Appendix A Defining PF+LF Minimalist Grammars

A.1 Overview

The revised Minimalist Grammars (MGs) used to perform the tests in this paper are constructed to simultaneously derive PF representations and Heim & Kratzer (1998)-style LF representations (including indexed traces and lambda abstractors).

Our formulation of MGs will use a **chain notation** similar to that of Stabler & Keenan (2003), though their chains deal in strings where ours deal in trees. In short, chain notation represents a step in a derivation as an ordered tuple of trees with features (a **chain**), where the first tree is the main tree being built, and all of the others represent movers waiting to reach their final landing spots. However, since we are simultaneously building PF and LF representations, the members of our chains will not be individual trees, but *pairs* of trees. Lexical items are themselves (very small) chains with PF and LF representations, which are then manipulated by the operations **merge** and **move** to create larger chains. Importantly, in a chain notation, movement does not involve merging some constituent into the tree and then moving it later; instead, traces are inserted in the movers' place from the get-go, with movers being added to the chain to await their final landing spot, at which point they are removed from the chain and added to the main tree being built. Within a given (PF or LF) tree, a structure $[_<$ A B] indicates that the head of the structure is A or some head within A, while $[_>$ A B] indicates that the head is B or some head within B.

One difference between the formulation here and that in the informal discussion in the body of the paper is that while the latter had a single type of selector feature `=f`, here we use two distinct selector features, `=f` (which selects something to the right, i.e., a complement) and `f=` (which selects something to the left, i.e., a specifier). This is not a necessary stipulation, but it is a convenient one; the MG defined here can be formulated equally well with a single selector feature `=f`.

After a full detailing of all of the formal definitions and **merge** and **move** (sub)rules, we will provide a brief, informal illustration of the basics of how the grammar works.

A.2 Formal definitions

A PF+LF Minimalist Grammar (PF+LF MG) is an ordered tuple $\langle \textsc{Feat}, \mathrm{in}, \mathsf{PF}, \mathsf{LF}, \textsc{Lex}\rangle$. $\textsc{Feat}$ is a set of **features**, defined by means of a set $\textsc{Base}$ of base features as follows:

(1) $\textsc{Feat} := \bigcup\{\{\mathtt{f}, \mathtt{=f}, \mathtt{f=}, \mathtt{+f}, \mathtt{+f_P}, \mathtt{+f_L}, \mathtt{-f}, \mathtt{-f_P}, \mathtt{-f_L}\} \mid \mathtt{f} \in \textsc{Base}\}$

in : $\textsc{Feat} \rightarrow \mathbb{N}$ is the **indexation function**, a one-to-one function from features to natural numbers. PF is a PF alphabet, and LF an LF alphabet, without lambda abstractors or traces. Let $\textsc{traces} := \{t_{\mathrm{in}(k)} \mid k \in \textsc{Feat}\}$, and let $\textsc{lambdas} := \{\lambda_{\mathrm{in}(k)} \mid k \in \textsc{Feat}\}$. Let $\textsc{Trees}_{\mathsf{PF}}$ be the set of all **PF trees**, i.e., those trees—defined in the usual way—whose leaf nodes are labeled with members of $\mathsf{PF} \cup \{\varepsilon\}$ (where ε is the empty string) and whose internal nodes are labeled < or > (indicating left- or right-headedness). Let $\textsc{Trees}_{\mathsf{LF}}$ be the set of all **LF trees**, i.e., those trees whose leaf nodes are labeled with members of $\mathsf{LF} \cup \textsc{traces} \cup \textsc{lambdas}$, and whose internal nodes are labeled < or >. A **feature-specified tree pair (FST2)** is an ordered triple (τ_a, τ_b, δ), where $\tau_a \in \textsc{Trees}_{\mathsf{PF}}$, $\tau_b \in \textsc{Trees}_{\mathsf{LF}}$, and $\delta \in \textsc{Feat}^*$ (the set of strings of features, including the empty string). For ease of reading, PF trees will be in red, and LF trees in blue. A **chain** is a non-empty ordered tuple of FST2s. The first FST2 in a chain is its **primary FST2**. The first feature in an FST2's string of features (if it is non-empty) is its **active feature**. $\textsc{Lex}$ is the **lexicon**: a set of chains, where each lexical entry is a chain consisting of a single FST2, with the PF and LF trees in that FST2 consisting of a single node (that lexical item's PF and LF representation).

There are two operations on chains, where each operation is a function whose output is itself a chain: binary **merge**, and unary **move**. In what follows, each of these operations is divided into eight sub-operations, depending on whether (and how) the selected-for phrase undergoes subsequent movement. These sub-operations have disjoint domains, so **merge** and **move** are in fact functions. For **merge** operations, the first argument is the one whose primary FST2 contains the selector feature (`=f` or `f=`), and the second the one whose primary FST2 contains the category feature (`f`). For ease of reading, we will rewrite **merge**(A,B) as A **merge** B, and **move**(A) as **move** A. For a given PF+LF MG G, the *language* of G is the closure of G's lexicon under **merge** and **move**.

For all definitions, α and β (with various indices) are variables over FST2s. In a given chain, $\alpha_1 \dots \alpha_m$ indicates a (possibly empty) subsequence of FST2s. δ (with varying indices) is a variable over (possibly empty) feature (sub)strings.

A.3 Merge operations

A.3.1 Merge in complement

merge-c-N (merge in complement, no subsequent movement)

$\langle(\tau_{1a},\ \tau_{1b},\ \texttt{=x}\ \ \delta_1);\ \alpha_1\dots\alpha_m\rangle$ **merge** $\langle(\tau_{2a},\ \tau_{2b},\ \texttt{x});\ \beta_1\dots\beta_n\rangle =$
$\langle([_<\ \tau_{1a}\ \tau_{2a}],\ [_<\ \tau_{1b}\ \tau_{2b}],\ \delta_1);\ \alpha_1\dots\alpha_m;\ \beta_1\dots\beta_n\rangle$

merge-c-PL (merge in complement, subsequent PF+LF movement)

$\langle(\tau_{1a},\ \tau_{1b},\ \texttt{=x}\ \ \delta_1);\ \alpha_1\dots\alpha_m\rangle$ **merge** $\langle(\tau_{2a},\ \tau_{2b},\ \texttt{x -f}\ \ \delta_2);\ \beta_1\dots\beta_n\rangle =$
$\langle([_<\ \tau_{1a}\ \varepsilon],\ [_<\ \tau_{1b}\ t_{\mathrm{in}(\texttt{-f})}],\ \delta_1);\ (\tau_{2a},\ \tau_{2b},\ \texttt{-f}\ \ \delta_2);\ \alpha_1\dots\alpha_m;\ \beta_1\dots\beta_n\rangle$

merge-c-P (merge in complement, subsequent PF-only movement)

$\langle(\tau_{1a},\ \tau_{1b},\ \texttt{=x}\ \ \delta_1);\ \alpha_1\dots\alpha_m\rangle$ **merge** $\langle(\tau_{2a},\ \tau_{2b},\ \texttt{x -f}_\texttt{P}\ \ \delta_2);\ \beta_1\dots\beta_n\rangle =$
$\langle([_<\ \tau_{1a}\ \varepsilon],\ [_<\ \tau_{1b}\ \tau_{2b}],\ \delta_1);\ (\tau_{2a},\ \varepsilon,\ \texttt{-f}_\texttt{P}\ \ \delta_2);\ \alpha_1\dots\alpha_m;\ \beta_1\dots\beta_n\rangle$

merge-c-L (merge in complement, subsequent LF-only movement)

$\langle(\tau_{1a},\ \tau_{1b},\ \texttt{=x}\ \ \delta_1);\ \alpha_1\dots\alpha_m\rangle$ **merge** $\langle(\tau_{2a},\ \tau_{2b},\ \texttt{x -f}_\texttt{L}\ \ \delta_2);\ \beta_1\dots\beta_n\rangle =$
$\langle([_<\ \tau_{1a}\ \tau_{2a}],\ [_<\ \tau_{1b}\ t_{\mathrm{in}(\texttt{-f}_\texttt{L})}],\ \delta_1);\ (\varepsilon,\ \tau_{2b},\ \texttt{-f}_\texttt{L}\ \ \delta_2);\ \alpha_1\dots\alpha_m;\ \beta_1\dots\beta_n\rangle$

A.3.2 Merge in specifier

merge-s-N (merge in specifier, no subsequent movement)

$\langle(\tau_{1a},\ \tau_{1b},\ \texttt{x=}\ \ \delta_1);\ \alpha_1\dots\alpha_m\rangle$ **merge** $\langle(\tau_{2a},\ \tau_{2b},\ \texttt{x});\ \beta_1\dots\beta_n\rangle =$
$\langle([_>\ \tau_{2a}\ \tau_{1a}],\ [_>\ \tau_{2b}\ \tau_{1b}],\ \delta_1);\ \alpha_1\dots\alpha_m;\ \beta_1\dots\beta_n\rangle$

merge-s-PL (merge in specifier, subsequent PF+LF movement)

$\langle(\tau_{1a},\ \tau_{1b},\ \texttt{x=}\ \ \delta_1);\ \alpha_1\dots\alpha_m\rangle$ **merge** $\langle(\tau_{2a},\ \tau_{2b},\ \texttt{x -f}\ \ \delta_2);\ \beta_1\dots\beta_n\rangle =$
$\langle([_>\ \varepsilon\ \tau_{1a}],\ [_>\ t_{\mathrm{in}(\texttt{-f})}\ \tau_{1b}],\ \delta_1);\ (\tau_{2a},\ \tau_{2b},\ \texttt{-f}\ \ \delta_2);\ \alpha_1\dots\alpha_m;\ \beta_1\dots\beta_n\rangle$

merge-s-P (merge in specifier, subsequent PF-only movement)

$\langle(\tau_{1a},\ \tau_{1b},\ \texttt{x=}\ \ \delta_1);\ \alpha_1\dots\alpha_m\rangle$ **merge** $\langle(\tau_{2a},\ \tau_{2b},\ \texttt{x -f}_\texttt{P}\ \ \delta_2);\ \beta_1\dots\beta_n\rangle) =$
$\langle([_>\ \varepsilon\ \tau_{1a}],\ [_>\ \tau_{2b}\ \tau_{1b}],\ \delta_1);\ (\tau_{2a},\ \varepsilon,\ \texttt{-f}_\texttt{P}\ \ \delta_2);\ \alpha_1\dots\alpha_m;\ \beta_1\dots\beta_n\rangle$

merge-s-L (merge in specifier, subsequent LF-only movement)

$\langle(\tau_{1a},\ \tau_{1b},\ \texttt{x=}\ \ \delta_1);\ \alpha_1\dots\alpha_m\rangle$ **merge** $\langle(\tau_{2a},\ \tau_{2b},\ \texttt{x -f}_\texttt{L}\ \ \delta_2);\ \beta_1\dots\beta_n\rangle =$
$\langle([_>\ \tau_{2a}\ \tau_{1a}],\ [_>\ t_{\mathrm{in}(\texttt{-f}_\texttt{L})}\ \tau_{1b}],\ \delta_1);\ (\varepsilon,\ \tau_{2b},\ \texttt{-f}_\texttt{L}\ \ \delta_2);\ \alpha_1\dots\alpha_m;\ \beta_1\dots\beta_n\rangle$

A.4 Move operations

move is defined only for those chains meeting the following conditions: (I) the primary FST2 has as its active feature a movement licensor feature $\texttt{+g}$ (or $\texttt{+g}_{\text{P}}$, $\texttt{+g}_{\text{L}}$), and (II) there is exactly one non-primary FST2 whose active feature is $\texttt{-g}$ (or $\texttt{-g}_{\text{P}}$, $\texttt{-g}_{\text{L}}$) (*Shortest Move Constraint*).[1]

