CS 341: Foundations of Computer Science II
Prof. Marvin Nakayama

Project 1

Section 010: Due 2/15/2024, 11:30am NJ local time

Be sure to read this entire document before starting the assignment.

Academic Integrity

Any student caught cheating on any assignment will be reported to the dean of students. Cheating includes, but is not limited to, getting solutions (including code) from or giving solutions to someone else. You may discuss the assignment with others, but ultimately you must do and turn in your own work.

1 Overview

We define the language L to consist of strings that represent certain email addresses (specified below). For this assignment you are to submit a drawing of a state diagram of a DFA that recognizes this language, a program that implements your DFA, and your code's output (following a specified format) on the provided test cases.

2 The Language L

To precisely specify L, first define $\Psi = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{z}\}$ as the set of lower-case Roman letters. Also, define $\Pi = \{\mathtt{.}\}$ and $\Phi = \{\mathtt{@}\}$, and let $\Sigma = \Psi \cup \Pi \cup \Phi$; i.e., Σ is the set consisting of all the lower-case Roman letters, the dot (or period), and the @ symbol. (The symbols Ψ, Π, and Φ are the capital Greek letters Psi, Pi, and Phi, respectively.) Define the following sets of strings:

- $S_1 = \Psi\Psi^*$, which consists of strings over Ψ of length one or more
- $S_2 = \Pi\Psi\Psi^*$, which consists of strings starting with a dot and followed by one or more symbols from Ψ
- $S_3 = \{\mathtt{.gov}\}$
- $S_4 = \{\mathtt{.gr}\}$

Then we define the following sets of strings over Σ:

- $L_1 = S_1 \Phi S_1 S_3$

- $L_2 = S_1 S_2^* \Phi S_1 S_2^* (S_3 \cup S_4)$
- $L = L_1 \cup L_2$

The sets L_1 and L_2 include (some but not all) email addresses. For example, the strings `abc@njit.gov` and `a@gov.gov` belong to L_1. The strings `abc@njit.gov`, `abc@njit.gr`, `abc.def@cs.njit.gov`, and `abc.def.g@a.b.cs.njit.gov.gr` are in L_2.

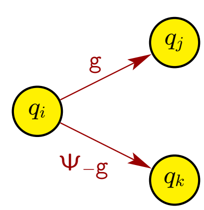
The specification of L does not include all valid email addresses because we included several restrictions to simplify the assignment. For example, only lower-case Roman letters, dots, and the @ symbol are allowed, strings in L must end with `.gov` or `.gr`, etc.

3 DFA for L

First construct a DFA $M = (Q, \Sigma, \delta, q_1, F)$ that recognizes L. The DFA must satisfy each of the following properties:

- The DFA must be defined with the above alphabet Σ. Your DFA does not have to handle symbols that are not in Σ.
- The states in the DFA must be labeled $q_1, q_2, q_3, \ldots, q_n$, where q_1 is the start state and n is the number of states in the DFA. (It is also acceptable for the states to be labeled $q_0, q_1, \ldots, q_{n-1}$, with q_0 the start state.)

To simplify your DFA drawing, you should omit any edges going from any state q to a "trap state" (i.e., a non-accepting state from which the DFA can never leave). All other edges must be included in your drawing. Clearly identify which state is the trap state in the drawing of your DFA, and your drawing should include a note stating that all edges not specified go to a trap state. Also, to simplify your drawing, you may define the set $\Psi_{-\ell} = \Psi - \{\ell\}$ for any symbol $\ell \in \Psi$; i.e., $\Psi_{-\ell}$ is the set of all lower-case Roman letters except for ℓ. For example, $\Psi_{-\mathtt{g}} = \{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{f}, \mathtt{h}, \mathtt{i}, \ldots, \mathtt{z}\}$ is the set of lower-case letters except for `g`, so your DFA might include something like the following:



There may be other transitions out of q_i corresponding to reading symbols from $\Sigma - \Psi$, but no other transitions out of q_i corresponding to reading any symbol from Ψ. Thus, if the DFA is currently in state q_i, then it moves to q_j on reading the letter `g`, and it moves to state q_k on reading any other lower-case Roman letter. You could also define the notation $\Psi_{-\mathtt{a},\mathtt{b}} = \Psi - \{\mathtt{a}, \mathtt{b}\}$. You can use any notation you like, as long as you define it appropriately.

Because a DFA must be deterministic, for each state $q_i \in Q$ and for each symbol $a \in \Sigma$, there must be exactly one edge leaving state q_i corresponding to reading a. For example, in the above figure, if the DFA is in state q_i and reads `g`, there is exactly one choice of where to move: state q_j. Similarly, if the DFA is in state q_i and reads `h`, there is exactly one choice of where to move: state q_k. Moreover, taking a transition must read exactly one symbol (no more and no less).

4 Program Specifications

You must write your program in either C, C++, Java, or Python. All input/output must be through standard input/output, and your program is to work as follows:

1. Your program first prints:

   `Project 1 for CS 341`
   `Section number:` *the section number you are enrolled in*
   `Semester: Spring 2024`
   `Written by:` *your first and last name, your NJIT UCID*
   `Instructor: Marvin Nakayama, marvin@njit.edu`

   where your NJIT UCID is typically your initials followed by some digits.

2. Your program asks the user to enter an integer $n \geq 0$ specifying the number of input strings to be processed, and your program prints out the value of n. If $n = 0$, the program terminates. If the user specified $n \geq 1$, your program enters a loop indexed by $i = 1, 2, \ldots, n$.

