cs6100 Spring 2004, Assignment 1 Answers

All questions are worth 5 points.

1. (Sipser, 3.3)
   A language is decidable iff some NTM decides it.

   Proof.
   $(\Rightarrow)$ Suppose $L$ is decidable. Then there is a DTM $M$ such that $M$ is a decider for $L$. Now $M$ can be straightforwardly coerced to be an NTM $N$ simply by mandating that $\delta_N(q,a) = \{\delta_M(q,a)\}$. Note that $N$ is a decider since it halts in all cases.

   $(\Leftarrow)$ Suppose $N$ is an NTM deciding $L$. That means that an execution of $N$ on input $w$ may be modelled with a computation tree. Since (a) there is only finite branching allowed at any transition of $N$ and, (b) each path through the tree is finite, the hint allows one to conclude that the entire computation tree is finite. (It seems obvious, and it is.) From an abstract point of view, we may implement a DTM $M$ deciding $L$ as follows: ‘run’ $N$ on $w$ and keep track of all branches on the tape of $M$. Depth-first search would do, since each branch terminates. We need simply to keep looking for a computation step that ends in the accept state. In fact, the breadth-first search implemented in the proof of 3.10 is also OK, since it is merely being cautious in avoiding infinite branches.

2. (Sipser, 3.9a)
   A 2-PDA is a machine with 2 stacks, described formally by a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where most fields of the tuple are as expected, except

   $\delta : Q \times \Sigma \times \Gamma \times \Gamma \rightarrow 2^Q \times \Gamma \times \Gamma$

   There are (at least) 2 answers to the question:

   • Show that, for every TM, there is a 2-PDA that simulates it
   • Show that there’s a single 2-PDA that can simulate any TM

   We’ll do the first, but the second is also possible (obviously, if the former is possible, the latter is as well, since universal TMs are just TMs).

   A TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R)$ may be simulated on input $w$ as follows. The basis of the simulation is that every state $q$ in $M$ has a corresponding state in the 2-PDA. The configuration of the TM is represented in the two stacks. There are 3 phases of simulation:

   (a) Initialization. $w$ is traversed left-to-right and its symbols are pushed into the left stack. Then the elements of the left stack are popped and pushed into the right stack. (We can’t simply push $w$ into the
right stack, since it would then be in reverse order.) After this step, the input has been consumed, and the 2-PDA has to execute via ε-transitions alone.

(b) Execution. If \( \delta_{TM}(q,a) = (q',a',d) \) then the actions we require of \( \delta_{PDA}(q,\varepsilon,\ell,r) = (q',\ell',r') \) are suggested by the following table, which uses \( L, R \) to represent the full stacks, not just the tops of the stacks, which is needed in the definition of \( \delta_{PDA} \).

<table>
<thead>
<tr>
<th>( d )</th>
<th>( (L, R) )</th>
<th>( (L', R') )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. right</td>
<td>( (L, a \cdot R) )</td>
<td>( (L \cdot a', R) )</td>
</tr>
<tr>
<td>2. right</td>
<td>( (L, \varepsilon) )</td>
<td>( (L, \omega \cdot R) )</td>
</tr>
<tr>
<td>3. left</td>
<td>( (L \cdot b, a \cdot R) )</td>
<td>( (L, b \cdot a' \cdot R) )</td>
</tr>
<tr>
<td>4. left</td>
<td>( (\varepsilon, a \cdot R) )</td>
<td>( (\varepsilon, a' \cdot R) )</td>
</tr>
</tbody>
</table>

In the table, rule (1) is straightforward; rule (2) allows the mimicry of the infinite series of blanks stretching off to the right; rule (3) shows what happens when the head moves left (and there is space to do so)—notice that this step actually performs two stack operations, which is easy to implement; and rule (4) shows what happens when the head wishes to move left, but can’t. This gives the following table describing \( \delta_{PDA}(q,\varepsilon,\ell,r) = (q',\ell',r') \)

<table>
<thead>
<tr>
<th>( d )</th>
<th>( (\ell, r) )</th>
<th>( (\ell', r') )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. right</td>
<td>( (\varepsilon, a) )</td>
<td>( (a', \varepsilon) )</td>
</tr>
<tr>
<td>2. right</td>
<td>( (\varepsilon, \varepsilon) )</td>
<td>( (\varepsilon, \omega) )</td>
</tr>
<tr>
<td>3. left</td>
<td>( (b, a) )</td>
<td>( (\varepsilon, b \cdot a') )</td>
</tr>
<tr>
<td>4. left</td>
<td>( (\varepsilon, a) )</td>
<td>( (\varepsilon, a') )</td>
</tr>
</tbody>
</table>

(c) Termination. \( \{q_A\} \) is the set of final states of the 2-PDA.

3. (Sipser, 3.9b) A Turing machine can easily simulate a 3-PDA and a 3-PDA can simulate a 2-PDA which we’ve shown can simulate a Turing machine. So we gain no power.

4. (Sipser, 3.10)
A write-once Turing machine has the same power as an ordinary TM.
The idea in the simulation is that one step of the TM will result in the entire contents of the tape being copied to a new segment at the end of the tape, with the altered cell being the only fresh information being written. Thus a sequence of steps taken by the TM will result in the tape contents being copied to the right repeatedly.

