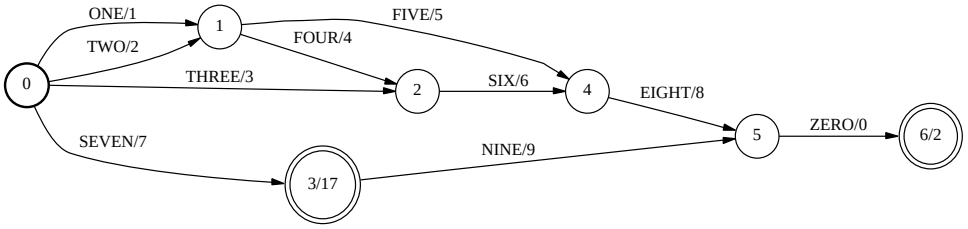
Pronunciation Modeling for LVCSR **Summer 2000**

Practical Session II: Phonological Rules

TeSTIA: The 8TH ELSNET European Summer School *Instructor: Eric Fosler-Lussier*

One thing that didn't get introduced in the last session is the concept of *weighted* finite state automata. Each transition arc can have associated with it a weight w_{arc}; also, each final state can have an associated weight w_{final} that is the cost of finishing in that state. Here's an example of a digits FSM from last time, with the weights equal to the digit. I've also included some arbitrary weights for finishing in states 3 and 6.

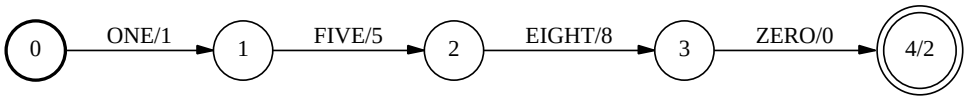


weighted.fsm

```
0 1 ONE   1
0 1 TWO   2
0 2 THREE 3
1 2 FOUR  4
1 3 FIVE  5
2 3 SIX   6
0 4 SEVEN 7
4 17
3 5 EIGHT 8
4 5 NINE  9
5 6 ZERO  0
6 2
```

Weights also enable you to have a concept of the *best* path, that is, the one with the lowest score. For example, the best path of this FSM can be found with:

```
prompt> fsmbestpath weighted.fsm > weighted-best.fsm
```



weighted-best.fsm

In this case, the FSM tools find the best path by summing along all of these paths, in the same way that the Viterbi search operates on graphs.[1] In fact, when the weights are the negative log probabilities of the transitions, using `fsmbestpath` corresponds to searching with the Viterbi criterion in ASR.

Weighted finite automata will be important in the implementation of probabilistic phonological rules and decision tree pronunciation models.

[1] Technically, you can use any semiring $(K, +, \cdot)$ as your mathematical basis in a weighted finite state transducer. When you have to combine two arcs, as in a union operation, the addition $(+)$ operation is used to combine scores, whereas for intersection, the multiplication $(\cdot)$ operator is used. For more information, see F. Pereira and M. Riley, "Speech Recognition by Composition of Weighted Finite Automata", cmp-lg archive 9603001, 7 March 1996. As distributed, the AT&T tools define *min* as the addition operation and $+$ as the multiplication operation. This, in effect, mimics the Viterbi criterion when the weights are interpreted as negative log probabilities.

1 Feasible pairs

Phonological rules are rewrite rules that map baseform phones to realized phone. In order to model pronunciation phenomena with FSMs, you first need to determine how each baseform phone can be realized. The reason for this will become clearer shortly. Let's consider a reduced set of English phones, with the following feasible pairs (the baseform phone is the left of the pair, the realization the right):

```
ae ae
ah ah
ah ax
ax ax
ax NULL
b b
ih ih
ix ax
ix ix
k k
m em
m m
r r
t dx
t t
s s
```

This means that we only expect "b" to be realized as "b", but "ix" might be pronounce "ix" or "ax".

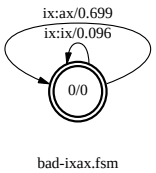
2 Phonological rules

Let's try to model a simple (but probably incorrect) rule first.

ix $\rightarrow$ ax (20%)
ix $\rightarrow$ ix (80%)

This means that we usually expect "ix" to be pronounced "ix", but sometimes it's pronounced "ax." We can model this with a transducer.

