CSCI3390-First Test Solutions

February 25, 2016

In this test you never have to give a low-level description of a Turing machine complete with states and transitions (like the kind given in Figure 1). In 1(e) and 3 you are asked to give *high-level* descriptions: these are written in language like, 'the machine scans its input left-to-right, replacing all occurrences of a by X and then writes aaa at the right end of the tape'.

In Problem 2, you should give the answers as informal descriptions of algorithms; you do not need to talk about Turing machines at all.

While Problem 4 refers to the construction in Problem 3, you can do Problem 4 even if you could not carry out the construction.

1 A Turing Machine

(30 points, 10 points for parts (b) and (e) and 5 points for each remaining part) The Turing machine $\mathcal{M}$ pictured in the figure recognizes the language $L \subseteq \{a, b\}^*$ consisting of all the strings in which the number of a's is *less than or equal to* the number of b's.

(a) Let Q be the set of the states of the machine and Γ the tape alphabet. Let

$$\delta : (Q - \{\text{accept}, \text{reject}\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

be the next-state function. What are the values of $\delta(1, X)$ and $\delta(1, a)$?

Solution.

$$\delta(1, X) = (1, X, L).$$

$$\delta(1, a) = (5, X, L).$$

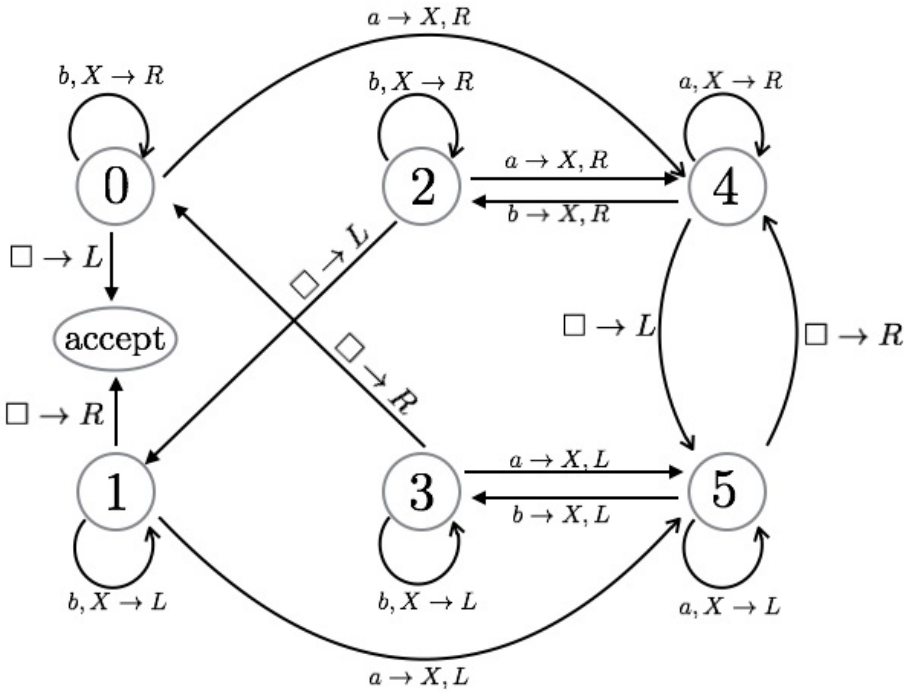

Figure 1: The Turing machine for Problem 1. The initial state is 0.

(b) Show the run of the machine (the complete configurations) on the inputs bab and a for ten steps, or until the machine halts, whichever comes first.

$$\mathbf{0}bab$$
$$b\mathbf{0}ab$$
$$bX\mathbf{4}b$$
$$bXX\mathbf{2}\square$$
$$bX\mathbf{1}X$$
$$b\mathbf{1}XX$$
$$\mathbf{1}bXX$$
$$\mathbf{1}\square bXX$$
accept

$$\mathbf{0}a$$
$$X\mathbf{4}\square$$
$$\mathbf{5}X$$
$$\mathbf{5}\square X$$
$$\mathbf{4}X$$
$$X\mathbf{4}\square$$

Note that we have repeated the configuration $X\mathbf{4}\square$, so the machine will now cycle endlessly.

(c) Does $\mathcal{M}$ *decide* the language L? Explain *briefly* (a sentence will do).

Solution. No, because it loops endlessly on some inputs (like a).

(d) Is the language L decidable? Explain *briefly.*

Solution. Yes, because obviously there is a pencil-and-paper algorithm for determining whether or not a string has more a's than b's! If you want something more detailed, you can alter the transitions in states 4 and 5 so that after the machine makes a complete scan while looking for a b to cross out, and finds no b, the machine rejects.

(e) On inputs of the form $a^n b^n$, $\mathcal{M}$ takes roughly $4n^2$ steps to accept. Describe a *two-tape* Turing machine that recognizes the same language L, and takes time *linear* in the length of the input string to accept strings in L.

Solution. Scan the first tape left to right. Each time you see an a on the first tape, move right and write an a on the second tape. Each time you see a b on the first tape, move left and erase the last a on the second tape. If there are at least as many b's as a's, when the end of the input is encountered on the first tape, the head of the second tape will be looking at a blank, because all the a's will have been cancelled by b's. Otherwise if there are more a's than b's, the head of the second tape will be looking at an a. This requires a single scan of the input on the first tape. Each step may require two or three transitions on the second tape, but all in all, no more than $3n$ transitions are executed on an input of length n. (Other solutions to this problem are possible. For instance, you might scan the first tape left to right and copy a's to the second tape, and then scan it right to left, canceling a's with b's. Or you might mark the starting cell on the second tape, move right for each a on the first tape, move left for each b, and record in the state whether you are to the left or right of the mark.)

