Logical Models

First we must translate the function file to Cython.

This is not fully automated yet. First you use the script regular-expression.py.

```
python reg-expression.py func-example.txt example.pyx
```

The output file should look like this.

```
from __future__ import division   return vector
from __future__ import division
import numpy as np
cimport numpy as np
cimport cython

DTYPE = np.int
ctypedef np.int_t DTYPE_t
@cython.boundscheck(False)
def function(np.ndarray[DTYPE_t, ndim=1] x):
    cdef int vectorsize = x.shape[0]
    cdef np.ndarray[DTYPE_t, ndim=1]    vector = np.zeros([vectorsize],dtype=int)
    vector[0] = x[1]
    vector[1] = x[0]
    vector[2] = 2+2*x[0]+x[0]*x[0]*x[1]+x[0]*x[1]*x[1]+x[0]*x[0]*x[1]*x[1]
    for i in xrange(0,len(vector)):
        while vector[i] > 3:
            vector[i] = vector[i]%3
    return vector
```

Compiling

We must now compile the cython code.

```
cython example.pyx

gcc -shared -pthread -fPIC -fwrapv -O2 -Wall -fno-strict-aliasing
-I/usr/include/python2.7 -o functions.so functions.c
```

Running on single processor computer

Now that the code is compiled, we can run it. The main program is called main_attractor_synch_cython.pyFrom command line

```
usage: main_attractor_synch_cython.py [-h] [-n NUMBERSTATES] [-nn NUMBERNODES]
                                      [-s START] [-e END] [-f FUNCTION]
                                      [-v VERBOSE]

optional arguments:
  -h, --help            show this help message and exit
  -n NUMBERSTATES, --numberstates NUMBERSTATES
                        provide a number of states
  -nn NUMBERNODES, --numbernodes NUMBERNODES
                        provide a number of nodes
  -s START, --start START
                        starting string to convert to base Nstates
  -e END, --end END     ending string to convert to based Nstates
  -f FUNCTION, --function FUNCTION
                        function to run simulation on
  -v VERBOSE, --verbose VERBOSE
                        if you want verbose updates (use with single
                        processor)
```

Thus we would run the program

```
python main_attractor_synch_cython.py -n 3 -nn 3 -s 0 -e 27 -v -f example
```

```
pinojc@LoLab-760:~/Projects/LogicModel$ python main_attractor_synch_cython.py -n 3 -nn
3 -s 0 -e 27 -v 1 -f example
example
Started
0 / 27
1 / 27
2 / 27
3 / 27
4 / 27
5 / 27
6 / 27
7 / 27
8 / 27
9 / 27
10 / 27
11 / 27
12 / 27
13 / 27
14 / 27
15 / 27
16 / 27
17 / 27
18 / 27
19 / 27
20 / 27
21 / 27
22 / 27
23 / 27
24 / 27
25 / 27
26 / 27
time of 27 calculations 7.89165496826e-05 minutes
{'011': 6, '121': 6, '002': 3, '111': 3, '020': 6, '222': 3}
```

Running on multiple processors/across nodes

To run on a multi core or cluster, you need to comment the main() (line 105) and uncomment all the multiprocessing lines below that. In python it is just the two lines that contain """, line 111 and 191.
Then you run using mpirun

```
mpirun -n #proc /path/to/python main_attractor_synch_cython.py -n 3 -nn 21 -s 0 -e
1000
```

There is a script that creates the PBS files. The file is currently set up to run the large model. Later I will make it to automatically split up based on the number of jobs you want and the number of states to sample.