

Problem Set 2 Solutions

Problem 1

One of the following two languages is regular and the other is not. Say which is which and prove your answer, either giving a regular expression for the language or proving non-regularity using the pumping lemma.

- $L_1 = \{a^n b^m \mid n \text{ and } m \text{ have the same remainder when divided by } 3\}$.
- $L_2 = \{a^n b^m \mid n \text{ and } m \text{ have the same quotient when divided by } 3\}$.

Solution:

The language L_1 is regular because keeping track of the remainder takes a finite number of states, and we can prove that by giving a regular expression for L_1 as the problem asks. The possible remainders when divided by 3 are 0, 1, and 2. Hence, we can write strings that belong to each of the three cases and make the union of them. $(aaa)^*(bbb)^*$ will recognize strings of the form $a^n b^m$, where n and m have the remainder 0 when divided by 3. $a(aaa)^*b(bbb)^*$ will recognize strings of the form $a^n b^m$, where n and m have the remainder 1 when divided by 3. $aa(aaa)^*bb(bbb)^*$ will recognize strings of the form $a^n b^m$, where n and m have the remainder 2 when divided by 3. We now make the union of the three regular expressions to give a regular expression that recognizes the language L_1 .

$$(aaa)^*(bbb)^* \cup a(aaa)^*b(bbb)^* \cup aa(aaa)^*bb(bbb)^*$$

The language L_2 is not regular and we prove it using the pumping lemma. Assume for contradiction that L_2 is a regular language. Then, there exists a number p such that any string $w \in L_2$ of length at least p can be divided into three pieces, $w = xyz$, such that the following conditions hold:

1. $xy^kz \in L_2$ for all $k \geq 0$
2. $y \neq \epsilon$
3. $|xy| \leq p$

Let q be the quotient of p when divided by 3. Then, p can be represented as $p = 3q + r$, where q is the quotient and r is the remainder. Let $w = a^p b^p$. Clearly, $w \in L_2$ and $|w| \geq p$, and we can divide $w = a^p b^p$ into three parts xyz such that

$|xy| \leq p$, where $|y| > 0$. Since $|xy| \leq p$, the string xy will consist of a 's only, and z will consist of 0 or more a 's and p b 's. So we can represent x , y , and z as follows: $x = a^i$, $y = a^j$, and $z = a^{p-i-j}b^p$, where $i \geq 0$, $j > 0$. Since the first condition states that $xy^kz \in L_2$ for all $k \geq 0$, we can let k to be any number and the resulting string should still be in the language L_2 . Let $k = 4$ in xy^kz (in other words let us pump y for 4 times). Then, $xy^4z = a^i(a^j)^4a^{p-i-j}b^p = a^{p+3j}b^p$. Let us examine if the resulting string is still in the language L_2 , meaning the number of a 's (i.e. $p + 3j$) and the number of b 's (i.e. p) have the same quotient when divided by 3. In order to do so, we need to check if the quotient of $p + 3j$ (the number of a 's) is equal to the quotient of p (the number of b 's). Recall that $p = 3q + r$ (i.e. the quotient of p is q). So, we can represent $p + 3j$ as $p + 3j = 3q + r + 3j = 3(q + j) + r$, which means that the quotient of $p + 3j$ is $q + j$. Since $j > 0$, we know that $q + j \neq q$. This means that the resulting string xy^4z is not in the language L_2 , which is a contradiction. Hence, the language L_2 is not regular.

Problem 2

In the last problem set we defined the even and odd part of a string ($even(w_1w_2w_3\dots) = w_2w_4w_6\dots$ and $odd(w_1w_2w_3\dots) = w_1w_3w_5\dots$) and proved that if L is a regular language, then also $even(L) = \{even(w) \mid w \in L\}$ and $odd(L) = \{odd(w) \mid w \in L\}$ are regular. (See problem set 1 for details.)

Prove that the converse is not necessarily true, i.e. show that there exists a language L such that both $even(L)$ and $odd(L)$ are regular, but L is not regular. [Hint: one of the languages from problem 1.23 in the textbook should do.] Notice: after you choose a language L , you have both to prove that $odd(L)$ and $even(L)$ are regular (e.g., giving a regular expression or finite state automaton for them), and prove that L is not regular (using the pumping lemma and/or the closure properties of regular languages.)

Solution:

Consider the language $L = \{0^n1^n : n \geq 0\}$. It is not hard to see that $\bar{L} = \{0^n1^m : n \neq m\} \cup (0 \cup 1)^*10(0 \cup 1)^*$. Let's denote $\{0^n1^m : n \neq m\}$ as L_1 and $(0 \cup 1)^*10(0 \cup 1)^*$ as L_2 . Then $\bar{L} = L_1 \cup L_2$.

