Monash University
Faculty of Information Technology
2^{nd} Semester 2023

FIT2014
Exercise Sheet 5
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.

(b) $\exists N\ \forall w$ such that $\mathsf{ShaunAccepts}(w)$ and $|w| > N$: $\exists x, y, z$ such that $w = xyz$ and $y \neq \varepsilon$ and $|xy| \leq N$: $\forall i \geq 0\quad \mathsf{ShaunAccepts}(xy^iz)$.

(c) $\forall N\ \exists w$ such that $\mathsf{ShaunAccepts}(w)$ and $|w| > N$: $\forall x, y, z$ such that $w = xyz$ and $y \neq \varepsilon$ and $|xy| \leq N$: $\exists i \geq 0\quad \neg\,\mathsf{ShaunAccepts}(xy^iz)$.

(d) Reg has a winning strategy, which begins by choosing $N \geq 6$, the number of states of this FA.[1] Then Nona must choose, from among all strings accepted by this FA, a string w whose length satisfies $|w| > N$, so its length is greater than the number of states. No matter what string she chooses, its path through the FA must visit some state twice. Let y be a substring of w between two visits to the same state. Then y traces out a directed circuit in the FA, starting and ending at that same state. Let x be the initial substring of w before y, and let z be the remainder of w after y. For his second move, Reg chooses x, y, z. Now, no matter which i Nona chooses as her second move, the string xy^iz will still be accepted by the FA, since all it does is cause the FA to go around the directed circuit i times instead of just once, with the same end result (acceptance).

All we are doing here is recapping the core of the proof of the Pumping Lemma for Regular Languages, for the particular case of our given FA.

2. SHAWN is almost the same as the language recognised by the Finite Automaton SHAUN, except that the one-letter string a belongs to SHAWN but not to SHAUN. So the games SHAUN and SHAWN are really the same when $N \geq 2$. So Reg has a winning strategy for SHAWN, and it's the same strategy he used for SHAUN.

3.

(b) Since HALF-AND-HALF can be shown by the Pumping Lemma to be non-regular, Nona has a winning strategy. Whatever value of N is chosen by Reg in his first move, her reply is to choose $w = \mathtt{a}^N\mathtt{b}^N$. Then, no matter how Reg moves — i.e., no matter how he chooses x, y, z according to the rules — the string y he chooses must fall entirely within the first half of w, using $|xy| \leq N$. So Nona chooses any $i \neq 1$ as her last move. She wins, because for $i \neq 1$, the string xy^iz has different numbers of as and bs, and therefore is not in HALF-AND-HALF.[2]

4.

For (a) and (b), we can use the answers to Q1(b) and Q1(c), respectively, as templates, with acceptance/rejection by SHAUN replaced throughout by membership/non-membership of L throughout.

[1] In fact, it's sufficient for N to be at least the total number of states minus the number of sink states. (Why?) In this case, that gives $N \geq 6 - 1 = 5$.

[2] Thanks to FIT2014 student Yisong Yu for advising a correction to an earlier version.

(a) $\exists N \ \forall w \in L$ such that $|w| > N$: $\exists x, y, z$ such that $w = xyz$ and $y \neq \varepsilon$ and $|xy| \leq N$: $\forall i \geq 0 \quad xy^iz \in L$.

(b) $\forall N \ \exists w \in L$ such that $|w| > N$: $\forall x, y, z$ such that $w = xyz$ and $y \neq \varepsilon$ and $|xy| \leq N$: $\exists i \geq 0 \quad xy^iz \notin L$.

(c) If L is regular, then the Pumping Lemma tells us that Reg has a winning strategy. His first move is to choose N to be $\geq$ the number of states in a FA that recognises L

If L is non-regular, then it is harder to determine, in general, who has a winning strategy. If L can be shown to be non-regular using a proof by contradiction based on the Pumping Lemma, then Nona has a winning strategy. But some non-regular languages cannot be shown to be non-regular using the Pumping Lemma. Indeed, there exist non-regular languages for which Reg has a winning strategy in the Pumping Game. Can you find one?

5.

(a) Assume, by way of contradiction, that VERY-EVEN is regular.

(b) N is the number of states of a hypothetical Finite Automaton for VERY-EVEN. Such an FA must exist, under our assumption that VERY-EVEN is regular, by Kleene's Theorem. But we can't make any assumptions about how many states it has. So we let N stand for that hypothetical number of states. We have to allow N to be any positive integer, without ever giving it a specific value.

