Assignment 1 Solutions

March 22, 2002

1. [1.4.15]

   (a)
   ```
   (1) for i = 1 to n
   (2)   for each subset S of size i excluding 1
   (3)     for each j in S
             if i = 1 then c[S,j] = d(1,j)
             else
               temp = infinity
   (4)         for each k in S\{j}
                 temp = min{temp, c[S\{j}, k] + d(j,k)}
               c[S,j] = temp
   ```

   (b) Loops (1) and (2) ensure that all subsets of $S\backslash\{1\}$ are looped through. There are $O(2^n)$ of these. Loops (3) and (4) can require as many as $O(n)$ steps. Thus the total running time is $O(n^2 2^n)$, as desired.

   For each subset S, $|S|$ space is required. Thus the total space used is $O(n2^n)$.

   (c) We will let $c[i, j]$ represent the least–weight path of length i from 1 to j. We will use the following algorithm:

   ```
   for i = 1 to n
     for j = 2 to n
       if i = 1 then c[i,j] = d(1,j)
       else
         temp = infinity
         for k = 2 to n
           temp = min{temp, c[i-1, k] + d(k,j)}
         c[i,j] = temp
   ```

   Then the desired path weight is simply the minimum of all $c[i, n]$. This algorithm clearly runs in polynomial time.

2. (a) [Base–10]

We will have two states, "remainder" and "no remainder". We start in state "no remainder". Scan through the input from left to right. For each digit d, overwrite with $d/2$ if in state "no remainder", and with $(d+10)/2$ if in state "remainder". If d was even, move to state "no remainder" and move to the right on the tape. Otherwise, if d was odd, move to state "remainder" and move right on the tape. Terminate when the end of the input is reached (note that, if the input was odd, the final remainder will be discarded, as desired)

Linear time is used.

(b) [Base–2]

Move to the last (far right) input symbol, and overwrite with a blank.

Linear time is used.

(c) [Base–1]

The easiest way is to use a machine with one in put and one output tape. Simply read through the input, and, for every two input "1"s read, output a single "1".

In this case, linear time and constant space are used.

To solve the problem using a single–tape machine, make one pass through the input, overwriting the 1st, 3rd, 5th, etc input "1" with a "#". Then rewind back to the beginning of the tape, and "delete" each "#". (See question 3 for a formal definition of "delete").

This takes quadratic time, since each "delete" operation takes linear time.

3. [2.8.8]

(a) A deletion, along with the transition from state q to q', maps configurations to configurations as follows:

$$(q, wa, u) \mapsto (q', w, u)$$

An insertion of a, along with the transition from state q to q', maps configurations to configurations as follows:

$$(q, w, u) \mapsto (q', wa, u)$$

(b) Assume that $f(n)$ is larger than the length of the input tape. If not, then we can simply ignore the last $n - f(n)$ symbols on the input tape, since they are never read anyway, and we can "pretend" that $n = f(n)$. The length of the tape can never longer than $f(n) + n$, and we can assume that this is in $O(f(n))$.

An insertion of a into the configuration (q, w, u) can be simulated by shifting all the character of u one position to the right, and then writing a into the blank which is created between w and u. This takes time $O(|u|) \leq O(f(n))$.

A deletion applied to the configuration (q, wa, u) can be simulated by shifting the characters of u one position to the left, overwriting a. Again, the time taken is $O(|u|) \leq O(f(n))$.

So, each step takes at most $O(f(n))$ time, so the total amount of time taken is $f(n) \times f(n) = (f(n))^2$.

4. [2.8.17] Suppose you have a k–string nondeterministic TM M. The idea here is to compress the k tapes into one k–track tape. This will be the first tape, which will be used to simulate M. At each step of the computation, the k tape heads may all be at different positions, but we only have one tape head. So, we will keep our tape head "synchronized" with tape 1 of M, and nondeterministically guess what is under all the other tape heads. Furthermore, we will store each nondeterministic guess on the second tape.

