Monash University
Faculty of Information Technology
2^{nd} Semester 2022

FIT2014 Theory of Computation
Tutorial 5
Context-Free Grammars and the Pumping Lemmas

SOLUTIONS

1.

Observe that i and n are both positive integers here.

(a) (solution by FIT2014 tutors, 2013)

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

(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 [illegible] e now have a derivation for it.

It remains to deal [illegible] ≥ 2 and $n = 1$. In this case, the string has the form $\mathbf{abb}^{i-1}\mathbf{cc}$. Now, the s[illegible] n the middle — namely, $\mathbf{ab}^{i-1}\mathbf{cc}$ — is also of the same general form, but has [illegible] ℓ. So by the Inductive Hypothesis it has a derivation

$$[illegible] \cdots \Rightarrow \mathbf{ab}^{i-1}\mathbf{cc}.$$

Now, observe that *the* [illegible] *which does not have a non-terminal symbol on its right side is the rule* $B \rightarrow$ [illegible] 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 $B \rightarrow B\mathbf{b}$ 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 $B \rightarrow \mathbf{b}$, which gives us a derivation of the string we are after, with the extra $\mathbf{b}$ in the middle:

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

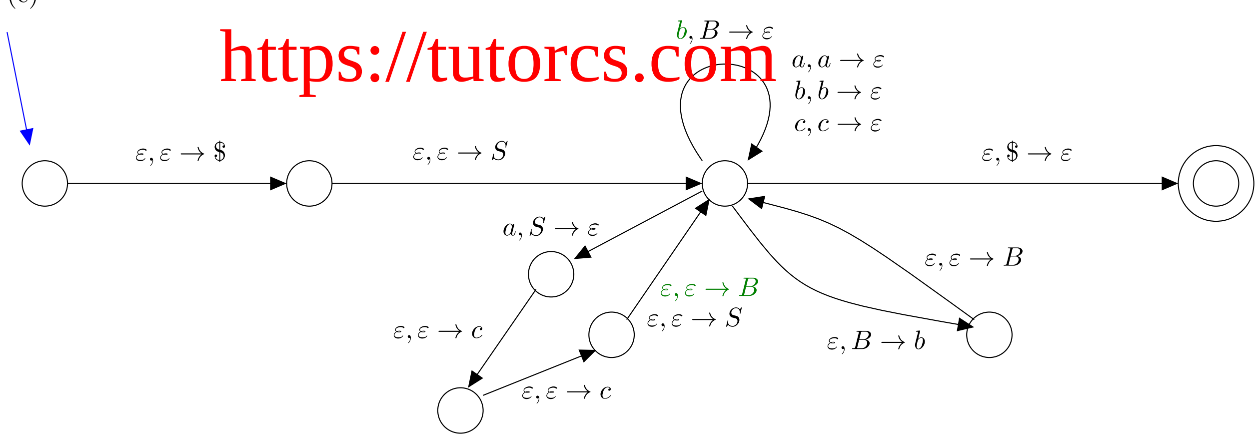
This is now a complete derivation of our string $\mathbf{ab}^i\mathbf{cc}$.

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.

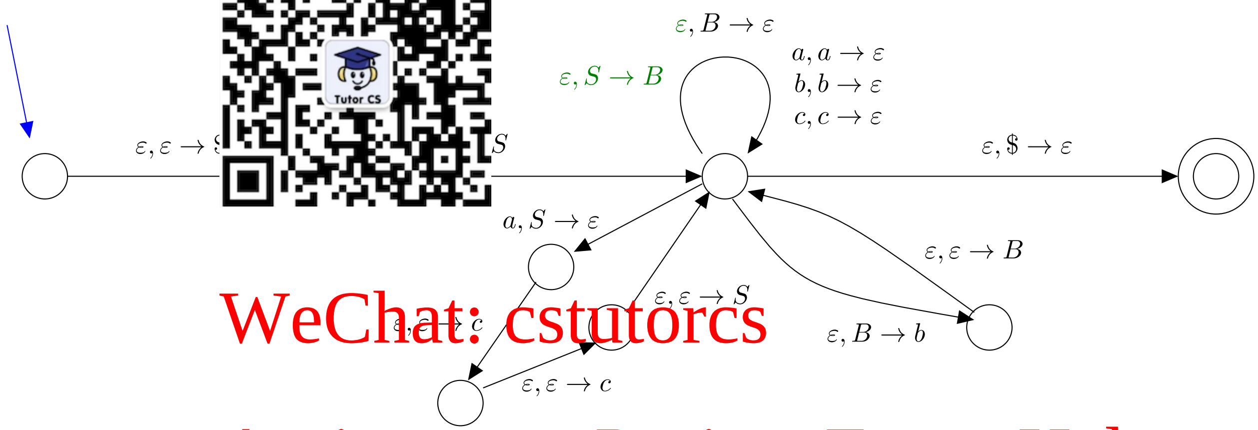
(c)



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):

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



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

2.

(a)

(i). Z.
Note, in particular, that X is ***not*** the logical negation of W.

(ii). Y, Z

(b)

(i).

W: $\forall L\ (\textbf{CFL}(L) \Rightarrow \textbf{Infinite}(L))$

X: $\forall L\ (\textbf{CFL}(L) \Rightarrow \neg\textbf{Infinite}(L))$

Y: $\exists L\ (\textbf{CFL}(L) \wedge \textbf{Infinite}(L))$
Note: the following answer is incorrect: $\exists L\ (\textbf{CFL}(L) \Rightarrow \textbf{Infinite}(L))$. This is because the expression $\textbf{CFL}(L) \Rightarrow \textbf{Infinite}(L)$ is True whenever L is *not* context-free, as well as when L is context-free and infinite.

