Concordia University

Dept. of Computer Science and Software Engineering

COMP 335 – Introduction to Theoretical Computer Science

Solutions for $\mathcal{A}$ssignment 4

1. (a). (Reversal): Let L be any context-free language and $G = (V, \Sigma, S, P)$ be a CFG in Chomsky normal form for $L - \{\lambda\}$. (Note that G always exists and that the right-hand side of every production in P consists of two variables or a terminal symbol only.) Let $G' = (V, \Sigma, S, P')$ be a CFG obtained from G, by reversing the right-hand side of every production in P. That is, for any $w = YZ$ or $w = a$, if P includes $X \to w$, then P' includes $X \to w^R$, where w^R is the reverse of string w.
   Using induction technique, we will show that for every string w of variables and terminals in $(\Sigma \cup V)^+$, if there is a derivation $X \Rightarrow_G w$ in G, then there is a derivation of w^R in G' as well (that is, $X \Rightarrow_{G'} w^R$). This in turn implies that G' generates $(L(G))^R$, and since G' is a CFG, we conclude that $(L(G))^R$ is a CFL.

   Basis case: For any variable $X \in V$, if $X \Rightarrow^0_G X$, then $X \Rightarrow^0_{G'} X^R$, and since $X = X^R$, we have that $X \Rightarrow^0_{G'} X$.
   Induction hypothesis: Assume that $X \Rightarrow^*_{G'} z^R Y u^R$ holds whenever $X \Rightarrow^*_G uYz$. If we apply a production $Y \to v$ in G, where v may consist of two variables or a terminal symbol, we derive $X \Rightarrow^* uvz$ in G, and if we apply the corresponding production $Y \Rightarrow v^R$ in G', we obtain the derivation $X \Rightarrow^* z^R v^R u^R$ in G', noting that $z^R v^R u^R = (uvz)^R$, which was to be shown.
   A final note is that if $\lambda \in L$, then G is modified by adding a fresh start variable S' to V and adding the productions $S' \to S \mid \lambda$ to P'. That is, $G' = (V \cup \{S'\}, \Sigma, S', P' \cup \{S' \to S \mid \lambda\})$. In this case, it is clear that we have the derivation $S \Rightarrow^*_G \lambda$ in G and the corresponding derivation $S' \Rightarrow^*_{G'} \lambda$ in G'.

   (b). (Homomorphism): Let L be any CFL and G be a CFG for $L - \{\lambda\}$. Let $G' = (V, \Sigma, S, P)$ be a CFG in Chomsky normal form such that $L(G') = L(G)$. For every terminal symbol $a \in \Sigma$, if P includes the production $X \to a$, replace the production by $X \to h(a)$. Furthermore, if $\lambda \in L$, add a fresh variable S' to V as the start variable of G' and add the productions $S' \to S \mid \lambda$ to P. This yields the CFG $G" = (V \cup \{S'\}, \Gamma, S', P \cup \{S' \to S \mid \lambda\})$. It is easy to see that $G"$ generates $h(L)$, and since it is a CFG, we conclude that $h(L)$ is a CFL.

2. **Pl. note correction below for L_d.**
   We will show that L_a and L_b are CFL's, L_c is non CF, and L_d is **regular**.

   (a). A CFG for L_a is as follows:
   $S \to AB \mid BA \mid A \mid B$
   $A \to aAa \mid aAb \mid bAa \mid bAb \mid a$
   $B \to aBa \mid aBb \mid bBa \mid bBb \mid b$
   Explanation: It should be clear that L_a includes every string over $\{a, b\}$ of odd length. For even length strings w in L_a, the proposed grammar generates strings of the form $w = w_1 w_2$, where w_1 and w_2 differ at least in the symbol appearing at the i^{th} position, for some $1 \le i \le \min(|w_1|, |w_2|)$.

(b). Below is a CFG for L_b.

$S \to XX$

$X \to 1X1 \,|\, 0$

Explanation: The proposed grammar generates strings of the form $w = 1^m01^m\, 1^n01^n$, which can be viewed as $uu = 1^m01^n\, 1^m01^n$, by "swapping" the two sequences of 1's (1^m and 1^n) which appear between the two 0's in w.

(c). The language L_c is not CF. Suppose it is CF, and since it is infinite, the P.L. applies. Let m be the integer in the P.L. Consider the string $w = a^{m+2}b^{m+1}c^m \in L_c$ whose length $|w| \geq m$. Then, according to the lemma, there is a decomposition of w into sub-strings u, v, x, y, z (that is, $w = uvxyz$) with $|vy| \geq 1$ and $|vxy| \leq m$, such that string $w_i = uv^ixy^iz$ is in L_c, $\forall i \geq 0$. We will examine all such decompositions and for each case we find an i such that w_i is not in L_c.

