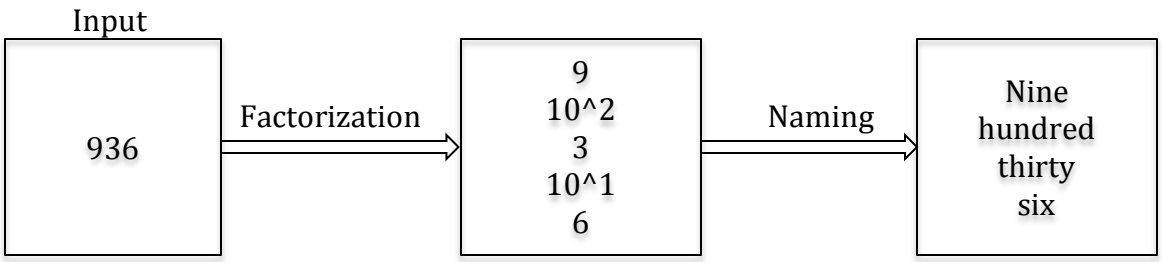
CS 655: Analyzing Sequences
Homework 2

Part 1: Number-name generator

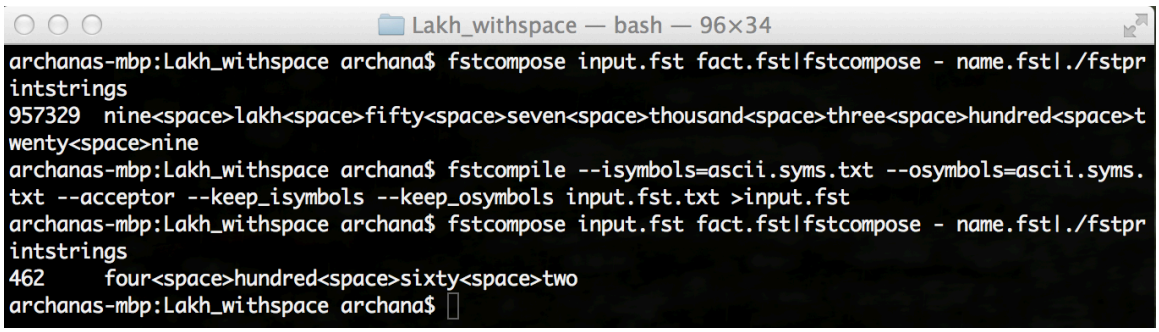
For this part I generated number names in three languages: English, Hindi and Telugu (South Indian language).

English:

I used OpenFST to generate number names upto 9,99,999 (nine lakh ninety nine thousand nine hundred ninety nine). In order to do this I first split the number to its powers of 10 and then converted the numbers and powers to their corresponding names. The final number name is generated by composing the input with the factorization transducer and then with the naming transducer.



After implementing the number name generator, when I added spaces between the names, it worked fine when there are no zeros, but when there are zeros it resulted in some extra <space>s. Removing the extra spaces by tracking the nodes was becoming tedious. By this time I had implemented the other two parts in Thrax and realized that it is much simpler to do it in Thrax. So I shifted to Thrax for proper implementation with spaces.

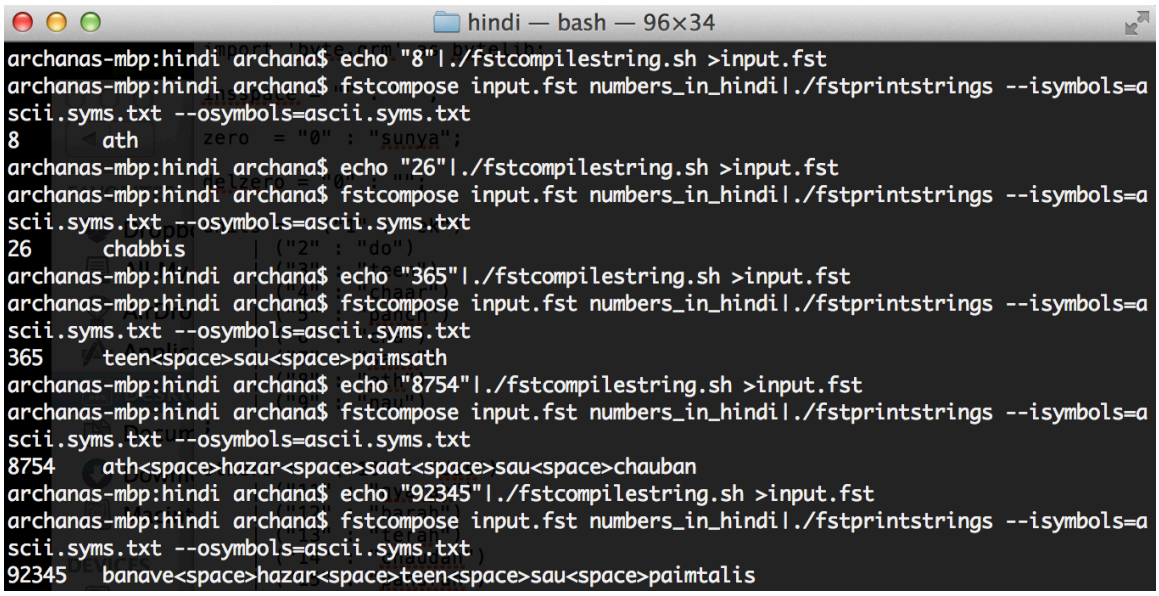


Hindi:

The Hindi number-naming system is quite different from English for numbers from 21 to 99. In Hindi there is a separate name for each number from 21 to 99, with not much of an obvious pattern like English. I implemented the number-name generator for Hindi in two ways, one using the full name directly (hindi.grm) and the other by grouping the similar strings at the end of the word (hindi2.grm).

