Midterm Exam 2: Study Guide

CSE 30151 Spring 2016

2016/04/07

You will have the whole class period of 75 minutes. The exam will be open book and open paper notes. No computers, smartphones, or any other Turing-equivalent machines are allowed. Regrettably, I can't think of any way to allow the use of notes taken on an electronic tablet that is fair to all students.

The exam covers everything up to and including HW7. It does not cover any of these special topics: neural networks and finite automata, human language and context-free grammars, human intelligence and Turing machines. There will be six questions, worth 10 points each, for a total of 60 points (10% of your grade), on the following topics:

- Design: One of each of the following types of questions.
  - Write a pushdown automaton or context-free grammar (your choice) that recognizes a given language (like 2.4ad, 2.6ac, 2.7ac). If you write a PDA, either a state transition diagram or a table is acceptable; an informal description gets partial credit.
  - Write a Turing machine that decides a given language (like 3.8a). An implementation-level description is acceptable.
- Proofs: One of the following types of questions.
  - Show that a certain language is not regular (like 1.29ac or 1.46b), or
  - Show that a certain language is not context-free (like 2.30bc), or
  - Show that context-free languages are closed or not closed under a certain operation (like 2.38, but not as hard).
- You'll be given a definition of an extension of NFAs, PDAs or TMs (like Problem 3.10 or 3.14) and will be asked:
  - Write a machine of this type that recognizes a given language.
  - Show that a machine of this type can be converted into an equivalent Turing machine (implementation-level description). You'll be given a skeleton proof that you will fill in the missing parts of.
  - Show that a Turing machine can be converted into an equivalent machine of this type (only an informal description required). You'll be given a skeleton proof that you will fill in the missing parts of.

Sample question: queue automata

See Problem 3.14 for a description of a queue automaton. On the exam, you will be given a formal definition to refer to, although you won't necessarily need it: A *queue automaton* (QA) is a tuple $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where

- Q, Σ, Γ, q_0, and F are as in a pushdown automaton
- δ is a transition function $\delta : Q \times \Sigma^? \times \Gamma^? \to Q \times \Gamma^?$, where $X^? = X \cup \{\varepsilon\}$.

The queue is initially empty. If the current state is q, with remaining input aw (where $a \in \Sigma^?$ and $w \in \Sigma^*$) and queue bx (where $b \in \Gamma^?$ and $x \in \Gamma^*$) and $\delta(q, a, b)$ contains (q', b'), then the QA can move to state q', with remaining input w and queue xb'.

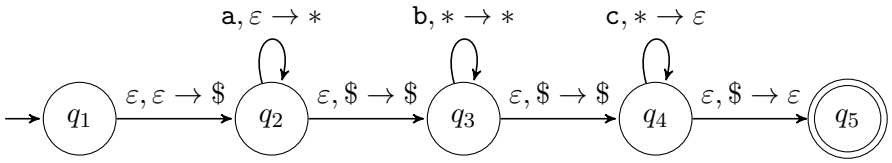
Problems

1. Design a QA that recognizes the language $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 0\}$.
   (a) Give a brief informal description of your QA.
   (b) Write the formal description (state transition diagram or table) of your QA.
2. Show that any QA P can be converted into an equivalent TM M. You may use any of the TM extensions that were discussed in the book or in class. For each of the following, give a brief implementation-level description. Assume that w is the input string.
   (a) If P has read the first k input symbols and the stack is x, how would you represent this in M? How would you initialize M?
   (b) How would you simulate reading in an input symbol a?
   (c) How would you simulate pulling (dequeueing) a symbol b?
   (d) How would you simulate pushing (enqueueing) a symbol c?
3. Show that any TM M can be converted into an equivalent QA P. For each of the following, give a brief informal description. (This is harder than the exam question.)
   (a) Suppose that M's tape is $t_1 t_2 \cdots t_n \sqcup\sqcup\sqcup\cdots$, where each $t_i \in \Gamma$, and the head is at position h. How would you represent this using a queue? How would you initialize the queue?
   (b) Now consider a single one of M's transitions, $\delta(q, a) = (r, b, d)$, where $d \in \{\mathrm{L}, \mathrm{R}\}$. How would you simulate reading symbol a and writing symbol b?
   (c) How would you simulate moving the head left? Remember that the head cannot move past the left end of the tape.
   (d) How would you simulate moving the head right? Remember that the tape has blank symbols ($\sqcup$) extending infinitely to the right.

