Limits of Computation : HW 1 Solutions and other problems

Rajiv Raman

February 23, 2007

1 Homework 1

1. We need to show that a turing machine with a doubly infinite tape, D is equivalent to an ordinary turing machine, M. In order to show equivalence between D and M, we need to show two things. First that any language L that can be recognized by M can be recognized by D as well, and second that any language L' that can be recognized by D can be recognized by M. We can show this by simulation. i.e., simulate D to behave exactly like M and vice-versa.

   In this case, this first part i.e., simulating M by D is fairly straightforward. Mark the left hand end of the input of D and prevent it from moving it's head to the left of this mark.

   To show the other direction, that we can simulate D with M, we can do one of two things.

   (a) Mark the end of the input of M with a special marker, and treat the tape to the right of this marker as a the tape to the left of the input of D. When D moves to the left of the input, M moves to the right of the marker. When D writes to the right of the input, we shift the contents of M's tape, from the marker onwards to the right end of the tape, one cell to the right.

   (b) M uses two tapes. The first tape represents the input and the part of the tape of D to the right of the input and the second tape represents the part of the tape of D to the left of the input. When D writes to the left of the input, M writes on the second tape and when D writes on the part of the input or to the right of the input, M writes on the first tape. The portion of the tape of D to the left of the input appears

in reverse order on the second tape. Since we know that 2-tape turing machines are equivalent to 1-tape turing machines, we are done.

It is clear in either case that M simulates D.

2. In this problem, we need to show that a turing machine R with only RIGHT and RESET moves is equivalent to an ordinary turing machine M.

   Again, one direction is straight forward. M simulates R as follows. When R moves RIGHT, M does the same. When R does a RESET, M moves it's tape head all the way to the left end of the input.

   The second part, that R simulates M can be shown as follows. When M moves it's head RIGHT, R does the same. When M moves left, R cannot but gets the same effect by doing the following :

   (a) R marks the current head location on the tape

   (b) Resets and copies the entire tape one cell to the right, except for the mark which is kept on the same tape cell.

   (c) R resets again and scans RIGHT until it finds the cell with the mark.

3. The language accepted by the machine are all strings starting with a 0. Consider any input string starting with a 1. The TM can only move from q_0 to q_4, the reject state on encountering a 1. Hence, all computation paths will *reject*. Now consider an input string starting with a 0. On each computation path, the TM does one of the following :

   (a) Converts the 0 to a 1, moves RIGHT and to state q_1, or

   (b) Converts the 0 to a 1, moves RIGHT and stays in state q_0.

   Hence, there is one computation path that stays at q_0 converting the 0's to 1's and moving RIGHT until just before the first 1 is encountered, when it moves to state q_1. On q_1, the machine can convert the 1 to 0, move the head LEFT and go to state q_2. On q_2, since the current cell contains a 1, moves to q_0 and moves RIGHT. Here, the 1 was converted to 0 and has the effect of a string starting with a 0. This gives us a method to convert the 1 to a 0 if it is not (unless the string starts with a 1) and move to the start state, in effect dealing with a smaller string starting with a 0. Now at the end of the input, the control moves to state q_1 where, upon seeing a B, moves to the accept state. Thus, the languages accepted by the TM are all strings starting with a 0.

4. This problem again asks to prove equivalence between a *2-dimensional* turing machine, N and an ordinary TM, M. So we need to show two things : How N can simulate M and M can simulate N.

   The first part again is straightforward. N just doesn't use the UP and DOWN moves thus simulating a single dimensional turing machine with a doubly infinite tape. This has been shown equivalent to M in Problem 1, and hence we are done.

   To show the second part, one approach is the following. Place a special *end of row* marker "#" at both ends of the input. This corresponds to the input row. All other cells are empty. We show how we simulate each of the moves of N using M.

   (a) **RIGHT/LEFT :** When N moves RIGHT by one cell, M does as well, unless it reaches an *end of row marker* "#". In this case, we expand each *row* of M by one cell to the right, namely mark the current tape position, rewind to the start of the tape, and move RIGHT. When an end of row marker is encountered, move all tape contents one cell to the RIGHT. Now we can rewind again and go back to the old head position, and move one cell to the right. The LEFT move is similar.

   (b) **UP/DOWN :** In order to simulate an UP move of N, we need to move left beyond the *end of row marker* to the the *row above*. However, we also need to ensure we are at the same column. This can be done by marking all the tape contents from the current head position to the end of row marker. Then, counting off the columns by unmarking the contents of the old row and marking the cell of the new row.i.e., the head moves LEFT from the current position, marking each cell until it reaches the *end of row* marker #. Upon encountering the #, we know we have moved to the row above. Now, we move RIGHT, unmark the rightmost marked cell, traverse LEFT till we find the first unmarked cell *after* the # and mark this cell. This way, we count the columns so at the end when there are no more marked cells to thr right of #, we have reached the corresponding column of the row above. Now we can unmark all the cells till the current cell and we have in effect executed an UP move. The DOWN move is similar.

