COMP272 Question Bank 3 Suggested Solution

Turing Machines

Question 1

When n is odd, the machine will erase all the a's in tape squares at the odd number position. When n is even, the machine will erase all the a's in tape squares at the even number position.

Question 2

Let M be the Turing machine $(K, \Sigma, \delta, s, \{h\})$, where

$$K = \{q_0, q_1, h\},$$
$$\Sigma = \{a, b, \sqcup, \triangleright\},$$
$$s = q_0,$$

and δ is given by the following table.

$q,$	σ	$\delta(q, \sigma)$
q_0	a	$(q_1, \rightarrow)$
q_0	b	$(q_0, \rightarrow)$
q_0	$\sqcup$	$(q_0, \rightarrow)$
q_0	$\triangleright$	$(q_0, \rightarrow)$
q_1	a	(h, a)
q_1	b	$(q_0, \rightarrow)$
q_1	$\sqcup$	$(q_0, \rightarrow)$
q_1	$\triangleright$	$(q_0, \rightarrow)$

Question 3

If the first two alphabets of w is not blank, then write them in order in the first two blank tape squares after w.

Question 4

(a) $\triangleright\underline{\sqcup}aabb$

$\triangleright \sqcup \underline{a}abb$

$\triangleright \sqcup \underline{\sqcup}abb$

$\triangleright \sqcup \sqcup abb \sqcup \underline{\sqcup}$

$\triangleright \sqcup \sqcup abb \sqcup \underline{a}$

$\triangleright \sqcup \underline{\sqcup}abb \sqcup a$

$\triangleright \sqcup \underline{a}abb \sqcup a$
$\triangleright \sqcup a\underline{a}bb \sqcup a$
$\vdots$
$\triangleright \sqcup aabb\underline{\sqcup}aabb$
$\triangleright \sqcup aabb \sqcup aabb\underline{\sqcup}$
The TM transforms $\triangleright\underline{\sqcup}w$ into $\triangleright \sqcup w \sqcup w\underline{\sqcup}$.

(b) $\triangleright \sqcup aabb\underline{\sqcup}$
$\triangleright \sqcup aab\underline{b}$
$\triangleright \sqcup aab\underline{\sqcup}$
$\triangleright \sqcup aab \sqcup \underline{\sqcup}$
$\triangleright \sqcup aab \sqcup \underline{b}$
$\triangleright \sqcup aab\underline{b}b$
$\triangleright \sqcup aa\underline{b}bb$
$\triangleright \sqcup aa\underline{\sqcup}bb$
$\triangleright \sqcup aa \sqcup bb\underline{\sqcup}$
$\triangleright \sqcup aa \sqcup bb\underline{b}$
$\triangleright \sqcup aa\underline{b}bbb$
$\triangleright \sqcup a\underline{a}bbbb$
$\vdots$
$\triangleright \sqcup aabbbbaa\underline{\sqcup}$
The TM transforms $\triangleright \sqcup w\underline{\sqcup}$ into $\triangleright \sqcup ww^R\underline{\sqcup}$.

Recursive & R.E. Languages

Question 1

a)

$$\begin{tikzcd}
{>R} \arrow[loop above, "a"] \arrow[r, "b"] \arrow[dr, "\sqcup"] & R \arrow[loop above, "a"] \arrow[r, "b"] \arrow[d, "\sqcup"] & R \arrow[r, "\sqcup"] \arrow[dl, "{a,b}"] & y \\
 & n & &
\end{tikzcd}$$

b)

$$\begin{tikzcd}
{>R} \arrow[loop above, "a"] \arrow[r, "b"] \arrow[dr, "\sqcup"] & R \arrow[loop above, "a"] \arrow[r, "b"] \arrow[d, "\sqcup"] & R \arrow[r, "\sqcup"] \arrow[dl, "{a,b}"'] & y \\
 & R \arrow[loop below, "{a,b,\sqcup}"] & &
\end{tikzcd}$$

Question 2

Consider $M = (K, \Sigma, \delta, s, H)$ where $K = \{s, y, n\}$, $\Sigma = \{a, \sqcup, \rhd\}$, $H = \{y, n\}$, and δ is defined as follows:

q	σ	$\delta(q, \sigma)$
s	a	(s, a)
s	$\sqcup$	$(s, \sqcup)$
s	$\rhd$	$(s, \rightarrow)$

On reading any string, M will stay in state s and will never halt. Hence M does not decide any language.

