Inverse Linking in Minimalist Grammars

Gregory M. Kobele

Institut für deutsche Sprache und Linguistik
Humboldt-Universität zu Berlin

Abstract

The phenomenon of Inverse Linking has proven challenging for theories of the syntax-semantics interface; a noun phrase within another behaves with respect to binding as though it were structurally independent. In this paper I show that, even within an LF-movement style approach to the syntax-semantics interface, we can derive all and only the appropriate meanings for such constructions using no semantic operations other than function application and composition. The solution relies not on a proliferation of lexical ambiguity, but rather on a straightforward (and standard) reification of assignment functions, which allows us to define abstraction operators within our models.

1 Introduction

Inverse linking (see May and Bale (2005) for an historical overview) refers to the phenomenon of a quantificational noun phrase (QNP) embedded as the argument of a prepositional phrase attached to another QNP taking semantic scope over that noun phrase, as in reading b of example 1, the gross surface structure of which is as sketched in figure 1.

(1) At least two senators on every committee voted against the bill.

a. At least two senators who are on every committee voted against the bill.

b. For every committee, there are at least two senators on that committee who voted against the bill.

The challenge the simple existence of inversely linked readings poses for a theory of grammar is to account for how the embedded PP-internal QNP is

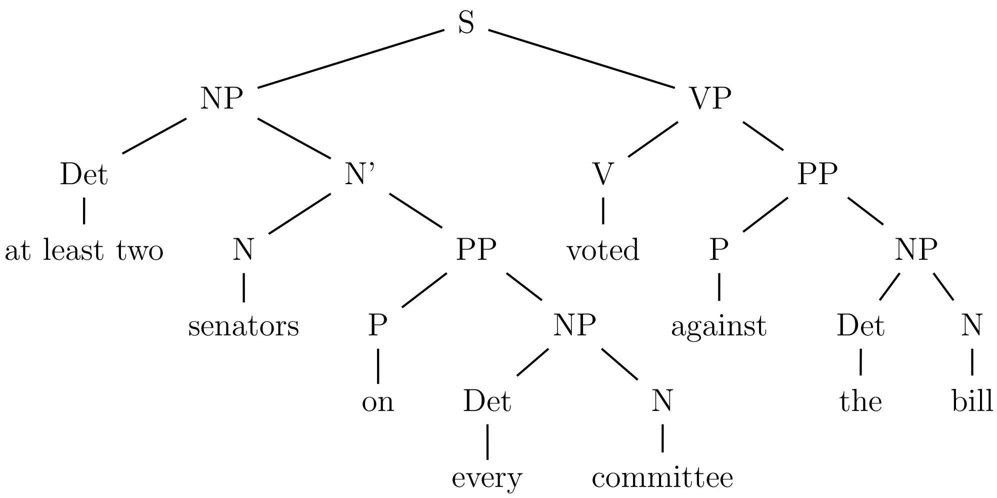


Figure 1: The gross surface structure of example 1

able to semantically outscope the noun phrase which contains it. There are two major classes of analyses.[1] The first option (May, 1977; Sauerland, 2005), sketched in figure 2, is to interpret the embedded QNP as taking sentential scope (i.e. as scoping entirely outside of the QNP which contains it). The second option (May, 1985; Larson, 1985; Heim and Kratzer, 1998), as in figure 3, is to interpret the embedded QNP as forming a complex quantificational element with the QNP which contains it. There are two additional empirical constraints on a theory of inverse linking. First is the fact, called *Larson's generalization* in May and Bale (2005), that quantificational noun phrases external to the containing QNP cannot intervene semantically between the embedded and containing QNPs, as shown in example 2.

(2) Two politicians spy on someone from every city.

If the sentence in 2 had a reading in which *every city* took widest scope, and *someone* narrowest, with *two politicians* in between, then it should be true of a situation in which each city has different politicians ($\forall < 2$), but where no two politicians spy on the same individual ($2 < \exists$). However, it doesn't seem to be.

[1] Another alternative is offered by continuations (see, for example,Barker (2002)). Syntactic movement, as presented in minimalist grammars, has at the very least intuitive relations to continuations (of the bounded variety (of which there are many)), but the analysis of quantification presented in Barker (2002) is not a natural fit with the syntactic analyses in the minimalist program, even if and when this latter is reanalyzed in terms of (string-)continuations. The same is true of analyses written in the context of syntactic frameworks, such as categorial grammar, in which type changing operations are more natural than in the minimalist framework used here.

