

# CS909: Natural Language Processing, Exercise 1

UID: 1690550

March 2, 2017

## 1 Part B

Operation costs:

- insertion: 1
- deletion: 1
- substitution: 2

### 1.1 Minimum edit distance from "refa" to "fear"

	#	f	e	a	r
#	0	1	2	3	4
r	1	2	3	4	3
e	2	3	2	3	4
f	3	2	3	4	5
a	4	3	4	3	4

Table 1: Minimum edit distance of "refa" to "fear" is 4.

There are 2 alignments, discovered by inspection. The alignments are:

*	*	f	e	a	r
r	e	f	*	a	*

*	f	e	*	a	r
r	*	e	f	a	*

### 1.2 Minimum edit distance of "drive" to "brief" and "drive" to "divers"

	#	b	r	i	e	f
#	0	1	2	3	4	5
d	1	2	3	4	5	6
r	2	3	2	3	4	5
i	3	4	3	2	3	4
v	4	5	4	3	4	5
e	5	6	5	4	3	4

Table 2: Minimum edit distance of "drive" to "brief" is 4

	#	d	i	v	e	r	s
#	0	1	2	3	4	5	6
d	1	0	1	2	3	4	5
r	2	1	2	3	4	3	4
i	3	2	1	2	3	4	5
v	4	3	2	1	2	3	4
e	5	4	3	2	1	2	3

Table 3: Minimum edit distance of "drive" to "divers" is 3

## 2 Part C

For the purpose of this problem, we define the following sets:

$$vowel = \{a, e, o, u, i\}$$

$$consonant = \Sigma - vowel$$

The set of consonants will be abbreviated to "cons.".

We are interested in an FST for the consonant doubling rule for verbs. It is important to stress that the FSTs presented below work only for regular verbs.

If the input is given on the lexical level, we can turn it into intermediate with the following FST:

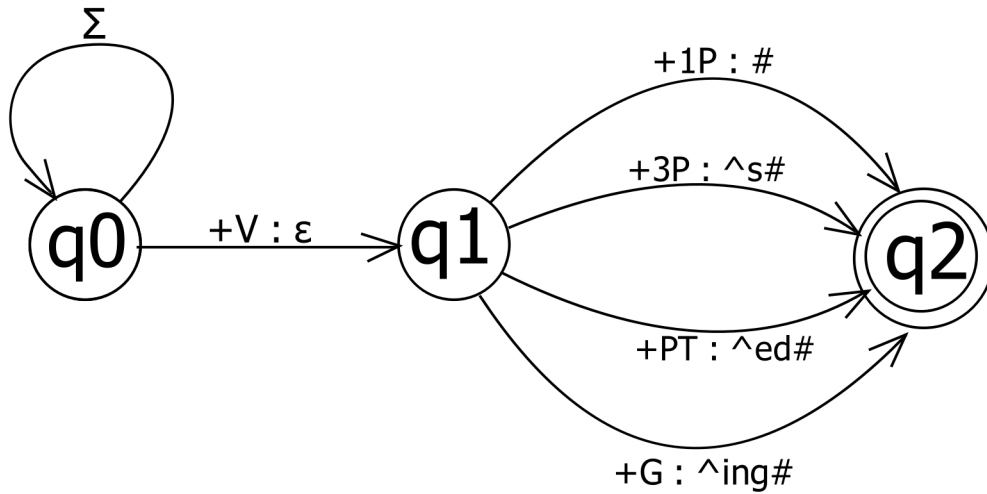


Figure 1: FST between lexical and intermediate level

Where  $\Sigma$  is any letter, '+1P' indicates first person ending (same as stem), '+3P' indicates third person form (i.e. -s suffix), '+PT' stands for past tense (-ed), and '+G' for gerund (-ing).

Figure 2 shows the simplest FST that encodes the consonant doubling rule just that for single syllable verbs, taking intermediate level input.