We’ll focus on a single step \( \delta(q, a) = (q', a', d) \) of computation of the TM \( M = (Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R) \). Let \( m \) be the number of states in \( Q \), and \( n \) be the number of elements of \( \Gamma \). We use an alphabet with \( m + m + n + m + n \) symbols. The \( m \) symbols are used to represent unmarked states. The \( m+n \) symbols are used to represent the current state and the current tape cell
The $m \times n \times 2$ symbols are used to represent an intermediate format. Thus, given input $a \cdot x$ to $M$, the write-once machine $M'$ starts in a configuration where the first tape cell is $(a, q_0) \cdot x$, where $(q_0, a)$ is a single symbol which represents the pairing of $a$ and $q_0$. We will require that only one such symbol—called the *marked square*—exists per tape segment.

The step $\delta(q, a) = (q', a', d)$ is implemented in two passes over the tape:

- **Pass 1.** Each symbol is copied to the end of the tape. An unmarked square is copied unchanged. A marked square $(q, a)$ is copied to a *marked square with direction* $(q', a', d)$. This latter kind of square records the result of the transition.

- **Pass 2.** We go to the beginning of the newly copied segment. Again, each symbol is copied to the end of the tape. We now have to use a little bit of look-ahead.
  - Two unmarked squares $a \cdot b \cdot x$ result in $a$ being copied to the end of the tape, and then the tape head returns to attempt to copy $b \cdot x$.
  - An unmarked square followed by a square marked with direction $L$, e.g., $a \cdot (q, b, L) \cdot x$ results in the marked square $(q, a)$ being copied and then the unmarked square $b$ being copied.
  - A square marked with direction $R$ followed by an unmarked square e.g., $(q, a, R) \cdot b \cdot x$, results in the unmarked square $a$ being copied and then the marked square $(q, b)$ being copied.

If any transition results in a marked square $(q_A, a)$ for some symbol $a$, the machine accepts, and the machine will reject when a marked square $(q_R, a)$ is transitioned into.

5. (Sipser, 3.11)
A Turing machine with a doubly infinite tape recognizes the class of Turing-recognizable languages.

Let $S$ denote the machine with the singly infinite tape and $D$ denote the machine with the doubly infinite tape.

To implement $D$ by $S$, we use a scheme where the tape of $S$ is divided into those cells indexed by even and odd numbers. The very first cell seems to need a sentinel value, which we will call $p$. No other cells will ever have this value. The cells of $S$ numbered $1, 3, 5, 7, \ldots$ will represent the part of the tape ‘going off to the right’, and the cells numbered $2, 4, 6, 8, \ldots$ represent the part of the tape going off to the left. The input $w_0 \ldots w_{n-1}$ will be written such that symbol $w_i$ is written to cell $2i + 1$, and $S$ will start with the tape head at cell 1. Tape symbols will be either dotted (occur on the right) or undotted (occur on the left). A transition step $\delta_D(q, a)$ of $D$ is implemented as follows by $S$:

- $\delta_D(q, a) = (q', a', R)$. There are two cases
– The symbol $a$ is undotted, signifying that we are on the ‘right’ portion of the tape. $S$ writes $a'$ and moves right. This puts the machine on a cell reserved for the tape going in the opposite direction, so the machine moves right again and enters state $q'$.

– The symbol $\dot{a}$ is dotted, signifying that we are on the ‘left’ portion of the tape. $S$ writes $\dot{a}'$ and moves left. This puts the machine on a cell reserved for the tape going in the opposite direction, so the machine moves left again. In normal operation, it enters state $q'$. However, it is also possible to ‘cross the boundary’ by having landed on the pivot cell (at $p$). In this case, $S$ should now move one cell to the right. Finally, the state enters $q'$.

• The cases when $D$ moves left are symmetric.

6. (Sipser, 3.12)
A Turing machine with left-reset recognizes the class of Turing-recognizable languages.

(Sketchy answer.) To implement this, we need merely keep an index to the current tape cell. If we have to go left in the TM, we reset and then move $n - 1$ steps to the right.

7. (Sipser, 3.13)
A Turing machine with stay put is not equivalent to ordinary Turing machines.

It is intuitively obvious that such a machine can implement a DFA only.

8. (Sipser, 3.16)
A language is decidable iff some enumerator enumerates the language in lexicographic order.

**Proof.**

$(\Rightarrow)$ Suppose $L$ is decidable. Thus it has a TM $M$ such that $L(M) = L$. A lexicographic enumerator can be built by enumerating the strings of $\Sigma^*$ in lexicographic order, one at a time, and only printing a string if it is accepted by $M$.

$(\Leftarrow)$ Suppose we have an enumerator $E$ that generates elements of $L$ in lexicographic order. A decider for $L$ can be built as follows:

(a) On input of $w$, let $n = \text{len}(w)$.

(b) Naively, we’d try the following:

Run $E$ until it produces a string having length $> n$. If $E$ has generated $w$ by then, accept; otherwise, reject as $E$ won’t ever generate $w$.

The problem with this solution is that $L$ might be finite, in which case $E$ may not generate a string having length $> n$. Now either $L$ is finite or infinite. If $L$ is finite, it is decidable, and we don’t have
to use $E$ to prove it is decidable. If $L$ is infinite, then $E$ will always produce a next larger string and the naive solution works.