HW7 Solutions

Tommy Schmitz

October 19, 2016

1 Exercise 6.2(d)

Let $L = \{a^n b^{2n} a^n \mid n \geq 0\}$. Suppose to the contrary that L is context-free. Consider the string $u = a^n b^{2n} a^n$, where n is the constant given by the pumping lemma. The pumping lemma tells us that $u = vwxyz$ for some strings v, w, x, y, z satisfying

$$\begin{aligned}\text{(i) } & |wxy| \leq n,\\ \text{(ii) } & wy \neq \Lambda, \text{ and}\\ \text{(iii) } & vw^i xy^i z \in L \text{ for all } i \geq 0.\end{aligned}$$

First, note that neither w nor y can contain both as and bs, because otherwise vw^2xy^2z would not have the form $a^*b^*a^*$, contradicting (iii).

Next, we can conclude that wy must contain an equal number of as and bs, because otherwise vw^0xy^0z would have non-equal numbers of as and bs, contradicting (iii) again.

We also know (from (ii)) that wy cannot be empty, so it must contain at least one a and at least one b. Furthermore, (i) tells us that wxy cannot overlap both of the two blocks of as, so we conclude that vw^0xy^0z is either equal to $a^{n-k}b^{2n-k}a^n$ or equal to $a^n b^{2n-k} a^{n-k}$ (for some $k \geq 1$). But neither of these strings begins and ends with the same number of as, so vw^0xy^0z cannot be a member of L. This contradicts (iii), as desired; we conclude that L must not be context-free.

2 Exercise 6.5

2.1 Part a

(Disclaimer: this solution is more verbose than necessary. It's OK if you write less than this, but it must be clear from your answer that you understand all these details, and that you were just too lazy too write it all out. Laziness is OK; lack of understanding is not OK. For an example of a "lazier" proof, see 6.5(b).)

The language $L = \{a^n b^m a^m b^n \mid m, n \geq 0\}$ is context-free. Here is a grammar:

$$\begin{aligned} S &\to aSb \mid A \\ A &\to bAa \mid \Lambda. \end{aligned}$$

It remains to show both (Claim 1) every string generated by S is in L, and (Claim 2) every string in L is generated by S.

First, let $L_2 = \{b^n a^n \mid n \geq 0\}$. It will be helpful to first prove both (Claim 3) every string generated by A is in L_2, and (Claim 4) every string in L_2 is generated by A.

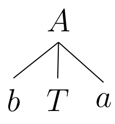
Proof of claim 3. Let x be some string generated by A. This means there must be some parse tree for x. By induction on this parse tree, I will prove that $x \in L_2$. There are two cases:

- Case 1: The root production in the parse tree is $A \to \Lambda$, so $x = \Lambda$. In this case, $x \in L_2$, as desired.
- Case 2: The root production in the parse tree is $A \to bAa$, so $x = bya$ for some string y generated by A. The induction hypothesis says that $y \in L_2$. Therefore, $y = b^n a^n$ for some n, and thus $x = b^{n+1}a^{n+1}$. Again, we see that $x \in L_2$, as desired.

Proof of claim 4. Let x be some string in L_2. This means $x = b^n a^n$ for some n. By induction on n, I will prove that x is generated by A. There are two cases:

- Case 1: $n = 0$. In this case, $x = \Lambda$, so we can build a parse tree for x using solely the production $A \to \Lambda$.

- Case 2: $n = m + 1$ for some m. In this case, $x = b^{m+1}a^{m+1} = bb^m a^m a$. The induction hypothesis says that $b^m a^m$ is generated by A, which means there must be a parse tree T for it. Therefore, we can build a parse tree for x by using $A \to bAa$ at the root, and by using T as its child:

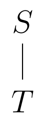


Proof of claim 1. Let x be some string generated by S. This means that there must be some parse tree for x. By induction on this parse tree, I will prove that $x \in L$. There are two cases:

- Case 1: The root production in the parse tree is $S \to A$, so we know from Claim 3 that $x = b^n a^n$ for some n. In this case, $x \in L$, as desired.
- Case 2: The root production in the parse tree is $S \to aSb$, so $x = ayb$ for some y generated by S. The induction hypothesis says that $y \in L$. Therefore, $y = a^n b^m a^m b^n$ for some m and n, and thus $x = a^{n+1}b^m a^m b^{n+1}$. Again, we see that $x \in L$, as desired.

