Colleen Toth
Dan LeBlanc
CS311, Project 1 Write Up
24 October 2014

Originally, I had planned on writing my code in Java since I am fairly familiar with it and it has some nice features that make coding a bit friendlier than C++ or C. However, after giving some thought to how I would define my delta function, which is the meat of this project, I settled on Python because of it's dictionary structure. I defined the delta function by making my set of states a key and associated it with another set of keys that were the elements of my language. By doing this I could easily walk through my states and corresponding elements and store the resultant state precisely. This works because the dictionary acts as a hashmap. So when we match on a state key, it returns a value which is a set of keys we could match to, which matches on an element of the language, ultimately returning a new state, which then maps to another set of keys, and so on until we walk through the string we are checking. Creating the map was particularly easy once I found the fromkey() function that allowed me to create keys from my states and elements before I had the transitional mappings. This way I could prompt the user the for a specific resultant state and store it in the correct key-value pair by incrementing through the dictionary. The way python does this seems rather magical, which is in fact why I chose it to encode my DFA Runner.

Though the syntax took some getting used to, the fact that it is loosely typed worked great for being able to represent a variety of different typed elements in my language as a group. In a strongly typed language I would have needed to build a structure around it to associate these things together. The hashmap was also incredibly easy to implement in Python, which allowed me to focus on creating a working program and not on data structures. I really could almost type English into the terminal and Python would understand what I wanted. I did not need to manage how things were incrementing through the dictionary, how they were being stored, what the data structure looked like or how the actual hashing worked. I could simply write the code in a reasonable way and Python did most of the heavy lifting. It did take me awhile to figure out how to store things in the dictionary correctly, since I tried a complicated series of string splitting and for loops originally. Once I understood how to store things, however, I realized I could almost pidgeon hole (not related to the principle) the user into entering the new state exactly where I wanted them to, without complicated indexing.

I assumed users could follow directions and were fairly familiar with what a DFA is and how they work. I also assumed the user would not input wildly incorrect things. There was some checking to ensure the DFA was correct, and that states were in the set of specified states. Beyond that, I did very little to ensure correct user input. I did however, include the ability to check multiple strings on a defined DFA.

Output for even-odd dfa:

```
/System/Library/Frameworks/Python.framework/Versions/2.6/bin/python2.6 /Users/ctizzle/PycharmProjects/DFARunner/dfaRunner.py
Welcome to Colleen's DFA Runner Program


This program will determine whether a string you input
will be accepted by the DFA you specify. Please follow the
directions below.



How many elements are in the language? (sigma): 2
Please enter an element in the language: 0
Please enter an element in the language: 1



Enter the number of states: 2
Please enter the designation for each state: q0
Please enter the designation for each state: q1






Please enter the start state: q0



Enter the number of accepting states: 1
Please enter each accepting state: q1


Please enter the transition function for each state (i.e. q(n): a-q):
The state and the transition on an element are provided. Enter only the new state:


At state: q0 :  on 0 : q0
At state: q0 :  on 1 : q1
At state: q1 :  on 0 : q1
At state: q1 :  on 1 : q0
The DFA you entered is defined as:

Sigma:
['0', '1']
States:
['q0', 'q1']
Start state:
q0
Delta function:
[]
F:
['q1']




 Your DFA looks like



q1          {'1': 'q0', '0': 'q1'}
q0          {'1': 'q1', '0': 'q0'}

Please enter a string to check:
011

The final state is: q0 and the accepting state(s) are: ['q1']


String rejected



Would you like to check another string?
 y
```

```
Please enter a string to check:
010

The final state is: q1 and the accepting state(s) are: ['q1']


String accepted



Would you like to check another string?
 n

Process finished with exit code 0
```

Output for dfa that accepts a 1 in the third position:

```
/System/Library/Frameworks/Python.framework/Versions/2.6/bin/python2.6 /Users/ctizzle/PycharmProjects/DFARunner/dfaRunner.py
Welcome to Colleen's DFA Runner Program


This program will determine whether a string you input
will be accepted by the DFA you specify. Please follow the
directions below.



How many elements are in the language? (sigma): 2
Please enter an element in the language: 0
Please enter an element in the language: 1



Enter the number of states: 4
Please enter the designation for each state: q0
Please enter the designation for each state: q1
Please enter the designation for each state: q2
Please enter the designation for each state: q3






Please enter the start state: q0



Enter the number of accepting states: 1
Please enter each accepting state: q3


Please enter the transition function for each state (i.e. q(n): a-q):
The state and the transition on an element are provided. Enter only the new state:


At state: q0 :  on 0 : q1
At state: q0 :  on 1 : 1
State provided not in set of states.
At state: q0 :  on 1 : q0
At state: q1 :  on 0 : q2
At state: q1 :  on 1 : q0
At state: q2 :  on 0 : q2
At state: q2 :  on 1 : q3
At state: q3 :  on 0 : q3
At state: q3 :  on 1 : q3
The DFA you entered is defined as:

Sigma:
['0', '1']
States:
['q0', 'q1', 'q2', 'q3']
Start state:
q0
Delta function:
[]
F:
['q3']
```

```
 Your DFA looks like


q1          {'1': 'q0', '0': 'q2'}
q0          {'1': 'q0', '0': 'q1'}
q3          {'1': 'q3', '0': 'q3'}
q2          {'1': 'q3', '0': 'q2'}

Please enter a string to check:
011

The final state is: q0 and the accepting state(s) are: ['q3']


String rejected



Would you like to check another string?
 y

Please enter a string to check:
001

The final state is: q3 and the accepting state(s) are: ['q3']


String accepted



Would you like to check another string?
 y

Please enter a string to check:
00100

The final state is: q3 and the accepting state(s) are: ['q3']


String accepted



Would you like to check another string?
 n

Process finished with exit code 0
```

Output for at least one 1 :

```
/System/Library/Frameworks/Python.framework/Versions/2.6/bin/python2.6 /Users/ctizzle/PycharmProjects/DFARunner/dfaRunner.py
Welcome to Colleen's DFA Runner Program


This program will determine whether a string you input
will be accepted by the DFA you specify. Please follow the
directions below.



How many elements are in the language? (sigma): 2
Please enter an element in the language: 0
Please enter an element in the language: 1



Enter the number of states: 3
Please enter the designation for each state: q0
Please enter the designation for each state: q1
Please enter the designation for each state: q2





Please enter the start state: q0
```

```
Enter the number of accepting states: 1
Please enter each accepting state: q1


Please enter the transition function for each state (i.e. q(n): a-q):
The state and the transition on an element are provided. Enter only the new state:


At state: q0 :  on 0 : q0
At state: q0 :  on 1 : q1
At state: q1 :  on 0 : q2
At state: q1 :  on 1 : q1
At state: q2 :  on 0 : q1
At state: q2 :  on 1 : q1
The DFA you entered is defined as:

Sigma:
['0', '1']
States:
['q0', 'q1', 'q2']
Start state:
q0
Delta function:
[]
F:
['q1']




 Your DFA looks like


q1          {'1': 'q1', '0': 'q2'}
q0          {'1': 'q1', '0': 'q0'}
q2          {'1': 'q1', '0': 'q1'}

Please enter a string to check:
101

The final state is: q1 and the accepting state(s) are: ['q1']


String accepted



Would you like to check another string?
 y

Please enter a string to check:
0

The final state is: q0 and the accepting state(s) are: ['q1']


String rejected



Would you like to check another string?
 y

Please enter a string to check:
10

The final state is: q2 and the accepting state(s) are: ['q1']


String rejected



Would you like to check another string?
 n

Process finished with exit code 0
```