Doing Experiments with the Cybernetic Automaton

The cybernetic automaton model has been reimplemented in Go and is currently available on tux. The directory:

```
/home/bls96/ca
```

has the source code, configurations, and driver scripts for most of the examples described in the dissertation on the subject. In addition to that directory, there's a copy of the executable in the

```
/home/bls96/bin
```

directory.

Model Implementation/Experimental Subject

The implementation of the model is embodied in a simple executable program where the standard input receives stimuli in the form of lists of input symbols and strengths. Similarly, the standard output presents the output symbols and strengths. The usage for the executable is:

```
ca [-d] cfg
```

where the `-d` option turns on some debugging output that goes to the standard error stream. The configuration file which appears as the required command line argument is described in the next section.

Configuration File

The configuration file gives the model parameters and a description of the initial state machine. Model parameters are given in the first seven lines of the file and must be specified in the same order as are given in the tables in the dissertation. In particular the first seven lines of the file give the values for α, β, γ, η, ζ, ν, and κ in that order. The eight line of the file gives the number of initial states, the ninth, the size of the input alphabet Σ, and the tenth the size of the output alphabet Δ. Lines 11 and 12 list the states of the initial machine that are in the sets R and P, respectively. For these, state numbers are separated by spaces, and a blank line indicates an empty set.

All remaining lines give the transitions of the initial state machine. The first four values on a transition line are the initial state, the terminal state, the input symbol, and the confidence value. Values on the remainder of the line are pairs giving an output symbol and its probability.

As an example, the following configuration file describes the initial state machine and the parameters for the Skinner box experiment described in Chapter 5.

```
0.05
0.05
0.001
1.0
0.1
0.7
0.9
10
10
5
9

0 1 1 0.1 0 0.2 1 0.2 2 0.2 3 0.2 4 0.2
0 2 2 0.1 0 0.2 1 0.2 2 0.2 3 0.2 4 0.2
0 3 3 0.1 0 0.2 1 0.2 2 0.2 3 0.2 4 0.2
0 4 4 0.1 0 0.2 1 0.2 2 0.2 3 0.2 4 0.2
0 5 5 0.1 0 0.2 1 0.2 2 0.2 3 0.2 4 0.2
0 6 6 0.1 0 0.2 1 0.2 2 0.2 3 0.2 4 0.2
0 7 7 0.1 0 0.2 1 0.2 2 0.2 3 0.2 4 0.2
0 8 8 0.1 0 0.2 1 0.2 2 0.2 3 0.2 4 0.2
0 9 9 0.1 0 1.0
9 0 0 1000000.0 0 1.0
```

Driver Programs

Where the `ca` program is the organism that's the subject of the experiment, the experimental protocol and the environment are most easily implemented by means of driver programs. These programs run `ca` as a child process, provide stimuli by way of `ca`'s standard input, and take and process the responses from `ca`'s standard output.

All of the examples currently available are written in Python to go along with your CS171/172 experience. The `Popen` function in the `subprocess` package can be used to start the `ca` program as a child process. The process object that is returned contains two other objects `stdin` and `stdout` that are open files to the pipes created for the child process's standard input and output, respectively. A write to the process's standard input allows you to present stimuli, and reads from the process's standard output give you the responses.

The following script runs a simple case of delayed first-order conditioning:

```
#!/usr/bin/python
from subprocess import *

pid = Popen(["/home/bls96/bin/ca", "basic.cfg"], stdin=PIPE, stdout=PIPE)

print 0, 0
for i in xrange(10):
    n = 0
    for j in xrange(10):
        pid.stdin.write("3/1\n")
        x = pid.stdout.readline()
        if x[0] == "1":
            n += 1
        pid.stdin.write("1/3\n")
        pid.stdout.readline()
        pid.stdin.write("4/1\n")
        pid.stdout.readline()
        pid.stdin.write("2/1\n")
        pid.stdout.readline()
        pid.stdin.write("0/1\n")
        pid.stdout.readline()
    print i+1, n
pid.stdin.close()
```

In cases of instrumental conditioning, the driver program also simulates the state of the environment. This requires using the subject's responses to update the external state of the subject and to control which stimuli are provided. An example of this is shown below for the simple two-position Skinner box.

```
#!/usr/bin/python
from subprocess import *
import random
import sys

#
# 1:1N, 2:1E, 3:1S, 4:1W, 5:2N, 6:2E, 7:2S, 8:2W, 9:F
#
# 1:Left, 2:Right, 3:Move, 4:Press
#

pid = Popen(["/home/bls96/bin/ca", "skinner.cfg"], stdin=PIPE, stdout=PIPE)

for i in xrange(500):
```

```
d = random.random()
if d > 0.5:
    loc = 2
else:
    loc = 1
d = random.random()
if d > 0.75:
    dir = 1
elif d > 0.5:
    dir = 2
elif d > 0.25:
    dir = 3
else:
    dir = 4
sys.stderr.write(str(i) + '\n')
for j in xrange(75):
    pid.stdin.write(str((loc - 1) * 4 + dir) + '/1\n')
    x = pid.stdout.readline()
    if x[0] == '1':
        dir -= 1
        if dir == 0:
            dir = 4
    elif x[0] == '2':
        dir += 1
        if dir > 4:
            dir = 1
    elif x[0] == '3':
        if loc == 1 and dir == 2:
            loc = 2
        elif loc == 2 and dir == 4:
            loc = 1
    elif x[0] == '4':
        if loc == 2 and dir == 2:
            pid.stdin.write("9/1\n")
            pid.stdout.readline()
            break
pid.stdin.write("0/1\n")
pid.stdout.readline()
print i, j
```