SOLUTIONS HW#6

1. (a) A *linear deterministic bounded automaton* is a 9-uple $M = (Q, \Sigma, G, \lceil, \rceil, \delta, s, t, r)$, where:

- Q is a finite set of states
- Σ is a finite set (input alphabet)
- G is a finite set (tape alphabet) containing Σ
- $\lceil$ is the left endmarker
- $\rceil$ is the right endmarker
- δ is the transition function
- s is the start state
- t is the accept state
- r is the reject state

We restrict δ to satisfy:
for all $p \in Q$ there exists $q \in Q$ such that $\delta(p, \lceil) = (q, \lceil, R)$
for all $p \in Q$ there exists $q \in Q$ such that $\delta(p, \rceil) = (q, \rceil, L)$
once the machine enters the accept state, it never leaves it; similarly for reject state.

(b) The machine M can write on *n* tape cells and on each cell it can write one of *m* symbols while being in one of *k* states. Moreover, the head can be pointing at any one of *n* symbols. The number of configurations it can have is the number of states times the length of the available tape times the number of symbols available to the power of the length of the tape: states*length*symbols$^{\text{length}}$.
Therefore there can be at most $km^n n$ configurations.

(c) Because of this being finite, an LBA running on a Universal Turing Machine (UTM) is decidable. If the LBA accepts, the UTM accepts. If the LBA rejects, the UTM rejects. However, the UTM counts the steps that the LBA is taking, and if the LBA takes more steps than what is the total number of possible configurations, then it has to be looping, and the UTM rejects, thus halting.

(d) Let $M_1, M_2, \ldots, M_i, \ldots$ be an enumeration of all LBA's and let $w_1, w_2, \ldots, w_j, \ldots$ be an enumeration of all strings in Σ^*. Consider the matrix A such that $[A]_{ij} = 1$ if LBA M_i accepts the word w_j and $[A]_{ij} = 0$ if LBA M_i does not accept the word w_j. Now construct the language L in the following way (this is the diagonalization argument). The word w_j is in the language L iff the LBA M_j does not accept w_j.

Claim: The language L is not accepted by any LBA.
Proof of Claim: This is the standard diagonalization argument. Suppose not and let L be accepted by LBA M_i. Then consider the action of M_i on the word w_i. If $w_i \in L$, then by construction of L, the machine M_i rejects w_i. On the other hand, if $w_i \notin L$, then by construction of L, the machine M_i accepts w_i. In either case we reach a contradiction!

Claim: The language L is recursive.
Proof of Claim: Given any string *w* the recursive TM for L finds out the index i of w in the enumeration of all strings and generates the description of the i-th LBA M. Then it runs M on *w* and accepts iff M rejects. Since M halts on all inputs the TM also halts and decides all inputs.

2. We want to reduce the halting problem to the state entry problem. Suppose we have an algorithm (Turing Machine R) that solves the state-entry problem. We will construct an algorithm that solves the halting problem.

Suppose we have an algorithm TM R that accepts if M enters in state *q*, and rejects if M does not enter in state *q*. We want to design an algorithm S that accepts if M halts on *w* and rejects if does not halt on *w*. If we can solve R, then we can solve S.

For this we will modify the input machine M to have only one halting state *q*; to do this, from any halting state we add transitions to *q*. We get that M halts if and only if M' halts on state *q*.

So the algorithm for the halting problem is the following:

1. construct machine M' with state *q* the only halting state
2. run the Turing Machine R (algorithm for state-entry problem) with inputs: M', *q*, *w*
3. If R rejects, reject
4. If R accepts, simulate M' on *w* until it halts
5. If M' has accepted, accept
   If M' has rejected, reject

From this we get that since the halting problem is undecidable, it must be that the state-entry problem is also undecidable.

3. We remember that in the book it is proved that the "totality problem" (accepts any string – page 235) is undecidable. For the equivalence problem we prove that the totality problem is reducible to it. That is, if an algorithm can solve the equivalence problem, it can be used to solve the totality problem. Since no algorithm can solve the totality problem, the equivalence problem must also be unsolvable.

The reduction is as follows: for any Turing Machine M, we can construct a Turing Machine M' that takes any input *w*, runs M on that input, and outputs "yes" if M halts on *w*. We can also construct a TM M'' that takes any input and simply outputs "yes". If an algorithm can tell us whether M' and M'' are equivalent, it can also tell us whether M' halts on all inputs, which would be a solution to the totality problem.

So we proved that also this problem is undecidable.

4. Follow the outline on pages 124-126. The proof is almost exactly the same.

Suppose we start with the string $x_0 \hat{I} G^w$, in which only the first n symbols $x = x_1x_2..x_n$ are not the blank symbol. We are not allowed to overwrite x. We have the start configuration $(s, éx_0, 0)$, so from the definition of the one-step → for Turing machines and from the fact that we cannot change x, we get exactly like in the proof at page 124 that when the algorithm crosses the boundary of x from right to left, it does it in some state q and every time it crosses the boundary from left to right it will come out of x in some state p, because its future action is completely determined by its current configuration. So p depends only on q and x, so we can define an operator $T_x(q)=p$, where $T_x: Q È\{\cdot\} ® Q È\{\wedge\}$, with • and ^ given by: "the first time the head crosses the boundary of x, will do it in some state $T_x(\cdot)$. The machine may never emerge from x, so in this case we call it $T_x(\cdot)=\wedge$".

If $T_x(\cdot)=\wedge$, M must be an infinite loop, so never accepts or rejects. If not, in the future it might move back to x from the right in state q. In that case we will either have $T_x(q)=\wedge$ or we define $T_x(q)=p$.

Note that there are only finitely many possible tables for T, so there is only a finite amount of information about x that can be passed across the boundary. So if $T_x=T_y$ and M accepts xz, then it will also accept yz, because all the movements in the first n symbols are determined by T, which is the same in both cases.

In order to accept xz the machine should enter at some point in the accept state. So this should also happen for yz.

Using Myhill-Nerode theorem and the fact that we only can have a finite number of equivalence classes we get that these machines accept only regular sets.

5. Suppose L is r.e. Then there is a Turing machine M such that M recognizes L. Let $x \in$ L. The witness y will be an accepting run of the TM M on the input x. Note that this can be described finitely: if M accepts x in n steps then we only need to consider the first n tape cells. Note that given a run of the TM M, one can construct a recursive function to check that the run is consistent with the description of the TM, and that the run is an accepting run of M on the input x. The recursive procedure to do this will check the following. Suppose we use some standard encoding of M. The recursive procedure checks that initially the machine M is in its initial state, and the input tape contains the string x followed by blanks. Then we check that for every two consecutive configurations α and β, we can reach β from α in one step using the transition relation for M. Finally check that the TM M is in an accepting state in the last configuration.

Conversely, suppose $A = \{x \mid \$\ y\ R(x,y)\}$ for some recursive procedure R. A TM recognizing L works as follows. On input x, the TM enumerates all possible strings y. For each y it runs the TM for R on the input (x, y). If R accepts, the TM accepts the string x. If R rejects, then the TM generates the next string in the enumeration for y. If there is a

string y such that $(x, y) \in R$, we finally generate that y and accept. On the other hand, if there is no such y, the TM rejects x by looping.