A.4.1 PF+LF movement

move-PL-N (PF+LF movement with no subsequent movement)

move $\langle(\tau_{1\text{a}},\ \tau_{1\text{b}},\ \texttt{+g}\ \ \delta_1);\ \alpha_1\ldots\alpha_m;\ (\tau_{2\text{a}},\ \tau_{2\text{b}},\ \texttt{-g});\ \beta_1\ldots\beta_n\rangle =$

$\langle([_{>}\ \tau_{2\text{a}}\ \tau_{1\text{a}}],\ [_{>}\ \tau_{2\text{b}}\ [_{>}\ \lambda_{\text{in}(\texttt{-g})}\ \tau_{1\text{b}}]],\ \delta_1);\ \alpha_1\ldots\alpha_m;\ \beta_1\ldots\beta_n\rangle$

move-PL-PL (PF+LF movement with subsequent PF+LF movement)

move $\langle(\tau_{1\text{a}},\ \tau_{1\text{b}},\ \texttt{+g}\ \ \delta_1);\ \alpha_1\ldots\alpha_m;\ (\tau_{2\text{a}},\ \tau_{2\text{b}},\ \texttt{-g}\ \ \texttt{-h}\ \ \delta_2);\ \beta_1\ldots\beta_n\rangle =$

$\langle([_{>}\ \varepsilon\ \tau_{1\text{a}}],\ [_{>}\ t_{\text{in}(\texttt{-h})}\ [_{>}\ \lambda_{\text{in}(\texttt{-g})}\ \tau_{1\text{b}}]],\ \delta_1);\ \alpha_1\ldots\alpha_m;\ (\tau_{2\text{a}},\ \tau_{2\text{b}},\ \texttt{-h}\ \ \delta_2);\ \beta_1\ldots\beta_n\rangle$

move-PL-P (PF+LF movement with subsequent PF-only movement)

move $\langle(\tau_{1\text{a}},\ \tau_{1\text{b}},\ \texttt{+g}\ \ \delta_1);\ \alpha_1\ldots\alpha_m;\ (\tau_{2\text{a}},\ \tau_{2\text{b}},\ \texttt{-g}\ \ \texttt{-h}_{\text{P}}\ \ \delta_2);\ \beta_1\ldots\beta_n\rangle =$

$\langle([_{>}\ \varepsilon\ \tau_{1\text{a}}],\ [_{>}\ \tau_{2\text{b}}\ [_{>}\ \lambda_{\text{in}(\texttt{-g})}\ \tau_{1\text{b}}]],\ \delta_1);\ \alpha_1\ldots\alpha_m;\ (\tau_{2\text{a}},\ \varepsilon,\ \texttt{-h}_{\text{P}}\ \ \delta_2);\ \beta_1\ldots\beta_n\rangle$

move-PL-L (PF+LF movement with subsequent LF-only movement)

move $\langle(\tau_{1\text{a}},\ \tau_{1\text{b}},\ \texttt{+g}\ \ \delta_1);\ \alpha_1\ldots\alpha_m;\ (\tau_{2\text{a}},\ \tau_{2\text{b}},\ \texttt{-g}\ \ \texttt{-h}_{\text{L}}\ \ \delta_2);\ \beta_1\ldots\beta_n\rangle =$

$\langle([_{>}\ \tau_{2\text{a}}\ \tau_{1\text{a}}],\ [_{>}\ t_{\text{in}(\texttt{-h}_{\text{L}})}\ [_{>}\ \lambda_{\text{in}(\texttt{-g})}\ \tau_{1\text{b}}]],\ \delta_1);\ \alpha_1\ldots\alpha_m;\ (\varepsilon,\ \tau_{2\text{b}},\ \texttt{-h}_{\text{L}}\ \ \delta_2);\ \beta_1\ldots\beta_n\rangle$

A.4.2 PF-only movement

move-P-N (PF-only movement with no subsequent movement)

move $\langle(\tau_{1\text{a}},\ \tau_{1\text{b}},\ \texttt{+g}_{\text{P}}\ \ \delta_1);\ \alpha_1\ldots\alpha_m;\ (\tau_{2\text{a}},\ \varepsilon,\ \texttt{-g}_{\text{P}});\ \beta_1\ldots\beta_n\rangle =$

$\langle([_{>}\ \tau_{2\text{a}}\ \tau_{1\text{a}}],\ \tau_{1\text{b}},\ \delta_1);\ \alpha_1\ldots\alpha_m;\ \beta_1\ldots\beta_n\rangle$

move-P-P (PF-only movement with subsequent PF-only movement)

move $\langle(\tau_{1\text{a}},\ \tau_{1\text{b}},\ \texttt{+g}_{\text{P}}\ \ \delta_1);\ \alpha_1\ldots\alpha_m;\ (\tau_{2\text{a}},\ \varepsilon,\ \texttt{-g}_{\text{P}}\ \ \texttt{-h}_{\text{P}}\ \ \delta_2);\ \beta_1\ldots\beta_n\rangle =$

$\langle([_{>}\ \varepsilon\ \tau_1],\ \tau_{1\text{b}},\ \delta_1);\ \alpha_1\ldots\alpha_m;\ (\tau_{2\text{a}},\ \varepsilon,\ \texttt{-h}_{\text{P}}\ \ \delta_2);\ \beta_1\ldots\beta_n\rangle$

A.4.3 LF-only movement

move-L-N (LF-only movement with no subsequent movement)

move $\langle(\tau_{1\text{a}},\ \tau_{1\text{b}},\ \texttt{+g}_{\text{L}}\ \ \delta_1);\ \alpha_1\ldots\alpha_m;\ (\varepsilon,\ \tau_{2\text{b}},\ \texttt{-g}_{\text{L}});\ \beta_1\ldots\beta_n\rangle =$

$\langle(\tau_{1\text{a}},\ [_{>}\ \tau_{2\text{b}}\ [_{>}\ \lambda_{\text{in}(\texttt{-g}_{\text{L}})}\ \tau_{1\text{b}}]],\ \delta_1);\ \alpha_1\ldots\alpha_m;\ \beta_1\ldots\beta_n\rangle$

move-L-L (LF-only movement with subsequent LF-only movement)

move $\langle(\tau_{1\text{a}},\ \tau_{1\text{b}},\ \texttt{+g}_{\text{L}}\ \ \delta_1);\ \alpha_1\ldots\alpha_m;\ (\varepsilon,\ \tau_{2\text{b}},\ \texttt{-g}_{\text{L}}\ \ \texttt{-h}_{\text{L}}\ \ \delta_2);\ \beta_1\ldots\beta_n\rangle =$

$\langle(\tau_{1\text{a}},\ [_{>}\ t_{\text{in}(\texttt{-h}_{\text{L}})}\ [_{>}\ \lambda_{\text{in}(\texttt{-g}_{\text{L}})}\ \tau_{1\text{b}}]],\ \delta_1);\ \alpha_1\ldots\alpha_m;\ (\varepsilon,\ \tau_{2\text{b}},\ \texttt{-h}_{\text{L}}\ \ \delta_2);\ \beta_1\ldots\beta_n\rangle$

[1] The Shortest Move Constraint (SMC) is a standard feature of MGs that has an additional benefit for our purposes: it ensures that there is no conflict in what indices are assigned to traces. Since trace indexation is determined by movement licensee features, by the SMC it must be the case that if two movers were to leave traces of the same index, one must complete its movement (and thus have its trace bound by lambda abstraction) before the other can start its movement. In other words, there will be no instances of movers "accidentally" binding other movers' traces.

A.5 Informal illustration

Here is a brief, informal illustration of how the grammar works. Say we have the determiner *every* and noun *plane*, with the following lexical entries:

(2) a. ⟨(every, EVERY, `=N D -sc`$_{\mathrm{L}}$)⟩
   b. ⟨(plane, PLANE, `N`)⟩

The feature `-sc`$_{\mathrm{L}}$ is the covert scope-taking licensee feature. When these two unary chains merge, they create another unary chain, with more complex PF and LF trees (**merge-c-N**):

(3) ⟨(every, EVERY, `=N D -sc`$_{\mathrm{L}}$)⟩ **merge** ⟨(plane, PLANE, `N`)⟩
   = ⟨([$_<$ every plane], [$_<$ EVERY PLANE], `D -sc`$_{\mathrm{L}}$)⟩

Now say this merges as the complement of the verb *fixed*, with the following lexical entry:

(4) ⟨(fixed, FIX, `=D V`)⟩

The licensee feature for *every plane* is `-sc`$_{\mathrm{L}}$: it will undergo LF-only movement. As a result, what happens when *fixed* and *every plane* merge is that the PF portion of *every plane* will stay with *fixed*, while in the LF portion an indexed trace will be inserted that will subsequently be lambda-abstracted over. The LF part of *every plane* (with a placeholder empty PF part ε) is then added to the chain, since it is now a mover looking for its final landing spot (**merge-c-L**):

(5) ⟨(fixed, FIX, `=D V`)⟩ **merge** ⟨([$_<$ every plane], [$_<$ EVERY PLANE], `D -sc`$_{\mathrm{L}}$)⟩
   = ⟨([$_<$ fixed [$_<$ every plane]], [$_<$ FIX $t_{\mathrm{in}(\texttt{-sc}_{\mathrm{L}})}$], `V`); ($\varepsilon$, [$_<$ EVERY PLANE], `-sc`$_{\mathrm{L}}$)⟩

Once the landing site is eventually reached, an appropriately indexed lambda abstractor is inserted below the final landing site, and the mover is removed from the chain; since the movement is LF-only, the PF tree is unaffected (**move-L-N**):

(6) **move** ⟨([...[$_<$ fixed [$_<$ every plane]]], [...[$_<$ FIX $t_{\mathrm{in}(\texttt{-sc}_{\mathrm{L}})}$]], `+sc`$_{\mathrm{L}}$ ...); (ε, [$_<$ EVERY PLANE], `-sc`$_{\mathrm{L}}$)⟩
   = ⟨([...[$_<$ fixed [$_<$ every plane]]], [$_>$ [$_<$ EVERY PLANE] [$_>$ $\lambda_{\mathrm{in}(\texttt{-sc}_{\mathrm{L}})}$ [...[$_<$ FIX $t_{\mathrm{in}(\texttt{-sc}_{\mathrm{L}})}$]]]], ...)⟩

PF-only movement is essentially the opposite, except no lambda-abstractors are inserted, and instead of indexed traces locations vacated by movement are simply occupied by the phonetically empty node ε. Standard PF+LF movement is, as one may suspect, a combination of the two. The diversity of **merge** and **move** subrules reflects the various combinations of PF-only, LF-only, and PF+LF movement that are possible.

Note in addition that for a phonetically empty head like C, the PF representation in its lexical entry will be ε; only the features and the LF representation will reveal that it is in fact a C head.

Appendix B Top-down parse rules for PF+LF MGs

B.1 Overview

Here we introduce the top-down parser for our revised PF+LF MGs. For reasons of space many details have had to be presented with little to no exposition. Readers unfamiliar with top-down MG parsing in general are encouraged to first familiarize themselves with the parser formalisms of Kobele et al. (2013) and work cited therein; our parser builds on this work.

A top-down MG parser reconstructs an MG derivation tree from the top downward, in a way that is sensitive to linear order in the PF phrase structure tree. Order is determined by **gorn addresses**, a common formalism for assigning addresses to locations in a tree. Whereas derivations involved chains, which were tuples of FST2s, parses utilize **parse items**, which are tuples of **doubly addressed feature strings** (**AFS2s**). An AFS2—an ordered triple consisting of two gorn addresses and a string of features—can be thought of as a prediction that some constituent occupies the first address at PF and the second address at LF, and has the associated string of features at the relevant point in the derivation. Parse items are tuples of AFS2s for much the same reason that chains were tuples of FST2s: the first AFS2 makes a prediction about the "main" tree, while subsequent AFS2s make predictions about currently-waiting movers. The gorn addresses in our formalism come with a twist: unlike in standard MGs, here it will be possible for a mover to be predicted, but for its address at either PF or LF to not yet be determined; in this case the relevant "address" will be the dummy address ε. In addition to defining **SLD** in terms of the annotated derivation tree, as is done in the paper, **SLD** can also be defined in terms of these dummy addresses: the **SLD** for a given parse is the number of tokens of dummy addresses occuring throughout the parse.