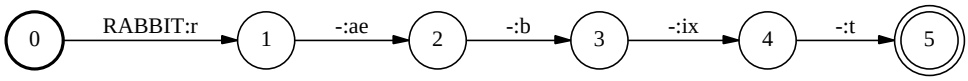
The key to constructing transducers for these rules is to remember to allow other feasible pairs to occur. For example, you might think that the following transducer would suffice:



bad-ixax.fsm

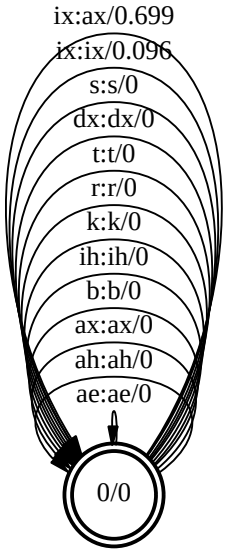
```
0 0 ix ix 0.0969
0 0 ix ax 0.6990
0
```

Note that the cost for transforming "ix" to "ix" is $0.969 \approx -\log_{10}(0.8)$, whereas "ax" costs $0.6990 \approx -\log_{10}(0.2)$. Higher scores are worse. There is a problem with this transducer, however: when you compose this with, e.g., the dictionary entry for *rabbit*:



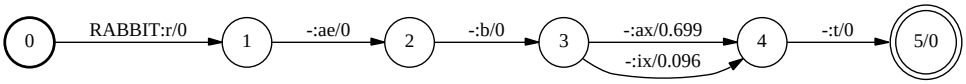
rabbit-canon.fsm

you'll get an empty FSM, because the transducer doesn't know what to do with inputs of "r," "ae," "b," and "t." To do this, you need to allow all other possible phones, i.e.:



good-ixax.fsm

```
prompt> fsmcompose rabbit-canon.fsm good-ixax.fsm > rabbit-alt.fsm
```



rabbit-alt.fsm

To do: how would you represent the following rule?

ah → ax (2%)
ah → ah (98%) $(-\log_{10}(0.02) \approx 1.699, -\log_{10}(0.98) \approx 0.0088)$

3 Combining rules

Usually we have more than one phonological rule that we're interested in. There's essentially two ways that you can combine rules: you can apply them in *sequential* order, or in a *parallel* fashion.

If our word *rabbit* is represented by the FSM W, and the *ix-ax* rule represented by FSM A, then the resulting composition is:

$$W \circ A$$

where $\circ$ represents composition. (The output of $W \circ A$ is pictured above as "rabbit-alt.fsm.") We can apply a second rule B, such as our *ah-ax* transducer, by just composing the output of $W \circ A$ with B:

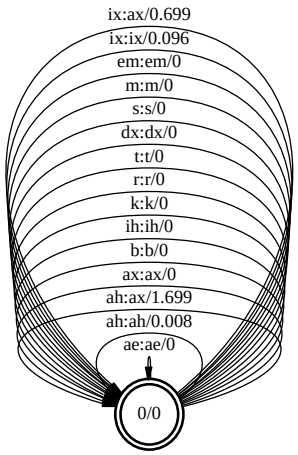
$$(W \circ A) \circ B$$

Of course, for the word *rabbit*, there is no difference in the output, since *rabbit* doesn't contain any *ah* sounds.

It turns out that the composition operation is *associative*, which means

$$(W \circ A) \circ B = W \circ (A \circ B)$$

This implies that we can compose all of our phonological rules into one big transducer and apply it to any word. The composition of A and B looks like this:



To do: Using the word *RUSSET*, with the pronunciation *r ah s ix t*, check to make sure that the associative property of composition is true.

It is important to remember that composition is *not* commutative, that is in general,

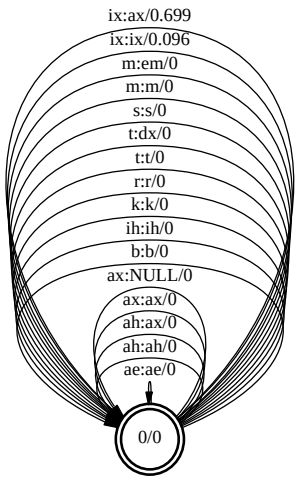
$$A \circ B \neq B \circ A$$

which you can see if you consider A = Transducer(ah $\rightarrow$ ax) and B = Transducer(ax $\rightarrow$ ix): composing $A \circ B$ *always* converts *ah* to *ix*, whereas $B \circ A$ converts *ah* to *ax*. This brings up the issue of rule ordering — choosing the order in which rules are to be applied can affect the output of the system. One way to get around this problem is to make all rules optional, and repeatedly apply the phonological rules until you stop getting new variants (this is what we did in Tajchman et al. 1995, using a non-FSM-based paradigm). Other researchers have suggested that the best method is to apply all rules in parallel, in effect:

