29/30

CMSC 651

Sandipan Dey

Homework 1

10/10

1. ($\Rightarrow$) Let's simulate a turing machine with the queue automata ~~any given step~~ transition of turing machine $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

Since the input tape is read only, ~~then~~ for queue automata we need to simulate the turing machine tape by the queue only.

a b c d e TM

R ↑ R

a $ e d c QA

(TM) Turing Machine

x | a | y

Left (read) already — current symbol, Head — Right

(QA) Queue Automata

Write (Enqueue) $x^R \$ y^R a$ Read (pop/Dequeue)

current Symbol to be Read

$\$ \in \Gamma$, special queue tape symbol (separating left & right of the TM to be simulated)

1. First push ~~all~~ the entire input string to the queue and terminate by \$. (For FIFO property of the queue the Read end will have the first symbol to be read)

   assume input has an end marker

2. To simulate $x q \underline{a} y \vdash_M x \underline{b} q' y$ (TM moves right after current tape symbol a is replaced by b) by QA, just pop the current symbol 'a' from right and push 'b' in the front.

3. To simulate $x x_1 q \underline{a} y \vdash_M x q'' \underline{x_1} b y$ (TM replaces a by b and moves left), QA needs to pop current symbol 'a' from rear, push 'b' in the front, by two circular shifts to the left (anti clockwise). For circular shifting to left, place a marker # at each end of q, replace each queue symbol x by (w, x) where w is the immediate left symbol (remember by extra state q_w), then dequeue (w, x) and enqueue (w, x) until x = #, then enque #, followed by w. Finally repeatedly dequeue (w, x) and enqueue w until # is poped.

($\Leftarrow$) Simulate QA with a 2-tape TM (equivalent to single tape TM). First tape contains the input string x, second tape the queue, we need to simulate the FIFO push/pop. Queue alphabet contains \$.

~~push~~ initialization of queue tape: Write a \$.

push: find the first blank space on the tape and write the symbol

pop: " " " symbol on tape ≠ \$, read the symbol, replace!

9/10

Let $L_1 = L(M_1)$

$L_2 = L(M_2)$

2. (b) Construct an NDTM M that ~~decides~~ recognizes concatenation of $L_1 L_2$

3.

$M(x)$ (M on input x)

1. for each possible way cut $x = x_1 x_2$ into two parts x_1, x_2
2. Run M_1 on x_1
3. Run M_2 on x_2. nondeterminism takes care of this
4. if both accepts, M accepts. o/w continue with next $x_1 x_2$
5. if all input cuts are tried without success, M rejects

this is a comment. You can't tell if M_1 or M_2 runs forever

(c) $L = L(M)$

Construct NDTM M^* that ~~decides~~ recognizes L^*

$M^*(x)$ not the same n

1. for each possible way cut $x = x_1 x_2 \dots x_n$ into n parts $n \in \mathbb{N}$
2. Run M on x_i $\forall i = 1, 2, \dots, n$
3. If M accepts on all of x_i, M^* accepts
4. o/w continue with next possible cut
5. if all cuts are tries without success, M^* rejects

what if M runs forever?

Since length of x is finite n will be finite.

8

(d) $L_1 = L(M_1)$

$L_2 = L(M_2)$

Construct TM M that ~~decides~~ recognizes $L_1 \cap L_2$

$M(x)$

1. Run M_1 on x, if M_1 rejects M rejects
2. Run M_2 on x, if M_2 rejects M rejects
3. if both M_1, M_2 accept, M accepts,

it's OK here if M_1 or M_2 runs forever

10

3. Let L be an infinite TR language

$\Leftrightarrow \exists$ enumerator E which enumerates all words $w \in L$.

Let's construct a new enumerator ~~M~~$'$ which simulates E.

TM ~~E~~ M'

M'~~E~~(x) Ignore input x — enumerators don't have input

Simulate E. $\forall$ enumerated word w do the following:

1. if w is the first enumerated word, remember it, print and proceed with next enumerated word. ~~smallest~~ largestword $\leftarrow w$.
2. else if $w \underset{\text{lexicographic}}{\leq}$ largestword (last remembered) then ignore w.

else if $w \underset{\text{lexicographic}}{>}$ largestword then

largestword $\leftarrow w$. (remember/cache it)

go to step 1 (proceed with next enumerated word).

if $L' = L(M')$, L' is still infinite since

$\forall w' \in L' : \exists w \in L : w' < w$

Also, since all $w' \in L$ are enumerated in lexicographic order, L' is decidable. $L' \subseteq L$ since the enumerator E for L and L' is infinite. (Proved)