CS2200 Homework 4

Evan Wilcox

Due March 19, 2019

1. Write a Python NDFSA+ϵ simulator.

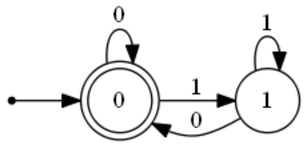
```python
# 1
# NDFSA+e Simulator
class Simulator():
    def __init__(self, file):
        f = open(file, "r")

        self.states = []
        self.delta = []
        outputs = []
        l = []

        for line in f:
            l.append(line)

            if line[0] == 'A':
                self.alphabet = line[2:]

            if line[0] == 'S':
                self.states.append([line[2:len(line)-4], int(line[len(line)-2:len(line)-1])])

            if line[0] == 'B':
                self.beginState = line[2:-1]

            if line[0] == 'D':
                s = line[2:line.index(',')]
                c = line[line.index(',')+2:line.index(',', line.index(',')+1)]
                e = line[line.index(',', line.index(',')+1)+2:-1]
                self.delta.append([s, c, e])

            if line[0] == 'T':
                t = line[2:-1]
                o = self.run(t)
                outputs.append(o)

        f.close()

        w = open(file, 'w')
        for line in l:
            if line[0] == 'O':
                line = line[:2] + outputs[0] + line[-1:]
                outputs = outputs[1:]

            w.write(line)

        w.close()

    def run(self, tape, ):
        state = self.beginState
        for c in tape:
            for d in self.delta:
                if d[0] == '@':
                    state == d[2]

                elif d[0] == state and d[1] == c:
                    state = d[2]

        if [state, 1] in self.states:
            return "Accepted"
        else:
            return "Rejected"
```

2. Write a program that can generate a Graphviz file from either a .fsa or .ndfsa file.

```python
# 2
class Generator():
    def __init__(self, file):
        f = open(file, "r")

        self.states = []
        self.delta = []

        for line in f:
            if line[0] == 'A':
                self.alphabet = line[2:]

            if line[0] == 'S':
                self.states.append([line[2:len(line)-4], int(line[len(line)-2:len(line)-1])])

            if line[0] == 'B':
                self.beginState = line[2:-1]

            if line[0] == 'D':
                s = line[2:line.index(',')]
                c = line[line.index(',')+2:line.index(',', line.index(',')+1)]
                e = line[line.index(',', line.index(',')+1)+2:-1]
                self.delta.append([s, c, e])

        f.close()

        output = file[:file.find('.')] + ".dot"
        tab = "    "
        w = open(output, 'w')

        w.write("digraph finite_state_machine {\n")
        w.write(tab + 'rankdir=LR;\n')
        w.write(tab + '_ize="8,5"\n\n')
        w.write(tab + 'node [shape = point] x\n')

        for state in self.states:
            w.write(tab + "node [shape = ")
            if state[1]:
                w.write("doublecircle] " + state[0] + "\n")
            else:
                w.write("circle] " + state[0] + "\n")

        w.write("\n" + tab + "x -> " + self.beginState + "\n")

        for delta in self.delta:
            w.write(tab + delta[0] + " -> " + delta[2] + ' [label = "')
            for d in self.delta:
                if delta[0] == d[0] and delta[2] == d[2] and delta != d:
                    delta[1] += (", " + d[1])
                    self.delta.remove(d)

            w.write(delta[1] + '"]\n')

        w.write("}")

        w.close()
```

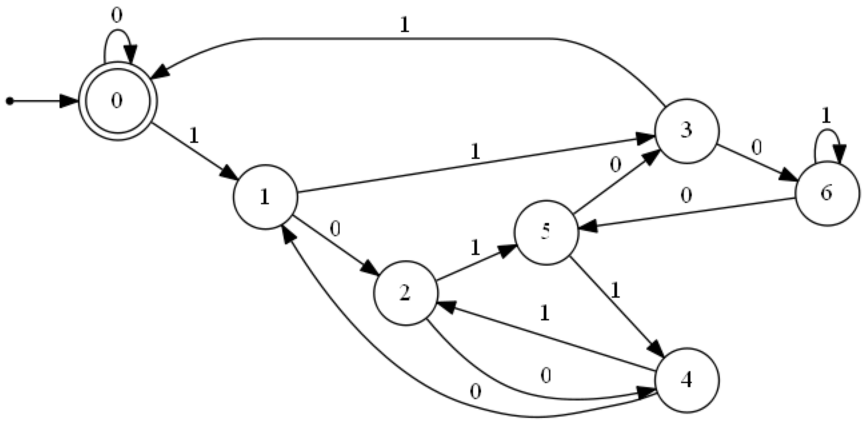
4. For each of the following, find a FSA automaton that recognizes the language or prove that there is no FSA that recognizes the language.

   (a) L_1 = { all binary strings divisible by 2 }




```
A 01
S 0, 1
S 1, 0
B 0
D 0, 0, 0
D 0, 1, 1
D 1, 0, 0
D 1, 1, 1
T
O Accepted
T 1
O Rejected
T 0
O Accepted
T 11
O Rejected
T 00
O Accepted
T 10
O Accepted
T 01
O Rejected
T 100101010100
O Accepted
T 0011001101001101
O Rejected
T 111111100000000110
O Accepted
```

(b) $L_2 = \{$ all binary strings divisible by 7 $\}$




```
A 01
S 0, 1
S 1, 0
S 2, 0
S 3, 0
S 4, 0
S 5, 0
S 6, 0
B 0
D 0, 0, 0
D 0, 1, 1
D 1, 0, 2
D 1, 1, 3
D 2, 0, 4
D 2, 1, 5
D 3, 0, 6
D 3, 1, 0
D 4, 0, 1
D 4, 1, 2
D 5, 0, 3
D 5, 1, 4
D 6, 0, 5
D 6, 1, 6
T
O Accepted
T 1
O Rejected
T 0
O Accepted
T 111
O Accepted
T 110
O Rejected
T 1110
```

```
O Accepted
T 11101
O Rejected
T 11100
O Accepted
T 0011001101001100
O Accepted
T 111111100000011111
O Accepted
```

(c) $L_3 = \{$ all unary strings that represent prime numbers $\}$

Let $w = 1^p$ where p is a prime number. $w \in L_3$. L_3 can not be represented by a FSA because FSA can only represent regular languages and L_3 does not produce a regular language. We know this because using the Pumping Lemma, 1^p can be pumped.

(d) $L_4 = \{$ all unary strings that represent composite numbers $\}$

Let $w = 1^p$ where p is a composite number. $w \in L_3$. L_3 can not be represented by a FSA because FSA can only represent regular languages and L_3 does not produce a regular language. We know this because using the Pumping Lemma, 1^p can be pumped.

(e) $L_5 = \{$ w $\in$ {a, b, c}* such that w is a palindrome $\}$

Using the pumping lemma, a^n can be pumped to create an infinite length palindrome.