Week 3: Mapping output to input

Problem of the adult listener
How do I recover the lexical entries that you, my interlocutor, used in your utterance?

(This is distinct from the task of constructing a lexicon, which we’ll address in 3 weeks.)

Previous work
None, really.
• Fosler speculates about the possibility of reversing Ellison’s idea about generation.
• Hammond (1997) gives an algorithm for the listener to figure out where the syllable boundaries are
• Tesar (1998) gives an algorithm for the listener to figure out the foot structure

Generation function is many-to-one

\[
\begin{array}{c|c}
\text{input} & \text{output} \\
/bou\theta/ & [bout] \\
/bou\delta/ & [boud] \\
/bou\delta+i\eta/ & [bouri\eta] \\
/bou+t+i\eta/ & [bouri\eta] \\
\end{array}
\]

⇒ the output-to-input mapping is one-to-many.

*COMPLEX, DEP >> MAX

\[
\begin{array}{c|c}
/\text{CV}/ & [\text{CV}] \\
/\text{CCV}/ & [\text{CV}] \\
/\text{CCC}/ & [\text{CV}] \\
/\text{CCCC}/ & [\text{CV}] \\
\text{etc.} & \\
\end{array}
\]

What should the comprehension procedure yield?

Possibility #1
A set of existing lexical entries that would give the observed output.

This is not quite powerful enough, because even adults encounter new words fairly often (including proper names).

Possibility #2
A description of the set of all possible inputs that could yield that output.
Mathematical aside: Regular expressions

A regular expression is...

- $C$ an individual character
- $C^*$ the Kleene star of a regular expressions ($X^* = \text{zero or more } X$s)
- $CV$ the concatenation of two regular expressions
- $C|V$ the disjunction of two regular expressions (“or”)

**Precedence**
1. Parentheses
2. Concatenation
3. Kleene star
4. Disjunction

$CC^*V = CV, CCV, CCCV, CCCCCV, \text{ etc.}$

A language that can be described by a regular expression is called a *regular language* and can also be described by a Finite Automaton.

**Reversing syllabification**

Let’s stick with Tesar’s simplified model that we saw last time.

Under the ranking $\text{ONS} \gg \text{NOCODA} \gg \text{FILL}^\text{NUC} \gg \text{PARSE} \gg \text{FILL}^\text{ONS}$, $/VC/ \rightarrow [\text{. V.<C>}]$

$\text{ONS, PARSE} \gg \text{FILL}^\text{ONS} \rightarrow \text{empty onset is inserted when necessary}$

$\text{NOCODA, FILL}^\text{NUC} \gg \text{PARSE} \rightarrow \text{coda Cs are deleted}$

<table>
<thead>
<tr>
<th></th>
<th>$/VC/$</th>
<th>$\text{ONS}$</th>
<th>$\text{NOCODA}$</th>
<th>$\text{FILL}^\text{NUC}$</th>
<th>$\text{PARSE}$</th>
<th>$\text{FILL}^\text{ONS}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bowtie$</td>
<td>[. V.&lt;C&gt;]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>[. V.&lt;C&gt;]</td>
<td></td>
<td></td>
<td>$\ast$</td>
<td>$\ast$</td>
<td></td>
</tr>
<tr>
<td>$b$</td>
<td>[. V.C .]</td>
<td></td>
<td></td>
<td>$\ast!$</td>
<td></td>
<td>$\ast$</td>
</tr>
<tr>
<td>$c$</td>
<td>[. VC.]</td>
<td></td>
<td></td>
<td>$\ast!$</td>
<td></td>
<td>$\ast$</td>
</tr>
<tr>
<td>$d$</td>
<td>[.V.&lt;C&gt;]</td>
<td></td>
<td></td>
<td>$\ast!$</td>
<td></td>
<td>$\ast$</td>
</tr>
<tr>
<td>$e$</td>
<td>[&lt;V&gt;&lt;C&gt;]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\ast!$</td>
</tr>
</tbody>
</table>

What else creates an output that sounds like $[CV]$ under this ranking?

