CSE 103 Computational Models

Fall 2021 Prof. Fremont **HW 6**

(3 questions, 280 points total)

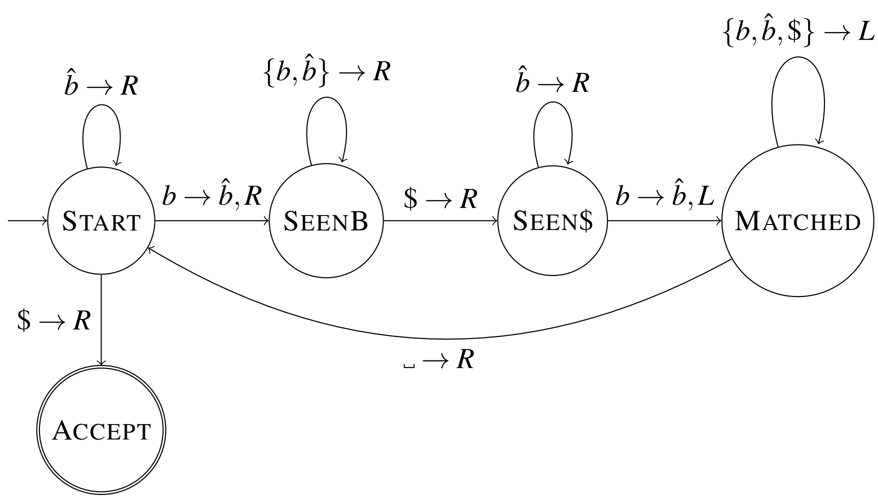
1. (100 pts.) Working with a TM

Consider the Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ where:

- $Q = \{\textsc{Start}, \textsc{SeenB}, \textsc{Seen\$}, \textsc{Matched}, \textsc{Accept}, \textsc{Reject}\}$
- $\Sigma = \{b, \$\}$
- $\Gamma = \{b, \hat{b}, \$, \text{␣}\}$
- $$\delta(q,s) = \begin{cases} (\textsc{Accept}, s, R) & q = \textsc{Start} \text{ and } s = \$ \\ (\textsc{SeenB}, \hat{b}, R) & q = \textsc{Start} \text{ and } s = b \\ (\textsc{Seen\$}, s, R) & q = \textsc{SeenB} \text{ and } s = \$ \\ (q, s, R) & q = \textsc{SeenB} \text{ and } s = b \\ (\textsc{Matched}, \hat{b}, L) & q = \textsc{Seen\$} \text{ and } s = b \\ (\textsc{Start}, s, R) & q = \textsc{Matched} \text{ and } s = \text{␣} \\ (q, s, L) & q = \textsc{Matched} \text{ and } s \in \{b, \hat{b}, \$\} \\ (q, s, R) & s = \hat{b} \\ (\textsc{Reject}, s, R) & \text{otherwise} \end{cases}$$
- $q_0 = \textsc{Start}$
- $q_{\text{accept}} = \textsc{Accept}$
- $q_{\text{reject}} = \textsc{Reject}$

(a) (*45 pts.*) Draw M as a graph, omitting the reject state and all transitions leading to it (following the convention used in class, we will say that all missing transitions lead to the reject state).

Solution:



(Note that labels like $\{b, \hat{b}\} \to R$ are just shorthand for the two transitions $b \to R$ and $\hat{b} \to R$, which in turn are shorthand for $b \to b, R$ and $\hat{b} \to \hat{b}, R$; any of these notations is fine.)

(b) (*15 pts.*) List the sequence of configurations of M (the computation history) when run on input $b\$bb$. Please put each configuration on a separate line.

Solution:

$$\begin{array}{r}
\textsc{Start}\ \ b\ \ \$\ \ b\ \ b \\
\hat{b}\ \ \textsc{SeenB}\ \ \$\ \ b\ \ b \\
\hat{b}\ \ \$\ \ \textsc{Seen\$}\ \ b\ \ b \\
\hat{b}\ \ \textsc{Matched}\ \ \$\ \ \hat{b}\ \ b \\
\textsc{Matched}\ \ \hat{b}\ \ \$\ \ \hat{b}\ \ b \\
\textsc{Matched}\ \ \text{\textvisiblespace}\ \ \hat{b}\ \ \$\ \ \hat{b}\ \ b \\
\textsc{Start}\ \ \hat{b}\ \ \$\ \ \hat{b}\ \ b \\
\hat{b}\ \ \textsc{Start}\ \ \$\ \ \hat{b}\ \ b \\
\hat{b}\ \ \$\ \ \textsc{Accept}\ \ \hat{b}\ \ b
\end{array}$$

(c) (*20 pts.*) For each of the following input strings, state whether M accepts, rejects, or does not halt.

