Evaluating Evaluation Measures for Minimalist Parsing

Thomas Graf
Stony Brook University
mail@thomasgraf.net
http://thomasgraf.net

Bradley Marcinek
Stony Brook University
bradley.marcinek@stonybrook.edu

CMCL 2014
July 26, 2014

Topic of This Talk

- MG parser could yield processing predictions for syntactic proposals that differ on abstract level (e.g. head movement VS remnant movement)
- **But:** need a linking hypothesis/difficulty metric
- Is there a **simple metric that is good enough** to distinguish syntactic analyses?

Results

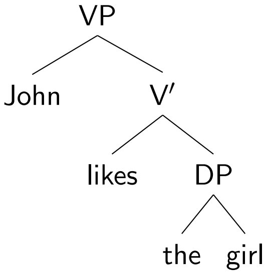
- Counting number of memorized items insufficient
- **Better:** max time pronounced lexical items stay in memory

Outline

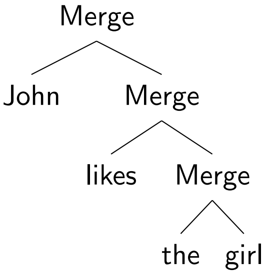
Minimalist Grammars (MGs)

- mildly context-sensitive formalization of Minimalist syntax (Chomsky 1995; Stabler 1997)
  - generates **all context-free** languages
  - generates **some context-sensitive** languages
- grammar is fully specified by lexicon
- lexicon = finite set of feature-annotated words
- features trigger structure-building operations Merge and Move
- **Merge:** combine two trees into a new tree
- **Move:** move a subtree of tree t to the left of the root of t

Sketch of a Simple Merge Derivation

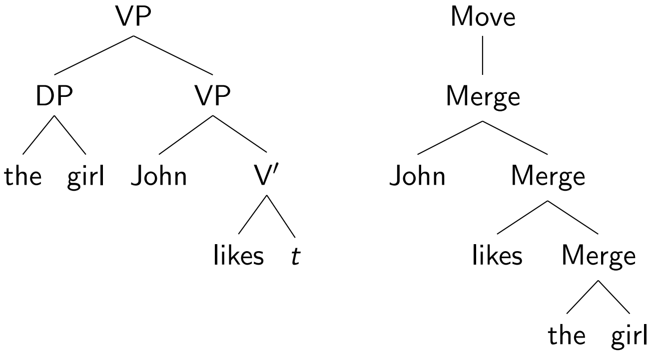


Phrase structure tree



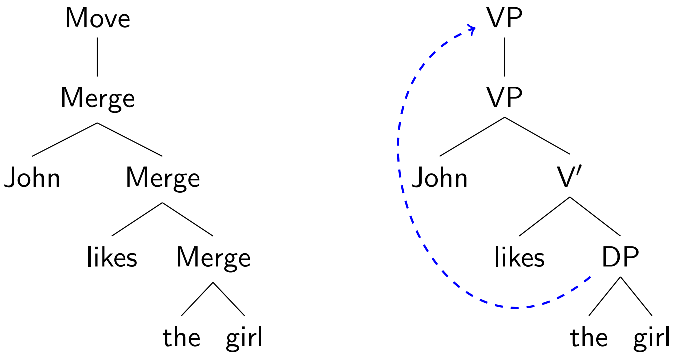
Derivation tree

Sketch of a Derivation with Move



Phrase structure tree **Derivation tree**

A More Readable Variant of Derivation Trees



Derivation tree　　**“Enhanced” derivation tree**

Why Derivation Trees Matter

- All information encoded in derivation trees
- Derivation trees automatically translated into corresponding phrase structure trees

Phrase structure trees are redundant!
Derivation tree = full description of sentence structure

- **Crucial:** derivation trees are **context-free**.
- Hence we can build on standard parsing techniques for CFGs.

Why Derivation Trees Matter

- All information encoded in derivation trees
- Derivation trees automatically translated into corresponding phrase structure trees

Phrase structure trees are redundant!
Derivation tree = full description of sentence structure

- **Crucial:** derivation trees are **context-free**.
- Hence we can build on standard parsing techniques for CFGs.

Why Derivation Trees Matter

- All information encoded in derivation trees
- Derivation trees automatically translated into corresponding phrase structure trees

Phrase structure trees are redundant!
Derivation tree = full description of sentence structure

- **Crucial:** derivation trees are **context-free**.
- Hence we can build on standard parsing techniques for CFGs.

Incremental Top-Down Parser for CFGs

Stabler (2011, 2012) presents an MG parser similar to top-down CFG parsers.

