Analyzing FSTs

Look at the following FST, implemented using Pynini:

```
 import pynini as pn
fst = (pn.accep("a") | pn.accep("e")) \
  + pn.cross("a", pn.accep("0").closure(0, 5)) \
  | pn.cross(pn.accep("a").star, "0") + pn.accep("xxx")
```

You may use a computer to answer the following questions:

1a. Give a regular expression that is equivalent to the input language of this finite state transducer. (1 point)

1b. Give inputs that are mapped by this FST to 0 outputs, 1 output, 2 outputs, and more than 2 outputs. (4 points)

French numbers to words

Make program that uses Pynini to map numbers (represented as strings) to French words (up to 999999). Use provided `numbers2words.py` as starter.

It's a good idea to implement this with two FSTs. The first FST will factorize a number into into sequences annotated with powers of ten:

```
 0    -> 0
1    -> 1
10   -> 1^
23   -> 2^ 3
203 ->  2^^ 3
```

A second FST will convert the factorized form into words:

```
 0    -> zero
1    -> un
1^   -> dix
2^ 3   -> vingt-et-un
```

Finally, you should also handle decimals. Decimals should be verbalized digit-by-digit: 0.046 whould be converted into "zero-virgule-zero-quatre-six".

You are not expected to know French to complete this task. Use materials on the web, e.g. https://tuto.pages-informatique.com/writing-out-numbers-in-french-letters.php

NB! The rules about where to put spaces in French number expressions are a bit ambiguous. Therefore, we will use a rule that there should be a hyphen ("-") between every numeric particle, e.g. "vingt-et-un", "cinq-cent-soixante-neuf "

You should submit the program, which is a modification of the provided template. It should run as is, without any additional dependencies.

The template uses the `pytest` unit test framework to include a set of tests that your completed implementation should successfully run. Use `python -m pytest numbers2words.py` to execute the tests. The template should produce the following output:

```
 [...]
FAILED numbers2words.py::test_numbers[0.046-zero-virgule-zero-quatre-six] - pywrapfst.FstOpError: Operation failed
================================================ 14 failed, 1 passed in 0.45s ===================================
```

Scoring: an undisclosed test set (similar to the provided unit test) will be used for scoring. Depending on the number of successful and failed tests, you will get 0 to 10 points. The scale is linear: `score = 10.0 * num_successful_tests / num_tests` The implementation should use Pynini (i.e., not some pure Python implementation).

NB! You should submit a Python program (the modified numbers2words.py file), not a Jupyter notebook, or a PDF file, or a HTML file.

NB! Your submission should also use Pynini and the `python -m pytest numbers2words.py` should execute without any problems using Python 3.7. If it fails to run for some reason (it always happens with some submission), I will reject the submission and you will get 1 day to fix it, and you will lose 10% of the points.

Hint: You should be able to implement it without using `cdrewrite`. Extensive use of `cdrewrite` can make the implementation very complex.

Developing on Colab

You can easily work on this assignment on Google Colab. The starting template is available here: https://colab.research.google.com/drive/14RautKJ8hcu8rqOy5wKgr2n5EJZH3J1m?usp=sharing

Note that now the whole program is in a single cell, and using the IPython magic expression `%%file numbers2words.py`, the cell is saved into file numbers2words.py each time you run the cell. In the next cell, you can execute the pytest tests by running `!python -m pytest numbers2words.py`.

You can easily download the resulting numbers2words.py file from Colab. There is a hidden file browser on the left part of the Colab window where you can access the files.