Homework #1 CS662
Fall 2016
Author : Karan Singla

1. Spanish FSA : finite-state acceptor (FSA) that accepts letter strings consisting of words from spanish-vocab.txt, and rejects all others

Data Provided : A sorted dictionary of spanish words
Method : There are various methods for creating an FSA for a collection of strings as compared by [1]. I used directed acyclic word graph (**DAWG**) for constructing my FSA. People also tend to call it “Minimal Acyclic Finite State Automaton”. I used python library for constructing and storing a DAWG. The algorithm is described in detail by [2]. It checks for duplicates before inserting each new word, so that the graph never grows too big. It has primarily two steps

1. We ensured that the data has is sorted we begin constructing the graph. When we insert a word, we will then know for sure whether the previous word ended a branch or not. For example, "A B A L" followed by "A B A L A N Z A R O N " does not result in the end of a branch, because the “A N Z O” just added to the end to make “A B A L A N Z O”. But when you follow it with "cats" you know that the "nip" part of the previous word needs checking.
2. Each time you complete a branch in the trie, check it for duplicate nodes. When a duplicate is found, redirect all incoming edges to the existing one and eliminate the duplicate.

I used a python-library called pyDAWG to construct this graph.

Number of states in result: 156713
Number of arcs in result: 219006

Another Method with better results :

1. Construct a normal FSA from dictionary if each new transition going to a new node and ending with same state.
   Example: For representing A, AARON, AB, ABA
   0 1 A 1
   12345678
   0 2 A 1
   2 3 A 1
   3 4 R 1
   4 5 O 1
   5 12345678 N 1
   12345678
   0 7 A 1
   7 12345678 B 1
   12345678
   0 9 A 1
   9 10 B 1
   10 12345678 A 1
   12345678

2. Remove all epsilon and then determinize it. (it is important to remove epsilon for determinizing it)
3. Minimize the FSA and then mark all the multiple ending states to a new final state with epsilon

NOTE : I did it using OpenFST toolkit

Some Stats of compiled FSA (using carmel -c) :

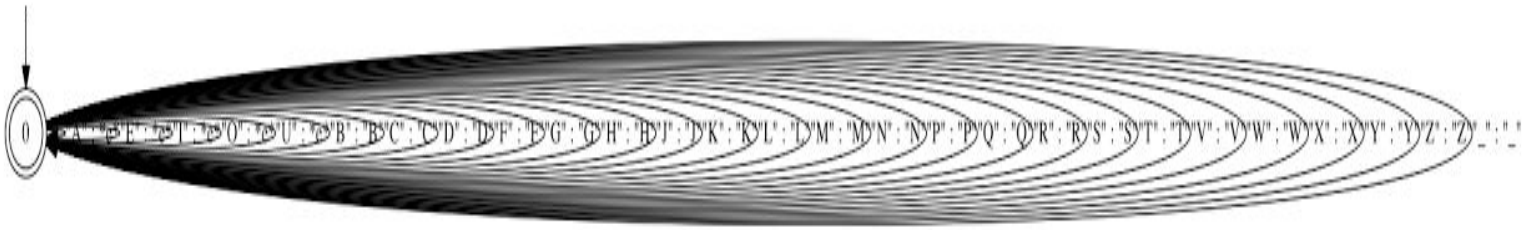
Number of states in result: 39178
Number of arcs in result: 101492

Usage : echo ' "L" "O" "S" "_" "G" "A" "T" "O" "S" ' | carmel -sli spanish.fsa

2. Vowel-deleter.fst : FST which removes all vowels from a string

- it returns *e*/eps for every vowel character otherwise returns the character itself
- The backward FST works poorly as now it generates word-sequences which have vowels at arbitrary places instead of the following the grammatical rules.

Figure :



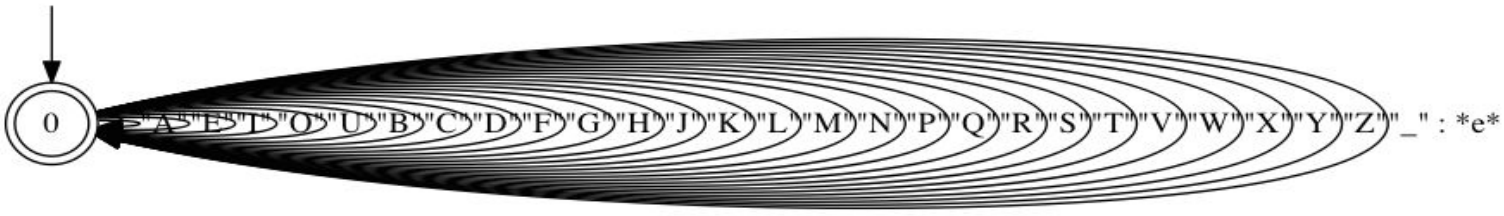
Forward
echo ' "G" "A" "T" "O" "_" "A" "Q" "U" "I" ' | carmel -sliOEWk 10 vowel-deleter.fst
Output : "G" "T" "_" "Q"
Backward
echo ' "P" "R" "R" "_" "Q" ' | carmel -sriIEWk 10 vowel-deleter.fst
Output : Vowels inserted at infinitely many places.