   We also need to take care that if we end up beyond the last row, we add a new row by inserting an end of row marker. Thus M simulates N.

5. This is again a simulation question, so we want to show that a turing machine M can simulate a queue automaton Q and vice-versa. We will show the first

part using a 2-tape turing machine M. This is sufficient since we know that 2-tape TMs are equivalent to 1-tape TMs. To show the first part, M maintains the input on the first tape, and uses the second tape as the queue. When Q moves it's input tape head to the right, M does the same. When Q does a *push*, M writes the symbol on the right end of the second tape. When Q does a *pull*, M removes the left most element from the second tape and moves all symbols on the second tape one space to the left. Thus M simulates Q.

To show Q simulates M, we augment M's alphabet with additional symbols $\tilde{a}$ for each symbol a of M's alphabet. This constitutes Q's alphabet. The symbols $\tilde{a}$ will be used to maintain the position of M's tape head. Q also uses a special end-of-tape marker #, which is placed at the end of the input. At the start of the simulation Q pushes the entire input into the queue, with the first symbol a replaced by $\tilde{a}$.

When M moves RIGHT, Q pulls each symbol from the queue and pushes them back into the queue, except for the marked alphabet. This is replaced with the new symbol written by M and the next symbol pulled, say a is replaced by $\tilde{a}$ and pushed into the queue.

Moving LEFT is trickier, since the symbol has already been pushed into the queue. We can simulate a LEFT move by maintaining a history of one more symbol in the control of Q. i.e., Q pulls two symbols at a time from the queue and pushes them, swapping the marker on the symbol if necessary. This can be seen to execute a left move. For example, say the tape contents are $c, \tilde{b}, a$. The configuration of M after the move is say, $c, x, \tilde{a}$. Thus, we pull both a and $\tilde{b}$ from the queue, swap the marker from b to a, replace b with x and push $\tilde{a}$ and b in order into the queue in order.

2 Problem Session 1

1. (Problem 3.15) Show that the collection of decidable languages is closed under

   (a) **Union:** (in the textbook).

   (b) **Concatenation:** Let K, L be decidable languages. The concatenation of languages K and L is the language $KL = \{xy | x \in K \text{ and } y \in L\}$. Since K and L are decidable languages, it follows that there exist turing machines M_K and M_L that decide the languages K and L respectively. In order to prove that KL is decidable, we can construct a turing machine that decides KL.

This machine, M_{KL} can use the machines M_K and M_L to decide if a string is in KL or not. The machine can be constructed as follows : Consider an input string w. We need to decide if w is of the form xy for $x \in K$ and $y \in L$. If this is the case, there must be a position at which we can partition w into x and y. Since there are only finitely many ways to partition the string, we can try all possibilities and accept if there is such a partition and reject otherwise. We will describe a non-deterministic turing machine since it is easier to describe.

i. On input w, non-deterministically partition w into strings xy.
ii. Input x to M_K and y to y on M_L.
iii. *accept* if both M_K and M_L accept, else *reject*.

If there is an accepting computation path, then we have found a successful split and the string is in KL. If all computation paths reject, then the string is not in KL. In either case, it is easy to see that the machine M_{KL} halts. Thus, KL is decidable.

(c) **Star:** For a language L, $L^* = \{x \in L \cup LL \cup LLL \cup \cdots\}$. i.e. all strings obtained by concatenating L with itself, and so on. To show that L^* is decidable, the idea is similar to the previous solution. We want to find cuts of the input string w, such that each of them is accepted by the TM M_L that decides L. Let M_{L^*} be the machine that that decides L^*.

i. On input w : For each way to cut w into parts $w_1 w_2 \cdots w_n$
ii. Run M_L on w_i for $i = 1, \cdots, n$.
iii. If M_L accepts each of the strings w_i *accept*.
iv. If all cuts have been tried without success, *reject*.

(d) **Complementation :** This is fairly straightforward, but the point to note is that *turing recognizable languages* are **NOT** closed under complementation, while *turing decidable* languages are. For a language L, let M_L denote the turing machine deciding L. Then the turing machine for the complement is $M_{L'}$ which on input w, accepts if M_L rejects, and accepts otherwise.

(e) **Intersection :** This is again fairly simple. Let K and L be two turing decidable languages, and let M_K and M_L denote the turing machines deciding K and L respectively. Let $M_{K \cap L}$ denote the turing machine deciding $K \cap L$. $M_{K \cap L}$ works as follows.

i. On input w to $M_{K \cap L}$,
ii. Input w to M_K.
iii. If M_K rejects, *reject*.

iv. Else Input w to M_L.

v. If M_L accepts, *accept*. Else *reject*.

2. Show that the collection of turing recognizable languages is closed under the following operations.

   (a) **Union :** (in the textbook).

   (b) **Concatenation :** Let K and L be two turing recognizable languages, and let M_K and M_L denote the turing machines that recognize K and L respectively. We construct a non-deterministic turing machine M_{KL} that recognizes the language KL.

      i. Non-deterministically cut input w into w_1 and w_2

      ii. Run M_K on w_1. If it halts and rejects, *reject*.

      iii. Run M_L on w_2. If it accepts, *accept*. If it halts and rejects, *reject*.

