600.405 — Finite-State Methods in NLP
Assignment 3: HMMs and Formal Power Series

Prof. J. Eisner — Fall 2000
Handed out: Sat., Dec. 2, 2000
Due date: Try to do problem 1 by the Dec. 5 lecture,
so that you'll understand the lecture. Assignment is due by
Thu., Dec. 7, 3pm, to NEB 224 mailbox or `jason@cs.jhu.edu`.

1. This week's practical exercise uses the FSM toolkit. Step by step, you will build a Hidden Markov Model (HMM) and use it to assign parts of speech to words. As usual, you are welcome to work in pairs. If you still run into trouble, please email me as soon as possible.

   (a) To familiarize yourself with the problem, manually give the appropriate part-of-speech tag for each word in the following sentence:

   Both of the other candidates eyed Nader suspiciously.

   You should use the tag set that is described in `http://www.ldc.upenn.edu/doc/treebank2/cl93.html`, section 2.2. By way of example, you can view over a million words of English text (the **Brown corpus**) that have been manually annotated with these tags:[1] `http://www.cs.jhu.edu/~jason/405/hw3files/brown.txt`.

   (b) The **cost** of an event is the negative logarithm (base e) of the event's probability. Suppose two events have probabilities p and q. (i) What are their costs? (ii) What is the sum of their costs? (iii) The sum of their costs is also a cost; what "event" is it the cost of?

[1]It appears that four of the punctuation tags described there are not used in our version of the Brown corpus. Other punctuation tags were instead made to do double duty.

