CS 1511 Homework 6

Mathew Varughese, Justin Kramer, Zach Smith

Monday, Feb 11

12. (a) A finite state machine could be constructed such that its language is the negation of the language of M. This can be done as it was in 1502. The accept states turn into reject states and vice versa.

12. (b) This can be done by creating a DFA (P) that has a language which the intersection of the language of L and the language of N. This can be done by making the states of P be the cross product of the states from L and the states of N. Then, the accept states are those states that have both accept states from L and N. The transition function would be determined by going to the state (l,n). l is what the transition function would do from DFA L and n is where the transition function for DFA N. The start state would be the the state that contains both of the start states.

12. (c) The states in Q will be the power set of the states of P. Say that the characters in the alphabet for P were 4 bits (for x, y, z, w). This can be assumed as we said they were properly encoded. Now, the characters in the alphabet will be 3 bits long (x, y, z). The transition function will change so that it will change to the state that contains the states that P would go to if w was a 0 or 1. This explanation works better with an example. Say in a DFA state A goes to B when w=0 and A goes to C when w=1. In this new DFA, the set of states will be the power set of $\{A, B, C\}$. So the states would be $\{\{A\}, \{B\}, \{C\}, \{A, B\}, \{A, C\}, \{B, C\}, \{A, B, C\}\}$. Since it is an existential, we want to consider all possibilities for w. This means that from state A, the next state (given the same x, y, and z) would be $\{B, C\}$. This would obviously get more complicated, but it is definitely possible.

12. (d) This will be done in a similar way to problem c. $\forall x P(x)$ is logically equivalent to $\neg\exists\neg P(x)$. So, $\forall z \exists w\ (x+y = z) \wedge \neg(y+z = w+x)$ is logically equivalent to $\neg\exists\ z(\neg\exists w\ (x+y = z) \wedge \neg(y + z = w + x))$ To construct this, we first create a DFA that has a language which is the opposite of Q. Then, following the same idea in Part C, create a DFA that has states which are the power set of Q. Then, take that DFA and construct a new one that contains the complement of that ones's language.

12. (e) The same idea can be done for machine S.

12. (f) This problem is the same as asking whether $L(S)$ is non empty. The language DFA is empty if there are no accept states, or if the accept states are not reachable from the start state. The algorithm for determining if $L(S)$ is not empty would be as follows:

1. Mark the start state of S
2. Repeat until no new states are marked:
3. Mark any state that has a transition coming into from a state that is marked
4. If no accept state is marked reject otherwise accept.

12. (g) Take the first order and break each piece between an and into a DFA of proper encoding. Then, continually construct each DFA from each "or" or "and" or "negation" by using the same techniques that were done in previous problems. Then from the inside out, construct new DFAs based on the quantifiers. If it is existential, then use the method in 12 c and if it is a universal, then use the method in 12 d. After this finite automata is constructed, call it S and use the algorithm in part f to determine if it accepts any string. If it does, then it is true. Otherwise it is false.

13. (a)

For this to be in $TIME(n^2)$ it would need to take steps proportional to the square of $|I|$. Saying $a = 100 * (b_I)^2$ doesn't work because it would depend on b_I making it not a constant. b_I is dependent on I, so that would mean a would change.. You need to show that the Turing Machine stops in $a|I|^c$ steps, so a needs to be a constant. In this case it is not a constant.

13. (b)

What we need to prove here is that with our Turing Machine S that it will take at most $dn^2 2^{2n}$ steps. In this case, our n would be equal to the length our input I. So we are saying that our Turing Machine S has input I and a string version of our Turing Machine and input b_I (with some constant c) such that $cI^2(b_I)^2 \leq dn^2 2^{2n}$. Given a string of length $|I|$, the maximum possible number of possibilities would be $2^{|I|}$, or 2^n. This means at maximum b_I can only be 2^n. Therefore, after substituting 2^n for b_I, we get the expression $cn^2(2^n)^2 \leq dn^2 2^{2n}$. This makes it clear that $cI^2(b_I)^2 \leq cn^2(2^n)^2 \leq dn^2 2^{2n}$

13. (c)

i.

L' is not in TIME(n) because the language is defined as a collection of strings that a Turing machine can run and it will not accept within $b|I|$ steps. This proof can be shown by the diagonalization we did in class. This diagonalization takes on the column the input I, and the rows it takes the string representation of a possible constant a Turing machine. Then when you have where these combinations of accept you put a 1 and 0 when you reject. When you take the diagonal and switch all 1's to 0's and all 0's to 1's, you get something that is not in TIME(n).

ii.

L' is in $TIME(n^c)$ for, at the smallest, a constant c = 4. This is because in unary, our

encoding of b_I will now be able to store n numbers with n digits. In this case, our formula for the least amount of steps will be (with a constant d) $dI^2(b_I)^2$. In this case, I^2 will be n^2 and $(b_I)^2$ will also be n^2, so the overall algorithm will take at minimum n^4 steps.