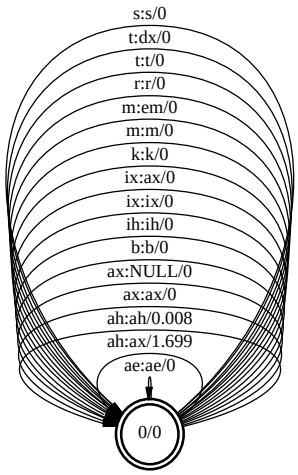
$$W \circ (A \cap B)$$

This is where the idea of *feasible pairs* comes in: remember that when you take the intersection of two FSMs, you keep only the paths that are common to *both* FSMs. This means that each FSM must *allow* all of the

feasible-pair alternations, i.e.:



full-ixax.fsm



full-ahax.fsm

In essence, by including all possible alternations with a score of zero, the transducer says "this might be a possible symbol pair, but I don't know anything about it". The responsibility for assigning weights to zero-weighted patterns is left up to other transducers. The obvious disadvantage to this method is that every transducer must be aware of the possible pronunciation alternatives provided by other rules.

Transducer intersection is only well defined for certain types of transducers — in particular, the two transducers must produce expressions of the same length. Intersection is accomplished by considering each pair of symbols as an atomic symbol.

To do: (if you have leftover time) The AT&T toolkit doesn't allow for intersection of transducers, but by manipulating the ASCII representations of transducers, one can convert transducers to automata and then use `fsmintersect` to intersect them. Write a perl program (or use some other scripting language) to do this automatically.

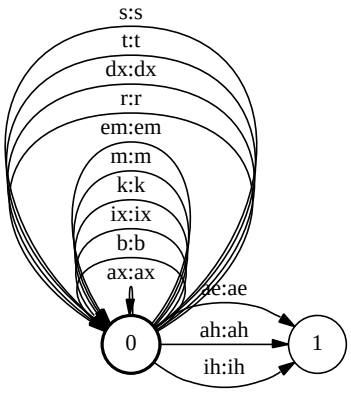
4 Adding context

The rules we've been considering to this point were very simple, context-free rules. In general, however, there is context to be considered. For instance, (part of) the *flapping* rule in English is as follows:

$$\text{t} \rightarrow \text{dx} / \begin{bmatrix} \text{+stress} \\ \text{+vowel} \end{bmatrix} \text{\_\_\_} \begin{bmatrix} \text{-stress} \\ \text{+vowel} \end{bmatrix}$$

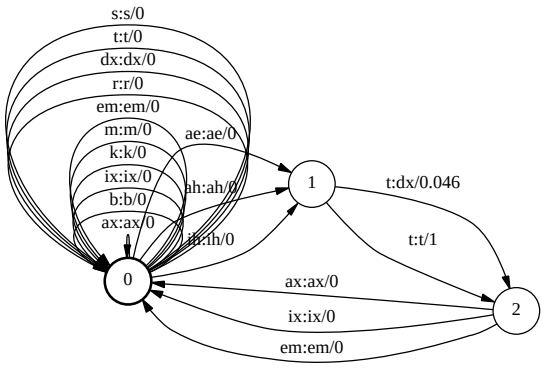
Let's assume that this rule applies 90% of the time. The trick to constructing the transducer for rules like this is to keep track of the purpose of each state. Let's start with the *start state*, i.e. state 0. If we're in state 0 and see a stressed vowel (*ae, ah,* or *ih*), then we've fulfilled the left-context precondition for the rule.

We'll use state 1 to signify that we've seen the stressed vowel, and add a transition from state 0 to 1:



flap1.fsm

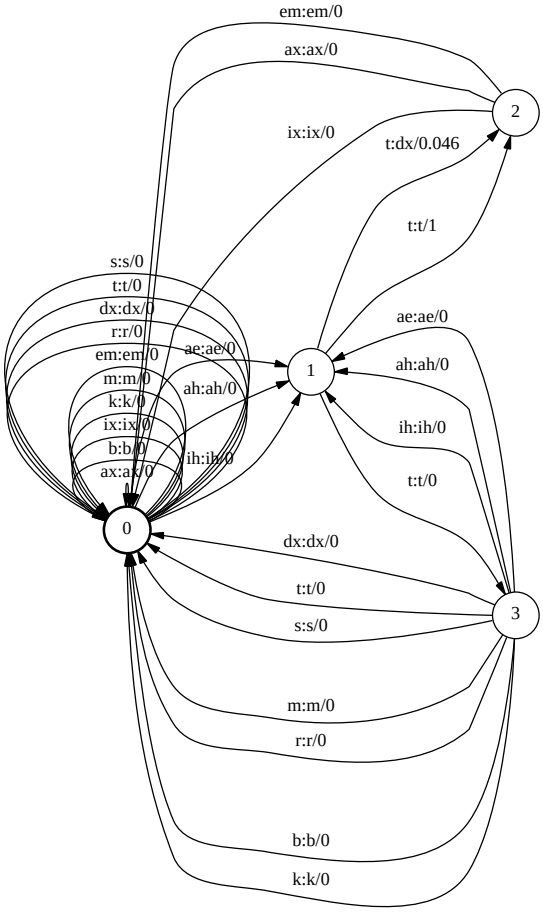
Note that for all other phones, we'll stay in state 0.[2] The question is, what happens next? Well, if we saw a *t* followed by an unstressed vowel (*ax, ix,* or *em*), then we'd want to convert the *t* to a *dx* with 90% probability. After seeing the unstressed vowel, we would be back where we started: not having seen the left context. Therefore, after the unstressed vowel, we transit back to state 0.



