Solutions: Homework 6

Prepared by Chien-Chung Huang

1)

Answer: Here is the strategy for how the TM should work:

(0) Remember the symbol scanned as "s", turn it into "x", and move right.

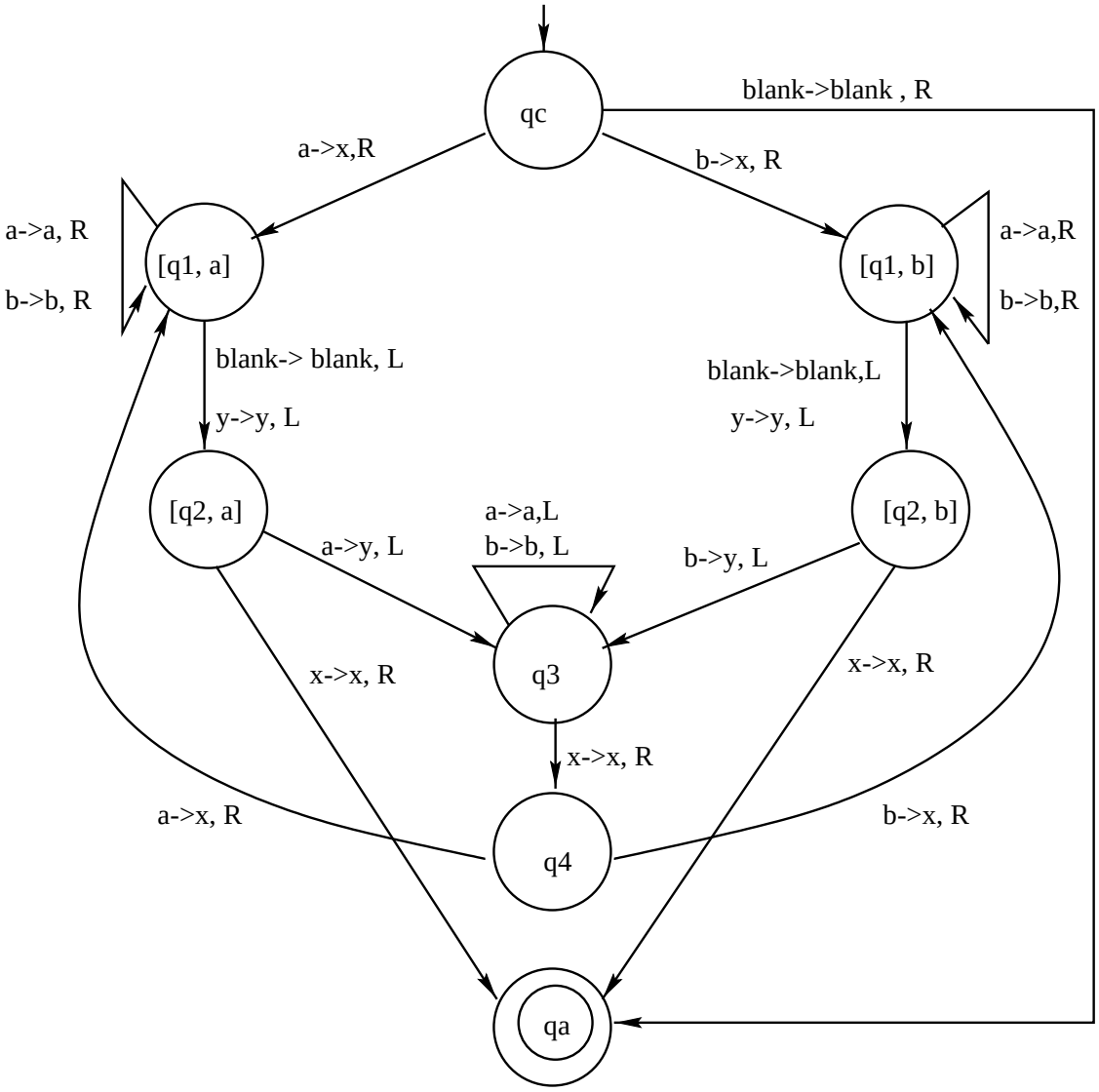
(1) Move right until hitting a blank or "y" and then move left by one position. Here we expect to find the match for "s".

(2) If the head finds "x", that means the string is a palindrome of odd length, so accept.
If the head finds opposite of "s", the string is not a palindrome, so reject.
If the head finds "s", mark it as "y" and move left.

(3)Move left until hitting "x", and then move right by one position. Here we expect to find the leftmost unconsumed "a" or "b".

(4) If the head scans "a" or "b", remember this symbol, and go to Step1.
If the head scans "y", it means that the first half of input already matched with the second half, so accept.

