Report for HW2

CS 519 Natural Language Processing

Prof. Liang Huang

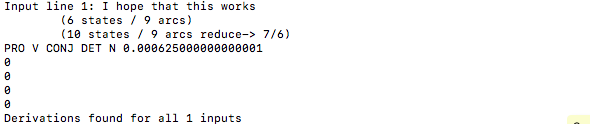
Group member: Gunea.Aditya, Hao.Liu, Ke.Huang

Part 1

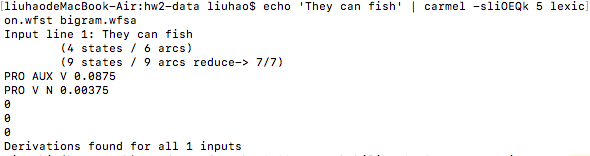
1. bigram.wfsa
2. lexicon.wfst

3.

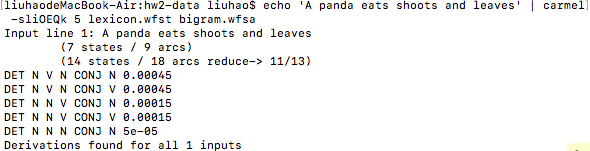
I hope that this works



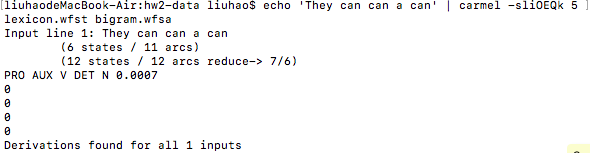
They can fish



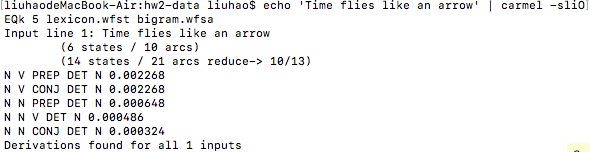
A panda eats shoots and leaves



They can can a can.



Time flies like an arrow



4. The combination of lexicon.wfst and input string will give a probability of

P1 (w…w | t…t)

And the probability of bigram.wfsa is like P2 (t…t)

So we need to composite these two probabilities.

Then the total result is P1 \* P2

Use “They can fish” as an example:

As I defined: P2 (PRO AUX V) = 0.25 \*1.0 = 0.25

P1 (they can fish | PRO AUX V) = 1.0\*0.7\*0.5 = 0.35

So the total probability P1 \* P2 = 0.25\*0.35 = 0.0875

5. We can see for the third and fifth sentences there are lot of means.

For the third one:

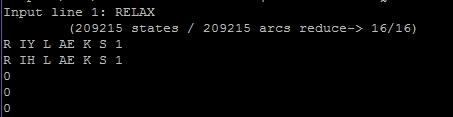
Such as panda eats something and goes away or a type of eat uses gun to shoot and goes away ☹

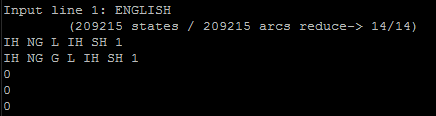
For the fifth one:

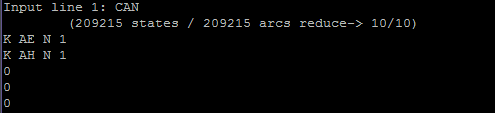
One mean can be time goes fast, or a kind of fly looks like an arrow ☹

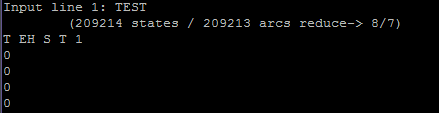
These are all meaningful sentences, if we want the mean become more common, we can use an N-gram to memory more high frequency sentence. In that case we may get a normal structure of each sentence.

