Mehryar Mohri
Speech Recognition
Courant Institute of Mathematical Sciences
Homework assignment 2
October 09, 2012
Due: October 22, 2012

A. Pronunciation dictionaries

The following is a sample of a pronunciation dictionary. The phonemic transcription as well as the probability of the transcription is given for each word.

1. $cray \rightarrow K\ r\ ey\ (P = .9)$
2. $dance \rightarrow D\ ae\ n\ s\ (P = .85)$
3. $data \rightarrow D\ ae\ T\ ax\ (P = .5)$
4. $data \rightarrow D\ ey\ T\ ax\ (P = .5)$
5. $date \rightarrow D\ ey\ T\ (P = .85)$
6. $day \rightarrow D\ ey\ (P = .9)$

Construct a weighted transducer that represents the corresponding weighted transduction (use negative log of probabilities to assign weights to the paths, words such as 'cray' for the input alphabet, and phones such as 'K' or 'ae' for the output alphabet). Take the inverse of that transducer and make it as compact as you can.

Solution: Figure 1 shows the pronoounciation transducer. The inverse of that transducer can be made compact by first determinizing it, then encoding each pair of input-output labels as a single symbol, and then applying weighted determinzation and minimization, and finally decoding:

```
fsminvert pron.fsm | fsmdeterminize | fsmencode -l - key.fsm
    | fsmdeterminize | fsmminimize
    | fsmencode -ld - key.fsm >ipron.fsm
```

Figure 2 shows the result.

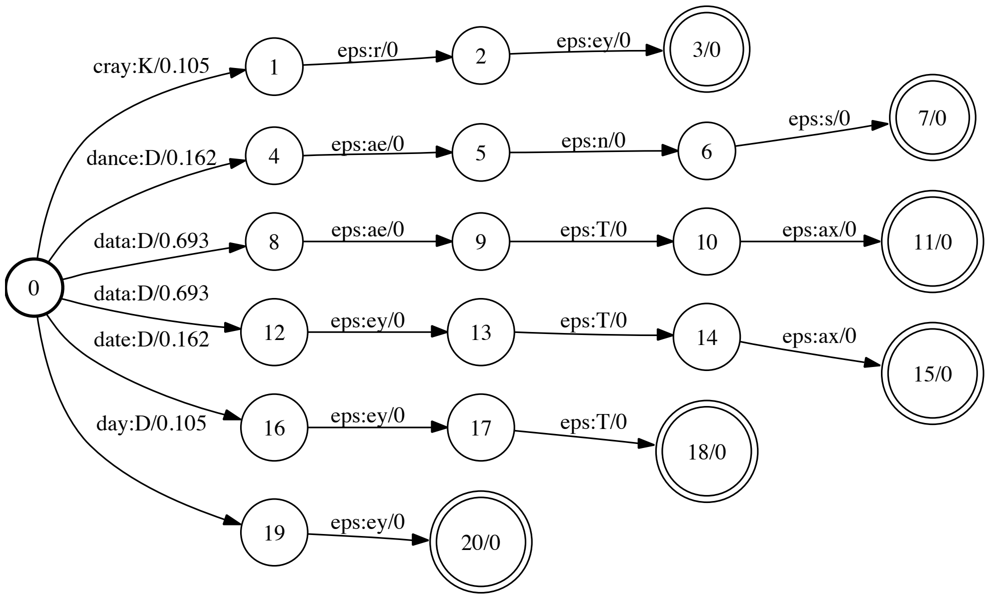


Figure 1: Pronounciation transducer.

B. Weighted Grammars

Here is a set of sentences with their corresponding probability of occurrence:

He comes (very) late, in the afternoon* (P = .2)
He comes (very) late, in the evening* (P= .3)
He will come (very) late, in the afternoon* (P = .1)
He will come (very) late, in the evening* (P = .4)

Construct a compact weighted automaton that accepts exactly these sentences (use negative log of probabilities to assign weights to the paths, use words such as 'he' to label transitions).

Solution: The solution can be obtained using weighted determinization and minimization.

