NFA TO DFA CONVERTER

Naren Akash R J

CS1.301 Automata Theory

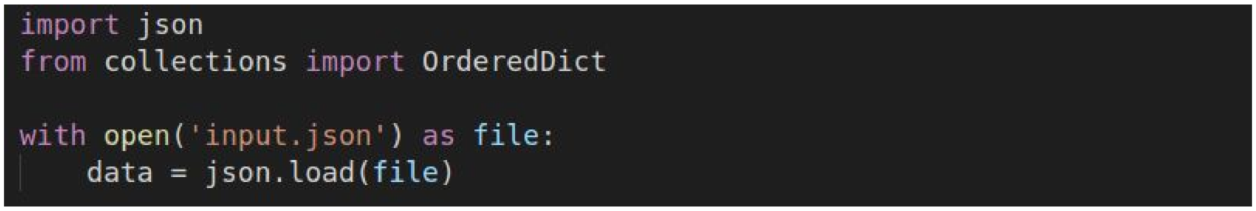
ALGORITHM

An NFA which recognizes a language L is represented formally by a 5-tuple, (Q, Σ, Δ, q_0, F). Then, DFA D = (Q', Σ, Δ', q_0, F') can be constructed for language L.

Step 01: Initially $Q' = \phi$. Add q_0 to Q'.
Step 02: For each state in Q', add the possible set of states for each input symbol using the transition function of NFA to Q' if not present already.
Step 03: The final state of DFA will be all states containing F.

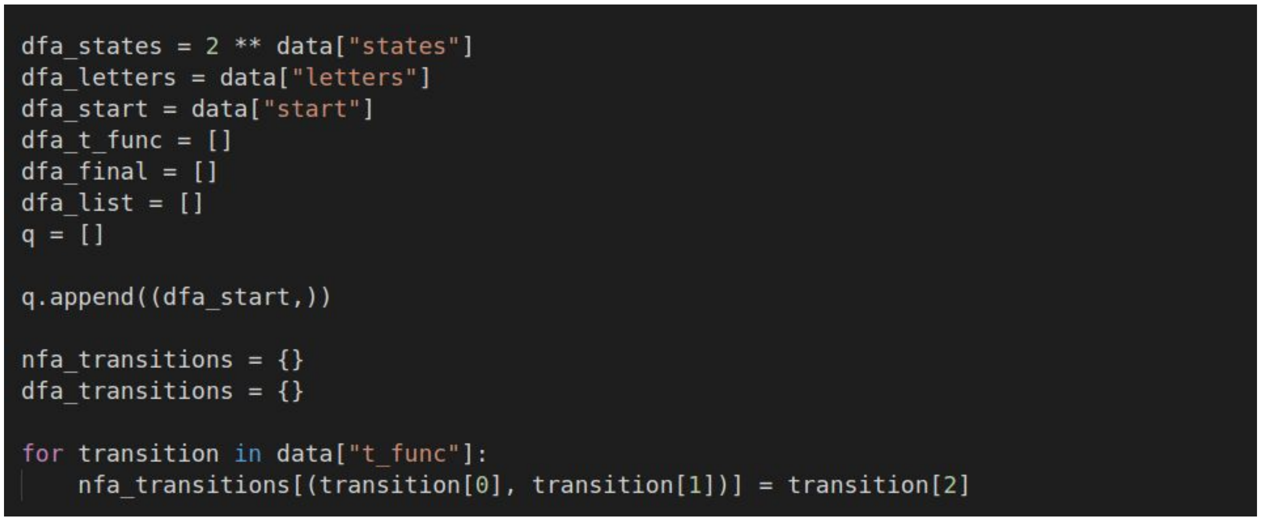
Note: The algorithm produces only the reduced transition function of the DFA.