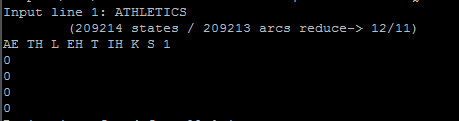
## 2.1: Here are 5 results:



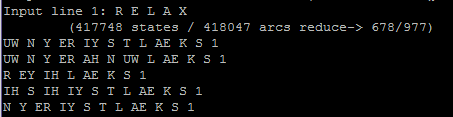


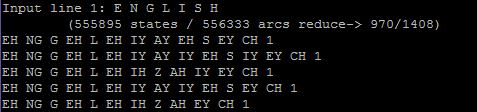


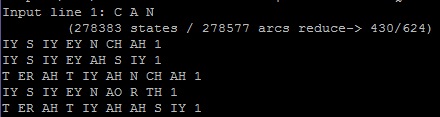


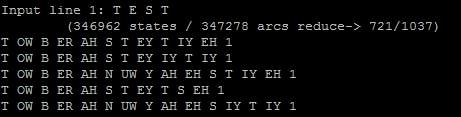


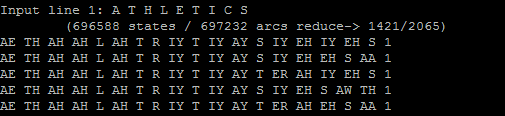
## 2.2: Here are 5 results:





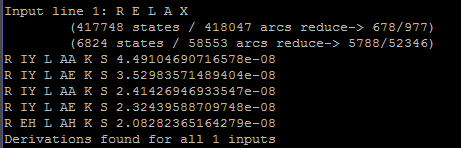


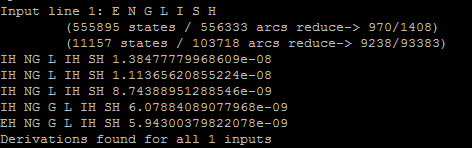


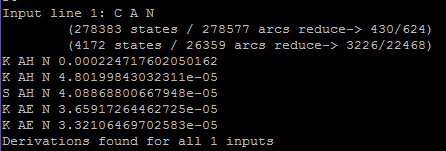


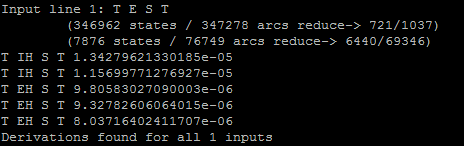
In this question, we get some non-sense output. Because the machine will go back and check all possibility of each letter and combine with them. For example, in word HELLO, “H” has many combination like H=HH-EH, H=EY-CH, H=AW-ER and so on. “HELLO” could also be separated as many combination like “H + ELLO” or “HE + LLO” or “H + ELL + O” and so on. This machine will try all kinds of result but not pronounce the word as English rule.

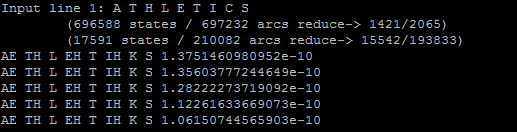
## 2.3: Here are 5 results:



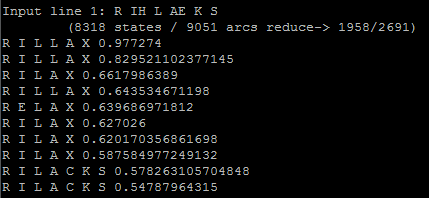


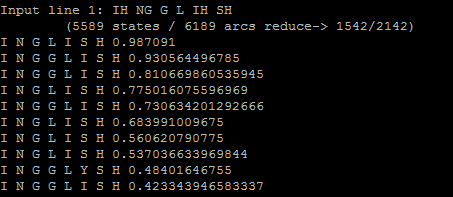


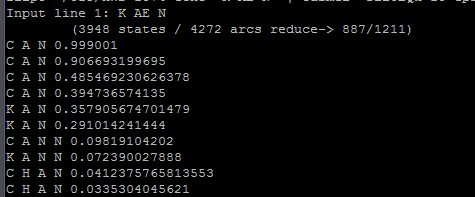


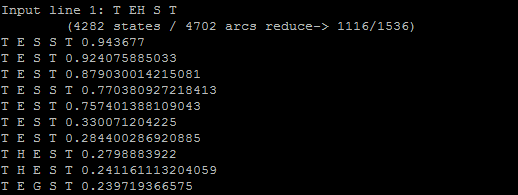


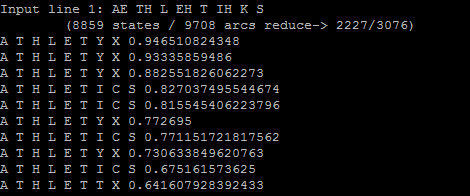
## 2.4: Here are 5 results (cause each result is too long, I only pick up first 10 line for each result)



