

# Grammars and Their Languages

Robert M. Keller  
Harvey Mudd College  
April 2010

## Review of Grammars

- In 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 the family of “L systems”.

## Definition of Grammars

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

## Definition of Grammars

- A grammar consists of 4 parts:
  - Terminal alphabet  $\Sigma$
  - Auxiliary alphabet  $A$  (such that  $A \cap \Sigma = \emptyset$ )
  - A finite set of productions  $\rightarrow$
  - Start symbol  $S \in A$
- 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  $\epsilon$  because there is nothing to rewrite in that case.

## Derivation in a Grammar

- A **derivation** in of a grammar is a sequence of strings  $x_0 \Rightarrow x_1 \Rightarrow x_2 \Rightarrow \dots$  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 The latter must have been replaced by rewriting according to productions first.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  $\rightarrow$ :
    - $S \rightarrow FE$      $E \rightarrow ABCE$      $E \rightarrow \epsilon$
    - $BA \rightarrow AB$      $CA \rightarrow AC$      $CB \rightarrow BC$
    - $FA \rightarrow a$      $aA \rightarrow aa$      $aB \rightarrow ab$
    - $bB \rightarrow bb$      $bC \rightarrow bc$      $cC \rightarrow cc$
  - Start symbol S
- A derivation (underlines show symbols replaced):  
 $S \Rightarrow FE \Rightarrow FABCE \Rightarrow FABCBCE \Rightarrow FABCBBC \Rightarrow FABACBC \Rightarrow FAABCBC \Rightarrow FAABBCC \Rightarrow aABBCC \Rightarrow aaBBCC \Rightarrow aabBCC \Rightarrow aabbCC \Rightarrow aabbcC \Rightarrow aabbbc$

## Types of Grammars

- Grammars are classified by the kinds of productions they allow, from least 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  $\epsilon$ , or  $\alpha A$  where A is an auxiliary and  $\alpha \in \Sigma$

See: [http://en.wikipedia.org/wiki/Chomsky\\_hierarchy](http://en.wikipedia.org/wiki/Chomsky_hierarchy)

## 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

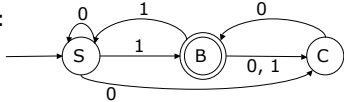
These types are called the "Chomsky Hierarchy", after linguist Noam Chomsky, who first named them.

## 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  $B, C \in A$ ,  $\alpha \in \Sigma$
  - $B \rightarrow \epsilon$
- To prove this result, identify the states of a NFA with auxiliaries in the grammar. Assume a single start state and no  $\epsilon$ -transitions (WLOG!).
  - $B \rightarrow \alpha C$  is a production if state B goes to state C via symbol  $\alpha$ .
  - $B \rightarrow \epsilon$  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:

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

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

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  $S^*$
  - $R \rightarrow \sigma$  if  $\sigma \in \Sigma$
  - $R \rightarrow \epsilon$  if R is  $\epsilon$
  - none if R is  $\emptyset$

## Example

- Regular expression:  $0((10)^* \cup 01)^*$ 
  - $R \rightarrow ST$       //  $R = 0((10)^* \cup 01)^* = ST$
  - $S \rightarrow 0$       //  $S = 0$
  - $T \rightarrow VT$       //  $T = ((10)^* \cup 01)^* = V^*$
  - $T \rightarrow \epsilon$
  - $V \rightarrow W$       //  $V = (10)^* \cup 01 = W \cup X$
  - $V \rightarrow X$
  - $W \rightarrow YW$       //  $W = (10)^* = Y^*$
  - $W \rightarrow \epsilon$
  - $Y \rightarrow 10$       //  $Y = 10$
  - $X \rightarrow 01$       //  $X = 01$
- Note the connection with solving language equations.

## Summary

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

## 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  $\Sigma$ .
- By a **substitution map**, we mean a function that assigns to each element of  $\Sigma$  a string from an alphabet  $\Delta$ .
- Example:  $\Sigma = \{0, 1\}$ ,  $\Delta = \{a, b, c\}$ ,  
 $s(0) = ab$ ,  $s(1) = cbaba$ .
- We can "extend"  $s$  to map any language over  $\Sigma$  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:  $A = V(+V)^*$

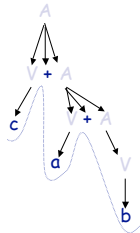
There are languages that are type 1 but not type 2.