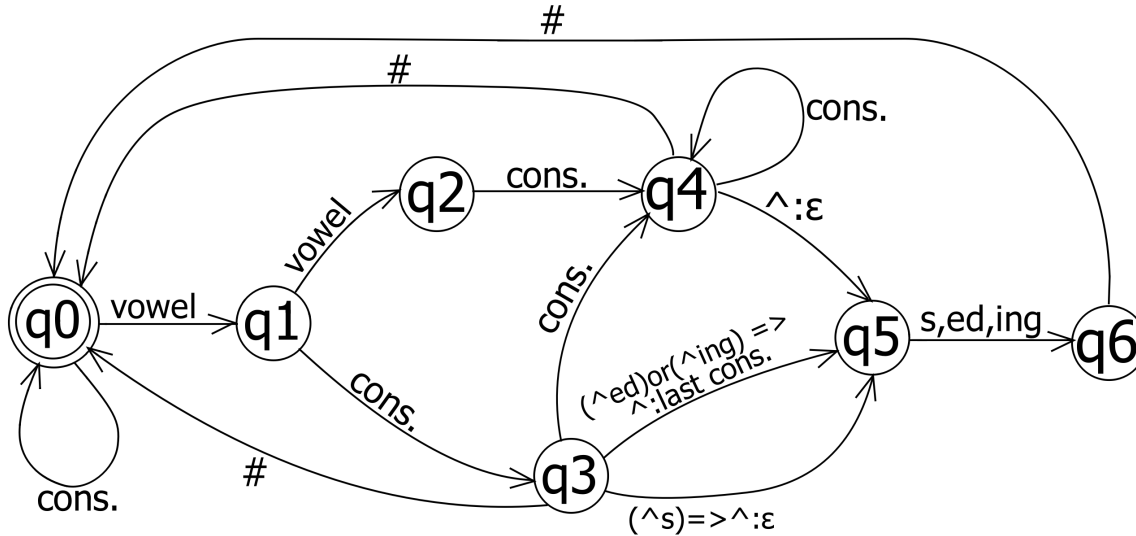


Figure 2: Simple version of the requested FST

The last consonant of the given verb will be doubled only if it is preceded by a single vowel, and if we are aiming for the regular past tense or the gerund of the verb.

Thus for example:

- "aim" will go from state  $q_0$  to  $q_1$ , then as we are seeing a second vowel, we go to  $q_4$ , and then either we see the end of the string symbol and go to the accepting state  $q_0$ , or add a required suffix.
- "bring" will loop over the "cons." edge around  $q_0$  twice, then go to  $q_1$  and on to  $q_3$ . Then, since we're seeing another consonant, we move to  $q_4$  and avoid doubling the last letter.
- "stop" or "fit" will loop around  $q_0$  until they get to  $q_1$  and  $q_3$ . If the next character is the end of string symbol, we move back to the accepting state  $q_0$ . Otherwise, we read ahead to see what ending we need. If it's "s", we replace " $\wedge$ " with empty symbol. Otherwise, we replace it with the last consonant.

However, this FST does not treat "e" at the end of the words correctly. Therefore we present a more robust version, which works for regular verbs, including those with multiple syllables and those ending in e (silent or otherwise).

It is perhaps surprising that only states directly concerned with the consonants doubling are  $q_0$ ,  $q_1$ ,  $q_2$ ,  $q_6$ ,  $q_7$ , and  $q_{10}$ . The state  $q_5$  is also important, as together with  $q_1$  and  $q_2$  it helps with detecting the 'consonant - vowel - consonant -  $\wedge$ ' pattern. The rest deals with rules regarding e at the end of words.

To explain what it does at each step, we shall inspect all outgoing edges from each node.

