Automata Programming Assignment

Submitted by **T.H.Arjun 2019111012 CSD**

Structure

- `q<n>.py` - The code for question `n`
- `tests`- the testcases shown in video
- video

Running the code

```
python3 q<n>.py <input_file> <output_file>
```

input_file and output_file should be in the format specified by the assignment

Report

Question 1 : Regex to NFA

In question 1, we are required to convert the given regex to NFA. For this I first converted the given regex to postfix using the Shunting Yard Algorithm. This step is done by `shunt(infix)` function which returns a postfix giving precedence as `*,.,+`. I am explicitly adding `.` to make the parsing easier as mentioned in the above link. Brackets are also taken care by the algorithm.

The next step in the conversion is converting the postfix to an expression tree. The expression tree is a binary tree in which each internal node corresponds to the operator and each leaf node corresponds to the operand.

After creating the Expression Tree, we can evaluate the NFA by Thompson's Construction

The algorithm is described in short below: The algorithm works recursively by splitting an expression into its constituent subexpressions, from which the NFA will be constructed using a set of rules.

Rules of Thompson's Construction

In what follows, N(s) and N(t) are the NFA of the subexpressions s and t, respectively.

- The empty-expression ε is converted to q to f with edge ε.
- A symbol `a` of the input alphabet is converted to q to f with edge `a`
- The union expression s + t is converted to: State q goes via ε either to the initial state of N(s) or N(t). Their final states become intermediate states of the whole NFA and merge via two ε-transitions into the final state of the NFA.
- The concatenation expression st is converted to-The initial state of N(s) is the initial state of the whole NFA. The final state of N(s) becomes the initial state of N(t). The final state of N(t) is the final state of the whole NFA.

- The Kleene star expression s* is converted to: An ε-transition connects initial and final state of the NFA with the sub-NFA N(s) in between. Another ε-transition from the inner final to the inner initial state of N(s) allows for repetition of expression s according to the star operator.
- The parenthesized expression (s) is converted to N(s) itself.

This is done by the recursive function `getNFA(et)` which takes the expression tree as argument.

The code is commented well to explain the method in detail.

Question 2: Convert NFA to DFA (Powerset Construction)

In question 2, we are required to convert an NFA to DFA. For this I used the method of Powerset or Subset Construction as we are required to output all states and their transition functions. The powerset construction or subset construction is a standard method for converting a nondeterministic finite automaton (NFA) into a deterministic finite automaton (DFA) which recognizes the same formal language.

The powerset construction applies most directly to an NFA that does not allow state transformations without consuming input symbols (aka: "ε-moves"). Such an automaton may be defined as a 5-tuple (Q, Σ, T, q0, F), in which Q is the set of states, Σ is the set of input symbols, T is the transition function (mapping a state and an input symbol to a set of states), q0 is the initial state, and F is the set of accepting states. The corresponding DFA has states corresponding to subsets of Q. The initial state of the DFA is {q0}, the (one-element) set of initial states. The transition function of the DFA maps a state S (representing a subset of Q) and an input symbol x to the set T(S,x) = ∪{T(q,x) | q ∈ S}, the set of all states that can be reached by an x-transition from a state in S. A state S of the DFA is an accepting state if and only if at least one member of S is an accepting state of the NFA. For an NFA with ε-moves (also called an ε-NFA), the construction must be modified to deal with these by computing the ε-closure of states: In the simplest version of the powerset construction, the set of all states of the DFA is the powerset of Q, the set of all possible subsets of Q (this is the method I used)

Code Explanation in Brief

Firstly I am dealing with epsilon closures with a BFS. (I did this part before clarification that ε-NFA will not be tested) I convert epsilon transitions to direct edges in the DFA.

After getting the Epsilon closure done by coverting edges, I generate all the subsets of states and make them the new states. After that I iterate all the vertices of the given NFA and update the adjacency list accordingly to get this DFA equivalent to given NFA.

The code is well commented to explain the execution.

Question 3: NFA to Regex

The third question requires us to convert a NFA to Regex for this I used the algorithm that converts an NFA to G-NFA and then uses state elimination method on G-NFA to get a regex. Consider an NFA N where we allowed to write any regular expression on the edges, and not only just symbols. The automata is allowed to travel on an edge, if it can matches a prefix of the unread input, to the regular expression written on the edge. We will refer to such an automata as a G-NFA.

G-NFA has the following conditions:

- There are transitions going from the initial state to all other states, and there are no transitions into the initial state.
- There is a single accept state that has only transitions coming into it (and no outgoing transitions)
- The accept state is distinct from the initial state.
- Except for the initial and accepting states, all other states are connected to all other states via a transition. In particular, each state has a transition to itself.

Top Level Outline of the Conversion

- Convert NFA to a G-NFA, adding new initial and final states.
- Remove all states one-by-one, until we have only the initial and final states.
- Output regex is the label on the (single) transition left in the G-NFA.

We first describe the construction. Since k > 2, there is at least one state in N which is not initial or accepting, and let qrip denote this state. We will “rip” this state out of N and fix the NFA, so that we get a NFA with one less state. Transition paths going through qrip might come from any of a variety of states q1, q2, etc. They might go from qrip to any of another set of states r1, r2, etc. For each pair of states qi and ri , we need to convert the transition through qrip into a direct transition from qi to ri .

Brief Explanation of Code for Q3

- First I add a new start `qinit` as the algo describes and connect old start to this and make the old one non-start.

- Then I add a new single accept state and and connect all the old accept states to this final state `qfin`

- Now until the number of states left are 2 we keep removing `qrip`.

- The process of removing `qrip` is as follows.

    - Remove all the deadstates in the G-NFA. Deadstates are states with no outgoing edge and is not an accept state. And update all states.
    - Then we pick a state say `qrip` that is to be removed.
    - We remove this state from the Automaton and make the required changes to the edges as commented in code.
    - We update the edges, after ripping of `qrip`
    - Update all other states to reflect these chnages.
    - Iterate the process until we are left with only `qfin` and `qinit`

- The final regex will be the value of transition from `qfin` to `qinit`

The algorithm and the explanation is given in detail here

The code is also well commented to explain the process.

Question 4: Minimize a DFA

In question 4 we are required to minimize a given DFA For this I used the table filling algorithma and removal of non reachable states to achieve minimimum DFA.

Brief Explanation of Code for Q4

- Firstly I construct a DFA from given inputs in terms of classes.
- Then I remove all the states that are not reachable from initial state. This is achieved by getting a list of all reachable states by doing a BFS. Then we follow the table filling Algorithm or Myphill-Nerode Theorem The algo goes like this: -Draw a table for all pairs of states (Qi, Qj) not necessarily connected directly [All are unmarked initially]
    - Consider every state pair (Qi, Qj) in the DFA where Qi ∈ F and Qj ∉ F or vice versa and mark them. [Here F is the set of final states]
    - Repeat this step until we cannot mark anymore states –If there is an unmarked pair (Qi, Qj), mark it if the pair {δ (Qi, A), δ (Qi, A)} is marked for some input alphabet.
    - Combine all the unmarked pair (Qi, Qj) and make them a single state in the reduced DFA.

If the equivalence of two states in undetermined, we can look for transitions of the two states on the same symbol, where the destinations are known to be distinct. If such transitions are found, the original states are known to be distinct also.

Then I build the list of equivalent classes and update the DFA as required, combining states and redirecting edges.