Segmenting an Unknown Language (3rd part)

Understanding bigramSourceModel

Import bengali.py and create a very simple segmentations array.

In [9]:
```
from bengali import *

segs = ['ace+ed']
(fsa,vocab,lm) = bigramSourceModel2(segs)
```

vocab

vocab is a dict used as a set to keep track of all the characters seen in segs.

In [10]:
```
vocab
```

Out[10]:
```
{'+': 1, 'a': 1, 'c': 1, 'd': 1, 'e': 1, 'end': 1}
```

lm

lm holds the normalized character bigram model probabilities

$$\forall c \in lm[h] \quad lm[h][c] = P(c|h)$$

For example, *a* is only followed by *c* in *ac+ed* and *e* is followed by both + and *d*. The reason why the other letters have non-zero values is smoothing.

In [11]:
```
print lm['a']
print lm['e']
```

```
{'a': 0.125, 'c': 0.375, 'end': 0.125, 'd': 0.125, '+': 0.125, 'e': 0.125}
{'a': 0.1, 'c': 0.1, 'end': 0.1, 'd': 0.3, '+': 0.3, 'e': 0.1}
```

a follows *start* and *end* follows *d*, but because of smoothing, *end* also follows *start*.

In [12]:
```
print lm['start']
print lm['d']
```

```
{'a': 0.375, 'c': 0.125, 'end': 0.125, 'd': 0.125, '+': 0.125, 'e': 0.125}
{'a': 0.125, 'c': 0.125, 'e': 0.125, 'd': 0.125, '+': 0.125, 'end': 0.375}
```

fsm

Nodes have integer ids.

In [13]:
```
print "%d nodes"%fsm.N
print fsa.nodes
```

```
7 nodes
{'a': 3, 'c': 4, 'end': 2, 'd': 5, '+': 6, 'start': 1, 'e': 7}
```

Edges. Here are all the edges starting from *start* and *a*. (Not sure why each edge dict is held in an array. Could there be more than one?)

In [14]:
```
print "%d edges"%fsa.M
for fromNode in ['start','a']:
    for toNode in fsa.edges[fromNode]:
        (consume,emit,probability) = fsa.edges[fromNode][toNode][0]
        print "%s ----%s:%s---P=%f----> %s" % (fromNode,consume,emit,probab
```

```
36 edges
start ----a:None---P=0.375000----> a
start ----c:None---P=0.125000----> c
start ----e:None---P=0.125000----> e
start ----d:None---P=0.125000----> d
start ----+:None---P=0.125000----> +
start ----None:None---P=0.125000----> end
a ----a:None---P=0.125000----> a
a ----c:None---P=0.375000----> c
a ----e:None---P=0.125000----> e
a ----d:None---P=0.125000----> d
a ----+:None---P=0.125000----> +
a ----None:None---P=0.125000----> end
```

Test with stipidChannelModel

Make a corresponding words array, and test with *stupidChannelModel*. Not sure if this is working, and if not then why.

In [21]:
```
words = ['aced']
fst = stupidChannelModel(words,segs)

preTrainOutput = FSM.runFST([fsa,fst],words,quiet=True)
for i in range(len(preTrainOutput)):
    if len(preTrainOutput[i]) == 0: preTrainOutput[i] = words[i]
    else:                           preTrainOutput[i] = preTrainOutput[i][0
preTrainEval    = evaluate(segs, preTrainOutput)
print 'before training, P/R/F = ', str(preTrainEval)
```

```
executing:  /Users/alex/opt/carmel_graehl/carmel/bin/macosx/carmel -rIQEb
-k 1 .tmp.fst.0 .tmp.fst.1 .tmp.fst.strings > .tmp.output
before training, P/R/F =  (1.0, 0.0, 0.0)
```

In [ ]: