Grammar in Performance and Acquisition: recognition

E Stabler, UCLA

ENS Paris • 2008 • day 3
Q1  How are utterances interpreted ‘incrementally’?
Q2  How is that ability acquired, from available evidence?
Q3  Why are some constituent orders unattested across languages?
Q4  What kind of grammar makes copying a natural option?

- we don’t need to start from zero (start from grammar!)
- frame explanations supported by convergent evidence
‘Pour the egg in the bowl over the flour’
• Recognition: sequences → \{true, false\}
• Parsing: sequences → Trees ∪ \{false\}

How to design a recognition/parsing strategy:

• **Understand what you are parsing!**
  - separate grammar definition from procedural issues
  - in parser, stay as close to grammar mechanisms as possible
  - consider time+memory after finding sound+complete algorithm

• **Lessons from well-understood problems, esp. CFG parsing:**
  - ∀algorithm ⇒ separate representations of each derivation
    (We can maybe exclude grammars with infinite ambiguity
     ... But in human languages, as in CFLs, the number of
     derivations/string not bounded by polynomial)
  - Two main strategies (can both be used at once!):
    • Store all trees built, sharing structure (‘chart’, ‘packed forest’)
    • Carefully select steps that look like they’re building the
      desired trees, backtracking and reanalyzing when necessary
review

<table>
<thead>
<tr>
<th></th>
<th>Pierre</th>
<th>Marie</th>
<th>praises</th>
<th>who</th>
<th>ε</th>
<th>and</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>:D</td>
<td>:D</td>
<td>:=D</td>
<td>:D</td>
<td>:=V</td>
<td>:=C</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>:=D</td>
<td></td>
<td>:=V</td>
<td>:=C</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>:=D</td>
<td></td>
<td>:=V</td>
<td>:=C</td>
</tr>
</tbody>
</table>

ε ::= V +wh C
and ::= C =C C
review: steps 1, 2, 3

merge(2,4) = \[7\]

merge(7,1) = \[8\]

merge(5,8) = \[9\]

\[\epsilon: +wh\ C\ \rightarrow\ \text{Marie} \rightarrow \text{praises} \rightarrow \text{who}:-wh\]

\[\text{VP} \rightarrow \text{DP} \rightarrow \text{D'} \rightarrow \text{D} \rightarrow \text{who} \rightarrow \text{D'} \rightarrow \text{D} \rightarrow \text{praises} \rightarrow \text{Marie} \rightarrow \text{who}:-wh\]
step 3: what do derived structures represent?
trees and labeled bracketing

```
<
\epsilon:+wh C >
Marie <
praises who:-wh
```
trees and labeled bracketing

\[
\begin{array}{c}
\langle \\
\epsilon:+wh \ C \\
\rangle \\
\langle \\
Marie \\
\rangle \\
\langle < \\
praises \\
who:-wh \\
\rangle
\end{array}
\]

\[
[\langle [\epsilon:+wh \ C] \rangle [\langle Marie \rangle [\langle praises \rangle [\langle who:-wh \rangle]]]]
\]
trees and labeled bracketing

\[
\langle \varepsilon : +wh \ C \rangle > \langle Marie \rangle \langle praises \ who:-wh \rangle
\]

\[
[\langle [\varepsilon : +wh \ C] \ [\rangle [Marie] [\langle praises \ who:-wh]]\rangle]
\]

\[
[\langle [\varepsilon : +wh \ C] \ [\rangle Marie [\langle praises \ (who:-wh)])\rangle]
\]
trees and labeled bracketing

\[
\begin{align*}
\text{[< $[\epsilon:+wh\ C]$ [>] [Marie] [< [praises][who:-wh]]]]} \\
\text{[< $[\epsilon:+wh\ C]$ [>] Marie [< praises (who:-wh)]]]} \\
\text{[< $[\epsilon:+wh\ C]$ [>] Marie praises (who:-wh)]]}
\end{align*}
\]
trees and labeled bracketing

\[
< \\
\varepsilon:+wh \ C \ > \\
\text{Marie} \\
\text{praises} \quad \text{who:-wh}
\]

\[
[< \[\varepsilon:+wh \ C\] \[> [\text{Marie} \ [< [\text{praises} \ [\text{who:-wh}]]]]\] \\
[< \[\varepsilon:+wh \ C\] \[> \text{Marie} \ [< \text{praises (who:-wh)}]]\] \\
[< \[\varepsilon:+wh \ C\] \[> \text{Marie praises (who:-wh)}]]\] \\
[< (\varepsilon:+wh \ C) \text{Marie praises (who:-wh)}]]\]
\]
trees and labeled bracketing

\[
\langle \epsilon:+wh \text{ C} \rangle < \\
\langle \text{Marie} \rangle < \\
\text{praises} \quad \text{who:-wh}
\]

\[
[< [\epsilon:+wh \text{ C}] [>] [\text{Marie}] [< [\text{praises}] [\text{who:-wh}]]]]