- ε
  Solution: This is rejected, since the symbol under the head is a blank, and $\delta(\textsc{Start}, \text{\textvisiblespace}) = (\textsc{Reject}, \text{\textvisiblespace}, R)$, so we transition to the reject state and halt.
- $bb\$b$
  Solution: This is rejected: following a similar pattern to the computation history above, M will mark the first b on either side of the \$, reaching the configuration $\textsc{Start}\ \hat{b}\ b\ \$\ \hat{b}$. It will then proceed to mark the remaining b and move past the \$ and the $\hat{b}$, reaching the configuration $\hat{b}\ \hat{b}\ \$\ \hat{b}\ \textsc{Seen\$}$. But now the symbol under the head is a blank, and in state $\textsc{Seen\$}$ this causes a transition to the reject state.
- $bb\$bbb$
  Solution: This is accepted, along the same lines as $b\$bb$ above. Each pass through the four states in the top row of the graph shown in part (a) marks the first b on each side of the \$, so that after two passes we will be in the configuration $\textsc{Start}\ \hat{b}\ \hat{b}\ \$\ \hat{b}\ \hat{b}\ b$. Then M reads past the $\hat{b}$s to get to the configuration $\hat{b}\ \hat{b}\ \textsc{Start}\ \$\ \hat{b}\ \hat{b}\ b$, whereupon M reads the \$ and transitions to the accept state.
- $\$b$
  Solution: This is accepted: the symbol under the head is a \$, so we transition immediately to the accept state.

(d) (*20 pts.*) What is the language of M?

Solution: As the examples above suggest, the behavior of the four main states is to find a b to the left of the \$, mark it, then proceed past the \$ and find another b to mark (rejecting if there is no \$ at all, or we find a second \$ before finding a b). If there is no other b to mark, we will reach the end of the tape in state $\textsc{Seen\$}$ and reject. Otherwise, we mark the b, and the $\textsc{Matched}$ state brings us back to the start of the input and we repeat. Whenever one of these passes succeeds (i.e. does not reject partway through), we mark one b on each side of the \$. Once there are no unmarked bs to the left of the \$, the $\textsc{Start}$ state will move past all $\hat{b}$s and reach the \$, then moving to the accept state. So M accepts if and only if the input string starts with a prefix of the form $b^n\$b^m$ where $m \geq n$. Formally, we have $L(M) = \{b^n\$b^m x \mid m \geq n \geq 0 \text{ and } x \in \Sigma^*\}$.

2. (80 pts.) Fleshing out a TM description

Consider the following language over $\Sigma = \{0, 1, \#, \$\}$, which represents a simple form of array lookup, namely checking if the kth element of the array $(e_0, e_1, \ldots, e_n)$ is equal to a given binary string v:

$$L = \{1^k\#v\$e_0\$e_1\$\ldots\$e_n\$ \mid 0 \le k \le n,\ \ v, e_0, \ldots, e_n \in \{0,1\}^*, \text{ and } e_k = v\}.$$

(So for example 11#011\$0\$100\$011\$11\$ $\in L$ but 11#011\$0\$100\$010\$11\$ $\notin L$.)

Consider the following medium-level description (what the textbook calls an "implementation description") of a TM deciding L, which uses the tape alphabet $\Gamma = \{0, 1, \#, \$, X, \sqcup\}$ where X is a new symbol representing a "crossed-off" part of the tape:

1. Go through each of the 1s at the start of the input; for each one, cross it off and scan right until you find a \$, and cross that off as well, rejecting if there are no \$s left. (After this step, the leftmost remaining \$ will be the one just before e_k.)
2. Go through each of the symbols of v; for each one, remember whether it is a 0 or a 1, then scan right until you find a \$. Scan right until you find a 0 or 1, and reject if it doesn't agree with the symbol of v we saw earlier.
3. Go back to the #, then move right to the first \$. Continue moving right, and if you see another \$ without encountering any 0s or 1s along the way, then accept.

Let's flesh out this description into a more detailed one listing all the states and what should be done at each one. Use English, like the low-level description given in the 11/8 lecture (on page 2 of the notes): do not write out the TM as a graph.

(a) (*20* pts.) Describe how to implement step (1) above. You should only need 3 states, but don't worry about having exactly this number.

Solution:

We'll use three states, A, B, and C, which will handle finding the next 1, finding the corresponding \$, and returning to the start of the input respectively.

A**:** Move right past all Xs (so that we ignore all 1s we've previously handled and crossed out). If we find a #, then we've finished going through the start of the input and we'll move on to part (b) of this question. If we find a 1, then replace it with an X and go to state B, which will handle finding the corresponding \$. Otherwise, we find a 0, \$, or blank, none of which are allowed in this part of the input, so we'll reject.

B**:** Move right past all symbols except \$ and blanks. If we reach a blank, then we read through the whole input without finding a \$, so we reject. Otherwise we replace the \$ with an X and go to state C, which will handle returning to the first part of the input so we can continue going through the 1s.

C**:** Move left past all symbols except blanks. Once we reach a blank, we have moved all the way past the input, so we move right (leaving the head at the beginning of the input) and go to state A to proceed.

