Complexity Lab 4: Applications

last edited on June 30, 2011 03:35 PM by autoplectic


```
%hide
```

Let's explore inferring machines from data. It is generally wise to start with a problem that you already know the answer to, so let's begin with the golden mean process.

First lets generate some data from the golden mean process and print the first 100 symbols:

```
gm_symbols = get_symbols(goldenmean, length=1000)
print ''.join(gm_symbols[:70])
```

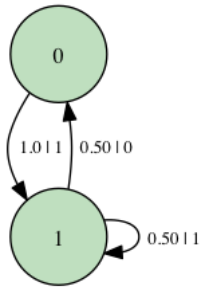
```
10101101011111010101011010101101101010101111011101010110101101010101 10
```

Let's see if we can infer the golden mean from it's symbols. We will use the tree merging algorithm with a morph length of 3 and a tree depth of 5:

```
gm_machine = infer_machine(gm_symbols, morph_length=3, tree_depth=5)
```

Draw the inferred machine to see what it looks like.

```
gm_machine.draw()
```



That's pretty close to what we started with. Now let's look at some of the machine properties of that inferred machine:

```
process_table([gm_machine], ['hmu', 'E', 'Cmu', 'R'])
```

	hμ	E	Cμ	R
Inferred Machine	0.66863	0.24765	0.91628	1

Now you try that with the even process.

Generate some symbols. 10,000 should be enough:

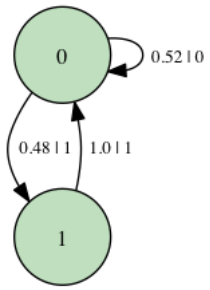
```
even_symbols = get_symbols(even, length=10000)
```

Now infer the machine. Like the golden mean, a morph length of 3 and a tree depth of 7 should be sufficient.

```
even_inferred = infer_machine(even_symbols, morph_length=3, tree_depth=7)
```

Now draw the inferred machine:

```
even_inferred.draw()
```



Use process_table to see what its various properties are:

```
process_table([even_inferred], ['hmu', 'E', 'Cmu', 'R'])
```

	hμ	E	Cμ	R
Inferred Machine	0.67363	0.91061	0.91061	∞

Now that we know we can reliably infer machines when we already know the epsilon machine, let's try getting epsilon machines for the logistic map at various values of r.

Girst, we generate 100000 symbols from the logistic map at r = 4.0:

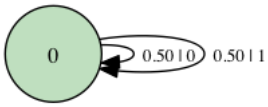
```
logistic_4_symbols = get_symbols(logistic(4.0), length=10000)
```

Now we infer the machine just like before, with a morph_length of 3 and a tree_depth of 7:

```
logistic_4 = infer_machine(logistic_4_symbols, morph_length=3, tree_depth=7)
```

Let's see what it looks like:

```
logistic_4.draw()
```



A fair coin! Is that what you expected?

For the sake of completeness, let's compute some process properties from the inferred machine:

```
process_table([logistic_4], ['hmu', 'E', 'Cmu', 'bmu', 'R'])
```

	hμ	E	Cμ	bμ	R
Inferred Machine	0.99998	0.00000	0.00000	0.00000	0

Now you're going to do the same, but four the logistic map at band merging (r = 3.6785735104283219).

First, generate 100,000 symbols:

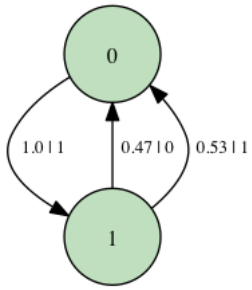
```
symbols = get_symbols(logistic(3.6785735104283219), length=10000)
```

Now infer the machine. Once again, a morph length of 3 and tree depth of 7 will work:

```
logistic_bm = infer_machine(symbols, morph_length=3, tree_depth=7)
```

Now draw the inferred machine:

```
logistic_bm.draw()
```



Why do you think the structure is like that?

Now compute the machine properties:

```
process_table([logistic_bm], ['hmu', 'E', 'Cmu', 'bmu', 'R'])
```

	hμ	E	Cμ	bμ	R
Inferred Machine	0.49833	1.00000	1.00000	0.00000	∞

Next go through the same steps, but for r = 3.8318740552833156.

