Sharpening the empirical claims of generative syntax through formalization

Tim Hunter

University of Minnesota, Twin Cities

NASSLLI, June 2014
Part 1: Grammars and cognitive hypotheses
  What is a grammar?
  What can grammars do?
  Concrete illustration of a target: Surprisal

Parts 2–4: Assembling the pieces
  Minimalist Grammars (MGs)
  MGs and MCFGs
  Probabilities on MGs

Part 5: Learning and wrap-up
  Something slightly different: Learning model
  Recap and open questions
Sharpening the empirical claims of generative syntax through formalization

Tim Hunter — NASSLLI, June 2014

Part 2

Minimalist Grammars
Outline

5 Notation and Basics

6 Example fragment

7 Recursion

8 Derivation trees
Wait a minute!

“I thought the whole point was deciding between candidate sets of primitive derivational operations! Isn’t it begging the question to set everything in stone at the beginning like this?”
Wait a minute!

“I thought the whole point was deciding between candidate sets of primitive derivational operations! Isn’t it begging the question to set everything in stone at the beginning like this?”

- We’re not setting this in stone — we will look at alternatives.
- But we need a concrete starting point so that we can make the differences concrete.
- What’s coming up is meant as a relatively neutral/“mainstream” starting point.
Defining a grammar in the MG formalism is defining a set $Lex$ of lexical items:

- A lexical item is a string with a sequence of features.
  e.g. $like :: =d =d v$, $mary :: d$, $who :: d \_wh$

- Generates the closure of the $Lex \subseteq Expr$ under two derivational operations:
  
  - $\text{MERGE} : Expr \times Expr \xrightarrow{\text{partial}} Expr$
  - $\text{MOVE} : Expr \xrightarrow{\text{partial}} Expr$

- Each feature encodes a requirement that must be met by applying a particular derivational operation.
  
  - $\text{MERGE}$ checks $=f$ and $f$
  - $\text{MOVE}$ checks $+f$ and $-f$

- A derived expression is complete when it has only a single feature remaining unchecked.
Merge and move

= f α

f β

MERGE

<

α

β

= f α

f β

MERGE

>

β

α

>+ f α

= -f β

MOVE

β

α
Examples

$$\text{MERGE} \left( \text{eat} :: =d \ v, \ \text{it} :: d \right) = \begin{array}{c}
\text{eat} :: v \\
\text{it} :: \\
\end{array}$$

$$\text{MERGE} \left( \text{the} :: =n \ d, \ \text{book} :: n \right) = \begin{array}{c}
\text{the} :: d \\
\text{book} :: \\
\end{array}$$

$$\text{MERGE} \left( \begin{array}{c}
\text{eat} :: =d \ v, \\
\text{the} :: d \\
\text{book} :: \\
\end{array} \right) = \begin{array}{c}
\text{eat} :: v \\
\begin{array}{c}
\text{the} :: \\
\text{book} :: \\
\end{array} \\
\end{array}$$

$$\text{MERGE} \left( \text{which} :: =n \ d \ -\text{wh}, \ \text{book} :: n \right) = \begin{array}{c}
\text{which} :: d \ -\text{wh} \\
\text{book} :: \\
\end{array}$$

$$\text{MERGE} \left( \begin{array}{c}
\text{eat} :: =d \ v, \\
\text{which} :: d \ -\text{wh} \\
\text{book} :: \\
\end{array} \right) = \begin{array}{c}
\text{eat} :: v \\
\begin{array}{c}
\text{which} :: \\
\text{book} :: \\
\end{array} \\
\end{array}$$
Examples

MERGE

\[
\begin{align*}
\text{will} &::= v & \text{=d} & \text{t}, & \text{eat} &::= v & < \\
& & & & \text{which} &::= -\text{wh} & \text{book} ::
\end{align*}
\]

= \[
\begin{align*}
\text{will} &::= \text{d} & \text{t} & < \\
\text{eat} &::= < \\
& & \text{which} &::= -\text{wh} & \text{book} ::
\end{align*}
\]

MERGE

\[
\begin{align*}
\text{will} &::= \text{d} & \text{t} & < \\
& & \text{eat} &::= < \\
& & & & \text{which} &::= -\text{wh} & \text{book} ::
\end{align*}
\]

, \text{John} :: \text{d}

= \[
\begin{align*}
\text{will} &::= \text{t} & < \\
\text{eat} &::= < \\
& & \text{which} &::= -\text{wh} & \text{book} ::
\end{align*}
\]
Examples

\[
\begin{align*}
\text{MERGE} & \quad \left( \begin{array}{l}
\epsilon :: = t + \text{wh } c, \\
\text{will} :: t \\
\text{eat} :: \\
\text{which} :: -\text{wh} \\
\text{book} :: 
\end{array} \right)
\end{align*}
\]

\[
\begin{align*}
\text{MOVE} & \quad \left( \begin{array}{l}
\epsilon :: +\text{wh } c \\
\text{John} :: \\
\text{will} :: \\
\text{eat} :: \\
\text{which} :: -\text{wh} \\
\text{book} :: 
\end{array} \right)
\end{align*}
\]
Merge and move

\[ = f \alpha \]

\[ = f \alpha \]

\[ = f \alpha \]

\[ = f \alpha \]

\[ + f \alpha \]

\[ - f \beta \]

\[ f \beta \]

\[ f \beta \]

\[ f \beta \]

\[ f \beta \]

\[ \alpha \]

\[ \beta \]

\[ \beta \]

\[ \alpha \]

\[ \beta \]

\[ \alpha \]

\[ \beta \]

\[ \alpha \]

\[ \beta \]

\[ \alpha \]

\[ \beta \]

\[ \alpha \]

\[ \beta \]
Definitions

\[
\text{MERGE}(e_1[=f \alpha], e_2[f \beta]) = \begin{cases} 
[< e_1[\alpha] e_2[\beta]] & \text{if } e_1[=f \alpha] \in \text{Lex} \\
[> e_2[\beta] e_1[\alpha]] & \text{otherwise}
\end{cases}
\]

\[
\text{MOVE}(e_1[+f \alpha]) = [> e_2[\beta] e_1'[\alpha]]
\]

where \(e_2[-f \beta]\) is a unique subtree of \(e_1[+f \alpha]\)

and \(e_1'\) is like \(e_1\) but with \(e_2[-f \beta]\) replaced by an empty leaf node
Shortest Move Constraint

How do we know which subtree should be displaced when we apply MOVE?

By stipulation, there can only ever be one candidate. This is the Shortest Move Constraint (SMC).

\[
\begin{align*}
&\epsilon :: \text{+wh}\ c \\
&\text{who} :: \text{-wh} \\
&\text{ate} :: \text{what} :: \text{-wh} \\
\end{align*}
\]

is undefined

Q: Multiple wh-movement?  
A: Clustering!
Shortest Move Constraint

How do we know which subtree should be displaced when we apply MOVE?

By stipulation, there can only ever be one candidate. This is the Shortest Move Constraint (SMC).

Q: Multiple wh-movement?
A: Clustering!
(7)  

a. 

```
>  \rightarrow \text{cluster}

what: \forall \text{wh} \Delta \text{wh}

<

gave: v
to-whom: \Delta \text{wh}

what: \Delta \text{wh}
to-whom
gave: v

\epsilon
```  

b. 

```
>  \rightarrow \text{cluster}

who: \forall \text{wh} \sim \text{wh}

<

\emptyset: i

<

<

what: \Delta \text{wh}
to-whom

\epsilon
gave
```

```
>  \rightarrow \text{cluster}

who: \sim \text{wh}

<

\emptyset: i

<

<

what

io-whom

\epsilon

<

<

gave

\epsilon
```  

c. 

```
<  \rightarrow \text{move}

\emptyset: +\text{wh}.c

<

<

<

who: \sim \text{wh}

<

<

\emptyset

<

<

\emptyset

<

<

what

io-whom

\epsilon

<

<

gave

\epsilon

<

gave

\epsilon
```
Notation

= d v or = dp vp?

Categorial grammar:
- Primitive symbols for “complete” things, e.g. S, NP
- Derived symbols for “incomplete” things, e.g. S\NP
- Lexical category can specify “what’s missing”

Traditional X-bar theory:
- Primitive symbols for “incomplete” things, e.g. V, T
- Derived symbols for “complete” things, e.g. VP, TP (= V″, T″)
- Separate subcategorization info specifies “what's missing”

MGs:
- Primitive symbols for “complete” things, like CG
- So t means “a complete projection of T”, not “a T head”
Notation comparison

<table>
<thead>
<tr>
<th>Conventional notation</th>
<th>MG notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘eat which book’ is a VP</td>
<td>VP label on root</td>
</tr>
<tr>
<td>‘which book’ must move</td>
<td>( \text{\textsuperscript{(\text{-wh})}} ) on ‘which’</td>
</tr>
<tr>
<td>‘will’ combines with a VP</td>
<td>implicit</td>
</tr>
</tbody>
</table>
### Notation comparison

- **Conventional notation**: VP label on root
- **MG notation**: \( v \) on ‘eat’, \( \neg \text{wh} \) on ‘which’, \( =v \) on ‘will’

#### Example fragment

**Recursion**

#### Derivation trees

- **T**
- **V**
- **DP**
- **D**
- **N**

- **VP**
- **which**
- **book**

- **will** combines with a VP
- **eat**

- **will** :: \( =v =d t \)
- **eat** :: \( v \)

- **which** :: \( \neg \text{wh} \)

- **book** ::
Outline

5 Notation and Basics

6 Example fragment

7 Recursion

8 Derivation trees
A Minimalist Grammar

cake :: d
John :: d -k
eat :: =d =d v
will :: =v +k t
what :: d -wh
who :: d -k -wh
ε :: =t +wh c
ε :: =t c
A Minimalist Grammar

cake :: d  
John :: d -k  
eat :: =d =d v  
will :: =v +k t

what :: d -wh  
who :: d -k -wh  
ε :: =t +wh c  
ε :: =t c

Example fragment

cake :: d  
john :: d -k  
eat :: =d =d v  
will :: =v +k t

what :: d -wh  
who :: d -k -wh  
ε :: =t +wh c  
ε :: =t c

Recursion

Derivation trees
A Minimalist Grammar

cake :: d
John :: d -k
eat :: =d =d v
will :: =v +k t

what :: d -wh
who :: d -k -wh
ε :: =t +wh c
ε :: =t c

[Diagram of derivation trees]
A Minimalist Grammar

cake :: d

what :: d -wh

John :: d -k

who :: d -k -wh

eat :: =d =d v

ε :: =t +wh c

will :: =v +k t

ε :: =t c
A Minimalist Grammar . . . which overgenerates

cake :: d  
what :: d -wh 
John :: d -k  
who :: d -k -wh 
eat :: =d =d v  
ε :: =t +wh c 
will :: =v +k t  
ε :: =t c
A Minimalist Grammar ... which overgenerates

cake :: d 
what :: d -wh 
John :: d -k 
who :: d -k -wh 
eat :: =d =d v 
epsilon :: =t +wh c 
will :: =v +k t 
epsilon :: =t c 

\[
\begin{array}{c}
\text{cake} \\
\downarrow \\
\text{eat} :: v \\
\text{John} :: -k \\
\text{will} :: +k t \\
\downarrow \\
\text{cake} \\
\downarrow \\
\text{eat} :: \\
\text{John} :: -k \\
\downarrow \\
\text{will} :: t \\
\end{array}
\]

\[
\begin{array}{c}
\text{cake} \\
\downarrow \\
\text{eat} :: \\
\text{John} :: -k \\
\downarrow \\
\text{will} :: t \\
\end{array}
\]
A Minimalist Grammar ... which overgenerates

cake :: d  
what :: d -wh

John :: d -k  
who :: d -k -wh

eat :: =d =d v  
ε :: =t +wh c

will :: =v +k t  
ε :: =t c
A Minimalist Grammar . . . which overgenerates

cake :: d  
what :: d -wh  
John :: d -k  
who :: d -k -wh  
eat :: =d =d v  
ε :: =t +wh c  
will :: =v +k t  
ε :: =t c  

```
<
will :: +k t

<
what :: -wh

<
eat ::  
John :: -k

>
what ::  

>
ε :: c

>
John ::  

>
will ::  

>
eat ::
```
A Minimalist Grammar . . . which overgenerates

cake :: d
what :: d -wh
John :: d -k
who :: d -k -wh
eat :: =d =d v
ε :: =t +wh c
will :: =v +k t
ε :: =t c

John will eat cake
what John will eat
who will eat cake

John will cake eat
what John will eat
who will cake eat
A Minimalist Grammar . . . which overgenerates

cake :: d  
what :: d -wh
John :: d -k  
who :: d -k -wh
eat :: =d =d v  
ε :: =t +wh c
will :: =v +k t  
ε :: =t c

S → NP VP  
VP → V NP
NP → John  
VP → runs
NP → Mary  
VP → walks
V → loves

John will eat cake  
what John will eat  
who will eat cake
John will cake eat  
what John will eat  
who will cake eat

John runs  
John walks  
John loves John  
John loves Mary
Mary runs  
Mary walks  
Mary loves John  
Mary loves Mary
First solution: covert movement

cake :: d -k  
John :: d -k  
eat :: =d +k =d v  
will :: =v +k t

what :: d -k -wh  
who :: d -k -wh  
ε :: =t +wh c  
ε :: =t c
First solution: covert movement

cake :: d \ -k
John :: d \ -k
eat :: =d \ +\overline{k} =d \ v
cake ::

will :: =v \ +k \ t
cake ::

cake :: d \ -k
what :: d \ -k \ -wh
John :: d \ -k
who :: d \ -k \ -wh
eat :: =d \ +\overline{k} =d \ v
cake ::

\epsilon :: =t \ +wh \ c
cake ::

\epsilon :: =t \ c
cake ::
First solution: covert movement

\[
\begin{align*}
\text{cake} &:: d \ -k \\
\text{what} &:: d \ -k \ -wh \\
\text{John} &:: d \ -k \\
\text{who} &:: d \ -k \ -wh \\
\text{eat} &:: =d + \bar{k} =d \ v \\
\epsilon &:: =t + wh \ c \\
\text{will} &:: =v + k \ t \\
\epsilon &:: =t \ c
\end{align*}
\]

Note order of features on \textit{eat}!
Second solution

Separate d into subj and obj

\[
\begin{align*}
\text{cake} &:: \text{obj} \\
\text{John} &:: \text{subj} -\text{k} \\
\text{eat} &:: =\text{obj} =\text{subj} \ \text{v} \\
\text{will} &:: =\text{v} +\text{k} \ \text{t}
\end{align*}
\]

\[
\begin{align*}
\text{what} &:: \text{obj} -\text{wh} \\
\text{who} &:: \text{subj} -\text{k} -\text{wh} \\
\epsilon &:: =\text{t} +\text{wh} \ \text{c} \\
\epsilon &:: =\text{t} \ \text{c}
\end{align*}
\]

Problem “solved”:

\[
\begin{align*}
\text{John will eat cake} \\
\text{what John will eat} \\
\text{who will eat cake}
\end{align*}
\]
### Adding recursion

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>cake</td>
<td>obj</td>
</tr>
<tr>
<td>John</td>
<td>subj -k</td>
</tr>
<tr>
<td>eat</td>
<td>=obj =subj v</td>
</tr>
<tr>
<td>will</td>
<td>=v +k t</td>
</tr>
<tr>
<td>what</td>
<td>obj -wh</td>
</tr>
<tr>
<td>who</td>
<td>subj -k -wh</td>
</tr>
<tr>
<td>to</td>
<td>=v inf</td>
</tr>
<tr>
<td>seem</td>
<td>=inf v</td>
</tr>
<tr>
<td>ε</td>
<td>=t +wh c</td>
</tr>
<tr>
<td>ε</td>
<td>=t c</td>
</tr>
</tbody>
</table>
Adding recursion

cake :: obj
John :: subj -k
eat :: =obj =subj v
will :: =v +k t
what :: obj -wh
to :: =v inf
who :: subj -k -wh
seem :: =inf v
ε :: =t +wh c
ε :: =t c

John will eat cake
what John will eat
who will eat cake
John will seem to eat cake
what John will seem to eat
who will seem to eat cake
Adding recursion

cake :: obj
John :: subj -k
eat :: =obj =subj v
will :: =v +k t

what :: obj -wh
who :: subj -k -wh
ε :: =t +wh c

to :: =v inf
seem :: =inf v

John will eat cake
what John will eat
who will eat cake
John will seem to eat cake
what John will seem to eat
who will seem to eat cake

<
   \text{to} :: \text{inf}
     <
       \text{John} :: -k
         <
           \text{eat} :: \text{cake} ::

<
  \text{seem} :: \text{v}
    <
      \text{to} ::
        <
          \text{John} :: -k
            <
              \text{eat} :: \text{cake} ::

<
  \text{will} :: +k t
    <
      \text{seem} ::
        <
          \text{to} ::
            <
              \text{John} :: -k
                <
                  \text{eat} :: \text{cake} ::

Reminder: Recursion in a CFG

S → NP VP
NP → Det N’
N’ → N
N’ → N PP
PP → P NP
VP → runs
Det → the
N → dog
N → cat
P → near

S
  ├── NP
  │   └── VP
  │       └── runs
  ├── S
  │   ├── NP
  │   │   └── N
  │   │       └── cat
  │   └── VP
  │       └── runs
  ├── Det
  │   │ └── the
  │   └── N
  │       └── N
  └── PP
      └── P
          └── near

N’
  └── N
      └── N
          └── N
              └── N
                  └── N
                      └── N
                          └── N
                              └── N
                                  └── N
                                      └── N
                                          └── N
                                              └── N
                                                  └── N
                                                      └── N
                                                          └── N
                                                              └── N
                                                                  └── N
                                                                      └── N
                                                                          └── N
                                                                              └── N
                                                                                  └── N
                                                                                      └── N
                                                                                          └── N
                                                                                                  └── N
                                                                                                      └── N
                                                                                                          └── N
                                                                                                              └── N
                                                                                                                  └── N
                                                                                                                      └── N
                                                                                                                          └── N
                                                                                                                                 └── N
                                                                                                                                              └── N
                                                                                                                                                └── N
                                                                                                                                                        └── N
                                                                                                                                                                └── N
                                                                                                                                                    └── N
                                                                                                               └── N
                                                                                                                           └── N
                                                                                                                               └── N
                                                                                                                                 └── N
                                                                                                                                                  └── N
                                                                                                                                                        └── N
                                                                                                                                                    └── N
                                                                                                                                                        └── N
                                                                                                                └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
                                                                                                            └── N
Which lexical items will produce recursion?

- **to** :: =v inf
- **seem** :: =inf v
- **schmink** :: =t v
- **think** :: =t =subj =v
The old derivation

\[
\text{cake} :: \text{eat} :: v
\]

\[
\text{John} :: -k
\]

\[
\text{v, \{-k\}}
\]

\[
\text{<}
\]

\[
\text{eat} :: v, \text{cake} ::
\]
The old derivation

```
Notation and Basics  Example fragment  Recursion  Derivation trees

The old derivation

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```
The old derivation

```
John ::

will :: t

eat ::

cake ::

+\ k \ t, \ \{-k\}

v, \ \{-k\}

> 
<
```

```
t, \ \{}\n```

```
> 
<
```
Derivation with *seem*

\[
\begin{align*}
to & ::= v \text{ inf} \\
seem & ::= \text{inf } v
\end{align*}
\]
Derivation with *seem*

\[ to :: = v \text{ inf} \]
\[ seem :: = \text{inf } v \]
Derivation with *seem*

\[ to :: = v \text{ inf} \]
\[ seem :: = \text{inf} \ v \]
Derivation with \textit{schmink}

\textit{schmink} ::= \texttt{t v}

\begin{center}
\begin{tikzpicture}[level distance=1.5cm, sibling distance=1cm, every node/.style={draw, rounded corners, align=center, font=\small}]
  \node (schmink) {\textit{schmink}}
    child {node (eat) {\textit{eat} :: v}}
    child {node (cake) {\textit{cake} ::}}
    child {node (will) {\textit{will} :: \{-k\}}}
    child {node (john) {\textit{John} :: \{-k\}}}
  \end{tikzpicture}
\end{center}
Derivation with \textit{schmink}

\textit{schmink} :: =t v

\begin{center}
\textbf{Diagram}
\end{center}
Derivation with *schmink*

schmink :: = t v

```
john ::

will :: t

>  <

v, {-k}

>  <

eat ::

cake ::

+ t, {-k}

t, {}
```
Derivation with *schmink*

\[
\text{schmink} :: = \text{t v}
\]
Derivation with *think*

\[
\text{think} ::= t = \text{subj} \ v
\]
Derivation with \textit{think}

\textit{think} ::= t = \textit{subj} v
Derivation with *think*

*think* :: =t =subj v

```
at, {}

John ::

> <

will :: t

> <

v, {-k}

+kt, {-k}

eat ::

cake ::
```
Derivation with *think*

\[ \text{think ::= t =subj v} \]

```
think ::= t =subj v

think ::= t =subj v

John ::= will ::= eat ::

will ::=

\text{John ::= will ::= eat ::}
```

\[ \text{t, \{} \]

\[ \text{+k t, \{-k\}} \]

\[ \text{v, \{-k\}} \]

\[ \text{cake ::} \]

\[ \text{+k t, \{-k\}} \]

\[ \text{v, \{-k\}} \]

\[ \text{cake ::} \]
Derivation with *think*

**think ::= t =subj v**

![Derivation tree for think](attachment:derivation_tree.png)
Which lexical items will produce recursion?

- `to :: = v inf`
- `seem :: = inf v`
- `schmink :: = t v`
- `think :: = t = subj = v`
Importance of the SMC

The SMC ensures that there is a finite number of types (that we care about).

Recall: $\text{MOVE}$ is undefined

\[
\begin{align*}
\epsilon :: +\text{wh c} \\
\text{who} :: -\text{wh} \\
\text{ate} :: \quad \text{what} :: -\text{wh}
\end{align*}
\]
Importance of the SMC

The SMC ensures that there is a finite number of types (that we care about).

Recall: \[
\text{MOVE}
\begin{cases}
  \epsilon :: \text{+wh c} \\
  \text{who} :: \text{-wh} \\
  \text{ate} :: \text{what} :: \text{-wh}
\end{cases}
\]
is undefined

So MOVE cannot be applied to expressions of type “+wh c with two -wh things moving out of it” (we might have written this +wh c, \{-wh, -wh\}).

(Michaelis 2001)
Importance of the SMC

The SMC ensures that there is a finite number of types (that we care about).

Recall: \( \text{MOVE} \)  

\[
\begin{array}{c}
\epsilon :: +\text{wh c} \\
\text{who} :: -\text{wh} \\
\text{ate} :: \text{what} :: -\text{wh}
\end{array}
\]

is undefined

- So \text{MOVE} cannot be applied to expressions of type “+\text{wh c} with two -\text{wh} things moving out of it” (we might have written this +\text{wh c}, \{-\text{wh}, -\text{wh}\}).
- Nor to expressions of type +\text{wh c}, \{-\text{wh} -k, -\text{wh}\}.
- These are “dead end” types.

(Michaelis 2001)
Importance of the SMC

The SMC ensures that there is a **finite number of types** (that we care about).

Recall: \( \text{MOVE} \left( \begin{array}{c}
\epsilon :: +\text{wh } c \\
who :: -\text{wh} \\
ate :: \text{what} :: -\text{wh}
\end{array} \right) \) is undefined

- So \( \text{MOVE} \) cannot be applied to expressions of type “\(+\text{wh } c \) with two \(-\text{wh} \) things moving out of it” (we might have written this \(+\text{wh } c, \{-\text{wh}, -\text{wh}\}\)).
- Nor to expressions of type \(+\text{wh } c, \{-\text{wh} -k, -\text{wh}\}\).
- These are “dead end” types.
- An expression of type \( t, \{-\text{wh} -k, -\text{wh}\} \) can be the input to \( \text{MERGE} \).
Importance of the SMC

The SMC ensures that there is a **finite number of types** (that we care about).

Recall: $\text{MOVE} \left( \begin{array}{c} < \\ ℰ :: +w \ c \\ > \\ \text{who} :: -w \\ < \\ \text{ate} :: \text{what} :: -w \end{array} \right)$ is **undefined**

- So **MOVE** cannot be applied to expressions of type "+$w \ c$ with two $-w$ things moving out of it" (we might have written this $+$+$w \ c$, $\{−w, −w\}$).
- Nor to expressions of type $+$+$w \ c$, $\{−w −k, −w\}$.
- These are “dead end” types.
- An expression of type $t$, $\{−w −k, −w\}$ can be the input to **MERGE**.
- But such types are also bound to lead to dead ends.

(Michaelis 2001)
The SMC ensures that there is a finite number of types (that we care about).

Recall: \( \text{MOVE} \) cannot be applied to expressions of type “+wh c with two -wh things moving out of it” (we might have written this +wh c, \{-wh, -wh\}).

Nor to expressions of type +wh c, \{-wh -k, -wh\}.

These are “dead end” types.

An expression of type \( t, \{-wh -k, -wh\} \) can be the input to \( \text{MERGE} \).

But such types are also bound to lead to dead ends.

So any type of the form \( \alpha, \{\ldots, -f\alpha_i, \ldots, -f\alpha_j, \ldots\} \) is not useful.

Thus there are only a finite number of useful types.

(Michaelis 2001)
Outline

5 Notation and Basics

6 Example fragment

7 Recursion

8 Derivation trees
<
  seem :: v
  to ::
    >
      John :: -k
      <
        eat :: cake ::

seem :: =inf v

<
  inf, {-k}
  to :: inf
  >
    John :: -k
    <
      eat :: cake ::

to :: =v inf

>
  v, {-k}
  John :: -k
  <
    eat :: v
    cake ::
A possible concern

Question

“But hasn’t our eventual derived expression lost the information that ‘cake’ is a DP?”
Derivations

\[
\text{John ate cake} :: \text{S} \\
\text{John} :: \text{NP} \quad \text{ate cake} :: \text{VP} \\
\text{ate} :: \text{V} \quad \text{cake} :: \text{NP}
\]
Derivations

John ate cake :: S

John :: NP  ate cake :: VP

ate :: V  cake :: NP

ate :: (S\NP)/NP  cake :: NP

John :: NP  ate cake :: S\NP

John ate cake :: S

John :: NP  ate cake :: S\NP

ate :: (S\NP)/NP  cake :: NP
A possible concern

Question

“But hasn’t our eventual derived expression lost the information that ‘cake’ is a DP?”

Answer

Yes, but only in the same way that John ate cake :: S has also lost this information.

The point is not that we can look at the whole derivation to retrieve this, it’s that that info has already done its job.
We separate the derivational precursor relation from the part-whole relation.
Labeling of internal nodes

```
John ate cake :: S
  John :: NP
  ate cake :: VP
    ate :: V
    cake :: NP
```
Labeling of internal nodes

```
John ate cake :: S
  John :: NP
  ate cake :: VP
    ate :: V
    cake :: NP

S
  John :: NP
  VP
    ate :: V
    cake :: NP
```
Labeling of internal nodes

\[
\begin{align*}
\text{ate} &:: (\text{S}\backslash \text{NP})/\text{NP} & \text{cake} &:: \text{NP} \\
\text{John} &:: \text{NP} & \text{ate cake} &:: \text{S}\backslash \text{NP} \\
\hline
\text{John ate cake} &:: \text{S}
\end{align*}
\]
Labeling of internal nodes

\[
\begin{array}{c}
\text{John} :: \text{NP} \\
\text{ate} :: (S \text{NP})/\text{NP} \quad \text{cake} :: \text{NP} \\
\hline
\text{ate cake} :: S \text{NP} \\
\hline
\text{John ate cake} :: S
\end{array}
\]
Notation and Basics

Example fragment

Recursion

Derivation trees

\[ t, \{ \} \]

\[ +k \ t, \{-k\} \]

\[ will ::= v +k \ t \]

\[ v, \{-k\} \]

\[ =subj \ v, \{\} \]

\[ Mary ::= subj \ -k \]

\[ think ::= t =subj \ v \]

\[ t, \{\} \]

\[ +k \ t, \{-k\} \]

\[ will ::= v +k \ t \]

\[ v, \{-k\} \]

\[ =subj \ v, \{\} \]

\[ John ::= subj \ -k \]

\[ eat ::= obj =subj \ v \]

\[ cake ::= \ obj \]
Context-free structure

Schemas for **MERGE** steps:

\[
\langle \gamma, \alpha_1, \ldots, \alpha_j, \beta_1, \ldots, \beta_k \rangle \rightarrow \langle =f \gamma, \alpha_1, \ldots, \alpha_j \rangle \langle f, \beta_1, \ldots, \beta_k \rangle \\
\langle \gamma, \alpha_1, \ldots, \alpha_j, \delta, \beta_1, \ldots, \beta_k \rangle \rightarrow \langle =f \gamma, \alpha_1, \ldots, \alpha_j \rangle \langle f \delta, \beta_1, \ldots, \beta_k \rangle
\]

Schemas for **MOVE** steps:

\[
\langle \gamma, \alpha_1, \ldots, \alpha_{i-1}, \alpha_i+1, \ldots, \alpha_k \rangle \rightarrow \langle +f \gamma, \alpha_1, \ldots, \alpha_{i-1}, -f, \alpha_i+1, \ldots, \alpha_k \rangle \\
\langle \gamma, \alpha_1, \ldots, \alpha_{i-1}, \delta, \alpha_i+1, \ldots, \alpha_k \rangle \rightarrow \langle +f \gamma, \alpha_1, \ldots, \alpha_{i-1}, -f \delta, \alpha_i+1, \ldots, \alpha_k \rangle
\]
Context-free structure

Schemas for **MERGE** steps:

\[
\langle \gamma, \alpha_1, \ldots, \alpha_j, \beta_1, \ldots, \beta_k \rangle \rightarrow \langle =f \gamma, \alpha_1, \ldots, \alpha_j \rangle \langle f, \beta_1, \ldots, \beta_k \rangle
\]

\[
\langle \gamma, \alpha_1, \ldots, \alpha_j, \delta, \beta_1, \ldots, \beta_k \rangle \rightarrow \langle =f \gamma, \alpha_1, \ldots, \alpha_j \rangle \langle f \delta, \beta_1, \ldots, \beta_k \rangle
\]

Schemas for **MOVE** steps:

\[
\langle \gamma, \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k \rangle \rightarrow \langle +f \gamma, \alpha_1, \ldots, \alpha_{i-1}, -f, \alpha_{i+1}, \ldots, \alpha_k \rangle
\]

\[
\langle \gamma, \alpha_1, \ldots, \alpha_{i-1}, \delta, \alpha_{i+1}, \ldots, \alpha_k \rangle \rightarrow \langle +f \gamma, \alpha_1, \ldots, \alpha_{i-1}, -f \delta, \alpha_{i+1}, \ldots, \alpha_k \rangle
\]

- **MOVE** steps **change** something without **combining** it with anything
- Compare with unary CFG rules, or type-raising in CCG, or ...
\[c, \{}\]

\[\epsilon ::= t +wh c\]

\[t, \{-wh\}\]

\[+k t, \{-k, -wh\}\]

\[will ::= v +k t\]

\[v, \{-k, -wh\}\]

\[=subj v, \{-wh\}\]

\[Mary ::= subj \ -k\]

\[think ::= t =subj v\]

\[t, \{-wh\}\]

\[+k t, \{-k, -wh\}\]

\[will ::= v +k t\]

\[v, \{-k, -wh\}\]

\[=subj v, \{-wh\}\]

\[John ::= subj \ -k\]

\[eat ::= obj =subj v\]

\[what ::= obj \ -wh\]
Notation and Basics

Example fragment

Recursion

Derivation trees
Importance of the SMC

The SMC ensures that there is a **finite number of types** (that we care about).

Recall: \( \text{MOVE} \) is undefined

- So \( \text{MOVE} \) cannot be applied to expressions of type "\(+\text{wh } c\) with two \(-\text{wh}\) things moving out of it" (we might have written this \( +\text{wh } c, \{-\text{wh}, -\text{wh}\} \)).
- Nor to expressions of type \( +\text{wh } c, \{-\text{wh} -k, -\text{wh}\} \).
- These are “dead end” types.

- An expression of type \( t, \{-\text{wh} -k, -\text{wh}\} \) can be the input to \( \text{MERGE} \).
- But such types are also bound to lead to dead ends.

So any type of the form \( \alpha, \{\ldots, -f\alpha_i, \ldots, -f\alpha_j, \ldots\} \) is not **useful**.

Thus there are only a finite number of useful types.

(Michaelis 2001)