Incremental Top-Down CFG Parser

- Conjecture start symbol
- If the leftmost symbol is

  non-terminal apply a matching rewrite rule

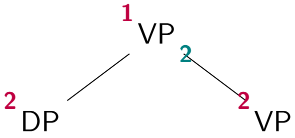
  terminal scan first unscanned word of input
- Stop if
  - all non-terminals have been expanded, and
  - all terminals have triggered a scan step, and
  - all words have been scanned
- Return derivation tree

Example Parse of *The girl, John likes*

1 VP

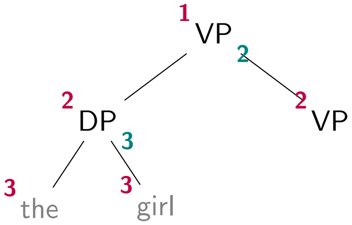
1. Start with VP
2. VP → DP VP
3. DP → the girl
4. Scan *the*
5. Scan *girl*
6. VP → John V′
7. Scan *John*
8. V′ → likes *t*
9. Scan *likes*
10. Scan *t* (= empty string)

Example Parse of *The girl, John likes*



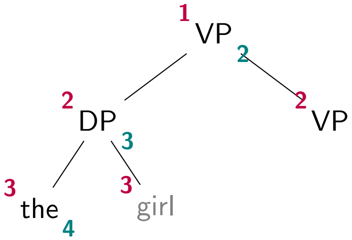
1. Start with VP
2. VP $\rightarrow$ DP VP
3. DP $\rightarrow$ the girl
4. Scan *the*
5. Scan *girl*
6. VP $\rightarrow$ John V′
7. Scan *John*
8. V′ $\rightarrow$ likes *t*
9. Scan *likes*
10. Scan *t* (= empty string)

Example Parse of *The girl, John likes*



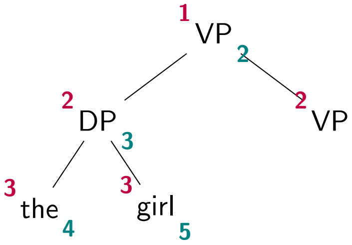
1. Start with VP
2. VP → DP VP
3. DP → the girl
4. Scan *the*
5. Scan *girl*
6. VP → John V′
7. Scan *John*
8. V′ → likes *t*
9. Scan *likes*
10. Scan *t* (= empty string)

Example Parse of *The girl, John likes*



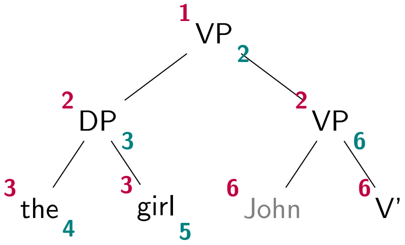
1. Start with VP
2. VP → DP VP
3. DP → the girl
4. Scan *the*
5. Scan *girl*
6. VP → John V′
7. Scan *John*
8. V′ → likes *t*
9. Scan *likes*
10. Scan *t* (= empty string)

Example Parse of *The girl, John likes*



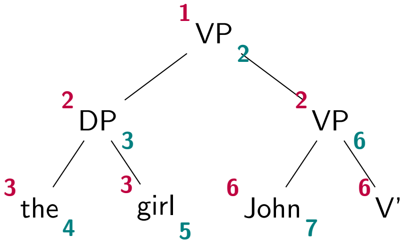
1. Start with VP
2. VP → DP VP
3. DP → the girl
4. Scan *the*
5. Scan *girl*
6. VP → John V′
7. Scan *John*
8. V′ → likes *t*
9. Scan *likes*
10. Scan *t* (= empty string)

Example Parse of *The girl, John likes*



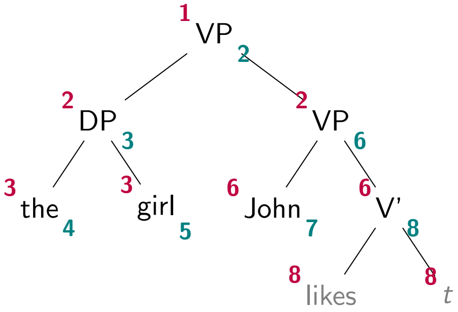
1. Start with VP
2. VP → DP VP
3. DP → the girl
4. Scan *the*
5. Scan *girl*
6. VP → John V′
7. Scan *John*
8. V′ → likes *t*
9. Scan *likes*
10. Scan *t* (= empty string)

Example Parse of *The girl, John likes*



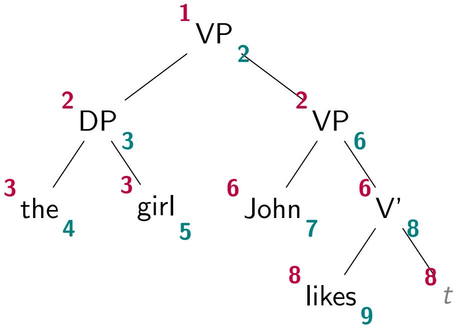
1. Start with VP
2. VP → DP VP
3. DP → the girl
4. Scan *the*
5. Scan *girl*
6. VP → John V′
7. Scan *John*
8. V′ → likes *t*
9. Scan *likes*
10. Scan *t* (= empty string)

