1. Take a preview of Chapter 2.

Please note that your reading assignment is an integral part of your homework. You should start working on the problems as soon as possible, don’t wait till the last minute.

1. 1.20.

2. Prove that the following regular expressions are equivalent (i.e., they represent the same set of strings.)

\[ 1 + 0(\epsilon + 00)^*(1 + 01) \text{ and } 0^*1. \]

Express in English what is the set of strings represented by these two equivalent regular expressions.

(In the following I will not emphasize with bold face for regular expressions.)

3. Describe in English the set of strings denoted by the following regular expressions

\[ (00 + 1)^*(11 + 0)^* \]
\[ (1 + 01)^*(\epsilon + 0 + 00) \]
\[ [00 + 11 + (01 + 10)(00 + 11)^*(01 + 10)]^* \]

4. Construct finite automata equivalent to the following regular expressions

\[ ((0 + 1)(0 + 1))^* + ((0 + 1)(0 + 1)(0 + 1))^* \]

5. Write a regular expression equivalent to the following:

The set of all strings in which every pair of adjacent 0’s appears before any pair of adjacent 1’s.

Also:

The set of all strings with at most one pair of consecutive 0’s and at most one pair of consecutive 1’s.

6. Prove the following for regular expressions: (Here \( r = s \) for two regular expressions, strictly speaking, means the language they denote are the same, namely \( L(r) = L(s) \).)
To show two sets of strings are the same, you should show that every string that is in one set is also in the other.

a) \((rs)t = r(st)\)
b) \(r(s + t) = rs + rt\)

c) \((r^*)^* = r^*\)

7. Prove or disprove the following for regular expressions:

a) \((rs + r)^* r = r(sr + r)^*\)
b) \((r + s)^* = r^* + s^*\)
c) \(s(rs + s)^* r = rr^* s(rr^* s)^*\)

8. (extra credit)

Let’s define the notion of a quotient language as follows: Given two languages \(L_1\) and \(L_2\), both over the same alphabet set \(\Sigma\). Then the quotient \(L_1 / L_2 = \{ x \in \Sigma^* \mid \text{there exists } y \in L_2, \text{ such that } xy \in L_1 \} \).

a) Prove that: If \(L_1\) and \(L_2\) are both regular, then so is the quotient \(L_1 / L_2\).

You should prove this by construction, namely, start with some finitary representation such as DFA, for \(L_1\) and \(L_2\) respectively, construct another finitary representation such as DFA or NFA, which accepts \(L_1 / L_2\).

b) Prove or disprove: If \(L_1\) is regular, and \(L_2\) is an arbitrary language, both over \(\Sigma\), then \(L_1 / L_2\) is always regular.

(To prove, you need to show it is so for all such \(L_1\) and \(L_2\). To disprove, you need to give one counter example. However in either cases, a rigorous proof is needed to earn full credit.)

9. 1.21. part b.

(You may use either procedure, from the book or from the class.)

10. Show that the following language is not regular:

\[ L = \{a^i \mid i \geq 1 \} \]

11. (extra credit) Suppose \(L\) is a regular language. Are the following languages regular or not? prove your answers.

a) \(\{a_1a_3a_5\ldots a_{2n-1} \mid a_1a_2a_3\ldots a_{2n} \in L\}\).

Here \(a_i\) are symbols from \(\Sigma\).
b) \[ \{ x_1x_2 | x_2x_1 \in L \}. \]

Here \( x_i \) are arbitrary strings \( \Sigma^* \).

12. **(extra credit)** This is a problem using dynamic programming as an algorithmic strategy.

Suppose \( M_1, M_2, \ldots, M_n \) are \( n \) rectangular matrices, of dimensions \( k_0 \times k_1, k_1 \times k_2, \ldots, k_{n-1} \times k_n \) respectively. Suppose multiplying two matrices \( A_{n \times m} \) and \( B_{m \times \ell} \) takes \( nm\ell \) arithmetic operations.

It is well known that matrix product is associative, namely if \( A_{a \times b}, B_{b \times c} \) and \( C_{c \times d} \) are three matrices, then \( A \cdot (B \cdot C) = (A \cdot B) \cdot C \). However the first way takes \( bcd + abd \) arithmetic operations, while the second way takes \( abc + acd \) arithmetic operations.

Devise an algorithm that finds the order to multiply \( M_1 \cdot M_2 \cdot \ldots \cdot M_n \) (in some associative order) that requires the minimum number of total arithmetic operations. (Your algorithm should not actually multiply the matrices \( M_1, M_2, \ldots, M_n \) (as they are not even given). The input to the algorithm is the list of their dimensions \( k_0, k_1, k_2, \ldots, k_{n-1}, k_n \). The output of the algorithm is an associative order to multiply them.)