Theory of Computation: Assignment 4

Arjun Chandrasekhar

Due Thursday, 02/17/2022 at 11:59 pm (50 points)

1. The following problems are taken from Sipser 1.6c-d. Give a regular expression describing each of the following four languages. In each case, the alphabet is $\Sigma = \{0, 1\}$

   - (5 points) $L = \{w | w$ contains the substring $0101\}$
   - (5 points) $L = \{w | w$ has length at least 3 and its third symbol is $0\}$

2. (10 points) The following problem is taken from Sipser 1.28a. Create an NFA to recognize the regular expression $a(abb)^* \cup b$. For full credit, you must use recursive procedure we described in classs. You should show state diagrams for all of the smaller NFAs, and show how you combined them into bigger NFAs.

3. (10 points) This problem is taken from Sipser 1.22. In certain programming languages, comments appear between delimiters such as /# and #/.Let C be the language of all valid delimited comment strings. A member of C must begin with /# and end with #/ but have no intervening #/.For simplicity, assume that the alphabet for C is $\Sigma = \{a, b, /, \#\}$.

   First, design a 5-state NFA to recognize C. Then, Use the procedure described in class to convert your NFA from part A into a regex. For full credit, you must give a diagram of the GNFA at each step in the process of ripping away states. You may omit $\emptyset$ transitions when you draw your GNFA.

4. (10 points) This problem is taken from problem 1.31 in Sipser. Recall for a string $w = w_1 w_s \dots w_n$, the **reverse** of w is $w^R = w_n \dots w_2 w_1$.

   For a language L, the reverse of L is

   $$L^R = \{w^R | w \in L\}$$

   As an example, suppose $L = \{john, paul, george, ringo, racecar\}$. Then $L^R = \{nhoj, luap, egroeg, ognir, racecar\}$

   Prove that if L is regular, then L^R is regular. You may use any technique discussed in class, although you may find it more convenient to work with regular expressions.

   Hint: If you are using regular expressions, follow the inductive proof blueprint discussed in class. If you prefer to design a state machine, try designing an NFA that mimics going through the original DFA in reverse: it should start at one of the original accept states, follow the original transitions backwards, and finish in the original start state.

5. (10 points) This problem is taken from 1.43 in Sipser. Let L be a formal language. Define DROP-OUT(L) operation as follows:

   $$\text{DROP-OUT}(L) = \{xz | xyz \in A \text{ for some } y \in \Sigma\}$$

   In other words, it is the set of all strings that can be obtained by taking a string from L and "dropping out" one symbol.

   As an example, suppose $L = \{001, 1010\}$. Then here are all of the "dropouts" we could perform:

- $0\mathbf{0}1 \to 01$
- $0\mathbf{0}1 \to 01$
- $00\mathbf{1} \to 00$
- $\mathbf{1}010 \to 010$
- $1\mathbf{0}10 \to 110$
- $10\mathbf{1}0 \to 100$
- $101\mathbf{0} \to 101$

Thus, DROP-OUT$(L) = \{01, 00, 010, 110, 100, 101\}$.

Prove that if L is regular, then DROP-OUT(L) is regular. You may use any of the techniques discussed in class, although you may find it more convenient to work with regular expressions.

Hint: If you are using regular expressions, follow the inductive proof blueprint discussed in class. If you prefer to design a state machine, try designing an NFA that reading going through the original DFA, but getting one chance to "skip" a character. Make "before" and "after" copies of the original DFA; add ϵ transitions to allow the NFA to move from one copy to the other, and "guess" where the missing character occurs.