30 - Tis
31 - Ikatis
32 - Battis
33 - Taimtis
34 - Chaumtis
35 - Paimtis
36 - Chattis
37 - Saimtis
38 - Aratis
39 - unchalis

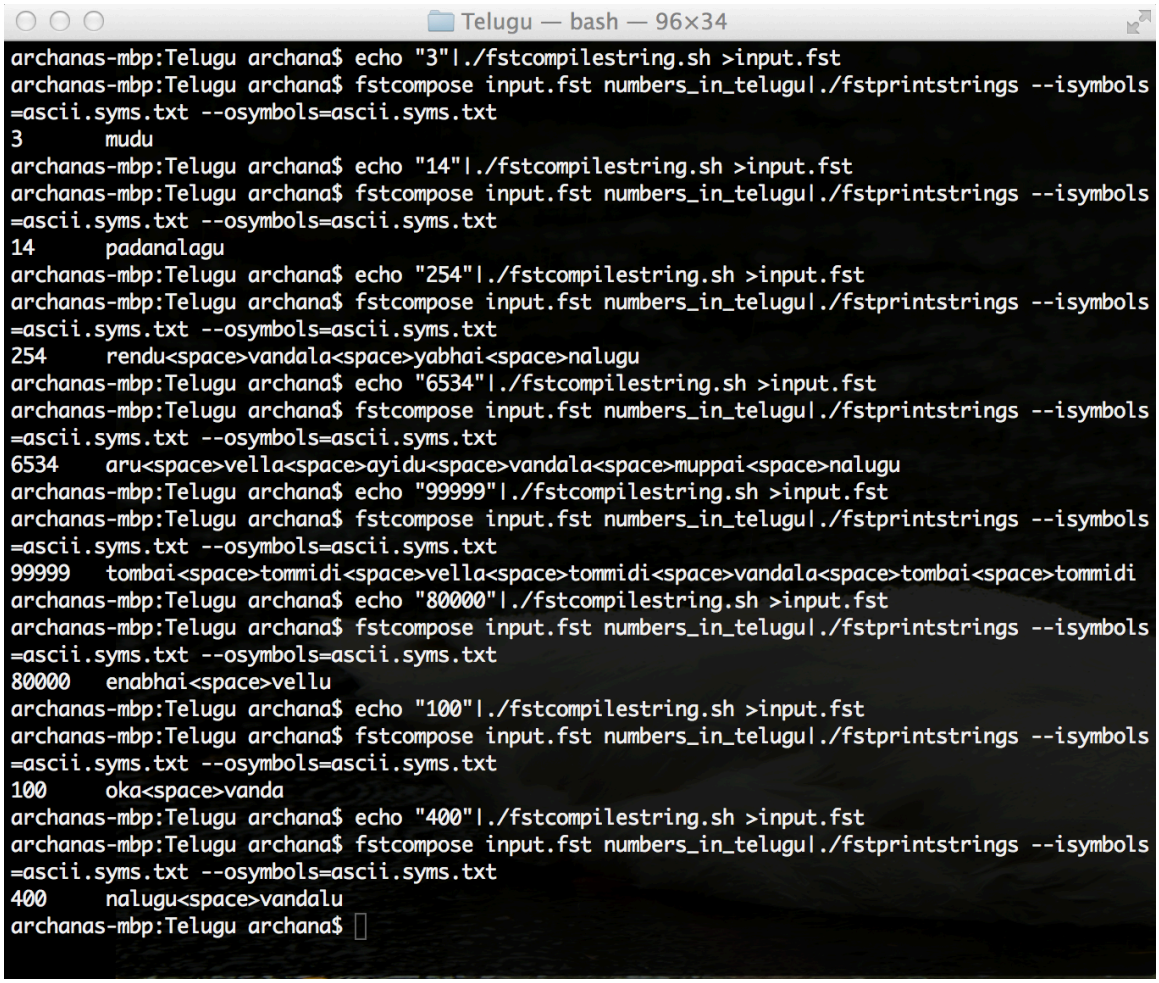
40 - Chalis
41 - Ikatalis
42 - Bayalis
43 - Taimtalis
44 - Chaumtalis
45 - Paimtalis
46 - Chiyalis
47 - Saimtalis
48 - Aratalis
49 - unachas



Telugu:

The Telugu number-naming system is very similar to English, but differs in the way hundreds and thousands are named. They vary based on number present in the hundreds or thousands place and also on the numbers following them. This is illustrated below:

100 - oka vanda
200 - rendu vandalu
231 - rendu vandala muppai oka
1000 - oka veyyi
2000 - rendu vellu
2400 - rendu vella nalugu vandalu

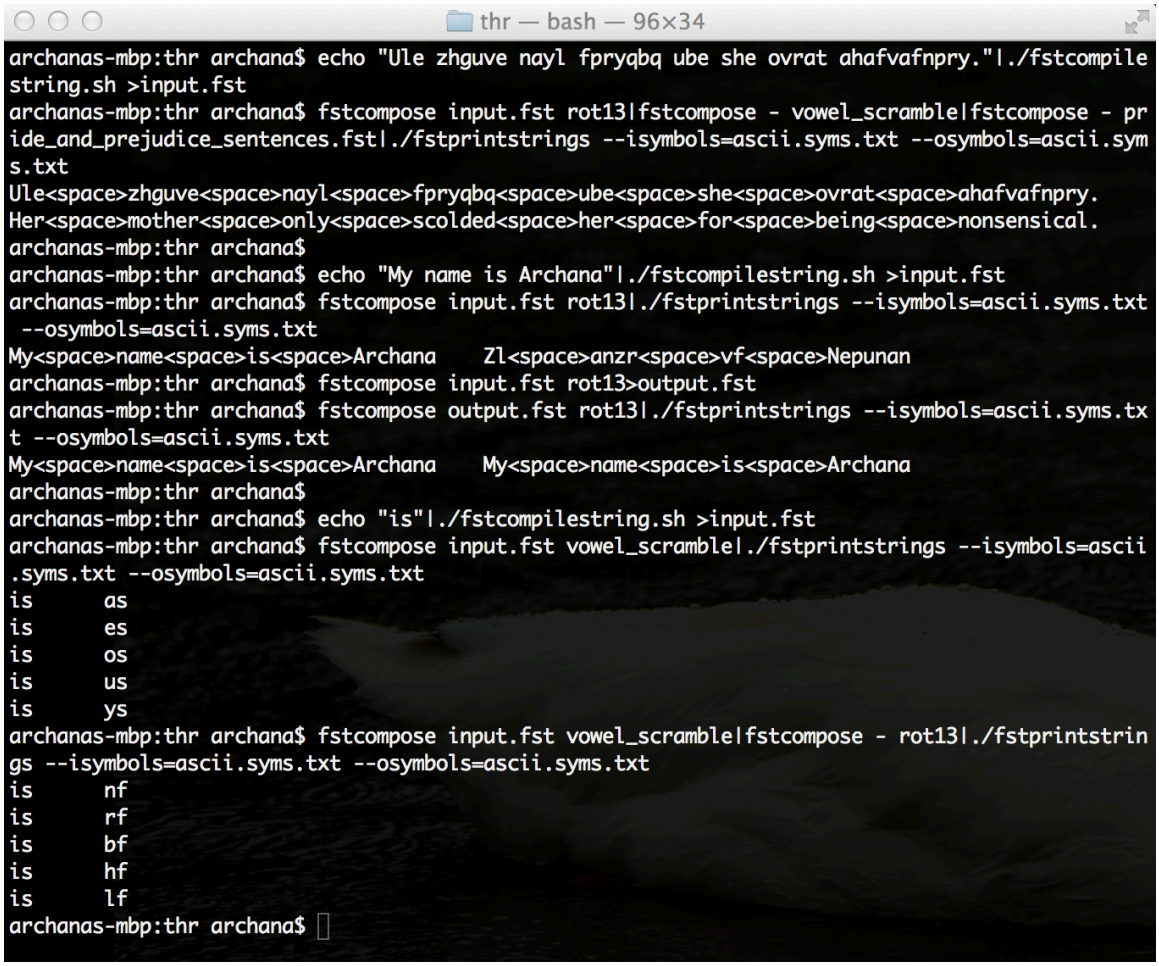


Part 2: ROT-13

The decoded sentence is

`Her mother only scolded her for being nonsensical.`

In order to decode the sentence, the FST of the input sentence was first composed with the ROT-13 FST, which deciphered the ROT-13-encoded message. It was then composed with the scrambling FST to produce sentences with all possible combination of vowels. This FST of sentences is in turn composed with `pride_and_prejudice_sentences.fst` to find a match to produce the final decoded sentence.



Part 3: T9 Text Entry

A transducer is defined to replace a number with its corresponding letters as per the telephone keypad code ("2" with "a", "b" or "c"; "3" with "d", "e" or "f", and so on). For a given group of numbers, composing with the above-defined FST gives all possible combinations of the letters that the numbers represent. Composing these combinations of letters with a wordlist containing valid English words, yields all possible English-language words those digits could represent.

The predictor can be made more accurate by decreasing the weight of the words that occur more commonly, and increasing the weight of words that occur rarely. For example when the user types '843', the possible outputs are *the, tid, tie* and *vie* as shown below. But most of the time the user would end up using it for the word *the*. Decreasing the weight for path *the* will increase the chance of predicting '843' as *the.*