Proof of claim 2. Let x be some string in L. This means $x = a^n b^m a^m b^n$ for some m and n. By induction on n, I will prove that x is generated by S. There are two cases:

- Case 1: $n = 0$. In this case, $x = b^m a^m$, so we know from Claim 4 that x must be generated by A. This means there is some parse tree T with A at its root and with x at its leaves. Using the production $S \to A$, we get the full parse tree for x:



- Case 2: $n = k + 1$ for some k. In this case, $x = a^{k+1}b^m a^m b^{k+1} = aa^k b^m a^m b^k b$. The induction hypothesis says that $a^k b^m a^m b^k$ is generated by S, which means there must be a parse tree T for it.

Therefore, we can build a parse tree for x by using $S \to aSb$ at the root, and by using T as its child:

$$\begin{tikzcd}
 & S \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
a & T & b
\end{tikzcd}$$

2.2 Part b

The language $L = \{xayb \mid x, y \in \{a,b\}^* \text{ and } |x| = |y|\}$ is context-free. Here is a grammar:

$$\begin{aligned}
S &\to Ab \\
A &\to XAX \mid a \\
X &\to a \mid b.
\end{aligned}$$

It remains to prove that S generates L. Since $S \to Ab$ is the only production for S, it suffices to show that A generates the language

$$L_2 = \{xay \mid x, y \in \{a,b\}^* \text{ and } |x| = |y|\}\,.$$

To do it, I will show both (1) every string generated by A is in L_2, and (2) every string in L_2 is generated by A.

Before we begin, please note that L_2 is simply the language of odd-length strings whose middle symbol is a.

1. The production $A \to a$ clearly generates a string in L_2, and since X only generates strings of length 1, it is also clear that the production $A \to XAX$ preserves membership in L_2.

2. Let $xay \in L_2$, so $|x| = |y| = n$ for some n. Then we can derive $X^n a X^n$ from A by using (n times) the production $A \to XAX$, followed by a single use of $A \to a$. By using the correct productions for X, we can easily get $X^n a X^n \Rightarrow^* xay$.

2.3 Part c

The language $L = \{xcx \mid x \in \{a,b\}^*\}$ is not context-free.

2.3.1 Solution 1

One of the closure properties of context-free languages says the following: if f is a homomorphism of strings and L is a context-free language, then $f(L)$ is also a context-free language.

Now we can prove by contradiction that L is not context-free, so assume that it is context-free. Then we can define the homomorphism:

$$f(a) = a \qquad\qquad f(b) = b \qquad\qquad f(c) = \Lambda.$$

Then $f(L) = \{ww \mid w \in \{a,b\}^*\}$. By the closure property, $f(L)$ must be context-free. But we already know that this language is not context-free, so this is a contradiction.

2.3.2 Solution 2

The proof is by contradiction, so assume that L is context-free. Then the pumping lemma applies, so consider the string $u = a^n b^n c a^n b^n$ (where n is the pumping lemma constant). The pumping lemma tells us that $u = vwxyz$ for some strings v, w, x, y, z satisfying

$$\begin{aligned}\text{(i) } & |wxy| \le n,\\ \text{(ii) } & wy \ne \Lambda, \text{ and}\\ \text{(iii) } & vw^ixy^iz \in L \text{ for all } i \ge 0.\end{aligned}$$

Notice that wy cannot contain c because otherwise vw^0xy^0z would have no c in it, contradicting (iii). Furthermore, x must contain the c and it must be that $|w| = |y|$, because otherwise vw^0xy^0z would not have c as its middle symbol, contradicting (iii) again.