[< [\epsilon:+wh \text{ C}] [>] \text{Marie} [< \text{praises} (\text{who:-wh})]]]

[< [\epsilon:+wh \text{ C}] [>] \text{Marie praises (who:-wh)]]

[< (\epsilon:+wh \text{ C}) \text{Marie praises (who:-wh)]}

((\text{Marie praises:+wh C),(who:-wh)))
EM: two different cases

\[
\text{praises::=}D = D \ V + \text{Pierre::}D \ \Rightarrow \ < \\
\text{praises::=}D \ V \ \text{Pierre}
\]

\[
\text{praises::=}D = D \ V + \text{who::}D - \text{wh} \ \Rightarrow \ < \\
\text{praises::=}D \ V \ \text{who:-wh}
\]

Unlike Pierre, the DP who is a mover

So second result has 2 active ‘chains’
what the syntax must see

<
praises:⇒D V Pierre
≡ praises Pierre:⇒D V

<
praises:⇒D V who:-wh
≡ praises:⇒D V, who:-wh

<
ε:+wh C
Marie <
praises who:-wh
≡ Marie praises:+wh C, who:-wh
example:

<table>
<thead>
<tr>
<th></th>
<th>Marie::D</th>
<th>praises::=D =D V</th>
<th>who::D -wh</th>
<th>ε::=V +wh C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
example:

\[
\begin{array}{c|c}
1 & \text{Marie}::D \\
2 & \text{praises}::=D =D V
\end{array}
\quad
\begin{array}{c|c}
4 & \text{who}::D -\text{wh} \\
5 & \epsilon::=V +\text{wh} C
\end{array}
\]

merge(2, 4) = praises::=D V, who:-wh

A
example:

\[
\begin{array}{ll}
1 & \text{Marie}::D \\
2 & \text{praises}::=D =D V
\end{array}
\]

who::D -wh

\[
\begin{array}{ll}
\epsilon::=V +wh C
\end{array}
\]

merge(2, 4) = praises: =D V, who: -wh

merge(A, 1) = Marie praises: V, who: -wh
example:

\[
\begin{array}{cccc}
1 & \text{Marie}::D & \text{who}::D & \text{wh} \\
2 & \text{praises}::=D & =D & V \\
4 & \epsilon::=V & +\text{wh} & C \\
5 & \\
\end{array}
\]

merge(2, 4) = praises::=D V, who::-wh

merge(A, 1) = Marie praises::V, who::-wh

merge(5, B) = Marie praises::+wh C, who::-wh
example:

\[
\begin{array}{c}
1 & \text{Marie::D} & \text{who::D -wh} \\
2 & \text{praises::=} & \text{=}D V \\
4 & \epsilon::=V & +wh C \\
5 & & \\
\end{array}
\]

merge(2,4) = praises::=D V, who:-wh  
merge(A,1) = Marie praises::V, who:-wh  
merge(5,B) = Marie praises::+wh C, who:-wh  
move(C) = who Marie praises::C
derived structures and derivation trees

\[ \text{who Marie praises: } C \]
\[ \text{Marie praises: } +\text{wh } C, \text{who:}-\text{wh} \]
\[ \epsilon::= V +\text{wh } C \]
\[ \text{Marie praises: } V, \text{who:}-\text{wh} \]
\[ \text{praises:}=D \quad V, \text{who:}-\text{wh} \]
\[ \text{Marie::D} \]
\[ \text{praises::=} D =D \quad V \quad \text{who::D} \quad -\text{wh} \]
derived structures and derivation trees

\[
\begin{align*}
\epsilon &::= V + \text{wh} \ C \\
\text{Marie} &::= D \\
\text{praises} &::= D = D \ V \\
\text{who} &::= D - \text{wh}
\end{align*}
\]
minimalist grammar $G=\langle \text{Lex}, \mathcal{F} \rangle$ reformulated

- **vocabulary** $\Sigma = \{\text{every, some, student, ...}\}$
- **types** $T = \{::, :\}$  
  
  "lexical" and "derived"  
- **syntactic features** $\mathcal{F}$:
  
  - $C, T, D, N, V, P, ...$ (selected categories)
  - $=C, =T, =D, =N, =V, =P, ...$ (selector features)
  - $+\text{wh}, +\text{case}, +\text{focus}, ...$ (licensors)
  - $-\text{wh}, -\text{case}, -\text{focus}, ...$ (licensees)