We operate within a parsing-as-deduction framework (Pereira & Warren 1983): parses are deductions in which the sole axiom is the privileged initial parse item (see below), the inference rules are the parse rules, and the goal for a successful deduction is the elimination of all parse items and the scanning of the whole input string.

B.2 Formal definitions

A **gorn address** is a non-empty string of non-negative integers, denoting a location in a (phrase structure) tree. Gorn addresses are assigned recursively, as follows:

(7) a. The address of the root node is 0.
  b. For a given node j with address n, j's leftmost daughter has address $n0$, its next leftmost daughter has address $n1$, etc.

Since all our trees are binary-branching, all digits in our gorn addresses will be 0 or 1. Let GORN be the set of all gorn addresses, and let GORN+ := GORN $\cup \{\varepsilon\}$, the set of **gorn+ addresses**. The "address" ε indicates that a given address is unknown. A **doubly addressed feature string (AFS2)** is an ordered triple where the first member is a PF gorn+ address, the second is an LF gorn+ address, and the third is a (non-empty) string of features. A **parse item** is an ordered tuple of AFS2s. All parses start with the privileged parse item $\langle(0, 0, \texttt{C})\rangle$, indicating that we predict the root at both PF and LF to be a phrase headed by C; note we retain our color conventions (red = PF, blue = LF). Parse items are ordered based on the linear order of their leftmost (non-dummy) PF address, regardless of whether that is the address of the primary AFS2 or some other AFS2 in the parse item. Parse operations only operate on the leftmost parse item, with all other parse items being passed down to the next step of the parse. Thus, MG parsing is sensitive to PF linear order: the parser takes the quickest path to scanning the leftmost unscanned element in the input string. A parse is successful if and only if (I) the entire string has been scanned, and (II) all parse items have been eliminated.

For the following parse rules, δ (and its subscripted variants) is a variable over feature strings, and γ (and its subscripted variants) a variable over gorn addresses. A and B are variables over (possibly empty) sequences of AFS2s. $A \oplus B$ indicates a (possibly empty) sequence of AFS2s whose members can be partitioned into A and B (not necessarily by order). For AFS2 κ, $B[\kappa]$ indicates that κ is in the sequence B, with $B[\kappa']$ being the sequence identical to B but replacing κ with κ', and $B[-]$ being the sequence identical to B but without κ.

B.3 Unmerge operations

B.3.1 Unmerge from complement

unmerge-c-N (undoes **merge-c-N**)

$$\frac{\langle(\gamma_{\mathrm{a}},\ \gamma_{\mathrm{b}},\ \delta_1);\ A \oplus B\rangle}{\langle(\gamma_{\mathrm{a}}0,\ \gamma_{\mathrm{b}}0,\ \texttt{=x}\ \delta_1);\ A\rangle\quad \langle(\gamma_{\mathrm{a}}1,\ \gamma_{\mathrm{b}}1,\ \texttt{x});\ B\rangle}$$

unmerge-c-PL (undoes **merge-c-PL**)

$$\frac{\langle(\gamma_{1\mathrm{a}},\ \gamma_{1\mathrm{b}},\ \delta_1);\ A \oplus B[(\gamma_{2\mathrm{a}},\ \gamma_{2\mathrm{b}},\ \texttt{-f}\ \delta_2)]\rangle}{\langle(\gamma_{1\mathrm{a}}0,\ \gamma_{1\mathrm{b}}0,\ \texttt{=x}\ \delta_1);\ A\rangle\quad \langle(\gamma_{2\mathrm{a}},\ \gamma_{2\mathrm{b}},\ \texttt{x -f}\ \delta_2);\ B[-]\rangle}$$

unmerge-c-P (undoes **merge-c-P**)

$$\frac{\langle(\gamma_{1\mathrm{a}},\ \gamma_{1\mathrm{b}},\ \delta_1);\ A \oplus B[(\gamma_{2\mathrm{a}},\ \varepsilon,\ \texttt{-f}_{\mathrm{P}}\ \delta_2)]\rangle}{\langle(\gamma_{1\mathrm{a}}0,\ \gamma_{1\mathrm{b}}0,\ \texttt{=x}\ \delta_1);\ A\rangle\quad \langle(\gamma_{2\mathrm{a}},\ \gamma_{1\mathrm{b}}1,\ \texttt{x -f}_{\mathrm{P}}\ \delta_2);\ B[-]\rangle}$$

unmerge-c-L (undoes **merge-c-L**)

$$\frac{\langle(\gamma_{1\mathrm{a}},\ \gamma_{1\mathrm{b}},\ \delta_1);\ A \oplus B[(\varepsilon,\ \gamma_{2\mathrm{b}},\ \texttt{-f}_{\mathrm{L}}\ \delta_2)]\rangle}{\langle(\gamma_{1\mathrm{a}}0,\ \gamma_{1\mathrm{b}}0,\ \texttt{=x}\ \delta_1);\ A\rangle\quad \langle(\gamma_{1\mathrm{a}}1,\ \gamma_{2\mathrm{b}},\ \texttt{x -f}_{\mathrm{L}}\ \delta_2);\ B[-]\rangle}$$

B.3.2 Unmerge from specifier

unmerge-s-N (undoes **merge-s-N**)

$$\frac{\langle(\gamma_{\mathrm{a}},\ \gamma_{\mathrm{b}},\ \delta_1);\ A \oplus B\rangle}{\langle(\gamma_{\mathrm{a}}1,\ \gamma_{\mathrm{b}}1,\ \texttt{x=}\ \delta_1);\ A\rangle\quad \langle(\gamma_{\mathrm{a}}0,\ \gamma_{\mathrm{b}}0,\ \texttt{x});\ B\rangle}$$

unmerge-s-PL (undoes **merge-s-PL**)

$$\frac{\langle(\gamma_{1\mathrm{a}},\ \gamma_{1\mathrm{b}},\ \delta_1);\ A \oplus B[(\gamma_{2\mathrm{a}},\ \gamma_{2\mathrm{b}},\ \texttt{-f}\ \delta_2)]\rangle}{\langle(\gamma_{1\mathrm{a}}1,\ \gamma_{1\mathrm{b}}1,\ \texttt{x=}\ \delta_1);\ A\rangle\quad \langle(\gamma_{2\mathrm{a}},\ \gamma_{2\mathrm{b}},\ \texttt{x -f}\ \delta_2);\ B[-]\rangle}$$

unmerge-s-P (undoes **merge-s-P**)

$$\frac{\langle(\gamma_{1\mathrm{a}},\ \gamma_{1\mathrm{b}},\ \delta_1);\ A \oplus B[(\gamma_{2\mathrm{a}},\ \varepsilon,\ \texttt{-f}_{\mathrm{P}}\ \delta_2)]\rangle}{\langle(\gamma_{1\mathrm{a}}1,\ \gamma_{1\mathrm{b}}1,\ \texttt{x=}\ \delta_1);\ A\rangle\quad \langle(\gamma_{2\mathrm{a}},\ \gamma_{1\mathrm{b}}0,\ \texttt{x -f}_{\mathrm{P}}\ \delta_2);\ B[-]\rangle}$$

unmerge-s-L (undoes **merge-s-L**)

$$\frac{\langle(\gamma_{1\mathrm{a}},\ \gamma_{1\mathrm{b}},\ \delta_1);\ A \oplus B[(\varepsilon,\ \gamma_{2\mathrm{b}},\ \texttt{-f}_{\mathrm{L}}\ \delta_2)]\rangle}{\langle(\gamma_{1\mathrm{a}}1,\ \gamma_{1\mathrm{b}}1,\ \texttt{x=}\ \delta_1);\ A\rangle\quad \langle(\gamma_{1\mathrm{a}}0,\ \gamma_{2\mathrm{b}},\ \texttt{x -f}_{\mathrm{L}}\ \delta_2);\ B[-]\rangle}$$

B.4 *Unmove operations*

B.4.1 PF+LF unmove

unmove-PL-N (undoes **move-PL-N**)

$$\frac{\langle(\gamma_{\mathrm{a}},\ \gamma_{\mathrm{b}},\ \delta);\ A\rangle}{\langle(\gamma_{\mathrm{a}}1,\ \gamma_{\mathrm{b}}11,\ \texttt{+f}\ \ \delta);\ A;\ (\gamma_{\mathrm{a}}0,\ \gamma_{\mathrm{b}}0,\ \texttt{-f})\rangle}$$

(⇑ Notice the extra 1 inserted in the LF gorn address: $\gamma_{\mathrm{b}}11$, not $\gamma_{\mathrm{b}}1$. This is because LF movement inserts a lambda-abstracting node just below the landing site, so undoing LF movement requires skipping past this extra inserted node.)

unmove-PL-PL (undoes **move-PL-PL**)

$$\frac{\langle(\gamma_{1\mathrm{a}},\ \gamma_{1\mathrm{b}},\ \delta_1);\ A[(\gamma_{2\mathrm{a}},\ \gamma_{2\mathrm{b}},\ \texttt{-f}\ \ \delta_2)]\rangle}{\langle(\gamma_{1\mathrm{a}}1,\ \gamma_{1\mathrm{b}}11,\ \texttt{+g}\ \ \delta_1);\ A[(\gamma_{2\mathrm{a}},\ \gamma_{2\mathrm{b}},\ \texttt{-g}\ \ \texttt{-f}\ \ \delta_2)]\rangle}$$

unmove-PL-P (undoes **move-PL-P**)

$$\frac{\langle(\gamma_{1\mathrm{a}},\ \gamma_{1\mathrm{b}},\ \delta_1);\ A[(\gamma_{2\mathrm{a}},\ \varepsilon,\ \texttt{-f}_{\texttt{P}}\ \ \delta_2)]\rangle}{\langle(\gamma_{1\mathrm{a}}1,\ \gamma_{1\mathrm{b}}11,\ \texttt{+g}\ \ \delta_1);\ A[(\gamma_{2\mathrm{a}},\ \gamma_{1\mathrm{b}}0,\ \texttt{-g}\ \ \texttt{-f}_{\texttt{P}}\ \ \delta_2)]\rangle}$$

unmove-PL-L (undoes **move-PL-L**)

$$\frac{\langle(\gamma_{1\mathrm{a}},\ \gamma_{1\mathrm{b}},\ \delta_1);\ A[(\varepsilon,\ \gamma_{2\mathrm{b}},\ \texttt{-f}_{\texttt{L}}\ \ \delta_2)]\rangle}{\langle(\gamma_{1\mathrm{a}}1,\ \gamma_{1\mathrm{b}}11,\ \texttt{+g}\ \ \delta_1);\ A[(\gamma_{1\mathrm{a}}0,\ \gamma_{2\mathrm{b}},\ \texttt{-g}\ \ \texttt{-f}_{\texttt{L}}\ \ \delta_2)]\rangle}$$