We want to compute the *odd* and *even* part of this language. Let's address $odd(\bar{L})$ first. Clearly,

$$odd(\bar{L}) = odd(L_1 \cup L_2) = odd(L_1) \cup odd(L_2)$$

Since L_2 is regular, we know that $odd(L_2)$ is regular (by problem 3, homework 1). Therefore, it remains to be proven that $odd(L_1)$ is regular. If that is the case, we would have that $odd(\bar{L})$ is the union of two regular languages, hence regular.

Let's prove that $odd(L_1)$ is regular. First, we need to have an idea about how $odd(L_1)$ looks like. For any $w \in L_1$, by definition of L_1 , $w = 0\dots 01\dots 1$ where the

number of 0's and 1's is not the same. Clearly, $w' \in \text{odd}(L_1)$ must look like $w' = 0^s 1^t$ where s, t are “around” $n/2$ and $m/2$ (respectively), for some $n \neq m$. If s, t are exactly $n/2, m/2$ or $\lfloor n/2 \rfloor, m/2$ or $\lceil n/2 \rceil, \lfloor m/2 \rfloor$ or something similar, depends on whether n, m are even or odd numbers. However, instead of checking all the different cases (n, m even, even; or even, odd; etc.), we take a different approach.

We will consider an arbitrary word $w' = 0^s 1^t$ in $\text{odd}(L_1)$ (for some s, t) and will prove that we can always “fill in the blanks” of a new word w such that $\text{odd}(w) = w'$ and $w \in L_1$. Indeed, consider $w' = 0^s 1^t$ for some s, t , such that $s + t > 0$. (The case $s = t = 0$ would imply that $\epsilon \in \text{odd}(L_1)$ which is not possible because $\epsilon \notin L_1$.) We imagine w as taking symbols of w' while placing blanks (“□”) every each other. Pictorially,

$$w = 0 \square 0 \square \dots \square 0 \square 1 \square 1 \square \dots \square 1 \square$$

How can we fill in the blanks? From the picture, we can conclude that if $w = 0^n 1^m$ then $n \in \{2s - 1, 2s\}$, and $m \in \{2t - 1, 2t, 2t + 1\}$. Moreover, the only two pairs not allowed are $(n, m) = (2s, 2t + 1)$ (it would mean that the blank between the 0 and 1 in the middle of w is filled with both a 0 and a 1) and $(n, m) = (2s - 1, 2t - 1)$ (which would mean that we did not fill in the middle blank at all). All the other pairs (ie. $(n, m) = (2s - 1, 2t), (2s - 1, 2t + 1), (2s, 2t)$) are allowed. To conclude, we only need to notice that, for *any* two values of $s, t \geq 0$, there exists one of the above pairs of values (n, m) such that $n \neq m$. Indeed, if $s = t$, we choose $(n, m) = (2s - 1, 2t)$ or $(2s - 1, 2t + 1)$; if $s \neq t$ we choose $(2s, 2t)$. Therefore, starting from any word $w' = 0^s 1^t \in \text{odd}(L_1)$ (where $s, t, s + t > 0$ are arbitrary positive integers) we can always create a word $w \in L_1$.

The argument above proved that $0^* 1^* \setminus \{\epsilon\} \subseteq \text{odd}(L_1)$. By definition of L_1 and the $\text{odd}(\cdot)$ operation, we also know that $\text{odd}(L_1) \subseteq 0^* 1^* \setminus \{\epsilon\}$. Thus, we finally have that $\text{odd}(L_1) = 0^* 1^* \setminus \{\epsilon\}$, and therefore, $\text{odd}(\overline{L})$ is regular.

The proof that $\text{even}(\overline{L})$ is regular is analogous.

Remark 1 Thanks to Chalena Jimenez for noticing that $\epsilon \notin \text{odd}(L_1)$.

Problem 3

For each of the following languages give a Push Down Automaton that accepts exactly that language.

- $L_3 = \{a^n b^m c^k \mid n + m \leq k\}$
- $L_4 = \{a^n b^m c^k \mid \min(n, m) \leq k\}$

(You should describe the PDA's giving a state transition diagram. You are not required to give the formal description with the table for the transition function.)

[Optional (don't write anything, just think about this part): what can you say about the language $L_5 = \{a^n, b^m c^k \mid \max(n, m) \leq k\}$? Can you give a Push down automaton for it?]