Case 1: Both v and y consist of one type of symbol σ. There are 3 sub-cases:
Case 1.1. If $\sigma = a$, then $vy = a^k$, for some $1 \leq k \leq m$. Picking $i = 0$, we obtain $w_0 = a^{m+2-k}b^{m+1}c^m$ which is not in L_c, because $k \geq 1$ and hence $n_a(w_0) \leq n_b(w_0)$.
Case 1.2. If $\sigma = b$, then $vy = b^k$, for some $1 \leq k \leq m$. Picking $i = 0$, we obtain the string $w_0 = a^{m+2}b^{m+1-k}c^m$ which is not in L_c, since $n_b(w_0) \leq n_c(w_0)$.
Case 1.3. If $\sigma = c$, then $vy = c^k$, for some $1 \leq k \leq m$. Picking $i = 2$, we obtain the string $w_0 = a^{m+2}b^{m+1}c^{m+k}$ which is not in L_c, since $n_b(w_0) \leq n_c(w_0)$.

Case 2: v and y consist of 2 different symbols: a and b. There are 3 sub-cases:
Case 2.1. $v = a^{k_1}$ and $y = b^{k_2}$, where $1 \leq k_1 + k_2 \leq m$ and $k_1, k_2 \geq 1$. Picking $i = 0$, we obtain $w_0 = a^{m+2-k_1}b^{m+1-k_2}c^m$, which is not in L_c, since $n_b(w_0) \leq n_c(w_0)$.
Case 2.2. $v = a^{k_1}b^{k_2}$ and $y = b^{k_3}$, where $1 \leq k_1 + k_2 + k_3 \leq m$ and $k_1, k_2, k_3 \geq 1$. Picking $i = 0$, we obtain $w_0 = a^{m+2-k_1}b^{m+1-k_2-k_3}c^m$, which is not in L_c, since $n_b(w_0) \leq n_c(w_0)$.
Case 2.3. $v = a^{k_1}$ and $y = a^{k_2}b^{k_3}$, where $1 \leq k_1 + k_2 + k_3 \leq m$ and $k_1, k_2, k_3 \geq 1$. Picking $i = 0$, we obtain $w_0 = a^{m+2-k_1-k_2}b^{m+1-k_3}c^m$, which is not in L_c, since $n_b(w_0) \leq n_c(w_0)$, because $k_3 \geq 1$.

Case 3: v and y consist of 2 different symbols: b and c. Analysis and treatment similar to Case 2 above.

Other two cases of decompositions include (1) v and y consist of the 2 different symbols a and c, and (2) v and y consist of 3 different symbols, neither of which is possible since each requires that $|vxy| > m$.

Since every possible decomposition of w failed, we conclude that L_c is not CF.

(d). L_d is a regular language; a regular expression for this language is a^*. This is because while the first type of strings in this language are of the form a^{n^2} whose lengths are squares, these strings are included in the second type that includes every string over the alphabet $\{a\}$.

3(a). Fig. 1 presents a Turing Machine M_a for L_e. A solution strategy used in the design of M_a is as follows. It considers the left most symbols, replaces it with a blank, and keeps moving to the right, while looking the other two symbols. It marks the first of the two using X, and keeps moving to the right while looking for the third symbol. Once the third symbol is found, it is marked with an X, and the head moves to the left most symbol. It removes all the X's that appear as the leftmost symbols. This process repeats until either all the symbols are removed (in which case M_a halts in state $q8$ and accepts the input) or it can't find an expected matching symbol (in which case M_a will halt in any state other than $q8$).

