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 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 "L systems".

Definition of Grammars

- A grammar consists of 4 parts:
  - Terminal alphabet ▯
  - Auxiliary alphabet
  - Productions
  - Start symbol S ▯

Definition of Grammars

- A grammar consists of 4 parts:
  - Terminal alphabet □
  - Auxiliary alphabet A (such that A □ □ = ∅)
  - A finite set of productions □
  - Start symbol S □ A
- Each production has the form x □ y, where x □ (□ □ A)$^+$, y □ (□ □ 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 □ x_1 □ x_2 □ ... each in (□ □ A)* where:
  - x_0 = S, the start symbol
  - For each i, x_i □ 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 □ v' is a production
- The **language generated by a grammar** is the set of strings x □ □* such that there is a derivation that ends with x.

Example of a Grammar

- Productions:
  - S → ab
  - S → aSb
  - S → SS
- Example derivations of strings in the language:
  - S → ab
  - S → aSb → aabb
  - S → aSb → aaSbb → aaabbb
  - S → SS → abS → abab
  - S → SS → SSS → ababab
  - S → SS → aSbS → aabbS → aabbaSb → aabbaabb

Example:
Grammar for Additive Arithmetic Expressions

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

Another Example of a Grammar for the Language {$a^n b^n c^n$ | n □ □, n > 0}

- The grammar:
  - Terminal alphabet □ = {a, b, c}
  - Auxiliary alphabet {S, A, B, C, F}
  - Productions □ :
    - S □ FE   E □ ABCE   E □ □
    - BA □ AB   CA □ AC   CB □ BC
    - FA □ a   aA □ aa   aB □ ab
    - bB □ bb   bC □ bc   cC □ cc
  - Start symbol S
- A derivation (underlines show symbols replaced):
  S □ FE □ FABCE □ FABCABCE □ FABCABC □
  FABACBC □ FAABCBC □ FAABBCC □ aABBCC □
  aaBBCC□ aabBCC □ aabbCC □ aabbcC □ aabbcc

Types of Grammars

- Grammars are classified by the kinds of productions they allow:
  - Type 0: no restriction
  - Type 1: length of LHS < length of RHS
  - Type 2: LHS is a single auxiliary only
  - Type 3: LHS is a single auxiliary, and RHS is either □, or □A where A is an auxiliary and □ □ □.
- 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.

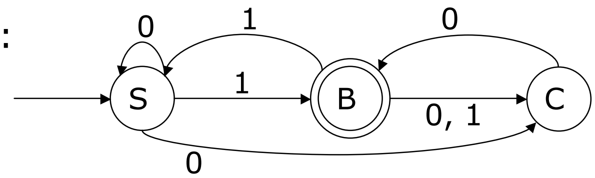
These types are called the “Chomsky Hierarchy”, after linguist Noam Chomsky, who first named them.

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

- Type 3 productions are of one of two types:
  - B □ □C, where B □ A, □ □ □
  - B □ □
- To prove this result, identify the states of a NFA with auxiliaries in the grammar. Assume a single start state and no □-transitions (WLOG!).
  - B □ □C is a production if state B goes to state C via symbol □.
  - B □ □ 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 □ □, which corresponds to the NFA being in an accepting state.

Example: NFA vs. Grammar

NFA:



Grammar:

- Start symbol is S
- Productions:

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

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

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

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 ▯ ST if R is a product of sub-expressions S and T
  - R ▯ S and R ▯ T if R is a union of sub-expressions S and T
  - R ▯ SR and R ▯ ▯ if R is S*
  - R ▯ ▯ if ▯ ▯ ▯
  - R ▯ ▯ if R is ▯
  - none if R is ∅

Example

- Regular expression: 0((10)* ▯ 01)*
  - R ▯ ST // R = 0((10)* ▯ 01)* = ST
  - S ▯ 0 // S = 0
  - T ▯ VT // T = ((10)* ▯ 01)* = V*
  - T ▯ ▯
  - V ▯ W // V = (10)* ▯ 01 = W ▯ X
  - V ▯ X
  - W ▯ YW // W = (10)* = Y*
  - W ▯ ▯
  - Y ▯ 10 // Y = 10
  - X ▯ 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 □.
- By a **substitution map**, we mean a function that assigns to each element of a string from an alphabet □.
- Example: □ = {0, 1}, □ = {a, b, c},
  s(0) = ab, s(1) = cbaba.
- We can "extend" s to map any language over 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.

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

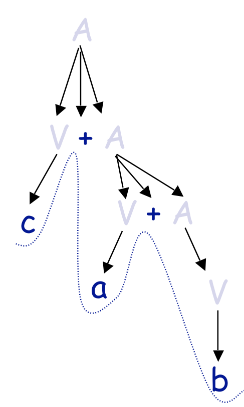
- This is due to the ***pumping lemma for context-free languages***.