Solution:

In order to build a PDA for L_3 , we need to be able to somehow keep track of the total number of a 's and b 's and compare it with the number of c 's. We can do that by using the stack in the PDA. First, when a 's are read as input, push them unto the stack and then when b 's read next, push them onto also unto the stack and finally when c 's are read, pop a symbol on the stack for every c read. When the stack is empty, and there are no more symbols to read, the input is accepted. If there are more c 's to read even if the stack is empty, just read the symbols (i.e. c 's) doing nothing to the stack until there is none left to read, and accept the input. This takes care of the case when the number of c 's is greater than the total number of a 's and b 's. One thing to notice is that the language accepts the empty string ϵ . Since $n + m \leq k$ does not impose any restriction on the values of n, m, k , they can be all 0's and still satisfy the equation. Note also that either n or m can be 0 as long as the equation $n + m \leq k$ holds (i.e. the string can be of the form $a^n c^k$ or $b^m c^k$ or c^k . Hence, you need to take these into account when you construct the PDA for L_3 (i.e. using the power of non-determinism, make ϵ transitions).

Let $M_3 = (Q, \Sigma, \Gamma, \delta, s, F)$ be a PDA for L_3 . The formal description of M_3 (except the transition table) is given below:

$$\begin{aligned} Q &= \{s, 1, 2, 3, 4, f\} \\ \Sigma &= \{a, b, c\} \\ \Gamma &= \{a, b, \$\} \\ s &= s \\ F &= \{f\} \end{aligned}$$

The state transition diagram for M_3 is given in Figure 1.

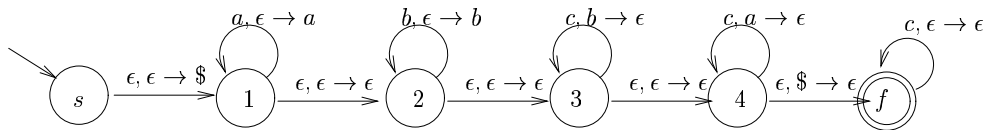


Figure 1: State transition diagram of the PDA M_3 that recognizes the language L_3

Now we move to the language L_4 . The idea is similar to that for L_3 , but it exploits the power of non-determinism even more than the PDA of L_3 . The language L_4 contains the strings of the form $a^n b^m c^k$ where $\min(n, m) \leq k$. Since we do not know

which one is less between n and m , we need to construct a PDA that can handle both cases: one for $n \leq m$ ($\min(n, m) = n$) and the other for $n > m$ ($\min(n, m) = m$). If we guess that $n \leq m$, then we can keep track of the number of a 's only pushing a 's on the stack and ignoring the number of b 's, and then for every c that is read, pop an a (in a similar way as done for M_3). For the other case ($\min(n, m) = m$), we ignore the number of a 's and keep track of b 's and pop a b for every c read. In the actual configuration, when the input symbols are processed, we don't know which path to take since we don't know whether the number of a 's is less than the number of b 's before we read the string. So, we take a guess and suppose we guessed that $n = \min(n, m)$. We read a 's and ignored b 's and suppose it turned out the string is accepted. What if our guess was wrong? Should we still accept? The answer is yes, because if we accepted the string guessing wrong, that means that the number of c 's is less than the number of the correct symbols anyway. For example, suppose we guessed $n \leq m$ and the string was accepted. Then it implies $n \leq k$. Then, even if our guess was wrong (i.e. $m \leq n$), and a wrong guess on the minimum implies $m \leq n \leq k$. If we rejected the input string by a wrong guess, we can still take the other path with the correct guess on the minimum and get the correct result. (Recall that a string is accepted if there exists at least one accepting configuration.) Hence, the PDA accepts exactly the strings in the language L_4 .

Let $M_4 = (Q, \Sigma, \Gamma, \delta, s, F)$ be a PDA for L_4 . The formal description of M_4 (except the transition table) is given below:

$$\begin{aligned} Q &= \{s, 1, 2, 3, 4, 5, 6, 7, 8\} \\ \Sigma &= \{a, b, c\} \\ \Gamma &= \{a, b, \$\} \\ s &= s \\ F &= \{4, 8\} \end{aligned}$$

The state transition diagram for M_4 is given in Figure 2.