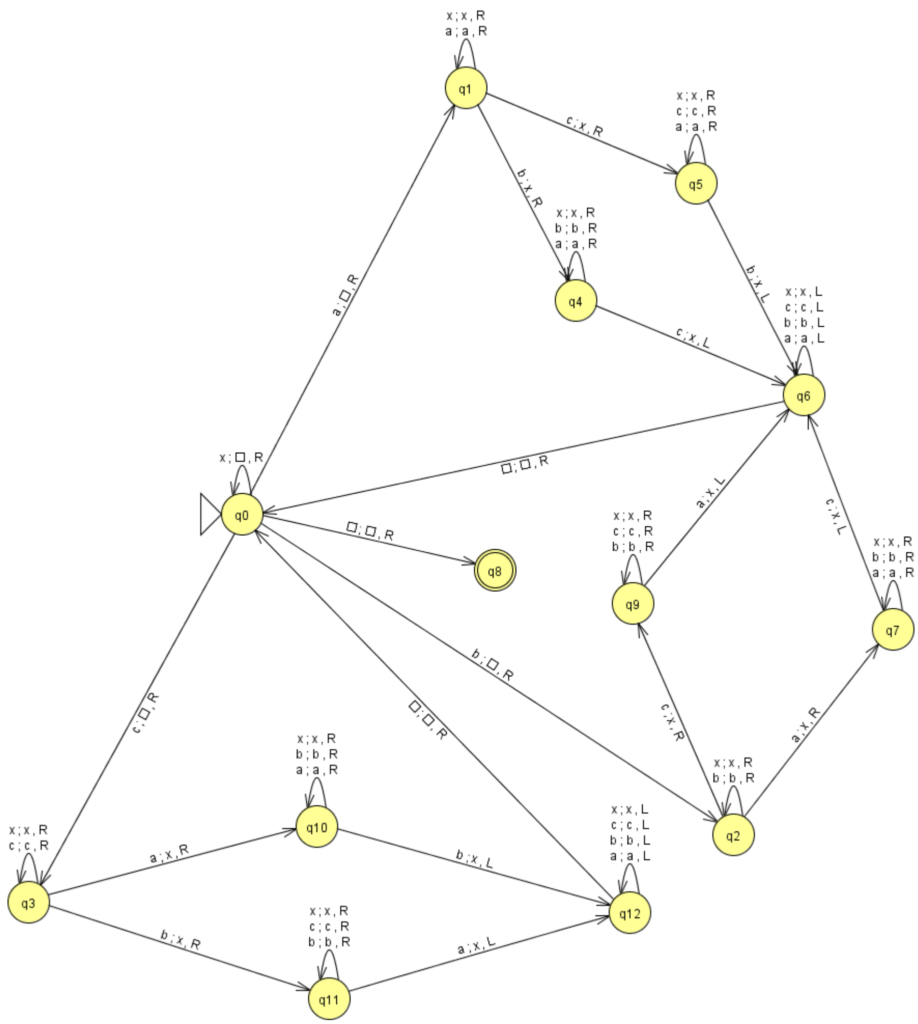

Figure 1: A Turing Machine for L_e in Q3(a).

3(b). We propose a two phase solution for the design of a desired Turing machine M_b, as follows. In phase 1, we find the "midpoint" symbol in the input string $u = ww$ located at position $n+1$, where $|u| = 2n$. When M finds the midpoint, it will be in state $q5$, shown in Figure 2, and the tape head will be under the first symbol in the second w, if the input string u is of even length. If $|u|$ is of odd length, M_b terminates in a non-final state and rejects the input. In phase 2, we match the ith symbol in the first half of input with the corresponding symbol in the second half, until we match them all and M_b halts in a final state, or the matching process fails at some position between 1 to n, at which M_b halts in a non-final state. For the first phase, we repeatedly mark the left most, unmarked symbol in the input string u with its right most, unmarked symbol, using A or B, if the marked symbol is a or b, respectively. We will use X as the marking symbol at each iteration in which the actual symbol a or b is remembered through the path for that symbol. This matching process is repeated in phase 2 for every symbol in the second half and the corresponding symbol in the first half. When the input is of the form ww, then M_b is in the final state $q11$, the first half of the input is replaced with blanks, all the symbols in the second half of input are replaced with X's, and the tape head is under the rightmost X.

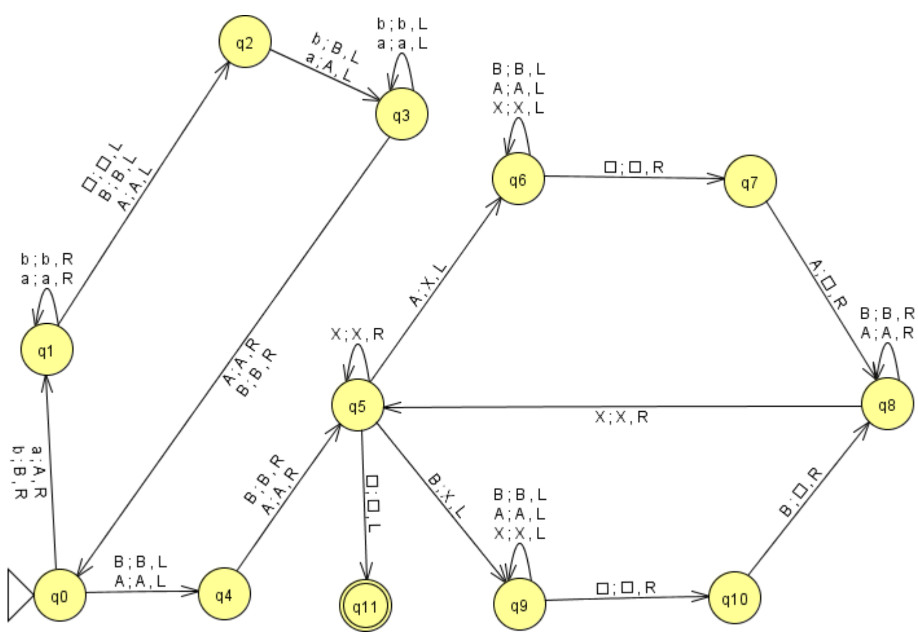

Figure 2: A Turing Machine for L_f in Q3(b).