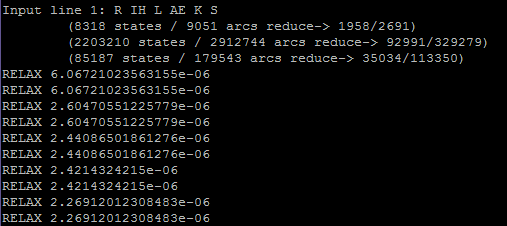


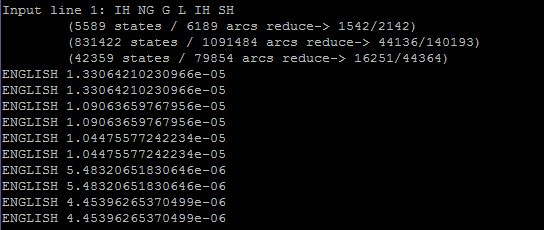


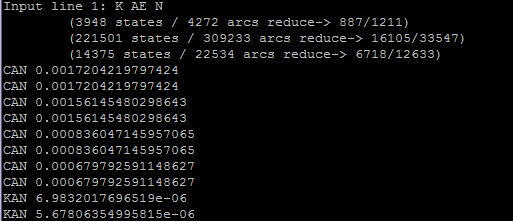


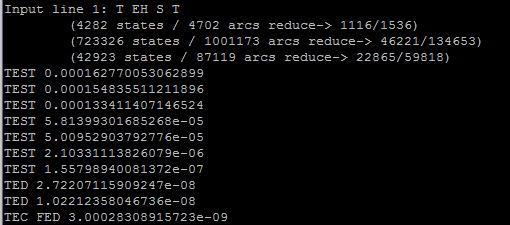


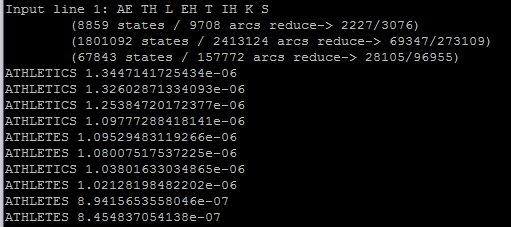
## 2.5 Here are 5 results (cause each result is too long, I only pick up first 10 line for each result)





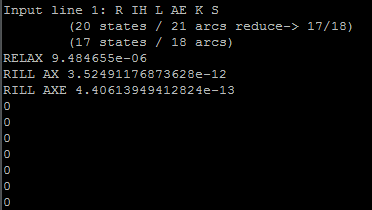


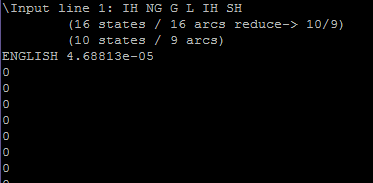


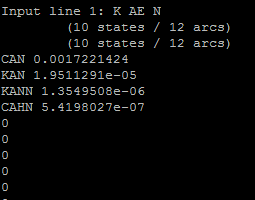


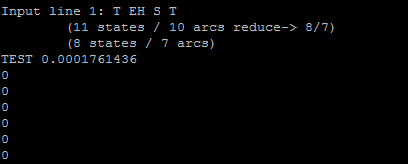
## 2.6: Here are 5 results:

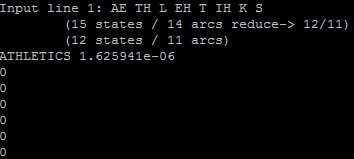
These results are better than question 5. It's much clearer (most of results only have 1-4-line output). It is because epron-eword.wfst is built from eword-epron.wfst which is built from database. That means all output from this wfst must be include in database (even it is a combination like RILL + AX). It makes the output less and specific.









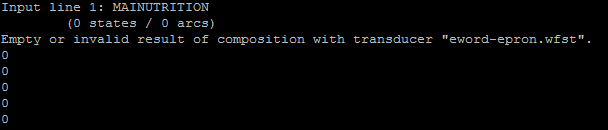


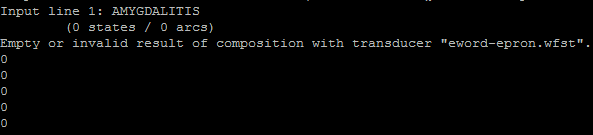
## 2.7 Here are some Results: The words I used here are:

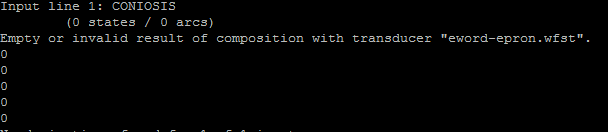
## mainutrition (M AE L N UW T R IH SH N),

