Explanation of steps and FST commands

1.a

Unvowel.txt is a transducer that removes the vowels aeiou (not y) from any string of the 26 lowercase letters and space. The FST is designed by substituting the character aeiuo with epsilon. For the consonants like b, c, d etc. we don't perform any substitution.

We compile the FST using the command:
fstcompile --isymbols=ascii.syms --osymbols=ascii.syms --keep_isymbols --keep_osymbols < unvowel.txt > unvowel.fst

To view the FST we can use: *fstprint unvowel.fst*

Next, we compose the FST with the given declaration.fst to remove all the vowels using the following command.
fstcompose declaration.fst unvowel.fst | fstproject --project_output | fstrmepsilon | fstprint

The output is as follows

```
0	1	w	w
1	2	h	h
2	3	n	n
3	4	<space>	<space>
4	5	n	n
5	6	<space>	<space>
6	7	t	t
7	8	h	h
8	9	<space>	<space>
...
```

The FST diagram for unvowel FST can be found in **unvowel.pdf**

1.b

To revowel the unvoweled FST, we invert the unvowel.fst with the following command to produce vowel.fst
fstinvert unvowel.fst > vowel.fst

Since there are an infinite number of ways to insert vowels, we compose the vowel.fst with the files provided c1.fst, c2.fst and so on to produce revowel1.fst, revowel2.fst and so on using the below commands

fstcompose vowel.fst c1.fst > revowel1.fst
fstcompose vowel.fst c2.fst > revowel2.fst
fstcompose vowel.fst c3.fst > revowel3.fst
fstcompose vowel.fst c5.fst > revowel5.fst
fstcompose vowel.fst c7.fst > revowel7.fst

The final pipeline for unvoweling and revoweling the transducers are as follows.

The declaration.fst is 1st unvoweled just as above and then revoweled by composing with the revowel FST files. At each revowel only a certain number of words are corrected. The counts for the same have been mentioned below.

fstcompose declaration.fst unvowel.fst | fstproject --project_output | fstrmepsilon | fstcompose - revowel1.fst | fstshortestpath | fstproject --project_output | fstrmepsilon | fsttopsort | fstprint

Using revowel1.fst we get 0/24 words were correct in the file.

fstcompose declaration.fst unvowel.fst | fstproject --project_output | fstrmepsilon | fstcompose - revowel2.fst | fstshortestpath | fstproject --project_output | fstrmepsilon | fsttopsort | fstprint

Using revowel2.fst we get 4/24 words were correct in the file.

fstcompose declaration.fst unvowel.fst | fstproject --project_output | fstrmepsilon | fstcompose - revowel3.fst | fstshortestpath | fstproject --project_output | fstrmepsilon | fsttopsort | fstprint

Using revowel3.fst we get 10/24 words were correct in the file.

fstcompose declaration.fst unvowel.fst | fstproject --project_output | fstrmepsilon | fstcompose - revowel5.fst | fstshortestpath | fstproject --project_output | fstrmepsilon | fsttopsort | fstprint

Using revowel5.fst we get 19/24 words were correct in the file.

fstcompose declaration.fst unvowel.fst | fstproject --project_output | fstrmepsilon | fstcompose - revowel7.fst | fstshortestpath | fstproject --project_output | fstrmepsilon | fsttopsort | fstprint

Using revowel7.fst we get 21/24 words were correct in the file.

What's wrong with n=1?

None of the words were corrected in the file while using revowel1.fst because, c1.fst which was used to generate revowel1.fst from vowel.fst does not contain any revoweling rules.

1.c

Next, we design a transducer, called **squash.txt**, to leave the first and last characters of each word alone, and remove only internal vowels

The transducer is designed with 3 states. On state 0, we have all the letters being substituted by itself. On transition from state 0 to 1 we have the vowels aeiou substituted by epsilon and all other characters remain the same. On state 2, we again have all the letters being substituted by itself. The final transition is from state 2 to 0 which is <space> transition for the next word.