(c) There are many possibilities for w. One possibility is $w = \mathtt{1}^{2N}\mathtt{0}^N$:

$$w = \underbrace{\mathtt{111}\cdots\cdots\mathtt{11}}_{2N \text{ bits}}\underbrace{\mathtt{00}\cdots\mathtt{0}}_{N \text{ bits}}$$

This belongs to VERY-EVEN because (i) it finishes with N zeros, so it is divisible by 2^N, and (ii) it has $2N + N = 3N$ bits, which is certainly $\leq 3N$. It also has length $\geq N$, in fact its length is $3N$.

Another possibility for w — feasible, but less helpful — is to give it the same overall pattern but to craft it very carefully so that it has exactly N bits: $w = \mathtt{1}^{\lfloor 2N/3 \rfloor}\mathtt{0}^{\lceil N/3 \rceil}$. This is a multiple of $2^{\lceil N/3 \rceil}$, because it finishes with $\lceil N/3 \rceil$ zeros, and its length is $\lfloor 2N/3 \rfloor + \lceil N/3 \rceil \leq 3 \lceil N/3 \rceil$, so it belongs to VERY-EVEN. Although the rest of the proof can be made to work for this choice of w, more cases need to be considered. This is very like our first proof that HALF-AND-HALF is non-regular in Lecture 11, slides 15–16, which required more cases than the second proof in slides 17–18. So, in fact, we don't *gain* anything by carefully ensuring that w only satisfies our length requirement minimally. In fact, our "stinginess" only makes our life more complicated! Our earlier choice of w (in the previous paragraph) is longer, but leads to a simpler proof.

There are many other possibilities for w. For example,

$$\begin{aligned} w &= \mathtt{1}^N\mathtt{0}^{3137 \cdot N}, \\ w &= \mathtt{1}(\mathtt{01})^N\mathtt{0}^{N+1}, \end{aligned}$$

can all do the job, with appropriate tinkering with the details of the proof.

(d) Using $w = \mathtt{1}^{2N}\mathtt{0}^N$ (our first suggestion in (c) above), y must lie within the first N bits, which means it lies within that initial stretch of 1s. We will be able to treat all these possible placements with a single argument in the next part of the proof. So we only have one case to consider, even though that case covers many possible placements.

Using $w = \mathtt{1}^{\lfloor 2N/3 \rfloor}\mathtt{0}^{\lceil N/3 \rceil}$, the possible placements of y fall naturally into three cases, according to how the position of y relates to the two different parts (initial 1s, final 0s) of w: y could lie entirely within the initial stretch of 1s, or it could lie entirely within the final stretch of 0s, or it could include 10, i.e., the point where the change from 1s to 0s occurs and at least one bit on either side.

(e) We use $w = \mathtt{1}^{2N}\mathtt{0}^N$ and the fact that y falls within the first N 1s. We can use $i = 2$ (in fact, any larger i will suffice too, but $i = 0$ won't work). Then xy^2z has more 1s in that initial stretch (since y is nonempty). So the number of 1s is now $> 2N$, which implies that the length of xy^2z is $> 3N$, which is more than three times as long as the stretch of zeros on the right (and note that the length of that stretch of zeros is still N: the pumping has not enlarged it). So, although the string — as a binary number — is still a multiple of 2^N, thereby satisfying part (i) of the definition of VERY-EVEN, it no longer satisfies the length requirement in part (ii) of the definition. So xy^2z is not in VERY-EVEN.

(f) This contradicts the conclusion of the Pumping Lemma, which all regular langauges must satisfy. Therefore VERY-EVEN is not regular.

(g) Motivated by evil intent, choose $w = \mathtt{10}^N$.

This string represents 2^N, so it satisfies part (i) of the definition of VERY-EVEN. Its length is $N + 1 \leq 3N$, so it satisfies part (ii) of the definition. So it belongs to VERY-EVEN. It also satisfies the Pumping Lemma requirement that its length is $\geq N$.

Now consider where a nonempty substring y might be positioned, within the first N letters of w. For the proof to work, we would need to be able to show that, no matter where y is, we can pump it by some amount to get a string outside the language.

In this case, *some* choices of y will work but *others* will not.

If y consisted of just the 1 at the start, then we can produce a string xy^iz outside VERY-EVEN by choosing $i = 0$. In that case, $xy^iz = xy^0z = xz$ no longer has that solitary 1 at the start, so it starts with 0, which violates the definition of VERY-EVEN because the language only allows binary representations of *positive* integers. In fact, the same thing works for *any* y that includes the initial 1.