## amygdalitis (AH M IH G D AE L AY T IH Z),

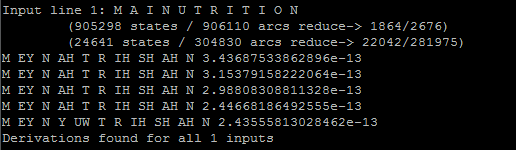
## coniosis (K OW N IH OW S IH S)

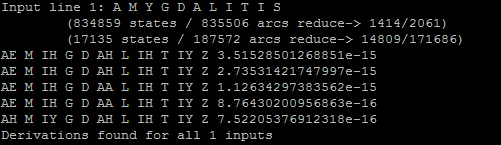


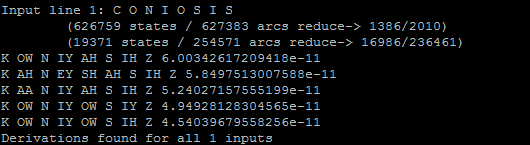




## 2.8 Here are the results



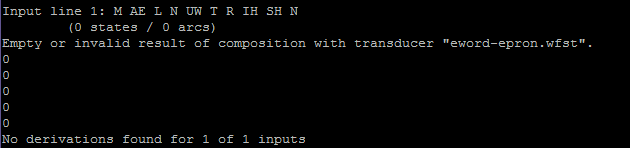


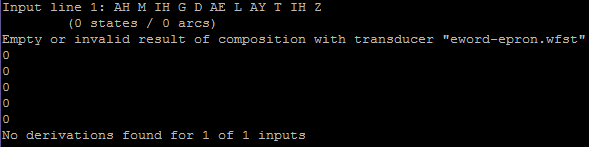


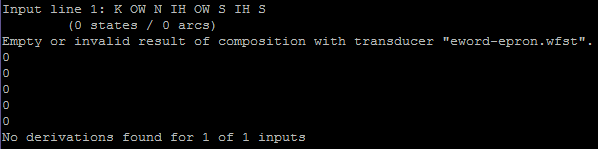
## 2.9 Here are 5 results

Problem 9 is very close to problem 7&8. If we use wfst which forming the database, because the word is not exist in the data, so this wfst can't find its path and pronunciation so the output is empty. But if we use epron-espell.wfst, it will work because it restore pronunciation with rule of English letter. Although most of the results are wrong because this wfst doesn’t consider the context of each phoneme in sequences.

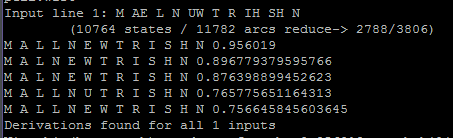
**1. For using eword-epron.wfst**

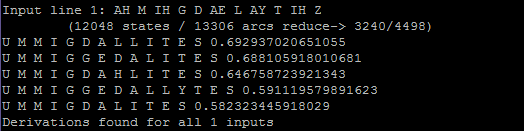


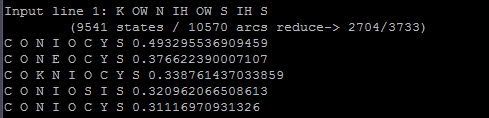




**For using epron-spell.wfst**







## 2.10 Here the result

## 

## 2.11 Analysis of these automata:

In problem 1, cause the eword-epron.wfst is build from data, the commands' function is searching word's pronunciation in the database and export it. Some words have single pronunciation, and some words have two kinds of pronunciation.

In problem 2, epron-espell.wfst is a wfst from phoneme sequence to letter sequences. If we send letter sequences backwards through this wfst, because each letter could have many kinds of pronunciation in different words/position. The results of this command will include a lot of nonsense. It is also why some conventional pronunciation will have some similar results.

In problem 3, after we add the epron.wfsa in our machine, in each letter -> phoneme step, this command will check the first two phoneme and find the most likely pronunciation. This way greatly enhances our results' accuracy.

Problem 4 is similar as problem 2. For some words with regular pronunciation, this wfst will could get the correct result. But for most of words' phoneme sequence input, this wfst will only output some common combination of these pronunciations.

In problem 5, because eword.wfsa could only accept words but not letter sequences, so we use python to build a espell-eword.wfst and delete all spacing. Like change “C A N” to “CAN”. Then we could use the wfsa as a filter to find the most reasonable letter arrangement (base on the letter in front of it).

