CS 321H Homework 5

Lyell Read

Submit to Canvas a pdf file containing verbal explanations and transition graphs for the Turing machines in problems 1 & 2 and the written answers to problems 3 & 4. Also submit JFLAP .jff files (named youronidnameP1a, youronidnameP1b, etc.) for problems 1 & 2.

1. (10 pts) Design single-tape Turing machines that accept the following languages using JFLAP

- a. $L_2 = \{w : n_a(w) = n_b(w) : w \in \{a, b\}^+\}$.

This Turing Machine is quite simple. It performes a search for pairs of a and b (to make sure that $n_a(w) = n_b(w)$), each time replacing them with x. Should it not find either, it either halts at the trap state or at the state $q2$.

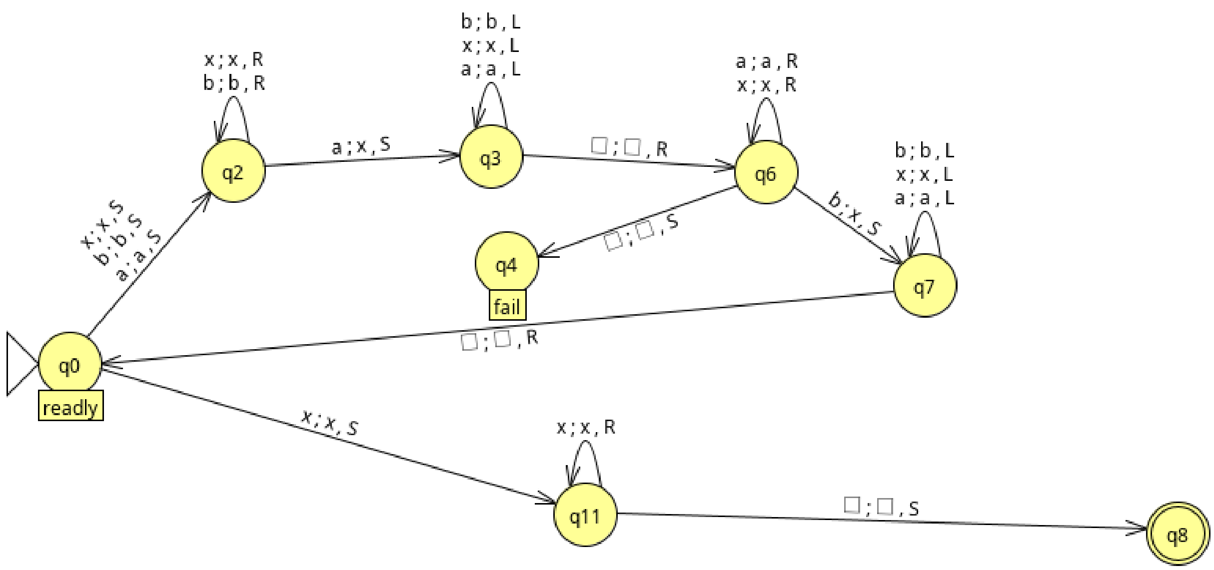

Figure 1: Turing Machine for 1a

- b. $L_3 = \{ww : w \in \{a,b\}^+\}$.

By far the most complicated turing machine I have composed yet, this turing machine is composed of three parts, each with their start state labeld. The first function is the center finding function, which uses an internal alphabet (mapped as $a \to x$, $b \to y$) in order to find the center of the word. The second part of the machine, the middle symbol inserter, is responsible for inserting an m between the two w. This requires the entirety of the second word after that point to be shifted right, so it also takes care of that operation, essentially leaving the word $\{wmw : w \in \{a,b\}^+\}$ on the tape. After that, pairs are parsed starting with the first pair off either end (where one end is the m character, and one is the left end of the string on tape), replacing characters with m as they are identified, in order to keep updating where to start the checks on the left edge and middle of the word.