C. Text Operations

1. Elementary automata. Create an automaton for each of the following questions, given the alphabet $\Sigma = \{a, b, \ldots, z, A, B, \ldots, Z, \langle space \rangle\}$:

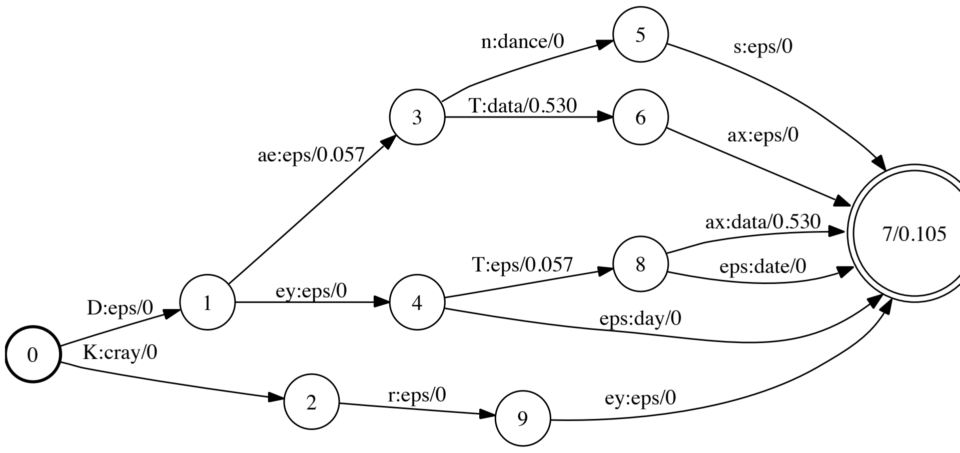


Figure 2: Compact inverse pronounciation transducer.

- (a) accepts a letter in Σ (excluding space),
- (b) accepts a single space,
- (c) accepts a capitalized word (where a word is a string of letters in Σ excluding space, and a capitalized word has its initial letter uppercase and remaining letters lowercase),
- (d) accepts a word containing the letter a.

2. Complex automata. Using the elementary automata of the previous exercise as the building blocks, use appropriate library operations on them to create an automaton that:

   - (a) accepts zero or more capitalized words each followed by a space,
   - (b) accepts a word beginning or ending in a capitalized letter,
   - (c) accepts a word that is capitalized and contains the letter a,
   - (d) accepts a word that is capitalized or does not contain an a,
   - (e) accepts a word that is capitalized or does not contains an a (this should be done without using the union operation of the library).

3. Optimizations. Epsilon-remove, determinize, and minimize each of the automata constructed in the previous question. Give the number of states and arcs before and after these operations.

Solution: The solution to the questions of this problem are straightforward applications of the software library.

D. Trim automata

Consider the automaton:

```
0 1 1
0 2 2
1 1 1
2
3 4 4
4 3 3
4
```

1. How many states can be reached from the initial state?

   Solution: Figure 3(a) shows a graphical representation of this automaton.

   Using `fsminfo` and the given textual representation `trim.txt`, we can determine both the number of accessible and coaccessible states:

   ```
   $ fsmcompile trim.txt | fsminfo -n
   class                    basic
   semiring                 tropical
   transducer               n
   # of states              5
   # of arcs                5
   initial state            0
   # of final states        2
   # of eps                 0
   # of accessible states   3
   # of coaccessible states 4
   # of connected states    2
   # strongly conn components 4
   ```

   There are 3 accessible states: 0, 1, and 2.

2. How many states can reach a final state?

   Solution: There are 4 coaccessible states (every state except from 1).

3. Compile this automaton and then remove all useless states.

   Solution: This can be done using `fsmconnect`:

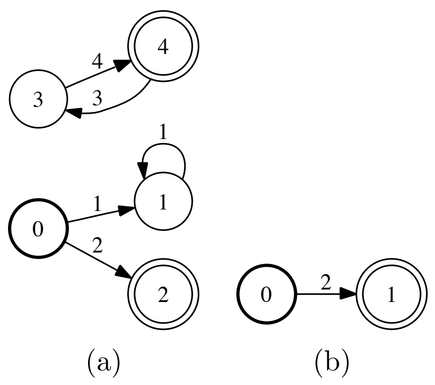


Figure 3:

```
$ fsmcompile trim.txt | fsmconnect > trim.fsm
```

Figure 3(b) shows a graphical representation of the automaton after connection.

E. Codes

Given the alphabet $\Sigma = \{a, b, \ldots, z, \langle \mathit{space} \rangle\}$,

1. create a transducer that implements the *rot13* cipher – $a \rightarrow n, b \rightarrow o, \ldots, m \rightarrow z, n \rightarrow a, o \rightarrow b, \ldots, z \rightarrow m$,

   Solution: Figure 4 shows that transducer.

