CS/ENGRI 172, Fall 2002
10/2/02: Homework Three

I could be bounded in a nutshell and count myself a king of infinite space. — Hamlet

Due at the beginning of class on Wednesday, October 9. Hand in the parts separately, with
your name on each part. Answers will be graded on both correctness and clarity. If a question asks
for explanation, then no credit will be assigned for answers without explanation. In general, you
should always include an explanation of your answers when appropriate (among other things, this
helps in assigning partial credit).

Part A

Recall that in lecture we began constructing a Turing machine that implements an algorithm
for computing the function \( f \), where the domain \( D \) of \( f \) is any finite sequence of A’s (with at least
one A required), and

\[
f(\underbrace{AA\ldots A}_n) = \underbrace{AA\ldots A}_{2n},\quad n \geq 1.
\]

Importantly, we translated intuitive actions to a state-based framework. Here is part of a control
table extending our work from class:

<table>
<thead>
<tr>
<th></th>
<th>⊥</th>
<th>⊥</th>
<th>A</th>
<th>M</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>find-b</td>
<td>saw in class</td>
<td>saw in class</td>
<td>saw in class</td>
<td>saw in class</td>
<td>—</td>
</tr>
<tr>
<td>find-A</td>
<td>(fix, ⊥, R)</td>
<td>—</td>
<td>(move-A, M, R)</td>
<td>?</td>
<td>??</td>
</tr>
<tr>
<td>fix</td>
<td>—</td>
<td>??</td>
<td>(fix, A, R)</td>
<td>(fix, A, R)</td>
<td></td>
</tr>
</tbody>
</table>

Note that the symbol alphabet doesn’t have to be restricted to symbols only appearing in sequences
in \( D \): a Turing machine can have “personal” symbols for scratch work. We assume the initial state
specified in class, and of course the symbol alphabet consists of ⊥, ⊥, A, M, and C.

1. Suppose we set the control table cells labeled ?, ??, and ??? all to —. Call the resulting Turing
machine \( M_1 \).

(a) This subproblem asks you to hand-simulate \( M_1 \) when given two A’s as input. What we would
like you to do is to draw not only the initial and final configuration of \( M_1 \) (we guarantee that \( M_1 
\)
does in fact halt given two A’s as input), but for each time that \( M_1 \) changes to a state different
than the one it was previously in, draw the new configuration the machine is in just after it has
made the move that accomplished the state change. Make sure in your configuration drawings to
include the left-most blank of the infinite sequence of blanks stretching to the right. To help make
this clear, we’ve done the first state change for you:

```
| ⊥ | A | A | ⊥ | ⊥ | ... |
```

(b) What function on the domain \( D \) is \( M_1 \) computing, if any?\(^1\) Explain your answer. Make sure
to specify what happens to any input \( d \) from \( D \).

\(^1\) An example (but incorrect) answer would be, \( f'(\underbrace{A\ldots A}_n) = A \).
2. Suppose instead we set \( ? = (\text{find-A, M, L}) \), \( ?? = (\text{find-A, C, L}) \), and \( ??? = (\text{fix, } \bot, L) \), yielding \( \vdash \bot \). 

Call the resulting Turing machine \( M_2 \). What function on the domain \( D \) is \( M_2 \) computing, if any? Explain your answer. Make sure to specify what happens to any input \( d \) from \( D \).

Part B

3. This problem considers the nearest-neighbor learning algorithm in the following setting:

- The oracle produces two-dimensional instance vectors with labels generated according to a \textit{fixed} (but unknown) binary\(^2\) function \( f^* \). The oracle is also restricted to produce only instances where the first coordinate is non-zero; for example, it could present \((-\sqrt{14}, 0)\) but not \((0, -\sqrt{14})\). Aside from these restrictions, the oracle can be arbitrarily adversarial.

- The nearest-neighbor learner breaks ties by guessing a label of 0.

Now, suppose the oracle has presented two examples so far:

\[
\vec{x}(1) = (-1, 0) \quad \text{with label 1} \\
\vec{x}(2) = (1, 0) \quad \text{with label 0}.
\]

(a) Compute the nearest-neighbor learner’s Voronoi partition corresponding to \( \vec{x}(1) \) and \( \vec{x}(2) \) by doing the following: show \textit{mathematically}, explaining your steps, that for any \( \vec{z} = (z_1, z_2) \),

- if \( z_1 > 0 \), then \( \left[ \text{dist}(\vec{z}, \vec{x}(1)) \right]^2 > \left[ \text{dist}(\vec{z}, \vec{x}(2)) \right]^2 \), that is, \( \vec{z} \) is closest to \( \vec{x}(2) \);

- if \( z_1 < 0 \), then \( \left[ \text{dist}(\vec{z}, \vec{x}(1)) \right]^2 < \left[ \text{dist}(\vec{z}, \vec{x}(2)) \right]^2 \), that is, \( \vec{z} \) is closest to \( \vec{x}(1) \).

(Note: it’s easier to work with the square of the distance rather than the distance itself, and using the square is justified because the distance between two vectors is always non-negative.)

(b) Now, suppose that, unbeknownst to the learner, the target function \( f^* \) is the following:

\[
f^*(z_1, z_2) = \begin{cases} 
1 & \text{if } z_1 < 0, \\
0 & \text{otherwise}.
\end{cases}
\]

This means that just after seeing \( \vec{x}(2) \), the learner’s (labeled) Voronoi partition corresponds \textit{exactly} to \( f^* \), although the learner is unaware of this fact.

Observe that for perceptrons, if the oracle generates labels according to a \textit{fixed} target weight vector \( \vec{w}^* \) and obeys the gap restriction, then, if the learner ever had a weight vector \( \vec{w} \) equal to \( \vec{w}^* \), it would classify all subsequent examples from the oracle correctly. (Of course, recall that this could also be the case when \( \vec{w} \neq \vec{w}^* \) if the oracle obeys all the perceptron learning restrictions.)

\(^2\)This means that the values that the function can take are either zero or one.
Does the analogous situation hold for nearest-neighbor learning? That is, if the oracle is restricted in the ways we have described above and in the beginning of this problem, is it true that no matter what instances the oracle produces after \( \vec{x}^{(1)} \) and \( \vec{x}^{(2)} \), it can never cause the learner to make another mistake? If yes, explain why the learner will be right on every subsequent oracle example. If no, explain what sequence of examples the oracle could produce to cause the learner to eventually err and why the mistake(s) occur.

**Part C**

4. It is natural to consider whether our proof of the impossibility of computing the halting function can be extended to show that other things also cannot be implemented by Turing machines. This problem considers one possible such extension.

Let us posit the existence of a *Termination Alarm* \( \hat{D} \), a Turing machine with the following behavioral restrictions: given a description of a Turing machine \( M \) and an input \( x \) to \( M \), where \( x \) consists of some number of A’s,

- if \( M \) would eventually halt when given \( x \) as input, \( \hat{D} \) eventually halts with output 1.
- if \( M \) would never halt when given \( x \) as input, \( \hat{D} \) may either halt with output 0 or never halt.

Can we use the proof from class to show that the existence of \( \hat{D} \) would lead to a contradiction? More specifically, can we simply substitute \( \hat{D} \) for \( D \) in the proof from class to construct a Turing machine \( \hat{X} = M_k \), for some \( k \), such that \( \hat{X} \) has the contradictory behavior of needing both to halt and not halt when presented with \( k \) A’s as input? Explain your answer to this question; that is, if we can reuse the proof, give an updated version of the proof; if we cannot reuse the proof, explain precisely where the proof breaks down (i.e., where one step no longer logically follows from the previous steps).