- $q_0$ : we are coming back to it until we see the first consonant in the verb. Then we move to  $q_1$

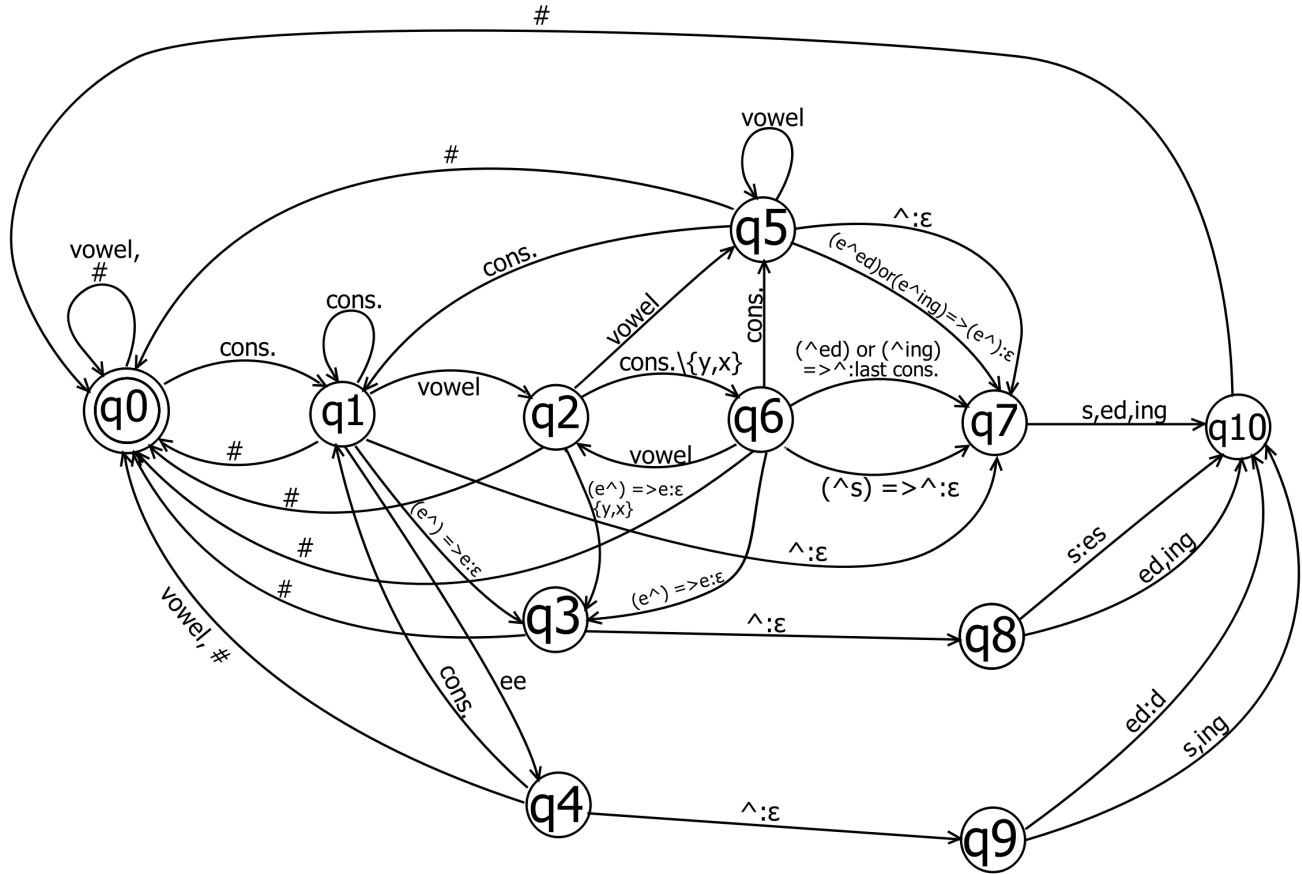


Figure 3: More robust FST

- $q_1$ : this is the state from which we begin counting the consonants. I.e. we get to it after following a consonant. If we then see a vowel and a consonant, we apply the doubling rule. Other options are: double e (in words such as "agree") or a silent e where we focus on dealing with the rules regarding this, or " $\wedge$ ".
- $q_2$ : the last consonant in a word here will get doubled, and so we proceed to  $q_6$ . However, the exceptions to this rule are the letters x and y. In these cases as well as a word-ending e (e.g. in "woe") we move to  $q_3$ . The last option is any vowel, leading to  $q_5$ .
- $q_3$ : this is the state which catches " $e \wedge$ " at the end of the stem. Note that we cannot get to  $q_3$  without seeing  $\wedge$ , so we do not need an outgoing edge marked with end of string  $\#$  character.
- $q_4$ : we treat double e separately, because in the case of past tense we lose the e in the suffix, but otherwise we don't alter the stem. There is a very subtle difference between states  $q_8$  and  $q_9$  which we will analyse later. Words such as "agree" or "need" will go through this state.
- $q_5$ : this state is part of the cycle detecting 'consonant - vowel - consonant' pattern. It will also detect the technically regular but difficult words such as "queue".<sup>1</sup>
- $q_6$ : We can only get to this state after seeing the desired "consonant - vowel - consonant" pattern, and indeed all the incoming edges are from vowels. The two most important outgoing