Generate 100,000 symbols:

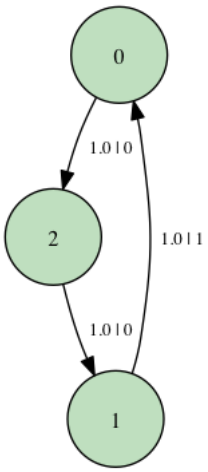
```
symbols = get_symbols(logistic(3.8318740552833156), length=10000)
```

Now infer the machine with morph_length=3 and tree_depth=7:

```
logistic_p3 = infer_machine(symbols, morph_length=3, tree_depth=7)
```

Let's draw it:

```
logistic_p3.draw()
```



Are you surprised? Do you remember any "windows" in the logistic map's bifurcation diagram?

Let's look at the machine properties now:

```
process_table([logistic_p3], ['hmu', 'E', 'Cmu', 'bmu', 'R'])
```

```
+------------------+----------+----------+----------+----------+---+
|                  |    hμ    |    E     |    Cμ    |    bμ    | R |
+------------------+----------+----------+----------+----------+---+
| Inferred Machine |  0.00000 |  1.58496 |  1.58496 |  0.00000 | 2 |
+------------------+----------+----------+----------+----------+---+
```

Finally, let's look at the logistic map at the onset of chaos, r = 3.5699456718695445.

Once again, generate 100,000 symbols:

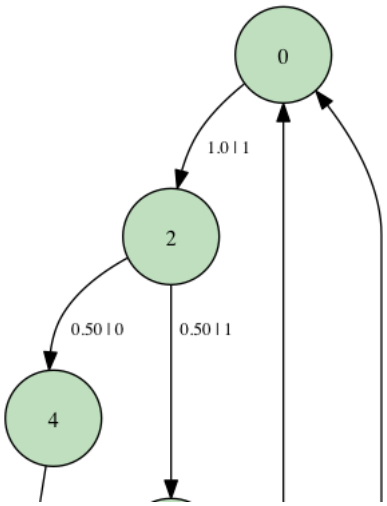
```
symbols = get_symbols(logistic(3.5699456718695445), length=100000)
```

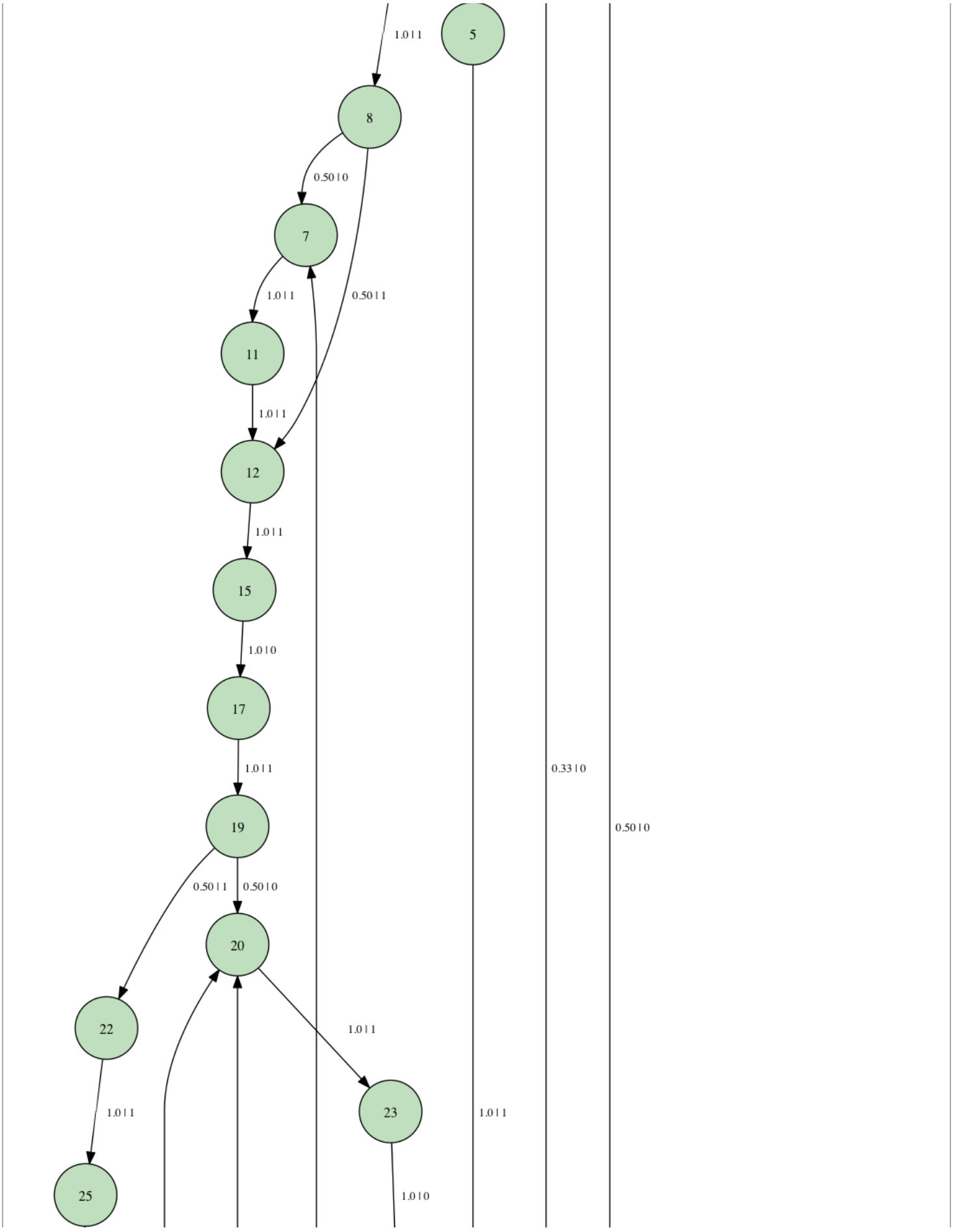
And infer the machine with morph_length=16 and tree_depth=32:

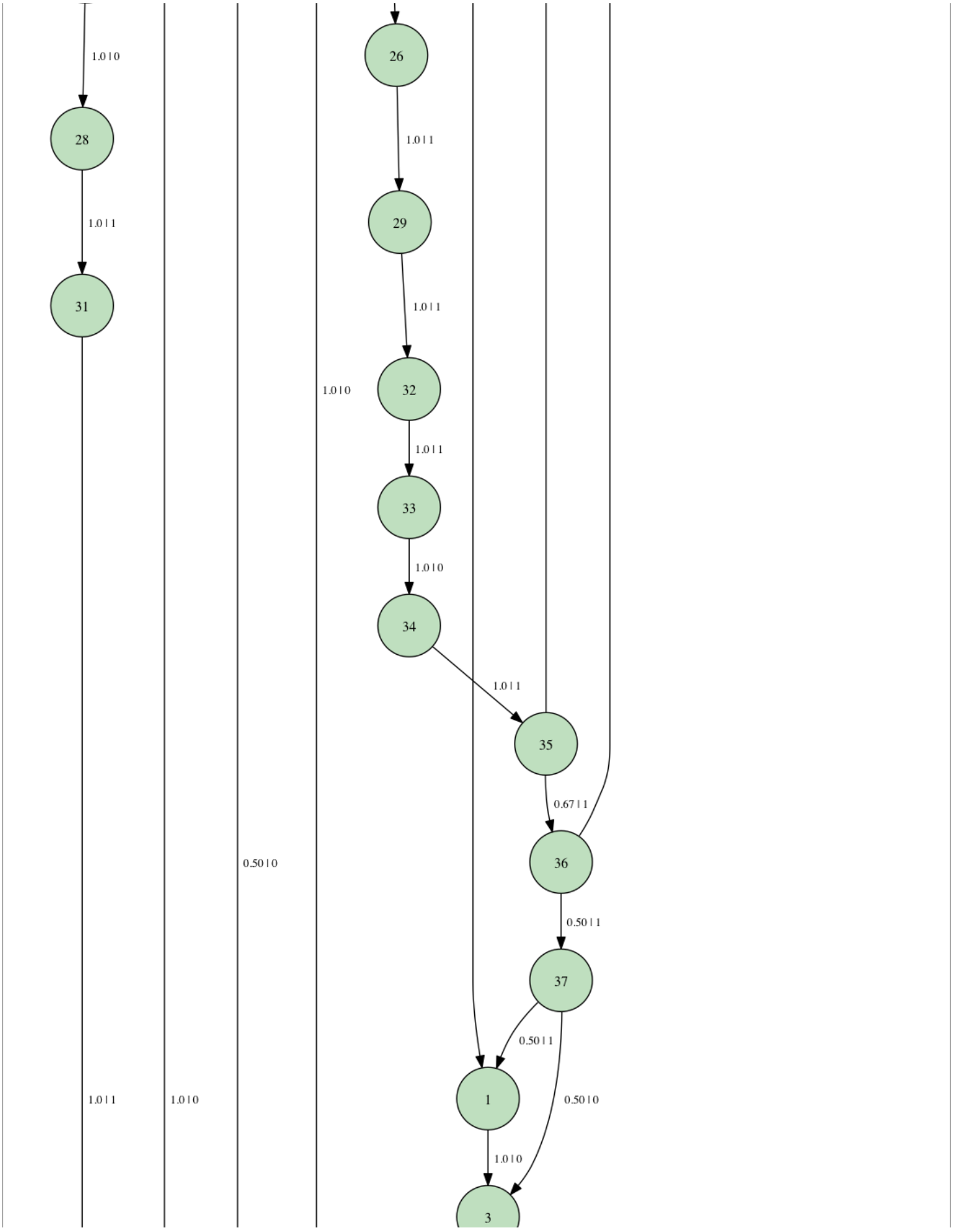
```
logistic_onset = infer_machine(symbols, morph_length=16, tree_depth=32)
```

