CSCI3390-First Test, with solutions

October 11, 2018

Note that the only problems in which you deal with the low-level details of a Turing machine are 1(a,b,c). Problems where you have to describe the operation of a TM (1(e) and 3) should be handled using a high-level description. Problems 2(c,d) are probably more difficult than the others. As with all tests, I suggest you read over all the problems at the outset, and first work the ones that you are reasonably sure you know how to do.

Point values: 1-25, 2-25, 3-15, 4-20. Total: 85

1 A Turing Machine

The first figure attached to the exam shows the state-transition diagram of a Turing machine $\mathcal{M}$. The input alphabet of $\mathcal{M}$ is $\{0, 1\}$.

(a) What is the tape alphabet of $\mathcal{M}$?

Solution.

$$\{0, 1, X, \#, \Box\}.$$

(b) Let δ denote the transition function

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

of $\mathcal{M}$. What is $\delta(2, 0)$? What is $\delta(2, \#)$?

Solution.

$$\delta(2, 0) = (3, X, R), \delta(2, \#) = (4, \Box, L).$$

(c) What are the two configurations immediately following

$$1X\mathbf{3}\#$$

(where the boldface number represents a state)?

Solution.

$$1X\#\mathbf{3}\Box$$

$$1X\mathbf{1}\#0$$

(You can see something very much like this in the transitions from steps 12 to 14 in the printed run.)

(d) Attached to the exam is a complete run of the machine on the input 010. You should be able to see from this that the machine takes an input $w \in \{0,1\}^*$ and produces the output 0^k, where k is the number of occurrences of 0 in w. You should also be able to see *how* the machine carries out this computation. About how many steps does this computation require in the worst case on inputs of length n? I am not looking for an exact formula here, just a precise asymptotic estimate.

Solution. The machine marks a $\#$ at the right end of the input, and returns to the left end. Then it repeatedly scans right, crossing off the leftmost occurrence of 0 before the $\#$, continues right to the right end of the tape, and writes a 0. It continues until there are no more 0's to the left of the $\#$, then does one last pass, erasing the $\#$ and everything to its left.

So the total number of passes over the whole tape is $2k+3$, where k is the number of 0's. The three additional passes are for the setup at the beginning and the cleanup at the end. The tape always contains between n and $2n$ symbols. We get the worst-case behavior when the input is entirely 0's, so that $k = n$. (Ironically, because in this case the desired output is present on the tape from the start!) The number S of steps is thus bounded below and above by

$$2n^2 + 3n \leq S(n) \leq 4n^2 + 6n.$$

So the answer is $O(n^2)$, actually $\Theta(n^2)$ in the worst case. (You didn't have to go quite this much detail, but you had to write something by way of explanation.)

(e) Describe a two-tape Turing machine that performs the same computation in time proportional to the length of the input in all cases. The machine should begin with the input on the first tape and end with the output also on the first tape—it doesn't matter what's left on the second tape. This should be a 'high-level' description—you don't have to write out details of states and transitions.

Solution. The machine makes a single pass to the right on both tapes, copying every 0 on the first tape to the second, and erasing every symbol from the first tape. It makes a pass to the left, copying the 0's on the second tape to the first tape. For inputs of length n, this takes about $2n$ steps.

2 Decision problems about graphs

Below is a list of decision problems in which part of the input is a directed graph. If the graph is finite, then we simply represent the vertices of the graph by the integers $1, \ldots, n$ and encode the graph by giving a list of the pairs (i, j) for which there is an arrow from i to j.

If the graph is infinite, then we cannot specify it in this way. Instead, we will represent the vertices by positive integers, and supply as part of the input an algorithm (*e.g.,* a Turing machine or a Python program) that decides, given positive integer inputs i, j, whether there is an arrow from i to j. (For instance, in the infinite directed graph pictured on the last page, the algorithm is: Yes if $j = i + 2$, or $j = i - 1$ and j is odd, or $j = i - 3$ and j is even; no otherwise.)

For each of the problems listed below, tell whether the problem is Turing-recognizable, or decidable, or neither. Briefly justify your answer. You can cite any theorem discussed in class as part of the justification.

(a) **Input:** A finite graph G, and two vertices i and j. **Output:** Yes, if there is a directed path in G from i to j, no otherwise.

Solution. If a path exists, its length is less than or equal to the number n of vertices in the graph. So we can just list all sequences of n vertices, beginning with i and ending with j, and checking if any of them is a path. Thus this problem is decidable.

(b) **Input:** An infinite graph, and a finite sequence

$$i_1, i_2, \ldots, i_k$$

of positive integers. **Output:** Yes if this sequence forms a directed path in G from i_1 to i_k, no otherwise.