Example Parse of *The girl, John likes*



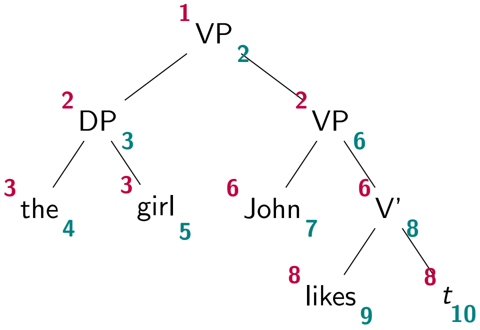
1. Start with VP
2. VP → DP VP
3. DP → the girl
4. Scan *the*
5. Scan *girl*
6. VP → John V′
7. Scan *John*
8. V′ → likes *t*
9. Scan *likes*
10. Scan *t* (= empty string)

Example Parse of *The girl, John likes*



1. Start with VP
2. VP $\rightarrow$ DP VP
3. DP $\rightarrow$ the girl
4. Scan *the*
5. Scan *girl*
6. VP $\rightarrow$ John V′
7. Scan *John*
8. V′ $\rightarrow$ likes *t*
9. Scan *likes*
10. Scan *t* (= empty string)

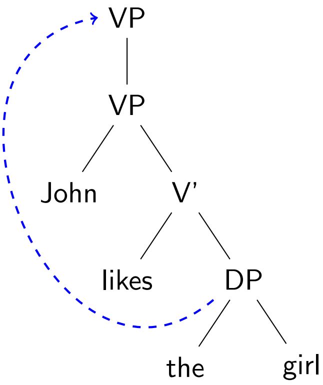
Example Parse of *The girl, John likes*



1. Start with VP
2. VP → DP VP
3. DP → the girl
4. Scan *the*
5. Scan *girl*
6. VP → John V′
7. Scan *John*
8. V′ → likes *t*
9. Scan *likes*
10. Scan *t* (= empty string)

The Problem With Derivation Trees

Derivation trees do not match string order
$\Rightarrow$ left-most terminal $\neq$ left-most word

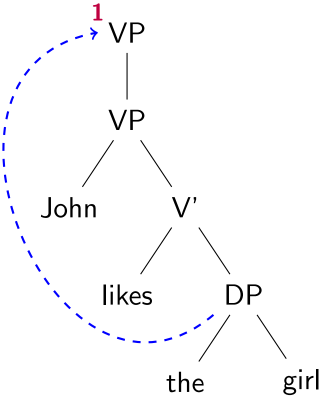


1. Start with Move
2. Move $\Rightarrow$ Merge
3. Merge $\Rightarrow$ John Merge
4. Scan *John*

Failure!

The Problem With Derivation Trees

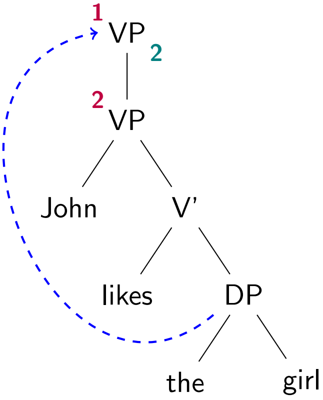
Derivation trees do not match string order
$\Rightarrow$ left-most terminal $\neq$ left-most word



1. Start with Move
2. Move $\Rightarrow$ Merge
3. Merge $\Rightarrow$ John Merge
4. Scan *John*
   Failure!

The Problem With Derivation Trees

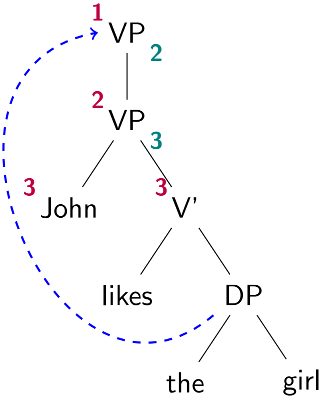
Derivation trees do not match string order
$\Rightarrow$ left-most terminal $\neq$ left-most word



1. Start with Move
2. Move $\Rightarrow$ Merge
3. Merge $\Rightarrow$ John Merge
4. Scan *John*
   Failure!

The Problem With Derivation Trees

Derivation trees do not match string order
⇒ left-most terminal ≠ left-most word

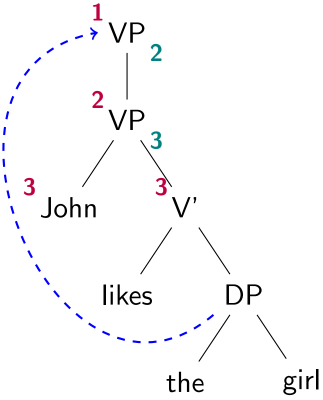


1. Start with Move
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Scan *John*

Failure!