Solutions

1. (a) We will store n occurrences of $*$ on the queue to ensure that all three letters have n occurrences. Initially the machine pushes a \$; then it reads in the `a`'s while pushing an equal number of $*$'s onto the queue. Then, it reads in the `b`'s while pulling $*$'s from the queue and pushing them back in. When it pulls a \$, it pushes it back in. Then it reads in the `c`'s while pulling $*$'s. When it pulls a \$, it stops reading input symbols.

   (b) State transition diagram:



2. (a) We can use a two-tape Turing machine. Tape 1 will hold w, with the head at position $k+1$, and tape 2 will hold x. To initialize, we don't have to do anything – tape 1 already has w and tape 2 is already empty.

   (b) Just read the symbol under head 1 and move it one square to the right.

   (c) Delete the first symbol on tape 2 and shift everything else one square to the left.

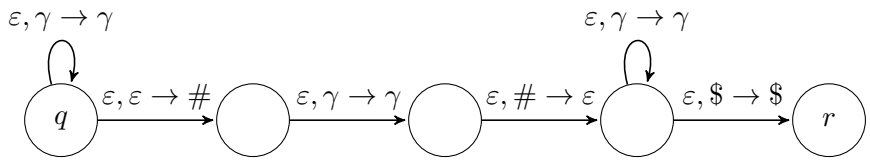
   (d) Scan head 2 to the right until a blank symbol is reached; write the pushed symbol on that square.

3. The exam question will be easier than this one, I promise.

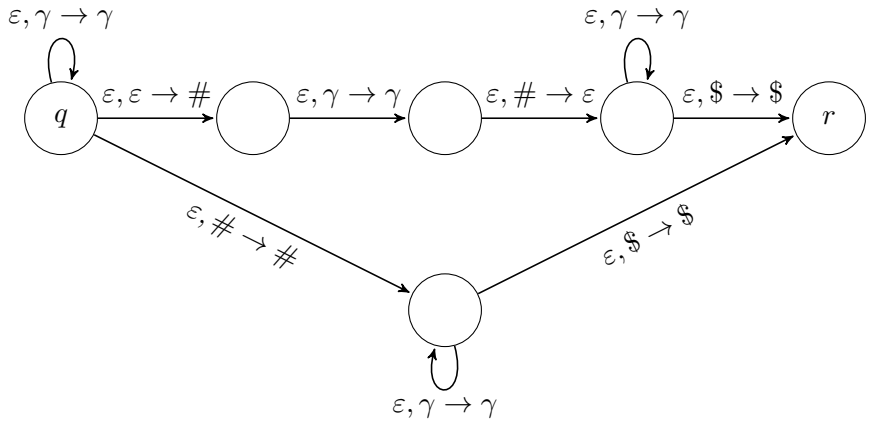
   (a) The queue will contain $t_1 \cdots t_{h-1} \# t_h \cdots t_n \$$, where # marks the head position. To initialize, push #, then read in the whole input string and push it into the queue, then push \$.

   (b) Cycle through the queue and change the current symbol (the one after #) from a to b. That is, pull symbols and push them back until # is reached; pull a and push b; then pull symbols and push them back until \$ is reached.

   (c) Cycle through the queue and move the # left by one position. That is, pull some symbols and push them back; push #, pull one symbol and push it; pull #; continue pulling and pushing until \$ is reached. Because of nondeterminism, this will succeed only if the # is moved one position left. This is more clear if we draw a transition diagram:



In this diagram, a transition on $\varepsilon, \gamma \to \gamma$ stands for multiple transitions, one for each tape symbol. To handle the left end of the tape, we'd add another path that checks for # at the very beginning and pulls/pushes symbols without any changes.



(d) Cycle through the queue and move the # right by one position. That is, pull symbols and push them back until we pull #; pull one symbol and push it; push #; continue pulling and pushing until $ is reached. To handle the right end of the tape, if $ immediately follows the new position of #, insert a blank.