Lecture 7

A Context-Free Top-Down Parser for Minimalist Grammars

Minimalist grammars are more powerful than CFGs, they generate mildly context-sensitive string languages, which are taken to be a good approximation of natural languages. They are also very malleable and can be used as a formal basis for the overwhelming majority of analyses in the generative literature. The question, however, is how MGs can be parsed. In general, more powerful grammar formalisms require more sophisticated parsing strategies — the higher expressivity comes at the cost of increased complexity. As we will see today, however, MGs can actually be represented in terms of CFGs, which makes it a lot easier to design MG parsers based on the CFG parsers we already know.

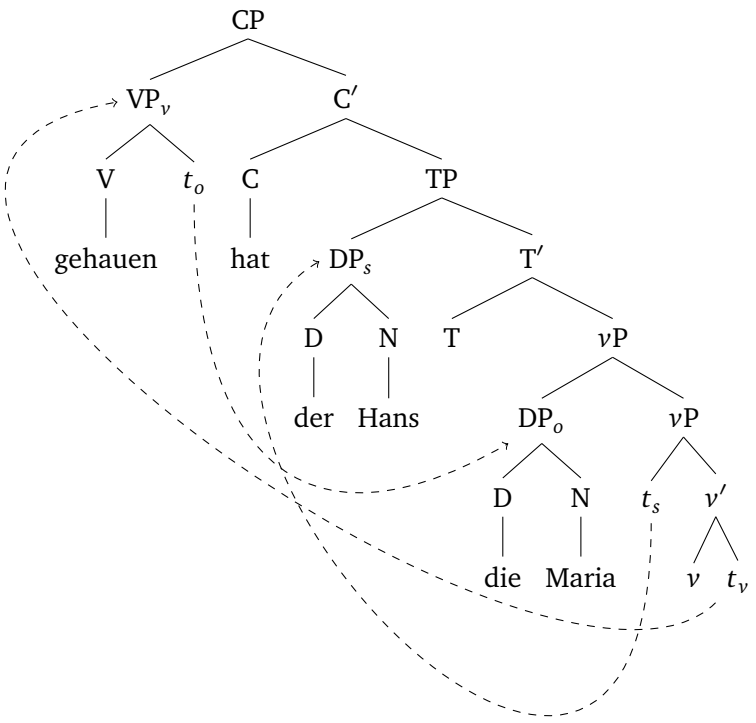
7.1 MG Derivations are Context-Free

The main difference between MG phrase structure trees and their corresponding derivation trees is that the latter only indicate when movement takes place, but no subtrees are actually being displaced. Surprisingly, this minor difference has a big effect on the complexity of these two data structures. Not all phrase structure trees can be generated by a context-free grammar — if this were the case, MGs would not be any more powerful than CFGs. MG derivation trees, however, are context-free.

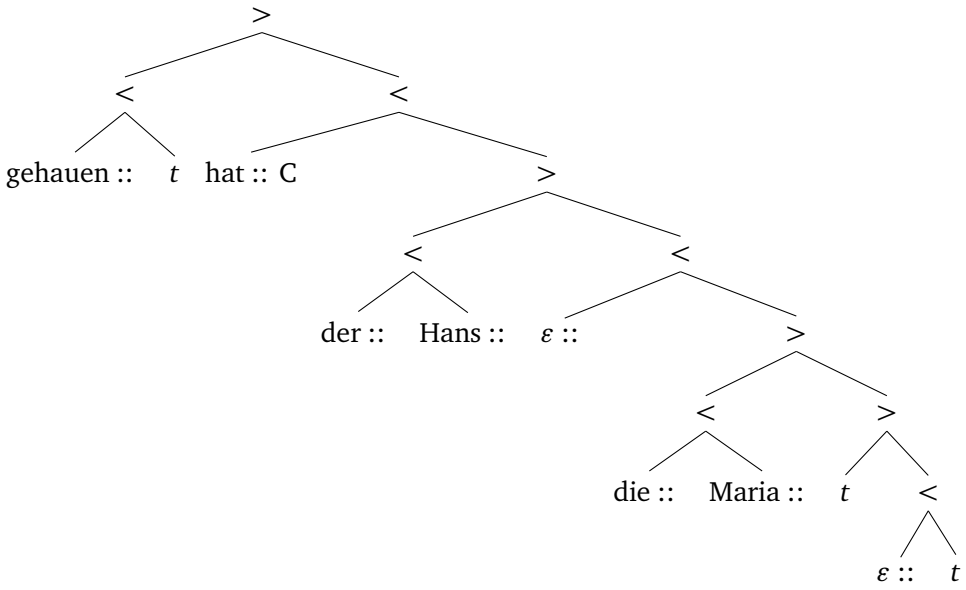
Consider the following sentence from Bavarian German, which displays topicalization of the V-head.

