CSE 105, Fall 2019 - Homework 6 Solutions

Due: Monday 11/25 midnight

Instructions: Upload a single file to Gradescope for each group. All group members’ names and PIDs should be on each page of the submission. Your assignments in this class will be evaluated not only on the correctness of your answers, but on your ability to present your ideas clearly and logically. You should always explain how you arrived at your conclusions, using mathematically sound reasoning. Whether you use formal proof techniques or write a more informal argument for why something is true, your answers should always be well-supported. Your goal should be to convince the reader that your results and methods are sound.

Reading: Sipser Sections 3.1, 3.2, 4.1

Key Concepts: TM, Decidability
Recall the terminology for describing Turing Machines as per the class slides:

- **Formal definition**: set of states, input alphabet, tape alphabet, transition function, state state, accept state, reject state. A state diagram is sufficient for defining the transition function.
- **Implementation-level definition**: English prose to describe Turing machine head movements relative to contents of tape.
- **High-level description**: Description of algorithm, without implementation details of machine. As part of this description, can "call" and run another TM as a subroutine.

### Problem 1 (10 points)

Let $L = \{ w \in \{a,b\}^* \mid |w|_a = 1 + |w|_b \}$ where $|w|_a$ denotes the number of occurrences of $a$ in $w$ and $|w|_b$ denotes the number of occurrences of $b$ in $w$.

Provide formal description of a Turing machine $M$ which decides $L$.

The transition function can be represented as a state diagram.

**Solution:**

Let # denote the blank symbol.

The formal description of $M$ is given by:

- $Q = \{q0, q1, q2, q3, q4, q_{acc}, q_{rej}\}$
- $Σ = \{a, b\}$
- $Γ = \{a, b, X, Y, #\}$
- $q_{accept} = q_{acc}$
- $q_{reject} = q_{rej}$

Transition function is given below:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>#</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>(q1, X, R)</td>
<td>(q4, X, R)</td>
<td>(q_{rej}, #, R)</td>
<td>(q_{rej}, X, R)</td>
<td>(q0, Y, R)</td>
</tr>
<tr>
<td>q1</td>
<td>(q2, a, R)</td>
<td>(q3, Y, L)</td>
<td>(q_{acc}, #, R)</td>
<td>(q_{rej}, X, R)</td>
<td>(q1, Y, R)</td>
</tr>
<tr>
<td>q2</td>
<td>(q2, a, R)</td>
<td>(q3, Y, L)</td>
<td>(q_{rej}, #, R)</td>
<td>(q_{rej}, X, R)</td>
<td>(q2, Y, R)</td>
</tr>
<tr>
<td>q3</td>
<td>(q3, a, L)</td>
<td>(q3, b, L)</td>
<td>(q_{rej}, #, R)</td>
<td>(q0, X, R)</td>
<td>(q3, Y, L)</td>
</tr>
<tr>
<td>q4</td>
<td>(q3, Y, L)</td>
<td>(q4, b, R)</td>
<td>(q_{rej}, #, R)</td>
<td>(q_{rej}, X, R)</td>
<td>(q4, Y, R)</td>
</tr>
</tbody>
</table>
Problem 2 (10 points)

L = \{1^n | n is a Fibonacci number and n>0\}. Construct a deterministic Turing Machine that decides L. Provide an implementation level description for M.

Solution:

We will construct a Turing Machine \( M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}) \) which decides L. We will use 3 additional tapes for our computation. We will denote these tapes as tape1, tape2 and tape3. Let \# denote the blank symbol.
Problem 3 (10 points)

Prove the following statements

a) The class of decidable languages are closed under concatenation
b) The class of recognizable languages are closed under Kleene star
Solution:

a) Let \( L_1 \) and \( L_2 \) be a Turing-decidable language. Then there exists a Turing Machine \( M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_1, q_{accept_1}, q_{reject_1}) \) that decides \( L_1 \) and a Turing Machine \( M_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2, q_2, q_{accept_2}, q_{reject_2}) \) that decides \( L_2 \) meaning \( M_1 \) and \( M_2 \) halt on every input, either in the accept state or in the reject state.

