HW 1: Wreck a nice beach

Re-submit Assignment

Due Jan 29 by 11:59pm **Points** 100 **Submitting** a file upload **File Types** pdf

Part 1: Pronunciation FST

The CMU pronunciation dictionary is a resource widely used for speech recognition (speech to text) and speech synthesis (text to speech). It is a large text file containing one word (in English) per line, along with a transcription of its pronunciation. It is in some ways similar to the word pronunciation information included with standard dictionaries.

Start by going to the **CMU pronunciation dictionary webpage (http://www.speech.cs.cmu.edu/cgi-bin/cmudict)** . Read the basic information contained there, but don't worry about the download and stress information; focus mainly on the Phoneme Set section, and try to see how plain text characters are used to represent how a word is pronounced. For example, the symbol G is used to represent the g sound in "green", the symbol AA is used to represent the sound the letter o makes in "odd", and the symbol EY is used to represent the sound the letter a makes in "ate". Don't worry about remembering each of the symbols and how they sound. Just get a general sense for how the word pronunciations are represented.

Now look at the **actual dictionary file (http://svn.code.sf.net/p/cmusphinx/code/trunk/cmudict/sphinxdict/cmudict.0.7a_SPHINX_40)** .

(The dictionary starts with pronunciation for punctuation and other symbols. Scroll down to see pronunciation for words. The words are in alphabetical order.)

Examine the dictionary entries for the following four words: language, languages, student, and students. **Draw an FST that will use pronunciation representations as input, one symbol at a time, and produce the corresponding words as output.** The **input alphabet** should be the set of phonemes in the CMU dictionary page (i.e. Σ = {AA, AE, AH, AO, AW, B, CH, D, DH, ...}), and the **output alphabet** should be exactly Γ = {language, languages, student, students}. The FST should map the pronunciations to their corresponding words. (Pay close attention: the output alphabet is not the set of letters; it is *those four symbols* in the set Γ.) The output language should be Γ*.

For example, if the input string is the sequence of symbols

```
L AE NG G W AH JH AH Z
```

the output string should be

```
Languages
```

(which is, in this case, a string made of just one symbol of the output alphabet).

If the input string is the sequence of symbols

```
L AE NG G W AH JH S T UW D AH N T S
```

the output string should be

```
language students
```

(This is similar, but not the same, as the spelling transducer we say in class, where the string "c a t" maps to the output "cat".)

Part 2: Prolog FST

Consider the following FST, represented as Prolog knowledge statements . You don't need to know Prolog to understand this way to represent FSTs; this is exactly the notation seen in class. This is just an example, but it could be helpful to draw what this FST looks like. (This is not required, but recommended if you can't see exactly what the transducer does when looking at the Prolog knowledge statements.)

```
transition(1, iy, 2, eps).
transition(2, z, 3, eps).
transition(3, iy, 1, easy).
transition(1, iy, 4, eps).
transition(4, z, 5, eps).
transition(5, iy, 6, eps).
transition(6, g, 7, eps).
transition(7, ow, 8, eps).
transition(8, ih, 9, eps).
transition(9, ng, 1, easygoing).
initial(1).
final(1).
```

Each line represents a transition from one state to the other. The first argument is the state where the transition starts, the second argument is the input symbol for the transition, the third argument is the state where the transition goes, and the last argument is the output symbol. Epsilon is represented as eps, and the pronunciation symbols are lowercased, so that they are not interpreted as prolog variables (we will discuss this in class, but make sure to use lowercase for the pronunciation symbols). The initial state is state 1, and the only final state is also state 1.

Your task in part 2 is to write the FST you designed in part 1 in the prolog format illustrated above (i.e. using the transition/3, initial/1 and final/1 predicates).

(Hint: if the answers to parts 1 and 2 don't seem clear initially, try examining the example above, and draw the FST diagram that corresponds to the prolog statements provided.)

If you haven't already installed SWI prolog, this is a good time to do it. Download SWI prolog from: **https://www.swi-prolog.org/download/stable (https://www.swi-prolog.org/download/stable)** (if you have already installed SWI prolog, skip the next paragraph).

There are downloads for Windows and Mac. The Windows version has a fairly straightforward installation procedure. The main difficulty with the Mac installation is that it requires xquartz (**https://www.xquartz.org (https://www.xquartz.org)**), so you need to install xquartz first, reboot, and install SWI prolog. Any students still having trouble with prolog installation should get in touch with our TAs.

Now examine the file fst.pl, available in the HW1 folders under Files on Canvas. To see the contents of the file, you can open it with SWI prolog editor (File -> Edit). You can try to understand the main idea of how the prolog code runs a transducer expressed like the one above, but don't worry too much about the details. If you would like to understand the code in detail, please feel free to ask about it, but the important aspects of this file for now are: what it does; how to use it in prolog; how to run an FST.

What you see in this file is a few rules, defining prolog predicates. The main predicate of interest at this point is fst/2, where the "slash 2" means that the predicate fst takes two arguments. The first argument of fst/2 is the input string, and the second argument is the output string. Given a transducer expressed as prolog facts (as the example above), this predicate can tell us if a specific input string is mapped to a specific output string. Using prolog variables, we can see what output string(s) correspond to an input string, and vice-versa. We will go through this in class. For example, using the FST provided above, the input string [iy, z, iy] corresponds to the output string ["easy"] (we will use square brackets to represent our strings, and use commas to separate each symbol in our strings).

