Automata Theory

Converting NFA to DFA

Objectives:

- Converting a Nondeterministic finite automaton to a deterministic finite automaton.
- Input as well as output taken as Json five tuples for a file input.json.
- Output as well as output taken as Json five tuples for a file Output.json

A Finite Automata consists of the following :

```
Q : Finite set of states.
&#x2211; : set of Input Symbols.
q : Initial state.
F : set of Final States.
&#x3B4; : Transition Function.
```

DFA (Deterministic Finite Automata):

```
DFA consists of 5 tuples {Q, &#x2211;, q, F, &#x3B4;}.
Q : set of all states.
&#x2211; : set of input symbols. ( Symbols which machine takes as input )
q : Initial state. ( Starting state of a machine )
F : set of final state.
&#x3B4; : Transition Function, defined as &#x3B4; : Q X &#x2211; --> Q.
```

NFA (Non Deterministic Automata):

```
&#x3B4;: Transition Function
&#x3B4;:  Q X (&#x2211; U &#x3F5; ) --> 2 ^ Q.
```

INPUT:

- Input is taken from a file input.json as shown below:

```
{
        "states": 8,
        "letters": ["a", "b", "c"],
        "t_func" : [1, 'a', [1,3,0]],
        "start" : 0,
        "final" : [4]
    }
    Here:
states : Number of states. Assume the states are numbered 0,1,2....n-1 for n states. lett
ers: Alphabet used by the NFA.
t func: The transition function for the NFA. Each transition is an array of 3 elements. o
riginal state, input and the new state.
start: The index of the starting state.
final: List of accepted states.
```

Executing the Script:

```
python3 script.py
```

Caution

- Both input.json and output.json should be the same directory as script.py

Code Expaination:

Function Description:

- Generating a POWER SET(Set of all subsets of a Set)

```
      def PowerSet(set, set_size):
```

- Taking Union Of a Set

```
def UNION(StateA, t_func, input):
```

- Combining Two Sets

```
def Set_Combiner(set1, set2):
```

- Generates Transition Function for DFA:

```
def Generator(TF_NFA, TF_DFA, alphabet, cur, states_dfa):
```

1 . PowerSet:

This function is used to calculate all the Possible States for the DFA. 2^n States

```
0,1,2.....2n-1.
```

code

```
pow_set_size = (int)(pow(2+check, set_size))
    power_set = list()
    for counter in range(0+check, pow_set_size):
        each_element = list()
        for j in range(0+check, set_size):
            if((counter & (1+check << j)) > 0+check):
                each_element.append(set[j])
        power_set.append(each_element)
    return power_set
```

pow calls function power.Pow_set_size of power set of a set with set_size 2^n.

Check if jth bit in the counter is set

If set then save jth element from set

2. Gernerator:

This uses Transition state of NFA ,Alphabet and Initial state of NFA as Inputs.

```
def Generator(TF_NFA, TF_DFA, alphabet, cur, states_dfa):
    states_dfa.append(cur)
    for i in range(len(alphabet))  :
        inp=alphabet[i]
        Present= []
        Present.append(cur);Present.append(inp);Present.append(UNION(cur, nfa["t_func"],
 inp))
        TF_DFA.append(Present)
        trace.append(Present)
        if(Present[2+check] not in states_dfa):
            TF_DFA = Generator(TF_NFA, TF_DFA,alphabet, Present[2+check], states_dfa)
    return TF_DFA

```

This uses Transition state of NFA ,Alphabet and Initial state of NFA as Inputs.

The function is used RECURSIVELY calls itself.

And later union is taken

3 Union :

This function is used to take union of two sets.Specifially for generation of new State.

OUTPUT:

Js Beautifier

```
opts = jsbeautifier.default_options()
dfa["final"] = Set_Combiner(Possible_States, nfa["final"])

opts.indent_size = 2

```

```
formatted_json = jsbeautifier.beautify(json.dumps(dfa), opts)

with open('./output.json', 'w') as json_file:
    json_file.write(formatted_json)
```

It is Usefd to stored files in a Orgsnised Json format.

Output is Stored in output.json