Monash University
Faculty of Information Technology
2^{nd} Semester 2017

FIT2014 Tutorial 3 Pumping Lemma, and Context Free Languages

SOLUTIONS

Although you may not need to do all the many exercises in this Tutorial Sheet, it is still important that you attempt all the main questions and a selection of the Supplementary Exercises.

Even for those Supplementary Exercises that you do not attempt seriously, you should still give some thought to how to do them before reading the solutions.

1.

(a) $(n+1)^2 - n^2 = 2n+1$ which increases as n increases.

(b) Suppose L is regular. Let N be the number of states of a finite automaton that recognises L. Let n be any positive integer such that $n^2 > N$. Then the string $\mathbf{a}^{n^2} \in L$, and has length $> N$, so by the Pumping Lemma for Regular Languages, there exist strings x, y, z such that $w = xyz$, and the length of xy is $\leq N$, and $y \neq \varepsilon$, and $xy^iz \in L$ for all $i \in \mathbb{N} \cup \{0\}$.

Let ℓ be the length of y. (Note, $\ell \geq 1$.) Then the length of xy^iz is $n^2 + (i-1)\ell$. So the strings xy^iz have lengths $n^2, n^2+\ell, n^2+2\ell, n^2+3\ell, \ldots$. This is an infinite arithmetic sequence of numbers, with each consecutive pair being ℓ apart. But the sequence of lengths of strings in L is the sequence of square numbers, and by part (a) the gaps between them increase, eventually exceeding any specific number you care to name. So there comes a point where the gaps exceed ℓ, and some of the numbers $n^2+(i-1)\ell$ fall between two squares. When that happens, $xy^iz \notin L$, a contradiction.

(c) Let L_1 be the language of all strings consisting entirely of 1s. This language is regular (since the regular expression 11* describes it).

Let L_2 be the language of binary string representations of adjacency matrices of graphs.

Assume L_2 is regular. Then $L_1 \cap L_2$ is also regular, since the class of regular languages is closed under intersection.

But $L_1 \cap L_2$ is the language of all adjacency matrices consisting entirely of 1s. Such matrices always exist, for any n: they are the adjacency matrices of the complete graphs. (The *complete graph* on n vertices has every pair of vertices adjacent.) So $L_1 \cap L_2$ is actually the language L. But we have just shown in (b) that this is not regular. So we have a contradiction.

Hence L_2 is not regular.

Regular languages are also closed under a transformation called *homomorphism*. We haven't covered it this unit, and it's not in the textbook by Sipser, but many books on the formal language theory do cover it. For example, it is treated briefly in *Introduction to Languages and the Theory of Computation (4th edn.)* by John C. Martin, McGraw-Hill, New York, 2011, exercise 3.53, pp. 127–128. Closure of regular languages under homomorphism (and inverse homomorphism) enables a very direct proof that the language of adjacency matrices of graphs is not regular (assuming we've already done part (b) above).

2.

(a) We prove that CENTRAL-ONE is not regular.

Assume, by way of contradiction, that CENTRAL-ONE is regular. Then it has a Finite Automaton, by Kleene's Theorem. Let N be the number of states of this FA.

Let $w := 0^N 1 0^N$. By the Pumping Lemma for Regular Languages, we can partition w into strings x, y, z with y nonempty (so $w = xyz$) such that $|xy| \leq N$ and for all $i \geq 0$ we have $xy^iz \in$ CENTRAL-ONE. Now, the condition $|xy| \leq N$ means that y lies in the first half of w, before the central 1. It therefore consists only of zeros. Repetition of y, in forming the string xy^iz (with $i \geq 2$), will not affect the second half (the central 1 and beyond), but it will make the first half longer (since y is nonempty). So the number of zeros before the solitary 1 no longer equals the number of zeros after the 1. So the string no longer belongs to CENTRAL-ONE. This contradicts the conclusion of the Pumping Lemma. So our assumption, that CENTRAL-ONE is regular, must be wrong. Therefore CENTRAL-ONE is not regular.

Other choices of w are possible. In particular, we could have had any bits at all after the first 1, as long as there are N of them. So, we could have

$$w = 0^N 1 \underbrace{{}^0/_1\,{}^0/_1 \cdots {}^0/_1}_{\text{any } N \text{ bits}};$$

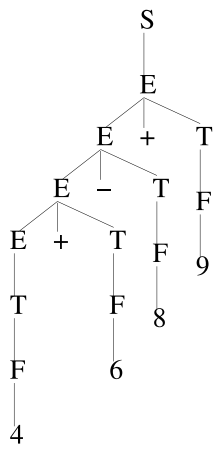
for example, we could have $w = 0^N 1 1^N$. The important thing is that y falls within the stretch of N zeros at the start, so that repeating y pushes the first 1 so far along that the middle bit is now a 0.

(b) We prove that CENTRAL-ONE is context-free by giving a context-free grammar for it.[1]

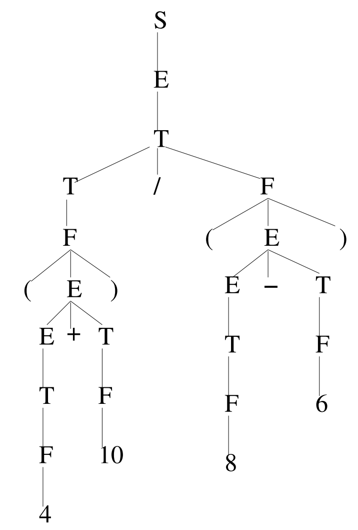
$$\begin{array}{lcl} \mathbf{S} & \to & \mathbf{ASA} \\ \mathbf{A} & \to & \mathbf{0} \\ \mathbf{A} & \to & \mathbf{1} \\ \mathbf{S} & \to & \mathbf{1} \end{array}$$

3.

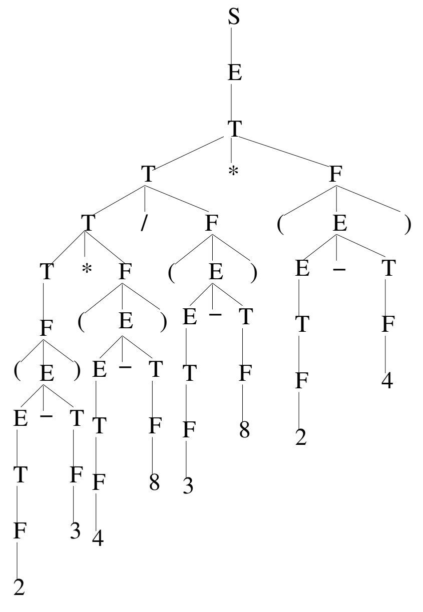
1(i)



1(ii)



1(iii)



[1]Thanks to FIT2014 tutor Srinibas Swain for advising a correction to an earlier version.

