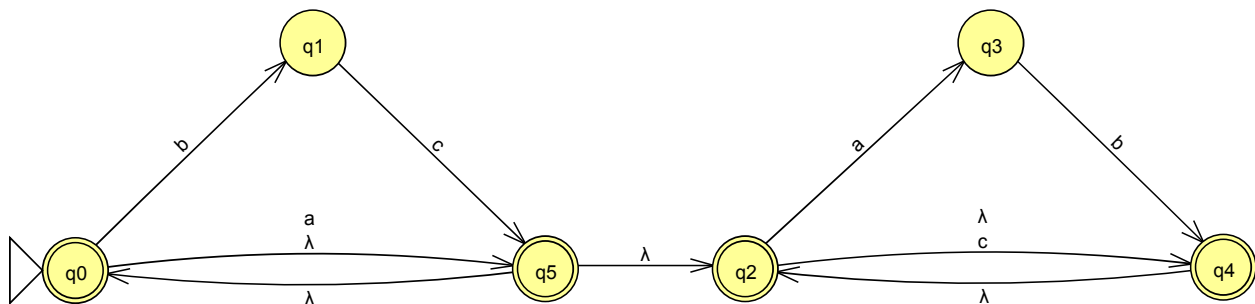


**1. Nondeterministic Finite Automata** Prove that every NFA can be converted into an equivalent NFA that has a single accept state. This proof should be very short, no more than 3 sentences.

Every NFA can be converted into an equivalent NFA that has a single accept state. Simply add a new final state to the original automaton, add epsilon transitions from every old final state to the new final state, and change every old final state into a regular state. This new NFA accepts exactly the same language as the original NFA. ■

**2. Regular Expression to NFA** Convert the regular expression  $(a + bc)^*(ab + c)^*$  into an NFA using the procedure described in class. Build the NFA in JFLAP.

The following NFA accepts the regular expression  $(a + bc)^*(ab + c)^*$ :



**3. Closure of Regular Languages** If  $L_1$  and  $L_2$  are languages, define a new language

$$\text{INTERLACE}(L_1, L_2) = \{w_1v_1w_2v_2 \dots w_nv_n \mid w_1w_2 \dots w_n \in L_1, v_1v_2 \dots v_n \in L_2\}.$$

For example, if  $abc \in L_1$  and  $123 \in L_2$ , then  $a1b2c3 \in \text{INTERLACE}(L_1, L_2)$ . Notice that the interlace operation always combines strings of equal length, so that if  $L_1 = \{ac, abc\}$  and  $L_2 = \{123, 45\}$ , then  $\text{INTERLACE}(L_1, L_2) = \{a1b2c3, a4c5\}$ .

Show that if  $L_1$  and  $L_2$  are regular languages, then  $\text{INTERLACE}(L_1, L_2)$  is a regular language.

The proof of this closure property is really similar to the proof for the union operation in p. 45-47 of Sipser.

First, Let  $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$  and  $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$  be DFAs such that  $L(M_1) = L_1$  and  $L(M_2) = L_2$ .

The idea here is to simulate both machines  $M_1$  and  $M_2$  alternatively as we read each input symbol. To do this, we need to keep track of which state we are in for both  $M_1$  and  $M_2$ , and also keep track of which machine we are currently simulating. Therefore, you need to remember a pair of states plus an additional flag for which machine we are currently in. This can be done using the cartesian product of  $Q_1, Q_2$ , and  $\{1, 2\}$ . Here, we've added the new flags "1" and "2" which keeps track of whether we're in  $M_1$  or  $M_2$ .

Now, to simulate both machines correctly, suppose that right now it's machine  $k$ 's turn to get the next symbol. We then simulate that symbol on the current machine  $k$ , update the state, and switch to the other machine. When we've finished reading all the symbols, and both machines are in final states, then we can accept the input!

Formally, we construct a new DFA  $M = (Q, \Sigma, \delta, s, F)$  as follows:

- $Q = Q_1 \times Q_2 \times \{1, 2\}$
- $\Sigma = \Sigma$
- $s = (s_1, s_2, 1)$
- $F = \{(q_1, q_2, 2) \mid q_1 \in F_1, q_2 \in F_2\}$
- $\delta((q_1, q_2, k), a) = \begin{cases} (\delta_1(q_1, a), q_2, 2) & \text{when } k = 1 \\ (q_1, \delta_2(q_2, a), 1) & \text{when } k = 2. \end{cases}$

We see that  $M$  accepts  $\text{INTERLACE}(L_1, L_2)$  just as our discussion explains above. ■

#### 4. More Regular Language Closure

1. If  $L$  is a language over the alphabet  $\Sigma$  and  $a \in \Sigma$ , define the quotient  $L/a = \{w \mid wa \in L\}$ . For example, if  $L = \{a, aab, baa\}$ , then  $L/a = \{\varepsilon, ba\}$ . Prove that if  $L$  is regular, then so is  $L/a$ .
2. If  $L$  is a language over the alphabet  $\Sigma$  and  $a \in \Sigma$ , define  $a \setminus L = \{w \mid aw \in L\}$ . Prove that if  $L$  is regular, then so is  $a \setminus L$ .

