

# CS112: Theory of Computation (LFA)

## Lecture13: Non-context-free Languages

Dumitru Bogdan

Faculty of Computer Science  
University of Bucharest

May 22, 2021

# Table of contents

1. Various stuff
2. Previously on CS112
3. Context setup
4. Pumping Lemma
5. Examples
6. Generating Harmony using Stochastic CFG
7. Conway's Game of Life

## Section 1

Various stuff

# Various stuff

- Lecture 14: Exam related
- Tomorrow 10-11: office hours (<- instead of lab)
- Homework submission form: <https://forms.gle/NqwbGr5Ni6J9srro9>
- Read: <https://fs.blog/2021/05/advice-for-young-scientists/>

## Section 2

Previously on CS112

# CFG Formal definition

## Definition

A **context-free grammar** is a 4-tuple  $(V, \Sigma, R, S)$  where:

- $V$  is a finite set called the **variables**
- $\Sigma$  is a finite set, disjoint from  $V$  called the **terminals**
- $R$  is a finite set of rules, with each rule being a variable and a string of variables and terminals
- $S \in V$  is the start variable

# Chomsky normal form

- Working with context-free grammars, it is often convenient to have them in simplified form
- The simplest and most useful forms is called **the Chomsky normal form**
- Chomsky normal form is useful in giving algorithms for working with context-free grammars

# CNF Formal definition

## Definition

A context-free grammar is in Chomsky normal form if every rule is of the form

$$A \rightarrow BC$$

$$A \rightarrow a$$

where  $a$  is any terminal and  $A$ ,  $B$ , and  $C$  are any variables — except that  $B$  and  $C$  may not be the start variable. In addition, we allow the rule  $S \rightarrow \epsilon$ , where  $S$  is the start variable



# Equivalence PDA - CFG

## Theorem

*A language is context free if and only if some pushdown automaton recognizes it*

As usual for "if and only if" theorems, we have two directions to prove. We formulate both directions as lemmas.

## Lemma

*If a language is context free, then some pushdown automaton recognizes it*

## Lemma

*If a pushdown automaton recognizes some language, then it is context free*

## Section 3

### Context setup

# Context setup

Corresponding to Sipser 2.3

# Context setup

- In this lecture we present a technique for proving that certain languages are not context free
- Recall that one previous lecture we introduced the pumping lemma for showing that certain languages are not regular
- Here we study a similar pumping lemma for context-free languages. It states that every context-free language has a special value called the pumping length such that all longer strings in the language can be "pumped"
- In this case the meaning of pumped is a bit more complex. It means that the string can be divided into five parts so that the second and the fourth parts may be repeated together any number of times and the resulting string still remains in the language

## Section 4

### Pumping Lemma

# Pumping lemma for context-free languages

## Theorem

*If  $A$  is a context-free language, then there is a number  $p$  (the pumping length) where, if  $s$  is any string in  $A$  of length at least  $p$ , then  $s$  may be divided into five pieces  $s = uvxyz$  satisfying the conditions*

1. *for each  $i \geq 0$ ,  $uv^i xy^i z \in A$*
2.  $|vy| > 0$
3.  $|vxy| \leq p$

- When  $s$  is being divided into  $uvxyz$ , condition 2 says that either  $v$  or  $y$  is not the empty string. Otherwise the theorem would be trivially true
- Condition 3 states that the pieces  $v, x$  and  $y$  together have length at most  $p$ . This technical condition sometimes is useful in proving that certain languages are not context free

# Proof idea

- Let  $A$  be a CFL and let  $G$  be a CFG that generates it. We must show that any sufficiently long string  $s$  in  $A$  can be pumped and remain in  $A$ . The idea behind this approach is simple
- Let  $s$  be a very long string in  $A$ . (later will be clear what we mean by "very long"). Because  $s$  is in  $A$ , it is derivable from  $G$  and so has a parse tree
- The parse tree for  $s$  must be very tall because  $s$  is very long. That is, the parse tree must contain some long path from the start variable at the root of the tree to one of the terminal symbols at a leaf
- On this long path, some variable symbol  $R$  must repeat because of the pigeonhole principle
- This repetition allows us to replace the subtree under the second occurrence of  $R$  with the subtree under the first occurrence of  $R$  and still get a legal parse tree
- So, we may cut  $s$  into five pieces  $uvxyz$  as the figure indicates, and we may repeat the second and fourth pieces and obtain a string still in the language
- In other words we have  $uv^i xy^i z$  in  $A$  for any  $i \geq 0$