Let \( L = L_1 \circ L_2 \). We will construct a Turing machine \( M = (Q, \Sigma, \Gamma, q_0, q_{accept}, q_{reject}) \) such that \( L(M) = L \).

Construction of \( M \):
On input \( w \), non-deterministically split \( w \) into strings \( x \) and \( y \).
Run \( x \) on \( M_1 \) and \( y \) on \( M_2 \).
Accept \( w \) if both \( M_1 \) and \( M_2 \) accept. Else, reject.

We will now prove that \( L(M) = L \).

If \( w \in L \), then there exist \( x \) and \( y \) such that \( w = xy \) and \( x \in L_1 \) and \( y \in L_2 \).
Therefore, for all strings \( w \) in \( L \), there exists a partition of the string \( w \) into \( x \) and \( y \) such that \( x \in L_1 \) and \( y \in L_2 \). Hence, one of the non-deterministic branches will produce the correct split and thereby accept the string as \( M_1 \) will accept \( x \) and \( M_2 \) will accept \( y \).

If \( w \) is not in \( L \), then there exists no \( x \) and \( y \) such that \( w = xy \) and \( x \in L_1 \) and \( y \in L_2 \).
Therefore, no branch of non deterministic computation can find \( x \) and \( y \) such that \( M_1 \) accepts \( x \) and \( M_2 \) accepts \( y \). Hence, \( M \) will reject \( w \).

There exists only \(|w|+1\) partitions of the string \( w \). As there exists only finite partitions of \( w \) and each branch of non deterministic computation halts as \( M_1 \) and \( M_2 \) halts on all inputs in accept or reject state, \( M \) halts on all inputs in accept or reject state.

b) Let \( L \) be a turing recognizable language and let \( M \) be the TM that recognizes it.

We construct a NTM \( M' \) that recognizes the star of \( L \):

On input \( w \):
1. Nondeterministically cut \( w \) into parts so that \( w = w_1w_2...w_n \).
2. Run \( M \) on \( w_i \) for all \( i \). If \( M \) accepts all of them, accept.
   If it halts and rejects any of them, reject.

Now we prove \( L(M^*) = L^* \).
Let \( w \in L^* \). Then there is some partition of \( w \) such that \( w = w_1w_2...w_n \) such that \( w_i \in L \) for all \( i = 1, \ldots, n \). Then, there is some \( t \geq 1 \) such \( M \) accepts all \( w_i \) after \( t \) steps and \( w \in L(M^*) \).
Now consider a \( w \in L(M^*) \). Then, there exists some partition of \( w \) such that \( M \) accepts each \( w_i \) after \( t \) steps. Therefore, \( L(M^*) = L^* \).

Problem 4 (10 points)
A queue automaton is like a PDA except that the stack is replaced by a queue. A queue is a tape allowing symbols to be written only on the left-hand end and read only at the right-hand end. Each write operation (we’ll call it a push) adds a symbol to the left-hand end of the queue and each read operation (we’ll call it a pull) reads and removes a symbol at the right-hand end. As with a PDA, the input is placed on a separate read-only input tape, and the head on the input tape can move only from left to right. The input tape contains a cell with a blank symbol following the input, so that the end of the input can be detected. A queue automaton accepts its input by entering a special accept state at anytime. Show that a language can be recognized by a deterministic queue automaton iff the language is Turing-recognizable.

Solution:
First, show that we can simulate a queue automaton with a Turing Machine. Assume that the input string is on TM tape. Move to the end of the string and insert a new character, say #, to denote the beginning of the queue. Then shift this delimiter and each character of the input tape to the right one symbol. At the beginning of the string, insert another delimiter $. The $ symbol will denote which symbol is currently being read in the queue automaton simulation, and the queue itself will be maintained at the end of the string between the $ and the end-of-tape character. Now, run as a subroutine of the TM the queue automaton. Whenever the queue automaton would pop an element on the queue, run a subroutine instead for the TM that would move to the leftmost edge of the queue portion of the tape, read in that symbol and then shift each element of the queue to the left by one position. Likewise, for push, move to the end of the queue portion of the tape, move the end-of-tape character one position to the right, and then insert the target symbol at the old position of the end of the tape character.