Z: $\exists L\ (\textbf{CFL}(L) \wedge \neg\textbf{Infinite}(L))$

(ii).

$$
\begin{aligned}
\neg W &= \neg\forall L\ (\textbf{CFL}(L) \Rightarrow \textbf{Infinite}(L)) \\
&= \exists L\ \neg(\textbf{CFL}(L) \Rightarrow \textbf{Infinite}(L)) \\
&= \exists L\ \neg(\neg\textbf{CFL}(L) \vee \textbf{Infinite}(L))
\end{aligned}
$$

$$
\begin{aligned}
& \quad \text{(since, in general, } A \Rightarrow B \text{ is equivalent to } \neg A \vee B\text{)}\\
&= \exists L\ (\neg\neg\mathbf{CFL}(L) \wedge \neg\mathbf{Infinite}(L))\\
& \quad \text{(by De Morgan's Law)}\\
&= \exists L\ (\mathbf{CFL}(L) \wedge \neg\mathbf{Infinite}(L))
\end{aligned}
$$

(iii). Z.
Note, in particu … ally equivalent to $\neg W$.

3. Here is the gram … ence.

$$
\begin{aligned}
S &\rightarrow P && (1)\\
P &\rightarrow PP && (2)\\
P &\rightarrow \mathtt{ha}Q && (3)\\
Q &\rightarrow Q\mathtt{a} && (4)\\
Q &\rightarrow \varepsilon && (5)
\end{aligned}
$$

(a)

(i). Here is a derivation of the string ha.

$$
\begin{aligned}
S &\Rightarrow P && \text{(using rule (1))}\\
&\Rightarrow \mathtt{ha}Q && \text{(using rule (3))}\\
&\Rightarrow \mathtt{ha}\varepsilon && \text{(using rule (5))}\\
&= \mathtt{ha}
\end{aligned}
$$

We now prove that this is a shortest string in LOL.

Any derivation (of any string) from S must start with the production $S \Rightarrow P$. So at least one of rules (2), (3) must be used in any derivation. If only (2) is used, then no symbol except P can ever be generated. So rule (3) must be used. This introduces two terminal symbols (in fact, the string ha). So every string in LOL must have *at least* two letters.

(In fact, we have shown that every string in LOL must contain the two-letter string ha. With this observation, we can say that ha is *the* shortest string in LOL, not just that it is *a* shortest string in LOL.)

(ii). The grammar is not regular, since not every production rule has, on its right-hand side, a semiword or a string of terminals. In particular, rules (2) and (4) are not of the required form.

(iii). LOL is regular language. It has the regular expression $\mathtt{haa}^*(\mathtt{haa}^*)^*$.

(iv). The grammar is not in Chomsky Normal Form. Apart from (2), the rules are not in the required form.

(b)
Step 1. How do we know such an x exists? In other words, how do we know LOL isn't empty?
Step 8. Consider: "…we can get strings that are even longer than that, and so on, indefinitely."

This needs formal justification. Don't "wave hands" to cover gaps in the reasoning, or rely on the reader to fill in gaps. This indicates a need for a proper proof by induction.

(c)

We prove this by induction on n.

Inductive basis ($n = 2$): we already know LOL has a string of length at least 2, since we saw in part (a)(i) that LOL contains the string ha.

Inductive step:

Suppose $n \geq 2$. As our Inductive Hypothesis, assume that LOL contains a string of length $\geq n$. Call it x. Since $x \in$ L[illegible]n for x using the given grammar.

Now we just repea[illegible] 3–7 in part (b).

This gives us a str[illegible]er longer than x. Since x has length $\geq n$, we deduce that y has length $\geq n$[illegible]a string of length at least $n + 1$.

Therefore, by Mat[illegible] all $n \geq 2$, LOL contains a string of length $\geq n$.

(d)

This proof is corre[illegible]mplicated and long-winded.

The contradiction hinges on the fact that a *finite* language has a longest string, and therefore cannot have arbitrarily long strings. The deduction that LOL has arbitrarily long strings (Step 2) does indeed contradict the finiteness of LOL (assumed in Step 1), as stated in Step 3.

But you can derive the desired contradiction much more simply. All you need to do is to *use the (assumed) finiteness of LOL to drive the construction*, rather than "parking" the finiteness at the start of the proof and coming back to it at the very end. Here's how you do it.

Any finite language has a longest string; but we have already seen how to take any string in LOL and make another string in LOL that is one letter longer. You don't need to go to the extra trouble of showing that LOL has arbitrarily long strings.

(e)

Assume, by way of contradiction, that LOL is finite.

We know it is nonempty (part (a)(i)).

Since LOL is finite and nonempty, it must have a longest string. Call it x. *(Note how the assumed finiteness of LOL drives our construction of x.)*

Our earlier argument shows how to construct a string that is one letter longer than x. This contradicts the maximality of x.

Therefore our assumption, that LOL is finite, is incorrect.

Therefore LOL is infinite.

4.

(b) Suppose L is context-free. Let k be the number of non-terminal symbols in a CNF CFG for L.

ONE APPROACH:

Let n be any positive integer such that $n^2 > 2^{k-1}$. Then $w := \mathbf{a}^{n^2} \in L$, so by the Pumping Lemma for Context-Free Languages, there exist strings u, v, x, y, z such that $w = uvxyz$, and the length of vxy is $\leq 2^k$, [1] and v, y are not both empty, and $uv^ixy^iz \in L$ for all $i \geq 0$.

Let ℓ be the sum of the lengths of v and y. (Note, $\ell \geq 1$.)

From now on, the proof is very close to Tute 4, Q6(b).

Then the length of uv^ixy^iz is $n^2 + (i-1)\ell$. So the strings uv^ixy^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 Tute 4, Q6(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, $uv^ixy^iz \notin L$, a contradiction. So the assumption that L

[1] Thanks to FIT2014 tutor Roger Lim for spotting an error in this upper bound in an earlier version.

is context-free must be false. Hence L is not context-free.