- $/VC/ \rightarrow [. \text{ V.<C>}]$
- $/CVC/ \rightarrow [.\text{CV.<C>}]$
- $/V/ \rightarrow [. \text{ V.}]$
- $/CV/ \rightarrow [.\text{CV.}]$

So far so good, but then there’s

- $/CVCC/ \rightarrow [.\text{CV.<C><C>}]$
- $/CVCCC/ \rightarrow [.\text{CV.<C><C><C>}]$

etc.
We could characterize Tesar’s system as assuming that *COMPLEX is undominated. Adopting that assumption,

/CCV/ \rightarrow [.C<C>V.] \quad (*COMPLEX^{CONS}, \textsc{fill}^{NUC} \gg \textsc{parse})

/CCCV/ \rightarrow [.C<C><C>V.]

eetc.

The set of inputs for this output is infinite, but we might still be able to define the set concisely (and in a way that might be usable by an algorithm that looks for a lexical match):

C*VC*

Aside: Potentially epenthetic vs. necessarily underlying segments

Under the ranking above, [ba] \leftrightarrow C*baC*

But if the output is [?a], we don’t know if the glottal stop is underlying or epenthetic.

And if the output is [bo], we don’t know if the schwa is underlying or epenthetic.

And if the output is [?o]… we can’t be sure about either segment.

In many languages, the epenthetic segment is never contrastive with zero. Could this be a protection against this kind of thing?

We’ll ignore this possibility for now, and assume that any segment could be underlying or epenthetic.

Let’s try some more rankings…

…and see what regular expressions we can use to describe the set of possible inputs for [CV].

Proposal for a comprehension algorithm
Use Tesar’s Dynamic Programming approach to build up structure from previous cells according to three operations: assume regular parsing occurred, assume underparsing occurred, assume regular parsing occurred.

[CV] \leftrightarrow ?? (shows first pass through “nothing” column)

<table>
<thead>
<tr>
<th>output read so far</th>
<th>input constructed so far</th>
<th>nothing</th>
<th>.C</th>
<th>V.</th>
</tr>
</thead>
<tbody>
<tr>
<td>nothing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>&lt;C&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>&lt;V&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;C&gt;&lt;V&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Every cell entry gets tested. To survive, it must match a cell in the output table for such a (partial) input.

For example, \(<V>.C\) would be eliminated in the second column of the comprehension table above, because it’s worse than \(\,.V.C\), as shown in the generation table below.

<table>
<thead>
<tr>
<th>input read so far</th>
<th>nothing</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>nothing</td>
<td></td>
<td>(&lt;V&gt;)</td>
<td>(&lt;V&gt;&lt;C&gt;)</td>
</tr>
<tr>
<td>Onset</td>
<td>.</td>
<td>(.,&lt;V&gt;)</td>
<td>(&lt;V&gt;.C) *Parse)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(.,&lt;V&gt;)</td>
<td>(&lt;V&gt;.C) *Parse)</td>
</tr>
<tr>
<td>Nucleus</td>
<td>.</td>
<td>,.V</td>
<td></td>
</tr>
<tr>
<td>Coda</td>
<td>.</td>
<td>,.V</td>
<td></td>
</tr>
</tbody>
</table>

The trick is to figure out when to stop building cells with deleted segments, and instead characterize the input as having a C* or V*.

Can it be proven, for instance, that once \(<C>X\) is viable and \(<C><C>X\) is viable and \(<C><C>,C>X\) is a viable, then we can replace them with \(<C>(<C>)X\)? And we have to prevent rejected structures from being considered again and again.

I haven’t proven (or even totally convinced myself) that this would work.

**Next week**
- Learning algorithms!

**For next time**
- Read Tesar & Smolensky 1996
Week 3 Lab: Generation software

Andrea Heiberg http://www.u.arizona.edu/ic/heiberg/
Does generation, using autosegmental representations.
Interleaves Gen and Eval so that the candidate set is always being pruned.

Avery Andrews OT for Windows
1. Create a directory somewhere
2. Save the file OTW11.zip in that directory
3. Double-click on the file to unzip it
Check out the paper on ROA for more info

Markus Walther OTSimple
http://www.phil-fak.uni-duesseldorf.de/~walther/otsimple.html
This runs only on UNIX, so we can’t play with it today, but you can try it on your own.

Mike Hammond 2 syllable parsers
www.u.arizona.edu/~hammond

I’ve had trouble with both the Heiberg and the Hammond, but we’ll try our luck…

Update: Heiberg and Hammond work fine with Netscape. Thanks to James Keidel for discovering this.