

Homework 4: Solutions

Due: March 10, 2004

Vinod Vaikuntanathan

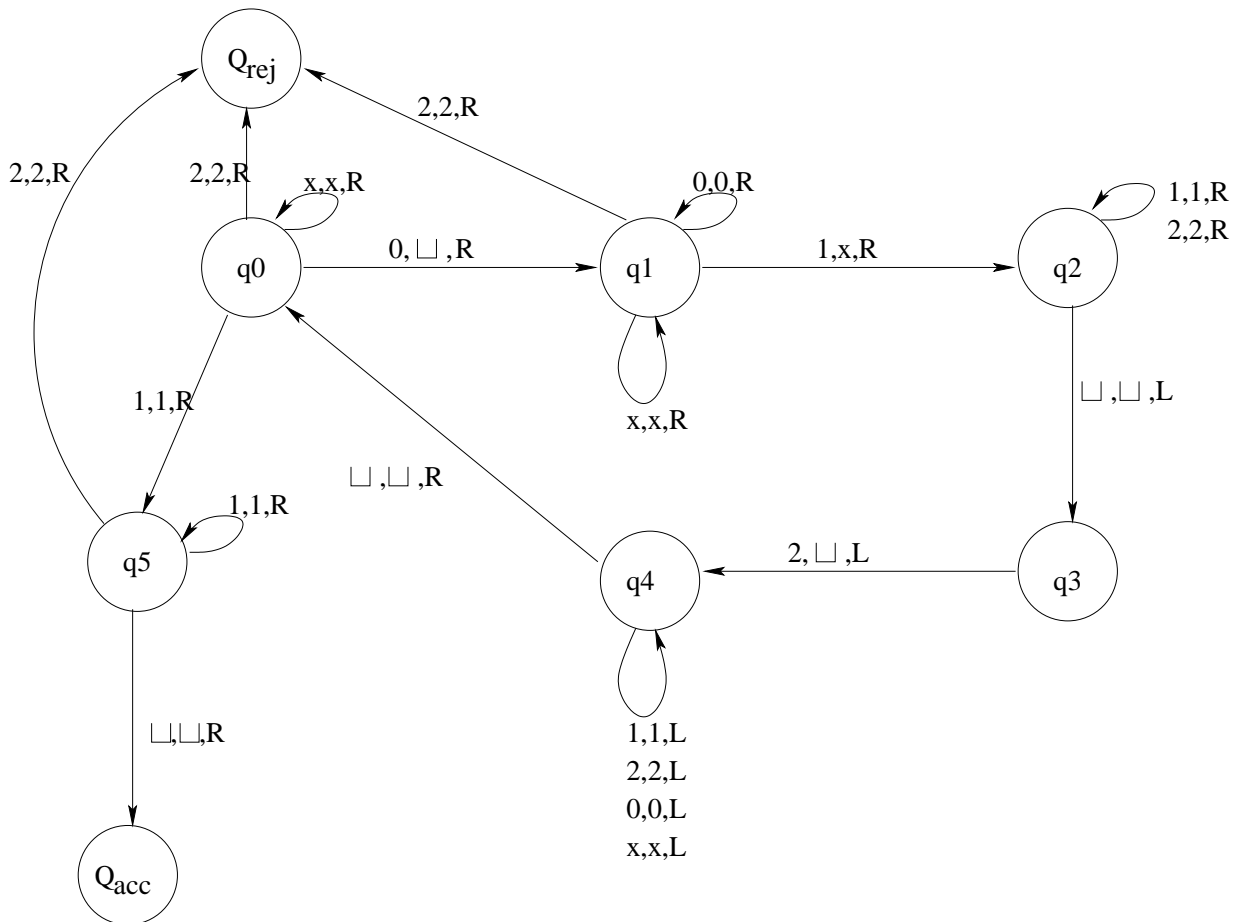
Problem 1: Detailed Turing machine description

Give a complete, formal description of a (basic one-head, one-tape) Turing machine that *decides* the language $L = \{0^i 1^j 2^i : 0 \leq i < j\}$. This should consist of a list of the tuple components of the Turing machine, with the transition function represented by a state transition diagram.

(Yes, we know that it's tedious to write Turing machine descriptions. But we think that everyone should do it once. We'll try to avoid asking for this on other homeworks.)

Solution 1: Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ be the Turing machine that decides L . Here $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject}\}$, $\Sigma = \{0, 1, 2\}$, $\Gamma = \{0, 1, 2, x, \sqcup\}$, and the other tuple members are described by the diagram below. (Omitted transitions go to q_{reject} .)

Notation : A transition labeled $(0, 1, R)$ means that the machine on reading 0 writes a 1 in the tape square and moves right.