ANOTHER APPROACH (due to FIT2014 tutor Harald Bögeholz):[2]

Let $n = 2^k$. Then $w := \mathbf{a}^{n^2} \in L$, and its length n^2 satisfies $n^2 > 2^{k-1}$ (since $n^2 > n = 2^k > 2^{k-1}$), so it's long enough for the Pumping Lemma for CFLs to apply. So there exist strings u, v, x, y, z such that $w = uvxyz$, [illegible] y are not both empty, and $uv^ixy^iz \in L$ for all $i \geq 0$.

Let ℓ be the sum [illegible]. (Note, $\ell \geq 1$.) Observe that $\ell \leq |vxy|$, so combining with $|vxy| \leq 2^k = n$ [illegible] Using $i = 2$ in the string uv^ixy^iz gives $uv^2xy^2z \in L$. Now, this string has l[illegible] $yz| + |v| + |y| = |w| + \ell = n^2 + \ell$, which falls between the following lower an[illegible]

$$[illegible] > n^2,$$
$$[illegible] < n^2 + 2n + 1 = (n+1)^2.$$

These are the lengths of two successive strings in L, namely $\mathbf{a}^{n^2}$ and $\mathbf{a}^{(n+1)^2}$. There are no strings in L whose length lies between the lengths of these strings. So the string uv^2xy^2z cannot belong to L, which is a contradiction. So the assumption that L is context-free must be false. Hence L is not context-free.

(c) This is mostly the same as the answer to Tute 4, Q6(c). The changed parts are underlined below.

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 context-free. Then $L_1 \cap L_2$ is also context-free, since the intersection of a context-free language and a regular language is context-free.

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 context-free. So we have a contradiction.

Hence L_2 is not context-free.

5.

(b) $\exists k \forall w \in L$ such that $|w| > 2^{k-1}$: $\exists u, v, x, y, z$ such that $w = uvxyz$ and $vy \neq \varepsilon$ and $|vxy| \leq 2^k$: $\forall i \geq 0 \quad uv^ixy^iz \in L$.

(c) $\forall k \exists w \in L$ such that $|w| > 2^{k-1}$: $\forall u, v, x, y, z$ such that $w = uvxyz$ and $vy \neq \varepsilon$ and $|vxy| \leq 2^k$: $\exists i \geq 0 \quad uv^ixy^iz \notin L$.

(d) If L is context-free, then the Pumping Lemma for CFLs tells us that Con has a winning strategy. He starts by choosing k to be $\geq$ the number of nonterminal symbols in a Chomsky Normal Form CFG for L.

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

[2]This is similar to the previous approach but pins down the details of getting a string uv^ixy^iz to fall in a gap between successive members of L. To do this, it helps to choose w to be longer than it was above.

6.

We prove that SPI is not context-free.

Assume, by way of contradiction, that SPI is context-free. Then it has a CFG in Chomsky Normal Form with k nonterminals. Then, by the Pumping Lemma for CFLs, every $w \in$ SPI with $|w| > 2^{k-1}$ can be written $w = uvxyz$ where $vy \neq \varepsilon$, $|vxy| \leq 2^k$, and for all $i \geq 0$ we have $uv^ixy^iz \in$ SPI.

Put $N > 2^{k-1}$ and [illegible] string:

$$[illegible]\mathtt{0}, \mathtt{0}, \mathtt{0}, \mathtt{1}\overbrace{\mathtt{00}\cdots\mathtt{0}}^{N\text{ zeros}}, \mathtt{1}\overbrace{\mathtt{00}\cdots\mathtt{0}}^{N\text{ zeros}})$$

$$[illegible]\mathtt{0}, \mathtt{10}^N, \mathtt{10}^N),$$

which represents the 5[illegible]$, 0, 2^N, 2^N)$. This satisfies the rules for SPI, so $w \in$ SPI. We also see that $|w| >$ [illegible]e size lower bound in the Pumping Lemma for CFLs.

Now consider all possible divisions of w into five parts, $w = uvxyz$. Consider, in particular, the possible locations for v and y within w. We have several cases.

Case 1: If either or both of v, y contains a comma or a parenthesis, then uv^2xy^2z contains either an excess of commas (i.e., more than four commas) or an excess of parentheses, so $uv^2xy^2z \notin$ SPI.

From now on, we can restrict to cases where neither v nor y contains a comma or a parenthesis.

Case 2: If vxy falls entirely within c or s, then $vxy = \mathtt{0}$, so in fact either $v = \mathtt{0}$ or $y = \mathtt{0}$ since we must have $vy \neq \varepsilon$. It follows that $x = \varepsilon$. So uv^2xy^2z has two consecutive zeros, $\mathtt{00}$, as its second or third number (c or s) in the 5-tuple. But we are not allowed to represent zero as $\mathtt{00}$. So $uv^2xy^2z \notin$ SPI.

From now on, we can restrict to cases where each of v and y falls entirely within one of the three numbers representing prices: i.e., entirely within p, or entirely within $p_{\max}$, or entirely within $p_{\min}$. It's possible that v and y don't fall within the same number, but they cannot intersect more than one of the three numbers.

Case 3: if vxy falls entirely within p, then uv^2xy^2z has all numbers the same as for w except that p is now larger (since $vy \neq \varepsilon$). But previously $p = p_{\max}$, so now $p > p_{\max}$, which breaks the rules of SPI. So $uv^2xy^2z \notin$ SPI.

Case 4 (very similar to Case 3): if vxy falls entirely within $p_{\min}$, then uv^2xy^2z has all numbers the same as for w except that $p_{\min}$ is now larger (since $vy \neq \varepsilon$). But previously $p_{\min} = p_{\max}$, so now $p_{\min} > p_{\max}$, which breaks the rules of SPI. So $uv^2xy^2z \notin$ SPI.