Problem 6's result is better than 5. It's very clear (most of results only have 1-4-line output). That's because epron-eword.wfst is built from eword-epron.wfst which is built from database. That means all output from this wfst must be include in database. It makes the output less and clear.

In problem 7, just like I mentioned in problem 6, cause eword-epron.wfst is built from database, if we input some words that are not in the database, the output is empty.

But in problem 8, if we use epron-espell.wfst. Because this wfst is based on the rule of pronunciation for each phoneme, it will only translate letter sequences as some possible phoneme. But just as the same as Problem2, most of these outputs are nonsense. (But at least it still works)

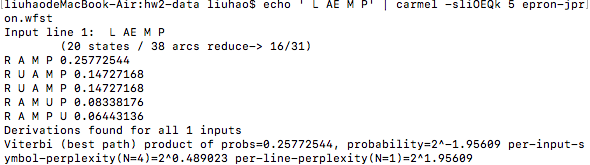
Problem 9 is very close to problem 7&8. If we use wfst which forming the database, because the word is not exist in data, so this wfst can't find its pronunciation and the output is empty. But if we use epron-espell.wfst, it will work(Although most of the results are wrong)

Problem 10 is interesting. This machine first use eword-epron to translate input(word) as its pronunciation (the word should exist in the database). Then it uses the pronunciation to check how many words have the same pronunciation in the database. So, we could see the results: BEAR, BARE, BAER, BEHR, BAHR. All of these words have a same pronunciation.

This series of automata experiments let me understand that: If we only form wfst from word to pronunciation with database, the wfst we get is more like a dictionary but not a translator. If we want a pronunciation translator, we should summarize the pronunciation's rule with double or tribble letter combination in the database (unigram wfsa or trigram wfsa). That's a real translation.

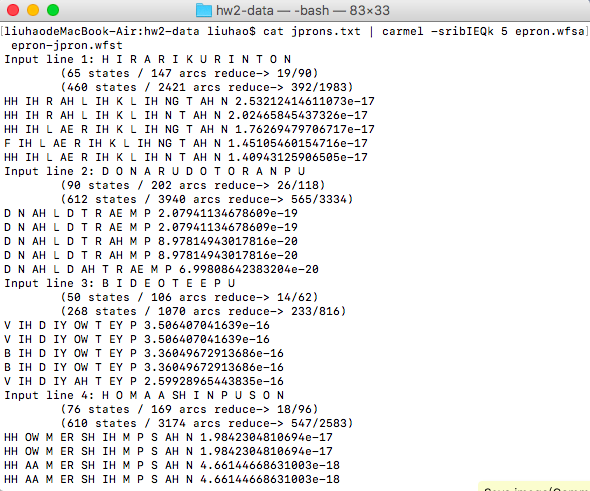
Part 3

1. estimate.py (For some reasons, this .py file only works for python 3. Sorry for any inconvenience) and epron-jpron.probs



1. It makes no sense. Because this is just a unigram for epron-jpron. The wfst will automatically pick the combination with high probability to form the Japanese pronunciation. There is no relationship between two single phonemes. That is we need history to decide what kinds of Japanese phonemes should be given to a certain two or three English phonemes.
2. Use bigram or tri-gram can get better result. (just as explained in question 3)
3. Command of combining epron.wfsa epron-jpron.wfst

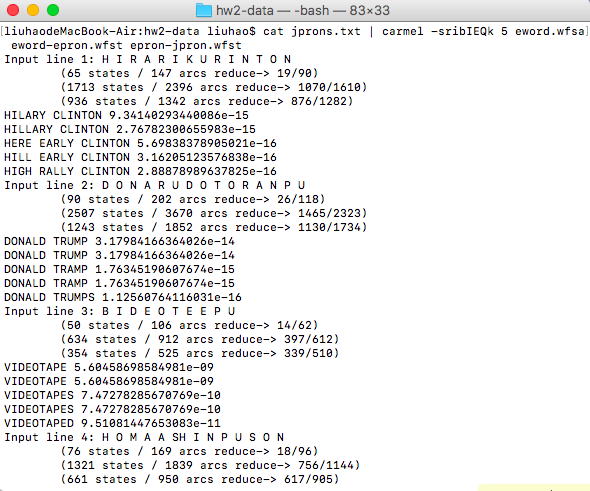
Below shows part of the total result:



It does make so sense if someone is familiar the English phoneme sequence. But these sequences are not real English words. Normal person can’t understand what theses sequences are.