$\{w \mid w \in (a \cup b)^*, w \text{ is a palindrome } \}$



.

The states there have the following meaning:
q_0: Start state, implementing Step 0.
$[q_1, s]$: State implementing Step 1, when the symbol turned earlier into "x" is "s".
$[q_2, s]$: State implementing Step 2.
q_3: State implementing Step 3.
q_4: State implementing Step 4.

2)

Answer: Here is the strategy for how the TM should work:

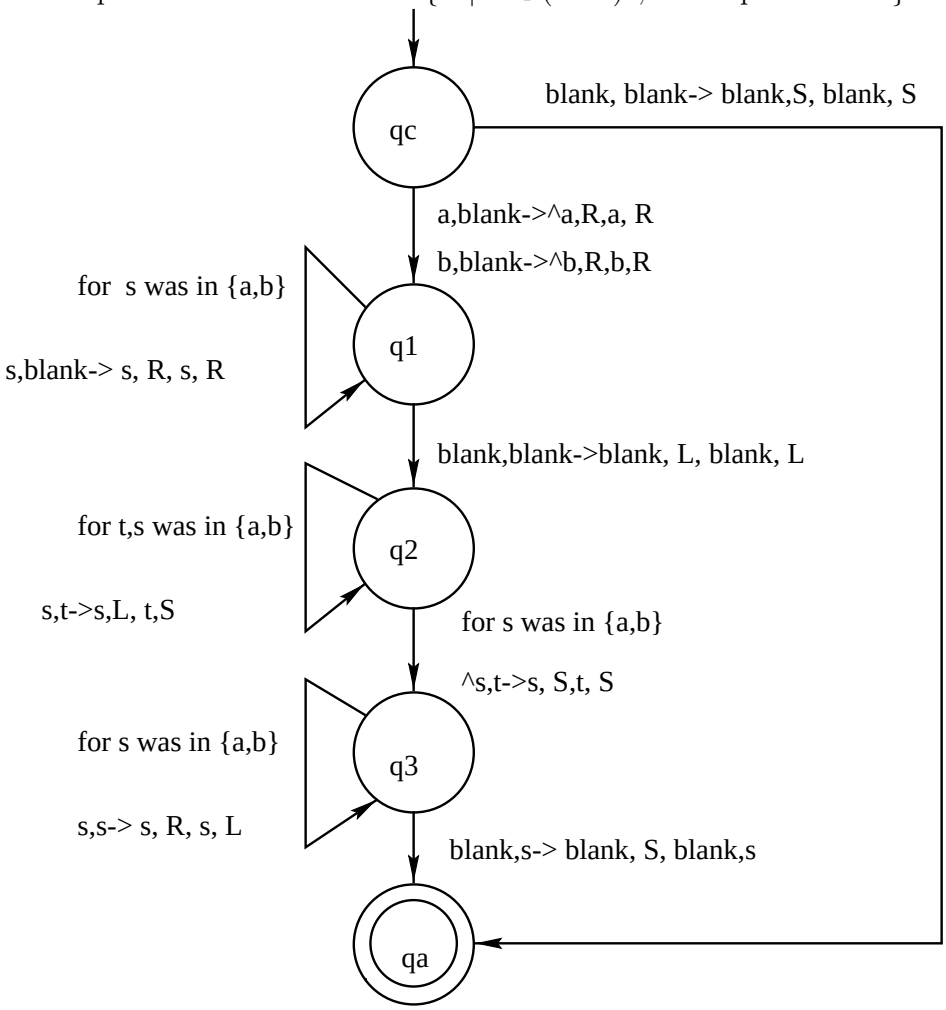
(0) Copy the symbol from first tape to second tape, turn the first symbol on first tape from "s" to "$\hat{s}$", and move right on both tapes.

(1) Keep moving to the right on both tapes, copying each symbol on the first tape on to the second, until hitting a blank on the first tape. At this point, move both heads to left by one position.

(2) Move the first tape's head all the way to the left (i.e., until it hits $\hat{a}$ or $\hat{b}$), keeping the second head stationary. Change "$\hat{s}$" back to "s" on first tape. At this point, the first head is on the first symbol of input, the second head is on the last symbol of input.

(3) Move first head to right and the second head to left, simultaneously checking that symbol under the heads match. Do this until first head hits a blank, and accept. If, at any point, the symbols under the two heads don't match, reject.

Two tapes deterministic TM for $\{w \mid w \in (a \cup b)^*, w \text{ is a palindrome } \}$



The states there have the following meaning: State q_i implements Step i above.

3)

Answer: Here is the strategy for how the nondeterministic TM should work:

(0) Change the first symbol from a or b to $\hat{a}$ or $\hat{b}$, respectively. This is done to be able to later identify the leftmost symbol on the tape.

