Formal Languages Assigment 7

Rachel Hwang

December 6, 2013

1. ***Excercise 0.1***

   (a) The transitions for the machine can be described as follows:

State	\$	0	1	B
q_0	$(q_0, \$, R)$	$(q_0, 0, R)$	$(q_0, 1, R)$	(q_1, B, L)
q_1	$(q_2, 1, L)$	$(q_2, 1, L)$	$(q_1, 0, L)$	–
q_2	$(q_f, \$, R)$	$(q_2, 0, L)$	$(q_2, 1, L)$	(q_f, B, R)
q_f	–	–	–	–

   q_0 moves the head to the right end of the input string, transitions to q_1 when it hits a blank.
   q_1 adds one to the string by changing the first 0 (or \$) it encounters to 1 and all intermediate 1s to 0, then transitions to q_2.
   q_2 returns the head to the leftmost position, then transitions to q_f.
   q_f is the accepting state.

   (b) The sequence of ID's for input q_0111 is:
   $q_0\$111 \vdash \$q_0111 \vdash \$1q_011 \vdash \$11q_01 \vdash \$111q_0B \vdash \$11q_11 \vdash \$1q_110 \vdash \$q_1100 \vdash q_1\$000 \vdash q_2B1000 \vdash q_f1000$

2. ***Excercise 0.2***

   (a) We may store information about x, the repeated 100 character string in the state, since we know each unique binary string of 100 characters represents a different number. Since we read from left to right, the binary is read backwards. For example, the string of all 0s will be represented by q_0. The string of 1, then 99 0s will be q_1, and so on. 100 1s will be $q_{2^{100}-1}$.

      i Subroutine 1: we move through the string recording x. For each character we read at position n (indexing from 0), we add the number (symbol $\cdot 2^n$) to our current state number. For example, if we read a 1 in position n = 0 from state q_0, then we transition to state q_1, thus, by the time we've scanned position 99, we scanned 100 characters and store them all.

      ii Subroutine 2: read 100 characters, each character c at some position n, subtracting $(c \cdot 2^n)$. If after scanning the character at n=99, we are in state q_0, then accept.

   Define the machine M for this language as follows: since we don't know where to begin reading x's 100 characters, M calls subroutine 1 from every position in the original input string. This means that at each position M non-deterministically either calls subroutine 1, or moves forward 1 position and calls subroutine 1 (recursive!). From each seperate calls of that subroutine 1, once finished reading 100 characters, we call subroutine 2 from every remaining position in the input string. If any of these sub-calls accept, the string is accepted. If at any point, the head encounters a blank before finished with both subroutines, it fails. After accepting, the head will be pointed just after the second instance of x.

   (b) This machine can be defined using two tapes, one which will hold the input string and one initially blank tape that will hold the current index j in binary.

      i Subroutine 1: Adds 1 to the number on the index tape (using the method in exercise 1) and then nondeterministically either call subroutine 2, or advance to the next '#' symbol, then call subroutine 1.

ii Subroutine 2: Compare the binary string on the index tape with the string on the input tape by sequentially comparing one digit at a time in the same positions, 0 and 0, 1 and 1, etc. If when the head on the index string reaches a blank, the head on the input string reaches a '#', then accept.

Define the machine M for this language as follows: at position 0 in the pnput string, call subroutine 1. This will check if w_i and i match for each i. If there is a match, the machine accepts immediately.

(c) The machine M for this language is very similar to the previous one, except it uses more states. Rather than having only enough states to perform the subrountines aboves, we now have additional states represnting whether we have seen a matching w_j and j yet. We can create two versions of each state in the machine for the previous problem sets 0 and 1. They function the same, but idicate how many matching pairs we've seen. We begin with the same two tapes, one for input, one for storing index.

i Subroutine 1: Adds 1 to the number on the index tape (using the method in exercise 1) and then nondeterministically either call subroutine 2, or advance to the next '#' symbol, then call subroutine 1.

ii Subroutine 2: Compare the binary string on the index tape with the string on the input tape by sequentially comparing one digit at a time in the same positions, 0 and 0, 1 and 1, etc. If when the head on the index string reaches a blank, the head on the input string reaches a '#', then if the machine is in the set 1 of states, then accept. Else, if the machine is in state 0, advance the head on the index tape back to the leftmost position and call subroutine 1.

Once again, it is sufficient for M to call subroutine 1 from the first position in the input string. This machine will accept only if the accept condition for the machine in exercise 2.b is met twice, which the machine keeps track of with state.

3. ***Excercise 0.3***
The language of this NTM is $\{0(0+1)^*\}$, the language of all binary strings that begin with 0.

Since there is no transition from the first input symbol that consumes ay other characters, the input must begin with a 0. From state q_0, the machine may non-deterministically either continue moving right through the string, scanning 0s, or switch to state q_1, in which it scans a 1, writes a 0, moves left and transitions to state q_2. In q_2, it scans a 1, and moves right, transitioning back to q_0. From q_0, if the machine encounters a blank, it transitions to the accepting state. Thus, after the first symbol, the machine can read any sequence of 0s and 1s.

4. ***Exercise 0.4*** Since any ordinary TM can be simulated with a single tape, any TM can be simulated with a k-head machine (all but one of the head are simply redundant and disregarded). So we need only show that any k-head machine can be simulated with an ordinary TM.

We can simulate a k-head machine M with a multi-tape TM M' as follows:
We create a machine M' with $k+1$ tapes. Each of the first k tapes will have the behavior of one of the k heads in M. The final tape will be used to keep the lists synchronized, call this the check tape. To begin, the input string is copied to each of the $k+1$ tapes.

i In the check tape, at each position of string, we check to see if in any of the other k tapes, there is a head at that position. If so, mark the symbol in the check tape (eg. $a \to \hat{a}$). Return the check tape head to the leftmost position.

ii Using the transition table from M, advance each head, writing and moving as directed, each head along its seperate tape.

iii Now advance through the check tape. If the symbol at the current position is marked, this means that it has just been scanned by a head. Now, begining with tape k and ending with tape 1, check that position in each tape to see if the symbol has just been overwritten. If we find that the symbol has been overwritten, copy the change to the check tape and immediately move on the next position in the check tape. Once all the marked symbols have been updated, move the check tape head back to the left-most position.

iv Advance through the check tape again. For each marked symbol, copy that symbol to each other string, writing it in the same postion (ie. if the first symbol in the check tape is marked, copy that symbol to the first position in every other tape. (In order to copy symbols, we will need to move the heads of

the k tapes, but we can store the current position of the k-heads using state. This means that for each state q_n in M, we will have a state $q_n p1p2 \dots pk$, where each p_i represents the position of the head in tape i. After copying, return each tape head to the position indicated in the state. For all other head movements, change state to reflect head movements appropriately.)

This process if repeated until the machine fails or transitions to any accepting $q_f p1p2 \dots pk$. Since as proved in class, any multi-tape TM can be simulated with a single-tape TM, any k-head machine can be simulated with an ordinary TM.