3. In the ith iteration of the loop, your program prompts the user, "Enter string i of n", where i is the iteration number and n is the total number of strings to enter, and your program then reads in the string. You may assume that the user will only enter a string over Σ. After reading in the string, your program prints it. Then your program processes the string on your DFA in the following manner.

   - Your program must begin in the start state of the DFA and print out the name of that state (q_1 or q_0).
   - After each character from the string is processed on the DFA, your program must print out the character and the name of the current state of the DFA. Even if your DFA is in a trap state, your program must do this for each character in the string until it reaches the end of the string.

4. After processing the nth string, your program terminates.

5. For all functions, subroutines, and classes you define in your code, their names must end with the last 3 digits of your NJIT UCID.

5 Test Cases

Test your program on each of the following input strings:

1. `njit@njit.gov`
2. `b@d.gov`
3. `tuv@wxyz.gr`
4. `opqr@.gov`
5. `web.rstuv.b.gov.cdef@z.gr.go.stanford.gov`
6. `gg@ggg.gov..gov`
7. `zyxw.@phone.gov`
8. `gov@govern.gov.`
9. `yellow@abc.gov.green`
10. `random@gov`
11. `rewq@g.gov.gr.gor`
12. `www@gov.govr`
13. `wwwc@gov.gr`
14. `.five@six.gov`
15. `governess@go.gov@`
16. `govern@government.gr.govt`
17. `govern@government.gov.gr`
18. `@mnop.gov`
19. `xyzz.bcde.grow@.gov`
20. `abcd..efg@fort.gov`

You must create an output file containing your program's output from each test case in the order above.

6 Deliverables

The due date/time for the assignment specified in the syllabus and at the beginning of this document. You must submit all of the following **through Canvas** by the due date/time:

1. A Microsoft Word document stating, "I certify that I have not violated the University Policy on Academic Integrity", followed by your first and last name, NJIT student ID, and UCID. If you do not include this pledge, then you will receive a 0 on the assignment. Anyone caught violating the University Policy on Academic Integrity will be immediately reported to the dean of students.

2. A drawing of the DFA for L that your program implements. This format of the file must be either Microsoft Word, pdf, or jpeg (e.g., a photo from your phone's camera, but make sure it's not blurry). The file must be smaller than 5MB in size.

3. A Microsoft Word document giving the 5-tuple specification for your DFA as

$$M = (Q, \Sigma, \delta, q_0, F) \quad \text{or} \quad M = (Q, \Sigma, \delta, q_1, F),$$

   depending on whether your DFA's start state is q_0 or q_1. You must explicitly give each of the elements in the 5-tuple, e.g., Q must be a set with all of the states in your DFA. Give the transition function $\delta : Q \times \Sigma \rightarrow Q$ as a table; e.g., see slide 1-8 of the lecture notes. Some transitions of your DFA will be taken on reading many different symbols, so you can simplify the table by combining these possibilities into a single column of the table. For example, in any state, your DFA on reading `y` or `z` should always make the same transition, so you can combine these columns in your table into a single column.

4. A **single file** of your source code, of type `.c`, `.cpp`, `.java`, or `.py`. Only submit the source code; do not include any class files. You must name your file

   `p1_24s_xyz_ucid.ext`

   where `xyz` is the section number in which you are enrolled, `ucid` is replaced by your NJIT UCID, and `.ext` is the appropriate file extension for the programming language you used, e.g., `.java`. The first few lines of your source code must be comments that include your full name, UCID, section number, and semester/year.

5. A **single file** containing the output (in the format specified in Section 4) from running your program on all of the test cases, in the order given in Section 5 of this document. The output file must be either .txt or in Microsoft Word.

The files **must not be compressed**. You will not receive any credit if you do not complete all of the above. **Late projects will be penalized as follows:**

Lateness (Hours)	Penalty
$0.0 <$ Lateness ≤ 24	10
$24 <$ Lateness ≤ 48	30
$48 <$ Lateness ≤ 72	60
$72 <$ Lateness	100

where "Hours" includes any partial hours, e.g., 0.0000001 hours late incurs a 10-point lateness penalty. A project is considered to be late if all of the above are not completed by the due date/time, and the lateness penalty will continue to accumulate until all of the above have been completed. Any submissions completed more than 72 hours after the due date/time will receive no credit.

7 Grading Criteria

The criteria for grading are as follows:

- correctness of your DFA for L (30 points; you will lose points if your DFA has more states than necessary or is overly complicated)
- 10 points: correctness of the 5-tuple specification of your DFA for L
- the program follows the specifications given in Section 4, matches your DFA for L, and follows the directions in Section 6 (15 points)
- your program is properly documented (each block of code should include comments) (5 points)
- your output is correct for the test cases and in the proper format specified in Section 4 (40 points).

Your grade will mainly be determined by examining the source code, the drawing of the DFA, and the output that you turn in; the source code will only be tested if there are questions about your code.

To receive any credit for this assignment, you must turn in a drawing of your DFA for L and a *minimally working* program. For a program to be minimally working, it must satisfy all of the following:

- compile without syntax errors;
- properly accept strings in L_1; and
- implement the drawing of your DFA for L.

If you do not hand in a minimally working program, then you will receive a 0 for the assignment *and* your grade in the course will be lowered by one step, e.g., from B to C+, or from C to D.

8 Hints

To design a DFA for L, start by drawing a DFA to handle only L_1, which includes the first two test cases. Initially include only the transitions that will eventually lead to acceptance. Then modify your DFA to handle the rest of L and the other test cases.