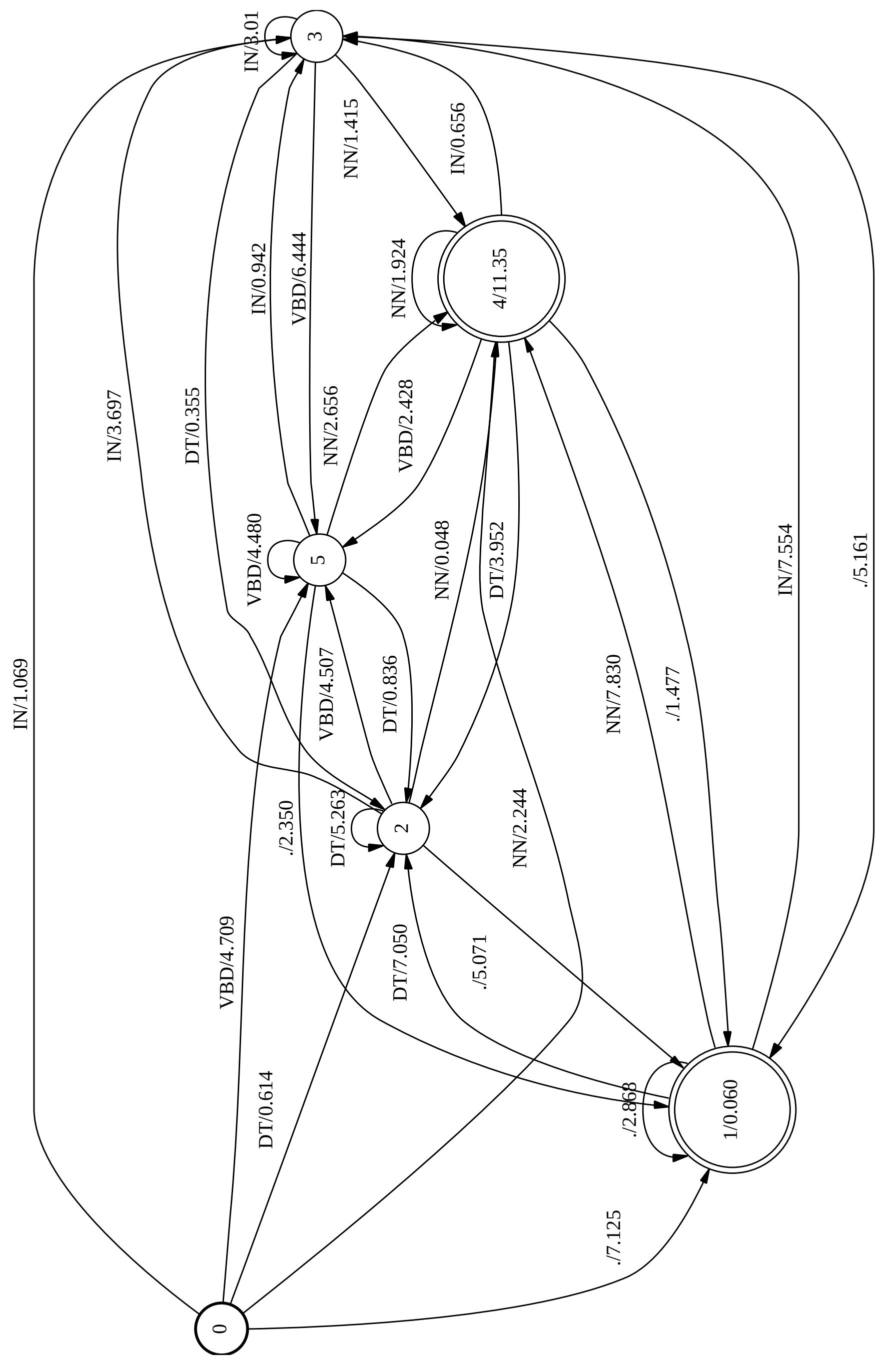

Figure 1: The acceptor `tagbigram-tiny.fsa`. (This is a version of `tagbigram.fsa` that has been simplified by removing some states and then renormalizing the probabilities; the real `tagbigram.fsa` is a large dense tangle that is too hard to read.)

(c) Figure 1 shows a finite-state machine over the semiring $(\mathbb{R}, \min, +)$. Compute the weight it assigns to the tag sequence "`DT NN NN VBD .`"? If this weight represents the cost of the tag sequence, then out of 10000 random English sentences, how many would you expect to have this part-of-speech sequence?

(d) Same question as in (1c) but with the final period removed from the tag sequence. (Warning: this is a bit of a trick question!)

(e) The machine of Figure 1 was constructed automatically from the Brown corpus. Notice that state 4 is the state of the machine if it has just read `NN`. The cost of reading a `VBD` next—i.e., the weight of the arc from 4 to 5—is 2.428. This is because 8.82% of the `NN` tags in the Brown corpus were immediately followed by `VBD`, and $(-\log 0.0882) = 2.428$. The machine was built by a short Perl script that prepared input to `fsmcompile`; the script had to read the Brown corpus and print lines like `4 5 VBD 2.428`.

This kind of machine is called a **Markov model**, which means that the next tag (or the option of stopping) is chosen randomly with a probability that depends only on the previous tag. The machine's state serves to remember that tag.

Actually, Figure 1 shows only a simplified version of the machine that fits on the page. Download the full machine: it is called `tagbigram.fsa`, and the corresponding label file (for `fsmprint` and `fsmdraw`) is `tags.lab`. All files in this problem can be found in the web directory `http://www.cs.jhu.edu/~jason/405/hw3files/` (or the ordinary directory `~jason/405/hw3files/` on the CS research network).

Answer the same question as in (1c) but with the real machine, using the FSM tools to the extent possible. What commands did you execute to get the answer? (Hint: you may find `fsmminimize` helpful at the last step, although the implementation seems to have some funny rounding error.)

Note: You could review `http://www.cs.jhu.edu/~jason/405/software.html`, including the `lexcompre` command.

(f) The simplified machine actually shown in Figure 1 can be found as `tagbigram-tiny.fsa`. Also, there is a machine called `deltag.fst` that recognizes $(\Sigma : \epsilon)$. (Examine it if you like.)

Construct the following machine out of those parts:

```
fsmclosure deltag.fst | fsmcompose tagbigram-tiny.fsa - |
    fsmproject -o | fsmrmepsilon
```

(i) What is its final-state weight? (ii) More important, what is the precise relation of this weight to Figure 1? (iii) Does this give you another way to answer problem 1e?

Note: `fsmproject` and `fsmrmepsilon` are documented on the `fsm` man page. You may wish to look at the intermediate stages in the pipeline above. Finally (big hint) if you find yourself studying Figure 1 closely, `fsmbestpath` may help you find what you're looking for.

