Computational Linguistics Midterm
Jean Mark Gawron
San Diego State University
Linguistics/CS 581
March 20, 2018
HARDCOPY ANSWERS DUE: April 5, 2018 (Start of class)

1 FSAs and Regular Languages

1.1 Problem 1

Draw an FSA corresponding to both of the following regular expressions (assume the alphabet is a,b):
(1.1.1) $(a + b?)+$
(1.1.2) $(bab?)+$

1.2 Problem 2

Draw an FSA that accepts the complement of

$$ba+!$$

That is, it accepts the complement of the sheep language. (assume the alphabet is a,b, !): For example, it should accept “baa”, “ba!”, “baaaab!”, and any string beginning with “a” or “!”, while failing on “baaa!”

2 Problem 3

Compute the minimal edit distance for the following string pair:

rile irddle

Note: Your answer **must** use the minimal edit distance algorithm and include an edit distance table of the sort presented in the textbook. You **must** use Levenshtein distance. Answers not satisfying these criteria will receive a score of 0, even if the distance given is correct.

Your answer must also include an alignment consistent with your minimal edit table. Use 0 for the empty string in an alignment. Show the cost under each pair in the alignment. For example in the homework problem that asked you to align *divers* with *drive*, the minimal edit distance was 3 and a a minimal cost alignment was:

```
d 0 i v e r s
d r i v e 0 0
- - - - - - -
0 1 0 0 0 1 1
```

3 Problem 4: XFST

Attention: For this part of the midterm, and only this part, you will email me your answer.

The purpose of this problem is to create another spelling rule. Consider the English *e*-drop spelling rule relating forms like the following:

1. realize + ed = realized
2. realize + ing = realizing
3. realize + er = realizer
4. hope + ing = hoping
5. face + ing = facing
6. flee + ing = fleeing
7. see + ing = seeing

Using XFST, construct a rule that handles the deletion of *e* in these cases (example output below). You will email me your rule in the form of an xfst script file named "edeletion.script." You should be able to load "edeletion.script" by executing the following command from the xfst prompt:

```
xfst[0]: source edeletion.script
```

Executing the source command will confirm that your file is readable.

Notice that there are *e*-deletion does not happen with all suffixes.

1. realize + s = realizes
2. hope + s = hopes
3. face + s = faces

To get full credit on this problem, you will only need to handle the suffixes *-ing, -ed, -er* (illustrated above) and the non-edeletion suffix *-s.* One way to accomplish most of what you need is to have the "edeletion.script" file end with the following lines, which we'll call Option A:

```
define Lex [{hope}|{realize}|{flee}|{see}|{sing}|{singe}];
define Suf [ %+Prog:{ing} | %+Pres3PSg:s | %+Past:{ed} | %+Agentive:{er} ];
define MorphLex [Lex (Suf)];
define FullLex MorphLex .o. edeletion ;
push FullLex
apply up < edeletion.txt;
```

Option A makes one assumption: earlier in the file, you define a rule named "edeletion". That part is your job. So basically, all you are being asked to do in this problem is write one rule one-line long called "edeletion".

A note on "+": The definition of Suf above makes "+" part of a complex symbol in the **upper language**. This means you cannot refer to it in your edeletion rule, because the environment of your rule refers to the lower language only. The rules for einsertion given in Jurafsky and Martin and this XFST tutorial do refer to "+". If you choose to refer to "+" in your edeletion rule, you must also introduce "+" everywhere else it is needed in the lower language. This leads to Option B as an alternative to option A:

```
define Lex [{hope}|{realize}|{flee}|{see}|{sing}|{singe}];
define Suf [ %+Prog:{ing} | %+Pres3PSg:s | %+Past:{ed} | %+Agentive:{er} ];
define MorphLex [Lex (0:%+ Suf)];
define delplus %+ -> [. .];
define FullLex MorphLex .o. edeletion .o. delplus;
push FullLex
apply up < edeletion.txt
```

Since "+" is a symbol of the lower language in Option B, option B must also include a rule deleting it. This is necessary in order to analyze words with no "+" in them (See the discussion of implementing einsertion Jurafsky-Martin style in the XFST tutorial).

To summarize, you can take two approaches to writing the edeletion rule. One assumes an edeletion rule that does not mention "+"; that's option A. Another approach assumes an edeletion rule that does mention "+". That's option B. One of these options is better. The superiority of one option over the other will become clear if you can find an appropriate example word. Let's call that example word your **secret word**. Your secret word need not be an actual word of English, although it can be. In either case you should add it to the definition of **Lex** to determine which option works better.

Both options A and options B facilitate testing your machine by loading a text file that contains words you want to transduce, and both assume the text file is named edeletion.txt. That file should just be a plain text file, one word per line, consisting of the words you want your transducer to test. Here are some words you should have in edeletion.txt:

```
hoping
hopeing
hope
hopes
```

```
hops
hoped
hopeed
hope
realizer
realizing
realized
realizeed
fleeing
seeing
xxxing
```

If you put that file in the same directory you start XFST in, your edeletion.script will load it and run it, if it includes the line

```
xfst[1]: apply up < edeletion.txt
```

Both options A and B end with that line.

So if your edeletion rule is correctly specified, you should see the following output when you source edeletion.script:

```
hoping
hope+Prog

hopeing
???

hopes
hope+Pres3PSg

hops
???

hoped
hope+Past

hopeed
???

hope
hope
```

```
realizer
realize+Agentive

realizing
realize+Prog

realized
realize+Past

realizeed
???

fleeing
flee+Prog

seeing
see+Prog

xxxing
???
Closing file edeletion.txt...
xfst[1]:
```

As suggested by the above output, a correctly defined transducer will not only correctly analyze words using the edeletion spelling rule (like *realized*). It will also rule out:

hope + ing = hopeing
hope + s = hops
realize + ed = realizeed

Note: the output above does not show *hops* analyzed as *hop* + *s* because *hop* is not in the lexicon. As shown by the "xxxing" example, the analyzer will not strip the affixes off when the resulting stem is not in **Lex**.

When you have decided which option is better, option A or option B, and you have found a secret word that demonstrates it, you should add that word to the list in "edeletion.txt".

By email, you should turn in a text XFST script file named "edeletion.script" and the file "edeletion.txt" (with your secret word). The edeletion.script file should implement **one** of the options, A or B. In your email, you should say in a sentence or two which option you've chosen, option A or option B, and why.

4 HMM Problems

Go here for the HMM tagging problems.

5 A final Thought

Good luck!