flap2.fsm

But what if we were in state 1 and didn't see an unstressed vowel after the *t*? Then the *t* should just be realized as *t*. So we add another state (3) from which we can only expect phones other than unstressed vowels. If we see a stressed vowel, we have the left-context precondition for flapping: we transit back to state 1. Otherwise, for non-vowels we just transit back to state 0.

[2] I've chosen to do this in the composition-based style rather than listing all feasible pairs for combination by intersection, but you can do this the other way as well.

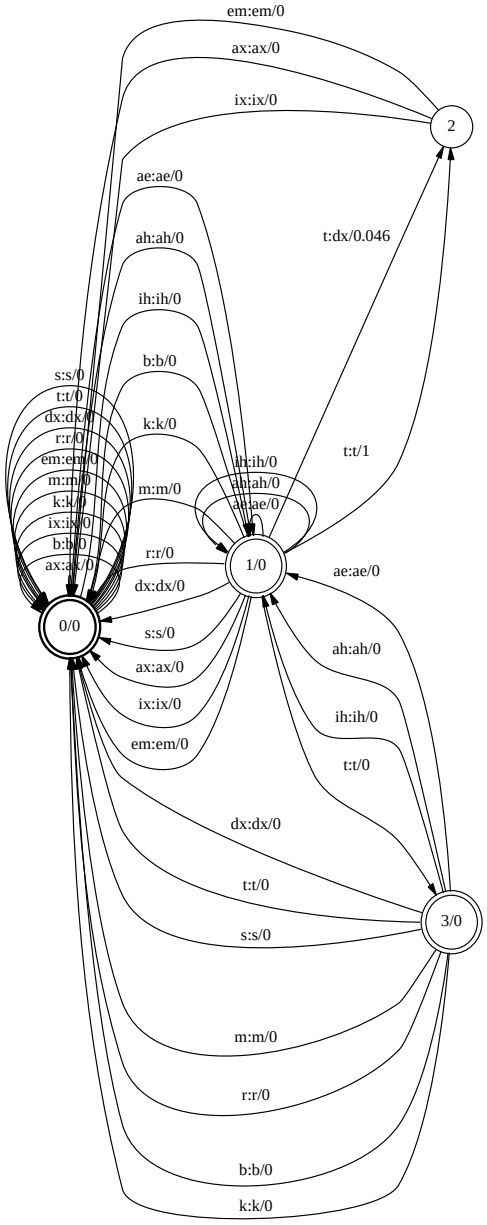


flap3.fsm

We're not quite done yet. What happens in state 1 if we something other than a *t*? If it's a stressed vowel, then that's a precondition for the flapping rule, so we stay in state 1; otherwise, we just transition back to state 0.

We also have to determine which states are final and which are not. The main criterion is that we can't allow a situation where a phonological rule is partially completed (i.e. its right context hasn't been seen) to be final. State 0 can be final because no context has been seen at all. State 1 has only seen the left context; no claims on the realization of *t* are made. State 2 *cannot* be final, because we need to see the right context if the transformation is to be allowed. State 3 can be final, since that's the state in which the rule didn't apply.

The final product is:



flap4.fsm

To do: Can you make a transducer corresponding to "ax m → em"? (Hint: model it as a deletion of ax (i.e. transformed to NULL) followed by a transformation of m.)

To do: What if the above rule was optional (50% probability)?

To do: Can you modify the flap transducer to allow for an optional *r* before the *t*?

$$\text{t} \rightarrow \text{dx} / \begin{bmatrix} +\text{stress} \\ +\text{vowel} \end{bmatrix} (\text{r})? \_\_\_\_ \begin{bmatrix} -\text{stress} \\ +\text{vowel} \end{bmatrix}$$