BUT consider what happens if y does not include that initial 1. In that case, y is a nonempty substring of 0s. Whatever value of y we choose, the string xy^iz still consists of a 1 followed by some number of 0s, so it is just a binary representation of some power of 2. It therefore belongs to VERY-EVEN. So we cannot find any $i \geq 0$ such that xy^iz is not in VERY-EVEN. So we are unable to derive our desired contradiction in this case.

So this evil choice of w does not work, in helping us use the Pumping Lemma to prove that VERY-EVEN is not regular. It does not matter that *some* choices of y can be used to pump outside the language; we need to be able to do that for *every* choice of y, and that's not possible here.

But the failure of *this* choice of w does not make the language regular! As we saw in (c)–(f), there is *another* choice of w for which the argument works, so the language is not regular.

(h)[3]

One possible CFG for the language, due to Anson Lean and Thomas Hendrey (independently), is:

$$\begin{aligned} S &\rightarrow \mathtt{1}BX\mathtt{0} && (1)\\ X &\rightarrow BBX\mathtt{0} && (2)\\ X &\rightarrow \varepsilon && (3)\\ B &\rightarrow \mathtt{1} && (4)\\ B &\rightarrow \mathtt{0} && (5)\\ B &\rightarrow \varepsilon && (6) \end{aligned}$$

[3] Thanks to FIT2014 student Anson Lean and FIT2014 staff Vee Voon Yee, Thomas Hendrey, Roger Lim and Rebecca Young for pointing out an error in a previous version of the solution of (h) and suggesting corrections.

We use this one in the next part, (i).

More CFGs, from staff:

From Vee Voon Yee:

$$\begin{array}{rcl} S & \to & \mathtt{1}X\mathtt{0} \\ X & \to & DDX\mathtt{0} \mid DX\mathtt{0} \mid X\mathtt{0} \mid D \\ D & \to & \mathtt{1} \mid \mathtt{0} \mid \varepsilon \end{array}$$

From Roger Lim:

$$\begin{array}{rcl} S & \to & \mathtt{1}X \mid \mathtt{10}X \mid \mathtt{11}X \\ X & \to & \mathtt{0} \mid X\mathtt{0} \mid \mathtt{0}X\mathtt{0} \mid \mathtt{1}X\mathtt{0} \mid \mathtt{00}X\mathtt{0} \mid \mathtt{01}X\mathtt{0} \mid \mathtt{10}X\mathtt{0} \mid \mathtt{11}X\mathtt{0} \end{array}$$

From Rebecca Young:

$$\begin{array}{rcl} S & \to & \mathtt{10} \mid \mathtt{100} \mid \mathtt{110} \mid \mathtt{1}A\mathtt{00} \\ A & \to & N \mid NN \mid NNN \mid NNA\,\mathtt{0} \\ N & \to & \mathtt{1} \mid \mathtt{0} \end{array}$$

(i) This derivation uses the first grammar above, with rules (1)–(6).

$$\begin{array}{rcl} S & \overset{(1)}{\Longrightarrow} & \mathtt{1}BX\mathtt{0} \\ & \overset{(2)}{\Longrightarrow} & \mathtt{1}BBBX\mathtt{00} \\ & \overset{(2)}{\Longrightarrow} & \mathtt{1}BBBBBX\mathtt{000} \\ & \overset{(4)}{\Longrightarrow} & \mathtt{10}BBBBX\mathtt{000} \\ & \overset{(4)}{\Longrightarrow} & \mathtt{100}BBBX\mathtt{000} \\ & \overset{(5)}{\Longrightarrow} & \mathtt{1001}BBX\mathtt{000} \\ & \overset{(4)}{\Longrightarrow} & \mathtt{10010}BX\mathtt{000} \\ & \overset{(4)}{\Longrightarrow} & \mathtt{100100}X\mathtt{000} \\ & \overset{(3)}{\Longrightarrow} & \mathtt{100100}\varepsilon\mathtt{000} \\ & = & \mathtt{100100000} \end{array}$$

6.

The extended regular expression (a*)b\1 matches the language $\{\mathbf{a}^n\mathbf{b}\mathbf{a}^n \mid n \geq 0\}$, but this is not regular, by the Pumping Lemma.

7.

(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

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

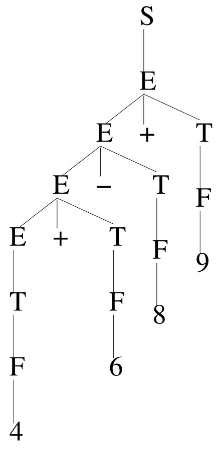
for example, we could have $w = 0^N 1 1^N$. The important thing is that y falls within the stretch of N zeros at the start, so that repeating y pushes the first 1 so far along that the middle bit is now a 0.