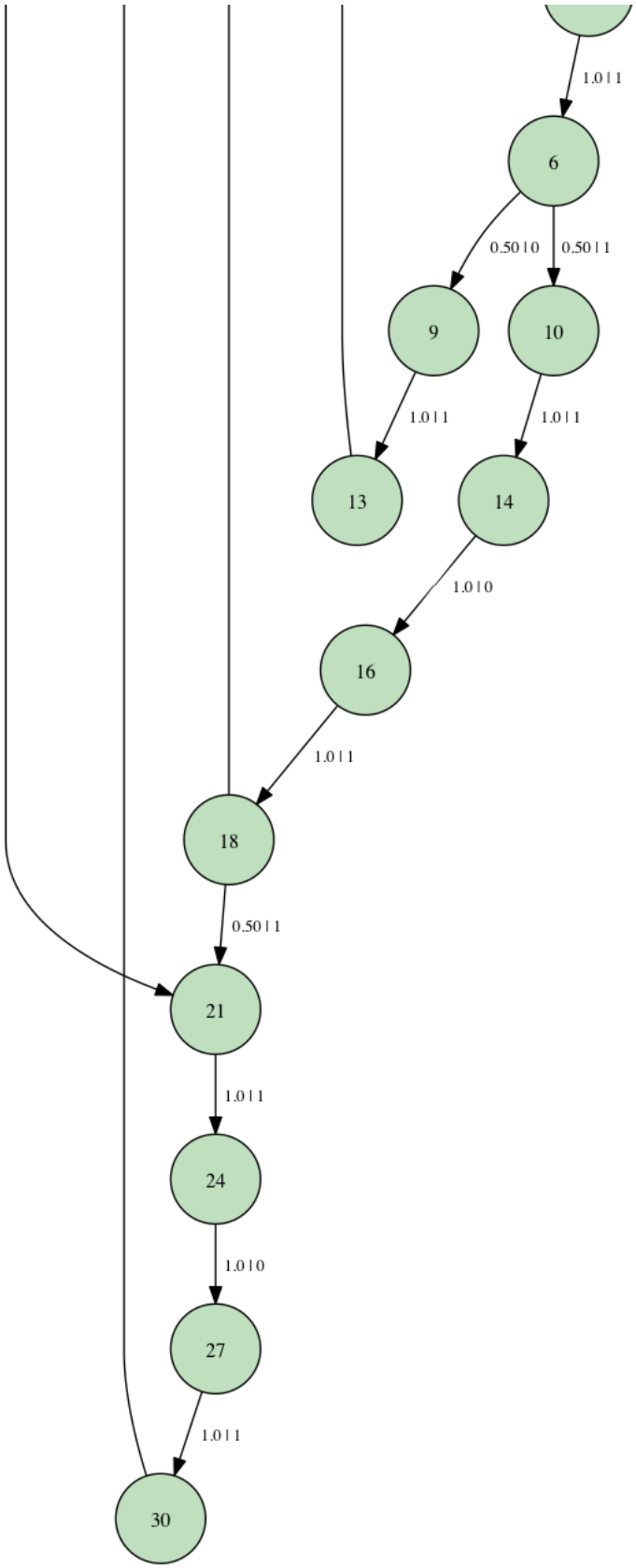
And draw it:

```
logistic_onset.draw()
```




