Interpretive Parsing

(1) *Filling in the inaudible structure of a perceived utterance*

- Prosodic structure: syllable boundaries, feet, moras...
- Morphological structure: morphemic affiliation of segments, suprasegmentals
- Syntactic structure
- Semantic structure?

In a containment approach to representations (the input is contained in the output), successful interpretive parsing also solves the problem of figuring out the input.

In a correspondence approach, it doesn’t.

(2) *Tesar & Smolensky’s approach for foot structure* (an aspect of interpretive parsing that doesn’t involve recovering the input)

- **Overt structure**: stress pattern, heavy and light syllables
- **Covert structure**: foot boundaries (left and right)
- **Assumptions about Gen**: requires exactly one stressed syllable per foot; requires exactly one head foot (gets main stress) per word; feet are maximally disyllabic; unfooted syllables are stressless
- **Grammar**: 12 constraints → 12! = 479,001,600 linear rankings (not all distinct)
- **Computation**: the set of candidates is finite (what to do if the set is infinite?). Each syllable can be at the beginning of a foot, at the end, or unfooted, so for \( n \) syllables there are \( 3^n \) footings. Moreover, for each disyllabic foot there are two possibilities for stress placement, and within a word there are as many possibilities for primary stress placement as there are feet.

(3) *T&S: Robust Interpretive Parsing*

- Given an overt form, supply the covert structure that is optimal under the current grammar. (Compare all candidates with the right overt structure.)
- **Robust**: works even for overt structures that would never be produced by the listener’s grammar.

This is important in a learning situation—the child’s grammar does not yet generate all the utterances it must interpret.
(4) Example

<table>
<thead>
<tr>
<th>σσσσσσσσ</th>
<th>FtBIN</th>
<th>FtNonFinal</th>
<th>Iambic</th>
<th>MAIN-L</th>
<th>PARSE</th>
<th>ALLFtR</th>
<th>NonFinal</th>
<th>MAIN-R</th>
</tr>
</thead>
<tbody>
<tr>
<td>(σσσσ)(σσσσ)</td>
<td>**</td>
<td>***</td>
<td></td>
<td>*</td>
<td></td>
<td>7</td>
<td>*</td>
<td>5</td>
</tr>
<tr>
<td>(σσσσ)(σσσσ)(σσσσ)</td>
<td>*!</td>
<td>*</td>
<td>***</td>
<td>*</td>
<td></td>
<td>8</td>
<td>*</td>
<td>5</td>
</tr>
<tr>
<td>(σσσσ)(σσσσ)(σσσσ)</td>
<td></td>
<td>*</td>
<td>*</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>(σσσσ)(σσσσ)(σσσσ)(σσσσ)</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>8</td>
<td>*</td>
<td>6</td>
</tr>
<tr>
<td>(σσσσ)(σσσσ)(σσσσ)(σσσσ)</td>
<td>*!</td>
<td>**</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td>9</td>
<td>*</td>
<td>6</td>
</tr>
<tr>
<td>(σσσσ)(σσσσ)(σσσσ)(σσσσ)</td>
<td>*!</td>
<td>***</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>10</td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

(5) RIP as a step in learning

- Hear an overt structure
- RIP: assign the optimal covert structure given current grammar
- Compare resulting full structure to what current grammar would have produced
- Perform Constraint Demotion accordingly

Let’s try an example.

(6) Where RIP can fail

T&S give three cases (did you find others?):

(7) Case 1: the parse can’t be an optimal output under any ranking

(a) Target grammar (assume undominated FtBin)

<table>
<thead>
<tr>
<th>σσσσσσσσ</th>
<th>FtNonFinal</th>
<th>Iambic</th>
<th>ALLFtR</th>
<th>MAIN-L</th>
<th>PARSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>σ(σσσσ)</td>
<td>**</td>
<td>**</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>σσσσσσσσ</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>σ(σσσσ)</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>σ(σσσσ)</td>
<td>*!</td>
<td></td>
<td>**</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>σ(σσσσ)</td>
<td>*!</td>
<td>**</td>
<td></td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>σ(σσσσ)</td>
<td>*!</td>
<td>**</td>
<td></td>
<td>***</td>
<td>*</td>
</tr>
<tr>
<td>σσσσσσσσ</td>
<td>*!</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td></td>
</tr>
<tr>
<td>σσσσσσσσ</td>
<td>*!</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td></td>
</tr>
<tr>
<td>σσσσσσσσ</td>
<td>*!</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td></td>
</tr>
</tbody>
</table>
(b) RIP when learner has IAMBIC and FtNonFinal (=TROCHAIC) low

<table>
<thead>
<tr>
<th></th>
<th>MAIN-L</th>
<th>PARSE</th>
<th>ALLFtR</th>
<th>IAMBIC</th>
<th>FtNonFinal</th>
</tr>
</thead>
<tbody>
<tr>
<td>σσσσσ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>σ(σσ)(σσ)</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td></td>
</tr>
<tr>
<td>σ(σσ)(σσ)</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>σ(σσ)(σσ)</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>σ(σσ)(σσ)</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>σ(σσ)(σσ)</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>σ(σσ)(σσ)</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>σ(σσ)(σσ)</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>σ(σσ)(σσ)</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>σ(σσ)(σσ)</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>σ(σσ)(σσ)</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>(σσσσσ)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(c) Learner’s current output

<table>
<thead>
<tr>
<th></th>
<th>MAIN-L</th>
<th>PARSE</th>
<th>ALLFtR</th>
<th>IAMBIC</th>
<th>FtNonFinal</th>
</tr>
</thead>
<tbody>
<tr>
<td>σσσσσ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>σ(σσ)(σσ)</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td></td>
</tr>
<tr>
<td>σ(σσ)(σσ)</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>σ(σσ)(σσ)</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>σ(σσ)(σσ)</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>σ(σσ)(σσ)</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>σ(σσ)(σσ)</td>
<td>*!</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>(σσσσσ)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(σσσσσ)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(σσσσσ)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(σσσσσ)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(σσσσσ)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(σσσσσ)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(σσσσσ)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(σσσσσ)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(σσσσσ)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(σσσσσ)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(σσσσσ)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(σσσσσ)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(σσσσσ)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(σσσσσ)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(σσσσσ)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

p. 3
(d) So learner demotes IAMBIC below FTNONFINAL.
Now the best parse is still (σσ)σ(δσ), and the optimal output is (δσ)σ(δσ).

(e) So learner demotes FTNONFINAL below IAMBIC.
¡Fatal loop!

(8) Case 2: the parse can’t be an optimal output under any ranking and moreover is harmonically bound by the current optimal output

(see p. 66)
In this case there’s nothing for CD to do—it knows which constraints to demote, but there’s nowhere to demote them to.

T&S’s solution: if this happens, stop working on this form for now, and hope that next time around, the grammar’s different.

(9) Case 3: fatal loop between two overt forms
(see p. 66)
It’s hard to come up with a general characterization of when this kind of problem will occur.

Another example of interpretive parsing

(10) Hammond’s syllable parser
- Overt structure: segments
- Covert structure: syllable boundaries (left and right)
- Computation: candidate set is finite; each segment could be parsed 4 ways (onset, nucleus, coda, extrasyllabic), so for an n-segment input, there are 4^n parses.

(11) Local encoding
Reduces candidate set by not forcing constraints to look at whole parses.

For example, *PEAK/C rules out parsing a C as a nucleus; it doesn’t care about the rest of the parse.
Construct $4 \times n$ grid:

<table>
<thead>
<tr>
<th></th>
<th>V</th>
<th>C</th>
<th>V</th>
<th>C</th>
<th>C</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>onset</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>nucleus</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>coda</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>unsyllabified</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
</tr>
</tbody>
</table>

Top-ranked constraint prunes cells that violate it, provided this doesn’t eliminate all remaining candidates.

Housekeeping at every step to ensure that every syllable has a nucleus.

Remove...
- initial coda (including if preceded by unsyllabified segments)
- final onset (including if followed by unsyllabified segments)
- onset immediately before coda (including if separated by unsyllabified segments)

(12) Hammond’s example

*PEAK/C $>>$ *MARGIN/V $>>$ PARSE $>>$ ONSET $>>$ *COMPLEX $>>$ NOCODA

*PEAK/C

```
  a  g  e  n  d  a
  o  o  o  o  o  o
  n  n  n  n  n  n
  c  c  c  c  c  c
  u  u  u  u  u  u
```

Housekeeping

```
  a  g  e  n  d  a
  o  o  o  o  o  o
  n  n  n  n  n  n
  c  c  c  c  c  c
  u  u  u  u  u  u
```

*MARGIN/V

```
  a  g  e  n  d  a
  o  o  o  o  o  o
  n  n  n  n  n  n
  c  c  c  c  c  c
  u  u  u  u  u  u
```

(housekeeping does nothing)
PARSE

<table>
<thead>
<tr>
<th>a</th>
<th>g</th>
<th>e</th>
<th>n</th>
<th>d</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
</tr>
</tbody>
</table>

(housekeeping does nothing)

ONSET

- *n* must be preceded by *o*
  - delete an *n* when the only option in the column to the left is non-*o* (as long as *n* isn’t the last remaining possibility in its column)
  - delete a *c*, *n* or *u* when only option in column to the right is *n* (as long as the *c*, *n*, or *u* isn’t the last remaining possibility in its column)

<table>
<thead>
<tr>
<th>a</th>
<th>g</th>
<th>e</th>
<th>n</th>
<th>d</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
</tr>
</tbody>
</table>

(housekeeping does nothing)

*COMPLEX

- delete *o* when obligatorily preceded or followed by *o*
- delete *c* when obligatorily preceded or followed by *c*

<table>
<thead>
<tr>
<th>a</th>
<th>g</th>
<th>e</th>
<th>n</th>
<th>d</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
</tr>
</tbody>
</table>

We have to prune the columns for [n] here, because there’s only one choice left in the [d] column.

(a)(gen)(da)

(13) *Issue for Hammond in evaluating nonlocal constraints*

What happens if *COMPLEX* encounters this configuration?

(after *PEAK/C, *MARGIN/V)

<table>
<thead>
<tr>
<th>V</th>
<th>C</th>
<th>C</th>
<th>V</th>
<th>C</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
</tr>
</tbody>
</table>
It has to skip both the potential complex margins, because they are not yet obligatory.

If NoCoda applies next, then Parse we get:

<table>
<thead>
<tr>
<th>NOCoda</th>
<th>housekeeping</th>
<th>PARSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>V C C V C C</td>
<td>V C C V C C</td>
<td>V C C V C C</td>
</tr>
<tr>
<td>o o o o o o</td>
<td>o o o o o o</td>
<td>o o o o o o</td>
</tr>
<tr>
<td>n n n n n n</td>
<td>n n n n n n</td>
<td>n n n n n n</td>
</tr>
<tr>
<td>c c c c c c</td>
<td>c c c c c c</td>
<td>c c c c c c</td>
</tr>
<tr>
<td>u u u u u u</td>
<td>u u u u u u</td>
<td>u u u u u u</td>
</tr>
</tbody>
</table>

Now there’s a complex onset: (V)(CCV)CC
Standard OT would have given us (VC)(CV)CC, because *complex >> NoCoda.

(14) Hammond sees two ways out
(i) revise the algorithm so it can consider full parses, at a cost to efficiency
(ii) consider that maybe this is how OT really works, at least in parsing.

I don’t know that anyone’s actually explored (ii)—is the factorial typology under this method of constraint evaluation a good match to reality

Mapping output to input

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 talked about last time.)

(15) Previous work
None, really. Fosler speculates about the possibility of reversing Ellison’s idea about generation using FAs.

(16) Generation function is many-to-one

\[
\begin{array}{c|c}
\text{input} & \text{output} \\
/bou\text{t}/ & [bou] \\
/bou\text{d}/ & [bou\text{d}] \\
/bou\text{d+\text{\textipa{\texteta}}}/ & [bou\text{r\textipa{\texteta}}] \\
/bou\text{t+\textipa{\texteta}}/ & \\
\end{array}
\]

⇒ comprehension function is one-to-many.
(17) *Even worse example: one-to-ininitely-many*

*COMPLEX, DEP >> MAX

/\CV/    [CV]
/\CCV/   
/\CCCV/   
/\CCCCV/  

etc.

(18) *What should the comprehension procedure yield?*

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

  This is not 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.

(19) *Mathematical aside: Regular expressions*

A regular expression is…

- **C**  an individual character
- **C**  the Kleene star of a regular expressions (X* = zero or more Xs)
- **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

\[CC^*V = CV, CCV, CCCV, CCCCV, \text{etc.}\]

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

(20) *Reversing syllabification*

Use Tesar’s simplified model (no complex margins), which we’ll see in two weeks.

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

\ONS, \PARSE >> \FILL^\ONS \rightarrow empty onset is inserted when necessary
NOCODA, FILL$^\text{NUC}$ >> PARSE $\rightarrow$ coda Cs are deleted

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

(21) What else creates an output that sounds like [CV] under this ranking?

/VC/ $\rightarrow$ [. V.<C>]
/CVC/ $\rightarrow$ [.CV.<C>]
/N/ $\rightarrow$ [. V.]
/CV/ $\rightarrow$ [.CV.]

/CVCC/ $\rightarrow$ [.CV.<C><C>]
/CVCCC/ $\rightarrow$ [.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.] (*COMPLEX$^\text{ONS}$, FILL$^\text{NUC}$ >> PARSE)
/CCCV/ $\rightarrow$ [.C<C><C>V.]

etc.

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*

Big question: can the set of inputs that generates some output under an OT grammar always be described with a regular expression?
(22) Phonological aside: Potentially epenthetic vs. necessarily underlying segments

Under the ranking above, [ba] \( \equiv /C^*baC^*/\)

Assuming we know that the epenthetic segments are [?] and [\(\alpha\)]...

- if the output is [?a], we don’t know if the glottal stop is underlying or epenthetic.
- if the output is [ba], we don’t know if the schwa is underlying or epenthetic.
- if the output is [?\(\alpha\)], 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.

(23) Let’s try some more rankings…

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

Three ideas for doing comprehension

(24) Idea #1

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 overparsing occurred.
(Sorry: this assumes some material we haven’t covered yet)

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

<table>
<thead>
<tr>
<th>input constructed so far</th>
<th>output read 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></td>
<td>&lt;C&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
<td>&lt;V&gt;</td>
<td>&lt;C&gt;&lt;V&gt;</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></td>
<td>$V.C *FillOns$</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.

(25) Idea #2
Colin Wilson suggested this, though I may be mangling the idea.

Start with the perceived string (CV) and try moving one step in every direction in a multidimensional phonological space:

```
  C
 CVV  CV  CCV
 V
```

From the viable possibilities found, branch out one further.

How many steps should you go in a direction that doesn’t seem to be working before giving up on that direction?

(26) Idea #3: Use the constraint hierarchy to prune a FA

Start with set of all syntactically well-formed inputs.

---

p. 11
For markedness constraint, “affirm” underlying forms that violate it (they’re OK, because the high ranking of the markedness constraint means they’d be repaired).

*COMPLEX:

\[
\begin{array}{c}
\text{C} \\
\text{C} \\
\text{V} \\
\text{C}
\end{array}
\]

ONSET:

\[
\begin{array}{c}
\text{C} \\
\text{C} \\
\text{V} \\
\text{C}
\end{array}
\]

NoCODA:

\[
\begin{array}{c}
\text{C} \\
\text{C} \\
\text{V} \\
\text{C}
\end{array}
\]

For a faithfulness constraint, eliminate arcs that, in being mapped to the known output, would violate it (unless already affirmed).

FILLNUC:

\[
\begin{array}{c}
\text{C} \\
\text{C} \\
\text{V} \\
\text{C}
\end{array}
\]
What’s left is taken to describe the set of possible inputs (C*VC*).

There’s still a lot to be worked out, and this FA is rigged to do syllable structure.

Let’s try some other rankings:
*COMPLEX >> FILLONS >> ONSET >> PARSE >> NOCODA >> FILLNUC
  - *COMPLEX >> PARSE >> FILLNUC is tricky
  - If we get rid of the possibility of complex margins (Tesar’s simplified model),
    I think it works.
  - How do we mechanize which arcs PARSE should eliminate?

Problem: this is fine if the output has one syllable, but what if it has more?

Perhaps we need to combine the generic input template with the observed output to make
a starting machine that will be of different size each time. That’s then the machine that
gets pruned.

Next Time
- Formalizing OT: standard model and Primitive OT

For next time
- Read
  o Samek-Lodovici & Prince (1999)
  o Eisner (1997)
- Homework
  o Ponder the questions below on the Samek-Lodovici & Prince article
  o If you’re enrolled for credit and you have a late assignment, get it to me by
    Wednesday at 6PM so I can take it with me to LA to grade.
1. How does S-L&P’s main goal relate to factorial typology?
2. What is a bounding set?
3. What is a favoring hierarchy?
4. Did you find anywhere where S-L&P had to make a simplifying assumption? Why was that assumption necessary? Do you have any ideas about how the proposal could be modified to remove that simplifying assumption?