We compile the FST using the command:
fstcompile --isymbols=ascii.syms --osymbols=ascii.syms --keep_isymbols --keep_osymbols < squash.txt > squash.fst

To view the FST we can use: fstprint squash.fst

Next, we compose the FST with the given declaration.fst to remove all the internal vowels using the following command.

fstcompose declaration.fst squash.fst | fstproject --project_output | fstrmepsilon | fstprint

The output is as follows

0	1	w	w
1	2	h	h
2	3	n	n
3	4	<space>	<space>
4	5	i	i
5	6	n	n
6	7	<space>	<space>
7	8	t	t
8	9	h	h
9	10	e	e

The FST diagram for squash FST can be found in **squash.pdf**

1.d

We perform similar unsquashing transducers for the squash transformation at all n-gram levels as in 1.b

The unsquash FST is created by inverting the squash.fst with the following command:
fstinvert squash.fst > unsquash.fst

Next, we compose the unsquash.fst with the files provided n-gram files, c1.fst, c2.fst and so on to produce resquash1.fst, resquash2.fst and so on using the below commands

fstcompose unsquash.fst c1.fst > resquash1.fst
fstcompose unsquash.fst c2.fst > resquash2.fst
fstcompose unsquash.fst c3.fst > resquash3.fst
fstcompose unsquash.fst c5.fst > resquash5.fst
fstcompose unsquash.fst c7.fst > resquash7.fst

The final pipeline for squashing and resquashing the transducers are as follows.

The declaration.fst is 1st squashed just as above and then resquashed by composing with the resqusah FST files. At each resquash only a certain number of words are corrected. The counts for the same have been mentioned below.

fstcompose declaration.fst squash.fst | fstproject --project_output | fstrmepsilon | fstcompose - resquash1.fst | fstshortestpath | fstproject --project_output | fstrmepsilon | fsttopsort | fstprint

Using resquash1.fst we get 8/24 words were correct in the file.

fstcompose declaration.fst squash.fst | fstproject --project_output | fstrmepsilon | fstcompose - resquash2.fst | fstshortestpath | fstproject --project_output | fstrmepsilon | fsttopsort | fstprint

Using resquash2.fst we get 10/24 words were correct in the file.

fstcompose declaration.fst squash.fst | fstproject --project_output | fstrmepsilon | fstcompose - resquash3.fst | fstshortestpath | fstproject --project_output | fstrmepsilon | fsttopsort | fstprint

Using resquash3.fst we get 17/24 words were correct in the file.

fstcompose declaration.fst squash.fst | fstproject --project_output | fstrmepsilon | fstcompose - resquash5.fst | fstshortestpath | fstproject --project_output | fstrmepsilon | fsttopsort | fstprint

Using resquash4.fst we get 23/24 words were correct in the file.

fstcompose declaration.fst squash.fst | fstproject --project_output | fstrmepsilon | fstcompose - resquash7.fst | fstshortestpath | fstproject --project_output | fstrmepsilon | fsttopsort | fstprint

Using resquash5.fst we get 23/24 words were correct in the file.

2.a

We write an FST that performs a single character substitution. We have 3 states in this FST. On state 0 and state 1, we have all the characters being substituted by itself. On the transition from state 0 to 1 we have each character being substituted by every other character in English. Hence, we will have 26X25 = 650 transition from state 0 to 1. This is written as **sub.fst**

We also write an FST for the word Cold as **cold.fst** and the diagram can be found in **cold.pdf**

We compile the sub FST using the command:
fstcompile --isymbols=ascii.syms --osymbols=ascii.syms --keep_isymbols --keep_osymbols < sub.txt > sub.fst

Next, we create an FST which contains all the possible 4 letter words which can be made using alice4up.fst. To do this we compose sub.fst with alice4up.fst to get **general_sub.fst**.

fstcompose sub.fst alice4up.fst > general_sub.fst

Now that we have general_sub.fst we can compose it with cold.fst and get all the words which can be generated by one step substitution on the word 'cold'.

fstcompose cold.fst general_sub.fst > 1_step.fst

The FST diagram: *fstdraw --portrait 1_step.fst | dot -Tpdf > 1_step.pdf*

The words generated are: told, hold

Next, to get two step substitution on the word 'cold' we compose the output of one step substitution with the general sub FST (general_sub contains all the possible 4 letter words combinations from alice4up.fst).

