Not All Languages Are Context-Free

What we plan to do. Many programming-related languages are context-free, in particular, many languages which are regular and thus, cannot be recognized by a finite automaton. In this lecture, we will prove, however, that there exists languages which are not context-free. The proof will be based on a special Pumping Lemma for context-free grammars (CFG).

Pumping Lemma for CFG: formulation. For every context-free language L there exists an integer p such that every word w from L whose length is at least p can be represented as $w = uvxyz$, where:

- $\text{len}(vxy) \le p$;
- $\text{len}(vy) > 0$, and
- for all natural numbers i, the word $uv^i xy^i z$ also belongs to the language L.

Note. Similarly to the Pumping Lemma for finite automata, we can use abbreviations and get a much shorter description:

$$\forall L\,((L \text{ is CFG}) \to \forall w \in L\,(\text{len}(w) \ge p \to$$

$$\exists u, v, x, y, z\,(w = uvxyz\,\&\,\text{len}(vxy) \le p\,\&\,\text{len}(vy) > 0\,\&\,\forall i\,(uv^i xy^i z \in L)))).$$

Proof: main idea. A CFG is determined by a finite number of rules. When we describe each derivation as a tree, each rule means going from a parent node to children nodes. For example, in the rule $S \to aSb$, the parent node S leads to three children nodes: a, S, and b.

Let us denote the maximal length of the right-hand side of these rules by B. This means that in the derivation tree, every node has no more than B children. Let us denote the total number of variables by V.

As a result of derivation, we have a word that consists only of terminal symbols. So, each branch of the derivation tree must end in a terminal symbol. Once we have a terminal symbol, the branch cannot continue: all rules transform a variable, not a terminal symbol. So, in each branch, only the last symbol is a terminal symbol, all other symbols are variables.

So, if all symbols along the branch are different, then – since we have only V different variables – we can have no more than $V + 1$ symbols in the branch: no more than V variables and one terminal symbol. If some branch has more

than $V + 1$ symbols, this means that some variable repeats at least twice along this branch – this is the same Pigeonhole Principle that we used in the finite-automata Pumping Lemma.

On the top level of the derivation tree – we will call it Level 0 – we have a single node, corresponding to the starting variable. On the next Level 1, we have no more than B nodes. If we only have the two levels, then each branch consists of only one edge, i.e., each branch has length 1. So, with branches of length 1, we can get only words of length $\leq B$.

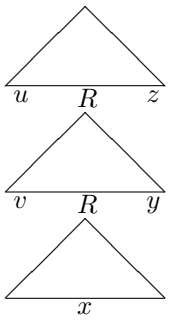
On the following Level 2, each of these nodes has $\leq B$ children, so we have no more than B^2 nodes. This means that with branches of length ≤ 2, we can have only words of length $\leq B^2$. In general, on Level k, we can have no more than B^k nodes. So, by using branches of length $\leq k$, we can generate only words of length $\leq B^k$.

In particular, on Level V, we can have no more than B^V nodes. So, by using branches of length $\leq V$, we can generate only words of length $\leq B^V$.

Thus, if we have a word whose length is greater than or equal to $B^V + 1$, then at least one branch must have length greater than V, and thus, this branch must contain more than $V + 1$ symbols. As we have argued, this means that at least two variables along this branch must be the same. This value $B^V + 1$ will be our value p.

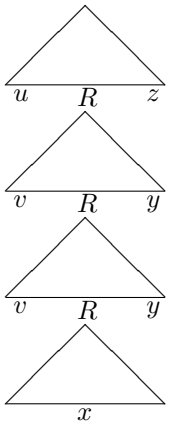
So, for each word from the language L whose length is at least p, we have repeating variables on one the branches of this word's derivation tree. There may be several such repeating variables, we pick the pair in which the top repeating variable is at the lowest possible position in the tree. If there are several such pairs, we pick one of them. Let us denote this repeating variable by R. Then:

- u is all the terminal symbols which are before the first of the two occurrences of the selected repeated variable to the left;
- v is all the terminal symbols which are between the two occurrences of the selected repeated variable to the left;
- x is all the terminal symbols which are below the second occurrence;
- y is all the terminal symbols which are between the two occurrences of the selected repeated variable to the right; and
- z is all the terminal symbols which are before the first of the two occurrences of the selected repeated variable to the right.



