

## Problem Set 4 Solutions

**Problem 1**

For **each** of the following languages, say whether the language is regular, context-free or decidable, and prove your answer. In particular,

- if the language is regular give a *regular expression* for  $L$ .
- If  $L$  is context free, but not regular, give a *context free grammar* for  $L$ , **and** prove that  $L$  is not regular using the *pumping lemma for regular languages*.
- Finally, if  $L$  is decidable but not context free, give a *Turing machine* for  $L$  (the transition function for this Turing machine should be described with state diagram) **and** prove that  $L$  is not context free using the *pumping lemma for context free languages*.

(a)

$$L_1 = \{a^n b^m c^{n+m} \mid n, m \geq 0\}$$

(b)

$$L_2 = \{a^n b^{2n} c^{3n} \mid n \geq 0\}$$

(c)

$$L_3 = \{a^n b^{2m} c^{3k} \mid n, m, k \geq 0\}$$

**Solution:**

(a) The language  $L_1 = \{a^n b^m c^{n+m} \mid n, m \geq 0\}$  is context-free. We prove that it is context free by giving a context free grammar  $G = (\{S, T\}, \{a, b, c\}, R, S)$  for  $L_1$ , where the set of rules,  $R$ , is defined as follows:

$$\begin{aligned} S &\rightarrow aSc \mid T \\ T &\rightarrow bTc \mid \epsilon \end{aligned}$$

We now prove that  $L_1$  is not regular using the pumping lemma for regular languages as follows:

Assume for contradiction that  $L_1$  is regular, 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  $w = a^p b^p c^{2p}$ . Clearly  $w \in L_1$  and  $|w| \geq p$ , and we can divide  $w$  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 and  $2p$   $c$ 's. So  $x, y, z$  can be represented as follows:  $x = a^i$ ,  $y = a^j$ , and  $z = a^{p-i-j} b^p c^{2p}$ , where  $i \geq 0$  and  $j > 0$ . Since the first condition states that  $xy^kz \in L_1$  for all  $k \geq 0$ , we can let  $k$  to be any number and the resulting string should still be in the language  $L_1$ . Let  $k = 0$ . Then,  $xy^0z = a^i (a^j)^0 a^{p-i-j} b^p c^{2p} = a^{p-j} b^p c^{2p}$ . Since  $j > 0$ ,  $p - j + p \neq 2p$ . Hence,  $xy^0z \notin L_1$ , which is a contradiction. This proves that  $L_1$  is not regular.

(b) The language  $L_2 = \{a^n b^{2n} c^{3n} \mid n \geq 0\}$  is decidable. We prove it by giving a Turing machine that decides  $L_2$ . The Turing machine, which we denote by  $M = (Q, \Sigma, \Gamma, \delta, q_s, q_a, q_r)$ , is described as follows:  $Q = \{q_s, 1, 2, 3, 4, 5, 6, q_a, q_r\}$ ,  $\Sigma = \{a, b, c\}$ ,  $\Gamma = \{a, b, c, /a, /b, /c, \sqcup\}$ , the start state is  $q_s$ , the accept state is  $q_a$ , and the reject state is  $q_r$ . We describe the transition function with a state diagram as the problem asks. The state diagram for the Turing machine  $M$  is given in Figure 1.