PYTHON CODE



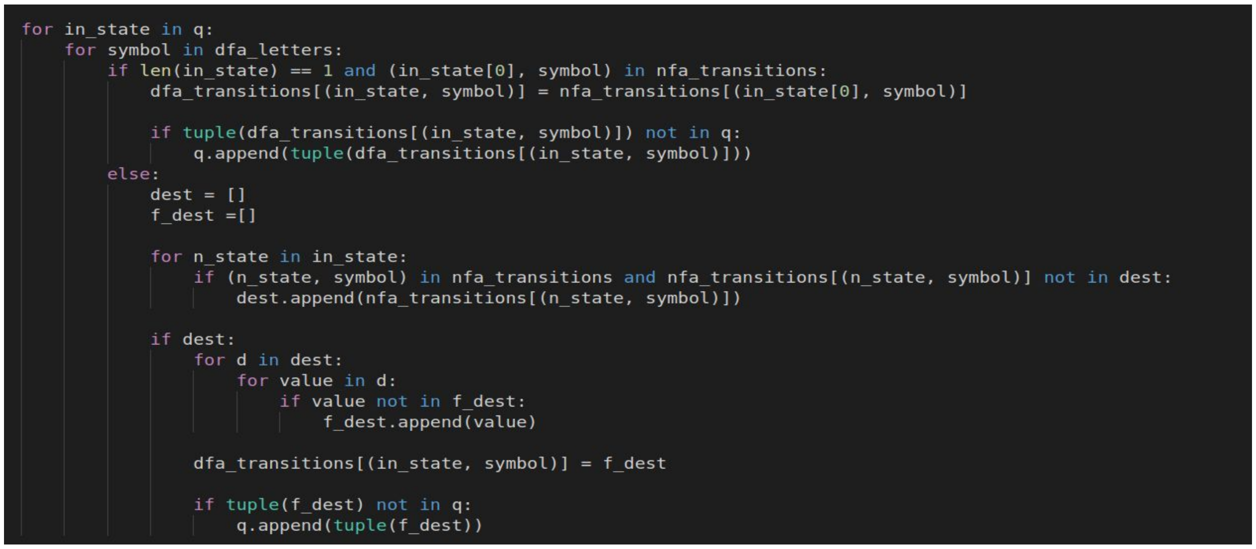
```python
import json
from collections import OrderedDict

with open('input.json') as file:
    data = json.load(file)
```

Python has a built-in package called JSON which can be used to work with JSON data. The JSON module can take a JSON string and convert it back to a dictionary structure. Reading in a JSON file uses the json.load() function.



```python
dfa_states = 2 ** data["states"]
dfa_letters = data["letters"]
dfa_start = data["start"]
dfa_t_func = []
dfa_final = []
dfa_list = []
q = []

q.append((dfa_start,))

nfa_transitions = {}
dfa_transitions = {}

for transition in data["t_func"]:
    nfa_transitions[(transition[0], transition[1])] = transition[2]
```

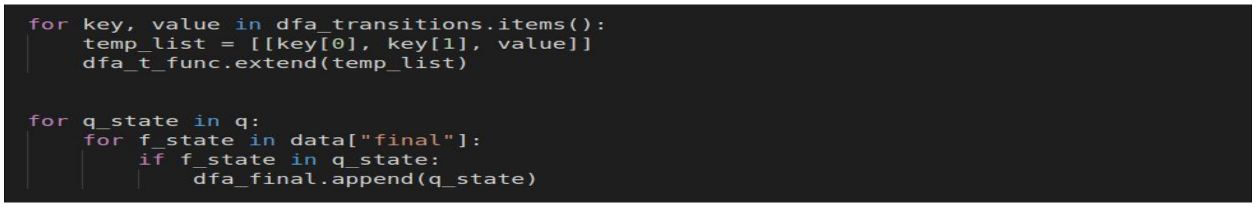
After initializing the variables required for describing the DFA, a Python dictionary is created to describe the transition relation of the NFA.



```
for in_state in q:
    for symbol in dfa_letters:
        if len(in_state) == 1 and (in_state[0], symbol) in nfa_transitions:
            dfa_transitions[(in_state, symbol)] = nfa_transitions[(in_state[0], symbol)]

            if tuple(dfa_transitions[(in_state, symbol)]) not in q:
                q.append(tuple(dfa_transitions[(in_state, symbol)]))
        else:
            dest = []
            f_dest =[]

            for n_state in in_state:
                if (n_state, symbol) in nfa_transitions and nfa_transitions[(n_state, symbol)] not in dest:
                    dest.append(nfa_transitions[(n_state, symbol)])

            if dest:
                for d in dest:
                    for value in d:
                        if value not in f_dest:
                            f_dest.append(value)

                dfa_transitions[(in_state, symbol)] = f_dest

                if tuple(f_dest) not in q:
                    q.append(tuple(f_dest))
```