We conclude (using (i) and (ii)) that $w = b^k$ and $y = a^k$ for some $k \ge 1$. But then $vw^0xy^0z = a^n b^{n-k} c a^{n-k} b^n$, which cannot be a member of L because the left side has more as than the right side.

2.4 Part d

The language $L = \{xyx \mid x, y \in \{a,b\}^* \text{ and } |x| \ge 1\}$ is not context-free.

Assume (for the sake of a contradiction) that L is context-free. Then the pumping lemma applies, so consider the string $u = a^n b^n a^n b^n$ (where n is the pumping lemma constant). The pumping lemma tells us that $u = vwxyz$ for some strings v, w, x, y, z satisfying conditions (i), (ii) and (iii).

Since wy cannot be empty, it must overlap at least one of the four "blocks" of letters in u. We will consider each of the four cases in turn.

Case 1 (wy overlaps the first block, namely a^n). In this case, let $i = 2$, so $vw^ixy^iz = a^{n+j}b^{n+k}a^nb^n$ for some $j \geq 1$ and $k \geq 0$ (the last two blocks can't be affected, thanks to (i)). But this string cannot be in L because none of its proper prefixes are equal to any of its proper suffixes; to see this, just notice that all of its short prefixes end in a, and none of its long suffixes begin with a^{n+j}.

Case 2 (wy overlaps the second of the four blocks, namely b^n). In this case, let $i = 0$, so $vw^ixy^iz = a^{n-j}b^{n-k}a^{n-m}b^n$ for some $k \geq 1$ and $0 \leq j \leq n-1$ and $0 \leq m \leq n-1$ (the last block can't be affected, thanks to (i)). But this string cannot be in L because none of its proper prefixes are equal to any of its proper suffixes; to see this, just notice that all of its short suffixes begin with b, and none of its long prefixes end with b^n.

Cases 3 and 4 are symmetric with cases 2 and 1, respectively.

3 Exercise 6.9(b)

3.1 The language is context-free

Here is a grammar:

$S \to J \mid K$	$\{a^ib^jc^k \mid i \neq j \text{ or } i \neq k\}$
$J \to Jc \mid A \mid B$	$\{a^ib^jc^k \mid i \neq j\}$
$A \to aAb \mid aA \mid a$	$\{a^ib^j \mid i > j\}$
$B \to aBb \mid Bb \mid b$	$\{a^ib^j \mid i < j\}$
$K \to C \mid D$	$\{a^ib^jc^k \mid i \neq k\}$
$C \to aCc \mid aC \mid aE$	$\{a^ib^jc^k \mid i > k\}$
$D \to aDc \mid Dc \mid Ec$	$\{a^ib^jc^k \mid i < k\}$
$E \to bE \mid \Lambda$	$\{b^j \mid j \geq 0\}$

It is straightforward to prove that each non-terminal generates the language mentioned to the right of it in the above table. Each one would involve two proofs by induction (in the manner of Exercise 6.5(a)).

3.2 Its complement is not context-free

The complement of the language is

$$\{x \mid x \text{ is not of the form } a^*b^*c^*\} \cup \{a^nb^nc^n \mid n \geq 0\}\,.$$

Let us call this language L'.

3.2.1 Solution 1

We can prove it by contradiction, so assume L' is context-free. By Theorem 6.13, we know that context-free languages are closed under intersection with regular languages; this tells us that $L' \cap (a^*b^*c^*)$ is context-free, but this is equal to $\{a^nb^nc^n \mid n \geq 0\}$, which we know to be non-context-free.

3.2.2 Solution 2

The proof is by contradiction. Assume L' is context-free, so the pumping lemma applies. Consider the string $u = a^nb^nc^n$, where n is the pumping lemma constant. Then the pumping lemma tells us that $u = vwxyz$ for some strings v, w, x, y, z satisfying conditions (i), (ii) and (iii).

