

Solutions for Problem Set 4
cs172

Allison Coates

Homework problems: 3.5, 2d tape, 3.13, 3.16

3.5 Examine the formal definition of a Turing machine to answer the following questions, and explain your reasoning.

- a. Can a Turing machine ever write the blank symbol $_$ on its tape?
Sure, it can write the symbol if that symbol is in the tape alphabet Γ . As defined in the book, the tape alphabet contains the blank symbol.
- b. Can the tape alphabet Γ be the same as the input alphabet Σ ?
No, the tape alphabet is defined to contain the blank symbol, while the input alphabet is defined to not contain it.
- c. Can a Turing machine's head *ever* be the same location in two successive steps?
As stated on page 128 of Sipser's book, if a Turing machine's head is on the leftmost tape position and attempts to move to the left, the head stays in the same place for that move, even though the transition function indicates L .
- d. Can a Turing machine contain just a single state?
No, part 7 of defn 3.1 says that there must be a reject state and an accept state; they can't be the same state.

2 dim Tape

The 2-dim tape Turing machine is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where $Q, \Sigma, \Gamma, q_0, q_{\text{accept}}, q_{\text{reject}}$ are defined as in the normal Turing machine definition and $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, U, D\}$.

Assume for simplicity that the input is still written on only one line of the tape, and further, assume that there is a leftmost boundary to the tape, and a topmost boundary to the tape. That is, assume that there is a corner to the input tape, and that the tape extends to infinity to the right and down.

A 2-dim tape Turing machine has a different configuration than a standard Turing machine.

Consider the following tape diagram:

	w_1	w_2	w_3
u_1	A	B	v_1
		q_i	
u_2	C	<u>D</u>	v_2
u_3	E	F	v_3
	x_1	x_2	x_3

This diagram indicates the current configuration of the Turing machine. The underline of the D indicates that the head is above the tape square in which the D is written. The q_i indicates that the current of the Turing machine's finite control is q_i .

We now show the possible transitions functions, and the corresponding changes to the configuration. If $\delta(q_i, D) = (q_j, N, L)$, then the Turing machine, on reading a D in state q_i moves to state q_j , writes an N , and moves the head to the Left. This yields a configuration

	w_1	w_2	w_3
u_1	A	B	v_1
		q_i	
u_2	<u>C</u>	N	v_2
u_3	E	F	v_3
	x_1	x_2	x_3

If $\delta(q_i, D) = (q_j, N, R)$, then the new configuration is

$$\begin{array}{cccc} & w_1 & w_2 & w_3 \\ u_1 & A & B & v_1 \\ & & & q_j \\ u_2 & C & N & v_2 \\ u_3 & E & F & v_3 \end{array}$$

If $\delta(q_i, D) = (q_j, N, U)$, then the new configuration is

$$\begin{array}{cccc} & w_1 & w_2 & w_3 \\ & & q_j & \\ u_1 & A & \underline{B} & v_1 \\ u_2 & C & N & v_2 \\ u_3 & E & F & v_3 \end{array}$$

If $\delta(q_i, D) = (q_j, N, D)$, then the new configuration is

$$\begin{array}{cccc} u_2 & C & N & v_2 \\ & & q_j & \\ u_3 & E & \underline{F} & v_3 \\ & x_1 & x_2 & x_3 \end{array}$$

There are other special cases to include, for example, if the machine tries to move off of the left or top end of the tape, its head stays in the same position.

Now, to prove that a 2-dim Turing machine is equivalent to a standard Turing machine.

First we prove that if N is a 2-dim Turing machine, then it is as powerful as a standard Turing machine M . That is, we prove that a 2-dim Turing machine can simulate a standard Turing machine. Using the above assumption that the input is written on the topmost row of the input, the 2-dim tape can easily simulate M since movements to the left and right are already allowed. N simply reads the input and performs the same movements that M would perform. Therefore, N is at least as powerful as M .

Now, we need to show that M can simulate the action of N .

One way to solve this problem is to figure out just how to map the set of infinite cells on the 2dim tape onto the infinite one dimensional tape. Another way to phrase this is: how to you make room on an infinite tape for an infinite number of infinite strings? Or, another way to ask this is: how do you map the infinite 2-d grid to the Counting numbers?

There's a very nice way to do this, actually. Here's a picture:

1	2	4	7	...
3	5	...		
6	...			
...				

now, let's invent a more useful coordinate system so that we can map all of these cells into a line. Consider the (i, j) th entry of the 2-d grid to refer to the i th element from the top of the j th diagonal. For example:

1,1	1,2	1,3	1,4	1,5
2,2	2,3	2,4	2,5	
3,3	3,4	3,5		
4,4	4,5			
5,5				

Now we have a one dimensional ordering of the 2-d tape. So now, we can write use this ordering on the 1-dimensional tape to simulate the tape cells of the 2-d tape.

Given this linear coordinate system, we can determine that moving the head left moves the head from position (i, j) to $(i, j - 1)$; moving the head right moves the head from position (i, j) to $(i, j + 1)$; moving the head up moves the head from position (i, j) to $(i - 1, j - 1)$; moving the head down moves the head from position (i, j) to $(i + 1, j + 1)$.

