Report

August 18, 2019

- Subject : Automata Theory
- Assignment 1
- Roll No : 20181130003

[1]:
```
### The script requires an nfa.json file as an input
```

This code :

- Imports JSON library which interprets the data given in the json
- Loads the data in nfa.json to the dictionary 'data'

[2]:
```
import json
data = json.load(open('nfa.json'))
```

This code:

- Finds the number of states (2Q) and saves it in the variable s

[3]:
```
s = 2**data['states']
print(s)
```

```
256
```

This code:

- Generates a list ls which stores all the states from 0 to Q-1

[4]:
```
ls=[]
for i in range(data['states']):
    ls.append(i)
print(ls)
```

```
[0, 1, 2, 3, 4, 5, 6, 7]
```

This code:

- Generates the powerset of the list given to it
- It does it by using the chain library to iterate through all the combinations of the items in the list

[5]:
```python
from itertools import chain, combinations
def powerset(iterable):
    "powerset([1,2,3]) --> () (1,) (2,) (3,) (1,2) (1,3) (2,3) (1,2,3)"
    s = list(iterable)
    return chain.from_iterable(combinations(s, r) for r in range(len(s)+1))
```

[6]:
```python
pq = list(powerset(ls))
```

This code:

- Generates the δ_D function from the δ_N according to the rule :

$$\delta_D(R_1, a) = \bigcup_{r \in R_1} \delta_N(r, a)$$

[7]:
```python
t = [] # The empty delta function
for i in pq: # Iterating over all the elements of the powerset ==> All the␣
↪states of the DFA
    for j in data['letters']: # Iterating over all the letters present
        # Creating a set to get the union operation when appending to it
        # Resets itself for every new letter and state
        nex_st = set()
        for k in data['t_func']:
            # Checking if the NFA has a transtion from i via j
            if k[0] in i and j == k[1]:
                # Updating for every transition the exists from i via j (Union␣
↪operaion)
                nex_st.update(k[2])
        #After exhausting through all the transition states(of NFA), can write␣
↪it to the delta function
        t.append([list(i),j,list(nex_st)])
```

[8]:
```python
for i in range(10):
    print(t[i]) # Looping for clearer print statement
```

```
[[], 'a', []]
[[], 'b', []]
[[], 'c', []]
[[0], 'a', []]
[[0], 'b', []]
[[0], 'c', []]
[[1], 'a', [0, 1, 3, 5]]
[[1], 'b', []]
[[1], 'c', []]
[[2], 'a', [0, 1, 3, 5]]
```

This code:

- Iterates over all the elements of the DFA and finds ones which have $i_N \in Q_D$

[9]:
```
st=[]
for i in pq:
    if data['start'] in i:
        st.append(list(i))
```

[10]:
```
for i in range(10):
    print(st[i])
```

```
[5]
[0, 5]
[1, 5]
[2, 5]
[3, 5]
[4, 5]
[5, 6]
[5, 7]
[0, 1, 5]
[0, 2, 5]
```

This code

- Finds all final states according to the rule:

$$F_D = \bigcup_{f_1 \in F_N} \{f_2 | f_2 \in Q_D and f_2 \cap f_1 \neq \phi\}$$

[11]:
```
fin=[]
for i in pq: # Iterating over all the input states f2
    for j in data['final']: # Iterating ove all the final states f1 in N
        if j in i: # chcking if they have any element in common
            fin.append(list(i))
```

[12]:
```
for i in range(10):
    print(fin[i])
```

```
[4]
[0, 4]
[1, 4]
[2, 4]
[3, 4]
[4, 5]
[4, 6]
[4, 7]
[0, 1, 4]
[0, 2, 4]
```

This code:

- Makes a dictionary to save the output as a JSON

[13]:
```
out = {}
out['states'] = s
out['letters'] = data['letters'] # The sigma set does not change from NFA to
 ↪DFA
out['t_func'] = t
out['start'] = st
out['final'] = fin
```

This code:

- Stores the given dictionary and saves it as a JSON file (out.json)

[14]:
```
with open('out.json', 'w') as outfile:
    json.dump(out, outfile,indent=4) # Formatting the output to make it
 ↪viewable
```