   So, after $f(n)$ steps, there will have been $kf(n)$ nondeterministic guesses, all stored on the second tape. If our simulation ends by rejecting the input, we reject. Otherwise, we must check to make sure that none of our nondeterministic guesses were incorrect (for we cannot accept an input based on an incorrect guess, or we may end up accepting a word which is not in the language). So, one tape at a time, we simulate M again, this time checking that the guesses stored on the second tape are all correct. If we find an erroneous guess, we reject. Otherwise, if all the guesses are correct, we accept. Clearly if a word is in the language then there will be one accepting path (the path which contains all the correct nondeterministic guesses). The time taken is $O(f(n))$ (since we require k passes through the second tape of length $O(kf(n))$, and k is a constant. This completes the proof.

5. [3.4.1 (a) (c) (e) (f) (g)]

   - (a) Not recursive: Given $M; x$, construct M' such that for all y,

$M'(y) = M(x)$. Then M' halts on input ϵ if and only if M halts on input x.

Recursively enumerable: Given a TM M, we can simulate it on input ϵ and answer **yes** if and only if M halts.

- (c) Not recursive: Given $M; x$, construct M' such that on any input y, M' simulates $M(x)$ (assume without loss of generality that σ is not in the tape alphabet for M). If the simulation of $M(x)$ halts, then M' should write σ and halt. So M' writes σ if and only if M hlats on input x.

  Recursively enumerable: For a TM M we want to answer **yes** if there is some input x for which M writes a Σ at some step n. The set of all possible choices for x is countable. The set of all possible choices for n is countable. Thus, the cross product of these two sets is countable. So, a TM test all possible ordered pairs (x, n) to see if M outputs σ at step n on input x. If so, answer **yes**.

- (e) Not recursive: Given $M; x$, construct a TM M' which, on any input y, simulates $M(x)$ answers **yes** when and if the simulation halts. So $L(M')$ is non-empty if and only if M halts on input x. This shows that the complement of our problem is non-recursive. Since recursive languages are closed under complementation, this is sufficient.

  Not recursively enumerable: Once again, we will examine the complement of our problem. We will show that the complement ("non–emptiness") is recursively enumerable, and so the oringinal problem ("emptiness") must not be r.e. (since if a language and its complement are both recursively enumerable, then it is recursive)

  To see that "non-emptiness" is recursively enumerable, we will use essentially the same argument as was used in part (c). Instead of checking each ordered pair (x, n) to see if M writes σ at step n, we will check each pair to see if M accepts at step n.

- (f) Not recursive: Using the same reduction as in part (e).

  Not recursively enumerable:

- (g) Not recursive: Given $M; x$, let M_1 be a TM that, on any input, simulates $M(x)$ and answers **yes** if the simulation halts. Let M_2 be the TM that answers **yes** on all inputs. So $L(M_1) = L(M_2)$ if and only if M halts on input x.

Not recursively enumerable:

6. [3.4.3] The solution to this problem can be found in many places, for example,

   `http://www.cs.wisc.edu/~condon/cs810/pcp.ps`

7. [3.4.8] Use g as defined in the question. Then:

$$\begin{aligned} g(M_{f(g)})(x) &= M_{f(g)}(M_{f(g)})(x) \\ &= f(g((M_{f(g)}))(x) \end{aligned}$$

   and so $g(M_{f(g)})$ is a fixed point of f, as desired.

8. [2.8.4]

   (a) For any TM M, let M_U represent the encoding of M as discussed in Chapter 3.1 of the text, that is, the encoding used as input to the UTM.

   Let μ_0 represent the following encoding of TMs: Suppose f is a function which maps encodings of TMs to encodings of TMs, and M is the encoding of a TM. Then the bit string $f_U; M$ is in μ_0 and represents $f(M)$, that is, $U(f_U; M)$.

   Now, let μ' be another encoding of TMs. There must be a TM F which maps encodings of TMs M μ' to equivalent encodings M_U. So, let M be a TM, with encoding M' in μ' Then $M(\epsilon) = U(F(M'); \epsilon) = U(U(F_U; M'); \epsilon)$, and so $F_U; M'$ is the desired encoding in μ_0 and the result holds, with $c = |F_U| + 1$.

   (b) Each x is trivially outputted by an $|x|$–state TM. Thus, if we use the universal exconding μ_U as discussed in Chapter 3.1 of the text, $K_{\mu_U}(x) \leq |x| + c$, and so $K(x) \leq |x|$, by the result of part (a).

   (c) For each n, there are 2^n strings of length n but only $2^n - 1$ encodings of length less than n. Thus, at least one encoding must have length longer at least n.