The Problem With Derivation Trees

Derivation trees do not match string order
$\Rightarrow$ left-most terminal $\neq$ left-most word

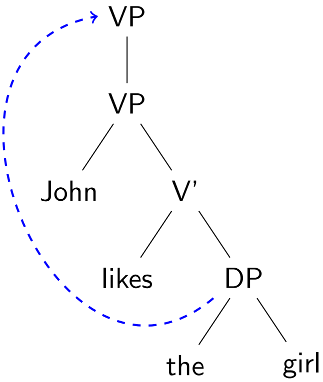


1. Start with Move
2. Move $\Rightarrow$ Merge
3. Merge $\Rightarrow$ John Merge
4. Scan *John*

Failure!

Derivation Trees Require Delayed Scanning

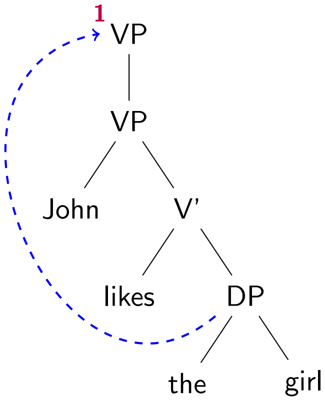
Steps must be **delayed** until we have found the leftmost word!
⇒ symbols crossed by mover must be kept in memory



1. Conjecture top-Mover
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Delay Scan *John*
   Merge ⇒ likes Merge
5. Delay Scan *likes*
   Merge ⇒ the[top] girl
6. **Mover found!**
   Scan *the*
7. Scan *girl*
8. Scan *John*
9. Scan *likes*

Derivation Trees Require Delayed Scanning

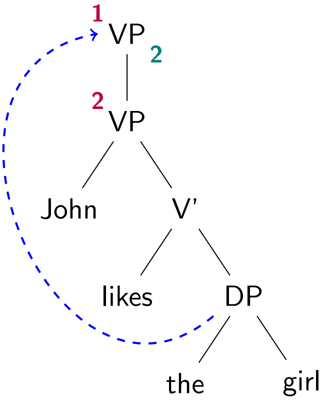
Steps must be **delayed** until we have found the leftmost word!
⇒ symbols crossed by mover must be kept in memory



1. Conjecture top-Mover
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Delay Scan *John*
   Merge ⇒ likes Merge
5. Delay Scan *likes*
   Merge ⇒ the[top] girl
6. **Mover found!**
   Scan *the*
7. Scan *girl*
8. Scan *John*
9. Scan *likes*

Derivation Trees Require Delayed Scanning

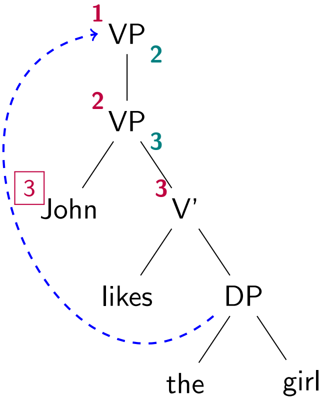
Steps must be **delayed** until we have found the leftmost word!
⇒ symbols crossed by mover must be kept in memory



1. Conjecture top-Mover
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Delay Scan *John*
   Merge ⇒ likes Merge
5. Delay Scan *likes*
   Merge ⇒ the[top] girl
6. **Mover found!**
   Scan *the*
7. Scan *girl*
8. Scan *John*
9. Scan *likes*

Derivation Trees Require Delayed Scanning

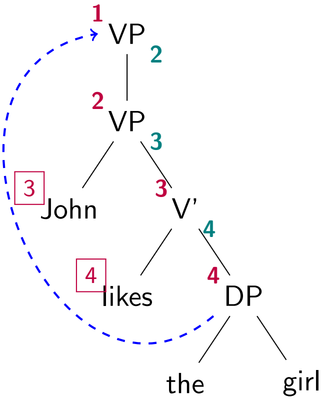
Steps must be **delayed** until we have found the leftmost word!
⇒ symbols crossed by mover must be kept in memory



1. Conjecture top-Mover
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Delay Scan *John*
   Merge ⇒ likes Merge
5. Delay Scan *likes*
   Merge ⇒ the[top] girl
6. **Mover found!**
   Scan *the*
7. Scan *girl*
8. Scan *John*
9. Scan *likes*

Derivation Trees Require Delayed Scanning

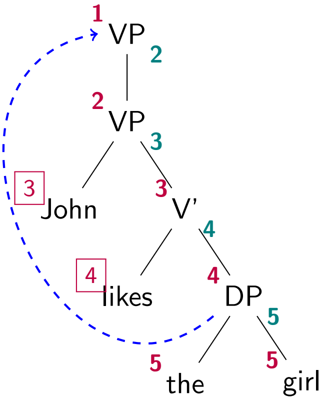
Steps must be **delayed** until we have found the leftmost word!
⇒ symbols crossed by mover must be kept in memory



1. Conjecture top-Mover
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Delay Scan *John*
   Merge ⇒ likes Merge
5. Delay Scan *likes*
   Merge ⇒ the[top] girl
6. **Mover found!**
   Scan *the*
7. Scan *girl*
8. Scan *John*
9. Scan *likes*

