CS 3100 – Models of Computation – Fall 2011
This assignment is worth 8% of the total points for assignments
100 points total

September 7, 2011

Assignment 3, Posted on: 9/6 Due: 9/15 Thursday 11:59pm

1. (**20 points**) Write a Python function `recognizes(D, N)` that returns all strings of length $0 \leq i \leq N$ recognized by the given DFA `D`. Assume that $N \geq 0$. Test it out on the the DFA that recognizes all strings ending in 0101 that you constructed in Assignment 2 for $N = 5$. Submit the function in a file `recognizes.py` as well as an ASCII record of your testing session as file `recognizes_tests.out`.

   Solution:

```
# The solution is below.

from math import *

from lang import *

from dfa import *

def nthnumeric(N):
    """Assume that Sigma is {a,b}. Produce the Nth string in numeric order, where N >= 0.
    Idea : Given N, get b = floor(log_2(N+1)) - need that many places; what to
    fill in the places is the binary code for N - (2^b - 1) with 0 as a and 1 as b.
    """
    if(N==0):
        return ''
    else:
        width = floor(log(N+1, 2))
        tofill = int(N - pow(2, width) + 1)
        relevant_binstr = bin(tofill)[2::] # strip the 0b leading string
        len_to_makeup = width - len(relevant_binstr)
        return "a"*len_to_makeup + homos(relevant_binstr, lambda x: 'b' if x=='1' else 'a')

def listall(D, frm, S):
    """Search in the nthnumeric order from 'frm' back through 0,
    exiting at -1. frm guaranteed to be >= 0. S guaranteed to be called
```

```
    with set({}).
    """
    if (frm == -1):
        return S
    else:
        nth_str = nthnumeric(frm)
        if accepts(D, D["q0"], nth_str):
            return listall(D, frm-1,  S | { nth_str })
        else:
            return listall(D, frm-1,  S)

def lang_lt_n(D, N):
    """Given a DFA D, find all strings of length <= N accepted by D.
    Strings listed in numeric order are:
    "", a, b, aa, ab, ba, bb, aaa, aab, ..., bba, bbb, aaaa, ...
    In this listing, note that the ordinal position of "" is 0,
    of a is 1, etc. Now all strings of length <= N are obtained
    by searching for strings in the nthnumeric enumeration from
    2^(N+1) - 2. For instance, all strings of length 3 or less
    are obtained by looking from 14 downwards in the nthnumeric
    listing.
    """
    ordinal_from = pow(2, N+1) - 2
    return listall(D, ordinal_from, set({}))


>>> DFA1
{'Q': {'S1', 'S0'}, 'q0': 'S0', 'F': {'S1'}, 'Sigma': {'a', 'b'}, 'Delta': {('S0', 'a'): 'S0', ('S1

>>> lang_lt_n(DFA1,4)
lang_lt_n(DFA1,3)
{'abb', 'ab', 'bab', 'bb', 'aab', 'b', 'bbb'}

>>> DFA1.update( { 'F' : {'S0', 'S1'}})

>>> DFA1
DFA1
{'Q': {'S1', 'S0'}, 'q0': 'S0', 'F': {'S1', 'S0'}, 'Delta': {('S0', 'a'): 'S0', ('S1', 'a'): 'S0',
>>>

lang_lt_n(DFA1, 4)
lang_lt_n(DFA1, 4)
{'baba', 'abab', 'aa', 'babb', 'abbb', 'abba', 'bbab', 'aaba', 'aabb', '', 'abb', 'aaaa', 'abaa', '
>>>
```

2. (**40 points**) Define a DFA that accepts all strings over {0,1} such that every block of four consecutive positions contains at least two 0s. (This means: **If** there are four consecutive positions, **Then** in those four positions, there must be at least two 0s.) Call this language L_{00}. Build this DFA using the `mk_dfa` call (we will supply you a working `mk_dfa` for this assignment). Next, use `dot_dfa` and print this DFA out. Submit the PDF drawing of this DFA, as file `L00.pdf`. Test this DFA on 12 strings including two (2) strings of length < 5, five (5) strings that are accepted and of length ≥ 6 and five (5) strings that are rejected and of length ≥ 6. Submit an ASCII record of your testing session as file `L00_tests.out`.

   Solution:

```
Here is how you do your work!
--------------------------------------------


S -0-> S0
S -1-> S1

S0 -0-> S00
S0 -1-> S01

S1 -0-> S10
S1 -1-> S11

S00 -0-> S000
S00 -1-> S001

S01 -0-> S010
S01 -1-> S011

S10 -0-> S100
S10 -1-> S101

S11 -0-> S110
S11 -1-> S111

S000 -0-> S0000
S000 -1-> S0001

S001 -0-> S0010
S001 -1-> S0011

S010 -0-> S0100
S010 -1-> S0101

S011 -0-> S0110
S011 -1-> BH
```

```
S100 -0-> S1000
S100 -1-> S1001

S101 -0-> S1010
S101 -1-> BH

S110 -0-> S1100
S110 -1-> BH


S0000 -0-> S0000

S0001 -1-> S0011

S0010

S0011

S0100

S0101

S0110

S1000

S1001

S1010

S1100


Once they get this trick, they fan finish up!
```

3. (**20 points**) Draw a DFA for Question 3 of `notes5.pdf`. Next, enter this DFA and generate a PDF drawing for it. Argue why this DFA works (in about 3-4 sentences), and also use function `accepts` to demonstrate that indeed it works on five (5) strings in the language and five (5) strings not in the language. Submit your PDF as `notes5_qn3_DFA.pdf` and your writeup as `notes5_qn3_DFA.out`.

   Solution:

   This question asks: Define a DFA that accepts all strings over {0,1} fed LSB-first such that these strings when interpreted according to standard binary conventions defines numbers which are evenly divisible by

3.

This is built by solving a recurrence.

```
N, 2^n --0--> N, 2^(n+1)

N, 2^n --1--> N + 2^n,  2^(n+1)

--

N%3, 2^n --1--> (N + 2^n)%3,  2^(n+1)

--

We need to remember only (2^n)%3

--

N%3,  (2^n)%3 --1--> (N%3 + (2^n)%3)%3,  (2 * (2^n)%3)%3

--

(0, 1) --0--> (0, 2)
(0, 1) --1--> (1, 2)

(0, 2) --0--> (0, 1)
(0, 2) --1--> (2, 1)

(1, 2) --0--> (1, 1)
(1, 2) --1--> (0, 1)

(2, 1) --0--> (2, 2)
(2, 1) --1--> (0, 2)

etc.


If we did MSB-first, the recurrence is easier

N -> 2*N + b

N%3 -> ((2 * N%3)%3 + b)%3
```

4. (**20 points**) Draw a DFA for Question 5 of `notes5.pdf`. Next, enter this DFA and generate a PDF drawing for it, and submit it. Argue why this DFA works (in about 3-4 sentences), and also use function

`accepts` to demonstrate that indeed it works on five (5) strings in the language and five (5) strings not in the language. Submit your PDF as `notes5_qn5_DFA.pdf` and your writeup as `notes5_qn5_DFA.out`.

Solution:

This question asks: Define a DFA for the language defined by the concatenation of the languages denoted by DFA of the two figures in those notes. Basically it is the concatenation of "all strings ending in 1" and "all equal 0-1 changes". This is all strings containing a 1. Why? Because if there is no 1, then we can't be a concat. If there is a 1, then there is a last 1. Pick that: the rest of the strings have equal changes. Now draw the DFA and solve easily!