DFA Analyzer Writeup - CS311 - Leblanc

Ben Reichert

October 20, 2014

I chose Python for this programming assignment since I am familiar with it, it is simple, and not having to deal with memory management is nice. I also chose it since it has built in ways to parse CSV files, as well as tons of built in data structures that I could use to represent a DFA. I chose to encode the DFA as a csv file. I chose this because it was simple enough to represent the final DFA, and easy to parse on the Python side of things. I chose the following representation for the csv files:

```
Representing lines of the csv file format:
1. Language
2. Start State
3. any other states (any state not including the start state)
4. transitions between states(S->1->S,S->0->Q,Q->0->Q,Q->1->S) with the
    arrows
5. accepting states
6. string to test against the DFA
```

Here is a sample csv representing a simple DFA:

```
0,1
Q1
Q2,Q3,Q4
Q1->0->Q1,Q1->1->Q2,Q2->1->Q3,Q3->1->Q4,Q2->0->Q1,Q3->0->Q1,Q4->0->Q1,Q4->1->Q4a
Q4
00000000000000000001110
```

The DFA is loaded from the csv file into a dictionary with the key of the dictionary being the pair of values:state and the character to transition on. The value stored in that dictionary entry being the state to transition to. This way I can easily lookup the correct path to take within the DFA class to traverse between states. The DFA class is handed a string to test against the DFA and from that, it just traverses until there is no more characters in the string, and then checks if it landed on an accepting state, and returns True/False depending.

A major assumption I made is that I assume the user follows the csv format precisely, since I do absolutlely no error checking. If the user enters something corrupt in the csv file, too bad, and the program will probably blow up.I also

assumed that the csv file that needs to be read exists, and has a .csv extension. I did do some basic error checking that the filename has .csv, but no checking that it exists in the first place. I saw the need for error checking as something that could be implemented in the future, but is not particularly necessesary in this environment for this assignment. A problem with the csv format the way I implemented it, is that all the transition functions are on the same line, so even a simple 5 state DFA can create a very long, hard to read line. This could be improved if need be, but for basic implementation and functionality, this worked fine for this programming assignment.

Below are some sample inputs, and outputs:

```
$ cat dfa-not-in-language.csv
1,0
S
Q
S->1->S,S->0->Q,Q->0->Q,Q->1->S
S
1000010101010010100101001000101000000

$ ./dfa-analyzer.py dfa-not-in-language.csv
Welcome to the DFA-Analyzer
String is not in the language.

$ cat dfa.csv
1,0
S
Q
S->1->S,S->0->Q,Q->0->Q,Q->1->S
S
100001

$ ./dfa-analyzer.py dfa.csv
Welcome to the DFA-Analyzer
String is in the language.

$ cat csv-files/example1.11-page38.csv
a,b
s
q1,q2,r1,r2
s->a->q1,s->b->r1,q1->a->q1,r1->b->r1,q1->b->q2,q2->a->q1,q2->b->q2,r1->a->r2,r2->a->r2,r2->b->r1
q1,r1
bab

$ ./dfa-analyzer.py csv-files/example1.11-page38.csv
Welcome to the DFA-Analyzer
String is in the language.

$ cat csv-files/example1.11-page38-invalid.csv
a,b
```

```
s
q1,q2,r1,r2
s->a->q1,s->b->r1,q1->a->q1,r1->b->r1,q1->b->q2,q2->a->q1,q2->b->q2,r1->a->r2,r2->a->r2,r2->b->r1
q1,r1
bbba

$ ./dfa-analyzer.py csv-files/example1.11-page38-invalid.csv
Welcome to the DFA-Analyzer
String is not in the language.
```