Automata Theory : assignment 1

Bhavyajeet Singh: 2018111022

August 19, 2019

1 Functionality

The python code essentially converts an NFA (non-deterministic finite state automaton) to a DFA (determinstc finite state automaton)

2 input format

The input NFA is always read from a file named "input.json" which contains a json object in the following format :
{
"states" : <Number of states>
"letters" :<set of possible input letters>
"t_func" : <transition function for NFA >
"start" : <the initial state>
"final" : < set of final states>
}

3 output format

The program writes the output in the form of a json object with the format being the same as the input format .
the output contains the following information :
{
"states" : <Set of all possible states>
"letters" :<set of possible input letters>
"t_func" : <transition function for DFA >
"start" : <the initial state>
"final" : < set of final states>
}

the transition function of the DFA is a list of $2^n \times |\Sigma|$ elements where n is the numbber of states in NFA and $|\Sigma|$ is the number to total possible input symbols .
Every element in itself is a list of length 3 where the first element is the current state , second element is the input symbol and the third element denotes

the output state .

the states of DFA are represented as the elements of power set of the states of NFA i.e every state in DFA is a set of the states of NFA and hence a total of 2^n states .

4 working of the code

4.1 taking input

we use the open command to open the file "input.json" and the load fuction from the json module to load the json object into a dictionary named data.
n stores the number of states
letters is a list which stores the set of input letters

```
import json
k= "input.json"
with open (k,"r") as inputjson:
        data = json.load(inputjson)
n=data["states"]
letters=data["letters"]
```

4.2 creating powerset

the power set is created by iterating through every number from 0 to $2^n - 1$. the numbers are converted into binary strings of n bit and if i^th is set to 1 , then that indicates that the state i from NFA is included in the current DFA state
hence for n=3 ,
0 equals 000 $\rightarrow$ []
1 equals 001 $\rightarrow$ [0]
2 equals 010 $\rightarrow$ [1]
3 equals 011 $\rightarrow$ [0, 1]
and so on ..

```
stateset=[]
for i in range (2**n):
```

```
        l=fostr.format(i)
        li=convbin(i)
        stateset.append(li)
```

given below is the function to convert a number to its equivalent set (element of the power set of NFA states)

```
def convbin(n):
        fostr="{:0"+str(n)+"b}"
        binstr=fostr.format(n)
        li=[]
        c=0
        for i in range(n-1,-1,-1):
                if binstr[i] == '1' :
                        li.append(c)
                c+=1
        return li
```

4.3 creating the transition function

the code below is used to create the transition function of the newly created DFA , the outermost loop iterates from 0 to 2^n , representing every state of the DFA .
for every state the inner loop iterates through the alphabet of the NFA and then, for every state alphabet combination, the output state is created .
At first an empty list is inserted as the output for every combination and then for every element in the input state set, the program iterates through the NFA trasition function list and their corresponding output is added to a list named "tomake" using python set union in order to avoid duplicacy

```
curr=0
for i in range (2**n):
  l=fostr.format(i)
  li=convbin(i)
  stateset.append(li)
  for alpha in letters:
    c=0
    tomake.append([li,alpha,[]]);
```

```
tomake1.append([i,alpha,0]);
for j in range (n-1,-1,-1) :
  if c in data["final"] and l[j]=='1' and alpha == letters[0]:
    finalstates.append(convbin(i))
  if l[j] == '1':
    for k in data["t_func"]:
      if k[0]==c and k[1]== alpha :
        tomake[curr][2] = list(set(tomake[curr][2]) | set(k[2]))
        tomake1[curr][2] = tomake1[curr][2] | tobin(k[2])
  c+=1
curr+=1
```

4.4 creating the final state set

for every element of the DFA state set, if it contains the any of the elements from the final state set of the NFA , it is appended to the list of final states of the DFA .
the above mentioned task is achieved through the following code segment :

```
if c in data["final"] and l[j]=='1' and alpha == letters[0]:
        finalstates.append(convbin(i))
```

4.5 writing final output

all the separately created lists are added as elements of the dictionary named outputjs which is finally written onto the file "output.json" using dump function of the json module

```
outputjs = {}
outputjs["states"]=stateset
outputjs["letters"]=letters
outputjs["t_func"]=tomake
outputjs["start"] = data["start"]
outputjs["final"]=finalstates
```

```
with open("output.json",'w') as outputfile :
        json.dump(outputjs,outputfile,indent=1)
```