4. (a)

$$
\begin{array}{rcll}
\text{S} & \longrightarrow & \text{Subject can Verbs} & (1)\\
\text{S} & \longrightarrow & \text{SubjectWithToBe Adjective} & (2)\\
\text{Subject} & \longrightarrow & \text{Tom} \mid \text{I} \mid \text{Tom and I} & (3)\\
\text{Verbs} & \longrightarrow & \text{OneVerb} \mid \text{Verbs and OneVerb} & (4)\\
\text{OneVerb} & \longrightarrow & \text{hop} \mid \text{run} \mid \text{stop} \mid \text{dig} & (5)\\
\text{SubjectWithToBe} & \longrightarrow & \text{I am} \mid \text{Tom is} & (6)\\
\text{Adjective} & \longrightarrow & \text{big} \mid \text{six} & (7)
\end{array}
$$

Variations on this are possible.

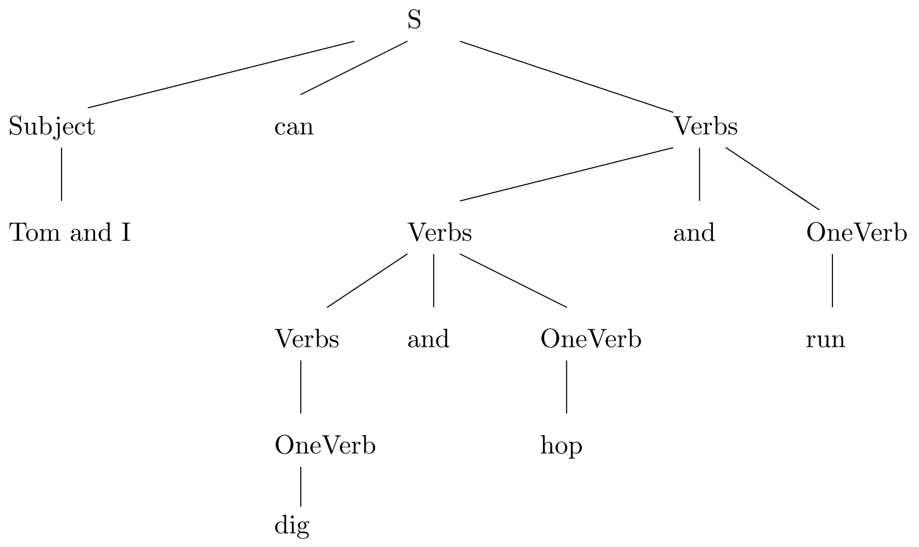
(b)

Derivation:

$$
\begin{array}{rcll}
\text{S} & \Longrightarrow & \text{Subject can Verbs} & \text{(Rule 1)}\\
 & \Longrightarrow & \text{Subject can Verbs and OneVerb} & \text{(Rule 4, 2nd part)}\\
 & \Longrightarrow & \text{Tom and I can Verbs and OneVerb} & \text{(Rule 3, 3rd part)}\\
 & \Longrightarrow & \text{Tom and I can Verbs and run} & \text{(Rule 5, 2nd part)}\\
 & \Longrightarrow & \text{Tom and I can Verbs and OneVerb and run} & \text{(Rule 4, 2nd part)}\\
 & \Longrightarrow & \text{Tom and I can OneVerb and OneVerb and run} & \text{(Rule 4, 1st part)}\\
 & \Longrightarrow & \text{Tom and I can dig and OneVerb and run} & \text{(Rule 5, 4th part)}\\
 & \Longrightarrow & \text{Tom and I can dig and hop and run} & \text{(Rule 5, 1st part).}
\end{array}
$$

Variations on the order of application of these rules are possible. (For some grammars, variation in the set of rules actually used in a derivation is possible too, though that is not the case here.) The above derivation is neither a leftmost nor a rightmost derivation.

Parse tree:



Note that "Tom and I" here is a string of three terminals. Although "Tom" starts with a capital letter, it is not intended to be a non-terminal; rather, it just follows the rule in English that proper names start with a capital letter. Similarly, "I" is capitalised because (as a pronoun) it always is in English, but is still a terminal here.

5.

Note that, in this grammar, ϵ is actually a symbol of the grammar, namely the symbol used in regular expressions to match just the empty string. This is not the same thing as the empty string itself, usually denoted ε.

(a) (i)

Leftmost:

$\mathbf{S} \Rightarrow \mathbf{E} \Rightarrow \mathbf{E} \cup \mathbf{T} \Rightarrow \mathbf{T} \cup \mathbf{T} \Rightarrow \mathbf{TF} \cup \mathbf{T} \Rightarrow \mathbf{FF} \cup \mathbf{T} \Rightarrow \mathbf{F}^*\mathbf{F} \cup \mathbf{T} \Rightarrow \mathbf{a}^*\mathbf{F} \cup \mathbf{T} \Rightarrow \mathbf{a}^*\mathbf{F}^* \cup \mathbf{T} \Rightarrow \mathbf{a}^*\mathbf{b}^* \cup \mathbf{T} \Rightarrow \mathbf{a}^*\mathbf{b}^* \cup \mathbf{TF} \Rightarrow \mathbf{a}^*\mathbf{b}^* \cup \mathbf{FF} \Rightarrow \mathbf{a}^*\mathbf{b}^* \cup \mathbf{F}^*\mathbf{F} \Rightarrow \mathbf{a}^*\mathbf{b}^* \cup \mathbf{b}^*\mathbf{F} \Rightarrow \mathbf{a}^*\mathbf{b}^* \cup \mathbf{b}^*\mathbf{F}^* \Rightarrow \mathbf{a}^*\mathbf{b}^* \cup \mathbf{b}^*\mathbf{a}^*$

Rightmost:

$\mathbf{S} \Rightarrow \mathbf{E} \Rightarrow \mathbf{E} \cup \mathbf{T} \Rightarrow \mathbf{E} \cup \mathbf{TF} \Rightarrow \mathbf{E} \cup \mathbf{TF}^* \Rightarrow \mathbf{E} \cup \mathbf{Ta}^* \Rightarrow \mathbf{E} \cup \mathbf{Fa}^* \Rightarrow \mathbf{E} \cup \mathbf{F}^*\mathbf{a}^* \Rightarrow \mathbf{E} \cup \mathbf{b}^*\mathbf{a}^* \Rightarrow \mathbf{T} \cup \mathbf{b}^*\mathbf{a}^* \Rightarrow \mathbf{TF} \cup \mathbf{b}^*\mathbf{a}^* \Rightarrow \mathbf{TF}^* \cup \mathbf{b}^*\mathbf{a}^* \Rightarrow \mathbf{Tb}^* \cup \mathbf{b}^*\mathbf{a}^* \Rightarrow \mathbf{Fb}^* \cup \mathbf{b}^*\mathbf{a}^* \Rightarrow \mathbf{F}^*\mathbf{b}^* \cup \mathbf{b}^*\mathbf{a}^* \Rightarrow \mathbf{a}^*\mathbf{b}^* \cup \mathbf{b}^*\mathbf{a}^*$