3(c). A solution method for the design of a TM M_g for L_g is as follows. Repeatedly we remember the left-most symbol, remove it, and try to match it with the rightmost symbol. This remembering is done through two paths, one for symbol a and the other for b. If the input string is an even length palindrome, M_g halts in state q_6 and the input is replaced with blanks. Otherwise , the machine halts in a non-final state, indicating that the input is rejected. Figure 3 shows the transition diagram of M_g.

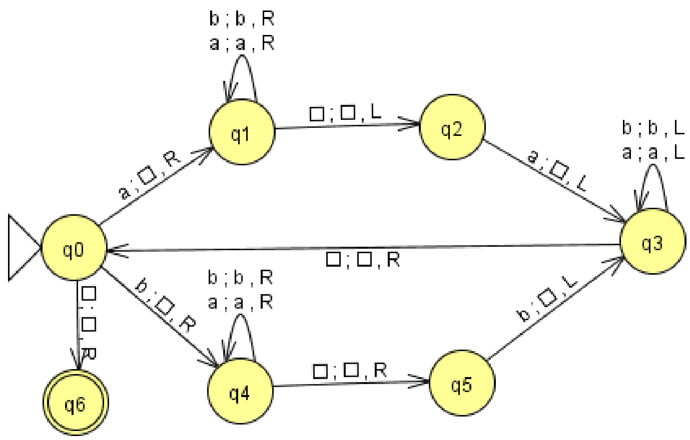

Figure 3: A Turing Machine for L_g in Q3(c).

4. (a). Figure 4 shows a desired TM M_a that computes the function $f(x) = x \ mod\ 5$. The solution idea used here is that M_a uses x to mark 5 ones at each round, if exist, then remove those 5 x's, and repeats the process. If at some iteration, the number of 1's left is less than 5, then the x's are replaced by 1's and halts in state q_{11}. The tape head is under the left most 1 in the output.

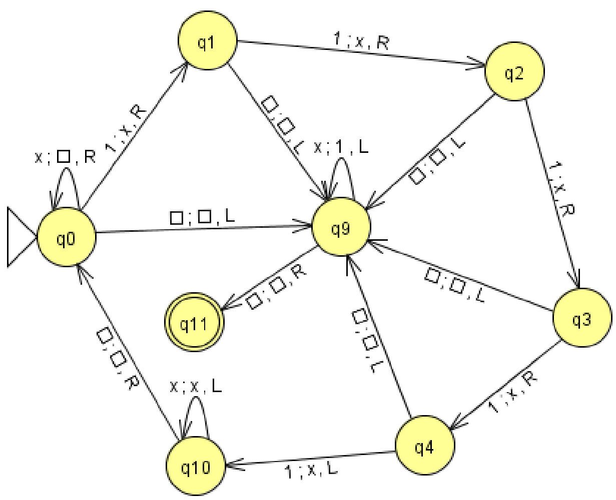

Figure 4: A Turing Machine for the function 4(a).

(b). A design method for this TM is as follows. The tape head moves to the right and replace every pair of 1's with a pair of blanks. If found, a 1 is written at the end of the output (located at the right of the input). When x is odd, the single symbol 1 left in the input will be replaced with blank. The TM terminates in state $q8$ and the r/w head will be under the leftmost 1 in the output. Figure 5 shows the transition diagram of this TM.

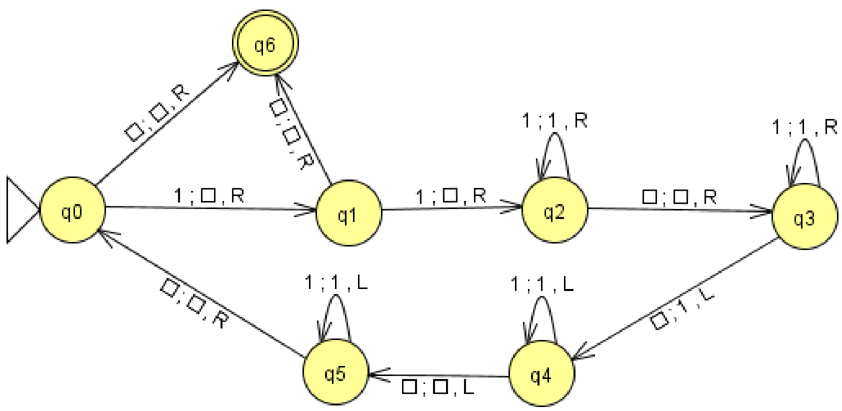

Figure 5: A Turing Machine for the function 4(b).