Derivation Trees Require Delayed Scanning

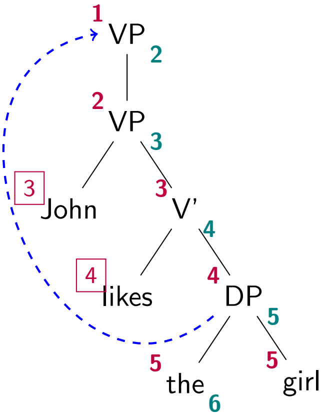
Steps must be **delayed** until we have found the leftmost word!
⇒ symbols crossed by mover must be kept in memory



1. Conjecture top-Mover
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Delay Scan *John*
   Merge ⇒ likes Merge
5. Delay Scan *likes*
   Merge ⇒ the[top] girl
6. **Mover found!**
   Scan *the*
7. Scan *girl*
8. Scan *John*
9. Scan *likes*

Derivation Trees Require Delayed Scanning

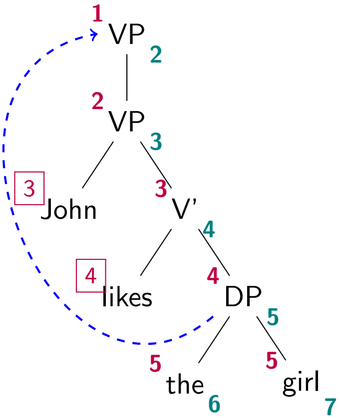
Steps must be **delayed** until we have found the leftmost word!
⇒ symbols crossed by mover must be kept in memory



1. Conjecture top-Mover
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Delay Scan *John*
   Merge ⇒ likes Merge
5. Delay Scan *likes*
   Merge ⇒ the[top] girl
6. **Mover found!**
   Scan *the*
7. Scan *girl*
8. Scan *John*
9. Scan *likes*

Derivation Trees Require Delayed Scanning

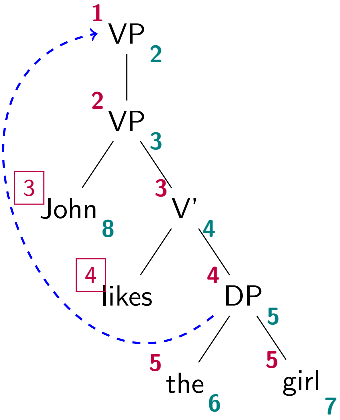
Steps must be **delayed** until we have found the leftmost word!
⇒ symbols crossed by mover must be kept in memory



1. Conjecture top-Mover
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Delay Scan *John*
   Merge ⇒ likes Merge
5. Delay Scan *likes*
   Merge ⇒ the[top] girl
6. **Mover found!**
   Scan *the*
7. Scan *girl*
8. Scan *John*
9. Scan *likes*

Derivation Trees Require Delayed Scanning

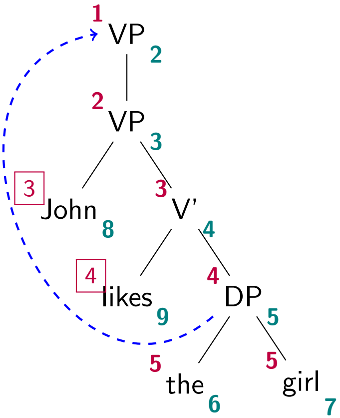
Steps must be **delayed** until we have found the leftmost word!
⇒ symbols crossed by mover must be kept in memory



1. Conjecture top-Mover
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Delay Scan *John*
   Merge ⇒ likes Merge
5. Delay Scan *likes*
   Merge ⇒ the[top] girl
6. **Mover found!**
   Scan *the*
7. Scan *girl*
8. Scan *John*
9. Scan *likes*

Derivation Trees Require Delayed Scanning

Steps must be **delayed** until we have found the leftmost word!
⇒ symbols crossed by mover must be kept in memory



1. Conjecture top-Mover
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Delay Scan *John*
   Merge ⇒ likes Merge
5. Delay Scan *likes*
   Merge ⇒ the[top] girl
6. **Mover found!**
   Scan *the*
7. Scan *girl*
8. Scan *John*
9. Scan *likes*

Tenure as Linking Hypothesis for Processing

Kobele et al. (2012) link parsing behavior to processing difficulty:

Tenure Time a symbol stays in memory
= Subscript−Superscript

Max Greatest tenure among all nodes in derivation

Max Linking Hypothesis

What Matters for Processing Difficulty

- **Max** value of the correct derivation

What Doesn't Matter

- Size of search space/number of conjectured derivations
- Number of items kept in memory
- Type of item memorized (e.g. R-expression vs anaphor)
- lexical frequency/probabilities

Tenure as Linking Hypothesis for Processing

Kobele et al. (2012) link parsing behavior to processing difficulty:

Tenure Time a symbol stays in memory
= Subscript−Superscript

Max Greatest tenure among all nodes in derivation

Max Linking Hypothesis

What Matters for Processing Difficulty

- **Max** value of the correct derivation

What Doesn't Matter

- Size of search space/number of conjectured derivations
- Number of items kept in memory
- Type of item memorized (e.g. R-expression vs anaphor)
- lexical frequency/probabilities

Why this is Attractive

- The MG parser is very simple.
- The linking hypothesis is very simple.
- Nonetheless we get some interesting predictions:
  - Crossing dependencies easier than nested dependencies (Bach et al. 1986)
  - Results can vary with syntactic analysis, for instance head movement VS remnant movement
    $\Rightarrow$ **processing data differentiates abstract analyses**

The Big Promise

- extremely simple processing model (definitely too simple)
  - no number crunching
  - pen and paper is enough
- yet good enough to distinguish between competing proposals from the Minimalist literature

Why this is Attractive

- The MG parser is very simple.
- The linking hypothesis is very simple.
- Nonetheless we get some interesting predictions:
  - Crossing dependencies easier than nested dependencies (Bach et al. 1986)
  - Results can vary with syntactic analysis,
    for instance head movement VS remnant movement
    $\Rightarrow$ **processing data differentiates abstract analyses**

The Big Promise

- extremely simple processing model (definitely too simple)
  - no number crunching
  - pen and paper is enough
- yet good enough to distinguish between competing proposals from the Minimalist literature

Too Good to be True?

Why should **Max** be the best metric?

- **MaxLex** — **Max** of lexical nodes
- **Box** — number of items kept in memory
  = number of boxed superscripts
- **BoxLex** — number of lexical items kept in memory
- **±Empty** — for each metric, another variant that does not count unpronounced nodes

Next Steps

- Pick phenomena that are most likely to be adequately explained by memory limitations
- Mark up correct derivation trees with indices
- See which metric gives best results across the board

Too Good to be True?

Why should **Max** be the best metric?

MaxLex **Max** of lexical nodes

Box number of items kept in memory
= number of boxed superscripts

BoxLex number of lexical items kept in memory

±Empty for each metric, another variant that does not count unpronounced nodes

Next Steps

- Pick phenomena that are most likely to be adequately explained by memory limitations
- Mark up correct derivation trees with indices
- See which metric gives best results across the board

SC/RC vs RC/SC

SC/RC vs RC/SC

A sentential complement (SC) containing a relative clause (RC) is easier to parse than an RC containing an SC.

(1) The fact $[_{SC}$ that the employee$_i$ $[_{RC}$ who the manager hired $t_i]$ stole office supplies$]$ worried the executive.

(2) The executive $[_{RC}$ who the fact $[_{SC}$ that the employee stole office supplies$]$ worried $t_i]$ hired the manager.

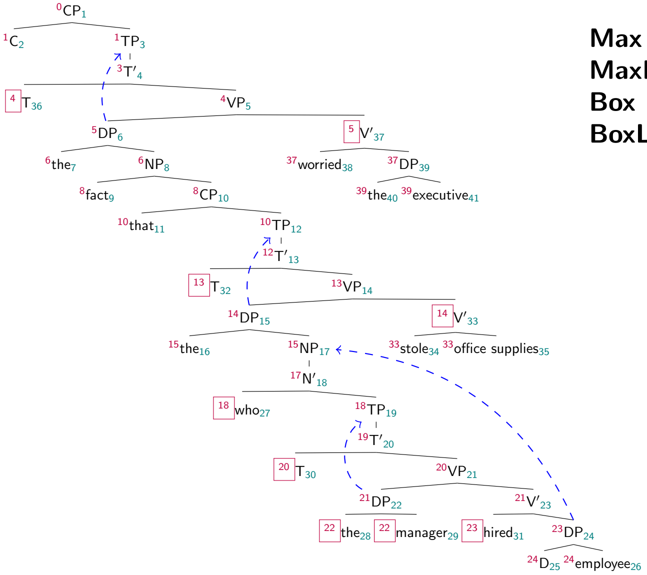
Syntactic Analysis

- Following Kobele et al. we use a **promotion analysis of RCs**.
  - Head noun is merged as argument DP of verb inside RCs
  - Head noun moves into Spec,CP of RC

$[_{\text{DP}}$ the $[_{\text{CP}}$ $[_{\text{DP}}$ ε employee$]$ $[_{\text{C'}}$ who the manager hired $t_{\text{DP}}]]$ $]$