      Note the difference between the turing machines for recognizable and decidable languages. Here, we need to take care of the fact that the machines M_K and M_L need not halt.

   (c) **Star :** For a turing recognizable language L, we construct a non-deterministic turing machine M_{L^*} that recognizes L^*. The idea is similar to the decidable case.

      i. On input w, non-deterministically cut w into parts $w_1 w_2 \cdots w_n$.

      ii. Run M_L on w_i for all i. If M_L accepts all of them, *accept*. If M_L halts and rejects for any i, *reject*.

      If there is a way to cut w into strings $w_1 w_2 \cdots w_n$ such that each $w_i \in L$, then there is a computation path in M_{L^*} that accepts w in a finite number of steps.

   (d) **Intersection :** Let K and L be two turing recognizable languages, and let $M_K, M_L, M_{K\cap L}$ denote the turing machines recognizing $K, L, K \cap L$ respectively. We use M_K and M_L to construct $M_{K\cap L}$. The machine $M_{K\cap L}$ works as follows.

      i. On input w, run M_K on w. If it halts and rejects, *reject*. If it accepts, goto step ii.

      ii. Run M_L on w. If it halts and rejects, *reject*. If it accepts, *accept*.

      $M_{K\cap L}$ accepts a string w only if both M_K and M_L accept, thus w belongs to $K \cap L$.

3. To answer question 3.17, we require a few definitions and the solution to Problem 3.18. So we start with the definitions.

Definition 1 *An enumerator is a turing machine with an attached* printer*, or output tape and a work tape. This machine does not accept any input, but uses the work tape and outputs strings on the output tape. The strings produced by an enumerator E is the language* enumerated by E*.*

We will use the following theorem in the textbook (Page 153, Theorem 3.21).

Theorem 2 *A language is turing-recognizable if and only if some enumerator enumerates it.*

We can now extend this theorem to show the following theorem (Ex. 3.18).

Theorem 3 *A language is turing-decidable if and only if some enumerator enumerates the strings of this language in lexicographic order.*

Proof: In order to prove this theorem, recall the proof technique in showing how different turing machine models are equivalent. Just like in the earlier cases, we need to show equivalence between a turing machine that decides a language and an enumerator that enumerates it. Thus we need show the proof in both directions.

Assume we have a turing machine to decide a language L. We can use this TM to construct an enumerator E as follows. We generate strings in lexicographic order, and input each string into the TM for L. If the TM accepts, print the string. Then, E prints all strings of L in lexicographic order.

Now we need to show the other direction. i.e., if we have an enumerator E for a language L, then we can use E to construct a turing machine that decides L. We consider two cases.

(a) If L is a finite language, it is decidable because all finite languages are decidable.

(b) If L is infinite, a decider for L operates as follows. On receiving input w, the decider enumerates all strings of L in lexicographic order until a string greater than w in the lexicographic order appears. This must eventually occur since L is infinite. If w has appeared in the enumeration already, then accept. If w has not appeared yet, then it will never appear, and hence we can reject.

□

Now, we can solve problem 3.17. Let E be an enumerator for B. We construct an enumerator D which outputs the strings of C in lexicographic order. The decidability of C follows from the previous theorem. Enumerator D simulates E. When E outputs the i^{th} TM $< M_i >$, enumerator D pads M_i by adding sufficiently many extra useless states to M_i to obtain a new TM M_i' where the length of $< M_i' >$ is greater than the length of $< M_{i-1}' >$. Then E outputs $< M_i' >$.

4. (Problem 4.17) We need to prove both directions. To handle the easier one first, assume that D exists. A TM recognizing C operates on input x by going through each possible string y and testing whether $< x, y > \in D$. If such a y is ever found, accept, else just continue searching.

   For the other direction, assume that C is recognized by TM M. Define a language B to be $\{(x, y) | M$ accepts x within $|y|$ steps $\}$. Language B is decidable, and if $x \in C$ then M accepts x within some number of steps, so $< x, y > \in B$ for sufficiently long y, but if $x \notin C$, then $< x, y > \notin C$ for any y.

5. (Problem 4.18) Let A and B be two languages such that $A \cap B = \emptyset$, and $\overline{A}$, and $\overline{B}$ are recognizable. Let J be the TM recognizing $\overline{A}$ and K be the TM recognizing $\overline{B}$. We will show that the language decided by TM T separates A and B. The TM T works as follows.

   (a) On input w :

   (b) Simulate J and K on w by alternating the steps of the two machines.

   (c) If J accepts first, reject. If K accepts first, accept.

   The algorithm T terminates because $\overline{A} \cap \overline{B} = \Sigma^*$. So either J or K will accept w eventually. $A \subseteq C$ because if $w \in A$, w will not be recognized by J and will be accepted by K first. $B \subseteq \overline{C}$ because if $w \in B$, w will not be recognized by K and will be accepted by J first. Therefore, C separates A and B.