B.4.2 PF-only unmove

unmove-P-N (undoes **move-P-N**)

$$\frac{\langle(\gamma_{\mathrm{a}},\ \gamma_{\mathrm{b}},\ \delta);\ A\rangle}{\langle(\gamma_{\mathrm{a}}1,\ \gamma_{\mathrm{b}},\ \texttt{+f}_{\texttt{P}}\ \ \delta);\ A;\ (\gamma_{\mathrm{a}}0,\ \varepsilon,\ \texttt{-f}_{\texttt{P}})\rangle}$$

unmove-P-P (undoes **move-P-P**)

$$\frac{\langle(\gamma_{1\mathrm{a}},\ \gamma_{1\mathrm{b}},\ \delta_1);\ A[(\gamma_{2\mathrm{a}},\ \varepsilon,\ \texttt{-f}_{\texttt{P}}\ \ \delta_2)]\rangle}{\langle(\gamma_{1\mathrm{a}}1,\ \gamma_{1\mathrm{b}},\ \texttt{+g}_{\texttt{P}}\ \ \delta_1);\ A[(\gamma_{2\mathrm{a}},\ \varepsilon,\ \texttt{-g}_{\texttt{P}}\ \ \texttt{-f}_{\texttt{P}}\ \ \delta_2)]\rangle}$$

B.4.3 LF-only unmove

unmove-L-N (undoes **move-L-N**)

$$\frac{\langle(\gamma_{\mathrm{a}},\ \gamma_{\mathrm{b}},\ \delta);\ A\rangle}{\langle(\gamma_{\mathrm{a}},\ \gamma_{\mathrm{b}}11,\ \texttt{+f}_{\texttt{L}}\ \ \delta);\ A;\ (\varepsilon,\ \gamma_{\mathrm{b}}0,\ \texttt{-f}_{\texttt{L}})\rangle}$$

unmove-L-L (undoes **move-L-L**)

$$\frac{\langle(\gamma_{1\mathrm{a}},\ \gamma_{1\mathrm{b}},\ \delta_1);\ A[(\varepsilon,\ \gamma_{2\mathrm{b}},\ \texttt{-f}_{\texttt{L}}\ \ \delta_2)]\rangle}{\langle(\gamma_{1\mathrm{a}},\ \gamma_{1\mathrm{b}}11,\ \texttt{+g}_{\texttt{L}}\ \ \delta_1);\ A[(\varepsilon,\ \gamma_{2\mathrm{b}},\ \texttt{-g}_{\texttt{L}}\ \ \texttt{-f}_{\texttt{L}}\ \ \delta_2)]\rangle}$$

B.4.4 Scan

scan

$$\frac{\langle(\gamma_{\mathrm{a}},\ \gamma_{\mathrm{b}},\ \delta)\rangle}{}$$

(where the lexicon contains some entry $\langle(\alpha,\ \beta,\ \delta)\rangle$, and $\alpha = \varepsilon$ or α is the next element in the input string)

Appendix C Annotated Derivation Trees

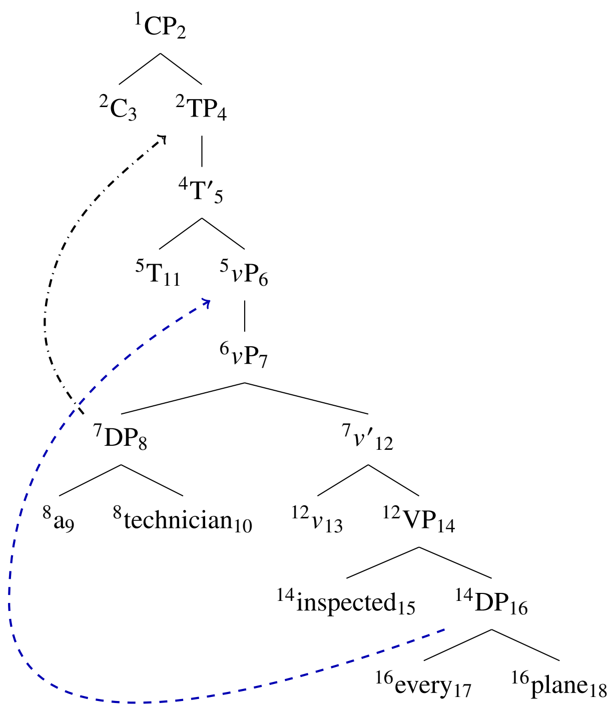
C.1 *Monoclausal (*A technician inspected every plane*)*

C.1.1 Surface scope with object QR

LEXICAL ENTRIES:

⟨(every, EVERY, `=N D -sc`$_{\text{L}}$)⟩	⟨(plane, PLANE, `N`)⟩
⟨(inspected, INSPECT, `=D V`)⟩	⟨(ε, V, `=V D= +sc`$_{\text{L}}$ `v`)⟩
⟨(a, A, `=N D -nom`)⟩	⟨(technician, TECHNICIAN, `N`)⟩
⟨(ε, T, `=v +nom T`)⟩	⟨(ε, C, `=T C`)⟩

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

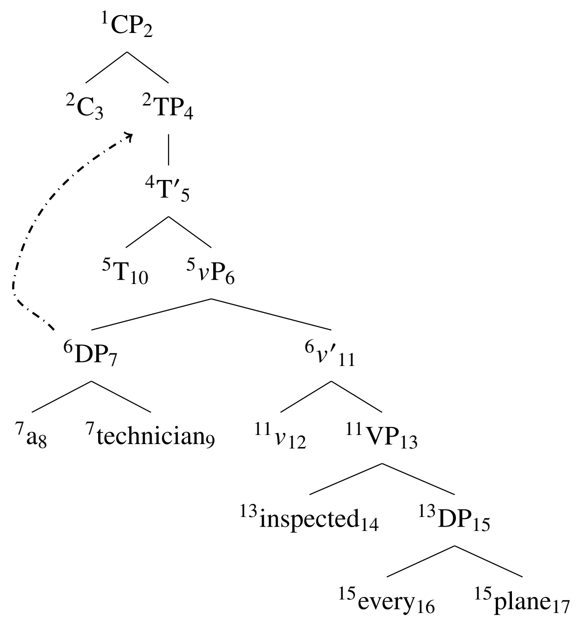
$(14-6) = \mathbf{8}$

C.1.2 Surface scope without object QR

LEXICAL ENTRIES:

⟨(every, EVERY, =N D)⟩	⟨(plane, PLANE, N)⟩
⟨(inspected, INSPECT, =D V)⟩	⟨(ε, V, =V D= v)⟩
⟨(a, A, =N D -nom)⟩	⟨(technician, TECHNICIAN, N)⟩
⟨(ε, T, =v +nom T)⟩	⟨(ε, C, =T C)⟩

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

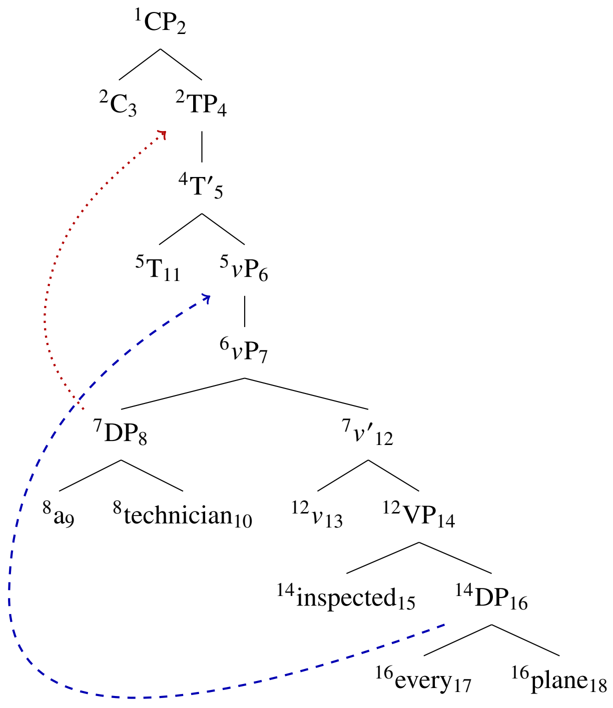
0

C.1.3 Inverse scope low

LEXICAL ENTRIES:

⟨(every, EVERY, =N D -sc_{L})⟩	⟨(plane, PLANE, N)⟩
⟨(inspected, INSPECT, =D V)⟩	⟨(ε, V, =V D= +sc_{L} v)⟩
⟨(a, A, =N D -nom_{P})⟩	⟨(technician, TECHNICIAN, N)⟩
⟨(ε, T, =v +nom_{P} T)⟩	⟨(ε, C, =T C)⟩

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

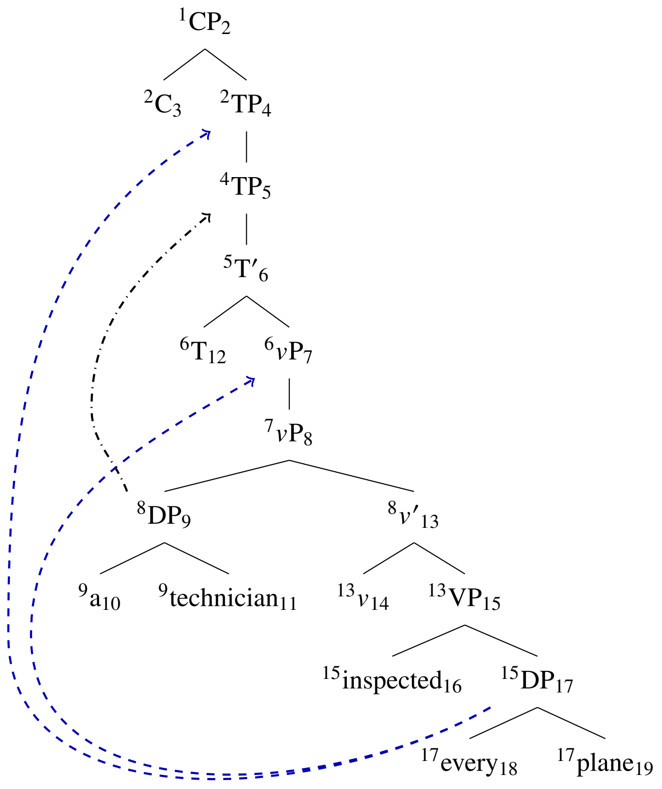
$$(7-4)+(14-6) = \mathbf{11}$$

C.1.4 Inverse scope high

LEXICAL ENTRIES:

$\langle$(every, EVERY, =N D -sc$_{\mathrm{L}}$ -sc$_{\mathrm{L}}$)$\rangle$	$\langle$(plane, PLANE, N)$\rangle$
$\langle$(inspected, INSPECT, =D V)$\rangle$	$\langle$(ε, V, =V D= +sc$_{\mathrm{L}}$ v)$\rangle$
$\langle$(a, A, =N D -nom)$\rangle$	$\langle$(technician, TECHNICIAN, N)$\rangle$
$\langle$(ε, T, =v +nom +sc$_{\mathrm{L}}$ T)$\rangle$	$\langle$(ε, C, =T C)$\rangle$

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

$(15-4) = \mathbf{11}$

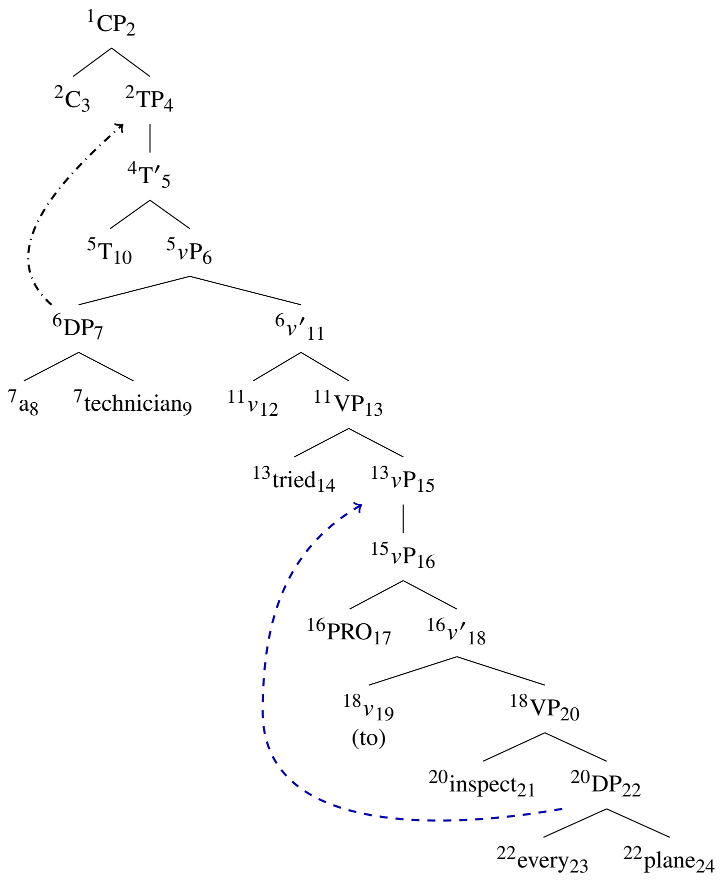
*C.2 try-type (*A technician tried to inspect every plane*)*