(b) (*40* pts.) Describe how to implement step (2) above, assuming the head of the TM is already at the first symbol of v. It's possible to do this with 6 states.

(*Hint:* If you're having trouble figuring this out, Example 3.9 in the textbook may be helpful.)

Solution:

We'll start with a state A which handles finding the next symbol of v, similarly to state A above. There is one important difference, however: we can't use the symbol X to cross off symbols of v we've already read, since we may have used X to cross out the \$ after v and so we'd end up reading into e_0 by mistake. So we'll use the blank symbol ␣ instead.

Next, we'll have a state B to find the \$ and a state C to find the next 0 or 1; however, since we need to remember whether the symbol we saw in state A was a 0 or a 1, we'll actually have two copies B_0, C_0 and B_1, C_1 of these states. Finally, we'll have a state D to handle returning to the start of v so we can continue iterating through its symbols.

A: Read past all blanks to skip the bits of v we've already handled. If we find a \$ or an X, then we've finished going through v and we'll move on to part (c) of this question. If we find a #, the string is not of the right form (there should only be a # before v), so we reject. Otherwise we find a 0 or a 1; we replace it with a blank and go to state B_0 or B_1 accordingly.

B_0: Move right until we find a \$. If we encounter a blank, then we've made it past the entire input without finding a \$, so we reject; we also reject if we find a #, since the string then has multiple #s. Otherwise we find a \$, so we move right and go to state C_0.

C_0: Move right past all Xs (to ignore bits of e_k that we've already handled). If we find a 0, then this matches the 0 we saw earlier in v, so we replace it with X, move left, and go to state D. If we find a 1, then $v \neq e_k$, so we reject. If we find a \$ or a blank, then e_k is shorter than v, so we also reject. If we find a #, then the string has multiple #s, so we reject.

B_1: The same as B_0, except we go to state C_1 if we find a \$.

C_1: The same as C_0, except with 0 and 1 exchanged: if we find a 1, it matches the 1 we saw earlier, so we blank it out, move left, and go to state D; if we find a 0, we reject.

D: Move left until we find a #, then move right and go to state A.

(c) (*20* pts.) Describe how to implement step (3) above. It's possible to do this with 4 states.

Solution: This is similar to part (a). We'll use a state A to go back to the #, B to search for the first \$, and C to scan for any additional \$s.

A: Move left until we find the # (which the earlier parts of the problem have confirmed to be unique), then move right and go to state B.

B: Move right until we find a \$ (which we're guaranteed to do if we've gotten this far); then move right and go to state C.

C: Move right past all Xs. If we find a 0 or a 1, then e_k is longer than v, so we reject. If we find a #, the string has multiple #s, so we reject. If we find a blank, then the string is not terminated by a \$ as required, so we reject. Finally, if we find a \$, then we accept.

3. (100 pts.) Designing a TM

Design a TM that can compare two integers represented in binary; more precisely, that recognizes the language L of strings $x\$y$ where $x, y \in \{0,1\}^*$, $|x| = |y|$ (for simplicity), and $x \leq y$ when interpreted as integers. For example, 010\$100 and 001\$010 are in L but 1\$0 and 0\$00 are not. Give a medium-level description of your machine, like the 3-step description in the statement of Problem 2 above.

Solution: First we will check that the input has the right form, namely $x\$y$ with $x, y \in \{0,1\}^*$ and $|x| = |y|$. We can do this using a similar procedure to that of Problem 1, making several passes through the input, marking one unmarked symbol of both x and y at a time. To determine whether $x \leq y$ as integers, we'll again iterate through their bits: if the MSB of x is less/greater than the the MSB of y, then x is less/greater than y, so we can accept/reject immediately. If instead the MSBs are the same, then we'll continue on to the second bit, since we can't yet tell which of x or y is larger (or if they're the same). If we get through all the bits of x and y without ever finding one bit to be different from its counterpart, then $x = y$ and we'll accept. More precisely:

1. For each symbol before the \$, mark it and move past the \$, marking the next unmarked symbol. We reject if we do not encounter \$ at all, if there are no unmarked symbols left beyond the \$ (which would mean $|y| < |x|$), or if we encounter a second \$.
2. Once all the symbols before the \$ are marked, scan the symbols after the \$ and reject if any are unmarked (since this would mean $|x| < |y|$) or are a \$ (since this would mean the input contains multiple \$s). Otherwise, steps (1) and (2) ensure that the string has the form $x\$y$ with $x, y \in \{0,1\}^*$ and $|x| = |y|$.
3. Go through each bit before the \$, i.e., each bit of x. For each one, remember whether it is a 0 or a 1, then move to the corresponding bit after the \$, i.e., the corresponding bit of y (we can mark or cross off symbols as we process them to identify the next bit to go to). If the bit of x was 0 and the bit of y is 1, accept. If the bit of x was 1 and the bit of y is 0, reject. Otherwise the bits are equal, so we can't yet tell which of x or y is greater, so we continue.
4. If we get through all bits of x without rejecting, then $x = y$, so we accept.