2. encode the message `"my secret message"` (assume $\langle \mathit{space} \rangle \rightarrow \langle \mathit{space} \rangle$),

   Solution: The message can be encoded using composition as follows:

```
$ echo "m y <space> s e c r e t <space> m e s s a g e" \
    | farcompilestrings -i rot13_syms.txt | fsmcompose - rot13.fsm \
    | fsmproject -2 | fsmprint -i rot13_syms.txt \
    | gawk '{ printf $3}END{printf "\n"}' \
    | sed -e 's/\<space\>/ /g'
zl frperg zrffntr
```

3. decode the encoded message from above.

   Since the inverse of the transducer is itself, the transducer itself can be used for decoding:

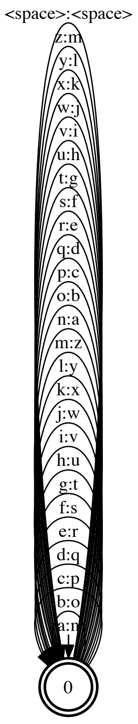


Figure 4:

```
$ echo "z l <space> f r p e r g <space> z r f f n t r" \
    | farcompilestrings -i rot13_syms.txt | fsmcompose - rot13.fsm \
    | fsmproject -2 | fsmprint -i rot13_syms.txt \
    | gawk '{ printf $3}END{printf "\n"}' \
    | sed -e 's/\<space\>/ /g'
my secret message
```

F. Numbers

Given the alphabet $\Sigma = \{0, 1, \ldots, 9\}$,

1. create an automaton that accepts numbers in the range $0 - 999999$.

   Solution: See automaton of Figure 5.

2. create a transducer that maps numbers (in the range $0 - 999999$) represented as strings of digits to their English read form, e.g.,

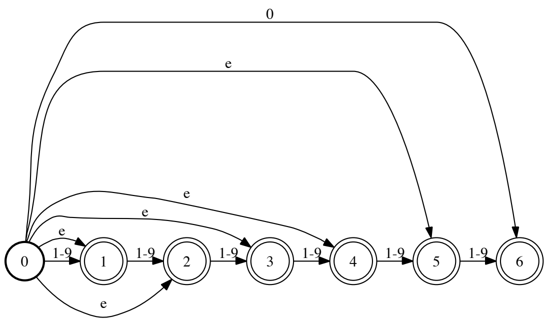


Figure 5:

1 → one
11 → eleven
111 → one hundred eleven
1111 → one thousand one hundred eleven
11111 → eleven thousand one hundred eleven

Solution: The transducer T can be constructed using rational operations. Start with a digit transducer D mapping single-digit numbers to their English expressions. Similarly, construct a transducer T_1 mapping numbers $11 - 19$ to their English expressions and T_2 mapping $10, 20, \ldots, 90$ to their English form, etc.

3. Randomly generate several numbers both as strings of digits and in their read form.

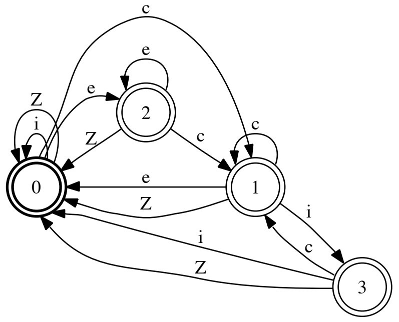
   Solution: Use `fsmrandgen`.

G. Spelling

Given the alphabet $\{a, b, \ldots, z, \langle space \rangle\}$, create a spelling corrector transducer that implements the (imperfect) traditional rule – 'i before e except after c'. Use it to correct the inputs 'yeild' and 'reciept'.

Solution: To simplify the presentation, we can consider the alphabet $\Delta = \{c, e, i, Z\}$. Z can be replaced with all the elements of the English alphabet and $\langle space \rangle$ except from c, e, and i.