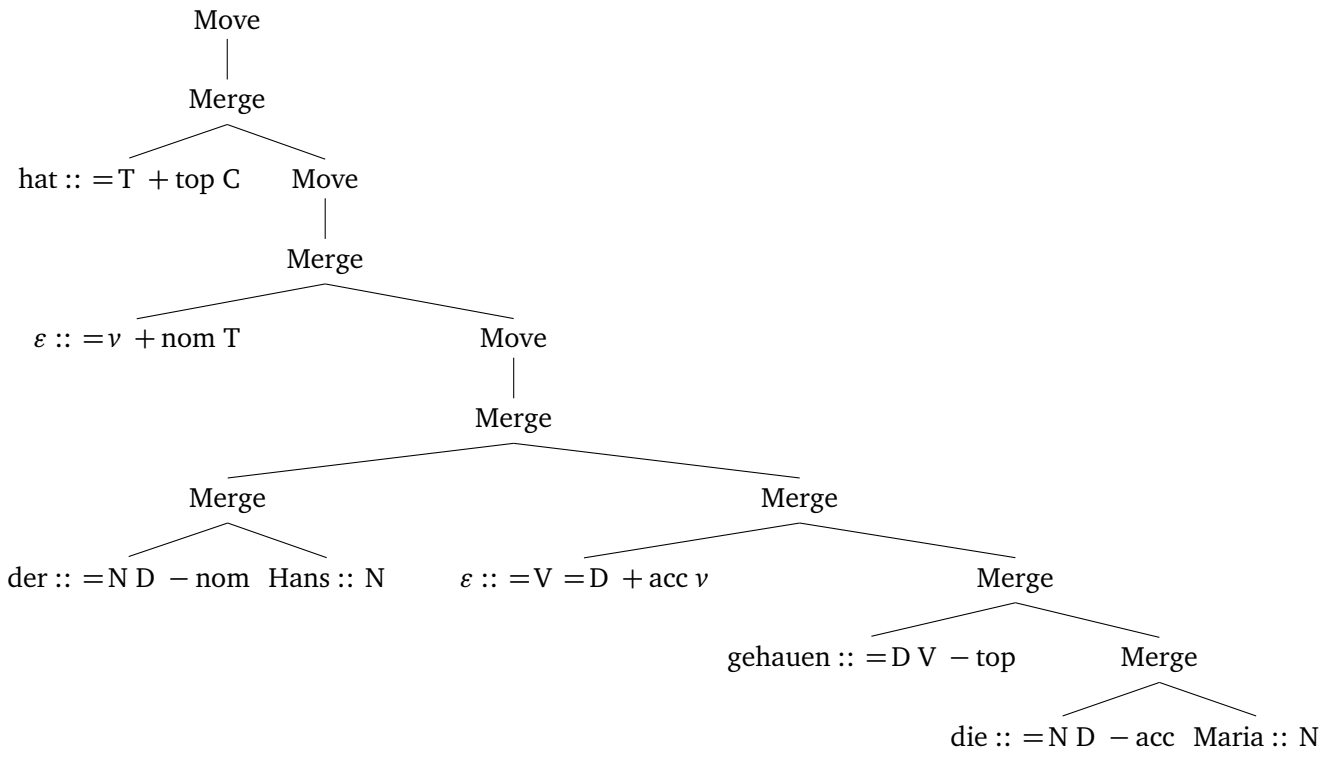
(1) Gehauen hat der Hans die Maria (, nicht geküsst).
beaten has the Hans the Maria (, not kissed).
'Hans beat Mary (he didn't kiss her).'

Since there are independent reasons to believe that the position before the finite verb can only be filled by phrases, the entire VP must have moved rather than the V-head itself. But since the object DP *die Maria* is part of the VP, it must have moved to a higher position outside the VP before the VP moved into the topic position. Given standard Minimalist assumptions — DP analysis of noun phrases, Larsonian shells, and the C-T-*v*-V clause spine — the phrase structure tree thus should be similar to the one below (for the sake of simplicity we ignore head movement of the auxiliary verb from *v* to T and C).

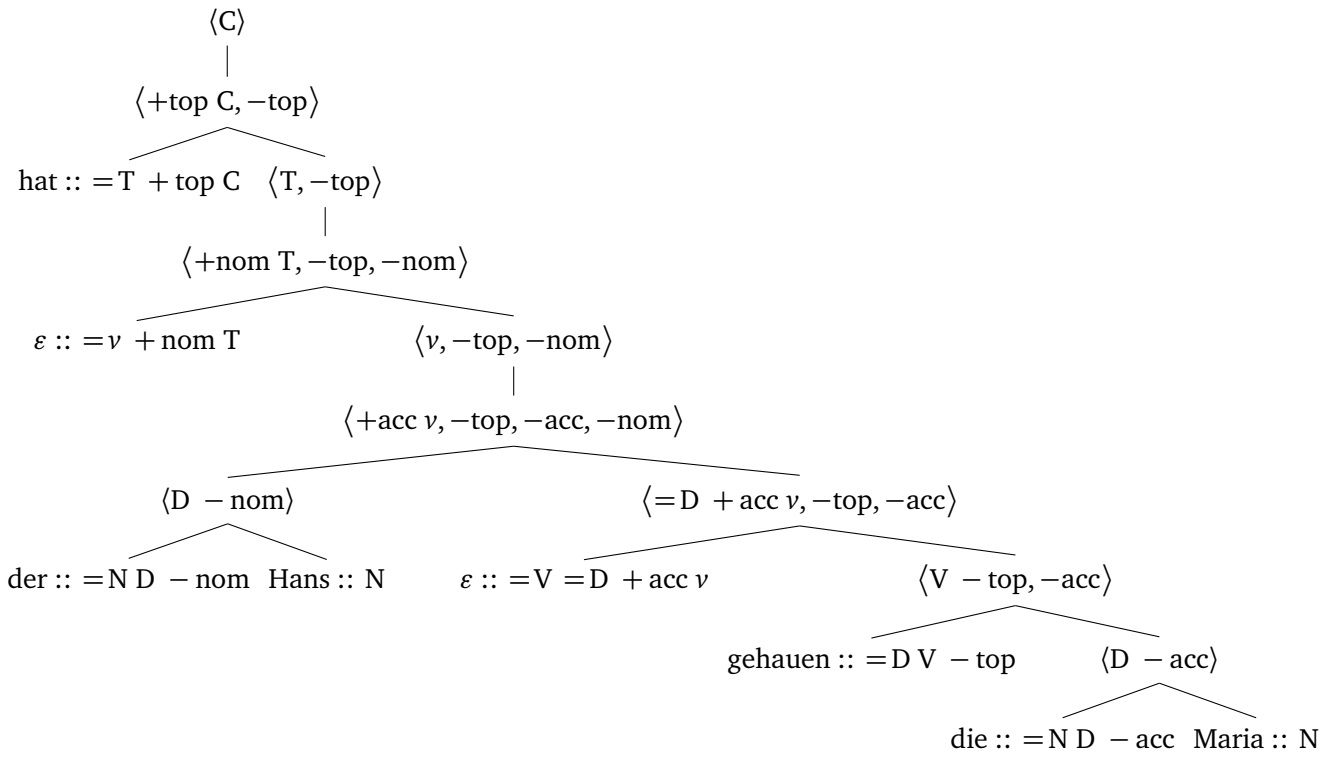


We can directly replicate this analysis with MGs.





At a quick glance it is actually hard to tell whether this derivation is well-formed. That's because the derivation tree does not directly keep track of which features are still unchecked at each point of the derivation. However, we can add this information by annotating every interior node with a tuple, each component of which is the list of unchecked features of some lexical item.



