Practice Exam Problems: Finite-State Machines

Natural Language Processing (JHU 601.465/665)

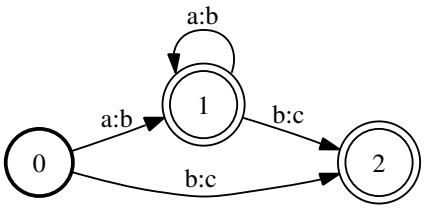
Prof. Jason Eisner

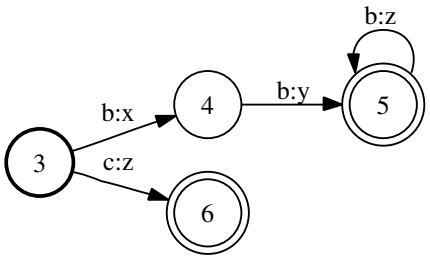
1. (a) Using xfst notation, write a regular expression for a transducer that will insert commas into an integer for readability, breaking the digits into groups of three in the standard way. For example, it should map the small integer `103` to `103` but map the large negative integer`-14321060` to `-14,321,060`.

   You may assume that `Digit` is a predefined regular expression that matches the class of digits (`0` through `9`).

   (b) Let `InsCommas` denote a regular expression that solves question (a). You wish to build a transducer that reads an arbitrary piece of text and inserts commas into all the integers that appear in that text. Write a regular expression for such a transducer in terms of `InsCommas`. (*Hint:* Directed replacement may or may not be the right approach.)

2. [6 points] Draw the composition of these two finite-state transducers. (The output of the left FST serves as input to the right FST.)





3. You have two large finite-state acceptors (FSAs). One has n states; the other has m states. You intersect the two FSAs, hoping that this will not take too long or consume too much memory.

   (a) [3 points] What is the *maximum* number of states that the result could have? Why?

   (b) [3 points] What is the *minimum* number of states that the result could have? When would you be so lucky as to achieve this minimum?

4. In this question, you will use an FST to detect rhyming.

   (a) [3 points] As warmup, write an XFST-style regular expression for an regular relation that matches a pair of strings if and only if they end in the same three letters.

   (Thus, your regexp would compile into an FST that will accept the pair (`moon`, `moon`) or (`moon`, `balloon`) but reject the pair (`moon`, `pylon`). I don't really care whether it matches (`on`, `on`)—don't worry about strings shorter than 3.)

   (b) [4 points] Now, suppose someone gives you an FST called Ending that transduces a word or phrase to its "rhyming ending." Two words rhyme if they have the same rhyming ending.

   The Ending FST is not trivial. Here are some pairs that it accepts:[1]

(`money`, `unny`)	(`moony`, `oony`)
(`funny`, `unny`)	(`loony`, `oony`)
(`sunny`, `unny`)	(`Rooney`, `oony`)
	(`Sunni`, `oony`)
(`moon`, `oon`)	
(`spoon`, `oon`)	(`scooter`, `ooder`)
(`balloon`, `oon`)	(`commuter`, `ooder`)
	(`computer`, `ooder`)
(`tablespoon`, `ablespoon`)	(`microcomputer`, `ooder`)
(`boomerang`, `oomerang`)	
(`kangaroo meringue`, `oomerang`)	

   Write a regular expression Rhyme that matches a pair of words if they have the same rhyming ending. This shouldn't be too hard because your regexp can make use of Ending (think of it as a "subroutine").

   (*Hint:* The words and their rhyming endings are quite different kinds of strings (spelling and pronunciation). Your solution should not act as if they used the same alphabet. It should work even with a better version of Ending that transduced from the Roman alphabet to the International Phonetic Alphabet, i.e., (`microcomputer`, ur∂r) rather than (`microcomputer`, `ooder`).)

[1] One of the rhymes shown appears in a poem by Ogden Nash (1902–1971):

O kangaroo, O kangaroo,
Be grateful that you're in the zoo
And not transmuted by a boomerang
To zestful, tasty kangaroo meringue.