fstcompose 1_step.fst general_sub.fst > 2_step.fst

The FST diagram*: fstdraw --portrait 2_step.fst | dot -Tpdf > 2_step.pdf*

The words generated are: hole, held, cold, told, hold

Next, before we generate the three step substitution we need to perform sorting to smoothly compose the files. To perform this sort, we use the following commands. (fstarcsort operation sorts the arcs in an FST per state)

fstarcsort 2_step.fst 2_step.fst
fstarcsort general_sub.fst general_sub.fst

Finally, we can generate the three step substitution on the word 'cold' by composing the output of two step substitution with the general_sub FST.

fstcompose 2_step.fst general_sub.fst > 3_step.fst

The FST diagram: *fstdraw --portrait 3_step.fst | dot -Tpdf > 3_step.pdf*

The words generated are: told, head, help, hope, home, hold, held, hole, cold

You will see repetition in the two step and three step substitutions as some of the words can be derived by multiple substitutions. For example, in the two step substitutions, the word 'cold' can be derived from both 'hold' and 'told' after these words themselves are derived from word 'cold'. The same applies for three step substitutions

2.b

We write an FST that performs a single character substitution. We have 3 states in this FST. On state 0 and state 1, we have all the characters being substitued by itself. On the transition from state 0 to 1 we have each character being substituted by every other character in English. To allow single deletion we add the transition from 0 to 1 for every character being substituted by <epsilon>. To allow single insertion we add the transition from 0 to 1 for <epsilon> being substituted by every character. This is stored in **sub_ins_del.fst** file.

We also write an FST for the word 'cold' as **cold.fst** and the diagram can be found in **cold.pdf**

We compile the sub FST using the command:
fstcompile --isymbols=ascii.syms --osymbols=ascii.syms --keep_isymbols --keep_osymbols < sub_ins_del.txt > sub_ins_del.fst

Next, we create an FST which contains all the possible words which can be made using alice4up.fst. To do this we compose sub_ins_del.fst with alice4up.fst to get **general_ins_del.fst**.

fstcompose sub_ins_del.fst alice4up.fst > general_ins_del.fst

Now that we have general_sub.fst we can compose it with cold.fst and get all the words which can be generated by one step substitution on the word 'cold' including insertion and deletion of a single character.

fstcompose cold.fst general_ins_del.fst | fstrmepsilon > 1_step_ins_del.fst

The FST diagram: *fstdraw --portrait 1_step_ins_del.fst | dot -Tpdf > 1_step_ins_del.pdf*

The words generated are: hold, told, could

Next, before we generate the two step substitution we need to perform sorting in order to smoothly compose the files. In order to perform this sort we use the following commands. (fstarcsort operation sorts the arcs in an FST per state)

fstarcsort 1_step_ins_del.fst 1_step_ins_del.fst
fstarcsort general_ins_del.fst general_ins_del.fst

Next, to get two step substitution on the word 'cold' we compose the output of one step substitution with the general_ins_del FST (general_ins_del contains all the possible words combinations from alice4up.fst).

fstcompose 1_step_ins_del.fst general_ins_del.fst | fstrmepsilon > 2_step_ins_del.fst

We again perform fstarcsort on 2_step_ins_del.fst to help in generating three step substitution
fstarcsort 2_step_ins_del.fst 2_step_ins_del.fst

The FST diagram: *fstdraw --portrait 2_step_ins_del.fst | dot -Tpdf > 2_step_ins_del.pdf*

The words generated are: hold, cold, hole, held, would

Finally, we can generate the three step substitution on the word 'cold' by composing the output of two step substitution with the general_sub FST.

fstcompose 2_step_ins_del.fst general_ins_del.fst | fstrmepsilon > 3_step_ins_del.fst

The FST diagram: *fstdraw --portrait 3_step_ins_del.fst | dot -Tpdf > 3_step_ins_del.pdf*

The words generated are: cold, told, hold, could, whole, help, head, hope, home, hole, held, would, world

You will see repetition in the two step and three step substitutions as some of the words can be derived by multiple substitutions. For example, in the two step substitutions, the word 'cold' can be derived from both 'hold' and 'could' after these words themselves are derived from word 'cold' ('could' is one step insertion of 'cold' and 'cold' is one step deletion of 'could') . The same applies for three step substitutions