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

Project 1

Due Date for Section 003: 10/4/2021, 2:30pm NJ local time
Due Date for Section 451: 9/29/2021, 2:30pm 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 design a DFA to recognize L and write a program that implements your DFA.

2 The Language L

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

- $S_1 = \Gamma\Gamma^*$, which consists of strings over Γ of length one or more
- $S_2 = \Delta\Gamma\Gamma^*$, which consists of strings starting with a dot and followed by one or more symbols from Γ
- $S_3 = \{\mathtt{.net}\}$

Then we define the following sets of strings over Σ:

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

Strings in L_1 and L_2 are (subsets of) email addresses. For example, the string `abc@njit.net` belongs to L_1. The strings `abc@njit.net`, `abc.def@cs.njit.net`, and `abc.def.g@a.b.cs.njit.net` are in L_2.

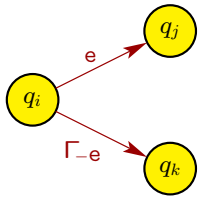
The specification of L does not include all valid email addresses because we included several restrictions to simplify the assignment. For example, only lower-case Roman letters, dots, and the `@` symbol are allowed, strings in L must end with `.net`, etc.

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.)

You will lose points if your DFA is overly complicated (e.g., having more states than necessary). To simplify your DFA drawing, you may 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 should define $\Gamma_{-\ell} = \Gamma - \{\ell\}$ for any symbol $\ell \in \Gamma$; i.e., $\Gamma_{-\ell}$ is the set of all lower-case Roman letters except for ℓ. For example, $\Gamma_{-\mathtt{e}} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{f}, \mathtt{g}, \ldots, \mathtt{z}\}$, so your DFA might include something like the following:



Thus, if the DFA is currently in state q_i, then it moves to q_j on reading `e`, and it moves to state q_k on reading any other lower-case Roman letter. You could also define the notation $\Gamma_{-\mathtt{a},\mathtt{b}} = \Gamma - \{\mathtt{a}, \mathtt{b}\}$.

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:  Fall 2021
   Written by:  your first and last name, your NJIT UCID
   Instructor:  Marvin Nakayama, marvin@njit.edu
   ```

2. Your program next asks the user if they want to enter a string. The user then enters "`y`" for "yes", or "`n`" for "no".

   - If the user enters "`n`", then the program terminates.
   - If the user enters "`y`", then the user is prompted to enter a string over Σ.

3. If the user chooses to input a string, your program then reads in the string. After reading in the string, your program prints it. Then your program processes the **entire** string on the DFA, one character at a time, 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.

   To simplify your program, you can check the ASCII code of each character of the string and process on the DFA accordingly.

4. After processing the entire string on the DFA, your program must indicate if the string is accepted or rejected based on the state in which the DFA ended. Your program then should return to step 2.

5 Test Cases

Test your program on each of the following input strings:

1. `foo@abcdef.net`
2. `z@n.net`
3. `ba@ba.ne`
4. `edfg@.net`
5. `webweb.ab.c.net@cs.defgh.net`

6. `foo@goo.net..net`
7. `abqe.@boom.net`
8. `educ@netw.netw`
9. `redblue@green..net`
10. `random@net`
11. `poke@a.net.net`
12. `www@net.nett`
13. `wwwb@net.ne`
14. `www.net@bbdef.net`
15. `food@for.net@`
16. `net@network.ne.net`
17. `network@network.net.ne`
18. `@abcde.net`
19. `people.dog.cat@.net`
20. `cable..cord@fort.net`

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

6 Deliverables

You must submit all of the following **through Canvas** by the due date/time given on the first page:

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 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 $\delta : Q \times \Sigma \to Q$ 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_21f_ucid.ext`

where `ucid` is replaced by your NJIT UCID (which is typically your initials followed by some digits) 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 and UCID.

5. A **single file** containing the output 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),

- correctness of the 5-tuple specification of your DFA for L (10 points),
- the program works according to the specifications given in Section 4, matches your DFA for L, and follows the directions in Section 6 (10 points),
- your program is properly documented (i.e., comments) (10 points),
- your output is correct for the test cases (40 points).

Your grade will mainly be determined by examining the source code, the drawing and 5-tuple 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 that end in "`.net`"; 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 and end in "`.net`", which includes the first three 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.