# Proof idea

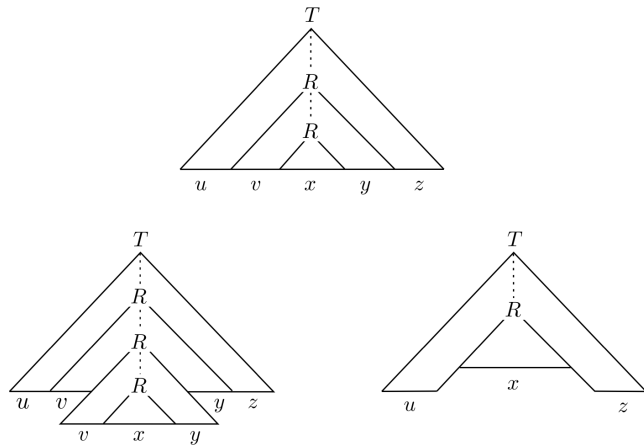


Figure: Surgery on parse trees



# Formal proof

## Proof.

Let  $G$  be a CFG for CFL  $A$ . Let  $b$  be the maximum number of symbols in the right-hand side of a rule, we assume at least 2. In any parse tree using this grammar, we know that a node can have no more than  $b$  children. In other words, at most  $b$  leaves are 1 step from the start variable; at most  $b^2$  leaves are within 2 steps of the start variable; and at most  $b^h$  leaves are within  $h$  steps of the start variable. So, if the height of the parse tree is at most  $h$ , the length of the string generated is at most  $b^h$ . Conversely, if a generated string is at least  $b^h + 1$  long, each of its parse trees must be at least  $h + 1$  high.

Say  $|V|$  is the number of variables in  $G$ . We set  $p$ , the pumping length, to be  $b|V| + 1$ . Now if  $s$  is a string in  $A$  and its length is  $p$  or more, its parse tree must be at least  $|V| + 1$  high, because  $b^{|V|+1} \geq b|V| + 1$ . □

# Formal proof

## Proof.

To see how to pump any such string  $s$ , let  $\tau$  be one of its parse trees. If  $s$  has several parse trees, choose  $\tau$  to be a parse tree that has the smallest number of nodes. We know that  $\tau$  must be at least  $|V| + 1$  high, so its longest path from the root to a leaf has length at least  $|V| + 1$ . That path has at least  $|V| + 2$  nodes; one at a terminal, the others at variables. Hence that path has at least  $|V| + 1$  variables. With  $G$  having only  $|V|$  variables, some variable  $R$  appears more than once on that path. For convenience later, we select  $R$  to be a variable that repeats among the lowest  $|V| + 1$  variables on this path. □

# Formal proof

## Proof.

We divide  $s$  into  $uvxyz$  as seen in previous image. Each occurrence of  $R$  has a subtree under it, generating a part of the string  $s$ . The upper occurrence of  $R$  has a larger subtree and generates  $vxy$ , whereas the lower occurrence generates just  $x$  with a smaller subtree. Both of these subtrees are generated by the same variable, so we may substitute one for the other and still obtain a valid parse tree. Replacing the smaller by the larger repeatedly gives parse trees for the strings  $uv^i xy^i z$  at each  $i > 1$ . Replacing the larger by the smaller generates the string  $uxz$ . That establishes condition 1 of the lemma. □

# Formal proof

## Proof.

To get condition 2, we must be sure that  $v$  and  $y$  are not both  $\epsilon$ . If they were, the parse tree obtained by substituting the smaller subtree for the larger would have fewer nodes than  $\tau$  does and would still generate  $s$ . This result isn't possible because we had already chosen  $\tau$  to be a parse tree for  $s$  with the smallest number of nodes. That is the reason for selecting  $\tau$  in this way.

To get condition 3, we need to be sure that  $vxy$  has length at most  $p$ . In the parse tree for  $s$  the upper occurrence of  $R$  generates  $vxy$ . We chose  $R$  so that both occurrences fall within the bottom  $|V| + 1$  variables on the path, and we chose the longest path in the parse tree, so the subtree where  $R$  generates  $vxy$  is at most  $|V| + 1$  high. A tree of this height can generate a string of length at most  $b^{|V|+1} = p$ . □

## Section 5

### Examples

# Example

- Let us use the pumping lemma to show that the language  $B = \{a^n b^n c^n \mid n \geq 0\}$  is not context free
- We assume that  $B$  is a CFL and obtain a contradiction
- Let  $p$  be the pumping length for  $B$  that is guaranteed to exist by the pumping lemma
- Select the string  $s = a^p b^p c^p$ . Clearly  $s$  is a member of  $B$  and of length at least  $p$
- The pumping lemma states that  $s$  can be pumped, but we show that it cannot
- We show that no matter how we divide  $s$  into  $uvxyz$ , one of the three conditions of the lemma is violated

# Example

- Condition 2 stipulates that either  $v$  or  $y$  is nonempty. Then we consider one of two cases, depending on whether substrings  $v$  and  $y$  contain more than one type of alphabet symbol
  1. When both  $v$  and  $y$  contain only one type of alphabet symbol,  $v$  does not contain both  $a$ 's and  $b$ 's or both  $b$ 's and  $c$ 's, and the same holds for  $y$ . In this case, the string  $uv^2xy^2z$  cannot contain equal numbers of  $a$ 's,  $b$ 's, and  $c$ 's. Therefore, it cannot be a member of  $B$ . That violates condition 1 of the lemma and is thus a contradiction
  2. When either  $v$  or  $y$  contains more than one type of symbol,  $uv^2xy^2z$  may contain equal numbers of the three alphabet symbols but not in the correct order. Hence it cannot be a member of  $B$  and a contradiction occurs.
- Since one of these cases must occur and because both cases result in a contradiction, a contradiction is unavoidable
- So the assumption that  $B$  is a CFL must be false. Thus we have proved that  $B$  is not a CFL

# Example

- Let  $C = \{a^i b^j c^k \mid 0 \leq i \leq j \leq k\}$ . We use the pumping lemma to show that  $C$  is not a CFL
- Assume that  $C$  is a CFL and obtain a contradiction
- Let  $p$  be the pumping length given by the pumping lemma. We use the string  $s = a^p b^p c^p$  that we used earlier, but this time we must "pump down" as well as "pump up"
- The pumping lemma states that  $s$  can be pumped, but we show that it cannot
- We show that no matter how we divide  $s$  into  $uvxyz$ , one of the three conditions of the lemma is violated
- Condition 2 stipulates that either  $v$  or  $y$  is nonempty. Then we consider one of two cases, depending on whether substrings  $v$  and  $y$  contain more than one type of alphabet symbol



# Example

1. When both  $v$  and  $y$  contain only one type of alphabet symbol,  $v$  does not contain both  $a$ 's and  $b$ 's or both  $b$ 's and  $c$ 's, and the same holds for  $y$ . Now, we must analyze the situation more carefully to show that  $s$  cannot be pumped. Observe that because  $v$  and  $y$  contain only one type of alphabet symbol, one of the symbols  $a$ ,  $b$ , or  $c$  doesn't appear in  $v$  or  $y$ . We further subdivide this case into three subcases according to which symbol does not appear:
  - 1.1 The  $a$ 's do not appear. Then we try pumping down to obtain the string  $uv^0xy^0z = uxz$ . That contains the same number of  $a$ 's as  $s$  does, but it contains fewer  $b$ 's or fewer  $c$ 's. Therefore, it is not a member of  $C$ , and a contradiction occurs
  - 1.2 The  $b$ 's do not appear. Then either  $a$ 's or  $c$ 's must appear in  $v$  or  $y$  because both can't be the empty string. If  $a$ 's appear, the string  $uv^2xy^2z$  contains more  $a$ 's than  $b$ 's, so it is not in  $C$ . If  $c$ 's appear, the string  $uv^0xy^0z$  contains more  $b$ 's than  $c$ 's, so it is not in  $C$ . Either way, a contradiction occurs
  - 1.3 The  $c$ 's do not appear. Then the string  $uv^2xy^2z$  contains more  $a$ 's or more  $b$ 's than  $c$ 's, so it is not in  $C$ , and a contradiction occurs

# Example

2. When either  $v$  or  $y$  contains more than one type of symbol,  $uv^2xy^2z$  will not contain the symbols in the correct order. Hence it cannot be a member of  $C$ , and a contradiction occurs
- Thus we have shown that  $s$  cannot be pumped in violation of the pumping lemma and that  $C$  is not context free

## Section 6

### Generating Harmony using Stochastic CFG

## Section 7

### Conway's Game of Life