(a) (ii)

Leftmost:

$\mathbf{S} \Rightarrow \mathbf{E} \Rightarrow \mathbf{T} \Rightarrow \mathbf{F} \Rightarrow \mathbf{F}^* \Rightarrow (\mathbf{E})^* \Rightarrow (\mathbf{E} \cup \mathbf{T})^* \Rightarrow (\mathbf{T} \cup \mathbf{T})^* \Rightarrow (\mathbf{TF} \cup \mathbf{T})^* \Rightarrow (\mathbf{FF} \cup \mathbf{T})^* \Rightarrow (\mathbf{aF} \cup \mathbf{T})^* \Rightarrow (\mathbf{aa} \cup \mathbf{T})^* \Rightarrow (\mathbf{aa} \cup \mathbf{TF})^* \Rightarrow (\mathbf{aa} \cup \mathbf{FF})^* \Rightarrow (\mathbf{aa} \cup \mathbf{bF})^* \Rightarrow (\mathbf{aa} \cup \mathbf{bb})^*$

Rightmost:

$\mathbf{S} \Rightarrow \mathbf{E} \Rightarrow \mathbf{T} \Rightarrow \mathbf{F} \Rightarrow \mathbf{F}^* \Rightarrow (\mathbf{E})^* \Rightarrow (\mathbf{E} \cup \mathbf{T})^* \Rightarrow (\mathbf{E} \cup \mathbf{TF})^* \Rightarrow (\mathbf{E} \cup \mathbf{Tb})^* \Rightarrow (\mathbf{E} \cup \mathbf{Fb})^* \Rightarrow (\mathbf{E} \cup \mathbf{bb})^* \Rightarrow (\mathbf{T} \cup \mathbf{bb})^* \Rightarrow (\mathbf{TF} \cup \mathbf{bb})^* \Rightarrow (\mathbf{Ta} \cup \mathbf{bb})^* \Rightarrow (\mathbf{aa} \cup \mathbf{bb})^*$

(a) (iii)

Leftmost:

$\mathbf{S} \Rightarrow \mathbf{E} \Rightarrow \mathbf{T} \Rightarrow \mathbf{TF} \Rightarrow \mathbf{TFF} \Rightarrow \mathbf{FFF} \Rightarrow (\mathbf{E})\mathbf{FF} \Rightarrow (\mathbf{E} \cup \mathbf{T})\mathbf{FF} \Rightarrow (\mathbf{T} \cup \mathbf{T})\mathbf{FF} \Rightarrow (\mathbf{F} \cup \mathbf{T})\mathbf{FF} \Rightarrow (\mathbf{a} \cup \mathbf{T})\mathbf{FF} \Rightarrow (\mathbf{a} \cup \mathbf{F})\mathbf{FF} \Rightarrow (\mathbf{a} \cup \varepsilon)\mathbf{FF} \Rightarrow (\mathbf{a} \cup \varepsilon)(\mathbf{E})\mathbf{F} \Rightarrow (\mathbf{a} \cup \varepsilon)(\mathbf{E} \cup \mathbf{T})\mathbf{F} \Rightarrow (\mathbf{a} \cup \varepsilon)(\mathbf{T} \cup \mathbf{T})\mathbf{F} \Rightarrow (\mathbf{a} \cup \varepsilon)(\mathbf{F} \cup \mathbf{T})\mathbf{F} \Rightarrow (\mathbf{a} \cup \varepsilon)(\mathbf{b} \cup \mathbf{T})\mathbf{F} \Rightarrow (\mathbf{a} \cup \varepsilon)(\mathbf{b} \cup \mathbf{F})\mathbf{F} \Rightarrow (\mathbf{a} \cup \varepsilon)(\mathbf{b} \cup \varepsilon)\mathbf{F} \Rightarrow (\mathbf{a} \cup \varepsilon)(\mathbf{b} \cup \varepsilon)(\mathbf{E}) \Rightarrow (\mathbf{a} \cup \varepsilon)(\mathbf{b} \cup \varepsilon)(\mathbf{E} \cup \mathbf{T}) \Rightarrow (\mathbf{a} \cup \varepsilon)(\mathbf{b} \cup \varepsilon)(\mathbf{T} \cup \mathbf{T}) \Rightarrow (\mathbf{a} \cup \varepsilon)(\mathbf{b} \cup \varepsilon)(\mathbf{F} \cup \mathbf{T}) \Rightarrow (\mathbf{a} \cup \varepsilon)(\mathbf{b} \cup \varepsilon)(\mathbf{a} \cup \mathbf{T}) \Rightarrow (\mathbf{a} \cup \varepsilon)(\mathbf{b} \cup \varepsilon)(\mathbf{a} \cup \mathbf{F}) \Rightarrow (\mathbf{a} \cup \varepsilon)(\mathbf{b} \cup \varepsilon)(\mathbf{a} \cup \varepsilon)$

Rightmost:

$\mathbf{S} \Rightarrow \mathbf{E} \Rightarrow \mathbf{T} \Rightarrow \mathbf{TF} \Rightarrow \mathbf{T}(\mathbf{E}) \Rightarrow \mathbf{T}(\mathbf{E} \cup \mathbf{T}) \Rightarrow \mathbf{T}(\mathbf{E} \cup \mathbf{F}) \Rightarrow \mathbf{T}(\mathbf{E} \cup \varepsilon) \Rightarrow \mathbf{T}(\mathbf{T} \cup \varepsilon) \Rightarrow \mathbf{T}(\mathbf{F} \cup \varepsilon) \Rightarrow \mathbf{T}(\mathbf{a} \cup \varepsilon) \Rightarrow \mathbf{TF}(\mathbf{a} \cup \varepsilon) \Rightarrow \mathbf{T}(\mathbf{E})(\mathbf{a} \cup \varepsilon) \Rightarrow \mathbf{T}(\mathbf{E} \cup \mathbf{T})(\mathbf{a} \cup \varepsilon) \Rightarrow \mathbf{T}(\mathbf{E} \cup \mathbf{F})(\mathbf{a} \cup \varepsilon) \Rightarrow \mathbf{T}(\mathbf{E} \cup \varepsilon)(\mathbf{a} \cup \varepsilon) \Rightarrow \mathbf{T}(\mathbf{T} \cup \varepsilon)(\mathbf{a} \cup \varepsilon) \Rightarrow \mathbf{T}(\mathbf{F} \cup \varepsilon)(\mathbf{a} \cup \varepsilon) \Rightarrow \mathbf{T}(\mathbf{b} \cup \varepsilon)(\mathbf{a} \cup \varepsilon) \Rightarrow \mathbf{F}(\mathbf{b} \cup \varepsilon)(\mathbf{a} \cup \varepsilon) \Rightarrow (\mathbf{E})(\mathbf{b} \cup \varepsilon)(\mathbf{a} \cup \varepsilon) \Rightarrow (\mathbf{E} \cup \mathbf{T})(\mathbf{b} \cup \varepsilon)(\mathbf{a} \cup \varepsilon) \Rightarrow (\mathbf{E} \cup \mathbf{F})(\mathbf{b} \cup \varepsilon)(\mathbf{a} \cup \varepsilon) \Rightarrow (\mathbf{E} \cup \varepsilon)(\mathbf{b} \cup \varepsilon)(\mathbf{a} \cup \varepsilon) \Rightarrow (\mathbf{T} \cup \varepsilon)(\mathbf{b} \cup \varepsilon)(\mathbf{a} \cup \varepsilon) \Rightarrow (\mathbf{F} \cup \varepsilon)(\mathbf{b} \cup \varepsilon)(\mathbf{a} \cup \varepsilon) \Rightarrow (\mathbf{a} \cup \varepsilon)(\mathbf{b} \cup \varepsilon)(\mathbf{a} \cup \varepsilon)$

(b)

The generated language includes the strings $\mathbf{(}^n\mathbf{a)}^n$, for all positive integers n. This has n left parentheses, followed by the letter **a** (though any letter will do here), followed by n right parentheses. The proof that the generated language is not regular is related to the proof that $\{\mathrm{a}^n\mathrm{b}^n \mid n \in \mathbb{N}\}$ is not regular. But some extra thought is needed, and we make use of what the Pumping Lemma tells us about where y is located within w.

Assume, by way of contradiction, that the generated language is regular. Then it has an FA (by Kleene's Theorem). Let N be the number of states of the FA. Let w be the string $\mathbf{(}^N\mathbf{a)}^N$.

(Note that we are being fussy in our choice of w. We are not going to reason about *every* long enough string in the language. Rather, we devise a particular type of string in the language which, if long enough, will enable the Pumping Lemma to do its job. Working out what string to use can involve some exploration.)

This string w has more than N letters, so the Pumping Lemma for Regular Languages applies. So w can be divided up into substrings x, y, z such that $w = xyz$, and $|xy| \le N$, and y is nonempty, and for all $i \in \mathbb{N} \cup \{0\}$, the string xy^iz is also in the language.

Since $|xy| \le N$, the string y must lie within the first N letters of w, so y lies within the stretch of opening parentheses before the letter **a** in the middle. (*This is a key step.* See how the fact that $|xy| \le N$ — one of the conclusions of the Pumping Lemma — enables us to gain some control over where y lies within w. Had we not used this fact, we could not have ruled out the possibility that y might consist of the single letter **a** in the middle. If that were the case, then repeating y would produce other strings in the language, so we would not get the hoped-for contradiction.)

Repeating y, to give a string xy^iz (with $i \ge 2$), must give another string in the language, by the Pumping Lemma. But, in this case, repeating y gives a string with more opening parentheses than it had before (using the fact that y is nonempty), and therefore more opening parentheses than closing parentheses (since the number of closing parentheses has not changed). So the resulting string is not in the language, since for every string in the generated language, any parentheses are in matching pairs. This is a contradiction. So our initial assumption, that the language is regular, is incorrect. Therefore the language is not regular.

Alternative approach, suggested by FIT2014 tutor Nathan Companez:

First show that the language of strings of the form $\mathbf{(}^n\mathbf{a)}^n$ is not regular. That proof would be similar to the one given here, or to the proof that the language of strings of the form $\mathbf{a}^n\mathbf{b}^n$ is not regular.

Assume the language generated by the given grammar is regular.

Consider the language of all strings containing a single **a** and otherwise consisting entirely of parentheses. The parentheses are not necessarily matching, and the numbers of parentheses on each side of the **a** do not have to be the same. This language is regular, since it is described by the regular expression $(\text{“}\mathbf{(}\text{”} \cup \text{“}\mathbf{)}\text{”})^*\mathbf{a}(\text{“}\mathbf{(}\text{”} \cup \text{“}\mathbf{)}\text{”})^*$. (Note that, here, the quoted parentheses are symbols in the alphabet for the language; they are not playing the special grouping role that parentheses play in forming regular expressions. But the unquoted parentheses are playing that grouping role.)

Observe that the intersection of this regular language with the generated language (assumed regular) is just the language of strings $\mathbf{(}^n\mathbf{a)}^n$. Now, the class of regular languages is closed under intersection. Therefore the language of strings $\mathbf{(}^n\mathbf{a)}^n$ must also be regular. But we already know it's not regular, so we have a contradiction. So our original assumption, that the generated language is regular, is wrong. Therefore the generated language is not regular.

6. Let $\sigma \Rightarrow \sigma_1 \Rightarrow \cdots \Rightarrow \sigma_n = w$ be a derivation of a string w of length n. We prove by induction on n that there is a leftmost derivation of w from σ of length n.

Base case:

Suppose $n = 1$. Since w is a string in the CFL, it consists only of terminal symbols. The single production used to produce w from σ can replace only one nonterminal symbol. So σ has only one nonterminal symbol. This single nonterminal symbol is the leftmost nonterminal symbol in σ. So this derivation of w from σ is a leftmost derivation.

Inductive step:

Suppose that, whenever there is a derivation of $n-1$ steps, there is also a leftmost derivation of $n-1$ steps.

Consider the "subderivation" of w from σ_1:

$$\sigma_1 \Rightarrow \sigma_2 \Rightarrow \cdots \Rightarrow \sigma_{n-1} \Rightarrow \sigma_n = w.$$

This has $n-1$ steps. So, by the inductive hypothesis, it has a leftmost derivation

$$\sigma_1 \overset{L}{\Rightarrow} \tau_2 \overset{L}{\Rightarrow} \cdots \overset{L}{\Rightarrow} \tau_{n-1} \overset{L}{\Rightarrow} \sigma_n = w, \tag{8}$$

using $\overset{L}{\Rightarrow}$ to denote an application of some production rule to the leftmost nonterminal symbol of the current string. Note that, although the beginning and end strings of this derivation — σ_1 and σ_n, respectively — are the same in the two derivations, the intermediate strings may well be different: σ_2 may be different from τ_2, and likewise for σ_3 and τ_3, …, and for σ_{n-1} and τ_{n-1}. [2]

If $\sigma \Rightarrow \sigma_1$ is also a leftmost production — i.e., $\sigma \overset{L}{\Rightarrow} \sigma_1$, then we can put this together with (8) to get a leftmost derivation of w from σ, using n steps, so we are done.