(c) [8 points] Suppose that instead of Ending, someone only gave you Pronounce, an FST that transduces a word (or phrase) to its pronunciation. The pronunciations is written in a style that is common in dictionaries. The syllables are conveniently separated by `-`, and each stressed syllable is immediately preceded by `'`.

For example, Pronounce accepts the pair (`microcomputer`, `'mai-kro-kum-'pyoo-der`). You are also given a simple FSA, Consonant, which accepts a string just if it is a single consonant.

Write a regexp for Ending in terms of Pronounce and Consonant. You should be able to figure out from the examples in part 4b what Ending ought to do, but I'll tell you specifically:

- use Pronounce to look up the pronunciation
- delete everything before the rightmost stressed syllable
- also delete the group of 0 or more Consonants at the start of the rightmost stressed syllable (this group is called the *onset* of the syllable)
- delete all remaining `'` and `-` symbols, just to clean up the result so that it matches the behavior of the previous question

For example, your Ending regexp should accept the pair (`microcomputer`, `ooder`) as before. The pronunciation was `'mai-kro-kum-'pyoo-der`, whose rightmost stressed syllable `pyoo` has onset `py`. The transducer therefore deletes `'mai-kro-kum-'py` from the pronunciation, leaving `oo-der`, which it cleans up by deleting the `-`.

(d) [2 points] There exist words of English, such as `the` and `of`, whose pronunciations have no stressed syllables. Question 4c didn't specify how you should handle such words—it was up to you.

So how did you handle them? Specifically, according to *your* definition of Ending (in your answer to question 4c above), what will the Ending of such a word be? As a result, what words will Rhyme with it?

(e) Suppose you are trying to (approximately) express some specific ideas in a poem, under the constraint that the poem must rhyme. Suppose $\text{score}_i(w_i)$ evaluates how happy you would be to end line i with word w_i.

You would therefore like to find a pair of *rhyming* words or phrases (w_1, w_2) such that $\text{score}_1(w_1) + \text{score}_2(w_2)$ is as high as possible.

Suppose score_1 has been specified by a *weighted* FSA, Score1. That is, $\text{score}_1(w_1)$ is defined as the total weight of the maximum-weight path in Score1 that accepts w_1.[2] Similarly, score_2 has been specified by Score2.

i. [4 points] How would you actually use Score1 to compute $\text{score}_1(\texttt{kangaroo\_meringue})$? That is, what FSA algorithm or algorithms would you have to run?

ii. [4 points] How would you use Score1, Score2, and Rhyme to find the single best pair (w_1, w_2) of rhyming words or phrases, as defined above?

Note: Your solution should work even if Score1 or Score2 can accept infinitely many different strings (e.g., they might allow made-up words, or long phrases). In other words, you can't just iterate over the set of possible pairs, because that set might be infinite. You have to use actual efficient finite-state methods that you learned in class. ☺

[2] Or as $-\infty$ if there is no such path.

(f) [5 **extra credit** points] There is something a bit funny about our solution so far, because Rhyme will match (`money`, `money`) or (`computer`, `microcomputer`). These would be considered suitable rhymes in French poetry—but in English poetry, they are usually considered "identities" rather than rhymes, because the final stressed syllable has the same onset in both words.

So you would like to write a regexp StrictRhyme that accepts only rhyming pairs that are *not* identities.

It is tempting to define StrictRhyme as something like Rhyme – Identity. Unfortunately, XFST does not allow you to take the difference (or the intersection) of two regular relations. Why not? Because as it turns out, there are "bad" cases where the result would *not* be a regular relation.

However, this is not one of those "bad" cases: StrictRhyme really is a regular relation. Thus, write a regexp for StrictRhyme. You can write it directly in terms of Pronounce, without Rhyme.

Hint: First define an FST Change that will transduce any string to any *other* string. In other words, it has the meaning `[?* .x. ?*] - ?*`, even though XFST doesn't let you write that.