- Before presenting this, we need to review **derivation trees**.

Derivation Tree Visualization

A → V | V + A
V → a | b | 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 derviations using .
- 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 unambigous 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

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

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 $u = a^n b^n c^n$. Let uvwxy = $a^n b^n c^n$.

- One of v and x is not ε. Suppose it's v. The other case is symmetric. By the PL, uv^2wx^2y 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.

Using Chomsky Normal Form for the proof of the 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 □ BC, where B and C are auxiliaries

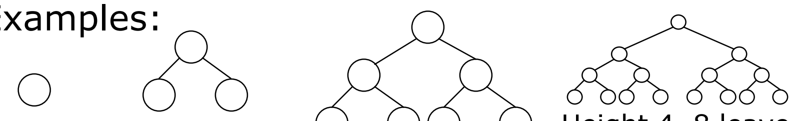
  A □ □, where □ □ □
- For every context-free language not containing □, is generated by some grammar in Chomsky Normal Form.
- Assume this for now, see book for proof.

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:



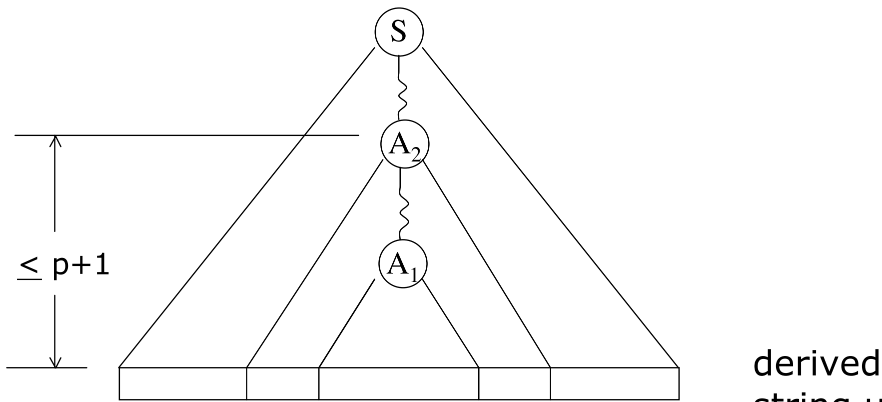
Height 1, 1 leaf
Height 2, 2 leaves
Height 3, 4 leaves
Height 4, 8 leaves

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)

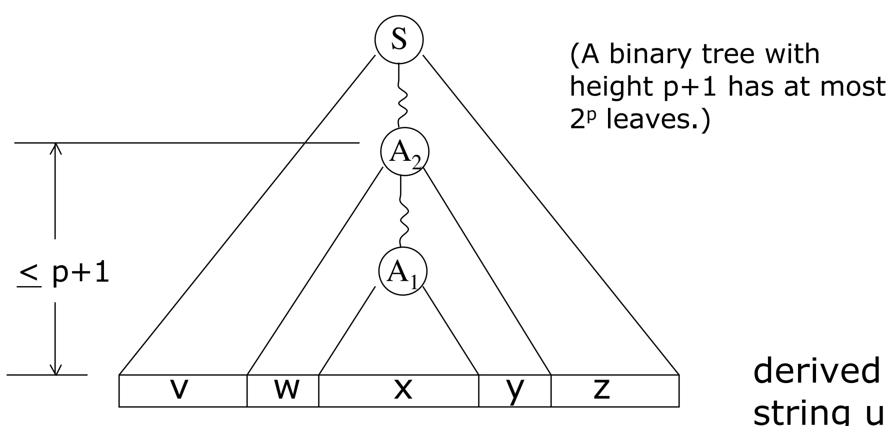
- Here is a picture of our derivation tree:



Proof of the Pumping Lemma (3)

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

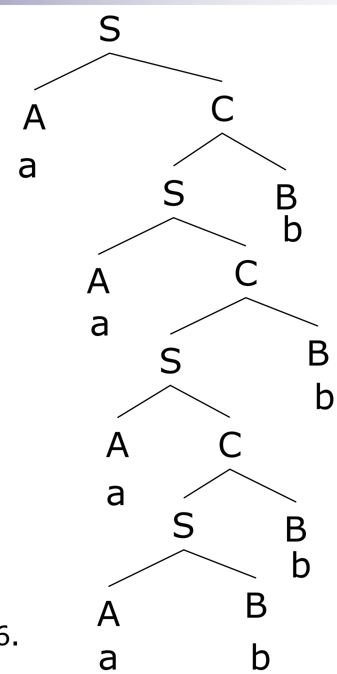
(A binary tree with height p+1 has at most 2^p leaves.)



