CSC236 fall 2014, Assignment 2

Sample solution for DFSA

1. Read dfsa.py, and try running it. Once you understand it, create new DFSA instances:

   times_three_a: Accepts lists of Symbols such that: the number of **a**s in the list is a multiple of 3. Rejects all other lists.

   first_last_a: Accepts lists of Symbols such that: the first and last symbol is an **a**. Rejects all other lists.

   Sample solution: For **times_three_a**, we need to keep track of whether we have seen a multiple of 3 plus a remainder of 0, 1, or 2 **a**'s. We accept any list that leaves us with a remainder of 0:

```
class State(Enum):
    '''States for times_three'''
    zero = 0
    one = 1
    two = 2
    dead = -1
def t(s, symb):
    '''(State, Symbol) -> State
    Return the state reached from State s using Symbol symb.
    '''
    if not symb in [Symbol.a, Symbol.b]:
        return State.dead
    elif s == State.zero and symb == Symbol.a:
        return State.one
    elif s == State.one and symb == Symbol.a:
        return State.two
    elif s == State.two and symb == Symbol.a:
        return State.zero
    else:
        return s
# a DFSA that accepts lists with a multiple of 3 a's
times_three_a = DFSA(State.zero, [State.zero], t)
input = []
assert times_three_a.accepts(input) # expect True
input.append(Symbol.b)
assert times_three_a.accepts(input) # expect True
input.append(Symbol.a)
assert not times_three_a.accepts(input) # expect False
input += [Symbol.a, Symbol.b]
assert not times_three_a.accepts(input) # expect False
input.append(Symbol.a)
assert (times_three_a.accepts(input)) # expect True
```

For **first_last_a**, we reject any list that doesn't begin with an **a**, and keep track of whether we have just seen an **a** or a **b**. Of the lists that weren't rejected for failing to start with an **a**, we accept those that leave us having just seen an **a**:

```
class State(Enum):
    '''States for first_last_a'''
    start = 0
    got_a = 1
    got_b = 2
    dead = -1
def t(s, symb):
    '''(State, Symbol) -> State
    Return new State reached from State s with Symbol symb.
    '''
    if s == State.start and not symb == Symbol.a:
        return State.dead
    elif symb == Symbol.a:
        return State.got_a
    elif symb == Symbol.b:
        return State.got_b
    else:
        return State.dead
# a DFSA that accepts lists that start and end with 'a'
first_last_a = DFSA(State.start, [State.got_a], t)
input = []
assert not first_last_a.accepts(input) # expect False
input.append(Symbol.b)
assert not first_last_a.accepts(input) # expect False
input = [Symbol.a]
assert first_last_a.accepts(input) # expect True
input += [Symbol.a, Symbol.b]
assert not first_last_a.accepts(input) # expect False
input.append(Symbol.a)
assert first_last_a.accepts(input) # expect True
```