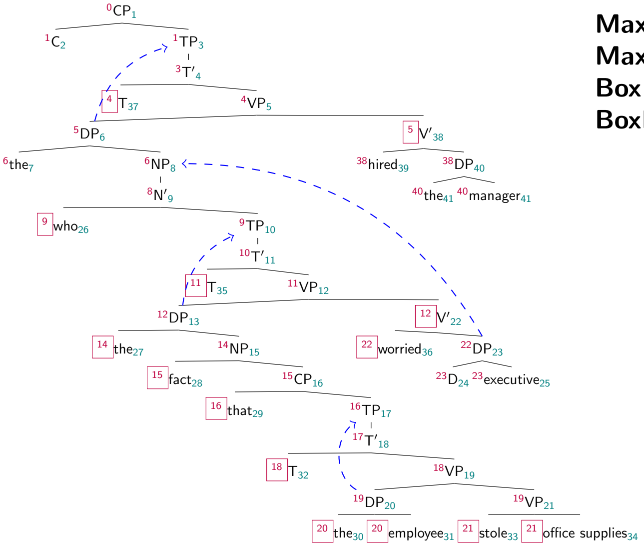
- **But**: Same results with other analyses as long as something moves from within RC to the left of *who*

SC/RC Derivation



Max	32/32
MaxLex	32/9
Box	9/6
BoxLex	7/4

RC/SC Derivation



Max	33/33
MaxLex	33/17
Box	14/11
BoxLex	12/9

Analysis

- Box metrics get the contrast.
  - **SC/RC**
    elements of SC preceding RC can be scanned right away, only RC delayed by movement of head noun
  - **RC/SC**
    both RC and SC delayed by movement of head noun
- Max metrics give mixed results.
  Max
  - highest value at matrix T-head due to size of subjects
  - both SC/RC and RC/SC yield big subjects
  - difference too small, a single adjective modifying *fact* can tip scale in favor of RC/SC

  MaxLex
  - if only pronounced words are considered, highest value at *who*
  - tenure of *who* increases with distance to head noun
  - RC/SC harder because of increased size of RC

Analysis

- Box metrics get the contrast.
  - **SC/RC**
    elements of SC preceding RC can be scanned right away, only RC delayed by movement of head noun
  - **RC/SC**
    both RC and SC delayed by movement of head noun
- Max metrics give mixed results.
  Max
  - highest value at matrix T-head due to size of subjects
  - both SC/RC and RC/SC yield big subjects
  - difference too small, a single adjective modifying *fact* can tip scale in favor of RC/SC

  MaxLex
  - if only pronounced words are considered, highest value at *who*
  - tenure of *who* increases with distance to head noun
  - RC/SC harder because of increased size of RC

Subject Gaps vs Object Gaps

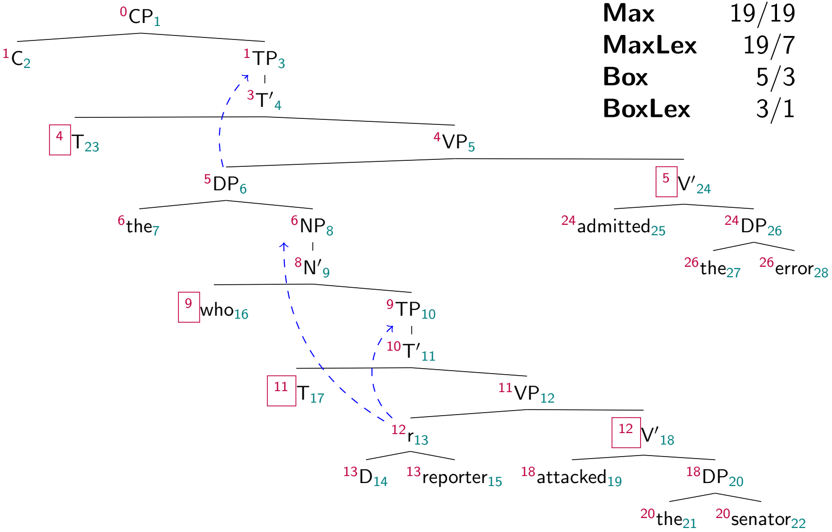
Subject Gaps vs Object Gaps

An RC containing a subject gap is easier to parse than an RC containing an object gap.

(3) The reporter$_i$ [$_{\text{CP}}$ who t_i attacked the senator] admitted the error.

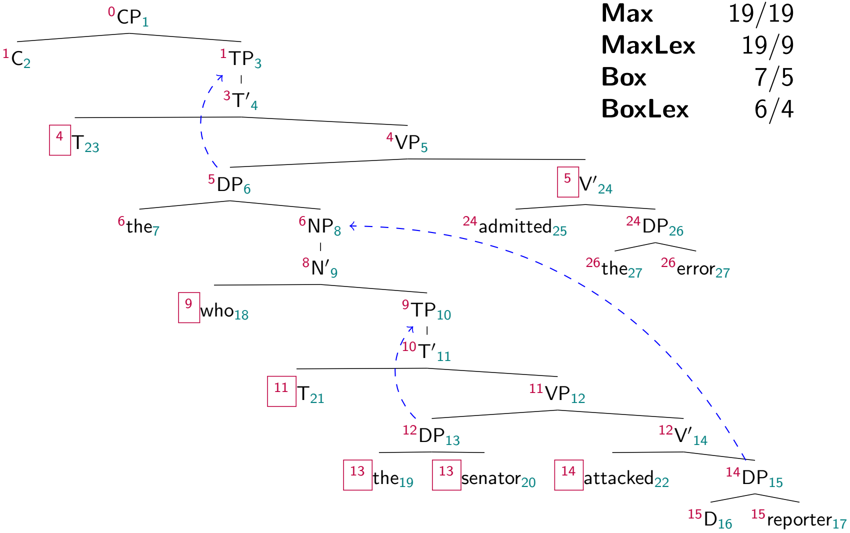
(4) The reporter$_i$ [$_{\text{CP}}$ who the senator attacked t_i] admitted the error.