3. FST to delete spaces : space-deleter.fst

- It just returns an epsilon (*e*) if string has a "_" (space) otherwise it just returns the character.
- The backward FST works poorly as now it generates word-sequences which have spaces at arbitrary places instead of the following the grammatical rules.

The backward FST will put "_" (space) at random places.

Figure :



Forward

echo ' "G" "_" "T" "O" "_" "A" "Q" "_" "I" ' | carmel -sliOEWk 10 space-deleter.fst

Output : "G" "T" "O" "A" "Q" "I"

Backward

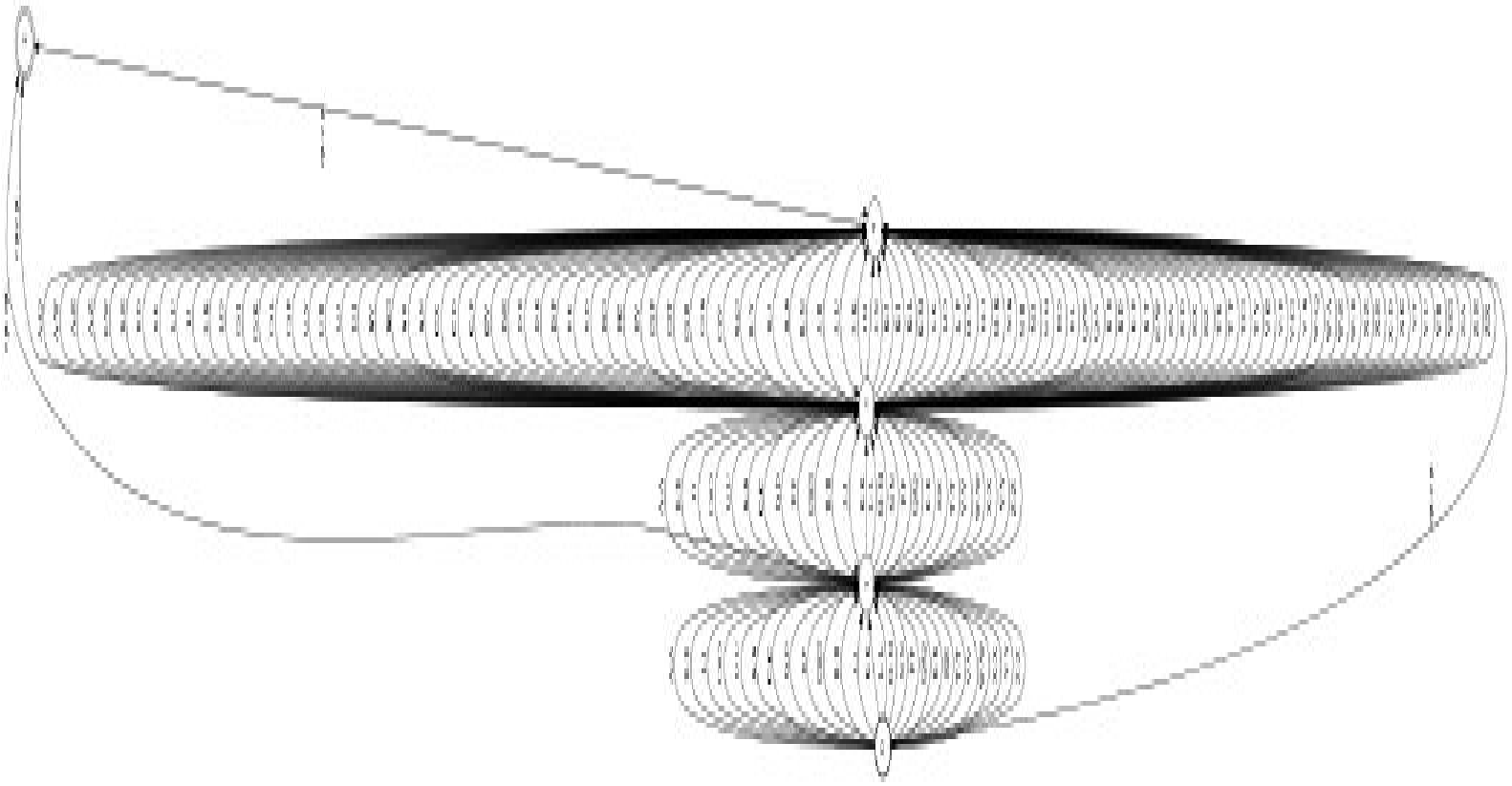
echo ' "P" "R" "R" "Q" ' | carmel -sriIEWk 10 space-deleter.fst

