INFLECTION
MG versus TLG

Willemijn Vermaat – UiL-OTS, Utrecht

31st January 2002
Contents

1  The problem  
2  The data  
3  Syntactic proposals  
4  Minimalist Grammar  
5  Type-logical grammar  
6  Recapitulation  

3  
5  
8  
10  
17  
21
1. The problem

Syntax

Generative syntax seeks answers to linguistic phenomena.

Provide an abstract theory that:

- captures the data descriptively
- can be applied cross-linguistically
- generalizes to similar phenomena in one language
- can be build in one bigger framework

Formal frameworks

Formal frameworks, such as Type Logical Grammar and Minimalist Grammar, might provide the basis for such an abstract theory.
Similarities

The computational system of the three frameworks are similar:

- basic operations: Merge and Move,
- lexicon: lexical and functional categories,
- important role played by features

Plan of action

- analyze empirical data
- implement data in both formal frameworks
- compare the two analysis
- make generalisations that might provide answers for generative syntacticians
2. The data

Verb movement to the functional category I

**Emonds (1978):**

- French seem to have a rule to move finite verbs out of VP
- English does not seem to have such a rule

**French**

(1) Guillaume aime Maxima
(2) Guillaume a aime Maxima
(3) Guillaume n’ aime pas Maxima \[ V_{lex fin} < \text{Neg} \]
(4) \* Guillaume (ne) pas aime Maxima
(5) Guillaume n’a jamais pas aimé Maxima \[ V_{aux fin} < \text{Neg} \]
English

(6) William loves Maxima
(7) William has loved Maxima
(8) * William loves not Maxima
(9) William does not love Maxima $\text{Neg} < V_{\text{lex} fin}$
(10) William has never loved Maxima $V_{\text{aux} fin} < \text{Neg}$

Data shows:

- ordering of French and English $\text{Neg}$ and $V_{\text{lex} fin}$ is reversed
- English auxiliaries ‘have’ and ‘be’ appear to move like French main verbs

Conclusion: data shows that finite lexical verbs occur in different positions in the two languages
Use of the inflection category

A functional category as INFL is used for:

1. to position *auxiliaries* or particle *to*

2. inflection: *affix* to mark grammatical properties such as *number, person, tense, case*

3. interpretational/semantic: *past* event

4. structurally (movement trigger): *splitINFL*
   - AgrS:agrees with the features of the subject
   - Tense: check [tense] feature of the verb
3. Syntactic proposals

Some ideas

SplitInfl  The I-node is split into a Tense and an Agreement node (Pollock, 1989)

HMC  Head Movement Constraint: a head can move only to next head higher up

Relativized Minimality  “if an item is going to move, it moves to the next appropriate position up” (Rizzi, 1990):

  - an XP (argument) can cross over a head, but not a specifier position, to the next XP-position
  - a head an cross over a specifier but not over a filled head-position

Proper binding condition  referentially incomplete expression (such anaphors) require antecedents’:

  - binding of traces: when something moves it has to move upward, in order to bind its own trace
Questions

- Why in English does V never move to AgrP?
- How does the affix bind with the finite verb if inflection in English is analysed as affixal?

Answer: Lasnik: a hybrid system to explain variety within English

1. Featural I in French; I is a bundle of abstract features \([F_1, \ldots, F_n]\) that need to be checked

2. Featural I in English for sentences with auxiliaries: John is singing

3. Affixal I in English for declarative sentences with lexical verbs: affix hopping, only when affix is adjacent to verb stem

4. Affixal I in English in negative sentence, the attachment of the affix to V is blocked by the NEG-position, so one needs do-insertion
4. Minimalist Grammar

A minimalist grammar $MG = (\sum, F, Types, Lex, F)$

**Features** $F$:
- base $B = \{v, n, np, case, wh, \ldots\}$
- selectors $S = \{=f|f \in B\}$
- licensees $M = \{-f|f \in B\}$
- licensors $N = \{+f|f \in B\}$
- features $F = B \cup S \cup M \cup N$