Case 5: if vxy falls entirely within $p_{\max}$, then $uv^0xy^0z = uxz$ has all numbers the same as for w except that either (a) $p_{\max}$ is now *smaller* (since $vy \neq \varepsilon$) or (b) it has lost its only nonzero bit (if v contains the $\mathtt{1}$ at the start of $p_{\max}$). For (a): previously $p_{\min} = p_{\max}$, so now $p_{\min} > p_{\max}$, which breaks the rules of SPI. So $uv^0xy^0z \notin$ SPI. For (b): $p_{\max}$ is now invalid, since it is either a string of two or more zeros (which is not an allowed number representation here) or it is a single zero (which is not a positive integer).

It remains to consider cases where v and y fall within two different numbers from p, $p_{\min}$, $p_{\max}$. Note that v comes before y in w, which restricts the possibilities.

Case 6: If v falls within p and y falls within $p_{\max}$, then forming uv^0xy^0z creates a string in which either p or $p_{\max}$ is invalid or reduced (or both are invalid or reduced), while $p_{\min}$ is unchanged. If uv^0xy^0z is valid then $p_{\min}$ is now greater than the new p or $p_{\max}$, which breaks the rules of SPI. So $uv^0xy^0z \notin$ SPI.

Case 7: If v falls within p and y falls within $p_{\min}$, then forming uv^2xy^2z creates a string in which either p or $p_{\min}$ is increased (or both are increased), while $p_{\max}$ is unchanged and hence is now smaller than p or $p_{\min}$, which breaks the rules of SPI. So $uv^2xy^2z \notin$ SPI.

Case 8: If v falls within $p_{\max}$ and y falls within $p_{\min}$, then:

- If $v \neq \varepsilon$, then forming uv^0xy^0z creates a string in which $p_{\max}$ is invalid or has decreased, while p is unchanged and hence is now greater than $p_{\max}$ (if the latter is still valid), which breaks the rules of SPI. So $uv^0xy^0z \notin$ SPI.

- If $y \neq \varepsilon$, then forming uv^2xy^2z creates a string in which $p_{\min}$ is increased, while p is unchanged and hence is now smaller than $p_{\min}$, which breaks the rules of SPI. So $uv^2xy^2z \notin$ SPI.

We have now covered all possibilities for the division of w into five parts, $w = uvxyz$ with $vy \neq \varepsilon$. In each case, we found a value of i such that $uv^ixy^iz \notin$ SPI. This contradicts the conclusion of the Pumping Lemma for [illegible] uv^ixy^iz always belongs to the language in question). So our initial assumption, th[illegible] was wrong. Therefore SPI is not context-free.

Notes:

- In Cases 3 & 4, [illegible] other values of i could we have used, at those points in the proof, in or[illegible] SPI?
- Which part of t[illegible] Pumping Lemma for CFLs did we *not* use in this proof? Had we used it, [illegible] eliminated any of the above cases?

7.

First iteration: single letters.

a can be generated by the nonterminals A, D.

b can be generated by the nonterminal C.

Second iteration: pairs of consecutive letters (a.k.a. *digraphs*). We only need to consider digraphs that actually appear in the target string. In this case, we consider all digraphs except aa.

ab: The a can be generated by A or D, and the b can be generated by C, so the pair ab can come from either of the nonterminal pairs AC and DC. The pair AC can be produced by A, but the pair DC cannot be produced by a single nonterminal. So the sole nonterminal that can produce ab is A.

bb: This pair can come only from the pair CC, but this pair cannot come from a single nonterminal. So there is no nonterminal that can produce bb.

ba: This pair can come from either CA or CD. The former cannot come from a single nonterminal; the latter can come from B. So the only possibility here is B.

We summarise what we have worked out for the digraphs in the following table.

digraph	nonterminals that can produce it
ab	A
bb	–
ba	B

Third iteration: triples of consecutive letters (a.k.a. *trigraphs*). We only need to consider trigraphs that actually appear in the target string. For our string abbba, these are abb, bbb, bba.

We view each triple as a concatenation of two nonempty shorter strings.

abb: This can be formed as a concatenation of a followed by bb, or as a concatenation of ab followed by b. The former is not possible, because bb cannot be produced by a nonterminal (as we saw above). But for the latter, ab can be produced by A and b can be produced by C. So abb can be produced by AC. This in turn can be produced from the single nonterminal A. No other single nonterminal can do the job.

bbb: This can be formed as a concatenation of b followed by bb, or as a concatenation of bb followed by b. But in each case, the bb cannot be produced by any nonterminal. So it is also not possible to produce bbb from a nonterminal.

bba: This can be formed as a concatenation of b followed by ba, or as a concatenation of bb followed by a. We can rule the latter out, because of bb. For the former, it can be produced from CB, but that in turn cannot be produced by just a single nonterminal.

We summarise what we have found.

trigraph	nonterminals that can produce it
abb	[illegible]
bbb	—
bba	—

Fourth iteration: 4[illegible]letters (a.k.a. *tetragraphs*). For our string `abbba`, the ones we need to consi[illegible]

`abbb`: The only [illegible]hich gives the nonterminal pair AC. This in turn can be produced from the[illegible]

`bbba`: There is [illegible]ossible split includes a string (either the one before, or the one after, the spli[illegible]ced by a single nonterminal.

[illegible]	[illegible]erminals that can produce it
[illegible]	A
`bbba`	—

Fifth and last iteration:

This is for our target string `abbba`.

The split `a`,`bbba` does not help, because `bbba` cannot be produced from a nonterminal. The split `ab`,`bba` is similarly unhelpful: see `bba` above. The split `abb`,`ba` can be produced by the nonterminal pair AB, which in turn can be produced only by the single nonterminal S. The split `abbb`,`a` can be produced by the nonterminal pairs AA and AD, but neither of these can come from a single nonterminal pair. So we are left with just S.

Since the starting nonterminal S is one of the nonterminals from which the target string can be produced, that string belongs to the language generated by the grammar.

8.

