nfaToDfaConverter

August 14, 2019

1 Automata Theory : Assignment 1

1.1 Authored by Bhuvanesh Sridharan

1.2 Roll No: 2018113002

2 Creating a program to convert an NFA into DFA with json files as input and output

Including necessary modules:

```
In [1]: import numpy , json
```

Taking input from the external json file:

```
In [3]: string = """
        {
            "states" : 3 ,
            "letters" : ["a","b"],
            "t_func" : [[0,"a",[0,1]],[0,"b",[0]],[1,"b",[2]]],
            "start" : 0,
            "final" : [2]

        }
        """

        ## Or we can take input using string by using :
        ## inData = json.loads(string)
        with open("input.json","r") as inFile:
            inData = json.load(inFile)
```

2.0.1 Conversion Notation :

We employ a system where each element of the powerset of states in NFA can be represented by just one unique integer in the DFA.

The relation is :

$$\{n_1, n_2, n_3, ...\} = \sum_{1} 2^{n_i}$$

Example:
The set $\{0, 1\}$ will be represented as the state $2^0 + 2^1 = 3$ in DFA.
Similary, the state 3 in NFA will be represented as the state $2^3 = 8$ in the DFA generated.
The following two functions convert these from one domain to another.
Example:
conv converts from powerset to DFA state.
and convR converts from DFA state to the powerset in NFA.

```
In [4]: conv = lambda x : sum([2**i for i in x])
        convR = lambda x : [i for i in range(inData["states"]) if x>>i&1]
```

Initialising the DFA machine:

```
In [5]: outData = { }
        outData["states"]=2**inData["states"]
        outData["letters"]=inData["letters"]
        outData["start"]=2**inData["start"]
        outData["t_func"]=[]
```

delta function returns the states from NFA when a state and input letter is passed as argument.

```
In [6]: def delta(state,letter):
            outStates=[]
            for i in inData["t_func"]:
                if i[0]==state and letter==i[1]:
                    outStates = i[2]
            return outStates
```

resolveState takes an input of a state in the domain of DFA and inserts all the elements of transition function in the DFA using the transition function of NFA and **taking their union** then converting them to the domain of DFA again.

```
In [7]: def resolveState(st):
            #Here st is in the space of states of DFA and not NFA.
            #Hence we need to convert it in to the space of NFA before proceeding.
            statesToResolve = convR(st)

            for letter in inData["letters"]:
                outStates=[]
                for state in statesToResolve:
                    outStates = outStates + delta(state,letter)


                outStates = conv(list(set(outStates)))
                outData["t_func"].append([st,letter,outStates])
```

resolveFinalStates finds all the states in DFA domain which have the final states of NFA and includes them in the final state of DFA

```
In [8]: def resolveFinalStates():
            outData["final"]=[]
            for f in inData["final"]:
                for state in range(outData["states"]):
                    if state>>f&1 :
                        outData["final"].append(state)
```

Finally we are calling all the functions required to resolve the transition and final states:

```
In [9]: for state in range(outData["states"]):
            resolveState(state)
        resolveFinalStates();
```

Writing it into an output file:

```
In [12]: with open("output.json","w") as outFile:
             json.dump(outData,outFile)

         print(outData)
```

```
{'states': 8, 'letters': ['a', 'b'], 'start': 1, 't_func': [[0, 'a', 0], [0, 'b', 0], [1, 'a',
```