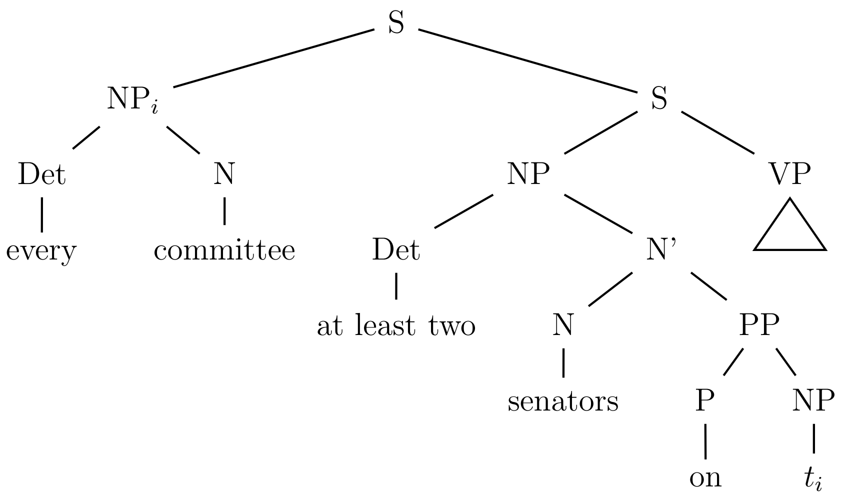


Figure 2: Explaining inverse linking: Sentential scope

The second empirical constraint on a theory of inverse linking comes from the fact that pronouns external to the containing QNP can be bound by the embedded QNP on the inversely linked reading, as in example 3.

(3) Some man from every city secretly despises it.

Example 3 can be true of a situation in which the city despised varies across individuals, i.e. Garrison from Lake Wobegon secretly despises Lake Wobegon, Garth from Waco secretly despises Waco, Gary from Wasilla secretly despises Wasilla, etc.

The sentential scope approach to inverse linking (as in figure 2) immediately accounts for the ability of embedded QNPs to bind variables external to their containing QNP, but requires additional stipulations to correctly rule out cases of scope-splitting. The complex quantifier approach to inverse linking (as in figure 3) on the other hand, while seeming as though it may provide a simple explanation of the scope-splitting prohibition, requires that some method of interpreting 'complex quantifiers' be specified before it is able to make predictions. What would seem to be needed is the following, where $\mathcal{Q}$ is the embedded QNP denotation, and $D(N(x))$ is the containing QNP denotation, where x is the interpretation of the trace of the embedded QNP.[2]

$$\lambda A_{et}.\mathcal{Q}(\lambda x_e.D(N(x))(A))$$

[2] Thus, the denotation of the NP in figure 3 should be the following.

$$\lambda A_{et}.\mathbf{every}(\mathbf{committee})(\lambda x_e.\mathbf{two}(\mathbf{senator} \wedge \mathbf{on}(x))(A))$$