Notice (by (i)) that wy cannot contain all three types of letters, and also (by (ii)) that wy must contain at least one letter. Thus, vw^0xy^0z fails to have an equal number of as, bs and cs. Furthermore, we know that vw^0xy^0z is of the form $a^*b^*c^*$ because it is simply obtained by removing some letters from u. Therefore, $vw^0xy^0z \notin L'$, contradicting (iii).

4 Exercise 6.12

4.1 Part a

4.1.1 Solution

Let L be a context-free language and let F be a finite language. I will prove that $L - F$ is a context-free language.

Since F is finite, we know that F is regular. Therefore, the complement F' of F must be regular too. By Theorem 6.13, $L \cap F'$ must be context-free. But $L - F = L \cap F'$ by definition, so $L - F$ must be context-free.

4.1.2 Discussion

What if F were regular, but not finite? In this case too (using the same reasoning), we see that $L - F$ is context-free.

4.2 Part b

4.2.1 Solution

Let L be a non-context-free language and let F be a finite language. I will prove by contradiction that $L - F$ is not a context-free language.

Suppose to the contrary that $L - F$ is a context-free language. Now notice that $L \cap F$ is finite, and therefore also context-free. Then $(L - F) \cup (L \cap F) = L$ is the union of two context-free languages, and therefore context-free. This contradicts the fact that L is not context-free.

4.2.2 Discussion

What if F were regular, but not finite? In this case, $L - F$ may be context-free. For example, if $F = \Sigma^*$ (which is regular, but not finite), then $L - F = \varnothing$, which is context-free.

5 Exercise 6.16(b)

Let G be a context-free grammar accepting a language L. I will make a grammar accepting prefixes(L).

Without loss of generality, we may assume that G is in Chomsky normal form and has no useless, unproductive symbols. The new grammar G' is defined as follows:

- G' has all the rules and symbols from G, plus some new stuff defined below.

- If X is a non-terminal in G, then X' is also a non-terminal in G'. So G' has twice as many non-terminals as G.

The idea is that X' is supposed to generate all prefixes of strings generated by X.

- $X' \to \Lambda$ is a production in G' (for each non-terminal X in G).
- If $X \to \sigma$ is a production in G, then $X' \to \sigma$ is also a production in G'.
- If $X \to YZ$ is a production in G, then $X' \to Y'$ and $X' \to YZ'$ are also productions in G'.
- S' is the start symbol of G' (where S is the start symbol of G).

It remains to prove that G' accepts prefixes(L). To do it, I will show that each symbol X' generates the prefixes of the strings generated by X. As usual, there are two parts:

Part 1 (everything generated by X' is a prefix). Let x' be generated by the non-terminal symbol X'. I will prove (by induction on the parse tree for x') that x' must be a prefix of some string x generated by X:

- If the root production is $X' \to \Lambda$, then $x' = \Lambda$, which is a prefix of everything. In particular, since X is not unproductive, we can take x to be any string generated by X in this case.
- If the root production is $X' \to \sigma$, then $x' = \sigma$. Additionally, we know that $X \to \sigma$ must also be in the grammar, so we can take $x = \sigma$.
- If the root production is $X' \to Y'$, then $x' = y'$ for some string y' generated by Y'. Our induction hypothesis says that y' is a prefix of some string y generated by Y. Additionally, we know that $X \to YZ$ must also be in the grammar (for some symbol Z). Since Z is not unproductive, we know that Z generates some string z. Now we can take $x = yz$; we can easily see that x' is a prefix of x, and we can also see that x is generated by X using the rule $X \to YZ$.
- If the root production is $X' \to YZ'$, then $x' = yz'$ for some string y generated by Y and some string z' generated by Z'. Our induction hypothesis says that z' is a prefix of some string z generated by Z. Now we can take $x = yz$; again we can easily see that x' is a prefix of x, and we can see that x is generated by X using the rule $X \to YZ$.