(1) The machine splits into two incarnations: the first incarnation believes that the rest of the input (on the first tape) comprises the second half of the input string, and the second incarnation believes that the midpoint of the input is farther away.

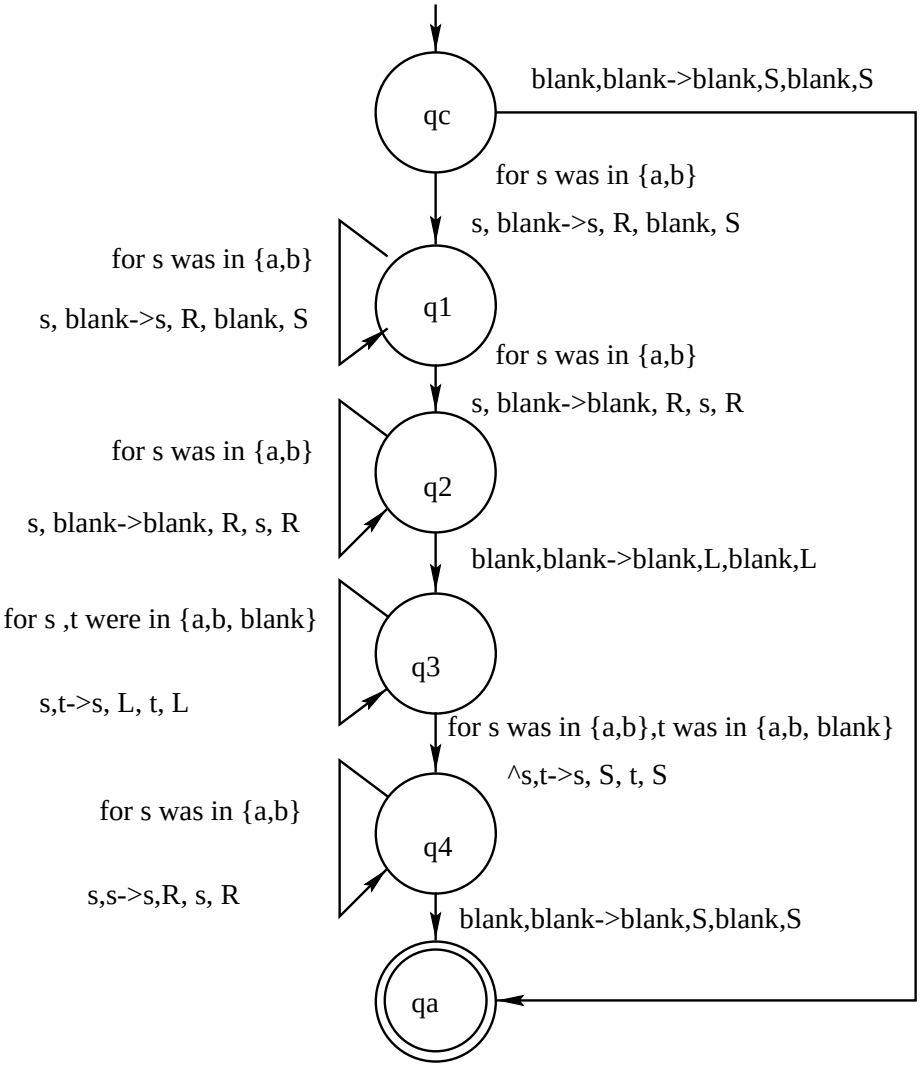
The second incarnation therefore simply moves to the next symbol on the first tape and repeats Step 1. The first incarnation proceeds to Step 2.

(2) Copy the rest of the input on the first tape onto the second tape, simultaneously erasing the first tape. Do this until hitting a blank on first tape.

(3) Move both heads to the left until the first tape's head is under $\hat{a}$ or $\hat{b}$. At this point, both heads are under the leftmost symbols of their respective tapes. Change "$\hat{a}$" on first tape to "a" (or "$\hat{b}$" to "b").

(4) Move both heads to right, simultaneously checking that symbols under the heads match. If, at any point, the symbols under the two heads don't match, reject. If both heads scan blanks, accept.

Design a nondeterministic two tapes TM for $\{w \mid w \in (a \cup b)^*\}$



The states there have the following meaning: State q_i implements Step i above.

In the following, let L_1 and L_2 be decidable languages, recognized by the decider TMs M_1 and M_2, respectively.

4.a) $L_1 \cup L_2$ is decidable because it is recognized by a decider Turing Machine M, described as follows:
M= "on input w
(1) Run M_1 on w and M_2 on w.
(2) If either M_1 or M_2 accepts w, accept; otherwise reject."
Clearly, M is a decider and it recognizes $L_1 \cup L_2$.