Note that the set of forbidden sequences can be described by the regular expression $F = \Delta^*(cie)\Delta^* + \Delta^*(\Delta - \{c\})(ei)\Delta^* + (ei)\Delta^*$. Using `fsmdifference`, we can



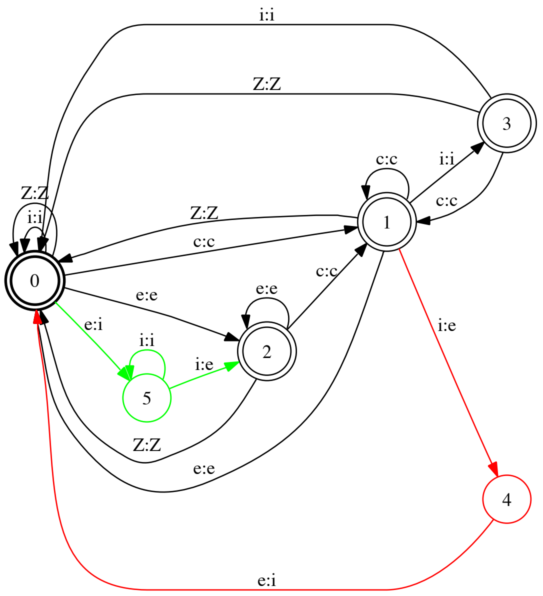


Figure 6: (a) Automaton of admissible sequences. (b) Spelling correction transducer.

compute the minimal deterministic automaton representing the set of admissible sequences $\Delta^* - F$:

```
$ fsmdifference Deltastar.fsm forbidden.fsm | \
     | fsmrmepsilon | fsmdeterminize | fsmminimize > admissible.fsm
```

The result is shown in Figure 6(a) . That automaton can be augmented to define an identity transducer restricted to that domain by augmenting it with output labels identical to the input ones. To allow the forbidden sequence, it then suffices to add a distinct path from state 1 mapping *ie* to *ei*. Figure 6(b) illustrates that construction.

A general method for constructing such transducers consists of compiling context-dependent rules into transducers.

H. Roman numerals

Given the alphabet $\{I, V, X, L, C, D, M\}$,

1. create a weighted automaton that assigns to Roman numerals their numeric value (hint: use `fsmbestpath`).

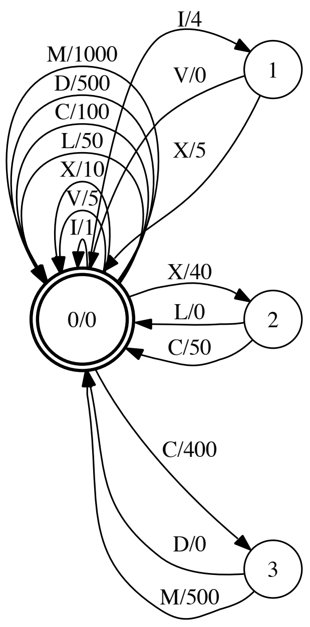


Figure 7: Weighted automaton computing the value of Roman numerals in the tropical semiring.

Solution: Observe that since the value of IV is less than that of I plus V and similarly with IX, XL, XC, CD, and CM. In view of this observation, the weighted automaton of Figure 7 defined in the tropical semiring is a solution. Its binary representation `roman.fsm` can be used as follows to compute the value of a Roman numeral if `romansyms.txt` is a textual symbols file for the Roman alphabet:

```
$ echo "M C M X L I V" | farcompilestrings -i romansyms.txt \
     | fsmcompose - roman.fsm | fsmbestpath | fsmpush -ic \
     | fsmprint | gawk 'NF<=2 {print $2}'
1944
```

2. ϵ-remove, determinize, and minimize this automaton. Draw the automaton before and after these operations.

Solution: The automaton admits no ϵ-transition. Using determinization and minimization:

```
$ fsmcompile -i romansyms.txt roman.txt \
    | fsmdeterminize | fsmminimize >roman_min.fsm
```