Given all of these pieces, we can now show how to simulate the 2-d tape Turing machine N with the one dimensional tape Turing machine M . We set up the 1 tape turing machine M to write all of the contents of N 's tape cells into a linear string on M 's tape. That is, M 's tape cells reference N 's in this order:

Given the transition function δ for N , we write the transition function δ' for M that simulates the transition function of N by using the coordinate system above. That is, if N had its head in the (i, j) position then moved to the left, M now moves from the cell representing (i, j) to the position representing $(i, j - 1)$. So M can now perform all of the operations that N can perform. Hence, M is just as powerful as N . Hence, the 2d tape Turing machine is equivalent to the normal 1d tape Turing machine.

- 3.13 Turing machine with stay-put instead of left is similar to an ordinary Turing machine except that the transition function has the form

$$\delta Q \times \Gamma \Rightarrow Q \times \Gamma \times \{R, S\}.$$

At each point the machine can move its head right or let it stay in the same position. Show that this Turing machine variant is *not* equivalent to the usual version. What class of languages do these machines recognize?

At any given step, this machine can read its current input, change its state, write a symbol, and then stay put or move to the right. However, given that the machine can not move left, once it moves right, it cannot go back and reread that input.

Pad the tape alphabet to ensure that $\Sigma \subset \Gamma$. At most, the machine has a total of $|\Gamma| \cdot |Q|$ states that it can be without moving to the right. After this many steps of computation, the machine will be in a configuration that it was in previously. When the machine does move to the right, it will again have $|\Sigma| \cdot |Q|$ states that it can be in without overwriting the remaining input. It will be unable to access more than $|\Sigma| \cdot |Q|$ states without returning to a configuration that it has seen previously. As a

result, the machine has at most $|\Sigma| \cdot |Q|$ configurations that it can reach without repetition and without overwriting the input. Therefore, this machine is a finite automaton with $|\Sigma| \cdot |Q|$ states, and therefore, the class of languages recognizable by this machine is the class of regular languages.

- 3.16 Show that a language is decidable (recursive) iff some enumerator enumerates the language in lexicographic order.

First, we prove that if some enumerator enumerates a language in lexicographic order, then that language is decidable.

Assume we have an enumerator E which outputs strings for a language $L(E)$ in lexicographic order. We design a Turing machine M that decides $L(E)$ as follows:

$M =$ “ On input w ,

- (a) Run E , until E reaches the point that w should appear on the tape. This point will be reached eventually, since E prints out strings in lexicographic order. (In particular, all strings of length $|w|$ will be printed after all strings of length $|w| - 1$ and before strings of length $|w| + 1$. There are $|\Sigma|^{|w|}$ strings of length $|w|$.)
- (b) Check that some string of length $|w|$ that is output by E matches w . At most, this requires checking through $|\Sigma|^{|w|}$ strings. If some string matches w , then accept. If no string of length $|w|$ matches w , then no string output by the enumerator will match. In that case, reject.

M accepts or rejects on all inputs; M accepts all strings in $L(E)$ and rejects all strings not in $L(E)$. Hence, M is a decider for $L(E)$.

Now we prove that if a language L is decidable, then there is some enumerator which enumerates L in lexicographic order.

If L is decidable, then there is some Turing machine M which decides L — that is, it accepts all strings in L , and rejects all strings not in L . We will use M to build a lexicographic enumerator E for L .

We build a Turing enumerator E as follows:

- (a) Generate strings as follows: for $i = 1, 2, 3, \dots$,
 - i. Generate all strings over the input alphabet of Σ of length i in lexicographic order. For example, given the input alphabet $\{0, 1\}$, the strings generated are

1
00
01
10
11
000
...

- ii. Feed each string to the machine M in order. If M accepts, print out that string. If M rejects, move on to testing the next string. Since M halts on all inputs, we can be assured that all strings will be tested without looping.

Practice Problems: 3.6, 3.7, 3.10, 3.14

I am including the solutions to the practice problems because I have received many questions about these problems in office hours. If after reading these answers, you still are unsure of some aspect of these questions, please come see me.

- 3.6 What is wrong with the following proof to prove that if some language L is Turing recognizable, then some enumerator enumerates L ?

$E =$ “ Ignore the input.

- (a) Repeat the following for $i = 1, 2, 3, \dots$
- (b) Run M on s_i .
- (c) If it accepts, print out s_i ."

The problem with the proof is that M on s_i might loop forever. If it loops forever, then E doesn't print out s_i . But more, E isn't going to move on to test the next string. Therefore, it won't be able to enumerate any other strings in L . For this reason, we need to simulate M on each of the strings for a fixed length of time so that no looping can occur.

3.7 What is wrong with the following description of a Turing machine?

$M_{bad} =$ " The input is a polynomial over variables x_1, \dots, x_k .

- (a) Try all possible settings of x_1, \dots, x_k to integer values.
- (b) Evaluate p on all of these settings.
- (c) If any of these settings evaluates to 0, *accept*; otherwise, *reject*."

What does it mean to "try all possible settings of x_1, \dots, x_k to integer values? Consider a polynomial of just two variables: x_0 and x_1 . Consider setting $x_0 = 0$, and then running over all possible values of x_1 . We will never be able to increment x_0 above 0. In this way, we can never try all possible settings, and therefore, we can never evaluate all possible settings of p . So, this description of a Turing machine is too vague— it does not describe a systematic method for enumerating all possible integers. (Is it clear that you can run over all possible settings at all?)

3.10

3.14