(a) Start with the given grammar:

$$
\begin{array}{lcll}
S & \rightarrow & P & (1) \\
P & \rightarrow & PP & (2) \\
P & \rightarrow & \texttt{ha}Q & (3) \\
Q & \rightarrow & Q\texttt{a} & (4) \\
Q & \rightarrow & \varepsilon & (5)
\end{array}
$$

First, eliminate the empty productions. There is just one here, $Q \rightarrow \varepsilon$. To ensure that we keep its effect, we create a new copy of every rule that has Q on its right-hand side, and delete that Q (i.e., replace it by the empty string; in effect, we apply the rule $Q \rightarrow \varepsilon$ to it). (It gets a bit more complicated for rules that have more than one Q on their right-hand side, but that doesn't happen here. What should we do in that case?) In this case, there are two such rules, (3) and (4). So we get two new rules as well: $P \rightarrow \texttt{ha}$ and $Q \rightarrow \texttt{a}$. We now have the following grammar, equivalent to the original one:

$$
\begin{array}{lcll}
S & \rightarrow & P & (1) \\
P & \rightarrow & PP & (2) \\
P & \rightarrow & \texttt{ha}Q & (3) \\
P & \rightarrow & \texttt{ha} & \\
Q & \rightarrow & Q\texttt{a} & (4) \\
Q & \rightarrow & \texttt{a} &
\end{array}
$$

We must now eliminate unit productions. We just have one, namely (1), which changes S to P. It can be replaced by rules that replace S by anything that can be produced, by application of a single rule, from P. In this case, we have three rules for P. We therefore get three new rules, and the grammar is now as follows.

$$
\begin{array}{lcll}
S & \rightarrow & PP & \\
S & \rightarrow & \mathtt{ha}Q & \\
S & \rightarrow & \mathtt{ha} & \\
P & \rightarrow & PP & (2) \\
P & \rightarrow & \mathtt{ha}Q & (3) \\
P & \rightarrow & \mathtt{ha} & \\
Q & \rightarrow & Q\mathtt{a} & (4) \\
Q & \rightarrow & \mathtt{a} &
\end{array}
$$

We now need rules to generate each terminal, by itself, *from a new nonterminal*. This gives new rules $A \rightarrow \mathtt{a}$ and $H \rightarrow \mathtt{h}$. (For $\mathtt{a}$, it's not enough to rely on the rule $Q \rightarrow \mathtt{a}$, which we already have, because nonterminal Q can be replaced by other things too; these new rules for the terminals need new symbols that will only ever be replaced by their corresponding terminals.) We obtain the following grammar.

$$
\begin{array}{lcll}
S & \rightarrow & PP & \\
S & \rightarrow & \mathtt{ha}Q & \\
S & \rightarrow & \mathtt{ha} & \\
P & \rightarrow & PP & (2) \\
P & \rightarrow & \mathtt{ha}Q & (3) \\
P & \rightarrow & \mathtt{ha} & \\
Q & \rightarrow & Q\mathtt{a} & (4) \\
Q & \rightarrow & \mathtt{a} & \\
A & \rightarrow & \mathtt{a} & \\
H & \rightarrow & \mathtt{h} &
\end{array}
$$

We then modify all rules with a terminal whose right-hand sides are not just a single terminal, replacing the terminals by the corresponding nonterminals we introduced above.

$$
\begin{array}{lcll}
S & \rightarrow & PP & \\
S & \rightarrow & HAQ & \\
S & \rightarrow & HA & \\
P & \rightarrow & PP & (2) \\
P & \rightarrow & HAQ & (3) \\
P & \rightarrow & HA & \\
Q & \rightarrow & QA & (4) \\
Q & \rightarrow & \mathtt{a} & \\
A & \rightarrow & \mathtt{a} & \\
H & \rightarrow & \mathtt{h} &
\end{array}
$$

The last step is to deal with the rules whose right-hand sides have three or more nonterminals. We introduce the new nonterminal J and the new rule $J \rightarrow HA$, and use it to modify the grammar.

$$
S \rightarrow PP
$$

$$
\begin{aligned}
S &\rightarrow JQ \\
S &\rightarrow HA \\
P &\rightarrow PP && (2) \\
P &\rightarrow JQ && (3) \\
& \quad\; HA \\
& \quad\; QA && (4) \\
& \quad\; \mathtt{a} \\
& \quad\; \mathtt{a} \\
& \quad\; \mathtt{h} \\
& \quad\; HA
\end{aligned}
$$

(b)

Applying the CYK algorithm to this grammar and the target string `hahaa`, gives the following results at the successive iterations.

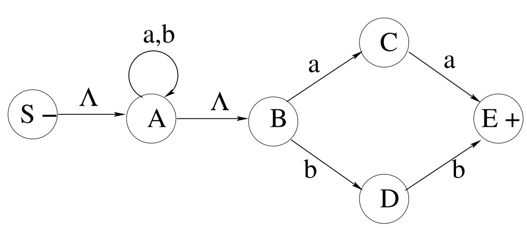
substring	nonterminals that can produce it	
`h`	H	
`a`	Q, A	
`ha`	S, P, J	... since these can all produce HA
`ah`	—	
`aa`	Q	... since $Q \Rightarrow QA \Rightarrow$ `aa`
`hah`	—	
`aha`	—	
`haa`	S, P	... using split `ha`,`a` and nonterminal pair JQ
`haha`	S, P	... using split `ha`,`ha` and nonterminal pair PP
`ahaa`	—	
`hahaa`	S, P	... using split `ha`,`haa`

Since S appears in the list of nonterminals that can produce `hahaa`, we conclude that `hahaa` belongs to the language generated by the grammar.

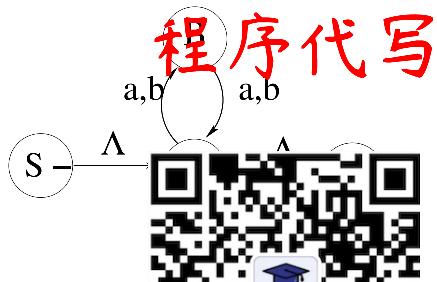
Supplementary exercises

9.