(g) Although the FSM toolkit is designed to work over arbitrary semirings, the compiled version we currently have only works over $(\mathbb{R}, \min, +)$. (A more flexible version is supposed to come out within half a year.)

But suppose you replaced the arc weights of Figure 1 with probabilities (rather than negative log probabilities), and used the semiring $(\mathbb{R}, +, \times)$. What would the answer to problem 1f be then?

Note: For any state, the sum of the weights on its out-arcs, plus its stopping weight, would be 1.

(h) Generate 5 random paths through tagbigram.fst, using `fsmrandgen -n 5 tagbigram.fsa`. Each path is a randomly generated string of tags. Use `fsmrandgen -?` for usage documentation.

To generate each path, `fsmrandgen` takes a random walk on the automaton, at each state choosing its next move probabilistically. (It assumes the semiring $(\mathbb{R}, +, \times)$, so that arc weights and stopping weights are costs; it uses these costs to compute the probabilities of all the options at a given state. These probabilities should sum to 1 as mentioned above; if for some reason they don't, `fsmrandgen` renormalizes them before choosing, but this isn't usually what you want.[2])

Save this automaton in a file. Look at its topology using `fsmdraw`. Then pass it through `lexfsmstrings` (documented on the `lextools` man page; you will want to use the `-ltags.lab` argument). In general I recommend piping the output of `lexfsmstrings` through an extra transducer or two to make

[2]It's not what you want because it depends on the structure of the automaton, not on the formal power series that it represents. Two equivalent automata (e.g., before and after minimization) would be affected differently by the renormalization.

In most such cases, what you really want is to renormalize the *path* weights so that the relative path probabilities are preserved. For example, if you're interested in the *conditional* probability distribution of tag strings that match a particular regular expression such as `VB DT` Σ^*, you intersect `tagbigram.fsa` with `VB DT` Σ^* and renormalize. *Extra credit:* How would you use FSM operations to renormalize a probabilistic automaton in this way, over $(\mathbb{R}, +, \times)$ as in problem 1g?

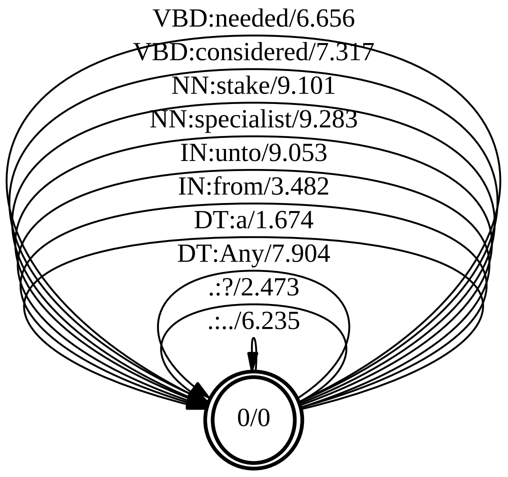

Figure 2: The tag-to-word transducer `tag2word.fst` has 53850 words, some of which appear with multiple parts of speech, for a total of 63762 arcs. (This diagram shows only a subset of the arcs; it is available as `tag2word-tiny.fst`).

it more readable, e.g., pipe it through

```
perl -pe 's/\[.*?\]|<.*?>|./$& /g; tr/][//d;'
```

Include the results in your writeup. Must longer strings always have higher costs?

(i) Figure 2 shows a transducer `tag2word.fst` using the same weight semiring $(\mathbb{R}, \min, +)$. This simply maps tags to words. Again the costs were determined automatically from the Brown corpus. For example, 0.011% of all singular common nouns (`NN`) in the corpus were the word `stake`, so we would like to replace `NN` with `stake` with probability 0.00011, i.e., cost 9.101 as shown.

In problem 1h you generated an automaton that has 5 strings of tags. Replace those tags with words nondeterministically as follows: Apply `tag2word.fst` to that automaton (using `fsmcompose`) and select some paths using `fsmrandgen`.

These new paths still have both input (tag) and output (word) symbols. For your answer, print out just the *words* on those paths. They should be plausible give the tags you already chose, and they should look vaguely like English.