- **Chains** $C = \Sigma^* \times T \times F^*$
- **expressions** $E = C^+$
- **lexicon** $\text{Lex} \subseteq C^+$, a finite subset of $\Sigma^* \times \{::\} \times F^*$
derived structures and derivation trees

(remember why this reformulation is being considered now: we need to decompose derivations to share structure)
**em:** \((E \times E) \rightarrow E\) is the union of the following 3 functions, for \(\cdot \in \{;,:,\}\), \(\gamma \in F^*, \delta \in F^+\)

\[
\begin{align*}
\text{s} :: &= f \gamma \cdot f, \alpha_1, \ldots, \alpha_k \\
\text{em1: lexical item selects non-mover} \\
\text{st} :&= \gamma, \alpha_1, \ldots, \alpha_k
\end{align*}
\]

\[
\begin{align*}
\text{s} :&= f \gamma, \alpha_1, \ldots, \alpha_k \cdot f, \iota_1, \ldots, \iota_l \\
\text{em2: non-lex selects non-mover} \\
\text{ts} :&= \gamma, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l
\end{align*}
\]

\[
\begin{align*}
\text{s} \cdot &= f \gamma, \alpha_1, \ldots, \alpha_k \cdot f \delta, \iota_1, \ldots, \iota_l \\
\text{em3: any item selects mover} \\
\text{s} :&= \gamma, \alpha_1, \ldots, \alpha_k, \text{ } t : \delta, \iota_1, \ldots, \iota_l
\end{align*}
\]

(Here, \(\alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l (0 \leq k, l)\) are any chains)
**im:** $E \rightarrow E$ is the union of the following 2 functions, for $\gamma \in F^*$, $\delta \in F^+$,

$$
s : +f \gamma, \alpha_1, \ldots, \alpha_{i-1}, t : -f, \alpha_{i+1}, \ldots, \alpha_k
$$

$$
ts : \gamma, \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k
$$

**im1:** final move

$$
s : +f \gamma, \alpha_1, \ldots, \alpha_{i-1}, t : -f \delta, \alpha_{i+1}, \ldots, \alpha_k
$$

$$
s : \gamma, \alpha_1, \ldots, \alpha_{i-1}, t : \delta, \alpha_{i+1}, \ldots, \alpha_k
$$

**im2:** nonfinal move

**(SMC)** none of the chains $\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k$ has $-f$ as its first feature,
Given a minimalist grammar $G$, how can we tell if string $s \in L(G)$?

0. Represent input by a finite state machine, with all possible empty elements:

\[
\begin{align*}
(0,0) & ::= VC \\
(0,0) & ::= VC + \text{wh} \\
(0,1) & ::= \text{who} \\
(1,1) & ::= VC \\
(1,1) & ::= VC + \text{wh} \\
(1,2) & ::= \text{Marie} \\
(2,2) & ::= VC \\
(2,2) & ::= VC + \text{wh} \\
(2,3) & ::= \text{praises} \\
(3,3) & ::= VC \\
(3,3) & ::= VC + \text{wh}
\end{align*}
\]
1. Replace each lexical item by its features, in a matrix $m$(input).

$$(0,0)::=VC \quad (1,1)::=VC \quad (2,2)::=VC \quad (3,3)::=VC$$

$$(0,0)::=VC+wh \quad (1,1)::=VC+wh \quad (2,2)::=VC+wh \quad (3,3)::=VC+wh$$

$$(0,1)::D-wh \quad (1,2)::D \quad (2,3)::=D =D V$$

2. Close $m$(input) with respect to merge, where each string is given now by the matrix indices, (enforcing adjacency reqs for $em1,em2,im1$; none for $em3,im2$)

3. **Success** if $(0, |input|) \cdot Start$

With the ‘deductive parsing’ implementation of *Shieber, Schabes & Pereira (1994)*, only a small bit of code is needed, available from the webpage. The method is called CKY because it is based on early work of Cocke, Kasami, and Younger on parsing context free languages (Aho and Ullman, 1972; Sikkel and Nijholt, 1997).
Example:

1. \((0,0)::=VC\) \hspace{1cm} \((1,1)::=VC\) \hspace{1cm} \((2,2)::=VC\) \hspace{1cm} \((3,3)::=VC\) \\
   \((0,0)::=VC+wh\) \hspace{1cm} \((1,1)::=VC+wh\) \hspace{1cm} \((2,2)::=VC+wh\) \hspace{1cm} \((3,3)::=VC+wh\) \\
   \((0,1)::D-wh\) \hspace{1cm} \((1,2)::D\) \hspace{1cm} \((2,3)::=D =D V\)