1. Since  $L$  is regular, there exists a DFA  $M = (Q, \Sigma, \delta, s, F)$  such that  $L(M) = L$ . We build a new automaton  $M' = (Q, \Sigma, \delta, s, F')$  with the same set of states, transition function, and start state as the original  $M$ . The only difference is the set of accepting states, which is defined as

$$F' = \{q \mid \delta(q, a) \in F\}.$$

The language of the new automaton is  $L(M') = L/a$ , because

“for every input  $w$  that results in a accepting state  $q \in F'$ , then  $wa \in L$ , because reading an additional  $a$  transitions to a state in  $F$ , i.e. an accepting state in the original machine.”

So,  $L/a$  is also regular.

2. Given a language  $L$ , denote the reversal of this language as  $L^R$ . Now, consider the language

$$L' = (L^R/a)^R.$$

This is also a regular language, since regular languages are closed under both the reversal and quotient operations. Then, what are the strings in  $L'$ ? It's exactly the strings in  $a \setminus L$ ! This is because:

$$\begin{aligned} L^R &= \{w^R \mid w \in L\} \\ L^R/a &= \{w^R a \mid w \in L\} \\ (L^R/a)^R &= \{(w^R a)^R = aw \mid w \in L\} \\ &= a \setminus L. \end{aligned}$$

■

**5. Practical Regular Expressions** For this problem, you are going to be working with real data taken from the UCSD library system and the UNIX command line tool `egrep`. Read the manual page for `egrep` (`man egrep` from the command line) to see what format is used for regular expressions. In particular, a single period matches any character while `|` is used for the union operation. The `*` operator is the same and `()` act as grouping. Lastly, `egrep` works a line at a time and it matches at any point in the line, not necessarily at the beginning. That means, it acts as though the regular expression starts and ends with `.*` to match any number of any character.

The data file `data.txt` can be downloaded from the course web page.

For example, to find every entry that is in the SSH 6th or 7th floor, run the command: `egrep 'SSH (6|7)th Floor' data.txt` This should print out quite a lot of data, 191 lines worth, in fact. Instead of printing out all of the lines, we can just get a count of the lines by using the `-c` option to `egrep` as in the command:

`egrep -c 'SSH (6|7)th Floor' data.txt` which prints out 191.

For each query below, you need to include in your write up the regular expression you used (`'SSH (6|7)th Floor'` in the above example) as well as the count of the lines that match *not the lines themselves*. All of them will have at least one, so if you get zero, something went wrong. I suggest examining the output without the `-c` to ensure that you are finding the entries you want. Note that each book can have multiple call numbers in the library. You should consider each of those to be a separate book. In addition, each call number can have multiple copies, these can be treated as a single book.

1. Modify the example above to find all books that are in the SSH 6th or 7th floor and have a year of 1999. Note that years are displayed as either 1999 or c1999. Make use of `.*` to skip over data you do not care about. How many are there?

$\Rightarrow$  `'SSH (6|7)th Floor.*c?1999'`

$\Rightarrow$  Count 2

2. How many books have a call number whose first part is exactly QA1? That is, QA1 .L471 v.642 should be matched but QA171 .M648 1961 should not.  
⇒ 'QA1 '  
⇒ Count 3
3. How many books are in the S&E Stacks which are available? (Look at the data file to see how available books are represented; it should be very obvious.)  
⇒ 'S&E Stacks.\*AVAILABLE'  
⇒ Count 109
4. How many books have a due date in June, 2008? (Note that multiple copies can be due on different dates. That's okay. If at least one copy is due in June, count it.)  
⇒ 'DUE 06-..-08'  
⇒ Count 26
5. How many books in the S&E Stacks are available and have a year between 1973 and 2001, inclusive?  
⇒ 'S&E Stacks.\*AVAILABLE.\*c?(19(7[3-9] | [89] [0-9]) | 200[01])'  
⇒ Count 81

■