C.2.1 Surface scope with object QR

LEXICAL ENTRIES:

⟨(every, EVERY, =N D -sc$_L$)⟩	⟨(plane, PLANE, N)⟩	⟨(inspect, INSPECT, =D V)⟩
⟨(to, V, =V D= +sc$_L$ v)⟩	⟨(ε, PRO, D)⟩	⟨(tried, TRY, =v V)⟩
⟨(ε, V, =V D= v)⟩	⟨(a, A, =N D -nom)⟩	⟨(technician, TECHNICIAN, N)⟩
⟨(ε, T, =v +nom T)⟩	⟨(ε, C, =T C)⟩	

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

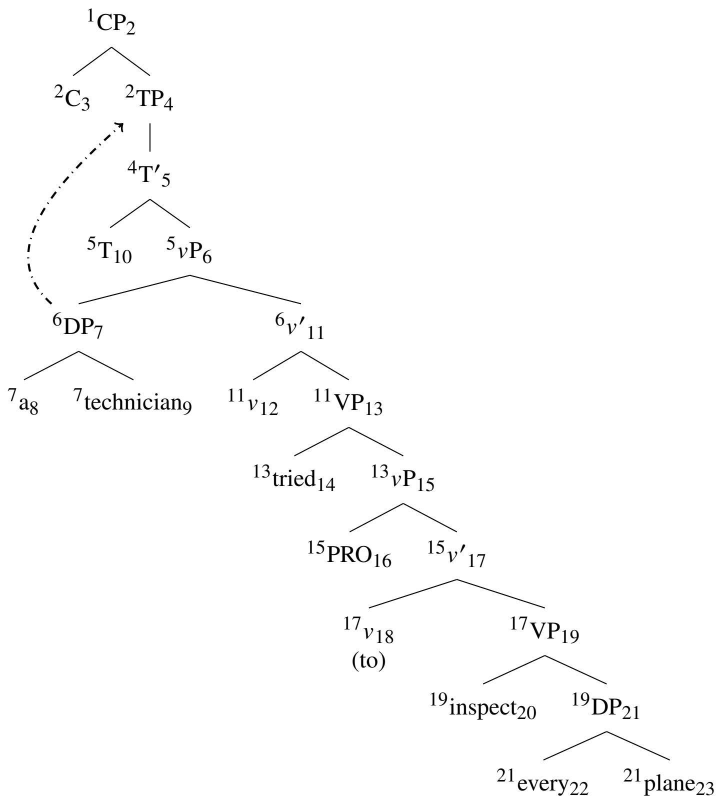
$(20-15)=\mathbf{5}$

C.2.2 Surface scope without object QR

LEXICAL ENTRIES:

⟨(every, EVERY, =N D)⟩	⟨(plane, PLANE, N)⟩	⟨(inspect, INSPECT, =D V)⟩
⟨(to, V, =V D= v)⟩	⟨(ε, PRO, D)⟩	⟨(tried, TRY, =v V)⟩
⟨(ε, V, =V D= v)⟩	⟨(a, A, =N D -nom)⟩	⟨(technician, TECHNICIAN, N)⟩
⟨(ε, T, =v +nom T)⟩	⟨(ε, C, =T C)⟩	

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

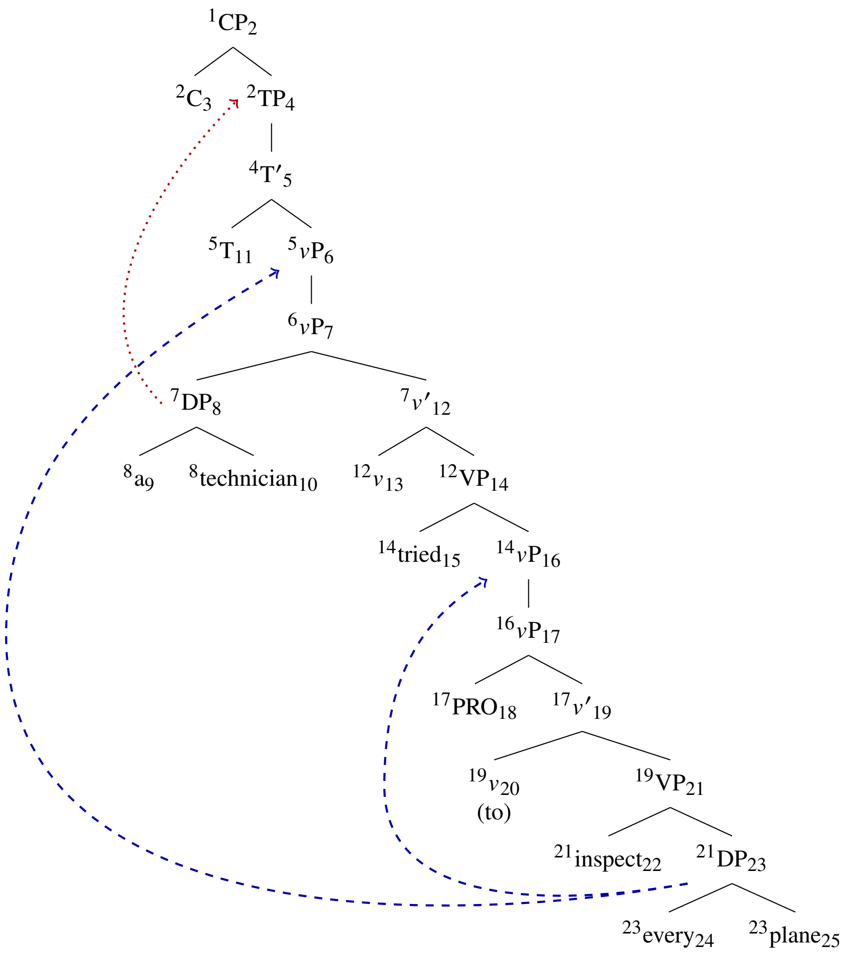
0

C.2.3 Inverse scope low

LEXICAL ENTRIES:

⟨(every, EVERY, =N D -sc$_{\mathrm{L}}$ -sc$_{\mathrm{L}}$)⟩	⟨(plane, PLANE, N)⟩	⟨(inspect, INSPECT, =D V)⟩
⟨(to, V, =V D= +sc$_{\mathrm{L}}$ v)⟩	⟨(ε, PRO, D)⟩	⟨(tried, TRY, =v V)⟩
⟨(ε, V, =V D= +sc$_{\mathrm{L}}$ v)⟩	⟨(a, A, =N D -nom$_{\mathrm{P}}$)⟩	⟨(technician, TECHNICIAN, N)⟩
⟨(ε, T, =v +nom$_{\mathrm{P}}$ T)⟩	⟨(ε, C, =T C)⟩	

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

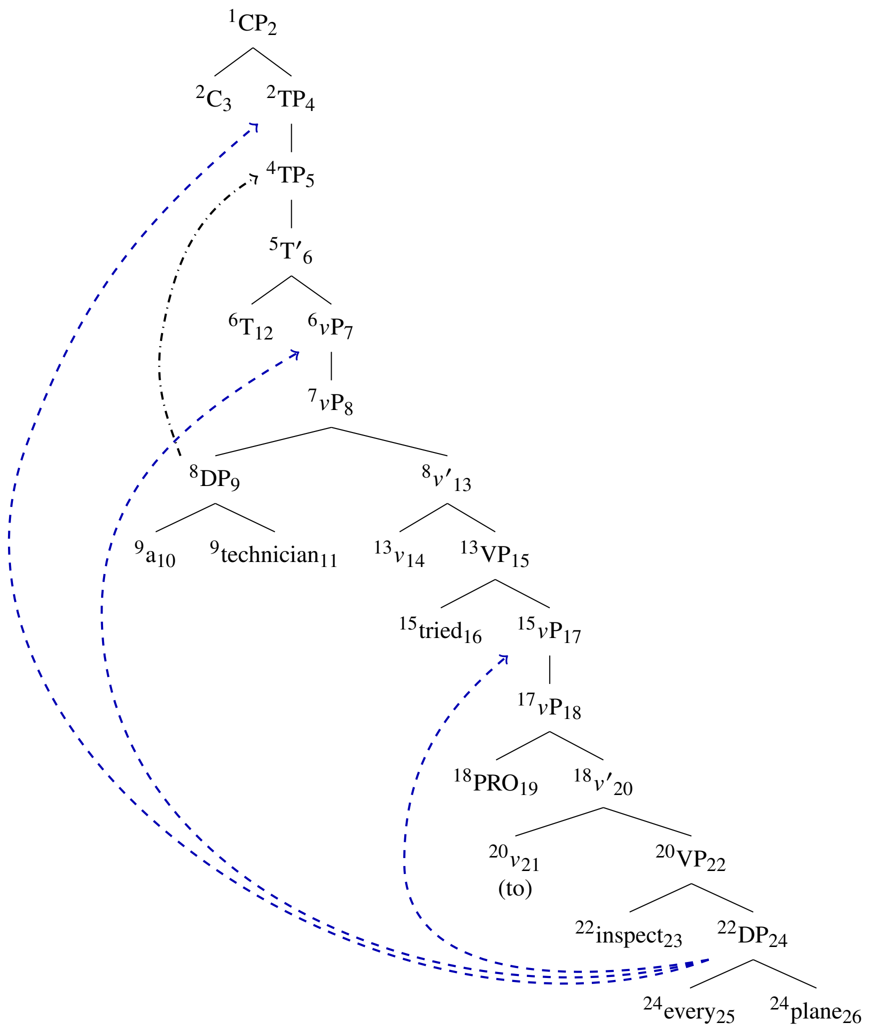
$$(7-4)+(21-6)=\mathbf{18}$$

C.2.4 Inverse scope high

LEXICAL ENTRIES:

$\langle$(every, EVERY, =N D -sc$_{\mathrm{L}}$ -sc$_{\mathrm{L}}$ -sc$_{\mathrm{L}}$)$\rangle$	$\langle$(plane, PLANE, N)$\rangle$	$\langle$(inspect, INSPECT, =D V)$\rangle$
$\langle$(to, V, =V D= +sc$_{\mathrm{L}}$ v)$\rangle$	$\langle$(ε, PRO, D)$\rangle$	$\langle$(tried, TRY, =v V)$\rangle$
$\langle$(ε, V, =V D= +sc$_{\mathrm{L}}$)$\rangle$	$\langle$(a, A, =N D -nom)$\rangle$	$\langle$(technician, TECHNICIAN, N)$\rangle$
$\langle$(ε, T, =v +nom +sc$_{\mathrm{L}}$ T)$\rangle$	$\langle$(ε, C, =T C)$\rangle$	