1. Command of combining eword.wfsa eword-epron.wfst epron-jpron.wfst

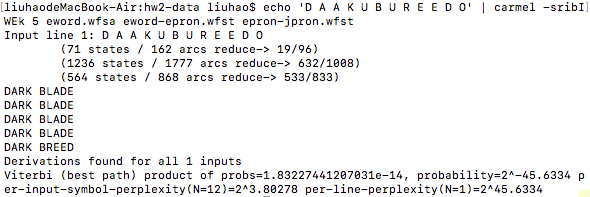
Below shows part of the result:



This time the result is much clearer than the English phonemes. We can recognize the meaning of the word.

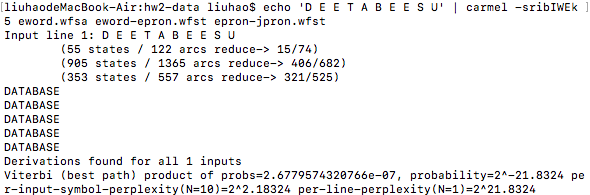
1. Because the machine can’t recognize the relationship between two words. In that case, the language model (.data) determines the result of translation. And also the mapping is based on highest probability coming out first. So the top rank is the word with highest translating probability, rather than the most frequent use. In order to achieve better result, we can provide better language model (model containing lots of common words).

Dark blade



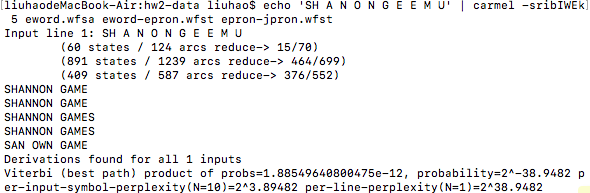
High probability getting correct.

database



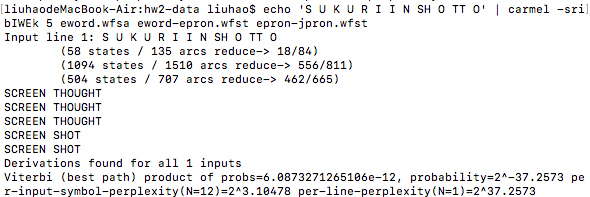
Perfect translate

Shannon game



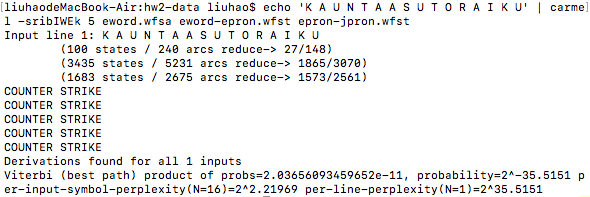
It can also recognize English name, but also has a chance to multiply the noun.

Screenshot



High probability of translate from shots to thoughts

Counter strike



Works well

1. The following sequence is what we have tried in this homework:

epron-jpron.wfst

eword.wfsa

eword-epron.wfst

What we can do:

insert epron.wfsa between eword-epron.wfst and epron-jporn.wfst

epron.wfsa

eword.wfsa

eword-epron.wfst

epron-jpron.wfst

Or just use epron.wfsa without eword.wfsa

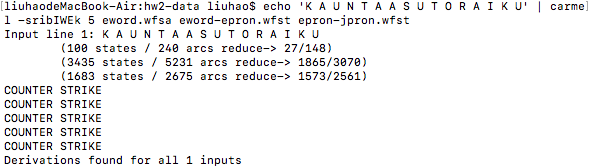
epron-jpron.wfst

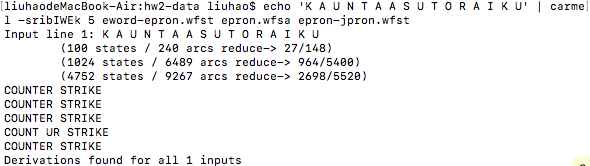
eword-epron.wfst

epron.wfsa

Add another machine means we could achieve high accuracy but low speed theoretically.

Here is the comparison between eword->eword-epron->epron-jpron and eword-epron->epron->epron-jpron





We can see the accuracy of eword-epron->epron->epron-jpron is low than eword->eword-epron->epron-jpron.

1. We can make the machine larger and larger which means the accuracy will definitely improve and high probability of gaining more states and transactions.