Homework #3: CMPT-413

Distributed on Wed, Feb 2, Due on Wed, Feb 9

Anoop Sarkar – anoop@cs.sfu.ca

(1) **Machine (Back) Transliteration**

Languages have different sound inventories. When translating from one language to another, names, technical terms and even some common nouns are routinely transliterated. *Transliteration* means the replacement of a loan word with some approximate phonetic equivalents taken from the sound inventory of the language. These phonetic equivalents are then written in the script of the language.

For example, the word "computer" in English comes out sounding as "konpyutaa" in Japanese, which is written using the katakana syllabic script typically used for loan words. Here are some more examples of transliteration:

Angela Johnson
アンジラ・ジョンソン
(anjira jyonson)

New York Times
ニューヨーク・タイムズ
(nyuuyooku taimuzu)

ice cream
アイスクリーム
(aisukuriimu)

Your task in this assignment is to back transliterate from Japanese into English. To avoid dealing with encoding issues with the Japanese script, we will assume that the input will be Japanese sound sequences rather than katakana characters. There is a simple mapping between katakana and these sound sequences shown in Figure 1.

Note that we will solve this problem for only two particular Japanese inputs. However, the technique we use can be extended to solve the full task of transliteration from Japanese to English simply by adding more entries for English/Japanese words in the appropriate place in our model.

ア (a)	カ (ka)	サ (sa)	タ (ta)	ナ (na)	ハ (ha)	マ (ma)	ラ (ra)
イ (i)	キ (ki)	シ (shi)	チ (chi)	ニ (ni)	ヒ (hi)	ミ (mi)	リ (ri)
ウ (u)	ク (ku)	ス (su)	ツ (tsu)	ヌ (nu)	フ (hu)	ム (mu)	ル (ru)
エ (e)	ケ (ke)	セ (se)	テ (te)	ネ (ne)	ヘ (he)	メ (me)	レ (re)
オ (o)	コ (ko)	ソ (so)	ト (to)	ノ (no)	ホ (ho)	モ (mo)	ロ (ro)
バ (ba)	ガ (ga)	パ (pa)	ザ (za)	ダ (da)	ァ (a)	ヤ (ya)	ャ (ya)
ビ (bi)	ギ (gi)	ピ (pi)	ジ (ji)	デ (de)	ィ (i)	ヨ (yo)	ョ (yo)
ブ (bu)	グ (gu)	プ (pu)	ズ (zu)	ド (do)	ゥ (u)	ユ (yu)	ュ (yu)
ベ (be)	ゲ (ge)	ペ (pe)	ゼ (ze)	ン (n)	ェ (e)	ヴ (v)	ッ
ボ (bo)	ゴ (go)	ポ (po)	ゾ (zo)	ヂ (chi)	ォ (o)	ワ (wa)	ー

Figure 1: Mapping from Japanese katakana characters to pronunciations.

Let us consider the model where we generate a Japanese sound sequence for given English word sequence. Once we have such a model we can run it backwards to produce all valid English word sequences for any input Japanese sound sequence.

In order to generate a Japanese sound sequence for a given English word sequence we take the following intermediate steps:

1. Represent all valid English words that can be used in an English word sequence.
2. Convert each English word to it's pronunciation (mapping English words to phonemes). For example, the English pronunciation of the word sequence *golf ball* is `G AA L F B AO L`.
3. Convert English pronunciations to their equivalent Japanese pronunciations. For example, the sound sequence `G AA L F B AO L` would be converted to the equivalent Japanese sound sequence `g o r u h u b o o r u`.

Finally we can invert the above steps so that it takes a Japanese sound sequence as input and produces English word sequences as output.

We can represent each of the above steps as a finite-state transducer. We will do this using the AT&T `fsm` toolkit. The advantage is that we can use the standard transducer operations which have been implemented in this toolkit. For example, we can use the program `fsmcompose` to compose the finite state transducers described above into a single transducer which computes the conversion from Japanese sound sequences into English words.

We stop here for this assignment, but in general for this approach to be useful for text-to-text back transliteration we will have to build a generative model mapping Japanese sound sequences into the katakana script.

You will be using the following data files for each intermediate step in the process of mapping English words to Japanese sound sequences:

- `sample-engwords.txt` – English words used for this assignment
- `sample-cmudict.txt` – Pronunciation dictionary for the English words used in this assignment
- `epron-jpron.map` – Mapping from English pronunciations to Japanese pronunciations
- `jpron-input1.txt` and `jpron-input2.txt` – Sample input files. Contains pronunciations of Japanese words.