4.b) $L_1 \cap L_2$ is decidable because it is recognized by a decider Turing Machine M, described as follows:
M= "on input w
(1) Run M_1 on w until it halts.
(2) If M_1 reject w, reject and halt;
(3) Run M_2 on w until it halts.
(4) If M_2 accept w, accept; otherwise reject."

4.c) Complement of L_1 is decidable because it is recognized by a decider Turing Machine M, described as follows:
M= "on input w
(1) Run M_1 on w until it halts.
(2) If M_1 accept w, reject; otherwise accept."

4.d) L_1L_2 is decidable because it is recognized by a nondeterministic decider Turing Machine M, described as follows:
M= "on input w
(1) Nondeterministically divide w into two parts u, v (i.e., w=uv).
(2) Run M_1 on u and M_2 on v until they halt.
(3) If both halted in accept states, accept; otherwise reject."
We make two observations:
(1) If w is in L_1L_2, then some incarnation of M accepts it.
Therefore, $L(M) = L_1L_2$.
(2) Since M_1 and M_2 are deciders, it is clear that every incarnation of M halts.
Therefore, M is a nondeterministic decider.

Since every nondeterministic decider TM can transformed into an equivalent deterministic TM, it follows that some deterministic TM recognizes L_1L_2. Therefore, L_1L_2 decidable.

4.e) L_1^* is decidable because it is recognized by a nondeterministic decider Turing Machine M, described as follows:
M= "on input w
(1) If $w = \varepsilon$, accept and halt.
(2) Nondeterministically divide w into $u_1u_2...u_k$, for some $k \geq 1$.
(3) Run M_1 on each of $u_1u_2...u_k$.
(4) If M_1 accept all of $u_1u_2...u_k$, accept; otherwise reject."
We make two observations:
(1) If w is in L_1^*, then clearly some incarnation of M accepts it.
Therefore, $L(M) = L_1^*$.
(2) Since M_1 is a decider, it is clear that every incarnation of M halts.

Therefore, M is a nondeterministic decider. We know from class that it can be transformed into an equivalent deterministic decider.

In the following, let L_1 and L_2 be recognizable languages, recognized by the turing machines TMs M_1 and M_2, respectively.

5.a) $L_1 \cup L_2$ is recognizble e because it is recognized by a Turing Machine M, described as follows:
M= "on input w
(1) Run M_1 on w and M_2 on w simultaneously.
(2) If either M_1 or M_2 accepts w, accept."
Clearly, M is a Turing Machine and it recognizes $L_1 \cup L_2$.

5.b) $L_1 \cap L_2$ is recognizable because it is recognized by a Turing Machine M, described as follows:
M= "on input w
(1) Run M_1 and M_2 on w simultaneously.
(2) If both M_1 and M_2 accepts w, accept.;
Clearly, M is a Turing Machine and it recognizes $L_1 \cap L2$.

5.c) L_1L_2 is recognizable because it is recognized by a nondeterministic decider Turing Machine M, described as follows:
M= "on input w
(1) Nondeterministically divide w into two parts u, v (i.e., w=uv).
(2) Run M_1 on u and M_2 on v simultaneously.
(3) If both halted in accept states, accept."
We make two observations:

(1) If w is in L_1L_2, then some incarnation of M accepts it.
Therefore, $L(M) = L_1L_2$.
(2) A nondeterministic turing machine is equivalent to a deterministic one in power. So L_1L_2 is reconginizable.

6) Answer: We arrange the strings on the tape of M' in such a way every ith symbol on j tape of M is put in the $k * i + j$th cell. Assuming that the maximum length of the string on the k tapes of M is $O(t)$, then the length of the M' tape is still $O(kt) = O(t)$.

To simulate every single move of M, M' has to first "track" the positions of the k heads that it wishes to simulate. To "track" the heads, M has to move its head across the tape k times. The complexity of finding the k heads is thus $O(k*$ size of M's tape). The size of M' tape is originally $O(t)$; note that it is possible every step of M simply adds one more character to the rightmost cell of the tape. In terms of M', this implies that a move of M may add up to k symbols to its tape. We know that M takes t steps, therefore, the size of M' tape will possibly grow to $O(t) + kt$, which, fortunately, is still $O(t)$.

From the above discussion, we know that the complexity of moving the single head of M' to simulate the k heads of M is $O(t)$, for each step. Since M takes t steps, then the total complexity of M' is $O(t^2)$.

As a sidenote, we don't have to worry about the complexity of simulating the transition of each single head, as this is constant. This constant timed with k still will not influence $O(t^2)$.