Be careful if your output looks fishy: you will have to use both `fsmproject`

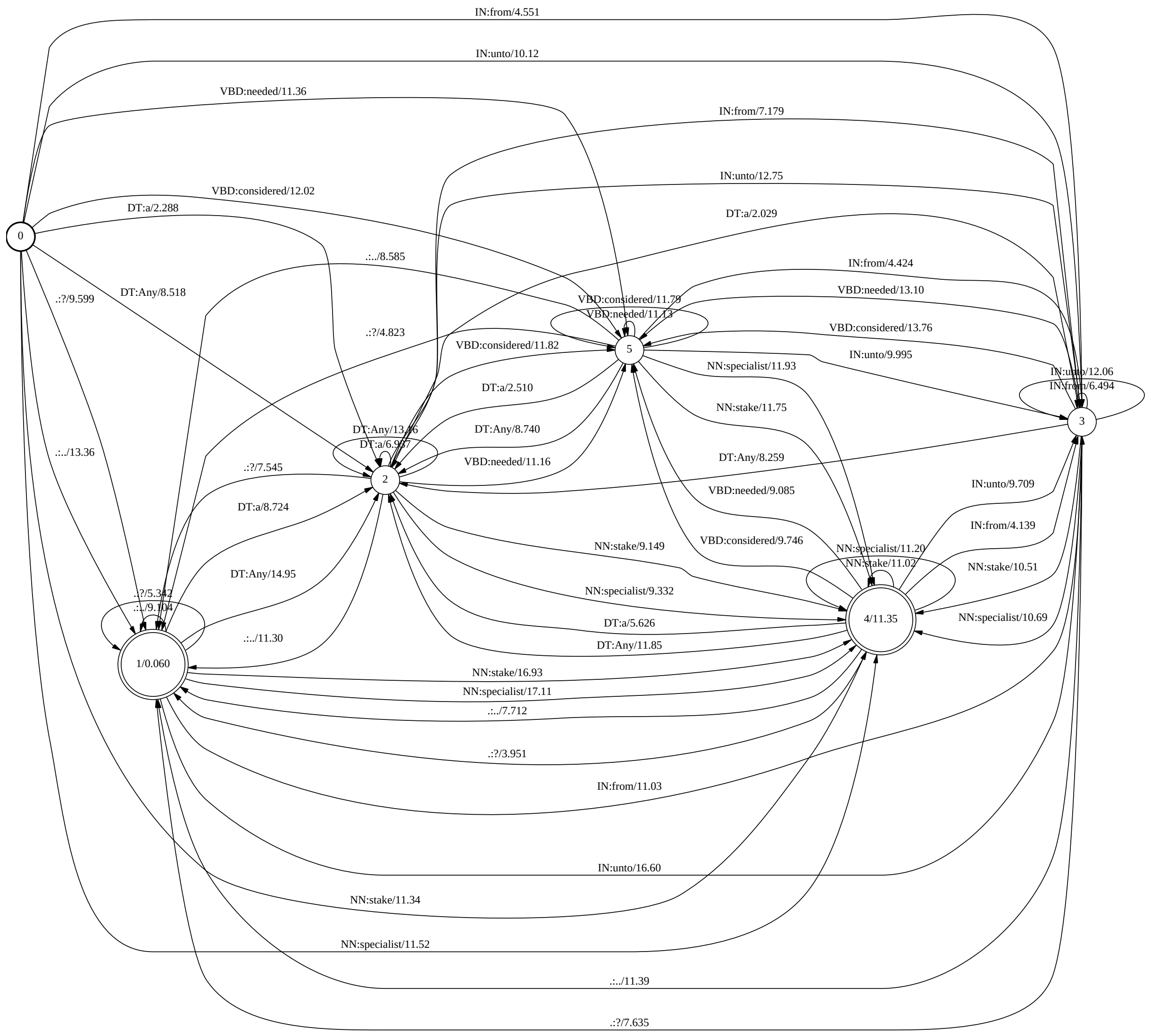

Figure 3: The composition of Figure 1 with Figure 2.

and the label file `words.lab` for the last step.

Note: Footnote 2 applies here, so using `fsmrandgen` is actually unwise. Question (1m) below explores a better approach.

(j) What we're *really* interested in is the composition $M_2 =$ `tagbigram.fsa` $\circ$ `tag2word.fst`. This is a subsequential (i.e., deterministic) transducer from tags to words because both the machines being composed are subsequential. A simplified version is shown in Figure 3.

