CS 3100 – Models of Computation – Fall 2011
Assignment 2 solution

Powerset.py Write a python function to compute the powerset of a given set or list (the function should work for both; hint: do a list(S) inside the function). Return a **list of lists**.

```
def pow(S):
    """Powerset of a set L. Since sets/lists are unhashable, we convert the set to a list,
    perform the powerset operations, leaving the result as a list (can't convert back to a set).
    pow(set(['ab', 'bc'])) --> [['ab', 'bc'], ['bc'], ['ab'], []]
    """
    L=list(S)
    if L==[]:
        return([[]])
    else:
        pow_rest0 = pow(L[1:])
        pow_rest1 = list(map(lambda ls: [L[0]]+ls, pow_rest0))
        return(pow_rest0 + pow_rest1)
```

MkDFA.py Define a function `mk_dfa` whose definition is sketched below. A DFA is represented using a dict of the form:

```
{"Q":Q, "Sigma":Sigma, "Delta":Delta, "q0":q0, "F":F})
```

Here, `Q` is a **non-empty** set of strings (state names), `Sigma` is a set of **non-empty single-character strings** (alphabet), `q0` is a **state belonging to** `Q`, and `F` is a **possibly non-empty set of states, and is also a subset of** `Q`. `Delta` is a **total** function represented as a hash-table, mapping a pair (q, c) (**where** `q` **in** `Q` and `c` **in** `Sigma`**)** to a new state $q1$ where `q1` **is also in** `Q`.

Implement all the checks in **boldface** font given above as `asserts` in Python. Test that all the checks are working. Submit this terminal session of the checks happening as file `MkDFATests.txt`.

```
def fst(p):
    """ First of a pair."""
    return p[0]

def snd(p):
    """ Second of a pair."""
    return p[1]

def fn_dom(F):
    """ For functions represented as hash-maps (dicts), return their domain as a set.
    """
    return {k for k in F.keys()}

def fn_range(F):
    """ For functions represented as hash-maps (dicts), return their range as a set.
    """
    return {v for v in F.values()}
```

```
# More checks are possible perhaps... but here is most
def mk_dfa(Q, Sigma, Delta, q0, F):
    """Make a DFA with the given traits. Delta is supplied as a hash-map (dict).
    """
    assert(Sigma != {})
    #
    # Sigma is a set of strings of length 1. This check does it in one line
    assert(not(False in list(map(lambda x: len(x)==1, Sigma))))
    #
    # Delta must be a total function
    dom = fn_dom(Delta)
    states_dom = set(map(fst,dom))
    input_dom = set(map(snd,dom))
    state_targ = set(fn_range(Delta))
    #-- num state and input entries to match
    assert(states_dom == Q)
    assert(input_dom == Sigma)
    #
    #-- Mapping for every pair must be present
    assert(len(Delta)==len(Q)*len(Sigma))
    #
    # Targets must be in Q and non-empty
    assert((state_targ <= Q)&(state_targ != {}))
    # Initial state in Q
    assert(q0 in Q)
    # Final states subset of Q (could be empty, could be Q)
    assert(set(F) <= Q)
    # If all OK, return DFA as a dict
    return({"Q":Q, "Sigma":Sigma, "Delta":Delta, "q0":q0, "F":F})
```

DFA0101.py (**30 points**)

Design (on paper) a DFA that accepts all strings over $\Sigma = \{0, 1\}$ that end in 0101.

Solution: Here is the solution outline with comments - after this, I provide the real details of the DFA coding.

```
This is only my "sketch" as I would sketch on paper. The real DFA coding
begins after this.

       -> s0 # Start from

s0 --1--> s0 # In state s0, upon a 1, stay in s0
s0 --0--> s1 # upon a 0, move to s1

s1 --0--> s1 # in s1, another 0 is not helping move along. stay in s1
s1 --1--> s2 # Move on 1 to s2

s2 --1--> s0 # Upon 1 fail back to s0
s2 --0--> s3 # upon 0, make progress to s3

s3 --0--> s1 # upon a 0, fail but don't go clear back; go upto s1 as we can "reuse" this 0
             # toward a 0101
```

```
s3 --1--> s4 # finally go to s4

s4 --0--> s3 # fail to s3 because we can have 010101 etc, and the third 0 is a promising 0
             # because it may help advance 01010 to a 010101
s4 --1--> s0 # fail back to s0

Q1 = {'S0','S1','S2','S3','S4'}

Sigma1 = {'0','1'}

Delta1 = {('S0', '0'): 'S1',
          ('S0', '1'): 'S0',
          ('S1', '0'): 'S1',
          ('S1', '1'): 'S2',
          ('S2', '1'): 'S0',
          ('S2', '0'): 'S3',
          ('S3', '0'): 'S1',
          ('S3', '1'): 'S4',
          ('S4', '0'): 'S3',
          ('S4', '1'): 'S0'}

q01 = 'S0'

F1 = {'S4'}
```

The generated DFA is