Theory of Computation: Programming Assignment 2 (NFA to DFA Conversion)

Arjun Chandrasekhar (borrowed from Dr. John Glick)

Due Sunday, 10/31/2021 at 11:59 pm (20 points)

For this assignment you will write a program that converts an NFA to a DFA. The behavior of your program should be exactly as described below.

The program should read input from a file that is specified as the first command line argument to the program, and write output to a file whose name is specified as the second command line argument to the program. The program should not prompt the user for interactive input.

You may use the powerset algorithm we discussed in class to convert the NFA to a DFA. You may also use the following algorithm:

1. Create the start state of the DFA, which is $E(q_0)$, where q_0 is the NFA start state and $E(.)$ is the ϵ-closure
2. For every new state R that was added in the previous step, and for every alphabet character σ, do the following:
   (a) Compute $\cup_{r \in R} E(\delta(r, \sigma))$. This applies the NFA's transition function to each state $(r \in R, \sigma)$, and then computes the ϵ-closure. This will return a set of states T. Add the DFA transition $\delta(R, \sigma) = T$
   (b) If the DFA did not already have T as a state, add it as a new state and go back to step 2
3. Whenever step 2 does not yield any new states, we are done adding states
4. Make every DFA state that contains an NFA accept state into a DFA accept state.

Rather than creating all possible states and transitions, we only consider states and transitions that are reachable from the start state.

The format of the NFA description is as follows (this is close to the same as the DFA specification in the first programming assignment, except that there can be nondeterminism, epsilon transitions, and a transition does not have to be specified for all state/symbol combinations):

1. An integer that is the number of states in the NFA. This appears by itself on the first line of input. (In the remainder of the description, each state must be referred to by an integer in the range $[1, 2, ..., n]$, where n is the number of states.)

2. The alphabet of the NFA. This appears by itself on the second line of input. Every character in the line (not including the terminating newline character) is a symbol of the alphabet. The alphabet cannot include the letter e, as this is reserved for specifying epsilon transitions.

3. The transition function of the NFA. There will be one line of input per possible transition in the transition function, starting with the third line of input. The format of an entry is

   ```
   qa 'c' qb
   ```

   This entry indicates that if the NFA is in state `qb` and the next symbol scanned is a `c`, then the NFA can transition to `qb`. `c` can also be the letter e, in which case is represents an epsilon transition.

   `qa` and `qb` must be valid states; that is, qa $\in \{1, 2, \ldots, n\}$ and qb $\in \{1, 2, \ldots, n\}$. `c` must be in the alphabet or be the letter e (representing epsilon), and the single quotes must be present (even for the letter e)

   The entries of the transition function can appear in any order.

4. A blank line terminates the transition function entries. We need this because we don't know in advance how many transition function entries there will be.

5. An integer that is the start state of the NFA. This appears by itself on the first line after the transition function lines.

6. The set of accept states of the NFA. These appear together on the line following the line containing the start state.

The format of the DFA description is as follows (this is the same as for the first programming assignment):

1. An integer that is the number of states in the DFA. This appears by itself on the first line of input. (In the remainder of the description, each state must be referred to by an integer in the range $[1, 2, ..., n]$, where n is the number of states.)

2. The alphabet of the DFA. This appears by itself on the second line of input. Every character in the line (not including the terminating newline character) is a symbol of the alphabet.

3. The transition function of the DFA. There should be one line of input per entry in the transition function table, starting with the third line of input. The format of an entry is

```
qa 'c' qb
```

   This entry indicates that if the DFA is in state `qa` and the next symbol scanned is a `c`, then the DFA transitions to `qb`.

   `qa` and `qb` must be valid states; that is, qa $\in \{1, 2, \ldots, n\}$ and qb $\in \{1, 2, \ldots, n\}$. `c` must be in the alphabet, and the single quotes must be present.

   The entries of the transition function can appear in any order.

4. An integer that is the start state of the DFA. This should appear by itself on the first line after the transition function lines.

5. The set of accept states of the DFA. These should appear together on the line following the line containing the start state.

Multiple entries on a line should be separated by 1 space character. No whitespace characters should precede the first entry on a line, and no characters should follow the last entry on a line (not counting the newline character).

Your program can assume that the input file will be correctly formatted, with no inconsistent data. For example, if there are only 6 states in the NFA, there will be no transition function entries that specify a state of greater than 6 or less than 1.

You may program in Java or Python3. I strongly suggest using Python if possible; this assignment will be much more convenient to code in Python, and it will be easier for me to help you.

You may work on this assignment alone or with a partner. On Canvas you should join a group (even if you are working alone) before you submit - here are instructions for joining a group. By joining a group, you only have to make one submission for each group. If you work with a partner, you should use the pair programming software development technique to ensure that both of you are contributing to and learning from the assignment.

If your program follows the input/output directions correctly, and if your program passes at least one of the test cases, you will have one week to debug your program and re-submit for

full points, starting from the time it is graded. If you fail to follow directions, e.g. interactive input or multiple files, or if your program does not pass any test cases, your submission will be treated as late and you will only be eligible for half of the points.

Test inputs and correct outputs are in a compressed testcases folder on the assignment webpage. Your program should work correctly on all of these tests. Because the numbering of your states in the dfa could be different than mine, your output dfa's might appear different than mine. That does not mean they are not correct. So, the sample input/output folder has 4 files for each test case. For example, for test case 1, the files will be:

- `nfa1.txt`: contains the nfa specification
- `dfa1.txt`: contains the dfa equivalent to the nfa in `nfa1.txt`. If your dfa matches this exactly, great. If not, the numbering of states may just be different than mine. In that case, the next two files are helpful
- `dfa1Input.txt`: contains test inputs for the dfa you have created.
- `dfa1Output.txt`: contains correct results for the test inputs in the `dfa1Input.txt` file.

Submit your assignment on Canvas. Your program should be contained in one source file (with the appropriate extension for the language you are using). **Do not submit more than one file, or an entire folder; all of your code should be in one source code file.** If you are coding in java, please do not include any "package ..." statements at the top of your file; my directory names are different from yours, and I will get an error when I try to compile your code (see section 4 of this page for a more detailed explanation).

Your work will be graded as follows:

- You will get 8 points for submitting a program that compiles and runs without errors, and passes at least one test.
- You will get 2 points for each test case that you pass
- If your program does not compile, does not run, does not follow the input/output directions (i.e. takes interactive input rather than a command line argument), or fails all test cases, you will receive 0 points.
- If your program follows the input/output directions correctly, and if your program passes at least one of the test cases, you will have one week to debug your program and re-submit for full points, starting from the time it is graded. If you fail to follow directions, e.g. interactive input or multiple files, or if your program does not pass any test cases, your submission will be treated as late and you will only be eligible for half of the points.