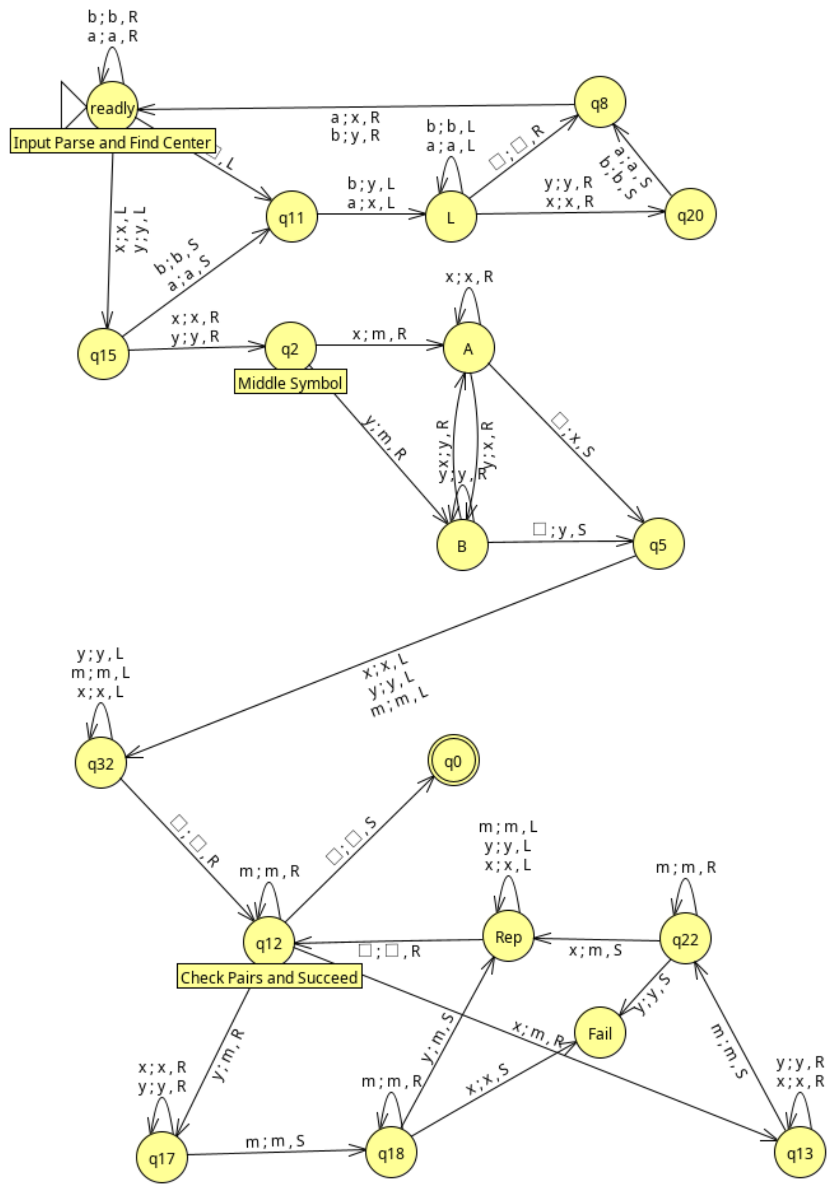

Figure 2: Turing Machine for 1b

2. (10 pts) Design Turing Machines using JFLAP to compute the following functions for x and y positive integers represented in unary. The value f(x) represented in unary should be on the tape surrounded by blanks after the calculation.