Output : space (“_”) inserted at infinitely many places.

4. <u>FST to create typographical errors : Typo.fst</u>

- State 0 (start state) takes input any character and returns it and jumps to state 1. It is also connected to 4 (final state) if there is just epsilon after it.
- State 1 is then mapped to state 2 for the symbols marking it has read n%3=2 characters. It is also connected to 4, if there is just epsilon marked after that.
- State 2 is then mapped to state 3 for the symbols marking it has read n%3=0 characters. It is also connected to 4, by an epsilon if the string terminates. Otherwise it is connected to state 0 to start the cycle again. Mapping from state 2 to state 3 generates the typographical errors.

The backward FST will also create problems as it will have many possible outcomes that are not grammatical according to the desired output



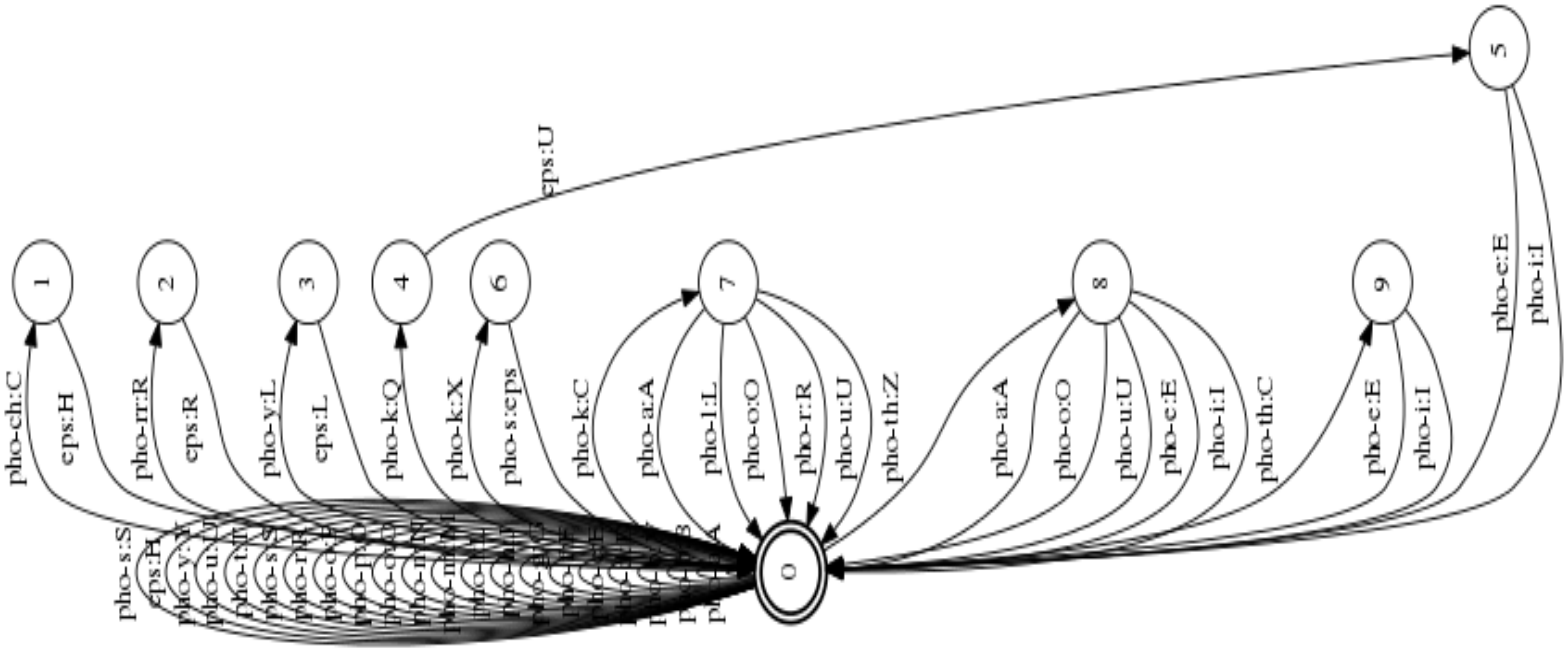
Forward :

echo ' "G" "A" "T" "_" "A" ' | carmel -sliOEWk 10 typo.fst
Output : "G" "A" "R" "_" "A", "G" "A" "F" "_" "A", "G" "A" "G" "_" "A", "G" "A" "Y" "_" "A"

