CS 3719 (Theory of Computation and Algorithms) – Lecture 14

Antonina Kolokolova*

February 8 2011

1 Pumping lemma for CFLs

Just like we proved that certain languages are not regular by defining a property (pumping lemma) which all regular languages satisfy, and then showing that some languages don't, we will define a same kind of property for context-free languages.

Consider a simple grammar $A \to aAb, A \to c$. It is clear that it generates a non-regular language: a pumping lemma proof would say that either a's or b's would have to be pumped separately, or c would be repeated multiple times, resulting in strings not in the language. However, if we could pump both a's and b's at the same time, this would work.

With this intuition, we state the pumping lemma for context-free languages.

Lemma 11. *Let L be a context-free language. Then there exists a natural number p such that $\forall s \in L$, $|s| \geq p$, $s = uvxyz$ where*

1) $\forall i \geq 0, uv^ixy^iz \in L$

2) $|vy| > 0$

3) $|vxy| \leq p$.

That is, if regular languages could be split into three parts so that the middle part could be iterated without creating strings not in the language, for CFLs the split is into 5 parts so that the middle and the sides stay, and the 2nd and 4th parts are iterated simultaneously. For the grammar we just saw u can consist of several a's, and y of the same number of b's; the middle part x contains the c.

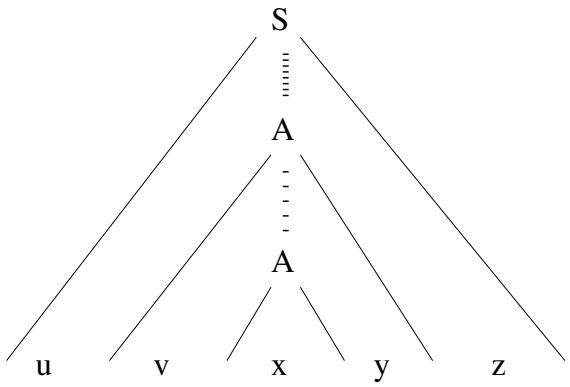
*The material in this set of notes came from many sources, in particular "Introduction to Theory of Computation" by Sipser and course notes of U. of Toronto CS 364.

Proof. In the proof for regular languages, we found a repeating part by looking at the states of the automaton and finding a repeating states. Here, we will do the same, but with parse trees.

Here is an idea. Consider a parse tree which is so high that its height is more than the number of variables in the grammar. Why should such a tree exist? It is because if the language is infinite, then there are infinitely many parse trees, at least one distinct tree for a different string in the language (remember that a string is a sequence of leaves of the parse tree). There are finitely many rules, and each rule is of finite length, so the only way to have infinitely many trees is to allow them to grow to an arbitrary height.

Let's look more carefully at the possible parse trees. Suppose that the number of symbols (variables and terminals) in the body of the longest rule is d: then our tree would be d-ary (every node having at most d children). We need to know what is the shortest string guaranteed to have a parse tree of height at least $|V| + 1$. Recall that the longest possible string produced by a d-ary tree of height h has length d^h: it corresponds to a sequence of leaves of a complete d-ary tree of height h. Therefore, if we take a string of length at least $d^{|V|} + 1$ it is guaranteed to have all parse trees of height at least $|V| + 1$. For convenience, we can choose even larger p; let $p = d^{|V|+1}$.

Now, why would we be interested in trees of this form? Consider the largest path from the root to a leaf in this tree. Since the height of the tree is greater than the number of variables, there would be a repeating variable, say A, on this path. A variable occurring in a parse tree means that a rule with this variable at the head was applied. But now notice that there is no way in the grammar to differentiate these occurrences and say how many there should be; thus, a subtree rooted at the first occurrence of A can be replaced by subtree in the second occurrence, and vice verse. The first one gives us $i = 0$ from the pumping lemma, and iterating the second one gives any i we desire (see the figure).



To check the second condition, suppose that we start with the smallest possible parse tree for the string. Then doing the substitution of the first occurrence by the second would give us an even smaller tree, which is a contradiction.

Finally, let's prove the third condition. Starting from the bottom of the tree, choose the first time from the bottom a variable repeats; call that variable A. A subtree under A is of height at most $|V| + 1$, so it generates a string of length at most p. Here, the second occurrence of A generates vxy where the bottom occurrence generates x. □

Now, let's see how this lemma can be used to show that some languages are not context-free.

Example 1. The language $\{a^n b^n c^n\}$ is not context-free.

Suppose it is. Then let p be the pumping length, and take $s = a^p b^p c^p$. Now, just like in the proof that $\{0^n 1^n\}$ is not regular, no matter how we split $s = uvxyz$, either v and y are monochromatic and thus repeating them disrupts the equality of the numbers, or they are not monochromatic and repeating them disrupts the order.

Now, recall that the language $\{a^n b^n c^m\}$ is context-free: it is a special case of $\{a^i b^j c^k | i = j \text{ or } i = k\}$. Also, $\{a^n b^m c^n\}$ is context-free, for the same reason. But their intersection is exactly $\{a^n b^n c^n\}$, which we just shown to be not context-free! This leads us to a surprising corollary:

Corollary 12. *The class of context-free languages are not closed under intersection.*

Example 2. The language $\{ww | w \in \{0,1\}^*\}$ is not context-free.

Take $s = 0^p 1^p 0^p 1^p$. By the 3rd condition of the pumping lemma, that $|vxy| \leq p$, the sequence vxy overlaps at most two out of four blocks in our string. Either it is part of $0^p 1^p$; then repeating any part of it will disrupt the equality with the second half of s. Or it overlaps the $1^p 0^p$ piece in the middle; but then either v or y are monochromatic: say y is, and it consists of just 0s. But now repeating y will increase the number of 0s in the second copy of w, but not the first, again disrupting the equality.