(b) We prove that CENTRAL-ONE is context-free by giving a context-free grammar for it.[4]

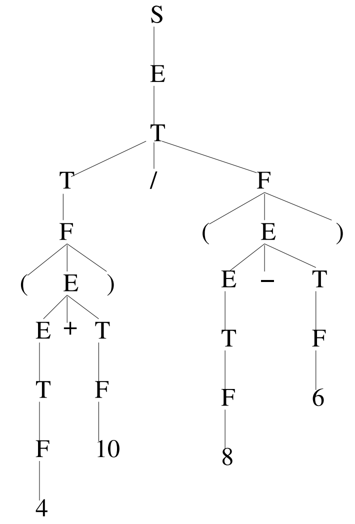
$$\begin{array}{lcl}
\mathbf{S} & \rightarrow & \mathbf{ASA} \\
\mathbf{A} & \rightarrow & \mathbf{0} \\
\mathbf{A} & \rightarrow & \mathbf{1} \\
\mathbf{S} & \rightarrow & \mathbf{1}
\end{array}$$

8.

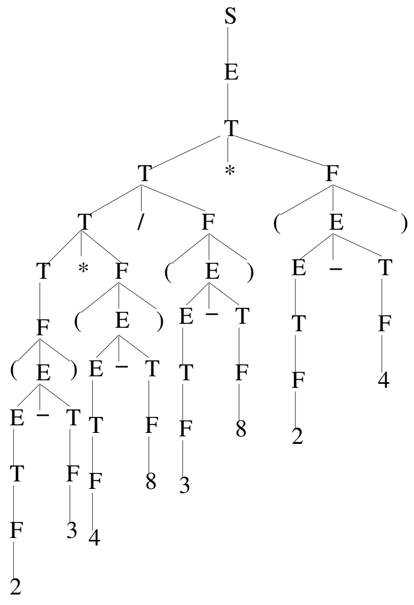
1(i)



1(ii)



1(iii)



9. (a)

$$\begin{array}{rcll}
\text{S} & \longrightarrow & \text{Subject can Verbs} & (7) \\
\text{S} & \longrightarrow & \text{SubjectWithToBe Adjective} & (8) \\
\text{Subject} & \longrightarrow & \text{Tom} \mid \text{I} \mid \text{Tom and I} & (9)
\end{array}$$

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

$$\text{Verbs} \longrightarrow \text{OneVerb} \mid \text{Verbs and OneVerb} \tag{10}$$

$$\text{OneVerb} \longrightarrow \text{hop} \mid \text{run} \mid \text{stop} \mid \text{dig} \tag{11}$$

$$\text{SubjectWithToBe} \longrightarrow \text{I am} \mid \text{Tom is} \tag{12}$$

$$\text{Adjective} \longrightarrow \text{big} \mid \text{six} \tag{13}$$

Variations on this are possible.

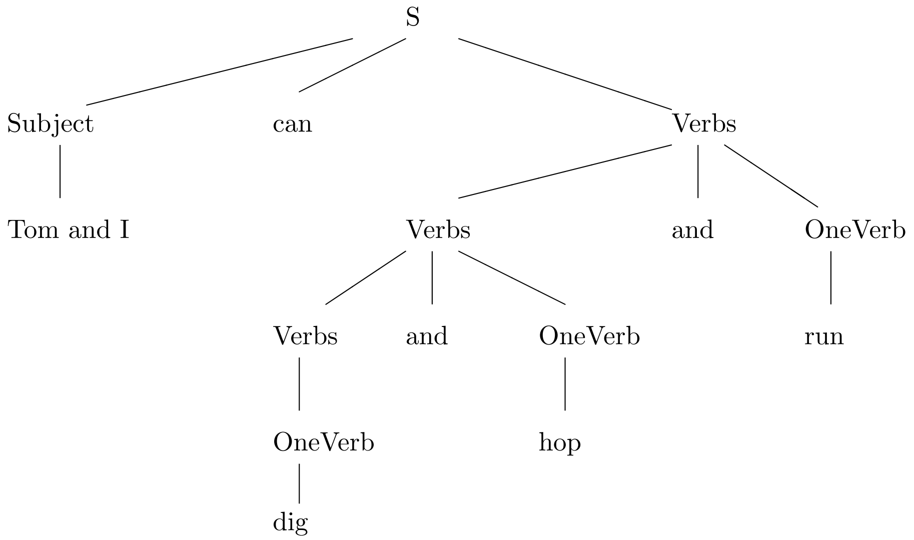
(b)

Derivation:

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

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.