Suppose there is a (unique) path in M_2 that reads a string of tags $\vec{t}$ from the upper language and a string of words $\vec{w}$ from the lower language. Suppose further that this path has weight C. You can think of this in any of the following ways:

- M_2 transduces $\vec{t}$ to $\vec{w}$ with weight C.
- The inverse transducer M_2^{inv} (see problem 7a) transduces $\vec{w}$ to $\vec{t}$ with weight C.
- M_2 accepts the pair $(\vec{t}, \vec{w})$ with weight C.

What event is C the cost of, and why? In 1000 tagged sentences, how often would you expect to see the pair $(\vec{t}, \vec{w})$?

(k) Generate 5 random paths and the 5 lowest-cost paths from the machine M_2 and print out their word sequences. Report the commands you used and the output strings. What does the lowest-cost path represent? Why isn't it the same length as the path in problem 1f?

Note: M_2 only takes a matter of seconds to create on a fast machine, but it's about 42 megabytes, so you probably don't want to save it to disk; instead pipe the output of `fsmcompose` directly into the commands that find random paths and best paths. If you do create any large files in the course of this assignment (e.g., in the `/tmp` directory), clean them up!

(l) Starting with the next question, we will replace `tagbigram.fsa` with `tagtrigram.fsa`. This is also derived automatically from the Brown corpus, but it is a **second-order Markov model** that captures more of the facts of English. The state encodes the previous *two* tags (not just the previous one), so the cost of reading `VBD` next is sensitive to more of the previous context.

How many states and arcs does each machine have? (Use `fsminfo -n` to find out.) The number of arcs in one machine seems to be related to the number of states in the other—is this a coincidence? If not, explain.

Note: These machines have not been minimized.

(m) $\star$ One would like to rerun problem 1k using the trigram model instead of the bigram model. This should generate random strings that look even more like English. Unfortunately, the composition $M_3 =$ `tagbigram.fsa` $\circ$ `tag2word.fst` is too large a machine to compute easily.[3]

Nonetheless, at least one of the following commands chooses a random path in the composition, with exactly the same probability as `fsmrandgen` M_3 would have:

i.
```
fsmrandgen tagtrigram.fsa | fsmcompose - tag2word.fst
    | fsmproject -o
```
ii.
```
fsmrandgen tagtrigram.fsa | fsmcompose - tag2word.fst
    | fsmproject -o | fsmrandgen
```
iii.
```
fsmrandgen tagtrigram.fsa | fsmcompose - tag2word.fst
    | fsmproject -o | fsmdeterminize | fsmrandgen
```
iv.
```
fsmrandgen tagtrigram.fsa | fsmarith -m 0 | fsmcompose
    - tag2word.fst | fsmproject -o | fsmrandgen
```

Try all of these out and include sample output. Which command does what we want? How exactly do the others differ? (Use `man fsm` and `fsmarith -?` for help.)

(n) $\star$ One might also like to generate the best path in the trigram model M_3. Is there also a trick here that will find the best path without actually constructing M_3?

A common approximation technique is to consider just the "very good" paths in the simpler model M_2, and choose the best of these according to M_3. What commands would do this efficiently?

(o) Now let's use the machine for its real purpose—part-of-speech tagging. We want to find the best path that is consistent with *a given string of words*. (In this context, M_3 is called a **hidden Markov model** because the arcs (tags) chosen by the Markov model, `tagtrigram.fsa`, are not observed; only their correlates, the words, are observed.)

Try composing M_3 with each of the following difficult sentences. You should do this efficiently, without actually constructing M_3. (Hint: composition is associative; alternatively, `fsmcompose` can take more than 2 arguments.)

[3]If you use the C library calls rather than the command-line utilities, large machines are not always a problem because their states and arcs are computed lazily on demand. In the command-line utilities, the whole machine has to be created and piped to the next command.

```
lexcompre -lwords.lab -s '[time][flies][like][an][arrow]\.'
lexcompre -lwords.lab -s '[can][you][can][this][can][of][soup]\?'
```

In each case, look at the resulting machine, then find and report the tag sequence of its best path. Was the tag sequence correct? If not, was it plausible?