2. Now, close w.r.t merge. First, using em3:

   \[
   (2,3)::=D =D V \quad (0,1)::D \quad -wh
   \]

   \[
   (2,3)::=D V , (0,1)::-wh
   \]

So we add this to the matrix...
Example: continuing 2...

\[(0,0)::=V C \quad (1,1)::=V C \quad (2,2)::=V C \quad (3,3)::=V C\]
\[(0,0)::=VC+wh \quad (1,1)::=VC+wh \quad (2,2)::=VC+wh \quad (3,3)::=VC+wh\]
\[(0,1)::D-wh \quad (1,2)::D \quad (2,3)::=D =D V \quad (2,3)::=D V , (0,1)::=wh\]

Now we can use em2:

\[(2,3)::=D V , (0,1)::=wh \quad (1,2)::D\]
\[(1,3)::=V , (0,1)::=wh\]

We add this to the matrix...
Example: continuing 2...

(0,0)::=VC  (1,1)::=VC  (2,2)::=VC  (3,3)::=VC
(0,0)::=VC+wh (1,1)::=VC+wh (2,2)::=VC+wh (3,3)::=VC+wh
(0,1)::D-wh (1,2)::D  (2,3)::=D =D V
(2,3)::=D V,(0,1)::−wh
(1,3)::V,(0,1)::−wh

Now we can use em1:

(1,1)::=V C  (1,3)::V,(0,1)::−wh
(1,3)::+wh C, (0,1)::−wh

We add this to the matrix...
Example: continuing 2...

\[(0,0):=VC \quad (1,1):=VC \quad (2,2):=VC \quad (3,3):=VC\]
\[(0,0):=VC+wh \quad (1,1):=VC+wh \quad (2,2):=VC+wh \quad (3,3):=VC+wh\]
\[(0,1):D-wh \quad (1,2):D \quad (2,3):=D =D V \quad (2,3):=D V,(0,1):-wh\]
\[(1,3):V,(0,1):-wh \quad (1,3):+wh C,(0,1):-wh\]

Now we can use im1:

\[(1,3):+wh C, (0,1):-wh\]
\[(0,3):C\]

We add this to the matrix...
Example: continuing 2...

\[
\begin{align*}
(0,0)::=VC & \quad (1,1)::=VC & \quad (2,2)::=VC & \quad (3,3)::=VC \\
(0,0)::=VC+wh & \quad (1,1)::=VC+wh & \quad (2,2)::=VC+wh & \quad (3,3)::=VC+wh \\
(0,1)::D-wh & \quad (1,2)::D & \quad (2,3)::=D = D \ V & \\
(1,3):V,(0,1):-wh & \quad (1,3):+wh C,(0,1):-wh & \quad (2,3):=D \ V,(0,1):-wh \\
(0,3):C & \\
\end{align*}
\]

We can now answer:

**Step 3.** Does the table contain \((0, 3) \cdot C\)? **Yes**
0 who 1 Marie 2 praises 3

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(=V C)</td>
<td>(D -wh)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(=V +wh C)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>(=V C)</td>
<td>(D)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(=V +wh C)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>(=V C)</td>
<td>(=D =D V)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(=V +wh C)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>(=V C)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(=V +wh C)</td>
</tr>
</tbody>
</table>

(Look up how to compute closures in Cormen et al’92, §26.2, or other text on algorithms)
**0** who **1** Marie **2** praises **3**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(=V C)</td>
<td>(D -wh)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(=V +wh C)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>(=V C)</td>
<td>(D)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(=V +wh C)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>(=V C)</td>
<td>(=D =D V)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(=V +wh C)</td>
<td>(=D V,(0,1):-wh)</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>(=V C)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(=V +wh C)</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>---</td>
<td>--------------------</td>
<td>--------------------</td>
<td>--------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>0</td>
<td>((\geq V \ C))</td>
<td>(D \ -wh))</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>((\geq V + wh \ C))</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>((\geq V \ C))</td>
<td>(D)</td>
<td>((\geq V,(0,1):-wh))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>((\geq V + wh \ C))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>((\geq V \ C))</td>
<td>((\geq D = D \ V))</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>((\geq V + wh \ C))</td>
<td>((\geq D \ V,(0,1):-wh))</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>((\geq V \ C))</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>((\geq V + wh \ C))</td>
</tr>
</tbody>
</table>
0 who 1 Marie 2 praises 3