2 A word game

(14 points, 7 for each part) Here is a game played with bit strings. If the string begins with 1, remove the 1 and append 00 to the right end of the string. If it begins with 00, remove the 00 and append 10. If it begins with 01, remove the 01 and append 11. Thus if you run this game for several steps, starting with 1, you get

$$1 \to 00 \to 10 \to 000 \to 010 \to 011 \to 111 \to 1100 \to \cdots$$

Let L be the set of all bit strings that can be derived from 1 in this manner. (So, for example, every string in the example above is in L.)

(a) Show that L is Turing-recognizable (that is, describe an algorithm that semi-decides L).

(b) Show that L is actually decidable (describe an algorithm that decides it).

Solution. The semi-deciding algorithm for (a) is just to keep applying the rewriting rule as shown in the example. If you see your input w appear, then $w \in L$ and you answer 'Yes'. (Unlike some of the homework problems, this does not require you to be clever about how you organize the work.) What about the deciding algorithm? When do we get to say No? Observe that when we apply the rewriting rule, the string never gets *shorter*, although it could stay the same length. So to test whether $w \in L$, we apply the same algorithm, but if we start to see strings longer than w, we can say No, because we can never get back to w. There is one other situation in which we might be able to say No–and that is if the strings stay the same length forever. But since there are only finitely many strings of a given length, if this happens, we will see a string repeat. So the complete algorithm for deciding L is:

Repeatedly apply the rewriting rule.

If you see w, answer Yes.

If you see a string longer than w, answer No.

If you see as string that has already appeared, answer No.

Anyway, that's my solution. I leafed through the student papers at the end of the class and saw a cleverer method. The idea is that you can always determine from the last two symbols of v what the preceding word was—that is, you can run the algorithm *backwards*. Thus if the string ends with 00, remove 00 and prepend 1, if it ends with 10, remove 10 and prepend 00, and if it ends with 11, remove 11 and prepend 01. Do this repeatedly until either (a) you reach 1 (answer Yes); (b) you reach a different string from which you can't make a legal move—that is, anything ending in 01 (answer No); (c) a string repeats (answer No).

3 A reduction...

(10 points) Let $\mathcal{M}$ be a Turing machine and $w \in \{0,1\}^*$. Describe how to construct a new Turing machine $\mathcal{M}'$ such that $\mathcal{M}'$ accepts w and *no other bit string*

if $\mathcal{M}$ accepts w, and $\mathcal{M}'$ accepts *no bit string at all* if $\mathcal{M}$ does not accept w.

Solution. Here is what $\mathcal{M}'$ does: It first scans its input to see if the input is equal to w, and if it is not, $\mathcal{M}'$ rejects its input. If the input is equal to w, $\mathcal{M}'$ returns to the start of the tape and runs identically to $\mathcal{M}$. Thus if $\mathcal{M}$ accepts w, $\mathcal{M}'$ will accept w, but reject every other string. If $\mathcal{M}$ does not accept w, then $\mathcal{M}'$ accepts no string at all.

4 ...and its consequences

(16 points, 2 for each correct answer) Now consider the following four decision problems, described as languages.

$$L_{TM} = \{< \mathcal{M}, w >: \mathcal{M} \text{ accepts } w\}.$$

$$EMPTY = \{< \mathcal{M} >: \mathcal{M} \text{ recognizes the empty language }\}.$$

$$NONEMPTY = \{< \mathcal{M} >: \mathcal{M} \text{accepts some bit string }\}.$$

$$UNIQUE = \{< \mathcal{M} >: \mathcal{M} \text{ accepts exactly one bit string}\}.$$

Tell whether the following statements are true or false. You may use the fact proved in class that L_{TM} is Turing-recognizable but not Turing-decidable.

(a) The construction in Problem 3 is a reduction of L_{TM} to $EMPTY$. **False.** Reductions have to take Yes instances to Yes instances, but the reduction described in Problem 3 takes a Yes instance of L_{TM} ($\mathcal{M}$ accepts w) to a No instance of $EMPTY$ ($\mathcal{M}'$ accepts some string).

(b) ...a reduction of L_{TM} to $NONEMPTY$. **True.**

(c) ...a reduction of L_{TM} to $UNIQUE$. **True.**

(d) ...a reduction of $NONEMPTY$ to L_{TM}. **False.** This is the wrong direction.

(e) This reduction proves that $EMPTY$ is undecidable. **True.** We reduced the undecidable problem L_{TM} to $NONEMPTY$, which proves $NONEMPTY$ is undecidable, but the complements of decidable problems are also decidable, so undecidability of $EMPTY$ is the same as undecidability of $NONEMPTY$.

(f) This reduction proves that L_{TM} is undecidable. **False** While L_{TM} is undecidable, the reduction here does not show it.

(g) This reduction proves that $UNIQUE$ is undecidable. **True.**

(h) This reduction proves that $EMPTY$ is not Turing-recognizable. **True,** but this is rather subtle. The construction in 3 is a reduction from the complement of L_{TM} (which is not Turing-recognizable) to $EMPTY$. If $EMPTY$ were Turing-recognizable, we could use this reduction to get an algorithm that semi-decides L_{TM}, which would make L_{TM} Turing-recognizable, a contradiction.

Incidentally, *all* of the assertions about decidability and recognizability in (e)-(h) are true, but you are being asked if the reduction in 3 proves the assertion.