(p) Another interesting application is to constrain the machine not by a string of words, but by a **lattice** of words:

```
... -s '[can][you][can](([this]([can]|[scan]))|([the][scan]))
                     (([of]|[a]|[us])[soup])|([us][up])\?'
... -s '[can][you][can](([this]([can]|[scan]))|([the][scan]))
                     (([of]|[a]|[us])[soup])|([us][up]<5>)\?'
```

A lattice represents an uncertain input: in this case, a set of strings that sound alike. The regexps above are hard to read, so use `fsmdraw` to visualize one of the lattices above after it is produced by `lexcompre`.

In general, lattices have many weights. The second lattice above puts a large cost on `up` to indicate that the input really didn't sound very much like "up."

After drawing each lattice, compose M_3 with the lattice, find the best path, and report the *word sequence* on that path.

This contrasts with the previous problem, where you reported the tag sequence. You are now using M_3 to disambiguate an ambiguous input while tagging it. A word sequence in the lattice is considered likely if it is the result of likely substitutions of words into a tag sequence that is itself likely.

Notice that when we added a weight to "up" in the second lattice, "up" was not the only word that changed in the best path. Why not?

(q) $\star$ Now let's get realistic and tag a lot of sentences at once. We'll do the first 200 sentences from the Brown corpus, containing 5190 words. (Since our automata were built with knowledge of those words and their correct tags, this is *not* a fair test of the tagging method.)

The word sequence is in `brown200-words.fsa`. This is a straight-line automaton that simply puts the words one after another. Each new sentence is preceded by the special word `*bos*` ("beginning of sentence"). The correct tag for `*bos*` is always `*BOS*`.

Find the best tag sequence for these words according to M_3, and put it in a file called `brown200-tritags.fsa`. The correct tag sequence is in the file `brown200-tags.fsa`. Report the weights of *both* sequences under M_3.

Similarly, find the best tag sequence for these words according to M_2, and put it in a file called `brown200-bitags.fsa`. The correct tag sequence is in the file `brown200-tags.fsa`. Report the weights of *both* sequences under M_2.

M_3 and M_2 don't have any arcs labeled with `*BOS*` or `*bos*`. You'll have to modify them so they will transduce a string of *several* tag sequences separated by `*BOS*` tags to a string of several word sequences separated by `*bos*` words.

As always, you should *not* try to do this by modifying the states and arcs of existing machines. Just use the FSM operations (perhaps together with some "glue" in the form of small regexps or machines) to build up new machines from old. Give the commands you used.

(r) $\star$ Finally, what's the error rate—how many tags did M_3 get wrong (likewise M_2)? You can also determine this with a transducer.

Compute the answer using nothing except operations on `deltag.fst` (see problem (1f)) and the two tag sequences to be compared.

Hint: First build a transducer `scorer.fst` such that `brown200-tritags.fsa` $\circ$ `scorer.fst` $\circ$ `brown200-tags.fsa` is most of the way to your answer. You'll need `fsmarith` (see problem (1m)).

(s) *Extra credit:* In problem 1o, you created a lattice of all tag sequences that were consistent with an input sentence. Suppose this lattice were weighted with probabilities rather than logarithmic costs (as in problem 1g). Also suppose it were extended over an entire corpus as in the previous problem.

The forward-backward (Baum-Welch) reestimation algorithm asks for the expected number of times a given trigram appears in the lattice. Can you compute this using a transducer? If so, sketch the best way; if not, why not?

Congratulations on finishing this very long problem!

2. Recall the following notation from class:

- $K\langle\langle\Sigma\rangle\rangle \overset{\text{def}}{=}$ the set of formal power series over Σ^* with coefficients in the semiring K. (This notation assumes that $\oplus$ and $\otimes$ will be clear from context. If not, mention them, or write them as subscripts: $K_{\oplus,\otimes}\langle\langle\Sigma\rangle\rangle$.)
- $(S, w) \overset{\text{def}}{=}$ the coefficient of w in $S \in K\langle\langle\Sigma\rangle\rangle$.
- $K\langle\Sigma\rangle \overset{\text{def}}{=} \{S \in K\langle\langle\Sigma\rangle\rangle : \{w : (S, w) \neq \underline{0}\}$ is finite$\}$ = the set of non-commutative polynomials over Σ^* with coefficients in K.