So assume that $\sigma \Rightarrow \sigma_1$ is not a leftmost production.

Let X be the leftmost nonterminal symbol of σ, and let Y be the nonterminal symbol on the left-hand side of the production used in the first step. The string σ has the general form

$$\sigma \quad = \quad \ldots \textit{terminals} \ldots X \ldots \textit{any symbols} \ldots Y \ldots$$

and the first step $\sigma \Rightarrow \sigma_1$ applies some production $Y \to \beta$ to give

$$\sigma_1 \quad = \quad \ldots \textit{terminals} \ldots X \ldots \textit{any symbols} \ldots \beta \ldots.$$

Since the rest of the derivation (i.e., of w from σ_1) is a leftmost derivation, the next step is to apply some production $X \to \alpha$ to obtain

$$\tau_2 \quad = \quad \ldots \textit{terminals} \ldots \alpha \ldots \textit{any symbols} \ldots \beta \ldots.$$

We can swap the order in which these two productions are applied. So we do $X \to \alpha$ first, followed by $Y \to \beta$. These two steps take us from σ to σ_2 in two steps, using a different intermediate string:

$$\begin{aligned}
\sigma &= \ldots \textit{terminals} \ldots X \ldots \textit{any symbols} \ldots Y \ldots, \\
\sigma_1' &= \ldots \textit{terminals} \ldots \alpha \ldots \textit{any symbols} \ldots Y \ldots, \\
\tau_2 &= \ldots \textit{terminals} \ldots \alpha \ldots \textit{any symbols} \ldots \beta \ldots.
\end{aligned}$$

This gives a new derivation of w from σ in n steps, whose first step is a leftmost derivation, but whose second step might not be:

$$\sigma \overset{L}{\Rightarrow} \sigma_1' \Rightarrow \tau_2 \overset{L}{\Rightarrow} \cdots \overset{L}{\Rightarrow} \tau_{n-1} \overset{L}{\Rightarrow} \sigma_n = w.$$

Now, the derivation of w from σ_1' may not be a leftmost derivation. But it has $n-1$ steps. So, by the inductive hypothesis, there is a leftmost derviation of w from σ_1' in $n-1$ steps:

$$\sigma_1' \overset{L}{\Rightarrow} \sigma_2' \overset{L}{\Rightarrow} \cdots \overset{L}{\Rightarrow} \sigma_{n-1}' \overset{L}{\Rightarrow} \sigma_n = w.$$

(Note that the intermediate strings $\sigma_2', \ldots, \sigma_{n-1}'$ may be different to the intermediate strings $\sigma_2, \ldots, \sigma_{n-1}$ in our first derivation of w from σ_1, and also from the intermediate strings $\tau_2, \ldots, \tau_{n-1}$ in our *left-most* derivation of w from σ_1. That's fine. It's still a derivation of w (a.k.a. σ_n) from σ_1' — in fact, a leftmost one.)

Appending this leftmost derivation of w from σ_1' to our one-step leftmost derivation $\sigma \overset{L}{\Rightarrow} \sigma_1'$ gives an n-step leftmost derivation of w from σ:

$$\sigma \overset{L}{\Rightarrow} \sigma_1' \overset{L}{\Rightarrow} \sigma_2' \overset{L}{\Rightarrow} \cdots \overset{L}{\Rightarrow} \sigma_{n-1}' \overset{L}{\Rightarrow} \sigma_n = w.$$

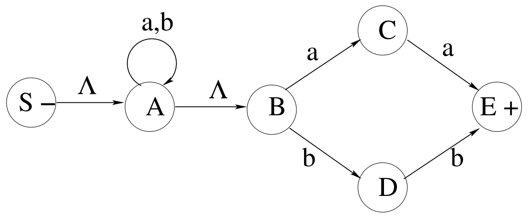
This completes the inductive step.

Therefore, by the principle of mathematical induction, the claim holds for all n.

[2]Thanks to FIT2014 tutor Michael Gill for suggesting a correction to this part of the argument.

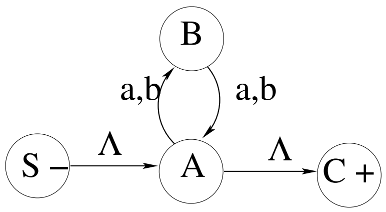
7.

(i)



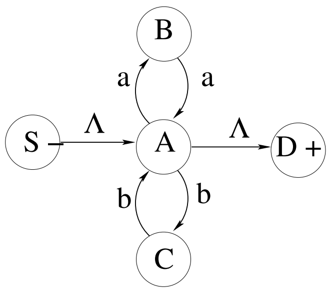
$$\begin{array}{lcl}
\mathbf{S} & \rightarrow & \mathbf{A} \\
\mathbf{A} & \rightarrow & \mathbf{aA} \mid \mathbf{bA} \mid \mathbf{B} \\
\mathbf{B} & \rightarrow & \mathbf{aC} \mid \mathbf{bD} \\
\mathbf{C} & \rightarrow & \mathbf{aE} \\
\mathbf{D} & \rightarrow & \mathbf{bE} \\
\mathbf{E} & \rightarrow & \mathbf{\Lambda}
\end{array}$$

(ii)



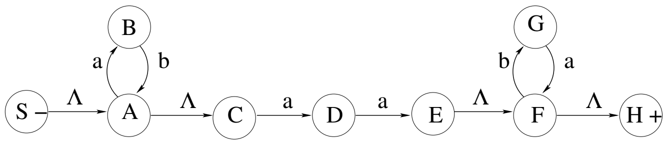
$$\begin{array}{lcl}
\mathbf{S} & \rightarrow & \mathbf{A} \\
\mathbf{A} & \rightarrow & \mathbf{aB} \mid \mathbf{bB} \mid \mathbf{C} \\
\mathbf{B} & \rightarrow & \mathbf{aA} \mid \mathbf{bA} \\
\mathbf{C} & \rightarrow & \mathbf{\Lambda}
\end{array}$$

(iii)