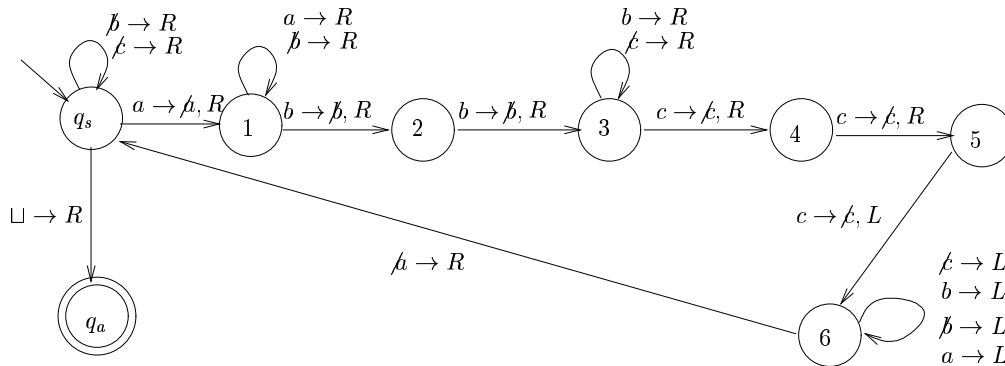


Figure 1: State diagram for the TM  $M$  that recognizes the language  $L_2$

Note that the reject state does not appear in the state diagram because, for simplicity, the state diagram is described using the convention that all the transitions for the tape alphabets that are not shown in each state go to the reject state; hence, the reject state itself and all the transitions to the reject states are omitted.

We now show that the language  $L_2$  is not context free using the pumping lemma for context free languages as follows:

Assume for contradiction that  $L_2$  is context free. Then, by the pumping lemma for context free languages, we know that there exists a pumping length  $p > 0$  such that any string  $w \in L_2$ ,  $|w| \geq p$ , can be divided into five parts,  $w = uvxyz$ , such that  $|vxy| \leq p$ ,  $|vy| > 0$ , and the string  $uv^kxy^kz$  also belongs to  $L_2$  for each  $k \geq 0$ .

Consider the string  $w = a^p b^{2p} c^{3p}$ . Clearly,  $w \in L_2$ , and  $|w| \geq p$ . Consider how  $w$  can be divided into five parts  $uvxyz$ —in particular, what kinds of symbols  $v$  and  $y$  can include. Note that since  $v$  and  $y$  can be “pumped up” (i.e. repeated more than once) and the resulting string should still be in the language, none of  $v$  and  $y$  should include different symbols (i.e. symbols of the form  $a^i b^j$  or  $b^i c^j$  for some  $i, j > 0$ ), since otherwise the resulting string will be out of order and therefore will not be in the language  $L_2$  when  $v$  and  $y$  are repeated more than once. Moreover, because of the condition  $|vxy| \leq p$ ,  $vxy$  cannot include both  $a$ 's and  $c$ 's. Hence, any “valid” partition of  $w$  into  $uvxyz$  must fall into exactly one of the following two cases:

1. Both  $v$  and  $y$  contain the same symbol—the substring  $vxy$  is composed of only  $a$ 's, or only  $b$ 's or only  $c$ 's. For example, in the case where  $v = a^i$  and  $y = a^j$ , if we pump down  $v$  and  $y$  (i.e.  $k = 0$  in  $uv^kxy^kz$ ), the resulting string  $uxz$  will have at least one  $a$  less than the original string  $w$  (because  $|vy| > 0$ ) while the number of  $b$ 's and  $c$ 's remain the same, and the relationship between the numbers of  $a$ 's and  $b$ 's and  $c$ 's would no longer hold; hence,  $uxz \notin L_2$ . Similar arguments hold for the other two cases, where  $vxy$  consists of only  $b$ 's or only  $c$ 's. Pumping down  $v$  and  $y$  reduces the number of the symbol that belongs to  $v$  and  $y$  by at least one (due to the condition  $|vy| > 0$ ) and makes the resulting string no longer belong to  $L_2$ .
2.  $v$  and  $y$  contain different symbols—the following two sub-cases are possible for this:  $v$  contains only  $a$ 's and  $y$  contains only  $b$ 's or  $v$  contains only  $b$ 's and  $y$  contains only  $c$ 's. In both cases, pumping down (or up)  $v$  and  $y$  will result in a string that is not in the language  $L_2$  since the number of at least one symbol that is not pumped down or up remains the same while the number of at least one symbol that is pumped down (or up) changes.

Notice that the contradictions described above can be easily obtained by observing that the structure of the string  $a^p b^{2p} c^{3p}$  ( $p > 0$ ) is pretty strict in that changing the numbers of any one or two symbols will make the resulting string not in the language as long as the number of at least one symbol remains the same. In order to make the string still be in the language after changing the numbers of symbols, one needs to change all three parts: for example, if the number of  $a$ 's is increased by 1, then the number of  $b$ 's and  $c$ 's need to be increased by 2 and 3, respectively, for the resulting string to be still in the language. However, we know that increasing the numbers of all three symbols is not possible since the pumping parts  $v$  and  $y$  cannot contain all

three symbols due to the condition  $|vxy| \leq p$ . Hence, when the string is pumped in some way, it no longer belongs to  $L_2$ , which is a contradiction. Therefore, We can conclude that  $L_2$  is not context free.

(c) The language  $L_3 = \{a^n b^{2m} c^{3k} \mid n, m, k \geq 0\}$  is regular, and we prove this by giving a regular expression for  $L_3$  as follows:  $a^*(bb)^*(ccc)^*$ . Note that the difference between  $L_3$  and  $L_2$  is in the relationship between the number of  $a$ 's and  $b$ 's and  $c$ 's. In  $L_3$ , there is no relationship between  $n, m, k$  and the number of  $a$ 's can be any non-negative integer and the number of  $b$ 's can be any multiple of 2 and the number of  $c$ 's can be any multiple of 3. The absence of the relationship between the numbers of the symbols (i.e.  $a$ 's,  $b$ 's, and  $c$ 's) makes the language  $L_3$  regular while its presence makes  $L_2$  not regular (nor context free).

## Problem 2

A Turing machine with doubly infinite tape is defined similarly to an ordinary Turing machine, except that the memory tape is infinite in both directions. Initially the memory tape contains the input string, and the read/write head is positioned on the first character of the input. The formal definition is identical to ordinary Turing machines except that the *yields* relation gives

$$qb\beta \Rightarrow q' \sqcup c\beta$$

instead of  $qb\beta \Rightarrow q'c\beta$  when  $\delta(q, b) = (q', c, L)$ . Prove that this type of Turing machine recognizes exactly the same class of languages as ordinary Turing machines. In particular you should

- (a) show that any ordinary Turing machine can be transformed into an equivalent Turing machine with doubly infinite tape
- (b) show that any Turing machine with doubly infinite tape can be transformed into an equivalent ordinary Turing machine.

Here “equivalent” means that the two machines recognize the same language. For both parts you should first give an informal description of the Turing machine you intend to build, followed by the formal definition of the machine.

### Solution:

**Part (a):** In this case, we have a Turing machine with doubly infinite tape (DIT)  $M_{\mathcal{D}}$  available. We want to show that, given a description of an ordinary Turing machine  $M_{\mathcal{O}}^1$ , we can simulate  $M_{\mathcal{O}}$  with  $M_{\mathcal{D}}$ ; this means it is possible to build

<sup>1</sup>Here the sub-indexes  $\mathcal{D}$  and  $\mathcal{O}$  stand for “double” and for “ordinary” respectively.

$M_{\mathcal{D}}$  using  $M_{\mathcal{O}}$ 's components (i.e. set of states, transition function, tape alphabet) so that both  $M_{\mathcal{O}}$  and  $M_{\mathcal{D}}$  accept the same language.

Let  $M_{\mathcal{O}} = (Q, \Sigma, \Gamma, \delta, q_1, q_{acc}, q_{rej})$  be the description of an ordinary Turing machine, and  $M_{\mathcal{D}} = (Q', \Sigma', \Gamma', \delta', q'_1, q'_{acc}, q'_{rej})$  be the new DIT Turing machine we want to build so  $\mathcal{L}(M_{\mathcal{O}}) = \mathcal{L}(M_{\mathcal{D}})$ .