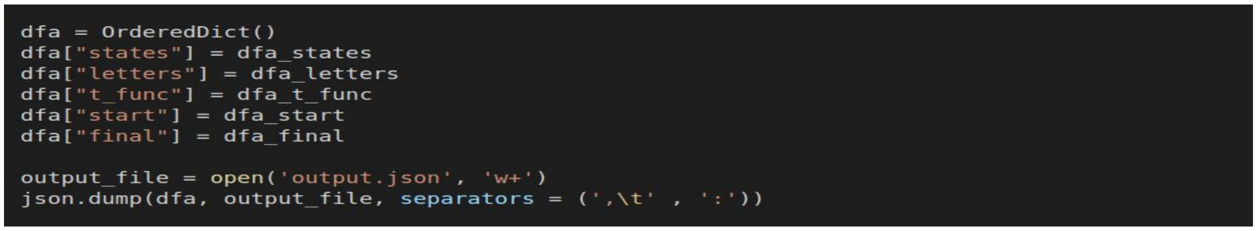
The above code snippet follows the algorithm mentioned in the previous page. Variable '*q*' denotes *Q'*. We have already added the start state of the NFA to the list.

For each element present in the list *q*, we iterate through the symbols of the alphabet to find the corresponding transition function which is represented as *<(input states)>, <symbol>, <(final states)>*. If the input state is a 1-member tuple, we just look for its transition relation in the NFA *t-func* list. Else, the final states will be the union of all the final states of each of the states in the input states tuple. In the end, we just add the final state tuple obtained into *q* if it's not present previously.



```
for key, value in dfa_transitions.items():
    temp_list = [[key[0], key[1], value]]
    dfa_t_func.extend(temp_list)


for q_state in q:
    for f_state in data["final"]:
        if f_state in q_state:
            dfa_final.append(q_state)
```

Convert the dictionary form of the DFA transition function to a list. Then, find the accepting states of the DFA, i.e., the states containing the accepting states of the NFA.



```
dfa = OrderedDict()
dfa["states"] = dfa_states
dfa["letters"] = dfa_letters
dfa["t_func"] = dfa_t_func
dfa["start"] = dfa_start
dfa["final"] = dfa_final

output_file = open('output.json', 'w+')
json.dump(dfa, output_file, separators = (',\t' , ':'))
```

Now, add the parameters for describing the DFA into a dictionary. Convert the dictionary into a JSON string and add it to *output.json* file.

A LOOK INTO THE SAMPLE RUN OF THE PROGRAM

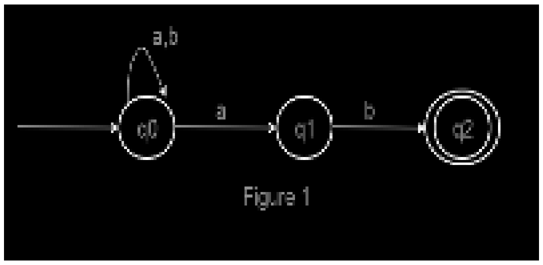
>> input.json

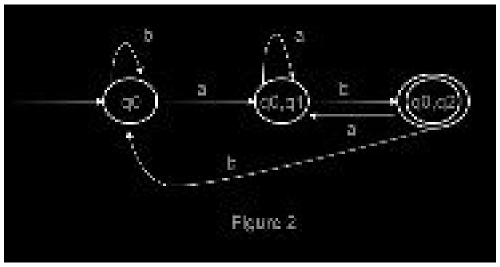
```
{
    "states": 3,
    "letters" : ["a","b"],
    "t_func" : [[0,"a",[0,1]], [0, "b", [0]], [1, "b", [2]]],
    "start" : 0,
    "final" : [1, 2]
}
```

>> output.json

```
{
    "states":8,
    "letters":["a", "b"],
    "t_func":[[[0,1], "a", [0,1]],  [[0], "b", [0]], [[0],"a", [0,1]], [[0, 2], "a", [0,1]], [[0,2], "b", [0]], [[0,1], "b",[0, 2]]],
    "start":0,
    "final":[[0,1], [0, 2]]
}
```

**Note that the DFA's transition function has the reduced state representation.*





(a) Figure 1 represents the input NFA without ε-transitions
(b) Figure 2 represents the desired output DFA.