Notice how the feature-annotated derivation tree can easily be described by a small number of rewrite rules:

$$\begin{aligned}
\langle \text{C} \rangle &\rightarrow \langle +\text{top C}, -\text{top} \rangle \\
\langle +\text{top C}, -\text{top} \rangle &\rightarrow \langle =\text{T} \ +\text{top C} \rangle \ \langle \text{T}, -\text{top} \rangle \\
\langle \text{T}, -\text{top} \rangle &\rightarrow \langle +\text{nom T}, -\text{top}, -\text{nom} \rangle \\
\langle +\text{nom T}, -\text{top}, -\text{nom} \rangle &\rightarrow \langle =v \ +\text{nom T} \rangle \ \langle v, -\text{top}, -\text{nom} \rangle \\
\langle v, -\text{top}, -\text{nom} \rangle &\rightarrow \langle +\text{acc } v, -\text{top}, -\text{acc}, -\text{nom} \rangle \\
\langle +\text{acc } v, -\text{top}, -\text{acc}, -\text{nom} \rangle &\rightarrow \langle \text{D} \ -\text{nom} \rangle \ \langle =\text{D} \ +\text{acc } v, -\text{top}, -\text{acc} \rangle \\
\langle \text{D} \ -\text{nom} \rangle &\rightarrow \langle =\text{N D} \ -\text{nom} \rangle \ \langle \text{N} \rangle \\
\langle =\text{D} \ +\text{acc } v, -\text{top}, -\text{acc} \rangle &\rightarrow \langle =\text{V} \ =\text{D} \ +\text{acc } v \rangle \ \langle \text{V} \ -\text{top}, -\text{acc} \rangle \\
\langle \text{V} \ -\text{top}, -\text{acc} \rangle &\rightarrow \langle =\text{D V} \ -\text{top} \rangle \ \langle \text{D} \ -\text{acc} \rangle \\
\langle \text{D} \ -\text{acc} \rangle &\rightarrow \langle =\text{N D} \ -\text{acc} \rangle \ \langle \text{N} \rangle
\end{aligned}$$

$$\begin{aligned}
\langle =\text{T} \ +\text{top C} \rangle &\rightarrow \text{hat} \\
\langle =v \ +\text{nom T} \rangle &\rightarrow \varepsilon \\
\langle =\text{V} \ =\text{D} \ +\text{acc } v \rangle &\rightarrow \varepsilon \\
\langle =\text{D V} \ -\text{top} \rangle &\rightarrow \text{gehauen} \\
\langle =\text{N D} \ -\text{nom} \rangle &\rightarrow \text{der} \\
\langle =\text{N D} \ -\text{acc} \rangle &\rightarrow \text{die} \\
\langle \text{N} \rangle &\rightarrow \text{Hans} \mid \text{Maria}
\end{aligned}$$

While these rewrite rules are incredibly difficult to make sense of for humans, they generate the desired derivation tree (with the minor difference that for the sake of readability every lexical item has been split into two nodes, with the phonetic exponent as the daughter of the feature component). In fact, there is a fully automatic procedure for converting an MG G with lexicon *Lex* into a CFG C such that C generates all well-formed derivation trees of G, and only those (this result hinges on the SMC; see Kobele et al. 2007 and section 2.1.3 of Graf 2013).

Clearly every phrase structure tree is fully described by its derivation tree (or trees, if there are multiple ways of building and one and the same phrase structure tree), since the latter is a set of instructions for building the former. This means that the structure of a sentence is completely specified by assigning it a derivation tree. So even though an MG parser has to work for string languages that are not context-free, it still only has to assign each sentence a context-free structure – the derivation trees. As we will see next, this idea works fairly well, but there is one major complication: the string yield of a derivation tree does not correspond to the sentence being parsed because movement can change the order of words.

7.2 Top-Down Parser with Feature Annotated Derivations

7.2.1 MGs without Movement — A Naive Top-Down Parser

For an MG without movement — i.e. an MG where no lexical item has any licensee features — designing a parser is straight-forward. In this case, the order of the leaves in the derivation tree can be taken to mirror the order of the words in the input sentence. So if MG G is movement-free, we can translate it into a CFG that generates G's derivation trees and use any one of our familiar parsers for this CFG.

But let's see if we can design a parser that reflects the MG feature calculus more directly. If there are only Merge nodes in the derivation, every interior node has

exactly two daughters. In addition, all the features of the node must have been contributed by exactly one of its daughters, namely the one with the selector feature (take a minute to convince yourself that this is indeed the case!). Since the selector may be linearized either to the left or to the right, we need two distinct inference rules.

$$\textbf{Merge1} \qquad \frac{\langle \alpha \rangle}{\langle =\mathrm{F}\ \alpha \rangle \qquad \langle \mathrm{F} \rangle} \; F \text{ a feature name of } G$$

$$\textbf{Merge2} \qquad \frac{\langle \alpha \rangle}{\langle \mathrm{F} \rangle \qquad \langle =\mathrm{F}\ \alpha \rangle} \; F \text{ a feature name of } G$$

There is one minor problem with those rules, though, and that's that the selector should always be to the left of its first argument, and always to the right of its other arguments. Our rules do not capture this fact. For example, if α is C then the parser could linearize the complementizer to the left or to the right of TP. But an MG would only entertain the first option. The simplest fix is to annotate each selector feature with information about how the argument is linearized, and that's the solution we will use for now.

$$\textbf{Merge Right} \qquad \frac{\langle \alpha \rangle}{\langle =\mathrm{F}_r\ \alpha \rangle \qquad \langle \mathrm{F} \rangle} \; \mathrm{F} \text{ a feature name of } G$$

$$\textbf{Merge Left} \qquad \frac{\langle \alpha \rangle}{\langle \mathrm{F} \rangle \qquad \langle =\mathrm{F}_l\ \alpha \rangle} \; \mathrm{F} \text{ a feature name of } G$$

The only thing remaining, then, is a rule for the removal of lexical items, similar to the scan rule.

$$\textbf{LI} \qquad \frac{\langle \alpha \rangle}{\mathrm{w} :: \alpha \in Lex_G}$$

7.2.2 Recursive Descent Parser for Movement-Free MGs

The previous discussion is sufficient to outline the feature-based logic of an MG parser, but it does not serve as a parsing schema. Just like in recursive descent parsing, our parsing items need to keep track of all the non-terminal symbols and put the in the right linear order. This, however, is very simply to do. For a recursive-descent MG parser, the only axiom is $[0, \bullet \langle \mathrm{C} \rangle]$, and the only goal is $[n, \bullet]$.

$$\textbf{Merge Left} \qquad \frac{[i, \bullet \langle \alpha \rangle \beta, j]}{[i, \bullet \langle \mathrm{F} \rangle \langle =\mathrm{F}_l\ \alpha \rangle \beta, j]} \; \mathrm{F} \text{ a feature name of } G$$