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

$(22-4) = \mathbf{18}$

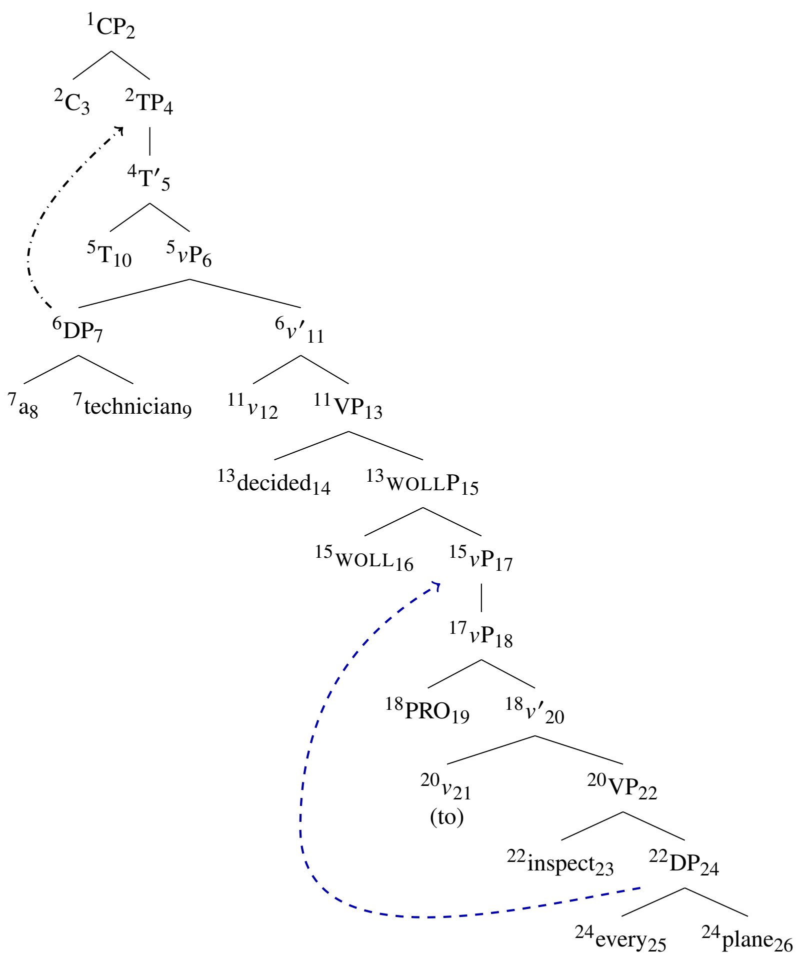
C.3 decide-*type* *(*A technician decided to inspect every plane*)*

C.3.1 Surface scope with object QR

LEXICAL ENTRIES:

⟨(every, EVERY, =N D -sc$_L$)⟩	⟨(plane, PLANE, N)⟩	⟨(inspect, INSPECT, =D V)⟩
⟨(to, V, =V D= +sc$_L$ v)⟩	⟨(ε, PRO, D)⟩	⟨(ε, WOLL, =v w)⟩
⟨(decided, DECIDE, =w V)⟩	⟨(ε, V, =V D= v)⟩	⟨(a, A, =N D -nom)⟩
⟨(technician, TECHNICIAN, N)⟩	⟨(ε, T, =v +nom T)⟩	⟨(ε, C, =T C)⟩

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

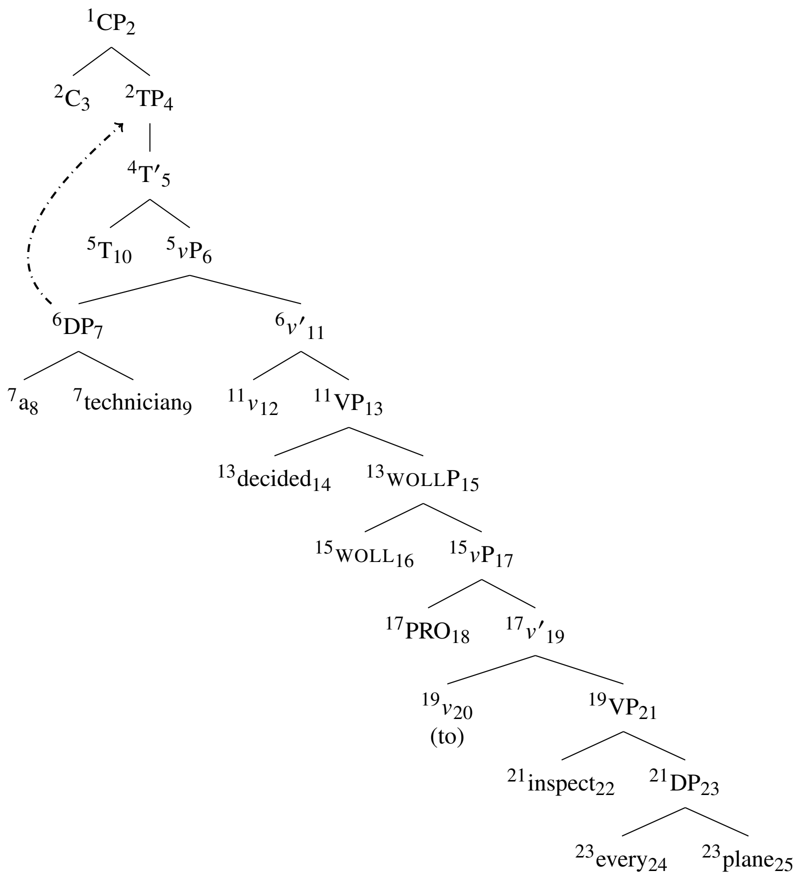
$(22 - 17) = \mathbf{5}$

C.3.2 Surface scope without object QR

LEXICAL ENTRIES:

⟨(every, EVERY, =N D)⟩	⟨(plane, PLANE, N)⟩	⟨(inspect, INSPECT, =D V)⟩
⟨(to, V, =V D= v)⟩	⟨(ε, PRO, D)⟩	⟨(ε, WOLL, =v w)⟩
⟨(decided, DECIDE, =w V)⟩	⟨(ε, V, =V D= v)⟩	⟨(a, A, =N D -nom)⟩
⟨(technician, TECHNICIAN, N)⟩	⟨(ε, T, =v +nom T)⟩	⟨(ε, C, =T C)⟩

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

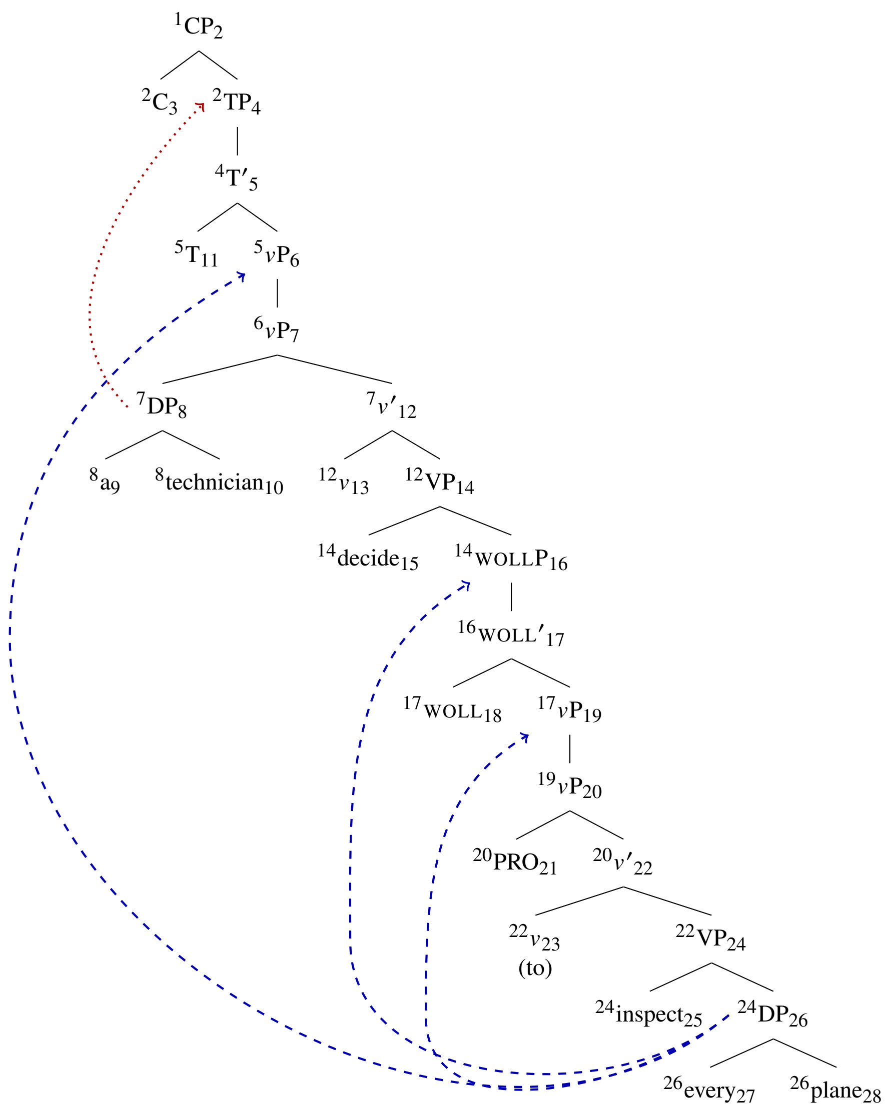
0

C.3.3 Inverse scope low, WOLLP a movement domain

LEXICAL ENTRIES:

⟨(every, EVERY, =N D -sc$_L$ -sc$_L$ -sc$_L$)⟩	⟨(plane, PLANE, N)⟩	⟨(inspect, INSPECT, =D V)⟩
⟨(to, V, =V D= +sc$_L$ v)⟩	⟨(ε, PRO, D)⟩	⟨(ε, WOLL, =v +sc$_L$ w)⟩
⟨(decided, DECIDE, =w V)⟩	⟨(ε, V, =V D= +sc$_L$ v)⟩	⟨(a, A, =N D -nom$_P$)⟩
⟨(technician, TECHNICIAN, N)⟩	⟨(ε, T, =v +nom$_P$ T)⟩	⟨(ε, C, =T C)⟩

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

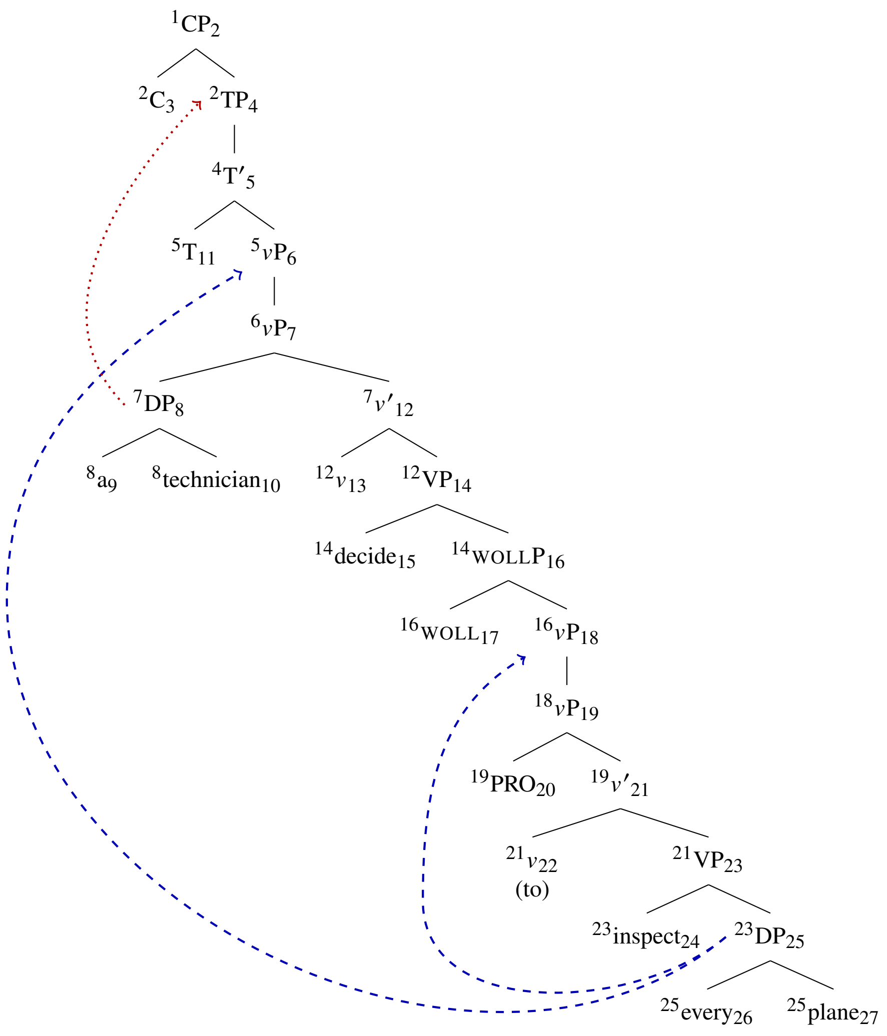
$$(7-4)+(24-6)=\mathbf{21}$$

C.3.4 Inverse scope low, WOLLP not a movement domain

LEXICAL ENTRIES:

⟨(every, EVERY, =N D -sc$_L$ -sc$_L$)⟩	⟨(plane, PLANE, N)⟩	⟨(inspect, INSPECT, =D V)⟩
⟨(to, V, =V D= +sc$_L$ v)⟩	⟨(ε, PRO, D)⟩	⟨(ε, WOLL, =v w)⟩
⟨(decided, DECIDE, =w V)⟩	⟨(ε, V, =V D= +sc$_L$ v)⟩	⟨(a, A, =N D -nom$_P$)⟩
⟨(technician, TECHNICIAN, N)⟩	⟨(ε, T, =v +nom$_P$ T)⟩	⟨(ε, C, =T C)⟩

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

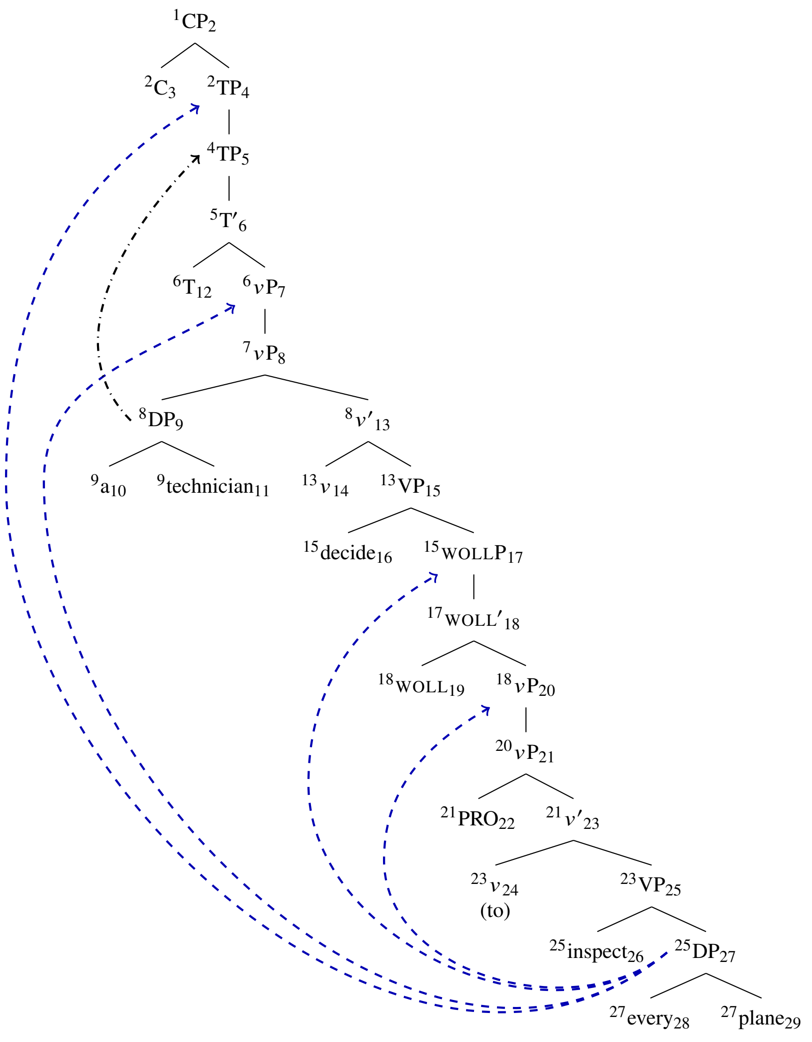
$$(7-4)+(23-6)=\mathbf{20}$$

C.3.5 Inverse scope high, WOLLP a movement domain

LEXICAL ENTRIES:

⟨(every, EVERY, =N D -sc$_{\text{L}}$ -sc$_{\text{L}}$ -sc$_{\text{L}}$ -sc$_{\text{L}}$)⟩	⟨(plane, PLANE, N)⟩	⟨(inspect, INSPECT, =D V)⟩
⟨(to, V, =V D= +sc$_{\text{L}}$ v)⟩	⟨(ε, PRO, D)⟩	⟨(ε, WOLL, =v +sc$_{\text{L}}$ w)⟩
⟨(decided, DECIDE, =w V)⟩	⟨(ε, V, =V D= +sc$_{\text{L}}$ v)⟩	⟨(a, A, =N D -nom)⟩
⟨(technician, TECHNICIAN, N)⟩	⟨(ε, T, =v +nom +sc$_{\text{L}}$ T)⟩	⟨(ε, C, =T C)⟩

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

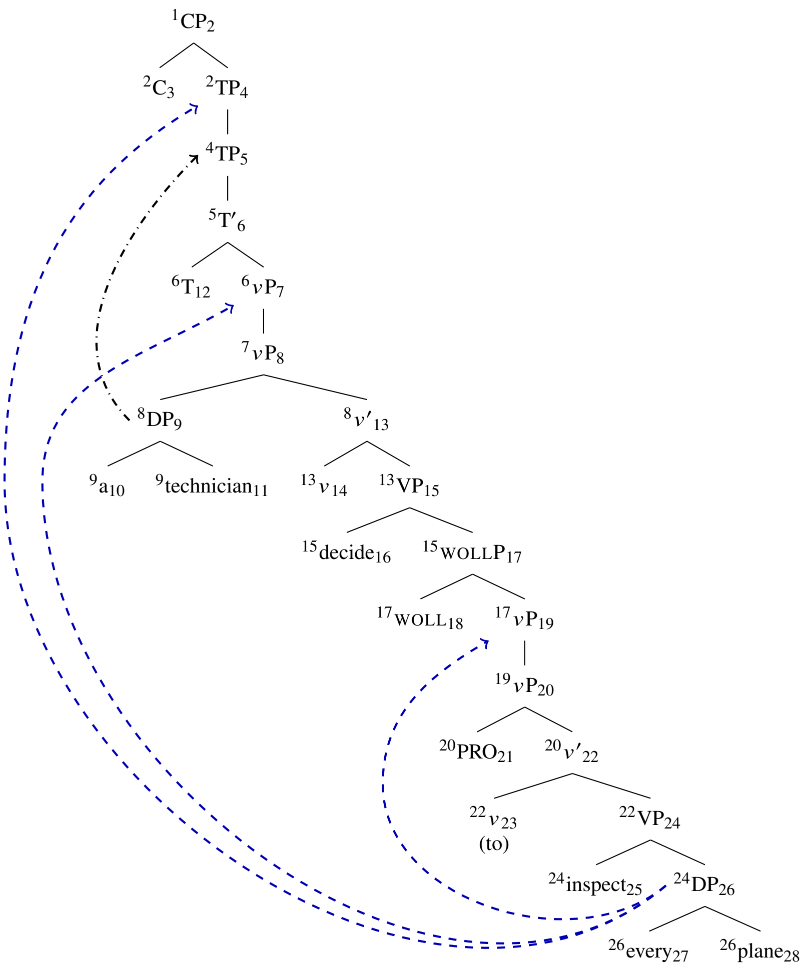
$(25 - 4) = \mathbf{21}$

C.3.6 Inverse scope high, WOLLP not a movement domain

LEXICAL ENTRIES:

⟨(every, EVERY, `=N D -sc`$_L$ `-sc`$_L$ `-sc`$_L$)⟩	⟨(plane, PLANE, `N`)⟩	⟨(inspect, INSPECT, `=D V`)⟩
⟨(to, V, `=V D= +sc`$_L$ `v`)⟩	⟨(ε, PRO, `D`)⟩	⟨(ε, WOLL, `=v w`)⟩
⟨(decided, DECIDE, `=w V`)⟩	⟨(ε, V, `=V D= +sc`$_L$ `v`)⟩	⟨(a, A, `=N D -nom`)⟩
⟨(technician, TECHNICIAN, `N`)⟩	⟨(ε, T, `=v +nom +sc`$_L$ `T`)⟩	⟨(ε, C, `=T C`)⟩

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

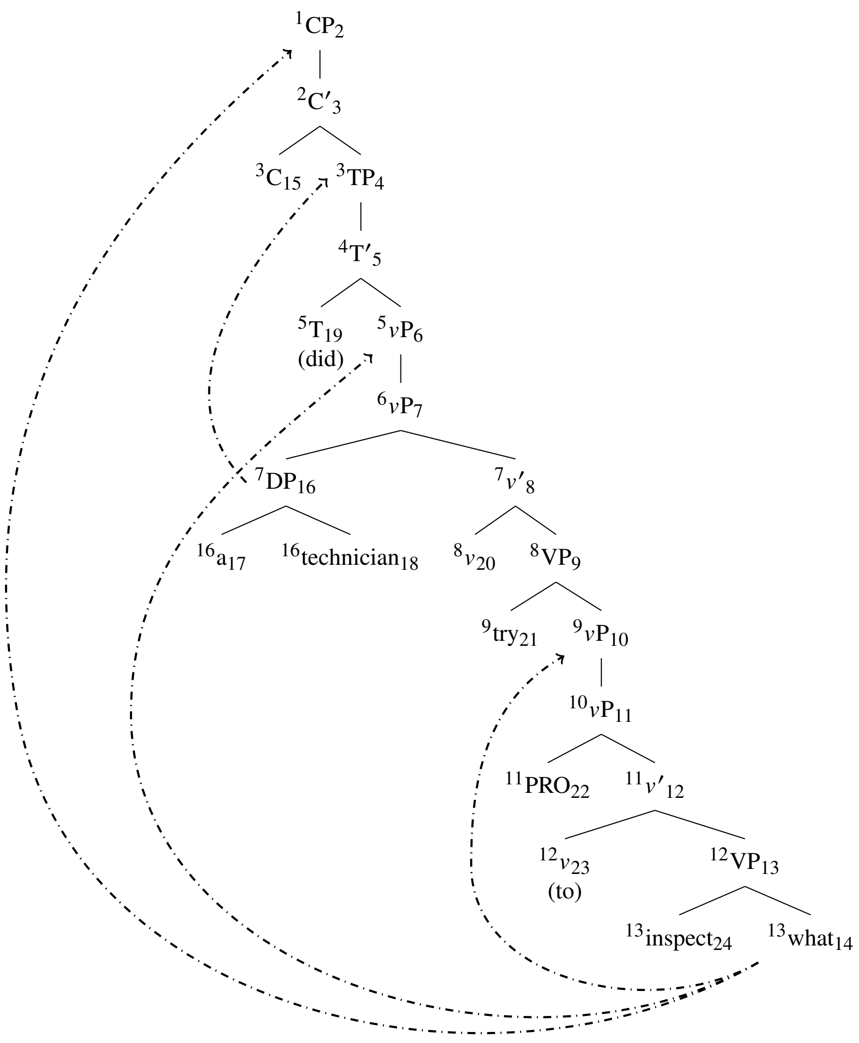
$(24-4) = \mathbf{20}$

C.4 wh-*movement*

LEXICAL ENTRIES:

⟨(what, WHAT, D -wh -wh -wh)⟩	⟨(inspect, INSPECT, =D V)⟩	⟨(to, V, =V D= +wh v)⟩
⟨(ε, PRO, D)⟩	⟨(try, TRY, =v V)⟩	⟨(ε, V, =V D= +wh v)⟩
⟨(a, A, =N D -nom)⟩	⟨(technician, TECHNICIAN, N)⟩	⟨(did, T, =v +nom T)⟩[2]
⟨(ε, C, =T +wh C)⟩		

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

0

[2] Because on our analysis T is pronounced *did*, and because for simplicity's sake we do not include the standard T-to-C head movement, the generated string is actually *What a technician did try to inspect*. This can of course be fixed by introducing head movement, something that is known to be non-problematic for normal MGs.