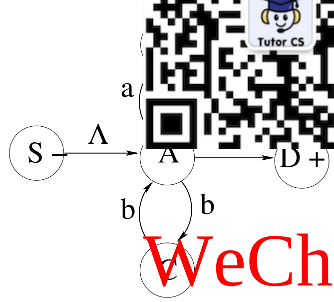
(i)



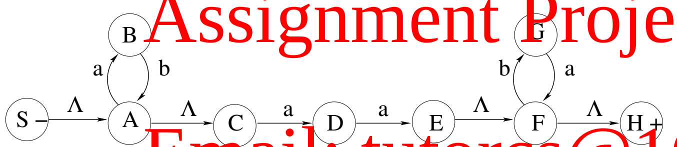
$$
\begin{aligned}
\mathbf{S} &\rightarrow \mathbf{A} \\
\mathbf{A} &\rightarrow \mathbf{aA} \mid \mathbf{bA} \mid \mathbf{B} \\
\mathbf{B} &\rightarrow \mathbf{aC} \mid \mathbf{bD} \\
\mathbf{C} &\rightarrow \mathbf{aE} \\
\mathbf{D} &\rightarrow \mathbf{bE} \\
\mathbf{E} &\rightarrow \mathbf{\Lambda}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

(ii) 

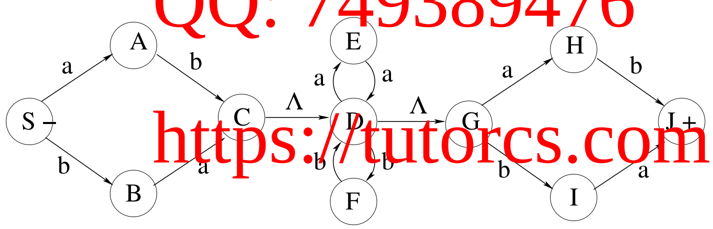
$$\begin{array}{lcl} \mathbf{S} & \to & \mathbf{A} \\ \mathbf{A} & \to & \mathbf{aB} \mid \mathbf{bB} \mid \mathbf{C} \\ \mathbf{B} & \to & \mathbf{aA} \mid \mathbf{bA} \\ \mathbf{C} & \to & \mathbf{\Lambda} \end{array}$$

(iii) 

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

(iv) 

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

(v) 

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

10. 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 $x, \varepsilon \to \varepsilon$ when the NFA is viewed as a PDA.)

11. The terminal symbols for our grammar will be just the letters of our alphabet. Create a new nonterminal symbol for each of the regular expressions formed during the construction of the regular expression using any of the three operations: concatenation, alternative, and Kleene star. (Recall the inductive definition of regular expressions.)

- Create a production rule of the form $S \to R$, where S is the start symbol and R is the new nonterminal symbol that we have introduced to stand for the entire regular expression.
- For each regular expression formed by concatenation, i.e., $R_1 = R_2R_3$, create the production rule $R_1 \to R_2R_3$.
- For each regular [illegible] he alternative operation, i.e., $R_1 = R_2 \cup R_3$, create two production rules [illegible] R_3.
- For each regular [illegible] he Kleene star, i.e., $R_1 = R_2^*$, create the three production rules $R_1 \to$ [illegible] R_1R_2.
- For each regula[illegible] st a string w of letters (where w may be empty or nonempty), i.e., [illegible] duction rule $R_1 \to w$.

12.

(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 $d \le k$, 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 $\le |Q| \times (|\Gamma| + 1)^k$, 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?

13.
(a)

We prove that Foo

Suppose FootySco s a Finite Automaton, by Kleene's Theorem. Let k be the number of states i or FootyScore. Let n be any positive integer such that $n \geq k$.

Let w be the strin y also be written

$$(\underbrace{11}_{\text{g}} \quad \underbrace{\cdots\cdots\cdots 1}_{\text{d-string, length } n}, \underbrace{11\cdots\cdots\cdots 1}_{\text{point-string, length } 7n})$$

This string represents a score of a team that has kicked n goals and n behinds, giving $6n + n = 7n$ points. It satisfies the definition of strings in FootyScore, so it belongs to the language. Therefore, 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 is non-empty, and $xy^iz \in$ FootyScore for all $i \geq 0$.

Firstly, observe that y cannot include either of the two commas, since if it did, repeating y (in forming xy^iz) would give more than two commas altogether, in violation of the definition of strings in FootyScore (which must have exactly two commas). Similarly, y cannot contain either of the two parentheses.

Therefore y must fall entirely within the goal-string, or entirely within the behind-string, or entirely within the point-string. In each of these cases, repeating y would upset the score: it would change the number of goals, *or* the number of behinds, *or* the number of points, without changing either of the other two numbers. When only one of the three numbers g, b, p is changed, it is no longer true that $6g + b = p$, so the corresponding string no longer belongs to FootyScore.

This contradicts the conclusion from the Pumping Lemma, that $xy^iz \in$ FootyScore for all i. Hence our assumption, that FootyScore is regular, was wrong.

Therefore FootyScore is not regular.

The above proof did not use the fact that $|xy| \leq n$ (guaranteed by the Pumping Lemma). We could do a slightly different proof by using it. It tells us that the initial part xy of the string has length at most n, and since the goal string is more than n letters long, the substring xy must come before the first comma. This tells us that y lies entirely within the goal string, except that it may also contain the initial opening parenthesis (if x is empty, which is possible). In the former case, repeating y increases the number of goals without a corresponding increase in the number of points; in the latter case, the initial opening parenthesis is repeated. Either way, the rules of the language are violated, and again we can deduce that $xy^iz \notin$ FootyScore, so obtaining a contradiction.

This argument illustrates the main purpose of $|xy| \leq n$: it gives us more control over where y lies within w.