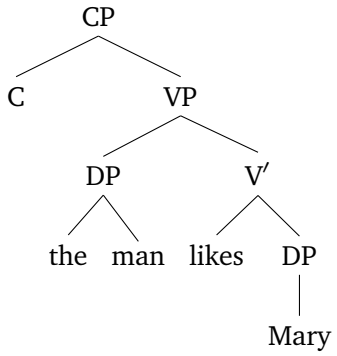
$$\textbf{Merge Right} \qquad \frac{[i, \bullet \langle \alpha \rangle \beta, j]}{[i, \bullet \langle =\mathrm{F}_r\ \alpha \rangle \langle \mathrm{F} \rangle \beta, j]} \; \mathrm{F} \text{ a feature name of } G$$

$$\textbf{LI (pronounced)} \qquad \frac{[i, \bullet \langle \alpha \rangle \beta, j]}{[i+1, \bullet \beta, j]} \; w_i :: \alpha \in Lex_G,\ w_i \neq \varepsilon$$

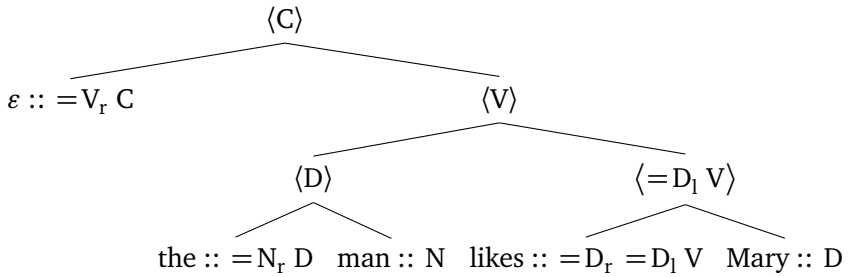
LI (empty) $\dfrac{[i, \bullet \langle \alpha \rangle \beta, j]}{[i, \bullet \beta, j]}\ w_i :: \alpha \in Lex_G,\ w_i = \varepsilon$

Example 7.1 MG Top-Down Parse of *The man likes Mary*

Suppose the sentence *The man likes Mary* has the structure below.



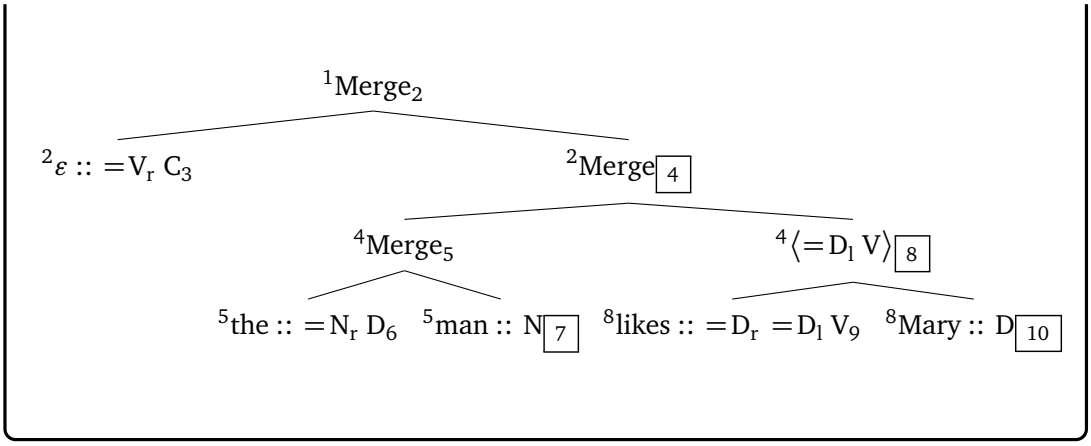
The feature-annotated Minimalist derivation tree for this sentence looks as follows.



The parse table directly reflects this structure.

parse item	inference rule
$[0, \bullet\langle C\rangle]$	axiom
$[0, \bullet\langle =V_r\ C\rangle\ \langle V\rangle]$	Merge Right
$[0, \bullet\langle V\rangle]$	LI (empty)
$[0, \bullet\langle D\rangle\ \langle =D_l\ V\rangle]$	Merge Left
$[0, \bullet\langle =N\ D\rangle\ \langle N\rangle\ \langle =D_l\ V\rangle]$	Merge Right
$[1, \bullet\langle N\rangle\ \langle =D_l\ V\rangle]$	LI (pronounced)
$[2, \bullet\langle =D_l\ V\rangle]$	LI (pronounced)
$[2, \bullet\langle =D_r =D_l\ V\rangle\ \langle D\rangle]$	Merge Right
$[3, \bullet\langle D\rangle]$	LI (pronounced)
$[4, \bullet]$	LI (pronounced)

We can annotate the final derivation tree as usual to indicate how it is built by the parser.



7.2.3 Adding Movement

In principle, movement inference rules aren't all that different from Merge inference rules: we have a given expression of features, and infer which expressions this could have been produced from. But there are three important differences:

- Move is a unary rule, so we go from $\langle \alpha \rangle$ to $\langle \beta \rangle$, where β differs minimally from α,
- a tuple can now contain multiple feature strings, all but one of which are sequences of licensee features,
- we need a way to incorporate how Move changes the linear order of lexical items.

Let's deal with points 1 and 2 first. In general, our feature tuples will now have the form $\langle \alpha, \beta_1, \ldots, \beta_n \rangle$, where each β_i is a string of licensee features. This means that we have to slightly change our Merge rules.

Merge Right $$\frac{\langle \alpha, \beta_1, \ldots, \beta_n \rangle}{\langle =\mathrm{F}_r\ \alpha, \gamma_1, \ldots, \gamma_k \rangle \qquad \langle \mathrm{F}\delta, \gamma_{k+1}, \ldots, \gamma_n \rangle}\ \text{F a feature name of } G$$

Merge Left $$\frac{\langle \alpha, \beta_1, \ldots, \beta_n \rangle}{\langle \mathrm{F}\delta, \gamma_1, \ldots, \gamma_k \rangle \qquad \langle =\mathrm{F}_l\ \alpha, \gamma_{k+1}, \ldots, \gamma_n \rangle}\ \text{F a feature name of } G$$

Notice that we now distribute the strings of licensee features over the two daughters of a Merge node in a non-deterministic fashion. That is to say, we require in both rules that

- for all $1 \leq i, j \leq n$, the first features of β_i and β_j are distinct,
- if $\delta = \varepsilon$, then each β_i is identical to exactly one γ_j, and *vice versa*,
- otherwise, there is some β_i such that $\beta_i = \delta$ and the previous condition holds for all β_j with $j \neq i$.