Question 3

Idea behind: the machine works by comparing w_1 with w_2 through w_n, then by comparing w_2 with w_3 through w_n, and so on.

1. Move right.
2. If the present symbol is a blank, *accept.* If the symbol is a #, mark it as $\overset{\bullet}{\#}$, and continue; otherwise, *reject.*
3. Scan right to the next # and mark it as $\overset{\bullet}{\#}$. If no # is encounted before a blank symbol, then only w_1 was present, so *accept.*
4. By zig-zagging, compare the two strings to the right of the marked #'s. If they are equal, *reject.*
5. Move the right marker to the next # symbol to the right. If no # symbol is encountered before a blank symbol, move the left marker to the next # to its right and the right marker to the # after that. This time, if no # is available for the right marker, all the strings have been compared, so *accept.* If both markers are successfully placed, goto 4.

$\rhd\underline{\sqcup}\#01\#11\#101\#111$

$\rhd \sqcup \underline{\overset{\bullet}{\#}}01\#11\#101\#111$

$\rhd\sqcup\ \overset{\bullet}{\#}\ 01\ \overset{\bullet}{\#}\ 11\#101\#111$ (then compare 01 and 11)

$\rhd\sqcup\ \overset{\bullet}{\#}\ 01\#11\ \overset{\bullet}{\#}\ 101\#111$ (then compare 01 and 101)

$\rhd\sqcup\ \overset{\bullet}{\#}\ 01\#11\#101\ \overset{\bullet}{\#}\ 111$ (then compare 01 and 111)

$\rhd \sqcup \#01\ \overset{\bullet}{\#}\ 11\ \overset{\bullet}{\#}\ 101\#111$ (then compare 11 and 101)

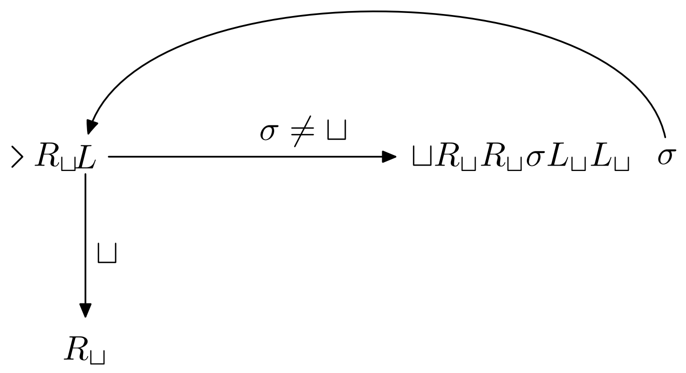
$\rhd \sqcup \#01\ \overset{\bullet}{\#}\ 11\#101\ \overset{\bullet}{\#}\ 111$ (then compare 11 and 111)

$\rhd \sqcup \#01\#11\ \overset{\bullet}{\#}\ 101\ \overset{\bullet}{\#}\ 111$ (then compare 101 and 111)

In the actual implementation, the machine has two different symbols, # and $\overset{\bullet}{\#}$, in its tape alphabet. Saying that the machine mark # means that the machine writes the symbol $\overset{\bullet}{\#}$ at that location. Unmarking it means that the machine writes back the symbol #.

Question 4

a)



$$\begin{tikzcd}
> R_\sqcup L \arrow[r, "\sigma \neq \sqcup"] \arrow[d, "\sqcup"] & \sqcup R_\sqcup R_\sqcup \sigma L_\sqcup L_\sqcup \quad \sigma \arrow[l, bend right] \\
R_\sqcup &
\end{tikzcd}$$

Turing machine for part a)

b) The idea is as follows:

1. Move the heads on both tapes to the right, copying each symbol on the first tape onto the second tape, until a blank is found on the first tape.