(b)

FootyScore is context-free, because it is generated by the following CFG.

$$\begin{aligned}
S &\rightarrow (G) && (6)\\
G &\rightarrow \mathtt{1}G\mathtt{111111} && (7)\\
G &\rightarrow \mathtt{,}B && (8)\\
B &\rightarrow \mathtt{1}B\mathtt{1} && (9)\\
B &\rightarrow \mathtt{,} && (10)
\end{aligned}$$

14.

We first prove by induction on n that, for every substring y of x, where $|y| \le n$, the algorithm correctly finds all nonterminals in G that can generate y.

For the inductive basis, $n = 1$. The substrings y consist just of one letter each, and a grammar in Chomsky Normal Form will have, for each letter, some rules with that letter alone on their right-hand sides. The CY[illegible]e nonterminals on the left of such rules, and specifies them as the nontermi[illegible]y, which is correct.

Inductive step: Le[illegible]aim is true for substrings y of length $\le n$. So, for each such substring, at the [illegible]he algorithm finds all nonterminals that generate it.

Now consider wha[illegible]on that considers substrings of length $n+1$. Let y be such a string. The al[illegible]way of splitting y into two substrings, a left substring y_L and a right substri[illegible]lit, we already know (from previous iterations) the set of nonterminals that c[illegible]set of nonterminals that can produce y_R. Since y_L and y_R are shorter than y, they each have $\le n$ letters, so the inductive hypothesis applies, and tells us that these sets of nonterminals are complete and correct. The algorithm then constructs the set of all nonterminal pairs XY where X is in the first set of nonterminals (for y_L) and Y is in the second set of nonterminals (for y_R). For each such pair XY, it finds all rules of the form $W \to XY$, and gives these W as the nonterminals that can produce y.

This yields a set of nonterminals that can produce y. We need also to show that *every* nonterminal that can produce y must be in this set.

For y to be produced from a nonterminal symbol W, the first production must replace W by two other nonterminals, by the structure of the rules in Chomsky Normal Form (and using the fact that $|y| > 1$, so we cannot replace W by just a single terminal). Call these nonterminals X and Y. The rest of the production of y produces a left substring from X and a right substring from Y. (It is routine also to show that these two substrings cannot be empty. Just note that empty productions do not occur in CNF.) So, if y can be produced from W, then we can split it into two shorter substrings such that, for some nonterminals X and Y that produce the left and right substrings respectively, the grammar has the rule $W \to XY$. By the arguments in the previous paragraph, this shows that the algorithm does indeed find W among the nonterminals that can produce y.

All this works for any generated string y of length $n+1$. This completes the inductive step.

Therefore, by Mathematical Induction, the claim is true for all n.

A string is generated by the CFG if and only if it can be produced from the Start symbol. We have just proved that the CYK algorithm correctly finds the list of all nonterminals that can produce a given string. Therefore, a string is generated by the CFG if and only if the set of nonterminals that the CYK algorithm finds for that string, as possible producers of it, includes the Start symbol.

15. [3]

Assume that SIS is context-free. Then there exists a CFG in CNF that generates SIS.[4] Let k be its number of nonterminals. Then, by the Pumping Lemma for CFLs, every $w \in$ SIS with $|w| > 2^{k-1}$ can be written $w = uvxyz$ where $vy \neq \varepsilon$, $|vxy| \le 2^k$, and for all $i \ge 0$ we have $uv^ixy^iz \in$ SIS.

For convenience, put $N := 2^k$.

Let us choose w to be the SIS-representation of the sequence $(2^N, 2^N+1, 2^N+2)$. This looks like:

$$\underbrace{\texttt{100...000}}_{N+1 \text{ bits}}\texttt{\#}\underbrace{\texttt{100...001}}_{N+1 \text{ bits}}\texttt{\#}\underbrace{\texttt{100...010}}_{N+1 \text{ bits}}$$

[3] Thanks to FIT2014 tutors Nathan Companez and Harald Bögeholz for pointing out errors with an earlier version of this proof in 2021.

[4] ...or, at least, SIS $\setminus \{\varepsilon\}$. But, when using a Pumping Lemma to prove that a given language is not regular or context-free, we only find ourselves looking at *sufficiently large* strings; we never look at the empty string. So the fact that a CFG in CNF does not generate the empty string is not an issue for these proofs.

So it is certainly long enough.

Let $uvxyz$ be any segmentation of w into five substrings such that $vy \neq \varepsilon$ and $|vxy| \leq 2^k$.

Case 1:

If either v or y includes a # then, in uv^3xy^3z, either v^3 or y^3 has three *equally spaced* #'s. The two numbers represented by the two stretches of letters between these #'s are equal, due to the way they are obtained by r[illegible] is not allowed, in SIS, for two numbers in the sequence to be equal. So uv^3xy[illegible]S.

Case 2:

If v and y each f[illegible]epresentation of one of the *first* two numbers in the sequence (i.e., either [illegible] since at least one of v, y is nonempty — pumping up (i.e., using some $i \geq$ [illegible] enlarge that number, and pumping sufficiently many times will make it larg[illegible]er in the sequence, $2^N + 2$. This violates the increasing nature of the sequenc[illegible] is not in SIS.

Case 3:

If v and y each fall within the binary representation of one of the *last* two numbers in the sequence (i.e., either $2^N + 1$ or $2^N + 2$), then instead of pumping *up*, we pump *down*, by choosing $i = 0$ to form uxz. Removing any bit from either of the numbers $2^N + 1$ or $2^N + 2$ changes an $(N+1)$-bit binary number to an N-bit binary number, which must be $< 2^N$. So it becomes smaller than the first number in the sequence. This destroys the strictly-increasing property, so the resulting string uxz is not in SIS.