The logic of the movement rule is even simpler.

$$\textbf{Move} \qquad \frac{\langle \alpha, \beta_1, \ldots, \beta_i, \ldots, \beta_n \rangle}{\langle +\text{f}\ \alpha, \beta_1, \ldots, -\text{f}\ \beta_i, \ldots, \beta_n \rangle} \; -\text{f a licensee feature of } G$$

Notice that β_i may be empty. In that case, we are dealing with the final landing site of whichever lexical item hosts the $-$f feature. Strictly speaking, then, the item in the antecedent line of the rule would have two adjacent commas with nothing inbetween the two since β_i is empty. Our rules never create an item of this form, so the Move rule couldn't apply in this case. Hence we should actually distinguish two cases of movement.

$$\textbf{Move (intermediate)} \qquad \frac{\langle \alpha, \beta_1, \ldots, \beta_i, \ldots, \beta_n \rangle}{\langle +\text{f}\ \alpha, \beta_1, \ldots, -\text{f}\ \beta_i, \ldots, \beta_n \rangle} \; -\text{f a licensee feature of } G$$

$$\textbf{Move (final)} \qquad \frac{\langle \alpha, \beta_1, \ldots, \beta_{i-1}, \beta_{i+1} \ldots, \beta_n \rangle}{\langle +\text{f}\ \alpha, \beta_1, \ldots, \beta_{i-1}, -\text{f}, \beta_{i+1} \ldots, \beta_n \rangle} \; -\text{f a licensee feature of } G$$

These movement rules do not take care of the true challenge posed by movement however, and that's keeping track of the order of words in the sentence, which no longer corresponds to the order of words in the derivation tree. Assume that we conjecture a Move node for topicalization. Then we have to keep track of the fact that we expect to see an LI later on, which is handled by the presence of $-$f or $-$f β_i in the tuples. But we also need to know that once we have found this item, the phrase it is a head of should match a sequence of words at the position where we initially conjectured topicalization movement.

7.2.4 Keeping Track of Conjectured Movers

In order to keep track of which positions in the string are associated with movers, we use place holders of the form $[f]$, where f is a feature name. These place holders are introduced by Move rules and allow us to reorder the tuples in a parse item where possible. Given a substring ϕ of a parse item, $\phi_{/[f_1 \cdots f_n]}$ is the result of removing the placeholders $[f_1], \ldots, [f_n]$ from ϕ.

Once again the axiom is $[0, \bullet \langle \text{C} \rangle]$ and the goal is $[n, \bullet]$. We need 4 Merge rule, 2 Move rules, and a few ancillary rules. The Merge rules are parameterized according to linearization and whether the merged item is a mover.

$$\textbf{Merge Right} \qquad \frac{[i, \phi \bullet \langle \alpha, \beta_1, \ldots, \beta_n \rangle \psi]}{[i, \phi \bullet \langle =\text{F}_r\ \alpha, \gamma_1, \ldots, \gamma_k \rangle \langle \text{F}\delta, \gamma_{k+1}, \ldots, \gamma_n \rangle \psi]}$$

$$\textbf{Merge Left} \qquad \frac{[i, \phi \bullet \langle \alpha, \beta_1, \ldots, \beta_n \rangle \psi]}{[i, \phi \bullet \langle \text{F}\delta, \gamma_{k+1}, \ldots, \gamma_n \rangle \langle =\text{F}_l\ \alpha, \gamma_1, \ldots, \gamma_k \rangle \psi]}$$

Merge Mover Right

$$\frac{[i, \phi \bullet \langle \alpha, \beta_1, \ldots, f_1 \cdots f_m, \beta_n \rangle \psi]}{[i, \phi_{/[f_1 \cdots f_{m-1}]} \bullet \langle =\text{F}_r\ \alpha, \gamma_1, \ldots, \gamma_{k-1} \rangle [_{f_m} \langle \text{F}\ f_1 \cdots f_m, \gamma_{k+1}, \ldots, \gamma_n \rangle] \psi_{/[f_1 \cdots f_{m-1}]}]}$$

Merge Mover Left

$$\frac{[i, \phi \bullet \langle \alpha, \beta_1, \ldots, f_1 \cdots f_m, \ldots \beta_n \rangle \psi]}{[i, \phi_{/[f_1 \cdots f_{m-1}]} \bullet [_{f_m} \langle \mathrm{F} f_1 \cdots f_n, \gamma_{k+1}, \ldots, \gamma_n \rangle] \langle = \mathrm{F}_l \ \alpha, \gamma_1, \ldots, \gamma_{k-1} \rangle \psi_{/[f_1 \cdots f_{m-1}]}]}$$

The f-subscripted brackets around a tuple mark it as an f-mover, which is necessary to identify it with the right placeholder in ϕ or ψ. Notice also that we eliminate all placeholders for intermediate movement as soon as the mover is merged. Since the mover will never occur at an intermediate landing site — after all, it has to move to its final target position — those placeholders are string vacuous. However, we have to introduce them first to make sure that the feature checking requirements are satisfied: every non-final licensee feature needs a corresponding move node, which corresponds to a placeholder in the parse item.

Move (intermediate) $$\frac{[i, \phi \bullet \langle \alpha, \beta_1, \ldots, \beta_i, \ldots, \beta_n \rangle \psi]}{[i, \phi\,[\mathrm{f}] \bullet \langle +\mathrm{f}\ \alpha, \beta_1, \ldots, -\mathrm{f}\ \beta_i, \ldots, \beta_n \rangle \psi]}$$

Move (final) $$\frac{[i, \phi \bullet \langle \alpha, \beta_1, \ldots, \beta_{i-1}, \beta_{i+1} \ldots, \beta_n \rangle \psi]}{[i, \phi\,[\mathrm{f}] \bullet \langle +\mathrm{f}\ \alpha, \beta_1, \ldots, \beta_{i-1}, -\mathrm{f}, \beta_{i+1} \ldots, \beta_n \rangle \psi]}$$

The Move rules are almost exactly the same, except that Move (intermediate) extends an existing β_i, whereas Move (final) adds a completely new one.

The actual reordering of tuples in a parse item is handled by the rule displace, which takes a mover and puts it in the position of the placeholder.

Displace $$\frac{[i, \phi\,[f\,]\phi' \bullet [_f\,\alpha]\psi]}{[i, \phi \bullet \alpha\phi'\psi]}$$

The LI rule replaces a feature tuple by an LI with that feature specification, and the scan rule eliminates LIs from the parse items. Scanning an LI is allowed only if it is at the very left edge of the parse item.