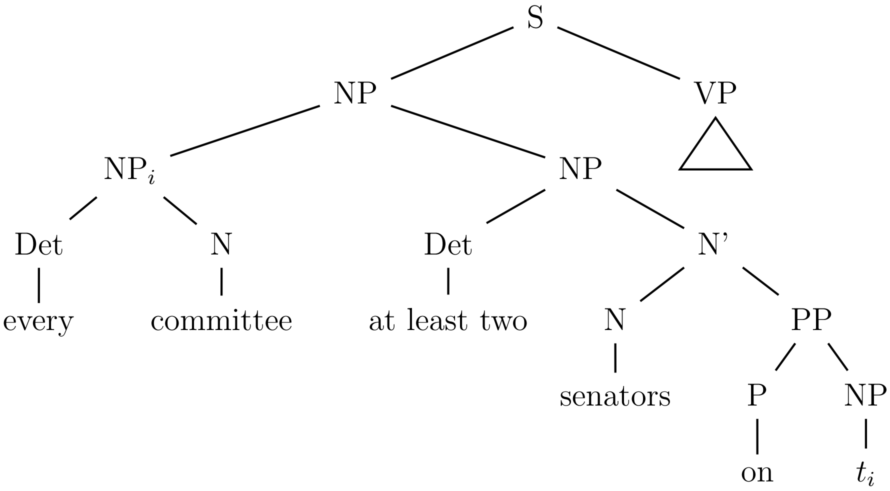


Figure 3: Explaining inverse linking: Complex quantifier formation

As can be verified by inspecting the term above, this denotation correctly rules out the scope splitting cases discussed above (see example 2). It also allows for the embedded quantifier (denoting $\mathcal{Q}$) to take scope over the rest of the sentence (replacing the variable A), which we want in order to account for the binding facts. However, the representation here *doesn't* allow pronouns in the sentence to be bound by $\mathcal{Q}$, as the λ-calculus prohibits 'variable capture.'[3] Thus, either pronoun binding in these sentences must not be identified with variable binding (an option might be something like E-type anaphora), or we need to find a way of implementing this idea without using the λ-calculus. Another problem with fleshing out of the complex quantifier approach to inverse linking in this way is that it seems to require construction specific semantic machinery for its description: how does the meaning of the embedded QNP, $\mathcal{Q}$ combine with the meaning of the containing QNP, $D(N(x))$, so as to result in the (or something like the) meaning representation above?[4]

[3] Λ-terms provide an overly fine grained representation of functions, in the sense that infinitely many distinct terms can represent the same function. An example: $\lambda x.x$ is a different term than $\lambda y.y$, although they stand for the very same function. β-conversion is defined so as to make sure that, if α and α' are terms denoting the same function, then $\alpha(\beta)$ and $\alpha'(\beta)$ also denote the same object. For example, $\lambda x.\lambda y.x$ and $\lambda x.\lambda z.x$ both denote the function $\mathbf{K}(a)(b) = a$, and so $(\lambda x.\lambda y.x)(y)$ must denote the same function as $(\lambda x.\lambda z.x)(y)$, namely the function denoted by $\lambda x.y$. For this reason, variable capture is ruled out, as it, being sensitive to accidental properties of our representations, disrupts the equivalences between terms that the λ-calculus enforces.

[4] An unsatisfying solution is to posit rampant otherwise unjustified lexical ambiguity.

The solution I will propose in §3 is that the mechanism underlying complex quantifier formation is simply function composition. Mixing meta- and object-language for the moment, the complex quantifier *a friend of every senator* will have the following form, where $\circ$ denotes function composition.

$$(\mathbf{every}(\mathbf{senator}) \circ \lambda y) \circ \mathbf{some}(\lambda x.\mathbf{friend}(x, y))$$

Applied to a predicate denotation, A, this will yield the following.

$$\begin{aligned}&((\mathbf{every}(\mathbf{senator}) \circ \lambda y) \circ \mathbf{some}(\lambda x.\mathbf{friend}(x, y)))(A)\\ &\qquad = (\mathbf{every}(\mathbf{senator}) \circ \lambda y)(\mathbf{some}(\lambda x.\mathbf{friend}(x, y))(A))\\ &\qquad = \mathbf{every}(\mathbf{senator})(\lambda y.\mathbf{some}(\lambda x.\mathbf{friend}(x, y))(A))\end{aligned}$$

This provides a simple solution to the ability of inversely scoped quantifiers to bind into the scope argument of their containing QNP; at the point at which the scope argument is introduced, the variable binding of the inversely scoping QNP has not yet been performed.

Strictly speaking, however, the formula written above is nonsense: λ-abstraction is not a function in our model, and thus it cannot be 'composed' with something that is. This, however, is due to the fact that it has become common to think of assignment functions as parameterizing model-theoretic interpretation, rather than, equivalently, as part of the models themselves. In §2 I review how to reify assignment functions, and interpret predicate logic, and dynamic predicate logic in these terms. This allows me to find a 'lambda abstraction' function in our models, which I use in §3 to make legitimate the solution outlined above. In contrast to the solution put forth in Sternefeld (1997), mine does not treat the binding relation in inverse-linking constructions dynamically. In the next section (§4) I provide an interpreted grammar fragment written in the minimalist grammar formalism, which assigns appropriate meanings to sentences with complex DP quantifiers. I show that the compositional semantic interpretation proposed herein, coupled with natural syntactic assumptions, derives Larson's generalization. Section 5 is the conclusion.

2 The Status of Assignment Functions

It is standard in presentations of the semantics of first-order logic to treat formulae as having interpretations in the models only with respect to assignment functions. In other words, there is no single monolithic interpretation function $[\![\cdot]\!]_{\mathcal{M}}$, but instead a family of interpretation functions $([\![\cdot]\!]^{g}_{\mathcal{M}})_{g\in G}$ parameterized by assignments. An equivalent perspective reifies the family G

of assignment functions, allowing there to be a single interpretation function $[\![\cdot]\!]_{\mathcal{M}}$, the relation of which to the family $([\![\cdot]\!]^{g}_{\mathcal{M}})_{g\in G}$ can be schematized as per the below.

$$[\![\phi]\!]_{\mathcal{M}} := \{g : [\![\phi]\!]^{g}_{\mathcal{M}} = \texttt{true}\}$$

In other words, this alternative perspective takes the meaning of a sentence ϕ to be the set of all assignment functions with respect to which ϕ is true on the standard perspective. Despite being equivalent from the perspective of entailment, moving the assignment functions into the model gives access to first rate model theoretic objects which behave like lambda abstraction, but which can be composed with, among other things, generalized quantifier denotations to yield semantic objects of the kind alluded to at the end of the previous section.

