Generating forms

(1) The complete Spanish parser from last time:

- As we discussed last time, this is not the most compact/minimal way to represent this information (what parts of the machine could be doubled up?)
- However, it does yield the correct output

(2) Reversing the problem: how could we design a machine to produce forms, rather than parse them?

(3) Let’s go back to the English example from the beginning of last week

\[
\begin{array}{c|c|c}
1 & j & j \\
2 & m & m \\
3 & t & t \\
4 & w & w \\
5 & u & u \\
6 & i & i \\
7 & a & a \\
8 & o & o \\
9 & m & m \\
10 & s & s \\
11 & l & l \\
12 & r & r \\
13 & p & p \\
14 & o & o \\
15 & k & k \\
16 & V & V \\
17 & e & e \\
18 & d & d \\
19 & s & s \\
20 & k & k \\
21 & T & T \\
22 & T & T \\
\end{array}
\]

- Output of this machine: talk → talk-V, worked → work-V-PAST
- We want: talk-V → talk, work-V-PAST → worked
(4) Solution: just flip the input and output in the transducer

```
0 1  j  j
0 2  m  m
0 3  t  t
0 4  w  w
1 5  u  u
2 6  i  i
3 7  a  a
4 8  o  o
5 9  m  m
6 10 s  s
7 11 l  l
8 11 r  r
9 13 p  p
10 13 s  s
11 13 k  k
13 14 - 0
14 15 V  0
15 16 -  0
16 17 P  0
17 18 A  0
18 19 S  0
19 20 T  0
20 21 0 e
21 22 0 d
```

(5) A problem: what if we wanted to extend this machine to other verbs?

- A similar machine, with a larger lexicon: work, talk, jump, miss, hope, like, race, hop, sag, rub

(6) Phonological rules as FST’s: an example

- Example: n → m / [+]labial
- How can we implement this as a finite state transducer?
- What sequences do we want to change? What sequences should be left along?

(7) Schematic representation:
• Three classes of segments:
  n (eligible as input for the rule)
  p, b, m (possible environment for the rule)
  – other (irrelevant to rule)
• What happens when try to pronounce words like /lupa/, /mata/, etc?
• What happens when you try to utter /rana/?
• How about the word /lanpa/?

(8) One way of representing the possibilities:


(9) Or, in all its gorey detail: (rotated to save space)

(10) Combining morphological rules with other machines by composition

• The composition of two machines \((FST_1 \circ FST_2)\) is the result of applying \(FST_2\) to the output of \(FST_1\)

• Example:

\[
\begin{align*}
0 & \xrightarrow{dc} 1 & 1 & \xrightarrow{oa} 2 & 2 & \xrightarrow{gt} 3 \\
0 & \xrightarrow{cp} 1 & 1 & \xrightarrow{ae} 2 & 2 & \xrightarrow{tt} 3 \\
\end{align*}
\]

Yields:

\[
\begin{align*}
0 & \xrightarrow{dp} 1 & 1 & \xrightarrow{oe} 2 & 2 & \xrightarrow{gt} 3 \\
\end{align*}
\]

(11) Example: a small lexicon containing [np], [nb], [nm] sequences:

\textit{lupa, mata, rana, lanpa, kanba, tunma, fonpa, granma, tenba}
(12) Composing this lexicon with the rule above:

Examples:
kanba -> kamba
 tuna -> tumma
 lanpa -> lampa
 rana -> rana
 gramma -> gramma
 tenba -> temba

(13) What are the rules that would be needed to solve the English past tenses problem above? What would they look like?
- We will not go through the exercise of actually constructing these rules and doing the relevant compositions, because they are a bit complex; it would make a good exercise to try it, though, if you are interested!

(14) More general questions about FST’s:
- What kind of knowledge is required to create a finite state transducer of the type we’ve been looking at?
- What does it “know”?
- How would it handle new, previously unseen words?
A system for morphological rule induction

(15) Goal: learn how to generate morphologically complex forms with rules, e.g.:
- Suffixation rule: $\emptyset \rightarrow d / X_{\text{verb}} / \_ / \text{past}$
  “Suffix a [d] after a verb to form the past tense”
- Schwa epentheses: $\emptyset \rightarrow \_ / \left[ \begin{array}{c} +\text{cor} \\ -\text{cont} \\ -\text{nas} \end{array} \right] / d_{\text{word}}$
  “Insert schwa between an oral coronal stop (d, t) and a word-final d”
- Obstruent voicing assimilation: $\left[ \begin{array}{c} +\text{cor} \\ -\text{cont} \\ -\text{nas} \end{array} \right] \rightarrow [-\text{voi}] / [-\text{voi}] / \_ / \text{word}$

(16) How to learn these rules form individual words? One approach:
- Start with a list of pairs of words, that you know to embody a particular morphological relation (for example, present and past)
- Example:
  - ([mst]_{\text{pres}}, [mst]_{\text{past}}) ‘miss(edi)’
  - ([pres]_{\text{pres}}, [press]_{\text{past}}) ‘press(edi)’
  - ([læf]_{\text{pres}}, [læft]_{\text{past}}) ‘laugh(edi)’
  - ([hæg]_{\text{pres}}, [hægd]_{\text{past}}) ‘hug(edi)’
  - ([ræb]_{\text{pres}}, [ræbd]_{\text{past}}) ‘rub(edi)’
  - ([nid]_{\text{pres}}, [nidæd]_{\text{past}}) ‘need(edi)’
  - ([dʒæmp]_{\text{pres}}, [dʒæmp]_{\text{past}}) ‘jump(edi)’
  - ([plæn]_{\text{pres}}, [plænd]_{\text{past}}) ‘plan(edi)’

(17) Starting the analysis:
- Look at each input pair and see what changes from form 1 to form 2
- Rephrase in the form of a rule $A \rightarrow B / C \_ \_ D$
- Example:

$$
\begin{array}{c}
A \\
\# \text{mis} \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
B \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
C \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
D \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\_ \_ \_ \\
\end{array}
$$

Translates to: $\emptyset \rightarrow t / \# \text{mis} \_ \_ \#$
- Carried out for more pairs:
  - $\emptyset \rightarrow t / \# \text{mis} \_ \_ \#$
  - $\emptyset \rightarrow t / \# \text{pres} \_ \_ \#$
  - $\emptyset \rightarrow t / \# \text{læf} \_ \_ \#$
  - $\emptyset \rightarrow d / \# \text{hæg} \_ \_ \#$
  - $\emptyset \rightarrow d / \# \text{ræb} \_ \_ \#$
  - $\emptyset \rightarrow \_ / \# \text{nid} \_ \_ \#$
  - $\emptyset \rightarrow t / \# \text{dʒæmp} \_ \_ \#$
  - $\emptyset \rightarrow d / \# \text{plæn} \_ \_ \#$
(18) Creating more general rules: find pairs that share the same change \((A \rightarrow B)\) and see what else they share:

\[
\begin{array}{ccc}
\emptyset & t & / & m & s & \_ & \# \\
+ & \emptyset & t & / & pr & s & \_ & \#
\end{array}
\]

\[
\emptyset & t & / & X \\
\]

- Continue such comparisons iteratively.
- What would this yield, given the forms in (16)?