To find out more about how fst.pl works, read the following **optional** explanation of fst/2 below. Otherwise, **feel free to jump to Part 3.** You don't need to understand how fst.pl works to complete part 2.

```
fst(Input, Output) :-
  initial(State),
  go(State, Input, Output).
```

fst/2 first finds a state State that makes initial(State) true. In the example above, the knowledge base (the specification of the transducer) includes initial(1). so prolog finds that State = 1 works, and proceeds to try go(1, Input, Output), where Input and Output are exactly the same as in how go/3 appeared in the query. If the query is fst([iy, z, iy], W), prolog tries go(1, [iy, z, iy], W), which we can interpret as: the current state is 1, the input symbols are [iy, z, iy], and the output is W (which is a variable). Prolog will then try to find a value of W that makes this true.

```
go(CurrentState, [A|InString], OutString) :-
  transition(CurrentState, A, NextState, eps),
  A \= eps,
  go(NextState, InString, OutString).
```

go/3 works by first finding a suitable transition from the knowledge base. Notice that the second argument is in the form [A|InString]. That means that we expect the second argument to be a list, and the first item in the list will go in the variable A, and the rest of the list will be in the variable InString. In our example above, go(1, [iy, z, iy], W), A will be iy (the first item in the list), and InString will be [z, iy] (the rest of the list). We then try transition(1, iy, NextState, eps), and we find that our knowledge base does include two transitions that match; one of them makes NextState = 2, and the other makes NextState = 4. Prolog will try them both, and see if one of them works out. A \= eps is true if A is not eps, which means that we don't want an epsilon as the input symbol at this point. Finally, go(NextState, InString, OutString) is the recursive call that continues the transduction. We move to the next state, with what's left of the input and output.

Part 3: Beach wrecking

In this last part of the assignment you will download both fst.pl and cmudict.pl (a prolog FST built from the CMU pronunciation dictionary, with phonemes as the input symbols, and words as the output symbols, as in the example in part 2). Save fst.pl and cmudict.pl in the same directory. Open SWI-prolog and change the current directory to the directory where you saved the files. For example:

```
?- cd('/Users/sagae/Desktop/LIN177').
```

Your directory will be different. Find out what it is (if you have no idea, ask for help!), and replace it in the query above.

If that succeeds, you can now consult the files fst.pl and cmudict.pl. Consulting a file is how prolog load the contents of the file so that it can be used. Consult (load) the files like this:

```
?- [fst].
?- [cmudict].
```

(Or, instead of using the commands above, you can just consult fst.pl and cmudict.pl by using the menu bar and choosing File -> Consult.)

Note the following: (1) you always need the period at the end; (2) do not type the "?-" part, that's only there so that you know that this is meant to be typed in the interpreter window (the one that shows you the ?- prompt).

Then try the following:

```
?- fst(T, [student]).
```

That should give you T = [s, t, uw, d, ah, n, t]. You used the transducer to find the pronunciation of the word student.

Now try

```
?- fst([s, t, uw, d, ah, n, t], W).
```

You should get W = ["student"]. This time, you used the transducer to find which word has the pronunciation S T UW D AH N T. In Prolog, we can have lists enclosed in square brackets, like the list of symbols above.

Now try

```
?- fst(T, [ice, cream]).
```

Note the value of T, which is a representation of the pronunciation of ice cream. Use this pronunciation as the input:

```
?- fst(enter the input here, W).
```

(Don't type "enter the input here"; you need to get the actual list from the previous query.)

The FST will give you the words that correspond to the pronunciation provided.

Once you get a result, press the ";" key. That will give you additional results, if there are any. Keep pressing ";" until you reach the end ("fail", which doesn't means it failed in general, it just means that you ran out of results after succeeding several times).

What values of W did you get?

Finally, think about whether there are any sequences of words in English that sound like "computational linguistics". Use the same idea as in the "ice cream" example above to see if our pronunciation FST can help us find sequences of English words that sound like "computational linguistics". (You will need to use the string [computational, linguistics], instead of the string [ice, cream]).

UPDATE: If prolog displays a truncated list of sounds (a list ending with |...), download fst.pl again, and consult the updated version. It has been updated to prevent prolog from truncating output.

First, find the string of sound symbols that represents the sounds corresponding to [computational, linguistics]. Then, find the output string(s) that correspond to that string of sound symbols.

What happens? Did you find sequences of English words that sound like *computational linguistics*? List what you typed in prolog to solve this problem, and describe the result.

(And that's only part of why it's hard to *wreck a nice beach*!)

Submission

Submit a single PDF through canvas. Your PDF must include at least:

- The FST from part 1 (a picture is fine; include the picture in your PDF);
- Your code for part 2 (only the code for your fst, also included in your PDF);
- The values of W for [ice, cream] part 3;
- The answer to the [computational, linguistics] questions in boldface, including at least what you typed in prolog, and a description of what the result was.

Other details

- This is an individual assignment. You must be the sole author of the ideas in your submission and their expression.
- If you get help from anyone (excluding the instructor and TA), you must disclose it in your submission.
- You may consult external sources (e.g. books, web pages, etc.), but you may not use ideas or code from these sources without attribution.