Solution. For each successive pair i_r, i_{r+1} of vertices in the sequence, we query the supplied algorithm to find out if this is an arrow in the graph. If every such pair is an arrow, then the sequence is a path.

(c) **Input:** An infinite graph G, and two vertices i and j. **Output:** Yes if there is a directed path in G from i to j, no otherwise. Explain why this problem is Turing-recognizable. (HINT: The preceding problem might help.)

Solution. We can effectively enumerate all finite sequences of vertices, by first listing all the sequences that sum to 2, to 3, etc. that is,

$$1, 1$$

$$1, 1, 1$$

$$1, 2$$

$$2, 1$$

For each of these we can check, using the algorithm from the previous part, whether the sequence is a path, and of course we can check whether the first term of the sequence is i and the last j. Thus, if there is a path from i to j, this algorithm will find one and answer 'Yes'. Note that it never answers 'No', but runs forever if no path exists. Thus this problem is Turing-recognizable.

(d) **Input:** ...and explain why it is *undecidable.*

Solution. In brief, if we could do this, we could decide the halting problem: Let us take a Turing machine $\mathcal{M}$ and an input string w. We can encode every possible configuration of the Turing machine as a positive integer. We then construct a graph where the vertices are configurations, and where there is an arrow from c to c' if and only if configuration c' immediately follows configuration c. Although this graph is infinite, we certainly have an algorithm for determining if one configuration of $\mathcal{M}$ immediately follows another. We can also add one more vertex, call it vertex 1 (we have to design our encoding so that no configuration of $\mathcal{M}$ has this value), and add an arrow from every configuration in the halted state to 1. The halting problem is not determining if there is a path in the graph from the initial configuration to vertex 1. (You could also describe a reduction from the string-rewriting problem, which we showed in class is undecidable).

3 A reduction...

Let $\mathcal{M}$ be a Turing machine with input alphabet $\{0, 1\}$, and let $w \in \{0, 1\}^*$. Describe how to construct a Turing machine $\mathcal{N}$ with the following behavior: If $\mathcal{M}$ halts when started on w, then $\mathcal{N}$ accepts every string over $\{0, 1\}$. If $\mathcal{M}$ does not halt when started on w, then $\mathcal{N}$ accepts the empty string ϵ and no other string.

Solution. Construct $\mathcal{N}$ so that it first checks if its input is empty (that is, if its head is positioned over a blank symbol), and accepts immediately if that is so. If the input is not empty, $\mathcal{N}$ erases its input, writes w on its tape, and then proceeds to act like $\mathcal{M}$. If it ever enters the halt state of $\mathcal{M}$, then $\mathcal{N}$ accepts. This just entails fixing up the description of $\mathcal{M}$ so that there is an initial phase of checking, erasing, and writing w, and adjusting so that the generic halt state is replaced by the accept state.

4 ...and its consequences

Consider the following four problems:

$$HALT_1 = \{< \mathcal{M}, w >: \mathcal{M} \text{ halts when started on } w\}$$

$$FINITE = \{< \mathcal{M} >: \text{The set of strings accepted by } \mathcal{M} \text{ is finite}\}.$$

$$INFINITE = \{< \mathcal{M} >: \text{The set of strings accepted by } \mathcal{M} \text{ is infinite}\}.$$

$$ALL = \{< \mathcal{M} >: \mathcal{M} \text{ accepts every string over } \{0, 1\}\}.$$

(a) The construction described in the preceding problem is a mapping reduction from one or more of the problems above to another one or more of the problems above. Which problems are these? Give all that apply.

(b) As a result we are able to conclude, using results we proved in class, that one or more of the problems above is undecidable. Which problem or problems are these? (Explain a bit of the reasoning as well.)

Solution. The construction has the following properties:

- $\mathcal{M}$ halts on w if and only if $\mathcal{N}$ accepts an infinite set of strings.
- $\mathcal{M}$ halts on w if and only if $\mathcal{N}$ accepts every string over $\{0, 1\}$.

Thus this construction is a mapping reduction from $HALT_1$ to $INFINITE$ and also from $HALT_1$ to ALL. Since $HALT_1$ was known to be undecidable, these two problems are as well.

It also shows that $FINITE$ is undecidable, because if we could decide $FINITE$ then we could decide $INFINITE$. (In this sense, you could also say that the construction reduces $HALT_1$ to $FINITE$, but it is not a mapping reduction.)