- $(S \oplus T, w) \overset{\text{def}}{=} (S, w) \oplus (T, w)$ ("union")
- $(S \otimes T, w) \overset{\text{def}}{=} \bigoplus_{uv=w}((S, u) \otimes (T, v))$ ("concatenation"; in fact $ST \overset{\text{def}}{=} S \otimes T$)
- $S^* \overset{\text{def}}{=} \bigoplus_{i \geq 0} S^i = \underline{1} \oplus S \oplus (S \otimes S) \oplus \cdots$ ("Kleene star")
- $(S \odot T, w) \overset{\text{def}}{=} (S, w) \otimes (T, w)$ ("Hadamard product")
- $K_{\text{Rat}}\langle\langle\Sigma\rangle\rangle \overset{\text{def}}{=}$ the closure of $K\langle\Sigma\rangle$ under union, concatenation, and Kleene star = the set of *rational* power series.

For each of the following regular expressions in the notation of the FSA Utilities (see http://www.cs.jhu.edu/~jason/405/software.html),

- write an equivalent expression over formal power series (preserving the structure of the expression but using the new notation);
- give the first few terms of the series $\in K\langle\langle\Sigma\rangle\rangle$ that your expression denotes;
- say what you assumed $K\langle\langle\Sigma\rangle\rangle$ was when you wrote out the first few terms.

I've given the answer for the first one.

(a) $\{a :: 0, b :: 3\}^*$
Answer: $(0a \oplus 3b)^* = 1 \oplus 0a \oplus 3b \oplus 0aa \oplus 3ab \oplus 3ba \oplus 6bb \oplus \cdots \in \mathbb{R}_{\min,+}\langle\langle\{a, b\}\rangle\rangle$. [Note that in this case the semiring's $\underline{0}$ is ∞, not 0.]
Alternative answer over a different semiring: $(0a \oplus 3b)^* = (3b)^* = 1 \oplus 3b \oplus 9bb \oplus 27bbb \oplus \cdots \in \mathbb{R}_{+,\times}\langle\langle\{a, b\}\rangle\rangle$. [Here $\underline{0} = 0$.]

(b) $[\{a :: 0, [b :: 2, c :: 2]\}^*, d, e]$

(c) $\{a :: 0, [b, c] :: 4\}^*$

(d) $[\{a :: 1, [a, a] :: 4, [a, a, a] :: 9\}, \{a :: 1, [a, a] :: 4, [a, a, a] :: 9\}]$

(e) $[\{a :: 1, [a, a] :: 4, [a, a, a] :: 9\}, \{a :: 1, [a, a] :: 4, [a, a, a] :: 9\}]$ & $(a : 2)^*$

(f) $\{a : x, b : y\}^*$

(g) $\{a, ab\}^*$

(h) $[\{a : u, [a, a] : [v, v], [a, a, a] : [u, u, u]\}, \{a : u, [a, a] : [v, v], [a, a, a] : [u, u, u]\}]$

(i) $\{a : [x, y], a : \epsilon, \epsilon : z\}^*$

(j) $\{a : [x, y], a : \epsilon, \epsilon : z\}^*$

(k) $\{a : x : 0, b : y : 3\}^*$

(l) $\{a : [x, y] : 3, a : \epsilon : 4, \epsilon : z : 5\}^*$

3. Draw a minimal weighted automaton recognizing the power series of (2d).

   (You don't know yet how to minimize weighted automata in general, but you should be able to figure this one out from the power series you wrote, especially since it has finitely many terms (i.e., is a polynomial).)

4. (a) In $\mathbb{R}_{+,\times}\langle\langle\{a, b\}\rangle\rangle$, simplify the expression $3ab \oplus 4ab \oplus (5ba \otimes 7)$.

   (b) In $\mathbb{R}_{\min,+}\langle\langle\{a, b\}\rangle\rangle$, simplify the expression $3ab \oplus 4ab \oplus (5ba \otimes 7)$.

   (c) In general, we write ST as an abbreviation for $S \otimes T$. There is a danger of confusion here, since $3ab$ is ambiguous: it could mean either (i) the function that maps $ab \mapsto 3$ and maps everything else $\mapsto \underline{0}$, or (ii) the product $3 \otimes a \otimes b$. Show that there is actually no danger because the power series (i) and (ii) are equal.