Backward
echo ' "G" "A" "R" "_" "A" ' | carmel -sriIEWk 10 typo.fst
Output :"G" "A" "E" "_" "A", "G" "A" "D" "_" "A", "G" "A" "F" "_" "A", "G" "A" "T" "_" "A"

5. **FST converts a Spanish phoneme (sound) sequence into a Spanish letter sequence**

- All spanish phoneme sequence which returns a single character and have no pre and postconditions are just marked from state 0 to state 0
- All spanish phoneme sequence which returns a single character and have postconditions (are followed by) will first go to a new state giving desired output and then output based on the input being followed. Example : state 6,7,8 and 9.
- All spanish phoneme sequence which returns multiple character and have no pre and postconditions are just marked from state 0 to state X and then state X to state 0. Example : 0 1 pho-ch : C, followed by 1 0 eps : H
- All spanish phoneme sequence which returns multiple character and have postconditions will have one more arbitrary state in between to return the extra symbol. Example : state 5



Forward
echo ' "pho-k" "pho-a" "pho-y" "pho-e" ' | ../nlp/carmel/graehl/carmel/bin/macosx/carmel -OQWEslik 5 spell.fst
Output : C A Y E, C A Y E H, C A Y H E, C A Y E H H, C A Y H E H
Backward
echo ' "C" "A" "L" "L" "E" ' | ../nlp/carmel/graehl/carmel/bin/macosx/carmel -IQWEsrik 5 spell.fst
Output : pho-k pho-a pho-y pho-e, pho-k pho-a pho-l pho-l pho-e

6. Improving reverse applications of FST's 2, 3, 4 & 5 using spanish.fsa

Naive Solution : Insertions filtered based upon Dictionary

reverse applications of 2(vowel deleter):vowel restoration, 3(space-deleter):space insertion and 4(typo.fst):typo_corrector is done by sticking spanish.fsa to these FST's. And then using that in reverse. It can be down by using following two commands.

a) carmel spanish.fsa <2,3,4> > combined.fst
b) echo ' "G" "A""F" "O" ' | ../nlp/carmel/graehl/carmel/bin/macosx/carmel -sriIEWk 20 combined.fst

Sticking spanish.fsa makes sure that it only returns the output that is confirmed by it. It can also be seen that in the backward direction spanish.fsa can be seen as a language model in the noisy channel model but here it just considers the full strings without any probabilities.

For 5, we need to do the other way, as we need to mind the output symbols not the input one.

a) carmel spell.fst spanish.fsa > combined.fst
b) echo ' "pho-a" "pho-b" "pho-a" ' | ../nlp/carmel/graehl/carmel/bin/macosx/carmel -OQWEslik 10 2.fsa

NOTE : we didn't do carmel spanish.fsa <5> because output input symbol of spell is "pho-a" and output "A...Z" etc so output should match the input set of symbols and act as output filter

Problem :

a) It will just return an output if it fits spanish.fsa otherwise NULL
b) It doesn't give any ranking on the outputs

Better Possible Solution : Instead of NULL it can just acquire probabilities from the data.

We should calculate P(y | x) + P(y), when can then be given weights by interpolation A. P(y | x) + (1-A) P(y). For this we need to keep a development set to evaluate and adjust A (weight).
Here we can't calculate probabilities for spell.fst but we can do it for spanish.fsa

Like making a 3-gram model out of it and calculate probabilities of tri-grams, bi-grams and uni-grams. Also do some form of smoothing so that it can back-off in case of unknow tri/bi/uni grams. In that case the output from P(y|X) or spell.fst will be just ranked according to cumulative transition probabilities based on spanish.fsa.

References

1. Daciuk, Jan. "Comparison of construction algorithms for minimal, acyclic, deterministic, finite-state automata from sets of strings." *International Conference on Implementation and Application of Automata*. Springer Berlin Heidelberg, 2002.
2. Daciuk, Jan, et al. "Incremental construction of minimal acyclic finite-state automata." *Computational linguistics* 26.1 (2000): 3-16.