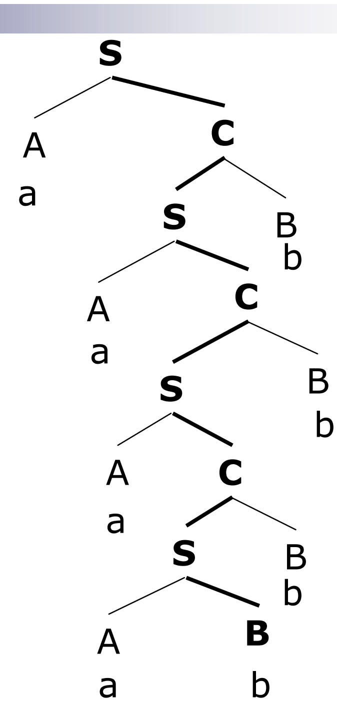
We see that $|wxy| \leq 2^p$. Also, $|wy| > 0$.

Example

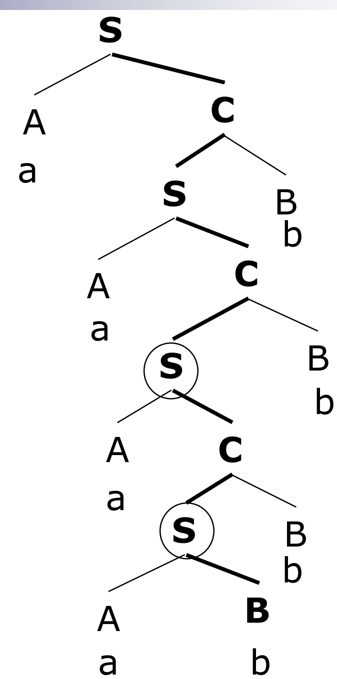
- S → AC
- S → AB
- C → SB
- A → a
- B → b
- Derviation tree for aaaabbbb
  - Note: We can illustrate the principle even tho' this string is not length or longer 16.



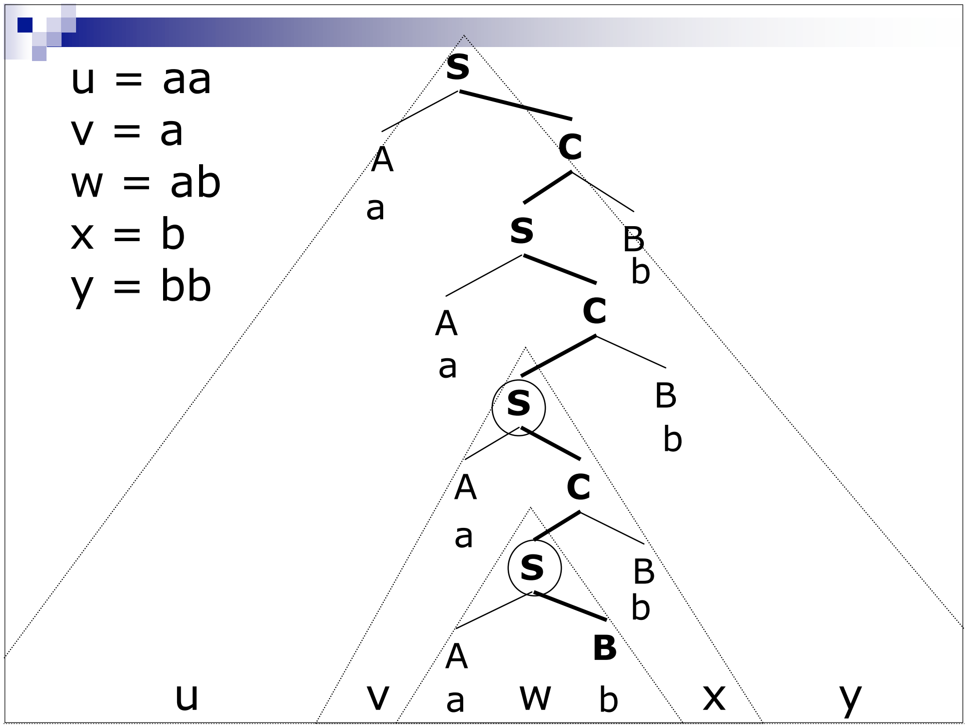
A Long Path



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: $aaa^k abb^k bb$ 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:
  - {$a^k b^k c^m$ | k, m □ □}
  - {$a^m b^k c^k$ | k, m □ □}
- However, their intersection is:
  - {$a^k b^k c^k$ | k □ □}
- 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 next.