Finite State Machine Minimization

—

Danielle Shwed (shweddc)
Xiaoying Wang (wangx64)
Shan Perera (pererali)

FSM Implementation

- Simple finite state machine model implemented in Python3
- Object Oriented
- 3 Classes: State, Transition, FiniteStateMachine
- The FiniteStateMachine class is the overall finite state machine object
- The State class is a simple class which stores the state's name
- The Transition class represents a transition rule

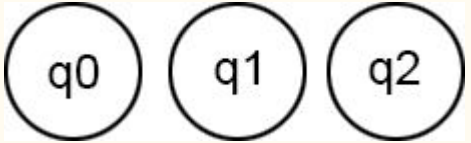
State Class

- Simple placeholder object that stores a state name

```
class State(object):
        def __init__(self, name):
                self.name = name;
```

```
newState = State('s0')
```


```
stateList = [State('q0'), State('q1'), State('q2')]
```



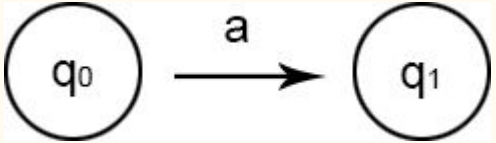
Transition Class

- A class that take the string arguments source, dest, and trigger
- Stores the valid input string required for a state machine to transition from the source node to the destination node
- Built-in transition function that checks if an input string has a valid transition

```
class Transition(object):
        def __init__(self, source, dest, trigger):

                self.source = source;
                self.dest = dest;
                self.trigger = trigger;
```

```
t1 = Transition('q0','q1','a')
```



FiniteStateMachine Class

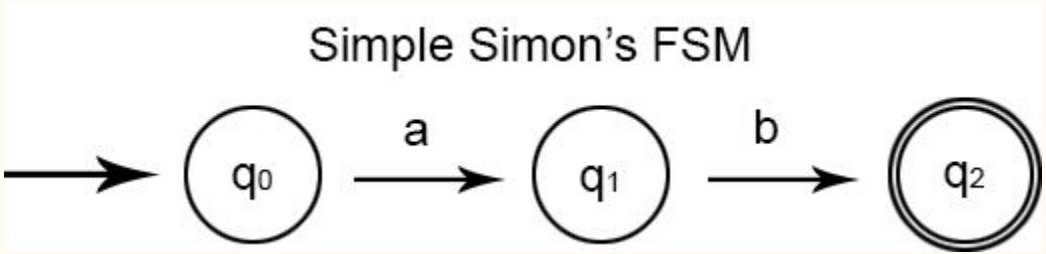
- Object class that represents the full finite state machine
- Requires the following input arguments:
  - name - String
  - states - [State]
  - initial - String
  - final - String
  - transitions - [Transition]
- Contains a global variable *currentState* to keep track of the current state of the FSM during input evaluation

FiniteStateMachine Class

```
class FiniteStateMachine(object):
        def __init__(self, name, states=None, initial='S', final='', transitions=None):
                global currentState;
                currentState = initial;

                … simple assignment statements follow...
```

```
x = [State("q0"), State("q1”), State("q2")]
y = [Transition('q0','q1','a'), Transition('q1', 'q2’, 'b')];
z = FiniteStateMachine("SimpleSimonsFSM", x, 'q0', 'q2', y);
```



Implementation Complexity

- Python3 implementation of Finite State Machines (Shan): ~100 lines of code
- FSM Visualization in HTML (Danielle): ~85 lines of code
- State Minimization Algorithm (Wendy): ~ 115 lines of code
- Overall Project: ~300 lines of code

Tested using manual test cases in IDLE

Visualization of FSM

- Embedded HTML in jupyter notebook
- HTML canvas with svg tags
- Clear cell and draw new shapes with changes applied for visualization effect
- Create Circle and Arrow classes

Visualization

Here’s a demo of the visualization….

State Minimization

State Minimization through Removing Redundant State(s)

1. Creating a state transition table
2. Identify which states have the same output behaviour
3. If a state transitions to the same next state, we call them equivalent
4. Combine these states into a single new renamed state
5. Repeat step 2 to 4 until we can no longer combine states into new states

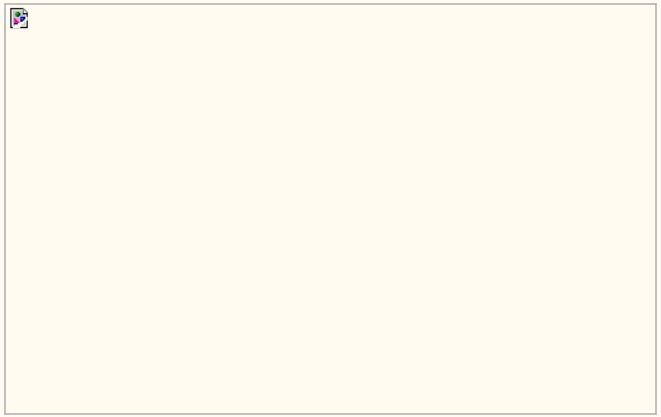
State Minimization

Step 1: Creating a state transition table

- Function genTransTable takes in a FiniteStateMachine object
  - def genTransTable(f)
- Returns a array called transTable