In this derivation, we get the word $uvxyz$.

According to this picture, if we are in the state R, then the middle part of the derivation leads us back to R. So, we can repeat this part, and get the following derivation:



This way, we derive the word $uvvxyyz$, i.e., by using notation v^2 for vv and y^2 for yy, the word uv^2xy^2z. Similarly, if we repeat the same portion three time, we derive the word uv^3xy^3z, etc.

How to actually represent a word w as a concatenation $uvxyz$: algorithm. We find the lowest pair of occurrences of the same variable on the same branch. Then:

- u is all the terminal symbols which are before the first of the two occurrences of the selected repeated variable to the left;
- v is all the terminal symbols which are between the two occurrences of the selected repeated variable to the left;
- x is all the terminal symbols which are below the second occurrence;

- y is all the terminal symbols which are between the two occurrences of the selected repeated variable to the right; and
- z is all the terminal symbols which are after the first of the two occurrences of the selected repeated variable to the right.

Note: it is important to make sure that:

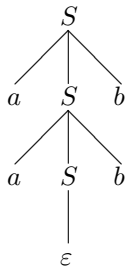
- we select the *lowest* pair and that
- the selected occurrences are *on the same branch*, and *not* on different branches.

First example. Let us start with the very first example with which we started studying context-free grammars: the grammar consisting of two rules $S \to \varepsilon$ and $S \to aSb$ that generates all the words from the language

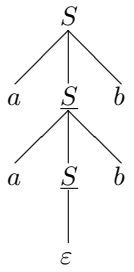
$$\{a^n b^n : n = 0, 1, 2, \ldots\} = \{\Lambda, ab, aabb, aaabbb, \ldots\}$$

In this grammar, we have $V = 1$ variable, and the largest length of the right-hand side is $B = 3$. Thus, here, $p = B^V + 1 = 3^1 + 1 = 4$. This means that the subdivision corresponding to the Pumping Lemma is possible for all the words of length 4 or larger.

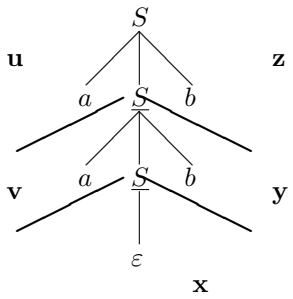
In particular, the derivation of the word $aabb$ of length 4 in this grammar has the following form:



In this derivation, the lowest two occurrences of the same variable on the same branch are the two lowest occurrences of the variable S:



So, we can find u, v, x, y, and z as follows:



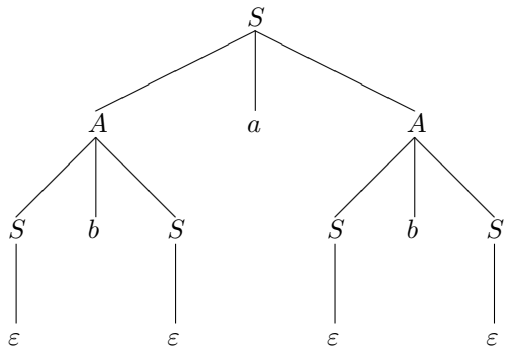
So here:

$$u = a, \quad v = a, \quad x = \varepsilon, \quad y = b, \quad z = b.$$

Second example. Let us consider another CFG, with rules $S \to AaA$, $A \to SbS$, and $S \to \varepsilon$.

In this case, we have $V = 2$ variables, and the largest length of the right-hand side is $B = 3$, so $p = B^V + 1 = 3^2 + 1 = 10$. This means that every word from this language whose length is at least 10 can be represented as a concatenation $uvxyz$ that satisfies the properties described in the Pumping Lemma.