LI $$\frac{[i, \phi \bullet \langle \alpha \rangle \psi, j]}{[i, \phi \bullet a\psi, j]}\ a :: \alpha \in Lex_G$$

Scan $$\frac{[i, \bullet a\beta, j]}{[k, \bullet\beta, j]}\ k = i \text{ if } a = \varepsilon \text{ and } i + 1 \text{ otherwise}$$

The shift rule moves • to the right if no other rule can be applied.

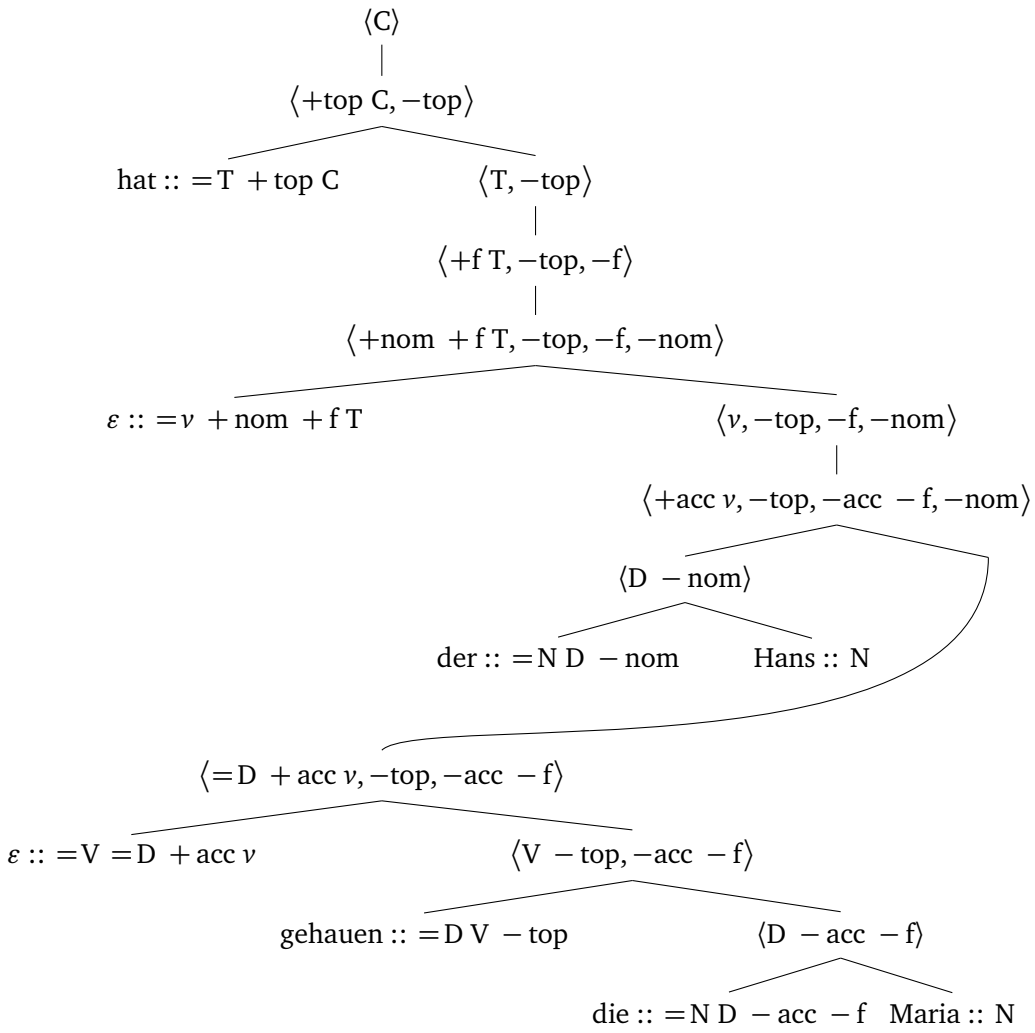
Shift $$\frac{[i, \phi \bullet x\psi]}{[i, \phi x \bullet \psi]}\ \phi \text{ not the empty string, } x \text{ a placeholder or an LI}$$

Example 7.2 MG Top-Down Parse of *Gehauen hat die Maria der Hans*

Let's look at a slightly more complicated version of our very first example sentence.

(2) Gehauen hat die Maria der Hans.
beaten has the Maria the Hans

Here the object *die Maria* not only moves out of the VP, but it also scrambles across the subject *der Hans* afterwards. A simplified derivation tree is given below.