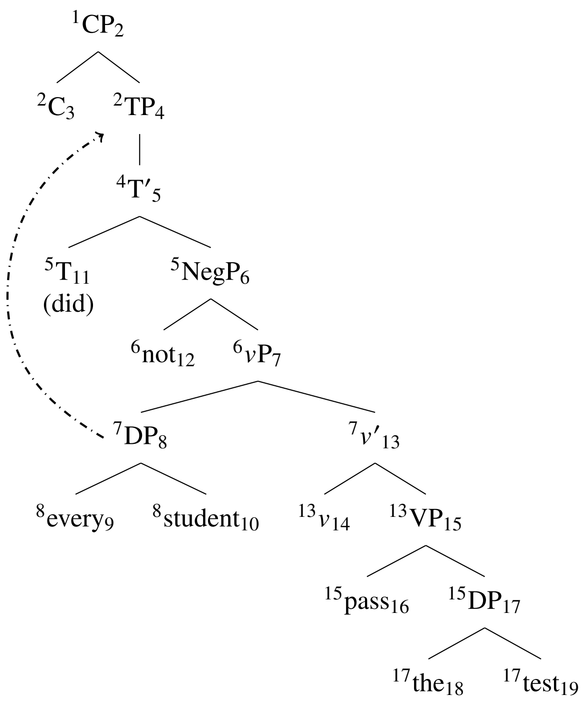
C.5 *Subject vs. negation (*Every student did not pass the test*)*

C.5.1 Surface scope

LEXICAL ENTRIES:

⟨(the, THE, `=N D`)⟩	⟨(test, TEST, `N`)⟩	⟨(pass, PASS, `=D V`)⟩
⟨(ε, V, `=V D= v`)⟩	⟨(every, EVERY, `=N D -nom`)⟩	⟨(student, STUDENT, `N`)⟩
⟨(not, NEG, `=v Neg`)⟩	⟨(did, T, `=Neg +nom T`)⟩	⟨(ε, C, `=T C`)⟩

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

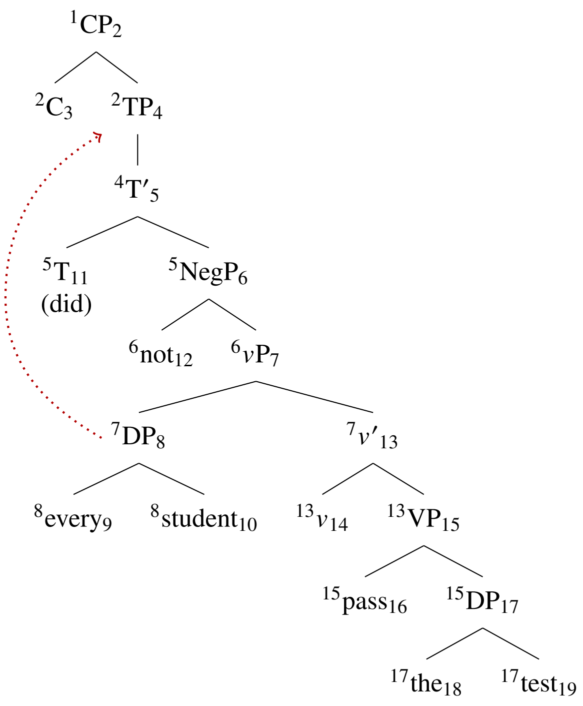
0

C.5.2 Inverse scope

LEXICAL ENTRIES:

⟨(the, THE, `=N D`)⟩	⟨(test, TEST, `N`)⟩	⟨(pass, PASS, `=D V`)⟩
⟨(ε, V, `=V D= v`)⟩	⟨(every, EVERY, `=N D -nom`$_P$)⟩	⟨(student, STUDENT, `N`)⟩
⟨(not, NEG, `=v Neg`)⟩	⟨(did, T, `=Neg +nom`$_P$ `T`)⟩	⟨(ε, C, `=T C`)⟩

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

$(7-4) = \mathbf{3}$

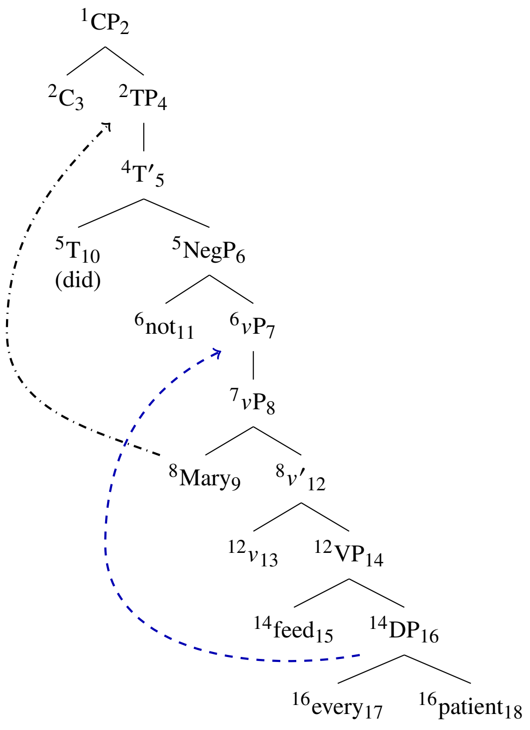
*C.6 Object vs. negation (*Mary did not feed every patient*)*

C.6.1 Surface scope with object QR

LEXICAL ENTRIES:

⟨(every, EVERY, `=N D -sc`$_L$)⟩	⟨(patient, PATIENT, `N`)⟩	⟨(feed, FEED, `=D V`)⟩
⟨(ε, V, `=V D= +sc`$_L$ `v`)⟩	⟨(Mary, MARY, `D -nom`)⟩	⟨(not, NEG, `=v Neg`)⟩
⟨(did, T, `=Neg +nom T`)⟩	⟨(ε, C, `=T C`)⟩	

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

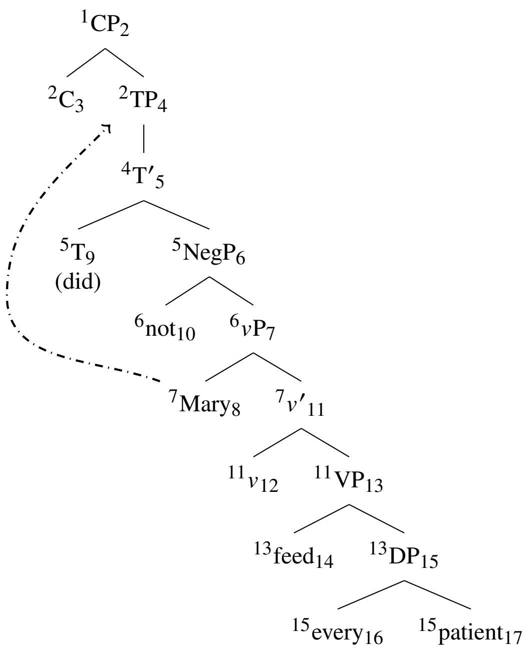
$(14-7)=\mathbf{7}$

C.6.2 Surface scope without object QR

LEXICAL ENTRIES:

⟨(every, EVERY, `=N D`)⟩	⟨(patient, PATIENT, `N`)⟩	⟨(feed, FEED, `=D V`)⟩
⟨(ε, V, `=V D= v`)⟩	⟨(Mary, MARY, `D -nom`)⟩	⟨(not, NEG, `=v Neg`)⟩
⟨(did, T, `=Neg +nom T`)⟩	⟨(ε, C, `=T C`)⟩	

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

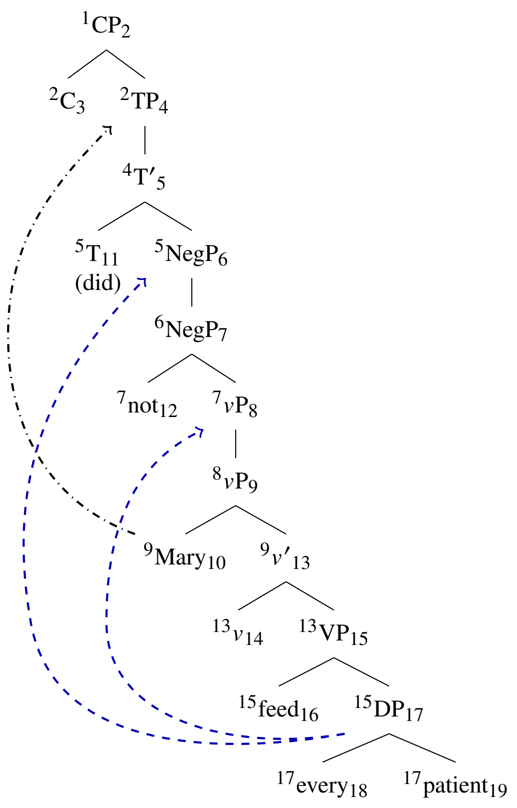
0

C.6.3 Inverse scope

LEXICAL ENTRIES:

⟨(every, EVERY, =N D -sc$_\text{L}$ -sc$_\text{L}$)⟩	⟨(patient, PATIENT, N)⟩	⟨(feed, FEED, =D V)⟩
⟨(ε, V, =V D= +sc$_\text{L}$ v)⟩	⟨(Mary, MARY, D -nom)⟩	⟨(not, NEG, =v +sc$_\text{L}$ Neg)⟩
⟨(did, T, =Neg +nom T)⟩	⟨(ε, C, =T C)⟩	

ANNOTATED DERIVATION TREE:



SUMMED LOCATION DIFFERENTIAL (SLD):

$(15 - 6) = \mathbf{9}$

C.7 Summary

C.7.1 Cyclic QR

	surface, object QR	**surface, no object QR**	**inverse, low**	**inverse, high**
monoclausal	8	0	11	11
***try*-type**	5	0	18	18
***decide*-type**	5	0	21/20	21/20

C.7.2 Quantifiers vs. negation

	surface, object QR	**surface, no object QR**	**inverse**
subject	N/A	0	3
object	7	0	9

References

Heim, Irene & Angelika Kratzer. 1998. *Semantics in generative grammar*. Oxford: Blackwell.

Kobele, Gregory M., Sabrina Gerth & John T. Hale. 2013. Memory resource allocation in top-down Minimalist parsing. In Glyn Morrill & Mark-Jan Nederhof (eds.), *Formal grammar: 17th and 18th international conferences*, 32–51.

Pereira, Fernando C.N. & David Warren. 1983. Parsing as deduction. In *21st annual meeting of the association for computational linguistics*, 137–144. Cambridge, MA: MIT.

Stabler, Edward & Edward Keenan. 2003. Structural similarity within and among languages. *Theoretical Computer Science* 293(2). 345–363.