Assignment 2
CSCI562
Prof. Kevin Knight
***** due at the beginning of class Thursday, September 21, 2006 *****

PART ONE (35pts)

In this part, you will build a language model that assigns a probability to any input string. The goal will not be to model all of English, but rather a subset of English that consists just of phrases describing time periods and points. Sample phrases are:

```
last year
october 18 , 1974
a century ago
```

You will find 10,000 sample training phrases in

```
/auto/home-scf-22/csci562/asst2/time-train
```

You will also find 100 sample "development test" phrases in

```
/auto/home-scf-22/csci562/asst2/time-devtest
```

Build a weighted finite-state acceptor (WFSA) that assigns probabilities to such phrases. Call this WFSA ***model***. You may assume that test data will never introduce any new vocabulary items beyond those found in the training data. However, test data may include phrases never seen before in training.

Your WFSA's transitions should be labeled with whole words, not letters.

You may use any method you like (n-grams, grammar, smoothing, linguistic constraints …). The best WFSA will concentrate its probability on well-formed, common phrases, and not waste much probability on ill-formed, infrequent phrases. A good WFSA will be useful for choosing among possible transcriptions (from speech), possible spelling corrections, possible translations from a foreign language, etc.

The "development test" set will be useful as it may contain unanticipated phrases. You can use this set when you test and re-test your WFSA. The final test, however, will be on a set of 100 phrases that you have not seen before – *please send your WFSA by email to* **jonmay@isi.edu**.

The –b option ("batch") in Carmel will send the whole time-devtest file through your WFSA:

```
% cat time-devtest | carmel -bsriOQk 5 model | more
```

This displays at most 5 paths through your WFSA for each phrase in time-devtest, and each path is shown with its probability. To get the probability for the whole time-devtest corpus, probabilities of paths within a phrase should be summed, and probabilities across phrases should be multiplied. You may use this script or write your own:

```
% cross-entropy model time-devtest
```

This script uses "`carmel –S`" which can sum path probabilities for a supplied set of phrases. Please ignore any messages from Carmel about "Corpus Probability" or "Model perplexity".

The command

```
% carmel -g 20 model
```

will stochastically generate phrases -- it will give you a good idea of what your WFSA considers to be a likely phrase, i.e., where your WFSA is concentrating its probability.

In addition to e-mail submission, please turn into a drawn sketch of your WFSA on paper. The sketch should show your basic strategy, but of course it should not include every state and transition. (Do not turn in computer plots spanning many pages or using unreadable font sizes). Also, please turn in a text description of your strategy.

Full points for assigning low cross-entropy to the test corpus

---

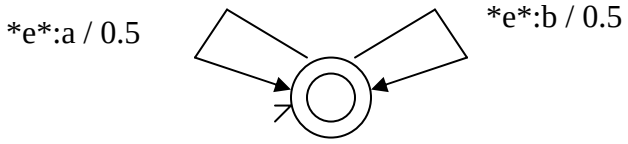
Each transition should include *e* as the input symbol and some word as the output symbol. Transitions should have the form: (start-state (destination-state *e* "word" probability))

Your WFSA should not assign a probability mass greater than one to the set of all strings. You can use the -n switch in Carmel to help ensure this, e.g.:

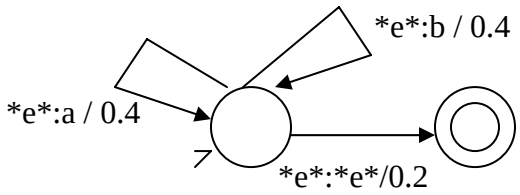
```
% carmel -n wfsa.bad >wfsa.good
```

This switch makes sure that all transitions leaving a state with the same input symbol (in this case *e*) will have probabilities that sum to one. The –n option only works if all transitions have *e* input.

Important: Your final state should have no exiting transitions. For example, this WFSA:



will assign P(a) = 0.5, P(b) = 0.5, P(ab) = 0.25, etc., which adds up to more than one. Correct is:



You will find a sample WFSA in:

```
/auto/home-scf-22/csci562/asst2/model
```

This unigram WFSA does not assign very good probabilities, but it demonstrates how training data can be turned into probabilities, and how the resulting WFSA can be applied to test data. It has a cross-entropy of 7.38 on the `time-devtest` data, as shown here (please ignore the line, from newer versions of Carmel, that says "Corpus probability…model perplexity…"):

```
% cross-entropy model time-devtest
Corpus probability=e^-1054.62637218982 ; Per-example model
perplexity(N=100)=38035.062927695
7.385943
```

This model's cross-entropy on `time-train` is 7.75:

```
% ./cross-entropy model time-train
Corpus probability=e^-103246.868043347 ; Per-example model
perplexity(N=10000)=30475.7573127106
7.753969
```

PART TWO (10pts)

Create an FST that transforms any string of words from PART ONE as follows: for each input word X, the FST should non-deterministically output either (1) the same word X or (2) the corrupted word "???". Call this FST ***corrupt***. Turn in this FST by email to **jonmay@isi.edu**, and draw a sketch on paper. For example, for the input `a year ago` , possible outputs include:

```
a ??? ago
??? year ago
a year ago
??? year ???
```

Combine your WFSA ***model*** with your FST ***corrupt*** in order to compute most likely English reconstructions for the following strings, and report your answers on paper:

```
(1) ??? hours
(2) a ??? ago
(3) ??? last ???
```

First you will need to create a version of ***model*** (call it ***model-no-e***) that does not have *e* on input transitions. Then a command like the following will be useful:

```
% echo ' "???" "hours" ' | carmel -sriIEQk 5 model-no-e corrupt
```

You should get reasonable guesses/reconstructions out. Your ***model*** + ***corrupt*** combination will also be tested on how well they reconstruct a collection of held-out (corrupted) phrases.

PART THREE (5pts)

Suppose you are given strings like these, which are phonetically garbled dates, as from a speech recognizer:

```
/auto/home-scf-22/csci562/asst2/test.phon
"L" "OW" "S" "T" "M" "AH" "N" "Z"
"AE" "N" "D" "R"
```

```
"JH" "IY" "B" "B" "UW" "EH" "R" "IY"
"N" "B" "N" "TH" "S"
```

Use your language model to correct these. We have provided these two files to help:

`sound-out.fst` -- turns word sequences into (correct) phoneme sequences
`phon-corrupt.fst` -- probabilistically corrupts phoneme sequences

Try these automata out in isolation, to see how they work, then put everything together to solve the problem. Try something like this:

```
carmel -brIEQk 5 model.noe sound-out.fst phon-corrupt.fst test.phon
```

Is the language model important? Show all your results and briefly discuss them. Your will be tested on how well your model helps un-garble new strings.

SUMMARY

TO TURN IN ON PAPER:

- A WFSA sketch of ***model***, and a text description of your strategy for PART ONE.
- An FST sketch of ***corrupt***.
- Your answers to problems in PART TWO and PART THREE.

TO TURN IN BY EMAIL:

- your WFSA ***model***
- your FST ***corrupt***

FINAL NOTE

The `asst2` directory has an updated Solaris binary for carmel, called `carmel.static`. This version has updated features and bug fixes. It is the same as the one available on the Carmel web site.