**Chains** $C$
- Types $T = \{;:\}$
- chains $C = \sum^* T F^*$
- lexical chains $LC = \sum^* :: F^*$
- expressions $E = C^+$
- lexicon $Lex \subset LC^+$
- minimalist grammar $G = Lex$
Operations

Merge \(:(E \times E) \rightarrow E\)

\[
\begin{align*}
  \text{s} :: &= f \gamma \cdot f, \alpha_1, \ldots, \alpha_k \\
  \text{st} : \gamma, \alpha_1, \ldots, \alpha_k & \quad r1
\end{align*}
\]

if \(s\) is lexical, and \(t\) has one \([f]\)

\[
\begin{align*}
  \text{s} : &= f \gamma, \alpha_1, \ldots, \alpha_k \cdot f, t_1, \ldots, t_l \\
  \text{ts} : \gamma, \alpha_1, \ldots, \alpha_k, t_1, \ldots, t_l & \quad r2
\end{align*}
\]

if \(s\) is derived, and \(t\) has one \([f]\)

\[
\begin{align*}
  \text{s} \cdot &= f \gamma, \alpha_1, \ldots, \alpha_k \cdot f \delta, t_1, \ldots, t_l \\
  \text{s} : \gamma, \alpha_1, \ldots, \alpha_k, t : \delta t_1, \ldots, t_l & \quad r3
\end{align*}
\]

if \(s\) is derived, and \(t\) has one \([f]\) and a set of features \(\delta\)
**Move** : $E \rightarrow E$

\[
\begin{align*}
  s : & \ +f \gamma, \alpha_1, \ldots, \alpha_{i-1} & t : & \ -f, \alpha_{i+1}, \ldots, \alpha_k \\
  ts : & \ \gamma, \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k & m1
\end{align*}
\]

if $s$ is derived, and $t$ in the chain is the only element (SMC) with one $[-f]$

\[
\begin{align*}
  s : & \ +f \gamma, \alpha_1, \ldots, \alpha_{i-1} & t : & \ -f \delta, \alpha_{i+1}, \ldots, \alpha_k \\
  s : & \ \gamma, \alpha_1, \ldots, \alpha_{i-1}, t : \ \delta, \alpha_{i+1}, \ldots, \alpha_k & m2
\end{align*}
\]

if $s$ is derived, and $t$ in the chain is the only element (SMC) with a $[-f]$ followed by a non-empty set of features $\delta$

**Example of phrasal movement**

Grammar is taken from “*Structural Similarity*” (Keenan; Stabler, 2000)

This grammar shows how the English surface order SVIO is derived within MG.
Lexicon:

<table>
<thead>
<tr>
<th>Lexical:</th>
<th>Functional:</th>
</tr>
</thead>
<tbody>
<tr>
<td>lavinia :: d -k</td>
<td>-s :: =pred +v +k i</td>
</tr>
<tr>
<td>titus :: d -k</td>
<td>ϵ :: =vp +k =d pred</td>
</tr>
<tr>
<td>praise :: =d vp -v</td>
<td>praise :: =d vp -v</td>
</tr>
</tbody>
</table>

Derivation

\[ \begin{align*}
\epsilon &::=vp +k =d \ pred \\
\epsilon &::=vp +k =d \ pred, \ praise : -v, \ lavinia : -k \\
lavinia &::=d \ pred, \ praise : -v \\
\text{r}^3 &::=d \ pred, \ praise : -v \\
lavinia &::=d \ pred, \ titus : -k, \ praise : -v \\
\text{m}^1 &::=d \ pred, \ titus : -k, \ praise : -v \\
-s &::=pred +v +k ~i \\
\text{r}^1 &::=d \ pred +v +k ~i, \ titus : -k, \ praise : -v \\
\text{m}^1 &::=d \ pred +v +k ~i, \ titus : -k, \ praise : -v \\
\text{m}^1 &::=d \ pred +v +k ~i, \ titus : -k, \ praise : -v \\
\text{m}^1 &::=d \ pred +v +k ~i, \ titus : -k, \ praise : -v \\
\end{align*} \]
Extending MG

- In order to allow head movement and affix hopping, we need an extra set of special selecting features:
  1. a left $f<=$ and a right $=>f$ incorporator
  2. a left $<=f$ and a right $f=>$ hopper