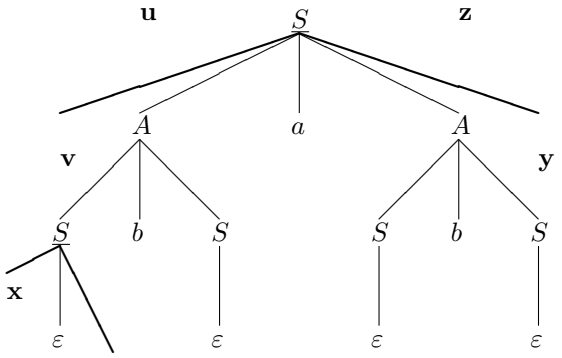
Such a representation is possible also for some shorter words. For example, let us consider the word bab which can be generated by this grammar as follows:



In this derivation, the variable A appears twice – but in different branches! So, the lowest pair of occurrences of the same variable on the same branch is the variable S at the top and one of the occurrences of this variables lower in the tree.

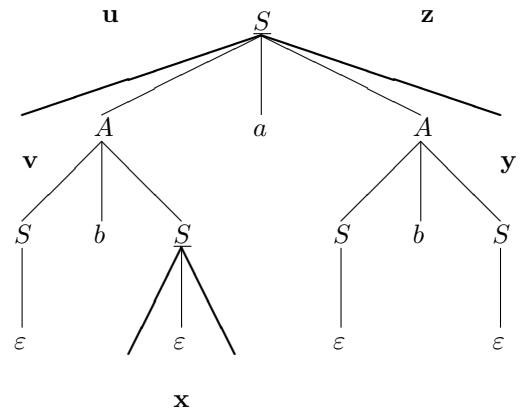
Depending on which of the four lower occurrences of the variable S you select, you get four different answers – and they are all correct. Let us show two of them, obtained by selecting the first two occurrences.

If we select the first of the four lower occurrences of S, we get the following picture:



In this case, $u = v = x = z = \varepsilon$ and $y = bab$. So, e.g., the word $uvvxyyz$ takes the form $babbab$.

If we select the second of the four lower occurrences of the variable S, then we get a different picture:



In this case, $u = x = z = \varepsilon$, $v = b$, and $y = ab$. So, e.g., the word $uvvxyyz$ takes the form $bbabab$.

Practice. Try finding the words u, v, x, y, and z for other words and other grammars.

Proof that there exists a language which is not context-free. Let us prove that the following language is not context-free:

$$L = \{a^n b^n c^n : n = 0, 1, 2, \ldots\} = \{\Lambda, abc, aabbcc, \ldots\}.$$

The proof will be by contradiction. Let us assume that this language is context-free. Then, by pumping lemma, there exists an integer p such that every word w from the language L whose length is at least p can be represented as $w = uvxyz$, where:

- $\text{len}(vxy) \leq p$;
- $\text{len}(vy) > 0$, and
- for all natural numbers i, the word $uv^i xy^i z$ also belongs to the language L.

Let us take the word

$$w = a^p b^p c^p = a \ldots ab \ldots bc \ldots c,$$

where each of the three letters is repeated p times. The length of this word – i.e., the number of symbols in this word – is equal to $3p$. Clearly, $3p \geq p$, so, according to the Pumping Lemma, this word can be described as $uvxyz$ with the above properties.

Where can the central part vxy of this word be? We know that the length $\text{len}(vxy)$ of this part cannot exceed p. Thus, it cannot contain all three types of letters: a's, b's, and c's – since then it would have to include all p letters b + additional a and c symbols, so its length would have been larger than p. So, there are only 5 cases remaining for the location of the part vxy:

1. it can be in the a's;
2. it can be in a's and b's;
3. it can be in b's;
4. it can be in b's and c's; and
5. it can be in c's.

Let us consider these cases one by one.

Case 1. If vxy is in a's, this means that the parts v and y contain only a's. Thus, when we pump, i.e., when we go from the original word $uvxyz$ to the word $uv^2xy^2z = uvvxyyz$, we add a's – but we do not add any b's or c's. In the original word $w = a^p b^p c^p$, there were exactly as many a's as b's and as c's. When we add more a's, the balance is disrupted. Since the language L only contains the words which has the exact same number of a's, b's, and c's, the word $uvvxyyz$ cannot belong to the language L.