10.

Theorem. If a string in a context-free language has a derivation of length n, then it has a leftmost derivation of length n.

Proof. (Thanks to FIT2014 tutor Harald Bögeholz for this proof.)

Convert the derivation to a parse tree by starting with a tree consisting of just the start symbol. For each rule applied in the derivation, add the symbols on the right hand side of the rule as children under the nonterminal symbol on the left hand side of the rule, in the correct left-to-right order. Note that at any step during this process the current string of symbols corresponds to all leaves of the current tree, in order from left to right.

In the resulting parse tree, the leaves form the derived string of terminal symbols and there are exactly n nonterminal symbols at the internal nodes of the tree, each corresponding to one step of the derivation.

A leftmost derivation can now be read off this tree. Start by marking the start symbol as active. The current string always consists of all active symbols in left-to-right order. Visit the leftmost active nonterminal symbol, mark it as inactive and mark as active its children in the tree. This corresponds to applying a rule of the context-free grammar. This process ends after visiting all nonterminal symbols. Since there are n nonterminal symbols in the tree, the length of this derivation is n.

Remark. The process of building the leftmost derivation can be viewed as traversing the parse tree in depth-first order left to right, applying a rule for each nonterminal symbol visited.

A rightmost derivation can be constructed by traversing the tree depth-first, right to left.

All derivations constructed from this tree in this manner, at each step choosing from any of the active nonterminal symbols, have length n, and among them is the original derivation.

Supplementary exercises

11.

(b) $\exists N\,\forall w \in$ EVEN-EVEN such that $|w| > N$: $\exists x, y, z$ such that $w = xyz$ and $y \neq \varepsilon$ and $|xy| \leq N$: $\forall i \geq 0 \quad xy^iz \in$ EVEN-EVEN.

(c) $\forall N\,\exists w \in$ EVEN-EVEN such that $|w| > N$: $\forall x, y, z$ such that $w = xyz$ and $y \neq \varepsilon$ and $|xy| \leq N$: $\exists i \geq 0 \quad xy^iz \notin$ EVEN-EVEN.

(d) The Pumping Lemma tells us that Reg has a winning strategy, since EVEN-EVEN is regular and Reg can start by choosing $N \geq$ the number of states of some FA that recognises EVEN-EVEN. (Recall the FA for EVEN-EVEN in lectures: it has four states. So Reg can just choose $N \geq 4$.)

Whichever string w is chosen by Nona in her first move (after Reg chooses N), Reg finds a nonempty substring that goes around a circuit in the FA for EVEN-EVEN. (Such a substring must exist, as the length of w is required to be greater than the number of states in a FA that recognises EVEN-EVEN.). That substring is y. Then x consists of the portion of w before y, and z consists of the portion of w after y. Then, no matter what i is chosen by Nona in her last move, the string xy^iz belongs to EVEN-EVEN, so Reg wins.

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

$$\begin{array}{ccc} S & \rightarrow & AA \\ A & \rightarrow & a \\ A & \rightarrow & b \end{array}$$

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.

13. 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.

14. 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 13, and clearly $\overleftarrow{x} \in \overleftarrow{L}$. So $\overleftarrow{x}$ has a leftmost derivation of length n, by Question 10. By reversing all the strings in the derivation, we obtain a rightmost derivation of x, of length n, using $\overleftarrow{G}$.

15.

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?)

16.

(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 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).

17. 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.

18.

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 $|xy| \leq N$, and y is nonempty, and for all $i \in \mathbb{N} \cup \{0\}$, the string xy^iz is also in the language.

Since $|xy| \leq N$, 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 $|xy| \leq N$ — 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 $i \geq 2$), 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.

19. Here is a regular expression for the language of context-free grammars over the given alphabet:

$(S \rightarrow (\varepsilon \cup (S \cup X_1 \cup \cdots \cup X_m \cup x_1 \cup \cdots \cup x_n) *)((S \cup X_1 \cup \cdots \cup X_m) \rightarrow (\varepsilon \cup (S \cup X_1 \cup \cdots \cup X_m \cup x_1 \cup \cdots \cup x_n) *) *$

20.

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

$$\sigma_1 \overset{L}{\Rightarrow} \tau_2 \overset{L}{\Rightarrow} \cdots \overset{L}{\Rightarrow} \tau_{n-1} \overset{L}{\Rightarrow} \sigma_n = w, \tag{14}$$

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}. [5]

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 (14) 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 *leftmost* 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.)

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

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.

It's actually much easier to prove this result *without* induction, by using the parse tree, as we saw in Q10.