Example:

The transition table for the machine in picture is:



state	a	b
q1	q2	q3
q2	q2	q4
q3	q2	q3
q4	q2	q5
q5	q2	q3

State Minimization

Step2&3: Find the equivalent state(s)

- Function findEquivalent takes in a FiniteStateMachine object
  - def findEquivalent(f)
- Returns an array called result whereas each of its elements represents a set of state(s) that have the same transition output

Example:

The equivalent list for the machine is the picture is :

0 equivalent: [[q1, q2, q3, q4], [q5]]

1 equivalent: [[q1, q2, q3], [q4], [q5]]

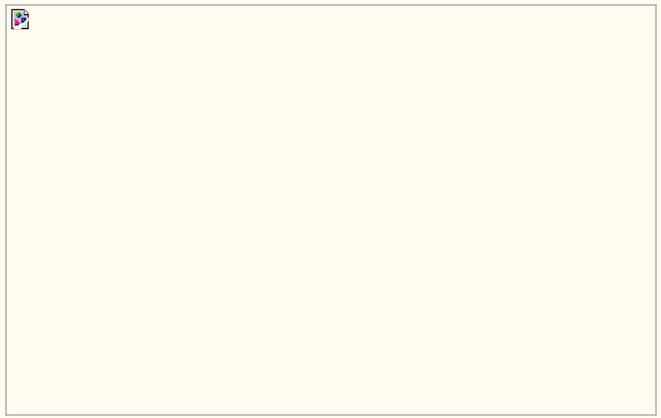
2 equivalent: [[q1, q3], [q2], [q4], [q5]]

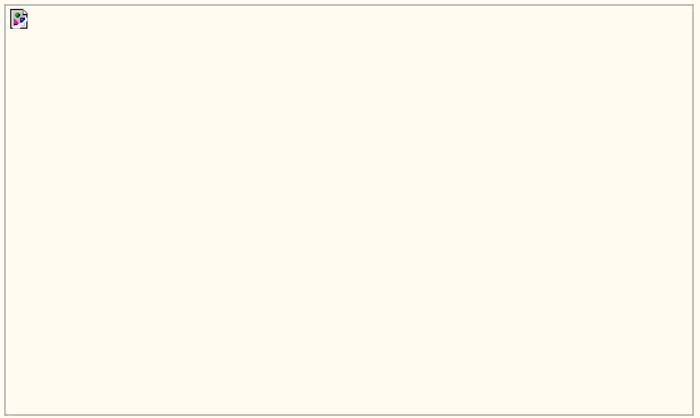
State Minimization

Step 4&5: Combines the equivalent state(s) and create a new machine

- Function minimization takes in a list of state and a FiniteStateMachine object
  - def minimization(newState, f)
  - Take the minimized state list and create the new transition list
  - Construct the new machine by using the new state list and new transition
- Returns a FiniteStateMachine object

Example:





State Minimization

Sample output from code

Create a FSM (DFA):

```
print("Creating the following States: q1, q2, q3, q4, q5");
x = [State("q1"), State("q2"), State("q3"), State("q4"), State("q5")];
y = [Transition('q1','q2','a'), Transition('q1', 'q3', 'b'), Transition('q2', 'q2', 'a'), Transition('q2', 'q4', 'b'),
     Transition('q3', 'q2', 'a'), Transition('q3', 'q3', 'b'), Transition('q4', 'q2', 'a'), Transition('q4', 'q5', 'b')
     Transition('q5', 'q2', 'a'), Transition('q5', 'q3', 'b')]
print("Creating Transitions for States:")
z = FiniteStateMachine_novisual("AlphabetSoup", x,'q1','q5' ,y);
```

Find the equivalent state(s) and construct the new DFA:

```
NewState = findEquivalent(z)
newMachine = minimization(NewState, z)
evaluateString(z, "abb")
evaluateString(newMachine, "abb")
```

State Minimization

Sample output from code continue...

```
Transitioning from state q1 to state q2
Transitioning from state q2 to state q2
Transitioning from state q2 to state q4
Transitioning from state q4 to state q5
The FSM has evaluated the string and is in an accepting state
The String "abb" is accepted by Finite State Machine "AlphabetSoup"
Transitioning from state q1q3 to state q2
Transitioning from state q2 to state q2
Transitioning from state q2 to state q4
Transitioning from state q4 to state q5
The FSM has evaluated the string and is in an accepting state
The String "abb" is accepted by Finite State Machine "MinimizedMachine"
```

Teaching Content

- Teaching lessons in regards to finite state machines
- Cells which user can type in a string they think will be accepted by the machine and see it visualized
- The additional content is great for students to understand the concept before implementing the code