The idea of the simulation comes from a rather obvious fact: to simulate a machine with “half” a tape using a machine with one “whole” tape, simply don't use half of it. Which “half”? The half containing the input string – from the the cell containing the first input symbol all the way to the right. Additionally, we need some mechanism to prevent our simulation violates the computation rule “ $qa\beta \Rightarrow q'c\beta$  if  $\delta(q, a) = (q', c, L)$ ” (i.e. the one which says that any attempt to move off the left hand of the tape leaves the head back in the first cell). In order to do so, we use a special character (say #) which will mark where our “half” tape begins; this is, we initially place the symbol # in the rightmost blank cell at the left of the input in  $M_{\mathcal{D}}$ 's tape.

Informally, the execution of  $M_{\mathcal{D}}$  is as follows:

$M_{\mathcal{D}}$  = “On input  $w = w_1 \cdots w_n$  :

1. First  $M_{\mathcal{D}}$  puts its tape into the format described above, this is, it inserts a # in the first cell to the left of the input  $w$ . Therefore, the contents of  $M_{\mathcal{D}}$ 's tape become “...  $\square \square \# w_1 w_2 \dots w_n \square \square \dots$ ”
2. We duplicate  $M_{\mathcal{O}}$ 's program into  $M_{\mathcal{D}}$  by embedding  $M_{\mathcal{O}}$ 's set of states and transition function into  $M_{\mathcal{D}}$  with only one caveat: if the simulation ever attempts to move  $M_{\mathcal{D}}$ 's head on the symbol #, we set  $M_{\mathcal{D}}$  to move its head back to the first cell to the right of the symbol # (we'll say that the head *bounces* back).
3. If the simulation of  $M_{\mathcal{O}}$  ends in an accepting state of  $M_{\mathcal{O}}$ ,  $M_{\mathcal{D}}$  accepts; otherwise,  $M_{\mathcal{D}}$  rejects.”

Formally, the description of machine  $M_{\mathcal{D}} = (Q', \Sigma', \Gamma', \delta', q'_1, q'_{acc}, q'_{rej})$ . is as follows:

- $Q' = Q \cup \{ r_1, r_2 \}$ , where  $r_1, r_2$  are the states needed to implement (a) the procedure to prepend a # to the input, and (b) the procedure that makes the head bounce back,
- $\Sigma' = \Sigma$ ,
- $\Gamma' = \Gamma \cup \{ \# \}$ ,
- $q'_1 = r_1$ ,
- $q'_{acc} = q_{acc}$ ,
- $q'_{rej} = q_{rej}$ ,

- The transition function  $\delta'$  is defined as follows:

$$\delta'(q, x) = \begin{cases} (r_2, x, L) & \text{if } q = r_1, \text{ for all } x \in \Sigma', \\ (q_1, \#, R) & \text{if } q = r_2, \text{ and } x = \sqcup, \\ (q, \#, R) & \text{if } x = \#', \text{ for all } q \in Q, \\ \delta(q, x) & \text{otherwise.} \end{cases}$$

**Part (b):** This case is more elaborated than the previous one. Now, we have only an ordinary Turing machine  $M_{\mathcal{O}} = (Q, \Sigma, \Gamma, \delta, q_1, q_{acc}, q_{rej})$  available and we want to show that, given a description of a Turing machine with doubly infinite tape (DIT)  $M_{\mathcal{D}} = (Q', \Sigma', \Gamma', \delta', q'_1, q'_{acc}, q'_{rej})$ , we can simulate  $M_{\mathcal{D}}$  with  $M_{\mathcal{O}}$ .