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(=V C)</td>
<td>(D -wh)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(=V +wh C)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(=V C)</td>
<td>(D)</td>
<td>(V,(0,1):-wh)</td>
<td>(+wh C,(0,1):-wh)</td>
</tr>
<tr>
<td></td>
<td>(=V +wh C)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(=V C)</td>
<td>(=D =D V)</td>
<td>(=D V,(0,1):-wh)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(=V +wh C)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(=V C)</td>
<td></td>
<td>(=V +wh C)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(=V +wh C)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
0 who 1 Marie 2 praises 3

(matrix guarantees a completed derivation, which we can now collect... )
Soundness, completeness, complexity:

- *(sound)* for every $G$, item derived only if licensed by $G$
- *(complete)* for every $G$, if licensed by $G$, item derived
- The number of possible entries in any cell is finitely bounded.
- No more than $O(n^{4m+4})$ steps (Harkema’00), $m$ a constant depending on the number of licensees in the grammar.
- Generalizes to copying (P-MCFG translation)
- Generalizes to arbitrary semi-rings (probabilities, weights)

**Incremental?**

- This CKY-like method is **bottom-up** and **all-paths-at-once**.
- Given

  `(you) pour the egg in the bowl over the flour`  

  when is *egg* or *egg in the bowl* related to the object position of *pour*?

  *(NB: question not clear!)*
'Pour the | egg | in the | bowl over the flour'

(Chambers et al., 2004)
From trees to tuples
CKY: From tuples to matrices
Assessment

E Stabler, UCLA

Grammar in Performance and Acquisition: recognition
cky

From trees to tuples

CKY: From tuples to matrices

Assessment

\[
[] ::= v + i + k \ T
\]

\[
[] ::= P + k + p = v \ v
\]

\[
[] ::= V + k = D \ v
\]

\[
[] ::= P + k + p = N \ N
\]

\[
in ::= D \ P - p
\]

\[
\text{the} ::= N \ D - k
\]

\[
\text{egg} ::= N
\]

\[
\text{bowl} ::= N
\]

E Stabler, UCLA

Grammar in Performance and Acquisition: recognition
**TD Motivations:**

- **CKY method not incremental(?)** Lacks ‘prefix property’:
  - A parser has the prefix property iff it halts on any prefix of the input that cannot be extended to a successful parse.

- **Marcus’80 proposes a bottom-up method to minimize local ambiguity.** He proves that even then, and even with lookahead, backtracking cannot be avoided. He shows, English is not LR($k$) for any $k$.

- Recently, a different idea is to build structures that are fully connected, so that they can be interpreted incrementally.

- **Top-down (TD) parsing builds fully connected trees at every point.** (+proposed to explain grammatical facts, Chesi et al)
**TD Motivations:**

- **CKY method not incremental (?)** Lacks ‘prefix property’:
  - A parser has the prefix property iff it halts on any prefix of the input that cannot be extended to a successful parse.

- **Marcus’80 proposes a bottom-up method to minimize local ambiguity.** He proves that even then, and even with lookahead, backtracking cannot be avoided. He shows, English is not LR(\(k\)) for any \(k\).

- Recently, a different idea is to build structures that are fully connected, so that they can be interpreted incrementally.

- **Top-down (TD) parsing builds fully connected trees at every point.** (+proposed to explain grammatical facts, Chesi et al) For CFGs, TD is **non-terminating**, and even when terminating is **intractable**. TD for MGs has these same problems and more... (discussed in Harkema’01)
Earley: A TD recognition strategy that works

- Earley’68 showed how TD and BU methods can be combined to avoid TD nontermination. The basic idea is simply:
  - Constituents are predicted TD, using chart representation
  - Predicted elements are completed BU, and then new predictions are generated TD

- Method extends to TAGs and MGs, with ‘prefix property’ (for MGs, see Harkema’01; for TAGs, Vijay-Shanker’87)

- For incremental parsing, one idea is: semantically analyze one TD prediction path from the chart at a time, while completing BU and storing completed elements as required by Earley, so that reanalysis is feasible.
So far 3

- simple formalisms can model many linguistic proposals
- a straightforward semantics values every constituent

Q1 What performance models allow incremental interpretation (and remnant movement, doubling constructions)?

- CKY efficiently parses every MGC
- Earley efficiently parses every MGC, with (factored) representation of TD derivations
- Probabilistic Earley may model TD choice
- Can we interpret TD partial constituents like [DP [V...]]? Yes, but many open questions!
  (Hale’08, Shieber&Johnson’94, Stabler’91, Steedman’89)

Q2 How is this ability acquired, from available evidence?

Becker, Tilman, Owen Rambow, and Michael Niv. 1992. The derivational generative power of formal systems, or, scrambling is beyond LCFRS. IRCS technical report 92-38, University of Pennsylvania.