   (Hint: Remember that 3 denotes 3ϵ, the power series that maps $\epsilon \mapsto 3$ and maps everything else $\mapsto \underline{0}$, and $a \in \Sigma$ denotes $\underline{1}a$ similarly.)

5. The series S is called **proper** iff $(S, \epsilon) = \underline{0}$. It was claimed in class that the series defining S^* will converge if S is proper. Let's explore that.

   (a) Suppose S is proper and you are given a string w. For all $m \geq 0$, let k_m denote the partial sum $\bigoplus_{i=0}^{m}(S^i, w)$. Show that $k_0, k_1, k_2, \ldots$ converges after finitely many terms, and write a closed-form expression for the limit value k^∞ in terms of S and w. (We can then define $(S^*, w) \overset{\text{def}}{=} k^\infty$.)

   (b) Give an example of a series S (over some semiring you choose) such that S is *not* proper but $k_0, k_1, k_2, \ldots$ converges after finitely many terms anyway.

6. Recall that a formal power series $S \in K\langle\langle X\rangle\rangle$ is **recognizable** iff there exist $n \geq 0$, $L, R \in K^{n\times 1}$, and $M : X \to K^{n\times n}$ such that for all strings $w = x_1 \ldots x_j$, we have $(S, w) = L^t M(x_1)M(x_2)\cdots M(x_j)R$. This gives us an effective way of calculating (S, w) even when the automaton is nondeterministic.

   Prove that L can be eliminated from the definition in favor of a single unweighted start state. That is, even if the definition were changed to require $L^t = [\underline{1}\ \underline{0}\ \underline{0}\ \cdots\ \underline{0}]$, the same series would be considered recognizable as before. To put this another way, one can transform any finite-state machine, without changing the function it computes, so as to eliminate multiple initial states and initial-state weights.

Similarly, show that R can be eliminated from the definition, but that L and R cannot *both* be eliminated without changing the meaning of recognizability.

7. In this problem, we consider special operations on (weighted) transducers.

   Let Σ, Δ be alphabets and let K be a semiring. Suppose $S \in (K_{\text{rat}}\langle\langle\Delta\rangle\rangle)_{\text{rat}}\langle\langle\Sigma\rangle\rangle$. This says that S is a rational power series mapping each string $w \in \Sigma^*$ to a rational power series (S, w) over Δ^*, i.e., to a weighted language. S may be regarded as a weighted transduction from $\Sigma^* \to \Delta^*$; we write $((S, w), v)$ to get the weight of mapping $w \in \Sigma^*$ to $v \in \Delta^*$.

   (a) $\star$ We define the inverse transduction S^{inv} by $((S^{\text{inv}}, v), w) \overset{\text{def}}{=} ((S, w), v)$.[4] Can we conclude from the fact that S is a *rational* power series that S^{inv} is too? Why or why not?

   (b) $\star$ Let Γ be another alphabet, and suppose $T \in (K_{\text{rat}}\langle\langle\Gamma\rangle\rangle)\langle\langle\Delta\rangle\rangle$. We define the composition $S \circ T$ by $(((S \circ T), u), w) \overset{\text{def}}{=} \bigoplus_{v \in \Delta^*} ((S, u), v) \otimes ((S, v), w) = \bigoplus_v ((S, u) \odot (S^{\text{inv}}, w), v)$. Can we conclude from the fact that S and T are rational power series that $S \circ T$ is too? Why or why not?

   You may or may not find it helpful to use the fact that the rational power series are exactly the recognizable power series.

[4] S^{inv} is usually written S^{-1}. However, I'm trying to avoid notational confusion: in class we used S^{-1} to denote the multiplicative inverse of a formal power series, whereas here we want the inverse under composition.

This ambiguity in exponents stems from the fact that both multiplication and composition are written with concatenation. You've already run into it in trigonometry: you might expect $\sin^2$ and $\sin^{-1}$ to be interpreted consistently—either as $\sin \cdot \sin$ and $1/\sin$ or else as $\sin \circ \sin$ and $\arcsin$—but by convention they're not.