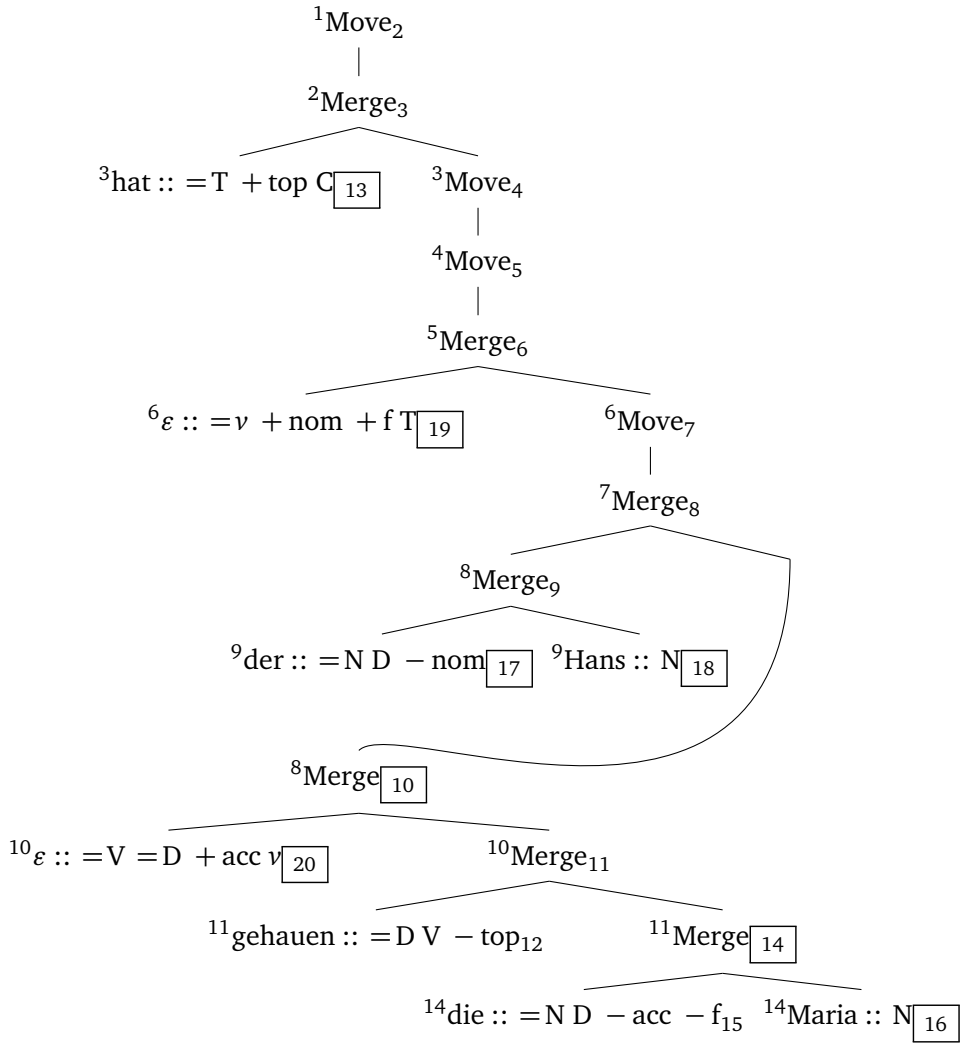


The parser infers the derivation with the following steps.

parse item	**inference rule**
[0, •⟨C⟩]	axiom
[0, [top] •⟨+top C, −top⟩]	Move (final)
[0, [top] •⟨=T +top C⟩ ⟨T, −top⟩]	Merge Right
[0, [top] hat •⟨T, −top⟩]	LI & Shift
[0, [top] hat [f] •⟨+f T, −top, −f⟩]	Move (final)
[0, [top] hat [f] [nom] •⟨+nom +f T, −top, −f, −nom⟩]	Move (final)
[0, [top] hat [f] [nom] •⟨=v +nom +f T⟩ ⟨v, −top, −f, −nom⟩]	Merge Right

[0, [top] hat [f] [nom] ε $\bullet\langle v, -\text{top}, -\text{f}, -\text{nom}\rangle$]	LI & Shift
[0, [top] hat [f] [nom] ε [acc] $\bullet\langle +\text{acc}\ v, -\text{top}, -\text{acc}\ -\text{f}, -\text{nom}\rangle$]	Move (intermediate)
[0, [top] hat [f] [nom] ε [acc] $\bullet[_{\text{nom}}\ \langle \text{D}\ -\text{nom}\rangle]\ \langle =\text{D}\ +\text{acc}\ v, -\text{top}, -\text{acc}\ -\text{f}, -\text{nom}\rangle$]	Merge Mover Left
[0, [top] hat [f] $\bullet\langle \text{D}\ -\text{nom}\rangle$ ε [acc] $\langle =\text{D}\ +\text{acc}\ v, -\text{top}, -\text{acc}\ -\text{f}, -\text{nom}\rangle$]	Displace
[0, [top] hat [f] $\bullet\langle =\text{N D}\ -\text{nom}\rangle\ \langle \text{N}\rangle$ ε [acc] $\langle =\text{D}\ +\text{acc}\ v, -\text{top}, -\text{acc}\ -\text{f}, -\text{nom}\rangle$]	Merge Right
[0, [top] hat [f] der $\bullet\langle \text{N}\rangle$ [f] [nom] ε [acc] $\langle =\text{D}\ +\text{acc}\ v, -\text{top}, -\text{acc}\ -\text{f}, -\text{nom}\rangle$]	LI & Shift
[0, [top] hat [f] der Hans $\bullet\varepsilon$ [acc] $\langle =\text{D}\ +\text{acc}\ v, -\text{top}, -\text{acc}\ -\text{f}, -\text{nom}\rangle$]	LI & Shift
[0, [top] hat [f] der Hans ε [acc] $\bullet\langle =\text{D}\ +\text{acc}\ v, -\text{top}, -\text{acc}\ -\text{f}, -\text{nom}\rangle$]	Shift*2
[0, [top] hat [f] der Hans ε [acc] $\bullet\langle =\text{V}\ =\text{D}\ +\text{acc}\ v\rangle\ [_{\text{top}}\ \langle \text{V}\ -\text{top}, -\text{acc}\ -\text{f}\rangle]$]	Merge Mover Right
[0, [top] hat [f] der Hans ε [acc] ε $\bullet[_{\text{top}}\ \langle \text{V}\ -\text{top}, -\text{acc}\ -\text{f}\rangle]$]	LI & Shift
[0, $\bullet\langle \text{V}\ -\text{top}, -\text{acc}\ -\text{f}\rangle$ hat [f] der Hans ε [acc] ε]	Displace
[0, $\bullet\langle =\text{D V}\ -\text{top}\rangle\ [_{\text{f}}\ \langle \text{D}\ -\text{acc}\ -\text{f}\rangle]$ hat [f] der Hans ε ε]	Merge Mover Right
[0, •gehauen $[_{\text{f}}\ \langle \text{D}\ -\text{acc}\ -\text{f}\rangle]$ hat [f] der Hans ε ε]	LI
[1, $\bullet[_{\text{f}}\ \langle \text{D}\ -\text{acc}\ -\text{f}\rangle]$ hat [f] der Hans ε ε]	Scan
[1, •hat $\langle \text{D}\ -\text{acc}\ -\text{f}\rangle$ der Hans ε ε]	Displace
[2, $\bullet\langle \text{D}\ -\text{acc}\ -\text{f}\rangle$ der Hans ε ε]	Scan
[2, $\bullet\langle =\text{N D}\ -\text{acc}\ -\text{f}\rangle\ \langle \text{N}\rangle$ der Hans ε ε]	Merge Right
[2, •die $\langle \text{N}\rangle$ der Hans ε ε]	LI
[3, $\bullet\langle \text{N}\rangle$ der Hans ε ε]	Scan
[3, •Maria der Hans ε ε]	LI
[4, •der Hans ε ε]	Scan
[5, •Hans ε ε]	Scan
[6, $\bullet\varepsilon$ ε]	Scan
[6, $\bullet\varepsilon$]	Scan
[6, •]	Scan

We can represent this parse more succinctly by annotating the derivation tree with indices in the usual fashion.



7.2.5 Two Issues

The parser as defined in the previous section has two issues. First, the Merge and Move rules are only limited by the requirement that the features F or f must be valid features of the grammar. This, however, can still lead to conjecturing tuples that could never be derived from the LIs of the grammar. These parses will eventually fail, of course, but for efficiency reasons it would be nice if the parser would not conjecture items that can never occur in a well-formed derivation.