> Remark:
>
> Had we pumped *up* (using $i \geq 2$) instead of *down*, we would have made the number(s) even larger. This would break the rules of SIS if both v and y lie within the middle number (i.e., $2^N + 1$), since the pumping would make it larger, so that it becomes at least as large as the last number, $2^N + 2$. But it might not break the rules of SIS if v lies within the second number and y lies within the third number, since in that case it's possible (depending on the length, position and content of v and y) for the second and third numbers to be pumped up "in tandem" so that the strictly-increasing property of the sequence is preserved.

The only locations for v and y that may appear not to be covered by these three cases are if v and y do not contain a # *and* they do not fall entirely within one or the other of two consecutive numbers in this sequence. But this would require v to fall entirely within the first number and y to fall entirely within the third number. Then, the string between them, namely x, includes all the $N+1$ bits of the middle number, so $|vxy| \geq N+1$. This contradicts the requirement that $|vxy| \leq 2^k$, since $N = 2^k$. So this possibility cannot arise.

> Having said that, it turns out that we could also deal with this case by pumping. We can either use $i = 0$, which reduces the third number so that it is no longer greater than the second number, or $i \geq 2$, which increases the first number so that it is no longer less than the second number.

So, in every case, there exists $i \geq 0$ such that $uv^ixy^iz \notin$ SIS. This violates the conclusion of the Pumping Lemma for CFLs. So we have a contradiction. So our initial assumption, that SIS is context-free, was wrong. Therefore SIS is not context-free.

16.

We prove that CricketBowlingFigures is not regular.

Suppose CricketBowlingFigures is regular. Then it has a Finite Automaton, by Kleene's Theorem. Let k be the number of states in a Finite Automaton for CricketBowlingFigures. Let n be any positive integer such that $n \geq k$.

Let w be the string $(\texttt{1}^n, \texttt{1}^n, ,)$, representing the bowling figures $(n, n, 0, 0)$ (for a bowler who bowls n overs without having any runs scored from the bowling at all, but without taking any wickets either). This may also be written

$$(\underbrace{\texttt{11}\cdots\cdots\cdots\cdots\texttt{1}}, \underbrace{\texttt{11}\cdots\cdots\cdots\cdots\texttt{1}}_{n \text{ maiden overs}}, ,)$$

It satisfies the definitio[illegible]owlingFigures, so it belongs to the language. Therefore, by the Pumping Lem[illegible]ges, there exist strings x, y, z such that $w = xyz$, and $|xy| \le n$, and y is non[illegible]cketBowlingFigures for all $i \ge 0$.

Since $|xy| \le n$, the[illegible]st lie within the initial portion of the string consisting just of "$(\texttt{11}\cdots\texttt{1}$", bef[illegible]

If y includes the le[illegible]ry start, then repeating or deleting y destroys the pair of parentheses so that[illegible]ketBowlingFigures for any $i \ne 1$.

If y does not include the left parenthesis, then it only includes 1s from the first unary number representing the number of overs, n. Then consider the string xz. The number of overs represented by this string is $< n$ (specifically, it is $n - |y|$). But the number of maiden overs is still n, so we have a violation of the constraint that $O \ge M$. Therefore $xz \notin$ CricketBowlingFigures.

This contradicts the conclusion of the Pumping Lemma for Regular Languages.

Therefore CricketBowlingFigures is not regular.

(b)

We prove that CricketBowlingFigures is not context-free.

Suppose CricketBowlingFigures is context-free. Then it has a CFG G in Chomsky Normal Form. Let k be the number of non-terminal symbols in G. Let n be any positive integer such that $n > 2^k$.

Let w be the string $(\texttt{1}^n, \texttt{1}^n, ,)$, representing the bowling figures $(n, n, 0, 6n)$ (for a bowler who bowls n overs without having any runs scored from the bowling at all, and taking a wicket from every ball). This may also be written

$$(\underbrace{\texttt{11}\cdots\cdots\cdots\cdots\texttt{1}}_{n \text{ overs}}, \underbrace{\texttt{11}\cdots\cdots\cdots\cdots\texttt{1}}_{n \text{ maiden overs}}, , \underbrace{\texttt{11}\cdots\cdots\cdots\cdots\texttt{1}}_{6n \text{ wickets}})$$

It satisfies the definition of strings in CricketBowlingFigures, so it belongs to the language. Therefore, by the Pumping Lemma for Context-Free Languages, there exist strings u, v, x, y, z such that $w = uvxyz$, and $|vxy| \le 2^k$, and v and y are not both empty, and $uv^ixy^iz \in$ CricketBowlingFigures for all $i \ge 0$.

Observe that $|vxy| < n$, since $|vxy| \le 2^k$ and $2^k < n$. Consider the implications of this for the location of vxy within the string w. They could both be within one of the strings of 1s, or they could lie across O and M or across M and W) including the comma(s) between them. But they could not include parts of both O and W.

If either of v or y includes a comma and/or a parenthesis, then choosing $i = 2$ gives $uv^2xy^2z \notin$ CricketBowlingFigures, since uv^2xy^2z has more than three commas and/or an excess of parentheses, violating the definition of the language.

If vxy lies within the string for O, then choosing $i = 0$ gives the string $uxz \notin$ CricketBowlingFigures, since uxz has $O < M$.

If v lies within the string for O and y lies within the string for M, then choosing $i = 0$ gives $uxz \notin$ CricketBowlingFigures, since uxz has $O = M < n$ and $W = 6n > 6\,O$.

If v lies within the string for M (regardless of where y is), then choosing $i = 2$ gives $uv^2xy^2z \notin$ CricketBowlingFigures, since uv^2xy^2z has $O < M$.

If v and y both lie within the string for W, then choosing $i = 2$ gives $uv^2xy^2z \notin$ CricketBowlingFigures, since uv^2xy^2z has $W > 6\,O$.

So, every possible location of vxy leads to a contradiction with the conclusion of the Pumping Lemma for Context-Free Languages.

So the initial assumption that CricketBowlingFigures is context-free was wrong.

Therefore CricketBowlingFigures is not context-free.