Morphological parsing, cont.

(1) Review of last time:
   - Two halves of the parsing problem:
     - Recognizing morphologically simple words by looking them up in the list of known roots
     - Recognizing morphologically complex words by breaking them into two parts and looking up the first part in the list of roots and the second part in the list of suffixes

(2) Goals today:
   - Finish up discussion of our Excel-based approach
   - Introduce a more powerful (and conventional) technique: Finite State Automata

(3) Wrapping up the Excel-based approach:
   - I cleaned up our example file from last time a bit, and put it in the course locker: /afs/cats.ucsc.edu/courses/ling163-aa/excel/ParseExample.xls
   - I did three things to it:
     - Labeled the columns to make it easier to read; the first three columns are the ones that try to recognize the word as an unanalyzed root from one of the lexicon pages, and the remaining columns try to break it apart into a root+suffix combination and look them up separately
     - Filled out the lookups a little more completely (added verb lookups, for both roots and also suffixed forms, with possible suffix lengths of 1, 2, or 3)
     - Added a label (by concatenation) to successful lookups, so we know that the recognized word is a noun, verb, etc.

(4) Ambiguity: sometimes a word could be more than one part of speech, and take different morphology depending on its part of speech
   - For example, bark could be a verb or a noun; try adding it to the noun lexicon. What is the effect on the possible parses? Why is barks parsed as N:bark-PL and as V:bark-3SG.PRES, but not, say, as N:bark-3SG.PRES or V:bark-PL?
   - Now add a new item in the list of words to parse: barking. What are its possible parses? Why?

(5) Irregularity:
   - As it stands, two words in this list cannot be parsed at all: feet and sat
   - What is the best place to store information about these words? How should we represent it?

(6) Does our Excel-based approach achieve the following? If so, how?
   - Exhaustivity (try adding a word catch to the list of words to parse)
   - Category restrictions
   - Ordering constraints

What are some obvious limitations on this implementation? Which limitations would be easy to get around, and which not?
   - As you can see, it is completely possible (even if rather cumbersome) to do some simply parsing with Excel
   - It might not be great for very complicated cases, but could be great for quick-and-dirty rough analyses (“Which of these words ends in a possible verbal suffix?” “Which of these words contains a noun root?”)
Rather than trying to refine out Excel “parser” to be neater/more robust, we’ll turn now to a more popular, power formalism: Finite State Automata

A more sophisticated approach to parsing: Finite State Machines

(7) The intuition:

What we really want is some function, or some little machine, that “listens” to words as they come in, and tells us whether the word can be understood (parsed) or not.

(8) The kinds of things we want this machine to tell us, or “be thinking”:

(9) In other words, there are certain “states” this machine could be in

- At rest, ready to go
- Listening to inputs, but hasn’t recognized anything yet
- Recognized a possible root, ready to end
- Already recognized a root, but now there’s more; trying to recognize it as a suffix, or maybe a different root altogether
- Recognized a root + suffix combination, ready to end
- The word ended, but the machine was not ready to end yet (nothing valid was recognized); parsing failure

(10) Finite State Machines

- State 0: ready to go, nothing heard yet (start state)
- State 1: heard cat, recognized it as a N; could quit now (possible end state)
- State 2: heard catch, recognized it as a V; could quit now (another possible end state)
- State 3: heard s after a recognized N; now have recognized a N-PL, and can quit (another possible end state)
Some graphical conventions for representing Finite State Machines

- The bold circle (state 0): a start state
- The double circles: possible end states
- Numbers in the middle of the circles: just number the states so we can refer to them
- Arcs between states: labeled to show what we hear/recognize
  - By A:B, we mean that when we hear an A, we can move from one state to the next, and get back a B in the process. In this example, the machine accepts the word cat and tells us in return that it’s an N. A machine that transforms an input (like cats) to a different output (like N-PL) is called a transducer.

Getting comfortable with the formalism, with a simple, non-linguistic example:

- Consider a soda machine that costs 25 cents
- Machine accepts 25 cents in any combination of quarters, dimes, and nickels (sorry, no pennies); it requires exact change
- If you put in a quarter, you’re finished in one fell swoop:
  - Or, more precisely, you get back a soda!
- What about when you put in a nickel? What can happen in response?
  - Draw the full set of possible states and transitions for putting nickels, dimes, and quarters in the machine until it equals 25 cents (exact change)

A slightly more linguistic example: “sheep-talk” (Jurafsky and Martin, 2000)

- Suppose that the language of sheep includes the following utterances: ba, baa, baaa, baaaa, baaaaa, etc. (Any number of a’s)
- Draw a finite state machine that can accept this language (but no other)
(14) A machine that can parse words: (Slightly more detail than above)

- **cat:**
  1. Hear ‘c’, move from start state (0) to state 1, say ‘c’
  2. Hear ‘a’, move from state 1 to state 2, say ‘a’
  3. Hear ‘t’, move from state 2 to state 3, say ‘t’
  4. We’ve now heard a complete noun! Without hearing anything else (indicated with ‘0’), can move from state 3 to state 4, mark as -N. We can stop here, it’s a legal end state.
     - Note that we don’t have to move to state 4; we could wait around to see if we hear anything else (such as a ‘c’). But if the word ends here, then the only way to interpret it by ending up in a valid end state is to take advantage of the fact that we can move to state 4 without hearing anything else, and label the string cat as a noun.

    Result: machine heard ‘cat’, responded ‘cat-N’

- **cats:**
  1. Hear ‘c’, move from start state (0) to state 1, say ‘c’
  2. Hear ‘a’, move from state 1 to state 2, say ‘a’
  3. Hear ‘t’, move from state 2 to state 3, say ‘t’
  4. Label cat as a noun by moving from state 3 to state 4, and say -N.
  5. Hear ‘s’, move from state 4 to state 8, say ‘-PL’; OK for word to end here, it’s a legal end state

    Result: the machine heard ‘cats’, responded ‘cat-N-PL’

- **catch:**
  1. Hear ‘c’, move from start state (0) to state 1, say ‘c’
  2. Hear ‘a’, move from state 1 to state 2, say ‘a’
  3. Hear ‘t’, move from state 2 to state 3, say ‘t’
     - How do we know not to go to state 4 and posit that we’ve seen the word cat-N? We don’t, until we hear the next character (‘c’); there is no path from state 4 that will let us accept a ‘c’, meaning we have to back up and try a different route. We won’t worry here about the best way to find one’s path through the machine; we’ll just try to determine whether there is any path that can accept the given string.
  4. Hear ‘c’, move from state 3 to state 5.
  5. Hear an ‘h’, move from state 5 to state 6, say ‘h’
  6. Now we’re home free, can move from state 6 to state 9 without hearing anything else, label what we’ve heard so far (catch) as a -V. OK for word to end here, it’s a legal end state

    Result: the machine heard ‘catch’, responded ‘catch-V’