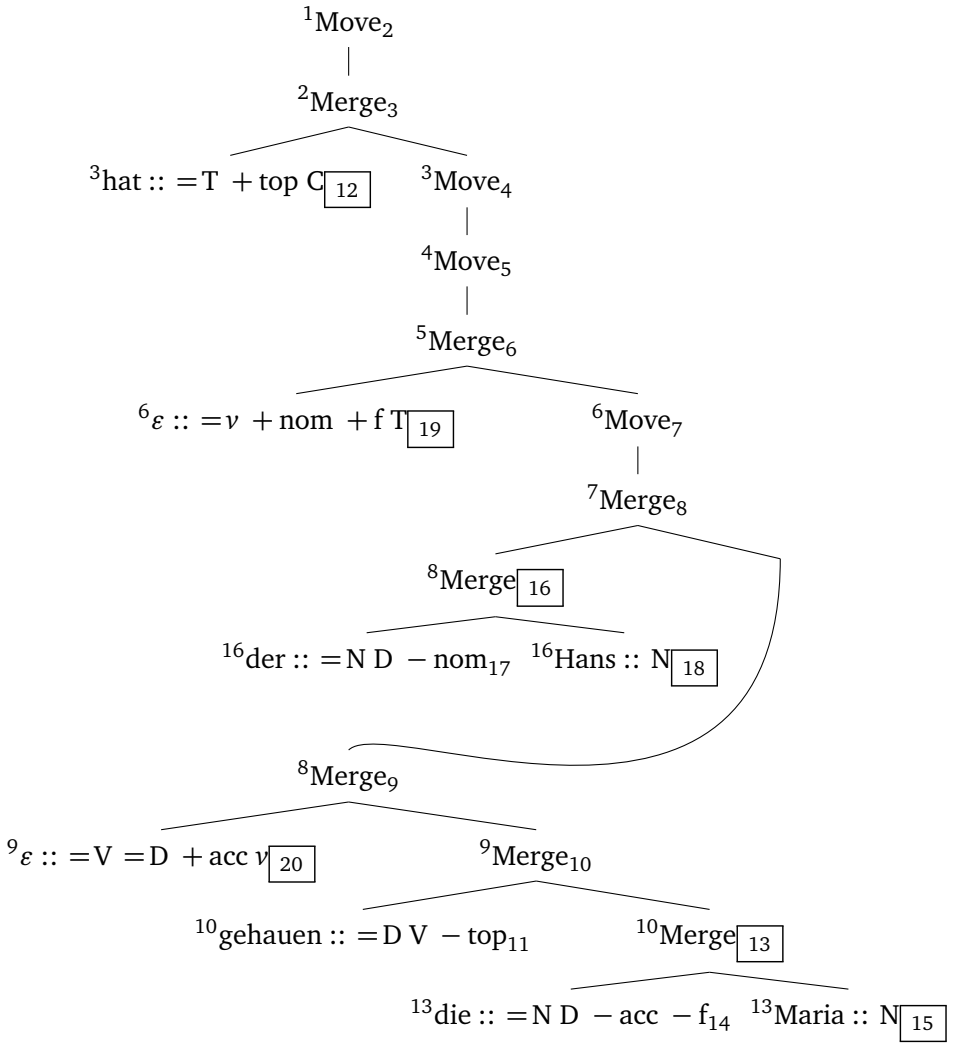
If we translate the MG into a CFG, the set of valid tuples corresponds to the set non-terminals of the CFG. So with a little bit of processing, the rules can be limited to only use non-terminals of this CFG. It would be more appealing, however, if we could

phrase our inference rules in a way so that they operate directly on the lexicon of our MG. This is exactly what is done in Stabler (2012).

Stabler's (2012) parser also fixes another problem with the current model. Right now, our parser does not sufficiently prioritize the search for movers. Since scanning of an LI l is delayed until all movers to the left of l have been found, memory usage would be minimized by searching for those movers. So once the parser stipulates the presence of a top-mover in the example above, it should first follow the branches that it believes will lead it to this mover. Once this mover has been found, it can scan *hat* and then continue it's search for an f-mover, and then for a nom-mover. Instead, our parser starts building the nom-mover before it has even found the top-mover; this increases the memory burden because there is no way the nom-mover could be fully scanned at this point.

Example 7.3 A More Efficient Traversal of the Derivation Tree

Using the search strategy outlined in Stabler (2012), the derivation tree from the previous example would be explored in the following order.



The Stabler parser has a lower payload because it does not fully expand the subject right away and instead moves on to the VP first. If the subject contained more than two LIs, the difference in payload would be even bigger. Maximum tenure stays the same because it is incurred at the T-head, which is introduced before the two parser diverge in their search path and cannot be scanned until the subject has been built and scanned. For this reason, the differences between the two parsers do not affect the node's tenure. We see a marked difference in summed tenure, however, because of the tenure contributed by *der* and *Hans*.

Metric	**CF Parser**	**Stabler Parser**
Payload	8	7
MaxTen	13	13
SumTen	57	48

Exercise 7.1. Draw a movement-free Minimalist derivation tree for *The anvil hit Daffy*. Assume a C-T-V spine where the subject is merged as the second argument of the T-head. The write down the parse table and annotate the derivation tree accordingly. Does the parser's behavior differ noticeably from what we observed for the recursive descent parser in Chapter 3? ⊙

Exercise 7.2. Following up on the previous exercise, assume that we actually have a C-T-v-V spine where the subject starts in Spec,vP and then moves to Spec,TP. Once again you have to write down the parse table and annotate the derivation tree. What differences do you observe? ⊙

Exercise 7.3. Draw Minimalist derivation trees for our left-embedding and right-embedding analyses of *John's father's car's exhaust pipe* disappeared. Assume a C-T-v-V spine where the subject moves from Spec,vP to Spec,TP. Then write down the parse table for each structure. Annotate the derivation trees according to how our parser would build them. Is the behavior of our parser similar to that of the recursive descent parser? ⊙

References and Further Reading

Graf, Thomas. 2013. *Local and transderivational constraints in syntax and semantics*. Doctoral Dissertation, UCLA.

Kobele, Gregory M., Christian Retoré, and Sylvain Salvati. 2007. An automata-theoretic approach to minimalism. In *Model Theoretic Syntax at 10*, ed. James Rogers and Stephan Kepser, 71–80.

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