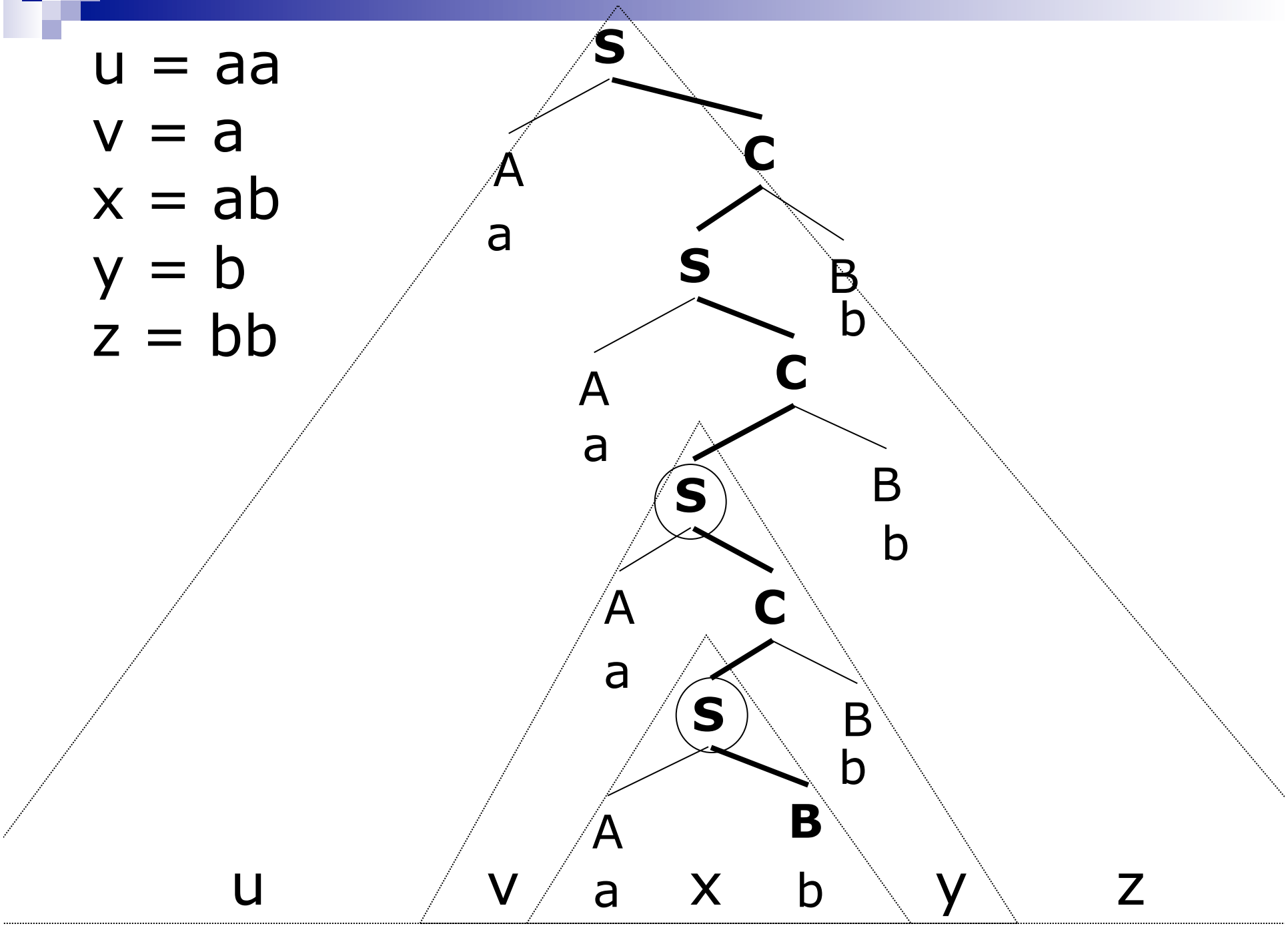
A Long Path



First repeated letters on path from leaf to root



u = aa
v = a
x = ab
y = b
z = bb



Conclusion drawn from pumping

u = aa
v = a
x = ab
y = b
z = bb

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

Non-Closure Under Intersection

- The context-free languages are not closed under intersection. Here's why:
- These can be shown to be context-free:
  - $\{a^k b^k c^m \mid k, m > 0\}$
  - $\{a^m b^k c^k \mid k, m > 0\}$
- However, their intersection is:
  - $\{a^k b^k c^k \mid k > 0\}$

  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∩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.
- Proof: Homework