1.0 | 1
5
8
0.50 | 0
7
1.0 | 1
0.50 | 1
11
1.0 | 1
12
1.0 | 1
15
1.0 | 0
17
1.0 | 1
0.33 | 0
19
0.50 | 0
0.50 | 1
0.50 | 0
20
22
1.0 | 1
1.0 | 1
23
1.0 | 1
25
1.0 | 0


1.0 | 0
28
1.0 | 1
31
1.0 | 1
1.0 | 0
0.50 | 0
1.0 | 0
26
1.0 | 1
29
1.0 | 1
32
1.0 | 1
33
1.0 | 0
34
1.0 | 1
35
0.67 | 1
36
0.50 | 1
37
0.50 | 1
0.50 | 0
1
1.0 | 0
3



Were you expecting that?

And lastly, the machine properties:

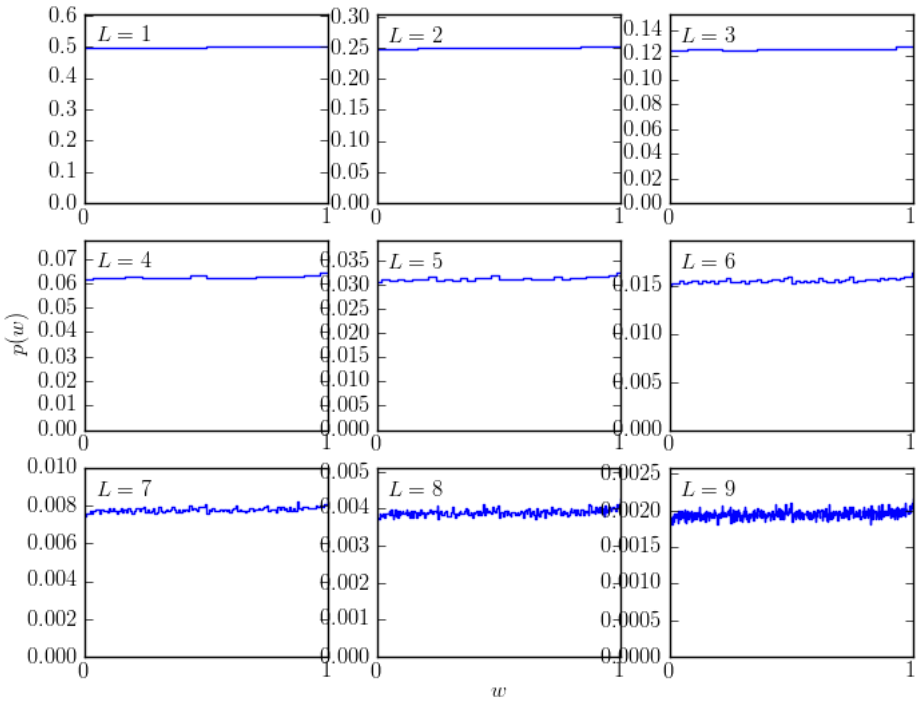
```
print "Number of states:", len(logistic_onset)
```

```
Number of states: 38
```