Subject Gap Derivation



Max	19/19
MaxLex	19/7
Box	5/3
BoxLex	3/1

Object Gap Derivation



Max	19/19
MaxLex	19/9
Box	7/5
BoxLex	6/4

Analysis

- Box metrics get the contrast, again.
  - object gap leaves more material between landing site and mover
  - number of delayed scan steps increases with moved distance
- Max metrics give mixed results, again.

  Max
  - highest value at matrix T-head due to size of subjects
  - type of RC has no effect on size of subject
  - both derivations must have same maximum tenure

  MaxLex
  - if only pronounced words are considered, highest value at *who*
  - tenure of *who* increases with distance to head noun
  - object gap harder because of increased distance

Analysis

- Box metrics get the contrast, again.
  - object gap leaves more material between landing site and mover
  - number of delayed scan steps increases with moved distance
- Max metrics give mixed results, again.
  Max
  - highest value at matrix T-head due to size of subjects
  - type of RC has no effect on size of subject
  - both derivations must have same maximum tenure

  MaxLex
  - if only pronounced words are considered, highest value at *who*
  - tenure of *who* increases with distance to head noun
  - object gap harder because of increased distance

All Metrics are Insufficient

Box/BoxLex

- good results for relative clauses
- **Box**: increasing difficulty for all left embedding constructions
- **BoxLex**: constant difficulty for some left embedding
- **But**: do not capture difference between crossing and nested dependencies.

Max/MaxLex

- Only **MaxLex** restricted to overt material captures RC contrasts.
- Both capture difference between crossing and nesting.
- **Max**: increasing difficulty for all left embedding constructions
- **MaxLex**: constant difficulty for some left embedding

All Metrics are Insufficient

Box/BoxLex

- good results for relative clauses
- **Box**: increasing difficulty for all left embedding constructions
- **BoxLex**: constant difficulty for some left embedding
- **But**: do not capture difference between crossing and nested dependencies.

Max/MaxLex

- Only **MaxLex** restricted to overt material captures RC contrasts.
- Both capture difference between crossing and nesting.
- **Max**: increasing difficulty for all left embedding constructions
- **MaxLex**: constant difficulty for some left embedding

Next Step: Head-Final RCs

- Even in languages with head-final RCs, subject gaps are preferred.
- This is **not captured** by the metrics. At best we get a tie.
- **Further complication**
  Basque may have a preference for object gaps.
  (Carreiras et al. 2010)

Summary

- MG derivation trees allow for very simple top-down parsing
- **Idea:** test syntactic proposals by linking parser behavior to processing difficulty
- **Problem:** Is there a simple yet good enough metric?

Phenomenon	Max	MaxLex	Box	BoxLex
SC/RC vs RC/SC	$\sim$ / $\sim$	$\sim$ /yes	yes/yes	yes/yes
S-Gap vs O-Gap	no/no	no/yes	yes/yes	yes/yes
Nesting vs Crossing	yes/yes	yes/yes	no/no	no/no
Left embedding	no/no	no/$\sim$	no/no	no/$\sim$
Head-Initial RC	no/no	no/no	no/no	no/no

References

Bach, Emmon, Colin Brown, and William Marslen-Wilson. 1986. Crossed and nested dependencies in German and Dutch: A psycholinguistic study. *Language and Cognitive Processes* 1:249–262.

Carreiras, Manuel, Jon Andoni Duñabeitia, Marta Vergara, Irene de la Cruz-Pavía, and Itziar Laka. 2010. Subject relative clauses are not universally easier to process: Evidence from basque. *Cognition* 115:79–92.

Chomsky, Noam. 1995. *The minimalist program*. Cambridge, Mass.: MIT Press.

Kobele, Gregory M., Sabrina Gerth, and John T. Hale. 2012. Memory resource allocation in top-down minimalist parsing. In *Proceedings of Formal Grammar 2012*.

Stabler, Edward P. 1997. Derivational minimalism. In *Logical aspects of computational linguistics*, ed. Christian Retoré, volume 1328 of *Lecture Notes in Computer Science*, 68–95. Berlin: Springer.

Stabler, Edward P. 2011. Computational perspectives on minimalism. In *Oxford handbook of linguistic minimalism*, ed. Cedric Boeckx, 617–643. Oxford: Oxford University Press.

Stabler, Edward P. 2012. Bayesian, minimalist, incremental syntactic analysis. *Topics in Cognitive Science* 5:611–633.