To show that a queue automaton can simulate a Turing Machine, start with the input on a separate read-only tape as per the format, and an empty queue. To initialize the queue, push a delimiter $ and then each character of the input string in succession. Our queue will contain the contents of the Turing Machine tape at all times, with the symbol currently being read at the head of the queue. The symbols between head and $ denotes the symbols to the right of the input symbol and the elements between $ and the tail of the queue denotes the symbols to the left of the input symbol.

To perform the operation $\delta(q, a) = (q', b, R)$:

The head of the queue will have the symbol currently being read, i.e., ‘a’. Pop the symbol at the head of the queue. Push the correct symbol according to the transition function (i.e., ‘b’ in this case at the tail of the queue. Move the state of the queue automation from q to q'.

To perform the operation $\delta(q, a) = (q', b, L)$:

The head of the queue will have the symbol currently being read, i.e., ‘a’. Pop the symbol at the head of the queue. Push the correct symbol according to the transition function (i.e., ‘b’ in this case at the tail of the queue. Move the state of the queue automation from q to q'. Perform the steps 1, 2 and 3 twice.
1. Push a marker # at the tail of the queue. Replace each queue symbol x with a new symbol (w,x), where w is the symbol immediately to x’s left in the queue. If a is the symbol at the head of the queue, a should be replaced with (#,a). For this we use a special set of separate states that “remember” the last symbol shifted. That is, for each w ∈ Σ, after having shifted symbol w, we are in a special state q_w. Then when we encounter the next symbol x, we enqueue not x, but the new symbol (w,x). To start the process we enqueue # and move to state q_. We then repeat the following: From a given state q_w, dequeue a symbol x, enqueue the new symbol (w,x), and move to state q_x. When we dequeue #, we enqueue the final new symbol.

2. We then repeatedly dequeue (w,x), enqueue (w,x) until we dequeue (w,x) where x = #. We then enqueue #, enqueue w, and dequeue whatever symbol is at the head of the queue.

3. We then repeatedly dequeue (w,x) and enqueue w, until we dequeue #. At this point our original queue will be cyclically shifted one position to the right.

Problem 5 (10 points)

Prove that a language is turing-decidable if and only if some enumerator enumerates the strings of this language in lexicographic order.

Solution:
Claim 1: If a language is Turing decidable, then some enumerator enumerates the strings of the language in lexicographic order
Proof:
Let L be a Turing decidable language. There exists a Turing machine M = (Q, Σ, Γ, δ, q_0, q_accept, q_reject) which decides L.
We will construct an enumerator E for this language which will enumerate the strings of L in lexicographic order.
Construction of E:
E generates each string in Σ* in lexicographic order and then runs the string on M. If M accepts, it prints the string. Otherwise, it does not.

Claim 2: If there exists some enumerator E which enumerates the strings on the language in lexicographic order, then the language is decidable
Proof:
Let L be the language for which E enumerates the strings in lexicographic order. We will construct a Turing machine M for L such that M halts on every input, either in the accept state or in the reject state.
Case 1 : L is finite.
Every finite language is decidable as we can construct a Turing machine $M$ for $L$ in which each string of $L$ is hardwired in $M$.

Case 2: $L$ is infinite
On receiving input $w$, $M$ runs $E$ to enumerate all strings in $L$ in lexicographic order until some string lexicographically after $w$ appears. If $w$ has appeared in the enumeration already, then accept. Otherwise reject.

$M$ is a decider as a string which is lexicographically after $w$ will be printed by $E$ as $L$ is infinite. Therefore, $M$ will halt in accept or reject state for every input.