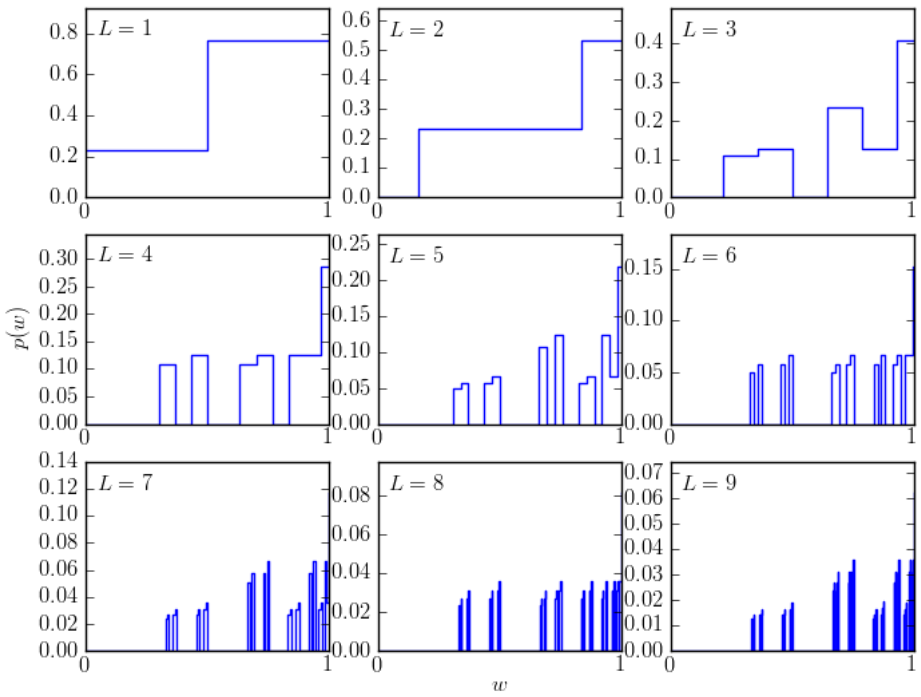
```
process_table([logistic_onset], ['hmu', 'E', 'Cmu', 'bmu', 'R'])
```

```
+------------------+----------+----------+----------+----+---+
|                  |    hμ    |    E     |    Cμ    | bμ | R |
+------------------+----------+----------+----------+----+---+
| Inferred Machine |  0.20685 |  3.43922 |  5.12020 | -- | ∞ |
+------------------+----------+----------+----------+----+---+
```

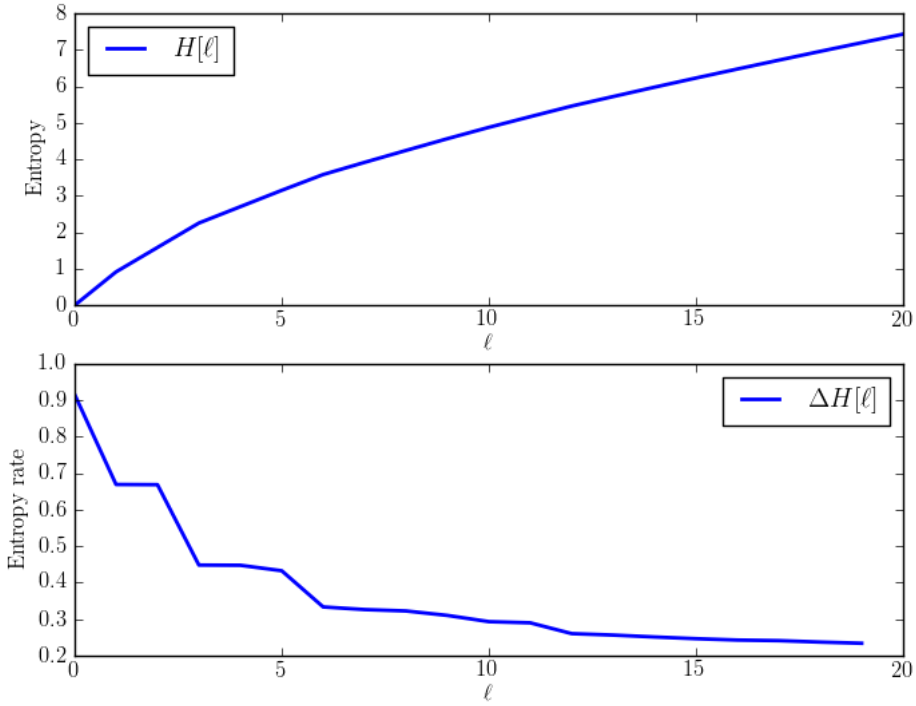
```
# We can also look word plots for these machines:
word_distribution_grid(logistic_4)
```




```
word_distribution_grid(logistic_bm)
```




```
# and we can look at the block entropy plot for the logistic map at the onset of chaos:
block_entropy(logistic_onset, length=20)
```



evaluate