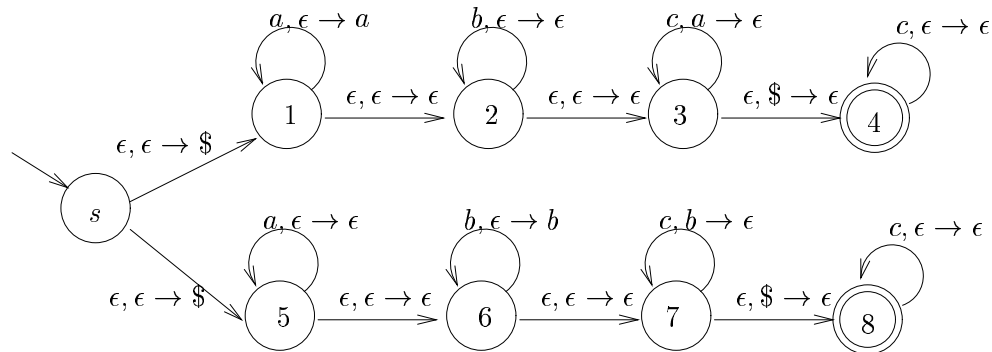


Figure 2: State transition diagram of the PDA M_4 that recognizes the language L_4

Problem 4

Give context free grammars that generates the languages L_3, L_4 from problem 3.

[Optional (don't write anything, just think about this part): as in problem 3, do you think the language $L_5 = \{a^n, b^m c^k \mid \max(n, m) \leq k\}$ can be described by a context free grammar?]

Solution:

In order to design a grammar that generates language $L_3 = \{a^n b^m c^k : n + m \leq k\}$ we have to “think on generating” instead of “recognizing” (as in DFAs or PDAs). The relation $n + m \leq k$ can be translated to $n + m + l = k$ where $l \geq 0$. Therefore, we have an alternative definition for language $L_3 = \{a^n b^m c^{n+m} c^l : n, m, l \geq 0\}$ or using regular expression notation $L_3 = \{a^n b^m c^{n+m} c^* : n, m \geq 0\}$. You may check that this definition is equivalent to the definition of L_3 already given.

If we look closely, all strings in this language have the following properties: (1) every time a symbol a is generated, one or more symbols c are generated too; (2) the same happens for b symbols; and (3) there are some c 's (zero or more) at the end of the word.

A grammar that generates the language L_3 is then the following:

$$\begin{array}{lcl} S & \longrightarrow & XT \\ X & \longrightarrow & aXc \mid Y \\ Y & \longrightarrow & bYc \mid \epsilon \\ T & \longrightarrow & cT \mid \epsilon \end{array}$$

Now, we need to design a grammar for language $L_4 = \{a^n b^m c^k : \min(n, m) \leq k\}$. If we notice that $\min(n, m) \leq k$ can be expressed as the compound statement “($n \leq m$ and $n \leq k$) or ($n \geq m$ and $m \leq k$)” we can re-write the definition of the language as follows:

$$\begin{aligned} L_4 &= \{a^n b^m c^k : \min(n, m) \leq k\} \\ &= \{a^n b^m c^m c^l : n, m, l \geq 0\} \cup \{a^n b^m c^n c^l : n, m, l \geq 0\} \\ &= L_A \cup L_B \end{aligned}$$

Therefore, we first create a grammar that recognizes L_A :

$$\begin{array}{lcl} S & \longrightarrow & X_1 T \\ X_1 & \longrightarrow & aX_1 \mid Y_1 \\ Y_1 & \longrightarrow & bY_1 c \mid \epsilon \\ T & \longrightarrow & cT \mid \epsilon \end{array}$$

and then a grammar that recognizes L_B :

$$\begin{array}{lcl}
S & \longrightarrow & X_2T \\
X_2 & \longrightarrow & aX_2c \quad | \quad Y_2 \\
Y_2 & \longrightarrow & bY_2 \quad | \quad \epsilon \\
T & \longrightarrow & cT \quad | \quad \epsilon
\end{array}$$

To obtain our grammar for L_4 we combine the two grammar shown above with a rule $S \rightarrow S_1 \mid S_2$, as follows:

$$\begin{array}{lcl}
S & \longrightarrow & S_1 \quad | \quad S_2 \\
\\
S_1 & \longrightarrow & X_1T \\
X_1 & \longrightarrow & aX_1 \quad | \quad Y_1 \\
Y_1 & \longrightarrow & bY_1c \quad | \quad \epsilon \\
\\
S_2 & \longrightarrow & X_2T \\
X_2 & \longrightarrow & aX_2c \quad | \quad Y_2 \\
Y_2 & \longrightarrow & bY_2 \quad | \quad \epsilon \\
T & \longrightarrow & cT \quad | \quad \epsilon
\end{array}$$
