

# 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 is 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 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 one'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  $a = 100 * (b_I)^2$  just changes the constant. This constant could be any positive integer, so while this may work for the constant 100, it may fail with another constant that you have not tested.

### 13. (b)

If S can run this simulation in less than  $a = 100 * (b_I)^2$  steps, then S can run this simulation in any amount of steps larger than  $a = 100 * (b_I)^2$  steps. If L is in  $TIME(n^2 2^{2n})$ , then there exists a constant such that S runs in  $dn^2 2^{2n}$  due to the Turing Machine S being able to increase its speed by a constant factor through the packing together tape symbols of the original tape symbols. Thus, this d may be equal to 100, and  $100(b_I)^2 2^{2(b_I)}$  will be a larger amount of steps than  $100 * (b_I)^2$  steps. This means that S will always be able to run in less than  $100(b_I)^2 2^{2(b_I)}$  steps, so L is in  $TIME(n^2 2^{2n})$ .