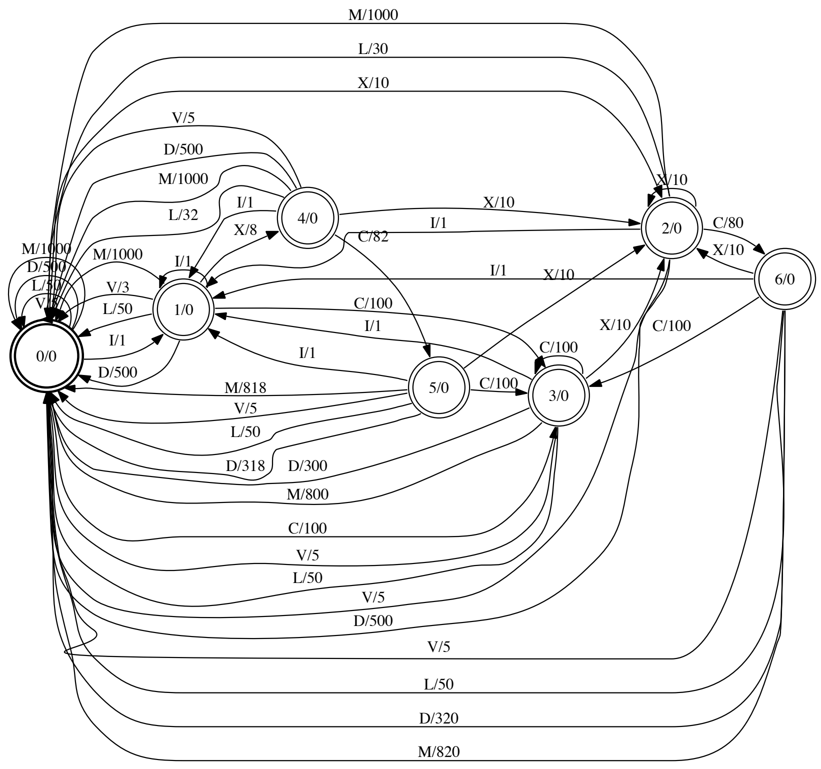


Figure 8: Deterministic and minimal weighted automaton computing the value of Roman numerals in the tropical semiring.

leads to the weighted automaton of Figure 8.

I. Genome

Given the alphabet $L = \{A, G, T, C\}$,

1. create a transducer T that implements the following edit distance:

$$\begin{array}{l} d(x,x) = 0, x \in L \\ d(x,\epsilon) = d(\epsilon,y) = 1, x \in L \quad . \\ d(x,y) = 1.5, x \neq y \in L \end{array}$$

Solution: Create the following textual symbols file 'lab':

```
e 0
```

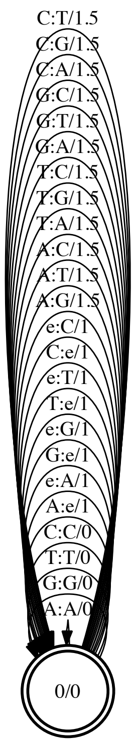


Figure 9: Edit-distance transducer defined in the tropical semiring.

```
A 1
G 2
T 3
C 4
```

Use that to create a textual representation 'tedit.txt' of the edit-distance transducer shown in Figure 9. Compile that to creat its binary representation:

```
$ fsmcompile -ilab -olab -t tedit.txt > tedit.fsm
```

2. using T, find the best alignment between the strings 'AGTCC' and 'GGTACC'

   Solution: Create two automata representing each 'AGTCC' and 'GGTACC':

```
$ echo "A G T C C" | farcompilestrings -ilab > agtcc.fsm
$ echo "G G T A C C" | farcompilestrings -ilab > ggtacc.fsm
```

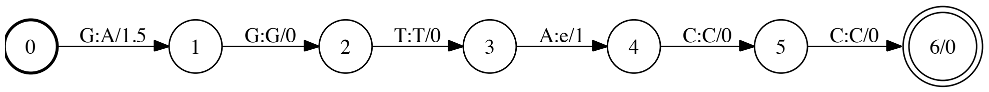


Figure 10: Transducer showing the best alignment of ‘AGTCC’ and ‘GGTACC’, whose cost is 2.5.

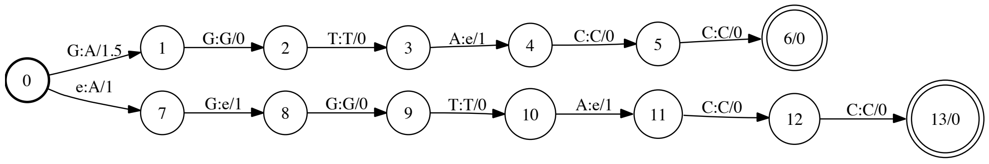


Figure 11: Transducer showing the two best alignments of ‘AGTCC’ and ‘GGTACC’, whose costs are 2.5 and 3.

Use composition and fsmbestpath to determine the best alignment:

```
$ fsmcompose ggtacc.fsm tedit.fsm agtcc.fsm \
      | fsmbestpath > best.fsm
```

3. find the second best alignment

   Solution: Similarly, use composition and fsmbestpath to determine the best alignment:

```
$ fsmcompose ggtacc.fsm tedit.fsm agtcc.fsm \
      | fsmbestpath -n2 > best2.fsm
```