2. Move the head on the first tape to the right and the head on the second tape to the left, copying the symbol on the second tape onto the first tape, until a blank is found on the second tape.
3. Move the head on the first tape to the left until a blank is found.

Question 5

The proof is divided into two parts:

(only if) *If L is recursive, L and $\overline{L}$ are both recursively enumerable.*

It has already been proved in class that, if L is recursive, $\overline{L}$ is also recursive.

If L is recursive, L is recursively enumerable.

If $\overline{L}$ is recursive, $\overline{L}$ is recursively enumerable. Hence the result follows.

(if) *If L and $\overline{L}$ are both recursively enumerable, L is recursive.*

Since L and $\overline{L}$ are recursively enumerable, there exist Turing machines M_1 and M_2 that semidecides L and $\overline{L}$ respectively.

Consider a double-tape Turing machine M. First, it copies the input string w in the first tape to the second tape. Then, it simulates in parallel the operations of M_1 on the first tape and the operations of M_2 on the second tape. M halts when either M_1 or M_2 halts. Since M_1 halts if $w \in L$, and M_2 halts if $w \in \overline{L}$, so M must halt on any input string. Based on whether M_1 or M_2 halts, M can decide whether $w \in L$ or $w \in \overline{L}$. The result thus follows.

Question 6

(a) If L_1 and L_2 are recursively enumerable languages then there exists two Turing machines M_1 and M_2 that semidecide L_1 and L_2, respectively.

Let M be the 2-tape Turing machine that operates as follows:

1. Copy the input string w from the first tape to the second tape.
2. Simulate M_1 on the first tape and M_2 on the second tape *alternatively* (i.e. do one step of M_1 on first tape, then one step of M_2 on second tape, and so on). If M_1 or M_2 halts, then M halts.

M halts if and only if M_1 or M_2 halts on input w iff $w \in L_1$ or $w \in L_2$. Thus M *semidecides* $L_1 \cup L_2$. Therefore, the class of recursively enumerable languages in closed under union.

(b) If L_1 and L_2 are recursively enumerable languages then there exist two Turing machines M_1 and M_2 that semidecide L_1 and L_2, respectively.

Let M be the Turing machine that operates as follows:

1. Simulate M_1 on the string w.
2. If M_1 halts, then simulate M_2 on the string w.
3. If M_2 halts, then M halts.

M halts if and only if M_1 and M_2 both halt on input w. Thus M *semidecides* $L_1 \cap L_2$. Therefore, the class of recursively enumerable languages in closed under intersection.

Undecidable problems

Question 1

First note the following two lemmas. Let M be a Turing machine with $|K| = k$ and $|\Sigma| = t$.

Lemma 1 *If M uses only c tape cells, then it can have at most ckt^c distinct configurations.*

Proof: A configuration consists of (i) the state of the control, (ii) position of the head, and (iii) contents of the tape. (i) can be any of the k states, (ii) can be any of the c positions, and (iii) can be any of the t^c possible strings of tape symbols. The product of the three quantities is ckt^c.

Lemma 2 *If M uses only c tape cells, then M halts if and only if it does not repeat any configuration.*

Proof: ($\Rightarrow$) If M repeats a configuration, then it will repeat all subsequent configurations since a configuration completely specifies subsequent movements of M. Hence M will not halt. Therefore, if M halts, then no configuration is repeated.
($\Leftarrow$) If M does not halt and uses only c tape cells, then by Lemma 1 and the pigeonhole principle, it must repeat a configuration. Therefore, if no configuration is repeated, then M halts.

(a) The following Turing machine solves the problem.
M': input is "M" "w" "k"

1. Simulate M on input w like the UTM. In addition, introduce a new tape that stores all past configurations; initially, this tape is empty.
2. At each computation step,
   (i) if the length of the string in the current configuration $\geq k$, then output y.
   (ii) else if the current configuration is a halting configuration, output n.
   (iii) else if the current configuration has appeared before, output n.
   (iv) else add the current configuration to the list of past configurations.