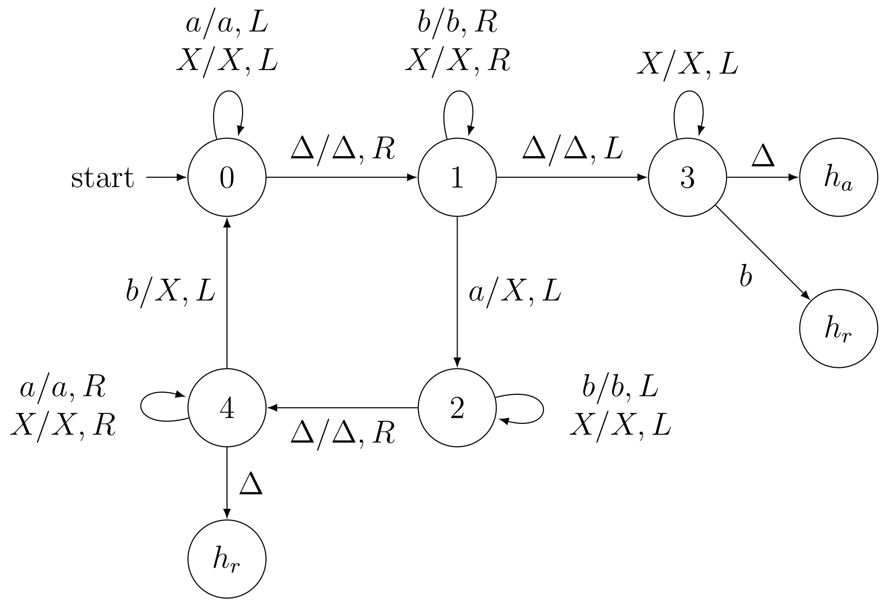
Part 2 (every prefix is generated by X'). Let x' be a prefix of a string x generated by the symbol X. I will prove (by induction on the parse tree for x) that x' is generated by X'.

- If the root production is $X \to \sigma$, then $x = \sigma$. In this case, either $x' = \Lambda$ or $x' = \sigma$, so x' can be generated using either $X' \to \Lambda$ or $X' \to \sigma$.
- If the root production is $X \to YZ$, then $x = yz$ for some strings y and z generated by Y and Z respectively. Our induction hypothesis says that every prefix of y is generated by Y' and that every every prefix of z is generated by Z'. Now, since x' is a prefix of x, we know that either x' is a prefix of y or $x' = yz'$ for some prefix z' of z. Therefore, we can generate x' using either $X' \to Y'$ or $X' \to YZ'$.

5.1 Alternative solution

You can also solve this problem by taking a PDA M for L and converting it into a PDA M' for prefixes(L). To do so, make two copies of M and smoosh them together to become M', so then M' has twice as many states as M. In the first copy of M, change all the final states into non-final states. In the second copy of M, change all the transitions into Λ transitions (so $\sigma, \gamma/\alpha$ becomes $\Lambda, \gamma/\alpha$). Finally, draw a transition labeled $\Lambda, \Lambda/\Lambda$ from each state in the first copy to its corresponding state in the second copy. The start state of M' should be the start state of the first copy.

6 Exercise 7.5(a)



In states 0, 1 and 3, the machine has changed an equal number of as and bs into Xs. In states 2 and 4, the number of as changed will be $n+1$ while the number of bs changed will be n.

- In state 0, the machine is moving left until it reaches the left edge of the string. Then it goes to state 1 to start searching for an a.
- In state 1, the machine is searching rightward for an a. If it finds one, then it changes it to an X and goes to state 2 to search for a b; otherwise, there must be no as remaining, so it goes to state 3 to make sure there are no bs remaining either.
- In state 2, the machine is moving left until it reaches the left edge of the string. Then it goes to state 4 to start searching for a b.
- In state 3, the machine is searching leftward for a b. If it finds one, then there must have been more bs than as in the string, so it rejects; otherwise, it accepts.
- In state 4, the machine is searching rightward for a b. If it finds one, then it changes it to an X and goes to state 0 to search for another

a; otherwise, there must have been more as than bs in the string, so it rejects.