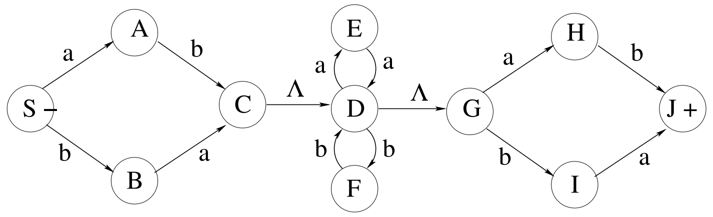


$$\begin{array}{lcl}
\mathbf{S} & \rightarrow & \mathbf{A} \\
\mathbf{A} & \rightarrow & \mathbf{aB} \mid \mathbf{bC} \mid \mathbf{D} \\
\mathbf{B} & \rightarrow & \mathbf{aA} \\
\mathbf{C} & \rightarrow & \mathbf{bA} \\
\mathbf{D} & \rightarrow & \mathbf{\Lambda}
\end{array}$$

(iv)



$$\begin{array}{lcl}
\mathbf{S} & \rightarrow & \mathbf{A} \\
\mathbf{A} & \rightarrow & \mathbf{aB} \mid \mathbf{C} \\
\mathbf{B} & \rightarrow & \mathbf{bA} \\
\mathbf{C} & \rightarrow & \mathbf{aD} \\
\mathbf{D} & \rightarrow & \mathbf{aE} \\
\mathbf{E} & \rightarrow & \mathbf{F} \\
\mathbf{F} & \rightarrow & \mathbf{bG} \mid \mathbf{H} \\
\mathbf{G} & \rightarrow & \mathbf{aF} \\
\mathbf{H} & \rightarrow & \mathbf{\Lambda}
\end{array}$$

(v) 

$$\begin{array}{lcl}
\mathbf{S} & \rightarrow & \mathbf{aA} \mid \mathbf{bB} \\
\mathbf{A} & \rightarrow & \mathbf{bC} \\
\mathbf{B} & \rightarrow & \mathbf{aC} \\
\mathbf{C} & \rightarrow & \mathbf{D} \\
\mathbf{D} & \rightarrow & \mathbf{aE} \mid \mathbf{bF} \mid \mathbf{G} \\
\mathbf{E} & \rightarrow & \mathbf{aD} \\
\mathbf{F} & \rightarrow & \mathbf{bD} \\
\mathbf{G} & \rightarrow & \mathbf{aH} \mid \mathbf{bI} \\
\mathbf{H} & \rightarrow & \mathbf{bJ} \\
\mathbf{I} & \rightarrow & \mathbf{aJ} \\
\mathbf{J} & \rightarrow & \mathbf{\Lambda}
\end{array}$$

8. The NFAs given above are already PDAs; they just don't use the stack.

(A transition in an NFA, labelled by a letter x, becomes a transition $x, \varepsilon \rightarrow \varepsilon$ when the NFA is viewed as a PDA.)

9.

Observe that i and n are both positive integers here.

(a) (solution by FIT2014 tutors, 2013)

$$\begin{array}{lcl}
S & \rightarrow & aBcc \\
S & \rightarrow & aScc \\
B & \rightarrow & Bb \\
B & \rightarrow & b
\end{array}$$

(b)

Inductive basis:

The shortest string in the language (assuming both n and i are ≥ 1, but the proof can easily be adapted to allow them both to be 0 as well) is $\mathbf{a}^1\mathbf{b}^1\mathbf{c}^2$ which equals **abcc** and has length 4. This string can be generated by the CFG as follows:

$$S \Rightarrow \mathbf{a}B\mathbf{cc} \Rightarrow \mathbf{abcc}.$$

Inductive step:

Let ℓ denote the length of the string, and assume $\ell \geq 5$ (else we are back in the inductive basis, which we've already dealt with).

Assume that *any string of the required form of length $< \ell$ has a derivation using the CFG.* (This is the *Inductive Hypothesis.*) Let $\mathbf{a}^n\mathbf{b}^i\mathbf{c}^{2n}$ be any string in the language of length ℓ, so that $\ell = 3n+i$. Since $\ell \geq 5$ (else we'd be back in the inductive basis), we must have either $n \geq 2$ or $i \geq 2$. We deal with these two cases in turn.

If $n \geq 2$, then the string has the form $\mathbf{aa}^{n-1}\mathbf{b}^i\mathbf{c}^{2(n-1)}\mathbf{cc}$. The inner string here, $\mathbf{a}^{n-1}\mathbf{b}^i\mathbf{c}^{2(n-1)}$, has length $\ell - 3$ which is $< \ell$, so we can use the Inductive Hypothesis, which implies that there is a derivation

$$S \Rightarrow \cdots \Rightarrow \mathbf{a}^{n-1}\mathbf{b}^i\mathbf{c}^{2(n-1)}.$$

Placing **a** at the start of every string in this derivation, and **cc** at the end of each such string, gives

$$\mathbf{a}S\mathbf{cc} \Rightarrow \cdots \Rightarrow \mathbf{aa}^{n-1}\mathbf{b}^i\mathbf{c}^{2(n-1)}\mathbf{cc}.$$

This is still a valid sequence of derivation steps, by the context-free property. (In effect, all we've done is change the *context* in the same way throughout, but the derivation steps don't depend on context, so all the derivation steps are still valid.) Now, the first string $\mathbf{a}S\mathbf{cc}$ can itself be derived in a single step from S, by using the second rule of the CFG. Putting this step at the start gives a new derivation, which now starts with S and therefore gives a complete derivation of the string at the end:

$$S \Rightarrow \mathbf{a}S\mathbf{cc} \Rightarrow \cdots \Rightarrow \mathbf{aa}^{n-1}\mathbf{b}^{i}\mathbf{c}^{2(n-1)}\mathbf{cc}.$$

Since the string at the end is just $\mathbf{a}^n\mathbf{b}^i\mathbf{c}^{2n}$, we now have a derivation for it.