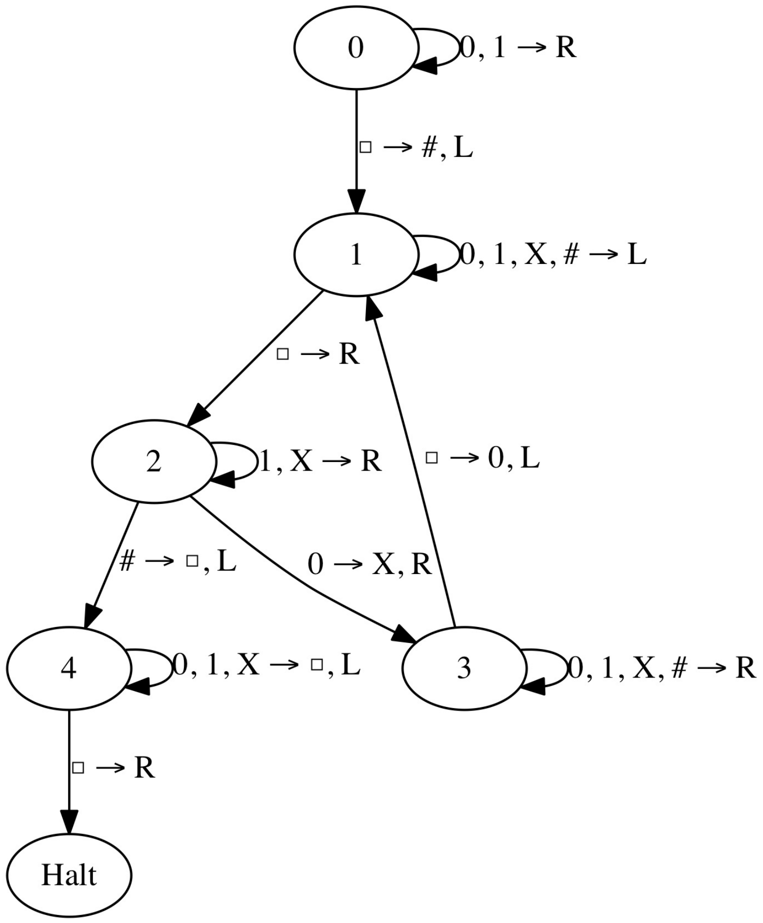

Figure 1: The state graph for the Turing machine in Problem 1

```
1 .state: 0
010
^
2 .
state: 0
010
 ^
3 .
state: 0
010
  ^
4 .
state: 0
010B
   ^
5 .
state: 1
010#
  ^
6 .
state: 1
010#
 ^
7 .
state: 1
010#
^
8 .
state: 1
B010#
^
9 .
state: 2
010#
^
10 .
state: 3
X10#
 ^
11 .
state: 3
X10#
  ^
12 .
state: 3
X10#
   ^
13 .
state: 3
X10#B
    ^
14 .
state: 1
X10#0
   ^
15 .
state: 1
X10#0
  ^
16 .
state: 1
X10#0
 ^
17 .
state: 1
X10#0
^
18 .
state: 1
BX10#0
^
19 .
state: 2
X10#0
^
20 .
state: 2
X10#0
 ^
21 .
state: 2
X10#0
  ^
22 .
state: 3
X1X#0
   ^
23 .
state: 3
X1X#0
    ^
24 .
state: 3
X1X#0B
     ^
25 .
state: 1
X1X#00
    ^
26 .
state: 1
X1X#00
   ^
27 .
state: 1
X1X#00
  ^
28 .
state: 1
X1X#00
 ^
29 .
state: 1
X1X#00
^
30 .
state: 1
BX1X#00
^
31 .
state: 2
X1X#00
^
32 .
state: 2
X1X#00
 ^
33 .
state: 2
X1X#00
  ^
34 .
state: 2
X1X#00
   ^
35 .
state: 4
X1XB00
  ^
36 .
state: 4
X1BB00
 ^
37 .
state: 4
XBBB00
^
38 .
state: 4
BBBBB00
^
39 .
state: -3
BBBB00
^
halt
39 steps
    00
```

Figure 2: Run of the Turing machine from Problem 1 on the input string 010.

$$\begin{tikzcd}
1 \arrow[r] & 3 \arrow[r] \arrow[dl] & 5 \arrow[r] \arrow[dl] & \cdots \\
2 \arrow[r] & 4 \arrow[r] \arrow[ul] & 6 \arrow[r] \arrow[ul] & \cdots
\end{tikzcd}$$

Figure 3: An example of an infinite graph. The graph is ‘given’ as input as long as we specify an algorithm for determining when there is an arrow from one vertex to another. Here the algorithm should be obvious!