- The lexical chains and the first chain of a string are broken into pieces:

  $s(pecifier), h(ead), c(omplement)$

Head-merge operation

$$
e, s, e :: f<=\gamma \quad t_s, t_h, t_c \cdot f, \alpha_1, \ldots, \alpha_k \\
es, s, e :: f<=\gamma \quad t_s, t_h, t_c :: \gamma, \alpha_1, \ldots, \alpha_k \\
\epsilon, s, e :: f<=\gamma \quad t_s, t_h, t_c :: \gamma, \alpha_1, \ldots, \alpha_k \quad r_1right$$

Hop Merge-operation

$$
e, s, e :: f=>\gamma \quad t_s, t_h, t_c \cdot f, \alpha_1, \ldots, \alpha_k \\
e, s, e :: f=>\gamma \quad t_s, t_h, t_c :: \gamma, \alpha_1, \ldots, \alpha_k \\
\epsilon, s, e :: f=>\gamma \quad t_s, t_h, t_c :: \gamma, \alpha_1, \ldots, \alpha_k \quad r_1hopright$$
Affix hopping in MG

Lexicon:

<table>
<thead>
<tr>
<th>Lexical:</th>
<th>Functional:</th>
</tr>
</thead>
<tbody>
<tr>
<td>lavinia :: d -k</td>
<td>ε :: =&gt;vp =d T</td>
</tr>
<tr>
<td>titus :: d -k</td>
<td>-s :: T=&gt; +k ip</td>
</tr>
<tr>
<td>praise :: =d +k vp</td>
<td></td>
</tr>
</tbody>
</table>

Derivation

Using affix hopping and head merge we can now derive correctly:

Titus praise-s Lavinia

* praise-s Titus Lavinia
5. Type-logical grammar

Binary connectives: \( \mathcal{F} ::= \mathcal{A} | \mathcal{F}/i \mathcal{F} | \mathcal{F} \cdot i \mathcal{F} | \mathcal{F}\setminus i \mathcal{F} \)

Unary Connectives: \( \mathcal{F} ::= \Box_i \mathcal{F} | \Diamond_i \mathcal{F} \)

Feature correspondence

<table>
<thead>
<tr>
<th>Kind of feature</th>
<th>MG</th>
<th>MMCG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic categories</td>
<td>c</td>
<td>( c^* )</td>
</tr>
<tr>
<td>Selector features</td>
<td>=c</td>
<td>( c^\circ ): for example ((c\setminus&gt;-)^<em>), ((-/&lt;c)^</em>)</td>
</tr>
<tr>
<td>Licensee features</td>
<td>([-f])</td>
<td>( \Box_f ) on a formula with polarity ( ^* )</td>
</tr>
<tr>
<td>Licensor features</td>
<td>([+f])</td>
<td>( \Diamond_f ) on a formula with polarity ( ^\circ )</td>
</tr>
</tbody>
</table>

**lexical categories** build up using the binary connective as *selector features* and unary connectives as *licensee* features

**functional categories** not a separate lexical item as in \( \mathcal{MG} \), but behavior as trigger of movement is taken over by the unary operators: \( \Diamond \) and \( \Box \).
Mapping of Operations

Merge as Modus Ponens

Leftheaded Merge:
\[
 s :: = \gamma \ t \cdot f, \alpha_1, \ldots, \alpha_k \\
 st : \gamma, \alpha_1, \ldots, \alpha_k \\
 t_1 \vdash B / i A \\
 t_2 \vdash A \\
 t_1 \circ_i t_2 \vdash B \\
[/E]
\]

Rightheaded Merge:
\[
 s :: = \gamma, \alpha_1, \ldots, \alpha_k \ t \cdot f, t_1, \ldots, t_l \\
 ts : \gamma, \alpha_1, \ldots, \alpha_k, t_1, \ldots, t_l \\
 t_1 \vdash A \\
 t_2 \vdash A \setminus_i B \\
 t_1 \circ_i t_2 \vdash B \\
[E]
\]

• Selector features \([= f]\) Merge with category features

• Selector features in Categorial Grammar: \(\setminus/A, A\setminus\)
Structural postulates for Affix hopping

\[ \diamond (B \bullet_2 A) \bullet_1 C) \rightarrow \diamond_a A \bullet_2 (B \bullet_1 C) \quad [AH1] \]
\[ \diamond (A \bullet_1 B) \rightarrow \diamond_a A \bullet_1 B \quad [AH2] \]

- [AH1] = Affix hopping to the next lower head
- [AH2] = Moving the agreement feature out of the structure

Lexicon

<table>
<thead>
<tr>
<th>TLG:</th>
<th>MG:</th>
</tr>
</thead>
<tbody>
<tr>
<td>lavinia ⊢ (vp/1d)_1vp</td>
<td>lavinia :: d -k</td>
</tr>
<tr>
<td>titus ⊢ ip/1_a(d_1ip)</td>
<td>titus :: d -k</td>
</tr>
<tr>
<td>praise ⊢ vp/1d</td>
<td>praise :: =d vp -v</td>
</tr>
<tr>
<td>s ⊢ □_a((d_1ip)/2vp)</td>
<td>-s :: =pred +v +k i</td>
</tr>
<tr>
<td>ϵ :: =vp +k =d pred</td>
<td></td>
</tr>
</tbody>
</table>
Derivation of affix hopping

\[ \begin{align*}
\text{titus} & \vdash \square_d^1 ((d\backslash_1 ip)_{/2 vp}) \\
& \vdash \square^1 (d\backslash_1 ip)_{/2 vp} \quad [\square^1 E] \\
\text{praise} & \vdash vp_{/1d} \quad \text{lavinia} \vdash (vp_{/1d})_{\backslash 1 vp} \quad [\backslash E] \\
\langle s \rangle^a & \vdash (d\backslash_1 ip)_{/2 vp} \quad [\backslash E] \\
\text{praise} \circ_1 \text{lavinia} & \vdash vp \quad [\backslash E] \\
\langle s \rangle^a & \vdash \text{praise} \circ_1 \text{lavinia} \vdash d\backslash_1 ip \quad [\backslash E] \\
\text{titus} \vdash ip_{/1d} \quad [\backslash E] \\
\text{titus} \circ_1 ((\text{praise} \circ_2 s) \circ_1 \text{lavinia}) & \vdash ip \quad [AH1] \\
\text{lavinia} \circ_1 \text{praise} & \vdash d\backslash_1 ip \quad [AH2] \\
\langle (\text{praise} \circ_2 s) \circ_1 \text{lavinia} \rangle^a & \vdash d\backslash_1 ip \quad [\langle I \rangle 1] \\
\text{titus} \circ_1 ((\text{praise} \circ_2 s) \circ_1 \text{lavinia}) & \vdash ip \quad [\backslash^1 I] \\
\langle (\text{praise} \circ_2 s) \circ_1 \text{lavinia} \rangle^a & \vdash d\backslash_1 ip \quad [\backslash^1 I] \\
\end{align*} \]
6. Recapitulation

Plan of action

- analyze empirical data in syntactic terms
- implement data in both formal frameworks
- compare the two analysis

**MG** uses abstract lexical entries and needs two extra operations to account for affix hopping:

\[
\begin{align*}
\epsilon, s, \epsilon & : f \rightarrow \gamma \\
\epsilon, \epsilon, t_s t_h t_c & : \gamma, \alpha_1, \ldots, \alpha_k \\
\end{align*}
\]

\[
\epsilon, \epsilon, t_s t_h s t_c & : \gamma, \alpha_1, \ldots, \alpha_k \\
\]

\[
\text{r1hopright}
\]

**TLG** needs higher order types and structural operations to account for affix hopping:

\[
\begin{align*}
\diamond (B \bullet_2 C) \bullet_1 C & \rightarrow \diamond_a A \bullet_2 (B \bullet_1 A) \quad [AH1] \\
\diamond (A \bullet_1 B) & \rightarrow \diamond_a A \bullet_1 B \quad [AH2]
\end{align*}
\]

- make generalisations