Automata-Theory

Ayush Sharma (2019101004)

Code for following conversion :

- Regular Expression → NFA
- NFA → DFA
- DFA → Regular Expression
- Minimizing a DFA

Regex → NFA

Code Flow

- Checking input file format. Loading if correct using `json` module.
- Called `convertToNFA()` function, which does the following:-
    1. If `regex` is empy string then it will return `NFA` containing 2 state with epsilon on the arc connecting them.
    2. Otherwise, I'll add character `.` , if possible, between the characters for recognising as concatenation operator for two NFA.
    3. Then, converted the given regex which is in `infix` format to new regex in `postfix` format using stack. Precedence order : `()` > `*` > `.` > `+` .
    4. Next, `makeNFA(postfixRegex)` is called, that utilises method known as `Thompson contruction rules` to finally produce the `ε-NFA` . (Explained in below points)
    5. Working of `makeNFA(postfixRegex)` function:
        - Made an empty stack.
        - Iterate through `postfixRegex` .
        - If the character is either of ε or an alphabet , push NFA for that using `unitLenRegexToNFA(alphabet)` function in the stack.
        - When our expression contains union or concatenation, we take two NFAs from the stack and combine them with the operator. Then push the new NFA in the stack. Function used : `unionNFA(NFA1, NFA2)` or `concatenationNFA(NFA1,NFA2)` repectively.
        - In the kleene star operation, only one element from the stack is popped and the operation is done on it using `starNFA(NFA1)` . The new NFA is again pushed in the stack.
        - At the end of string `postfixRegex` , we have only one NFA in the stack. That is our final `ε-NFA` .

Thomson Construction Rules:-

This construction exploits these 2 facts:-

1. Closure property of a NFA over `Kleene star` , `Union` & `concatenation` operation.

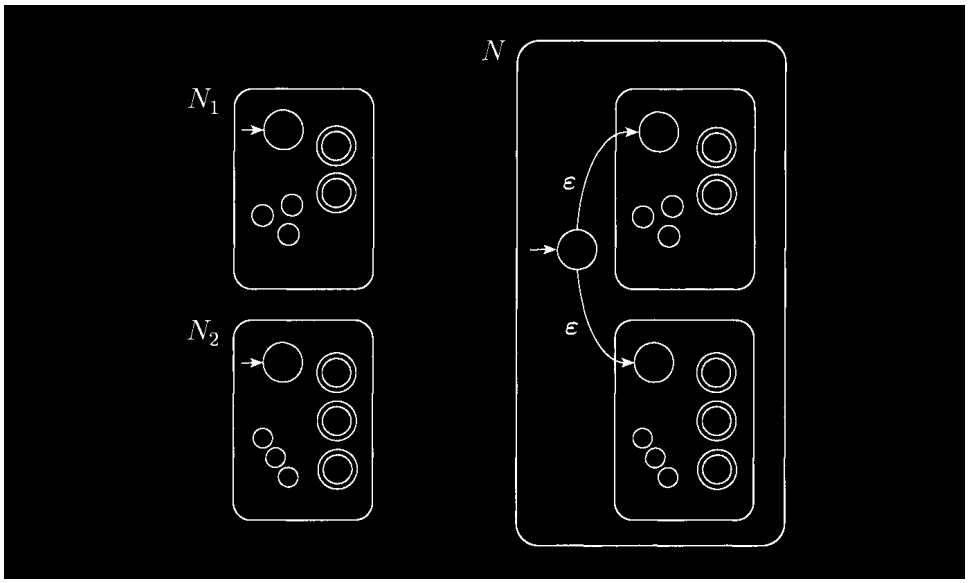
2. `ε-transitions`. For example for a given NFA, `ε-transitions` of start set will be set of all states that can be reached by traversing the graph on edges have label `ε` on it.

Note : Using `ε-transitions`, this approach creates a regular expression from its elements. The `ε-transitions` serve as a "glue" or "mortar" for the NFA subcomponents. Since concatenation with the empty string leaves a regular expression unchanged (concatenation ε with is the identity operation), hence ε-transition contributes nothing.

1. The NFA's for single character regular expressions ε, a, b :

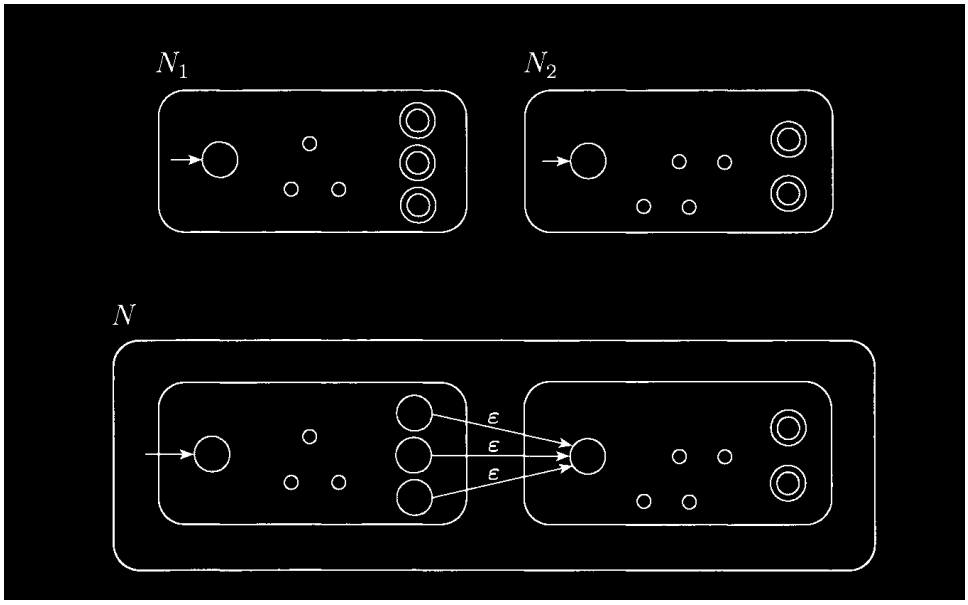
```
          a/b/ε
→(Q0)----------->((Q1)) ; Q0 start states & Q1 is final States
```

2. **UNION** : The NFA for the union of N1 and N2: `N1+N2` is made up of individual NFAs with the NFA acting as "glue." Individual accepting states can be removed and replaced with the general accepting state as following:
   - Function `unionNFA(NFA1, NFA2)` return union of two provided NFAs.



3. **Concatenation** : The initial state of `N1` is the initial state of the whole NFA. The final state of `N1` becomes the initial state of `N2` . The final state of `N2` is the final state of the whole NFA.

- Function `concatenationNFA(NFA1,NFA2)` return union of two provided NFAs.



4. **Kleene Star** : The NFA for the star of N1: `N*` is made by having `ε-transitions` in two ways:
    1. from new Global start state to all old start states and from all final states to old start states.
    2. from new Global start state to all old start states , from all final states to old start states, from all final states to new Final state and rom new Global start state to new Final state.

    - Function `starNFA(NFA)` perform second type operation on provided `NFA` .