<sup>1</sup>It is very likely that some simplifications can be made around this state

edges describe the consonant doubling rule and lead to  $q_7$ . The others are useful for when we have not reached the end of the word after spotting the pattern.

- $q_7$ ,  $q_8$ , and  $q_9$  all deal with adding the desired suffix. In the first case, we take care of verbs ending with consonants (doubled or not). The latter two deal with words ending with e. Most of the words ending with a silent e will end up in  $q_8$ . To simplify dealing with it, we have removed the silent e, and so we can simply append the '-ed' and '-ing' suffixes. For the '-s' suffix, we need to add the e back again.  $q_9$  is concerned with words ending in double e, such as 'agree' or 'free'. In these cases we did not remove any of the e's previously, and we can simply append '-ing' and '-s', but '-ed' gets shortened to just '-d'.
- Finally  $q_{10}$  is reached after appending the desired suffix, and the only character we expect to see here is the end of string character, so we send the input to the accepting state  $q_0$ .

Here are some example of words accepted by this FST:

- "aim#" (aim):  $q_0 \rightarrow q_0 \rightarrow q_0 \rightarrow q_1 \rightarrow q_0$
- "aim^ing#" (aiming):  $q_0 \rightarrow q_0 \rightarrow q_0 \rightarrow q_1 \rightarrow q_7 \rightarrow q_{10} \rightarrow q_0$
- "stop^s#" (stops):  $q_0 \rightarrow q_1 \rightarrow q_1 \rightarrow q_2 \rightarrow q_6 \rightarrow$  (following the bottom edge, no doubling)  $q_7 \rightarrow q_{10} \rightarrow q_0$
- "stop^ed#" (stopped):  $q_0 \rightarrow q_1 \rightarrow q_1 \rightarrow q_2 \rightarrow q_6 \rightarrow$  (following the top edge, doubling)  $q_7 \rightarrow q_{10} \rightarrow q_0$
- "free^ed#" (freed):  $q_0 \rightarrow q_1 \rightarrow q_1 \rightarrow q_4 \rightarrow q_9 \rightarrow$  (following the top edge, to avoid triple e)  $q_{10} \rightarrow q_0$
- "like^ing#" (liking):  $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_6 \rightarrow q_3 \rightarrow q_8 \rightarrow$  (following the bottom edge)  $\rightarrow q_{10} \rightarrow q_0$
- "queue^ed#" (queued):  $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_5 \rightarrow q_5 \rightarrow q_7 \rightarrow q_{10} \rightarrow q_0$

### 3 Part D

We are given the corpus:

```
<s> I am Sam </s>
<s> Sam I am </s>
<s> I am Sam </s>
<s> I do not like green eggs and Sam </s>
```

We see the word 'Sam' 4 times, twice following 'am', and the word 'am' 3 times. The total number of tokens is 25. With  $\lambda_1 = \frac{1}{2}$  and  $\lambda_2 = \frac{1}{2}$ , the linear interpolation smoothing for unigrams and bigrams is given by:

$$\begin{aligned}\hat{\mathbb{P}}(Sam|am) &= \lambda_1 \mathbb{P}(Sam|am) + \lambda_2 \mathbb{P}(Sam) \\ &= \frac{1}{2} \cdot \frac{2}{3} + \frac{1}{2} \cdot \frac{4}{25} = \frac{1}{3} + \frac{2}{25} = \frac{31}{75} \approx 0.413\end{aligned}$$