It remains to deal with the case when $i \geq 2$ and $n = 1$. In this case, the string has the form $\mathbf{abb}^{i-1}\mathbf{cc}$. Now, the string with one fewer $\mathbf{b}$ in the middle — namely, $\mathbf{ab}^{i-1}\mathbf{cc}$ — is also of the same general form, but has length $\ell - 1$, which is $< \ell$. So by the Inductive Hypothesis it has a derivation

$$S \Rightarrow \cdots \Rightarrow \mathbf{ab}^{i-1}\mathbf{cc}.$$

Now, observe that *the only rule of our CFG which does not have a non-terminal symbol on its right side is the rule* $B \to \mathbf{b}$. It follows that this must be the last rule used in the above derivation. Furthermore, the $\mathbf{b}$ created by that rule is always the leftmost $\mathbf{b}$ in the string. If we were to replace *just the last step* in the above derivation by applying the rule $B \to B\mathbf{b}$ instead, leaving all earlier steps unchanged, then we would get a derivation of the string $\mathbf{a}B\mathbf{b}^{i-1}\mathbf{cc}$:

$$S \Rightarrow \cdots \Rightarrow \mathbf{a}B\mathbf{b}^{i-1}\mathbf{cc}.$$

We now add to this a single application of the rule $B \to \mathbf{b}$, which gives us a derivation of the string we are after, with the extra $\mathbf{b}$ in the middle:

$$S \Rightarrow \cdots \Rightarrow \mathbf{a}B\mathbf{b}^{i-1}\mathbf{cc} \Rightarrow \mathbf{abb}^{i-1}\mathbf{cc}.$$

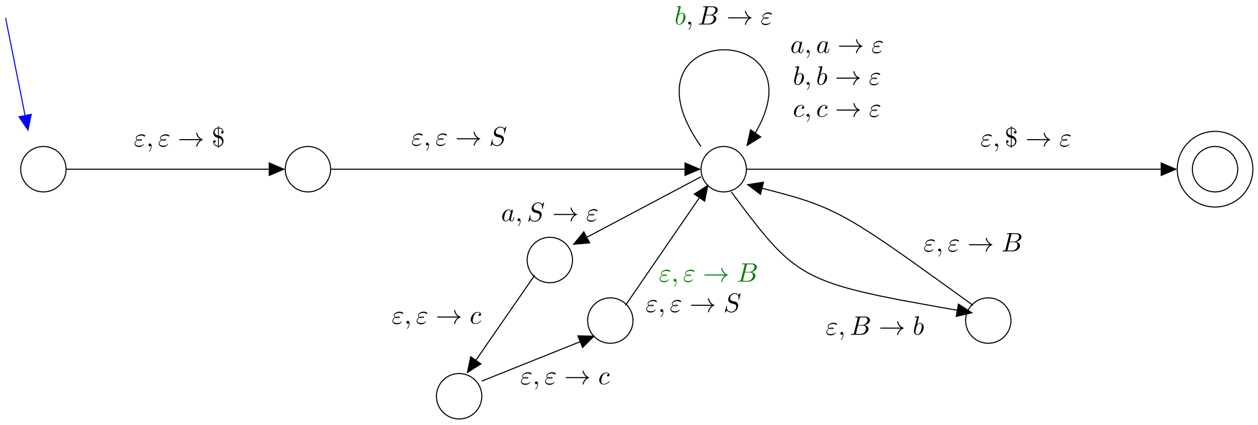
This is now a complete derivation of our string $\mathbf{ab}^i\mathbf{cc}$.

So, putting the two cases (the first was $n \geq 2$, the second was $n = 1$ and $i \geq 2$) together, we have seen that, whatever form our string of length ℓ takes, we can use the Inductive Hypothesis (applied to a shorter string) to construct a derivation for the string of length ℓ.

Conclusion:

By Mathematical Induction, *any* string of the required form, of *any* length, can be generated by this CFG.

(c)

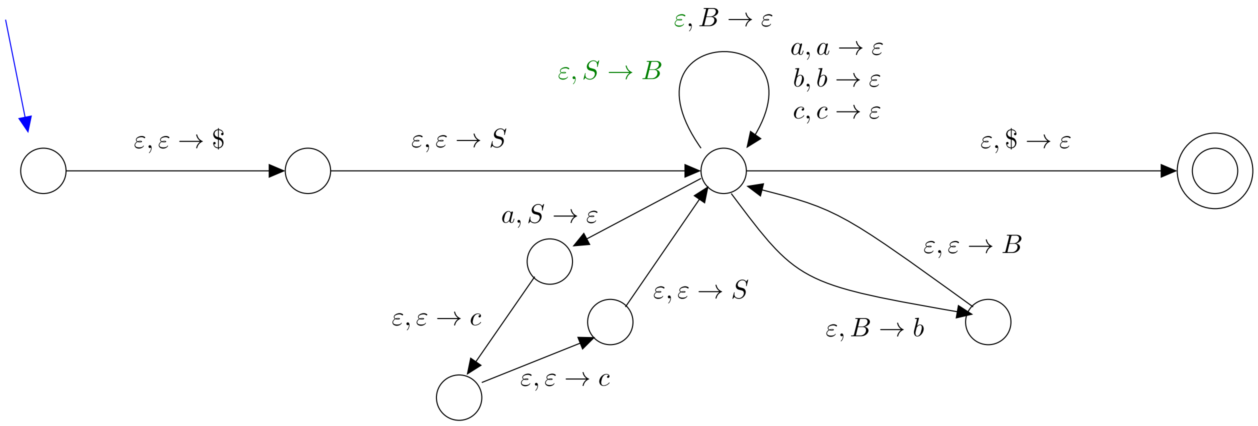


$b, B \to \varepsilon$
$a, a \to \varepsilon$
$b, b \to \varepsilon$
$c, c \to \varepsilon$
$\varepsilon, \varepsilon \to \$$
$\varepsilon, \varepsilon \to S$
$\varepsilon, \$ \to \varepsilon$
$a, S \to \varepsilon$
$\varepsilon, \varepsilon \to B$
$\varepsilon, \varepsilon \to B$
$\varepsilon, \varepsilon \to S$
$\varepsilon, \varepsilon \to c$
$\varepsilon, B \to b$
$\varepsilon, \varepsilon \to c$

Exercise: prove that the language generated by this grammar is not regular.

Suppose we allowed both i and n to be 0, too. Then the grammar and PDA become (with differences between the two solutions shown in green in each):

$$
\begin{aligned}
S &\to B \\
S &\to aScc \\
B &\to Bb \\
B &\to \varepsilon
\end{aligned}
$$



$\varepsilon, B \to \varepsilon$
$\varepsilon, S \to B$
$a, a \to \varepsilon$
$b, b \to \varepsilon$
$c, c \to \varepsilon$
$\varepsilon, \varepsilon \to \$$
$\varepsilon, \varepsilon \to S$
$\varepsilon, \$ \to \varepsilon$
$a, S \to \varepsilon$
$\varepsilon, \varepsilon \to B$
$\varepsilon, \varepsilon \to S$
$\varepsilon, \varepsilon \to c$
$\varepsilon, B \to b$
$\varepsilon, \varepsilon \to c$

Exercise: prove that the language generated by this grammar is not regular.

10.

(a) Given a k-limited PDA P, define a NFA N from it as follows.