- $\{a^k b^k c^k \mid k > 0\}$  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  $\Rightarrow$ .
- 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

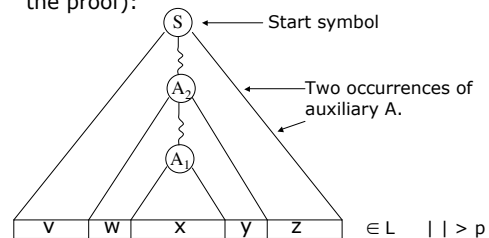
- Let  $L$  be a context-free language. Then there is a number  $p$  such that if  $u \in L$  and  $|u| > p$  then there are strings  $v, w, x, y, z$ , such that
  - $u = vwxyz$
  - $|wy| > 0$  (at least one of  $w$  or  $y$  is non-empty)
  - $|wxy| \leq p$
  - $(\forall m \geq 0) v w^m x y^m z \in L$
- We can use a Skolem function to give  $p$ . We name it  $\text{pump}(L)$ , the **pumping length** of  $L$ .

## Pumping Lemma Expressed in Logic

$\forall L (\text{context-free}(L) \rightarrow$   
 $(\exists p \forall u ($   
 $(u \in L \wedge |u| > p) \rightarrow$   
 $(\exists v \exists w \exists x \exists y \exists z$   
 $u = vwxyz$   
 $\wedge |wy| > 0$   
 $\wedge |wxy| \leq p$   
 $\wedge \forall m (m \geq 0 \rightarrow v w^m x y^m z \in L))))))$

## 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  $\rightarrow$   $L$  pumpable), thus ( $L$  not pumpable  $L \rightarrow$  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.
- They do not violate the PL, as we can see by having the pumping length be one 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  $vwxyz$ .
- One of  $w$  and  $y$  is not  $\epsilon$ . Suppose  $w \neq \epsilon$ . The other case is symmetric. By the lemma,  $vw^2xy^2z$  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.

## More detailed case analysis

- $a^p b^p c^p = vwxyz \in L$ , where  $|wxy| \leq p$  and  $|wy| > 0$ .
- Due to  $|wxy| \leq p$ ,  $wxy$  can contain at most two letters, not three.
- Further,  $wxy \in \{a\}^* \{b\}^*$  or  $wxy \in \{b\}^* \{c\}^*$ .
- If  $wxy \in \{a\}^* \{b\}^*$ , then  $vw^2xy^2z$  has the wrong number of  $c$ 's to be in  $L$ .
- If  $wxy \in \{b\}^* \{c\}^*$ , then  $vw^2xy^2z$  has the wrong number of  $a$ 's to be in  $L$ .
- So in either case we have a contradiction.

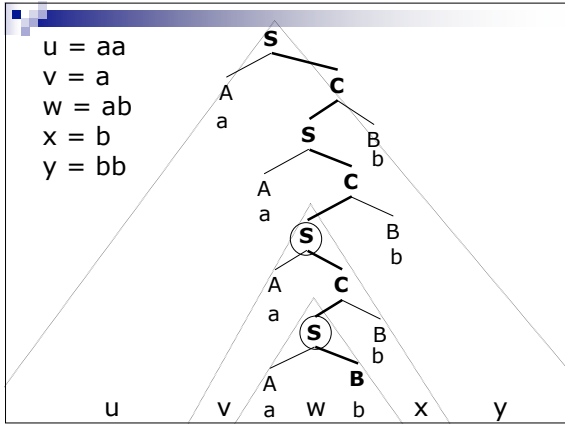
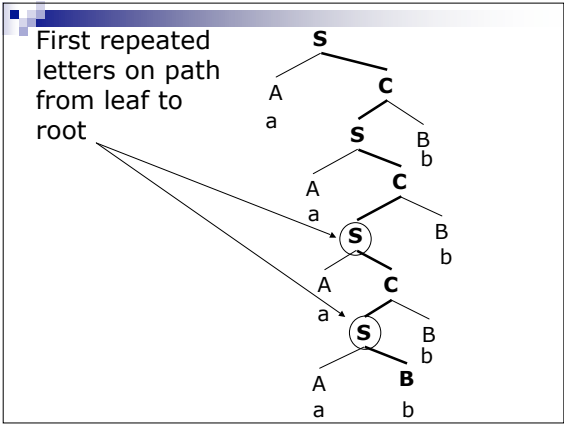
## Proof of the CFL Pumping Lemma

- 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:  $S \rightarrow \epsilon$  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





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:
  - $\{a^k b^k c^m \mid k, m > 0\}$
  - $\{a^m b^k c^k \mid k, m > 0\}$
- 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  $L \cap R$  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.
- The following 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  $\{0, 1\}^* - \{ww \mid w \in \{0, 1\}^*\}$

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

$A \rightarrow CAC \mid 0$

$B \rightarrow CBC \mid 1$

$C \rightarrow 0 \mid 1$

- Correctness of this grammar 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.
- Intersecting the original 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  $p$  be the number that exists by the pumping lemma. Select  $u = 0^p 1^p 0^p 1^p$  and decompose  $u$  in  $uvwxy$  where  $|vx| > 0$  and  $|vwx| \leq p$ .
- Show that  $uv^2wx^2y$  cannot be of the form  $0^r 1^s 0^r 1^s$ .