**Problem 2: Implementation-Level (also known as “higher-level” Turing machine**

description)

Describe the operation of a basic Turing machine that *recognizes* the language

$$L = \{w : w \text{ does not contain twice as many 0s as 1s}\}$$

This time, your description should not be completely formal. Rather, it should consist of a list of the tuple components, with the transitions described in words. (This is what Sipser calls an “implementation description” on p. 157.)

You may describe your construction in terms of a variant of the basic Turing machine model presented in class or in Sipser’s book (e.g., multitape), provided that you quote the correct transformation result to show how one would turn your construction into a description of a basic Turing machine to recognize the same language.

Solution 2: Let T be a deterministic Turing machine (with a single tape and a single head) that behaves as follows:

T = “On input x ,

1. Start from the left end of the tape and scan for a 1.
2. (*Find a 1*) If a 1 is found, rewrite it with a special symbol x and return to the left end of the tape. If no 1 is found go to Step 4.
3. (*We have found a 1. Now find two 0s and mark them off*) Scan for two 0s. If two zeros are found, rewrite them with x and go to step 1. If not, go to the accept state.
4. (*The TM reaches Step 4 when it cannot find a 1.*) Scan for a zero. If a zero is found, go to the accept state, else go to the reject state.

Note, this is only one of *many* possible implementations.

Problem 3: Robustness of the Turing Machine model

Consider a Turing machine model that uses a 2-dimensional tape, corresponding to the upper right quadrant of the plane. The head of such a Turing machine could move to the right, left, up or down. Sketch a proof that such a model does not add extra computing power; that is, the class of languages recognized by such Turing machines is the same as the class recognized by basic Turing machines.

Solution 3: *The key idea is to notice that at any point of time, the TM has looked at only a finite region of the two-dimensional tape.*

Say the input of the machine is in the first row of the tape, initially the tape head is at the square $(0, 0)$ and all the other tape squares are blank.

Simulate the 2D tape with a 1D tape as follows:

1. The rows of the 2D tape starting from the “bottom-most” row are written in the 1D tape next to each other, as in the figure below. Note that this is indeed possible, because at any point of time, the TM has looked at only a finite number of rows, and a finite number of tape squares in each row.
2. When the machine tries to move right from a square in the 2D tape, we can simulate it in the 1D tape trivially, except when it tries to move right from the rightmost square in a row. In that case, we “make room” for the new tape square by moving all the symbols to the right. For instance, when the machine tries to move right after reading x_7 in the figure, we simulate the effect in the 1D tape as follows: We move all the symbols starting from the \$ right after x_7 in the 1D tape, one square to the right. In the newly vacant square, we write a blank, and continue simulating.

- *Intersection*: Run the machines M_1 and M_2 in parallel. If both of them accepts, we accept. Otherwise, we either reject or loop.
 - *Concatenation*: There are $n + 1$ ways of breaking a string x of length n into strings y and z such that $x = y \cdot z$. For each of them, test $y \in L_1$ and $z \in L_2$. If $y \in L_1$ and $z \in L_2$ for at least one such choice of y and z , we accept. Else, we reject. *The important thing to note is that the $n + 1$ executions of the machines, for $n + 1$ different choices of y and z have to be done in parallel.*
 - *Star (L^*)*: Again, there are 2^{n-1} ways of splitting a string x into substrings x_1, \dots, x_ℓ such that $x = x_1 \cdot \dots \cdot x_\ell$. For each of these ways, test if all of x_1 to x_ℓ are in L . If this is the case for at least one choice of x_1, \dots, x_ℓ (for some ℓ), then accept. Else, reject. *Again, it is important to run the 2^{n-1} executions in parallel, because we do not want to get stuck on a single looping execution.*
2. The corresponding constructions for recursive languages (Turing-decidable languages) are similar, except that now, we do not have to run the executions in parallel. This is so, since the machines are guaranteed to halt.

Problem 5:

1. Let L be a set of natural numbers that can be enumerated by an enumerator Turing machine in nondecreasing order, possibly with repeats. Prove that L is a decidable set of numbers.

Solution 5: Suppose there exists such an enumerator E . We have two cases.

- *L is finite*: If L is finite, it is decidable, trivially. (It is even a regular language !!)
- *L is infinite*: To decide if a given $x \in \mathbb{N}$ is in L , run the enumerator E . If x is in the language, the enumerator prints x at some point of time, and we accept. Suppose x is not in the language. Then, since L is infinite, there is a string $y > x$ which is in L . The enumerator will print y at some finite point of time. If we do not see x in the output of the enumerator, but see a $y > x$, then we reject.