Homework 8: Nitty Gritty Turing Machines

CSCI 161

Due 9pm, Saturday, November 14, 2019

Ben Dreyer

Overview. This week's homework concerns the mechanics of Turing machines discussed in Chapter 3. Submit a file containing all your solutions including a screenshot of your JFLAP Turing machine for Q2, but also submit a JFLAP file containing your Q2 Turing machine. Q2 will be auto-graded for correctness based on several test inputs, so you should confirm that it is formatted correctly and produces the correct output for strings in and out of the language. As usual, 5% extra credit for editing this file to generate your solutions file.

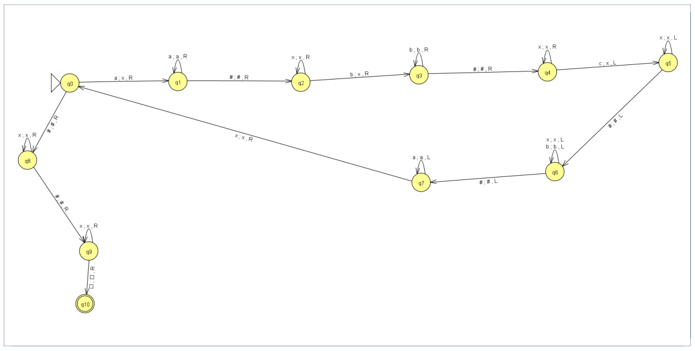
Question 1. Give pseudocode for a Turing machine that decides the language $L = \{a^n\#b^n\#c^n \mid n \geq 0\}$.

Solution.

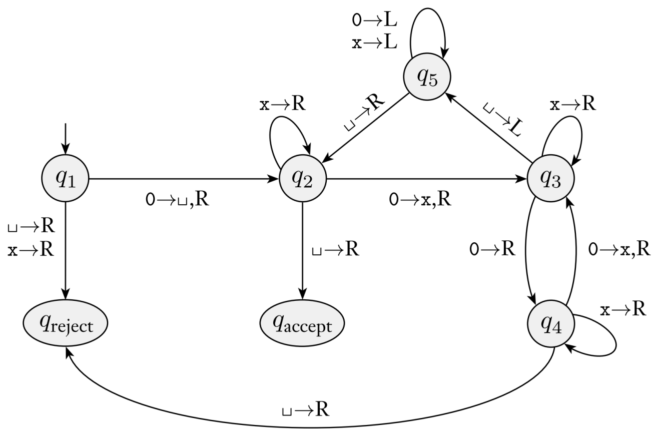
```
1. For each unmarked a
        mark off one a as x
        move right until the pound sign is read
        move right until first unmakred b
        mark the first unmarked b as x
        move right until the pound sign is read
        move right until the first unmarked c
        mark that c as x
        move left all the way until unmarked a
2. Accept if only marked off x's and  remain
```

Question 2. Give a JFLAP Turing machine that corresponds to your algorithm for Q1.

Diagram.



Question 3. The machine below (Example 3.7) recognizes any string of 0s whose length is a power of two.



a) Give the configuration of the machine on input $0^{(2^3)}$ immediately after its <u>second</u> q_2 to q_3 transition.

xxq_3x0x0x

b) Give the reject configuration of the machine on input 0^7.

$␣x0x0x0␣qreject$

c) Give the reject configuration of the machine on input 0^6.

$␣xxx0x␣qreject$

Question 4. At the end of Unit 2, we remarked that nondeterministic PDAs are more powerful than deterministic ones without going into detail why. A deterministic PDA can recognize the language $\{w\#w^{\mathcal{R}} \mid w \in \{0,1\}^*\}$ but only nondeterministic PDAs can recognize $\{ww^{\mathcal{R}} \mid w \in \{0,1\}^*\}$. (Think about why!) Of course even nondeterministic PDAs cannot recognize $\{ww \mid w \in \{0,1\}^*\}$ since we saw that this language fails the conditions of the context-free pumping lemma, but enhanced memory power allows a Turing machine to recognize this language. Although nondeterminism can be simulated (often inefficiently!) on a Turing machine (Section 3.3), Turing machines themselves are deterministic. For this question, you will explore the idea of determinism on Turing machines in the context of two context-sensitive languages:

$$L_1 = \{w\#w \mid w \in \{0,1\}^*\} \text{ and } L_2 = \{ww \mid w \in \{0,1\}^*\}.$$

Document your exploration by answering the following questions. All of your answers can be high-level:

a) Give high-level pseudocode for a Turing machine that recognizes L_1 in quadratic $O(|w|^2)$ time.

b) Describe how you could modify this algorithm *nondeterministically* to recognize L_2 in quadratic time. How long would a deterministic simulation of this nondeterministic algorithm take? (A deterministic simulation of a nondeterministic process considers all possible computation paths in sequence.)

c) Describe a more clever Turing machine that deterministically recognizes L_2 in quadratic time.

Solution. Part A

1. For each char in w, left of the pound sign

```
    (a) if char == 0
            mark off 0 as x
            move right until pound sign
            move past all x
            if char == 0 mark as x,
            else reject
    (b) if char == 1
            mark off 1 as x
            move right until pound sign
            move past all x
            if char == 1 mark as x
            else reject
2. Move past x's left of pound until pound sign
3. Move past pound sign
4. Move past x's
5. if char == ␣
   Accept
   Else: reject
```

Part B

You could modify this algorithm in a non deterministic way to accept L2 by doing the following: inside of the for loop, instead of the code after moving right until pound, say move right until the respective 0 or 1 is found. This however is a non deterministic find, so whenever a matching character is found, two branches spawn, one where those two matches are marked as x's, and one where the head keeps moving right until it finds the next match. This will ensure that if the string is valid, eventually there will be a branch that matches each character correctly.

Part C

A more clever Turing Machine could recognize L2 deterministically by finding the midpoint of the string, and checking for matching characters in a similar way we did for L1. We can find the midpoint of the string by converting the outsides of the string to a Y, then moving in one character and convert to an X, then move in and convert to a Y etc... When you are left with a X surrounded by two Y's or a Y surrounded by two X's or two consecutive Y's or X's, you have found the midpoint. EX: YXY(X)YXY, YXY(Y)XY, YX(Y)XY. All that's left is to match the characters on opposite sides of the midpoint.