(b) Suppose that the problem is solvable. Then, there exists a a Turing machine (algorithm) M_A that solves it. We will show that we can then use algorithm M_A to construct an Turing machine to solve the halting problem.

M_H: on input "M" "w"

1. Construct a Turing machine M^* that operates as follows:
   (a) Ignore the input (or erase it)
   (b) $i = 1$.
   (c) Write w on tape and simulate M on input w for i steps.
   (d) If M halts within i steps, then M^* also halts.
   (e) Else
      (i) Move the head i squares to the right of left-most square. (i.e. if M does not halt, force M^* to use more and more tape cells).
      (ii) Add 1 to i.
      (iii) Goto Step (c).

   (Note: M^* uses finite amount of tape on any input iff M halts on w)
2. Run M_A on input "M^*" "w'" where w' is any string.
3. If M_A outputs YES, output YES;
   if M_A outputs NO, output NO.

Correctness of Algorithm for Halting problem:
Case I: "M""w" $\in H$, i.e. M halts on input w.
Let's assume that M halts on input w in k steps. Then the simulation of M on input w will end in k steps, and M^* will also halt. This implies that M^* uses only a finite amount of tape, and so M_A will output YES and so will M_H.

Case II: "M""w" $\notin H$, i.e. M does not halt on input w.
Then M^* on any input will not halt and will perform the simulation of M to larger and larger values of i (that is, $i = 1, 2, 3, \ldots$). Since the tape head is moved i squares to the right after each simulation, the number of tape squares used by M^* will be unbounded. Thus, M_A will output NO, and so will M_H.

Question 2

(a) This problem is undecidable. Suppose it were solvable, then there exists some Turing machine M_A that solves it. It can be used to solve the halting problem:
M_H: on input "M""w"

1. Run M_A("M""w""h") where h is the halting state of M.
2. If M_A output y, M_H output y;
   if M_A output n, M_H output n.

(If there are more than one halting state, run M_A with the other halting state as input if the previous test of running M_A output n.)

(b) This problem is undecidable. Suppose it were solvable, then there exists some Turing machine M_B that solves it. Then, it can be used to solve the problem of determining whether an arbitrary Turing machine halts on the empty tape.
M_E: On input "M",

1. Let a be a symbol that is not in the alphabet of M. Construct a Turing machine M^* that is identical to M except that whenever it halts it also writes an a. (Clearly, M^* writes an a when started on the empty tape if and only if M halts when started on the empty tape.)
2. Run M_B("M^*""a").
3. If M_B output y, M_E output y;
   If M_B output n, M_E output n.

(c) This problem is solvable. The main idea is based on the following fact: If M has moved more than $|K|$ squares from the left, then one of the $\delta(q, \sqcup)$ transition rules must have been used at least twice (by pigeon-hole principle), thus M will be caught in an infinite loop, moving right infinitely.

Here is an algorithm that solves the problem. Start by simulating M on the empty tape and stop when M either writes a nonblank symbol, enters a configuration for the second time, or moves more than $|K|$ squares from the left end of the tape. If it has not written a nonblank symbol by this time, it never will. (If M has moved more than $|K|$ squares to the left, it will keep moving right without writing any nonblank symbol; if M has entered some configuration twice, it is caught in an infinite loop and keeps moving back and forth within the $|K|$ squares.)

(d) This problem is undecidable. Suppose it were solvable, then there exists some Turing machine M_D that solves it. Then, it can be used to solve the problem of determining whether an arbitrary Turing machine halts on the empty tape.
M_E: On input "M",

1. Construct a Turing machine M^* that erases its input and then simulates the action of M on the empty tape, halting if and only if M halts. (If M halts on e, then M^* halts on all strings and thus semidecides an infinite language. If M does not halt on e, then M^* halts on no input and thus semidecides the empty language, which is definitely finite.)
2. Run M_D("M^*").
3. If M_D output y, M_E output n;
   if M_D output n, M_E output y.