States of N:

For every possible combination of state and stack contents of P, we are going to create a state of N. We represent the stack, with symbols $s_1, \ldots, s_d$ (from top down) where $d \leq k$, by the k-tuple $(s_1, \ldots, s_k)$, with $s_i = \varepsilon$ for $j > d$. Let Q be the set of states of P, let Σ be the input alphabet of P, and let Γ be the stack alphabet of P. For every $q \in Q$ and every k-tuple $(s_1, \ldots, s_k)$ where each $s_i \in \Gamma \cup \{\varepsilon\}$, we create a state $r(q, s_1, \ldots, s_k)$ for N. (We can suppose that, if $s_i = \varepsilon$, then $s_j = \varepsilon$ for all $j > i$.)

The number of states we create by this process is $\leq |Q| \times (|\Gamma| + 1)^k$, which is finite since Q and Γ are both finite.

Transitions for N:

Take any transition in M, from some state q to another state q'. Suppose its label is $x, s_1 \to s_1'$. We create corresponding transitions in N for $x \in \Sigma \cup \{\varepsilon\}$ from state $r(q, s_1, \ldots, s_k)$ to state $r(q', s_1', s_2, \ldots, s_k)$. We do this for every pair of states in N whose representations have this form. The state changes simulate reading (popping) s_1 from the stack and pushing s_1' onto it; the rest of the stack is unchanged.

Suppose instead that our transition has label $x, \varepsilon \to s_1'$. We create corresponding transitions in N for x from state $r(q, s_1, \ldots, s_k)$ to state $r(q', s_1', s_1, \ldots, s_{k-1})$. (Here, the state change simulates no reading from the stack, and s_1' is pushed onto it.)

Now suppose the transition has label $x, s_1 \to \varepsilon$. We create corresponding transitions in N for x from state $r(q, s_1, \ldots, s_k)$ to state $r(q', s_2, \ldots, s_k, \varepsilon)$.

Finally, suppose our transition has label $x, \varepsilon \to \varepsilon$. We create corresponding transitions in N for x from state $r(q, s_1, \ldots, s_k)$ to state $r(q', s_1, \ldots, s_k)$. (There is no change to the stack.)

The start state of N is the state $r(S, \varepsilon, \ldots, \varepsilon)$, where S is the start state of P and $\varepsilon, \ldots, \varepsilon$ represents the initially-empty stack.

The Final States of N are all states $r(F, s_1, \ldots, s_k)$ where F is a final state of P.

It can be shown that the NFA N we have constructed simulates the operation of the PDA P, and that an input string is accepted by P if and only if it is accepted by N.

(b) If a language is regular, then it is recognised by a NFA. Now, a NFA is just a PDA which never uses its stack, or in other words, a 0-limited PDA. This, in turn, is a special case of a k-limited PDA. So the language is recognised by a k-limited PDA.

Now suppose that a language is recognised by a k-limited PDA. We know from part (a) that this can be simulated by a NFA, which recognises the same language. So the language is recognised by a NFA. So it is regular.

Challenge: What if the limit on stack size is $\log n$, where n is the input string length, rather than a constant? Are $(\log n)$-limited PDAs equivalent to NFAs?

Supplementary exercises

11. This assertion is false. Consider the following CFG:

$$\begin{aligned} S &\to AA \\ A &\to a \\ A &\to b \end{aligned}$$

Here, the right-hand side of every production is a palindrome. But the grammar can generate strings which are not palindromes, for example **ab**.

Challenge: Determine which CFGs generate only palindromes.

12. Let G be a Context-Free Grammar for L. Let $\overleftarrow{G}$ be the CFG obtained from G by reversing every string in every production in G.

We claim that $\overleftarrow{G}$ is a CFG for $\overleftarrow{L}$.

To prove this, we need to show that every $x \in \overleftarrow{L}$ has a derivation using $\overleftarrow{G}$.

Take any $x \in \overleftarrow{L}$. By defintion of $\overleftarrow{L}$, we know $\overleftarrow{x} \in L$. So there must be a derivation of $\overleftarrow{x}$ in G. Reversing all strings in each step of this derivation gives a derivation of x using $\overleftarrow{G}$.

Hence $\overleftarrow{L}$ is a Context-Free Language.

13. Let L be a CFL with CFG G, and let $x \in L$ be a string in L with a derivation of length n. Now, $\overleftarrow{L}$ is also a CFL, by Question 12, and clearly $\overleftarrow{x} \in \overleftarrow{L}$. So $\overleftarrow{x}$ has a leftmost derivation of length n, by Question 5. By reversing all the strings in the derivation, we obtain a rightmost derivation of x, of length n, using $\overleftarrow{G}$.

14.

We use the Pumping Lemma for regular languages.

Let L be this language. Suppose it is regular. Then by Kleene's Theorem it has a Finite Automaton. Let N be the number of states in this FA.

Let w be any word in this language whose length is $> N$. By the Pumping Lemma, there exist strings x, y, z such that $w = xyz$, and $|xy| \leq N$, and y is nonempty, and $xy^iz \in L$ for all $i \in \mathbb{N}$.

If y does not contain a comma, then xy^iz is still a list of n numbers, and the string y falls inside one of these numbers; let's call it m. Repetition of y causes m to be replaced by a larger number (since y is nonempty). Doing this sufficiently many times (i.e., making i large enough) will make m greater than n. Then, since the list contains only n numbers and is a permutation, we have a contradiction with $xy^iz \in L$.

Suppose y contains exactly one comma. Let y^L and y^R be the portions before and after this comma, respectively, so that $y = y^L{,}y^R$. Then $yyy = y^L{,}y^Ry^L{,}y^Ry^L{,}y^R$, which contains the string y^Ry^L twice, in each case being the entire text between two commas, so it represents two identical numbers in the list, which violates the definition of a permutation. So $xy^3z \notin L$, again a contradiction.

If y contains at least two commas, then it contains the entire binary representation of one of the numbers in the permutation. Call it m. Then, repetition of y gives a list of numbers in which m is repeated, so it cannot be a permutation.

So, in every case, we obtain a contradiction. So L cannot be regular.

(In fact, we could use virtually the same argument to show that some much simpler languages, based on lists of numbers of this form, are not regular. Can you think of any?)

15. This is very similar to the previous question. Once again, we start by assuming that the language is of the type in question, in order to set up a contradiction later. We pick a sufficiently large member of the language, and use the appropriate Pumping Lemma to deduce the existence of a fixed-size portion of w that can be repeated within it arbitrarily often, always giving other words in the language. We observe that the lengths of these are increasing but with constant-size steps from one to the next. So some such word has a length that falls in the gap between two successive prime numbers. So that word cannot belong to the language, a contradiction. So the language cannot be of the assumed type.