Case 2. If vxy is in a's and in b's, this means that the parts v and y contain only a's and b's. Thus, when we pump, i.e., when we go from the original word $uvxyz$ to the word $uv^2xy^2z = uvvxyyz$, we add a's and b's – but we do not add any c's. In the original word $w = a^p b^p c^p$, there were exactly as many a's as b's and as c's. When we add more a's and b's, the balance is disrupted. Since the

language L only contains the words which has the exact same number of a's, b's, and c's, the word $uvvxyyz$ cannot belong to the language L.

Case 3. If vxy is in b's, this means that the parts v and y contain only b's. Thus, when we pump, i.e., when we go from the original word $uvxyz$ to the word $uv^2xy^2z = uvvxyyz$, we add b's – but we do not add any a's or c's. In the original word $w = a^p b^p c^p$, there were exactly as many a's as b's and as c's. When we add more b's, the balance is disrupted. Since the language L only contains the words which has the exact same number of a's, b's, and c's, the word $uvvxyyz$ cannot belong to the language L.

Case 4. If vxy is in b's and c's, this means that the parts v and y contain only b's and c's. Thus, when we pump, i.e., when we go from the original word $uvxyz$ to the word $uv^2xy^2z = uvvxyyz$, we add b's and c's – but we do not add any a's. In the original word $w = a^p b^p c^p$, there were exactly as many a's as b's and as c's. When we add more b's and c's, the balance is disrupted. Since the language L only contains the words which has the exact same number of a's, b's, and c's, the word $uvvxyyz$ cannot belong to the language L.

Case 5. If vxy is in c's, this means that the parts v and y contain only c's. Thus, when we pump, i.e., when we go from the original word $uvxyz$ to the word $uv^2xy^2z = uvvxyyz$, we add c's – but we do not add any a's or b's. In the original word $w = a^p b^p c^p$, there were exactly as many a's as b's and as c's. When we add more c's, the balance is disrupted. Since the language L only contains the wordswhich has the exact same number of a's, b's, and c's, the word $uvvxyyz$ cannot belong to the language L.

So, in all five cases, we get a contradiction. This means that the original assumption – that the language L is context-free – is wrong. Thus, the language $\{a^n b^n c^n\}$ is not context-free.

First comment. In this class, the main purpose of this section is to show that there are languages that are not context-free. There are many different languages for which it is proven that they are not context-free, and for different languages the corresponding proofs use different ideas. To simplify the exposition, we follow the example of most textbooks and provide the simplest of such proofs – the proof based on the arguments about the order of balance between different symbols.

- For some not-context-free languages – e.g., for the language $\{a^n b^n c^n\}$ – such arguments are sufficient to prove that the language is not context-free.
- For other languages, more complex arguments are needed to show that the language is not context-free.

Second comment. In this proof, we got a contradiction in all possible cases. What if we have a contradiction only in some cases?

Well, the answer is simple: then, we do not have a proof, and the corresponding language can actually be context-free.

For example, let us consider the language

$$L = \{a^n b^n c^m : n = 0, 1, 2, \ldots, m = 0, 1, 2, \ldots\} =$$

$$\{\Lambda, c, cc, \ldots, ab, abc, abcc, \ldots, aabb, aabbc, aabbcc, \ldots\}.$$

For this language, we can similarly take $w = a^p b^p c^p$, and similarly conclude that the fragment vxy cannot be in a's, cannot be in b's, but we cannot get a contradiction if this fragment is in c's.

And this language is actually context-free: it can be generated by the rules $S \to AB$, $A \to \varepsilon$, $A \to aAb$, $B \to \varepsilon$, and $B \to cB$. For example, the word $aabbc$ can be generated as follows:

$$\underline{S} \to \underline{A}B \to a\underline{A}bB \to aa\underline{A}bbB \to aabb\underline{B} \to aabbc\underline{B} \to aabbc.$$

Practice. Try the same arguments to prove that other similar languages are not context-free, e.g., the languages

$$\{a^n b^n c^n d^n, n = 0, 1, 2, \ldots\} = \{\Lambda, abcd, aabbccdd, \ldots\}$$

or

$$\{a^n b^{2n} c^{3n}, n = 0, 1, 2, \ldots\} = \{\Lambda, abbccc, aabbbbccccccc, \ldots\}.$$