The idea of the simulation in this case is to add new symbols to  $M_{\mathcal{O}}$ 's tape alphabet so each new symbol represents *two* symbols of  $M_{\mathcal{D}}$ 's, in a solution known as “folding the tape”. This new encoding will allow us to operate as there were two tracks on  $M_{\mathcal{O}}$ 's tape. For example, if  $M_{\mathcal{D}}$ 's double infinite tape contains

...	□	□	$u_4$	$u_3$	$u_2$	$u_1$	$v_0$	$v_1$	$v_2$	$v_3$	$v_4$	$v_5$	□	□	...
-----	---	---	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	---	---	-----

where  $u_i, v_i \in \Sigma'$  and the initial position of the input head is on symbol  $v_0$ , then we represent this tape in  $M_{\mathcal{O}}$  as

$u_1$	$u_2$	$u_3$	$u_4$	□	□	□	□	...
#	$v_1$	$v_2$	$v_3$	$v_4$	$v_5$			

where  $\begin{matrix} u_i \\ v_i \end{matrix}$  will be a new symbol in  $M_{\mathcal{O}}$ 's tape alphabet for all pair of symbols  $u_i, v_i$  in  $M_{\mathcal{D}}$ 's tape alphabet.

Thus,  $M_{\mathcal{O}}$ 's tape now contains two “tracks”: the upper track stores the contents of the left “half” of  $M_{\mathcal{D}}$ 's tape (all the cells to the left of the cell where the input head is) while the lower track stores the contents of the right “half” (all the cells to the right of, and including, the cell where the input head is). We can now simulate the operation of  $M_{\mathcal{D}}$  by using  $M_{\mathcal{O}}$ 's upper track if  $M_{\mathcal{D}}$ 's head is in the left “half” of the tape and the lower track if  $M_{\mathcal{D}}$ 's head is in the right “half” of the tape.

Informally, the machine  $M_{\mathcal{O}}$  executes as follows:

$M_{\mathcal{O}}$  = “On input  $w = w_1 \cdots w_n$  :

1. First  $M_{\mathcal{O}}$  puts its tape into the format described above, this is, if the tape initially contains

$w_1$	$w_2$	$w_3$	$w_4$	...	$w_n$	□	...
-------	-------	-------	-------	-----	-------	---	-----

it gets transformed into

□	□	□	□	□	□	□	□	□	...
#	$w_1$	$w_2$	$w_3$	$w_4$	...	$w_n$			

Notice that this step involves two procedures: one that inserts the symbol  $\#$  in the first cell and another that transforms each symbol  $w_i$  into a symbol  $\begin{array}{|c|} \hline \square \\ \hline w_i \\ \hline \end{array}$ .

2. While  $M_{\mathcal{D}}$ 's head is to the right of the initial position of its input head,  $M_{\mathcal{O}}$  works on the upper track; while  $M_{\mathcal{D}}$ 's head is to the left of its initial tape head position,  $M_{\mathcal{O}}$  works on its lower track, moving in the direction opposite to the direction  $M_{\mathcal{D}}$  moves.
3. If the simulation of  $M_{\mathcal{O}}$  ends in an accepting state of  $M_{\mathcal{D}}$ 's,  $M_{\mathcal{O}}$  accepts; otherwise,  $M_{\mathcal{O}}$  rejects."

Formally, the description of machine  $M_{\mathcal{O}}$  is as follows:

- $Q = Q' \cup \{s_1\} \cup \{s_x : x \in \Sigma'\} \cup \{q_{rew}, q_{conv}, t_{rew}\} \cup \{q_i^U, q_i^D : q_i \in Q'\}$ , where states  $\{s_1\} \cup \{s_x : x \in \Sigma'\}$  are needed to implement the procedure that inserts a  $\#$  in the first cell (the *#-insertion procedure*),  $\{q_{rew}, t_{rew}\}$  are auxiliary states to reposition the head on the first cell,  $\{q_{conv}\}$  implements the procedure that initially transforms each input symbol  $w_i$  to a symbol  $\begin{array}{|c|} \hline \square \\ \hline w_i \\ \hline \end{array}$  (the *input conversion procedure*), and  $\{q_i^U, q_i^D : q_i \in Q'\}$  are states to "remember" that  $M_{\mathcal{D}}$  was on state  $q_i$  either using the upper track ( $U$  for up) or the lower track ( $D$  for down).
- $\Sigma = \Sigma'$ ,
- $\Gamma = \Sigma \cup \left\{ \begin{array}{|c|} \hline u \\ \hline v \\ \hline \end{array} : u, v \in \Gamma' \right\} \cup \left\{ \begin{array}{|c|} \hline \square \\ \hline \# \\ \hline \end{array} \right\}$ ,
- $q_1 = s_1$ ,
- $q_{acc} = q'_{acc}$ ,
- $q_{rej} = q'_{rej}$ ,