In order to do back transliteration from Japanese sound sequences into English words, you will have to convert the above files into the AT&T fsm toolkit format. You will need to create a text file with the finite state machine/transducer and an additional file with the mapping between the input/output symbols used and a unique number for each symbol (the `.syms` file). *An important thing to keep in mind is that the same token must get the same number in the* `.syms` *file across all these finite-state trandsducers.* This is how the output alphabet of one transducer is recognized to be the same as the input alphabet of another transducer.

This conversion is not entirely trivial. For example, consider the fragment of the pronunciation dictionary given below:

```
GOLF  G AA1 L F
BALL  B AO1 L
```

The numbers `0,1,2` are to be removed from this input (they represent stress markers which are not needed for this task). The first column is the English word and the remaining columns contain the pronunciation for that word.

Once you write your code to convert the above format into the AT&T fsm toolkit, the transducer text file should look like this (call it `tiny.stxt`):

```
0 2 <eps> G
2 3 <eps> AA
3 4 <eps> L
4 5 <eps> F
5 1 GOLF <eps>
0 7 <eps> B
7 8 <eps> AO
8 9 <eps> L
9 1 BALL <eps>
1 0 <eps> <eps>
1 0 <eps> PAUSE
1
```

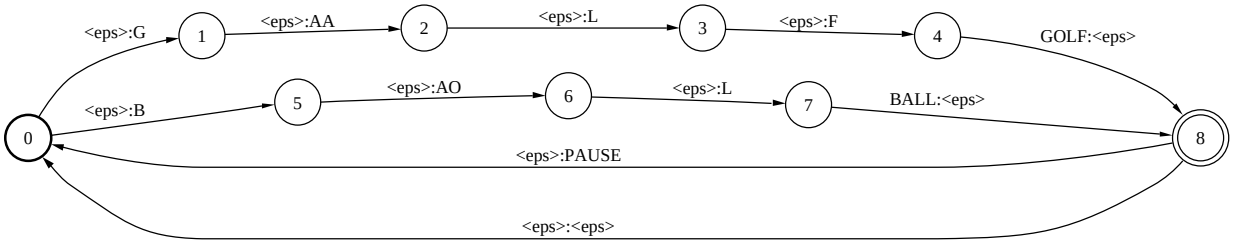
And the symbols file should look like this (called `tiny.syms`). The number `0` is reserved for the empty string, denoted here by `<eps>`:

```
<eps> 0
PAUSE 1
G 2
AA 3
L 4
F 5
GOLF 6
B 7
AO 8
BALL 9
```

Once you have these two files you can compile it to the binary format used by the AT&T fsm toolkit by using the program `fsmcompile`.

```
fsmcompile -t -i tiny.syms -o tiny.syms tiny.stxt > tiny.fsm
```

The `-t` indicates that the input file is a transducer and the `-i` and `-o` options indicate the symbols for the input and output alphabet respectively.
Graphically the transducer looks like this:[1]



Once you have compiled all the individual finite state transducers for each model, you should use the command `fsmcompose` to combine them together into one big model which maps Japanese sounds into English words.
Once this is done you can then compose this combined fsm file with a particular input Japanese sound sequence using `fsmcompose` once again. To extract the English word sequence you will need to use the following programs from the fsm toolkit (they are implementations of standard transducer algorithms):

- `fsmrmepsilon fstfile`: removes epsilon transitions from `fstfile`

[1] Using `fsmdraw -i tiny.syms -o tiny.syms tiny.fsm | dot -Tps > tiny.ps`

- `fsmprint -i symsfile -o symsfile fstfile`: prints out the compiled `fstfile` as text
- `fsmproject -1 fstfile`: projects the transducer `fstfile` to a finite-state machine over the input language
- `fsmproject -2 fstfile`: projects the transducer `fstfile` to a finite-state machine over the output language

The input Japanese sound sequence `anjirajyonson` represented as an fsm is given to you in the file `jpron-input1.txt` and the syms file is in the file `jpron-syms.txt`.

The input Japanese sound sequence `goruhubooru` represented as an fsm is given to you in the file `jpron-input2.txt` and the syms file is in the file `jpron-syms.txt`.

Submit files: `cmudictToFSM.pl` (Perl program that converts the supplied input text files into the text files required for the command `fsmcompile` in the AT&T finite-state transducer toolkit) and `createfsmfiles.sh` (a sequence of AT&T fsm toolkit commands to create each transducer in the model which can use your Perl program, plus the commands to create the composed transducer and to print the output after composition with both of the Japanese sound sequence input files).