- a. $f(x) = \begin{cases} x - y & x > y \\ 0 & otherwise \end{cases}$

This machine makes use of the 0 character to zero out subtracted unary bits. If there are more 1's to be removed than we have, we run off the left end of the tape, and remove all characters from the tape except the $-$ which gets converted to 0, leaving a Zero. The alternative, where the calculation resulted in a positive unary number is cleaned up by removing all characters but the 1

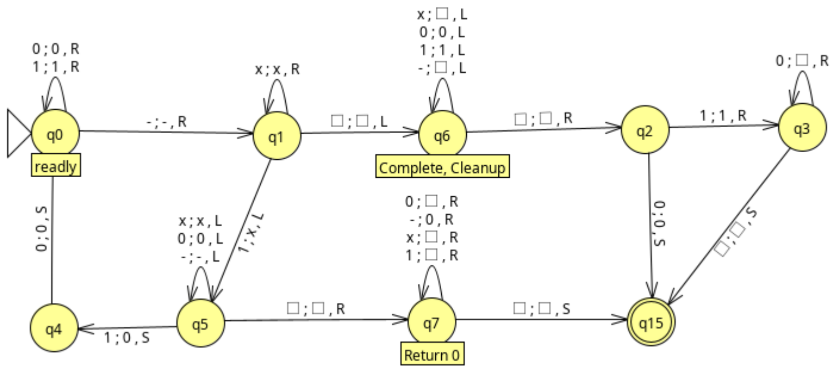

Figure 3: Turing Machine for 2a

- b. $f(x) = x \mod 5$

This machine operates in sets of 5. Should it encounter a number of 1s less than 5, it terminates. If it encounters exactly 5, it returns zero, and if it encounters more than 5, it clears out all the previous 1 characters (5 of them) and repeats.

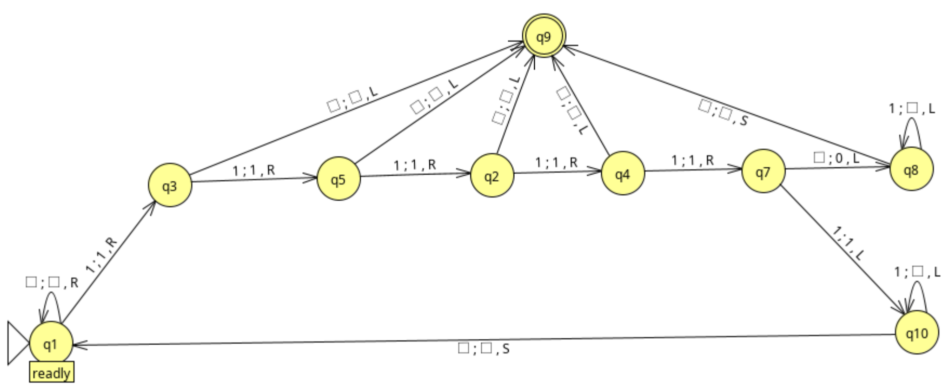

Figure 4: Turing Machine for 2b

3. (5 pts) The nor of two languages is defined below, prove that recursive languages are closed under the *nor* operation.:

$$nor(L_1, L_2) = \{w : w \notin L_1 \text{ } and \text{ } w \notin L_2\}.$$

Given that Recursive languages are closed under complementation and intersection, we can assert that recursive languages are closed under the *nor* operator if we can construct it based on other operators under which recursive languages are closed.

Therefore, we can rewrite the $nor(L_1, L_2)$ operator as $nor_1(L_1, L_2) = \{w : w \in L_1^C \cap L_2^C\}$. Given that nor_1 uses only set operations under which recursive languages are closed, we can conclude that recursive languages are closed under the *nor* operator.

4. (5 pts) Suppose we make the requirement that a Turing machine can only halt in a final state, that is, we require that $\delta(q, a)$ be defined for all pairs (q, a) with $q \notin F$ *and* $a \notin \Gamma$. Does this restrict the power of the Turing machine? Prove your answer.

This constraint would not impact the power of the turing machine. We can modify an existing turing machine to meet the requirement of halting at a final state without impacting it's power By doing the following:

Define a new final state q_{halt} which includes a transition to it from every other state $q_x \in Q$ for every character in Γ that does not already have a transition out of state q_x to another state.

This will create a turing machine that has the same power as the original that will always halt in a final state, either $q_f \in F$ *or* q_{halt}.