- The transition function  $\delta$  is defined as follows:

$$\delta(q, x) = \left\{ \begin{array}{ll} (s_x, \#, R) & \text{if } q = s_1, \text{ for all } x \in \Sigma', \\ (s_x, y, R) & \text{if } q = s_y, \text{ for all } x \in \Sigma' \setminus \{\sqcup\}, \\ (q_{rew}, y, L) & \text{if } q = s_y, \text{ and } x = \sqcup, \\ (q_{rew}, x, L) & \text{if } q = q_{rew}, \text{ for all } x \in \Sigma' \setminus \{\#\}, \\ (q_{conv}, \begin{array}{|c|} \hline \sqcup \\ \hline \# \\ \hline \end{array}, R) & \text{if } q = q_{rew} \text{ and } x = \#', \\ (q_{conv}, \begin{array}{|c|} \hline \sqcup \\ \hline x \\ \hline \end{array}, R) & \text{if } q = q_{conv}, \text{ for all } x \in \Sigma' \setminus \{\sqcup\}, \\ (t_{rew}, \sqcup, L) & \text{if } q = q_{conv}, \text{ and } x = \sqcup, \\ (t_{rew}, \sqcup, L) & \text{if } q = t_{rew}, \text{ for all } x \in \Sigma \setminus \left\{ \begin{array}{|c|} \hline \sqcup \\ \hline \# \\ \hline \end{array} \right\}, \\ (q_1^D, \begin{array}{|c|} \hline \sqcup \\ \hline \# \\ \hline \end{array}, R) & \text{if } q = t_{rew}, \text{ and } x = \begin{array}{|c|} \hline \sqcup \\ \hline \# \\ \hline \end{array}, \\ (q_j^D, \begin{array}{|c|} \hline v \\ \hline z \\ \hline \end{array}, T) & \text{if } q = q_i^D, x = \begin{array}{|c|} \hline v \\ \hline y \\ \hline \end{array}, \text{ and } \delta'(q_i, y) = (q_j, z, T), \\ & \text{where } T \in \{L, R\}, \\ (q_i^U, \begin{array}{|c|} \hline v \\ \hline z \\ \hline \end{array}, R) & \text{if } q = q_i^D, x = \begin{array}{|c|} \hline \sqcup \\ \hline \# \\ \hline \end{array}, \\ (q_j^U, \begin{array}{|c|} \hline z \\ \hline v \\ \hline \end{array}, S) & \text{if } q = q_i^U, x = \begin{array}{|c|} \hline y \\ \hline v \\ \hline \end{array}, \text{ and } \delta'(q_i, y) = (q_j, z, T), \\ & \text{where } S, T \in \{L, R\}, \text{ but } S \neq T, \end{array} \right.$$

We can understand this transition function as follows: the first four cases of the transition function implement the  $\#$ -insertion procedure, the next two cases implement the input conversion procedure, the 7-th through 8-th cases allow repositioning the head on the second cell, the 9-th case implements the simulation when working on the upper track, the 10-th case handles moving from upper to lower track, the 11-th case implements the simulation when working on the lower track, and the 10-th case handles moving from lower to upper track.

**Remark 1** It is also possible to simulate  $M_{\mathcal{D}}$  with  $M_{\mathcal{O}}$  by inserting a space on the tape each time  $M_{\mathcal{D}}$  attempts to move to the left of the initial input head position. Since this solution is easier to describe is left as exercise  $\smile$ .