In order to familiarize the reader with this perspective, I use the remainder of this section to present a semantics for predicate logic and (a slightly modified version of) dynamic predicate logic in these terms. Subsection 2.1 introduces the notation used in further subsections, §2.2 presents the standard syntax and (this alternative perspective on the standard) semantics of predicate logic, which is in §2.3 modified so as to recast sentential quantifiers (of type tt) as generalized quantifiers (of type $(et)t$), and introduce a (restricted) lambda abstraction operator. The syntax presented in §2.3 is carried over to §2.4, where it is fitted with a dynamic interpretation.

2.1 Models

Our models are built from the following objects:

- E is the set of *entities*
- T is the set of *propositions*
- G is the set of *contexts of use*

Here, I will take T to be the set $\{0, 1\}$ of truth values, and G to be the set $E^{\mathbb{N}}$ of assignment functions, where $\mathbb{N} = \{0, 1, 2, \ldots\}$ is the set of natural numbers and B^A the set of functions with domain A and codomain B.[5] Given $g, h \in G$, I write g_i for $g(i)$, and $g \approx_i h$ is true just in case if g and h differ, then only in the value they take at i (i.e. for any j, if $g_j \neq h_j$ then $j = i$). The notation $g^{[i:=a]}$ denotes the assignment h, such that $h_i = a$, and $g \approx_i h$. I write $x \in y$ as an abbreviation for $y(x) = 1$.

[5] To deal with intensionality, we can instead take $T = \{0, 1\}^W$, for W an arbitrary set (of *possible worlds*) (Keenan and Faltz, 1985). Furthermore, as will be discussed later, we can take $G = E^{\mathbb{N}} \times E^{\mathbb{N}}$ to deal with dynamic binding (Groenendijk and Stokhof, 1991).

Natural language expressions denote in domains built from these sets in the following way.

- $D_e := E^G$
- $D_t := T^G$
- $D_{\alpha\beta} := {D_\beta}^{D_\alpha}$

My use of G is thus similar to Montague's type s, as used in Montague (1970). Viewed from this perspective, my type e is Montague's type se (individual concepts), and my type t Montague's type st (propositions). The difference between my use of G and Montague's s is that the 'denotation domain' of type s was for Montague not just the set of assignment functions (as it is here), but rather the set of pairs of assignment functions and possible worlds.[6]

I will call functions in the set D_e *individuals*, those in D_t *sets of assignments*, functions from individuals to sets of assignments *properties*, functions from properties to sets of assignments *generalized quantifiers*, and functions from properties to generalized quantifiers *determiners*. I will also call sets of assignments *nullary relations*, and functions from individuals to n-ary relations *n+1-ary relations* (and so properties are unary relations).

2.2 Predicate Logic

I give an interpretation of a simple fragment of the predicate calculus, with individual constants john, mary, one-place predicate constant giggle, and two-place predicate constant tickle. In addition, our language contains a countably infinite set of variable symbols $\textsc{Var} = \{x_0, x_1, x_2, \ldots\}$, and the logical symbols $\exists$, $\forall$, &, and $\neg$, as well as the punctuation symbols '(' and ')'.

The well-formed formulae of our language are defined as follows.

1. if $v_1, \ldots, v_i$ are variables or individual constants, and r^i is an i-place predicate, then $r^i(v_1, \ldots, v_i)$ is a well-formed formula
2. if ϕ is a well-formed formula, so too is $\neg\phi$
3. if ϕ and ψ are well-formed formulae, so is $(\phi \,\&\, \psi)$
4. if ϕ is a well-formed formula, and x is a variable, then $\exists x.\phi$ is a well-formed formula

[6] In Montague (1973), the 'denotation domain' of type s was understood of pairs of worlds and times, with assignment functions relegated to parameters on the interpretation function.

5. if ϕ is a well-formed formula, and x is a variable, then $\forall x.\phi$ is a well-formed formula

A model $\mathcal{M} = \langle E, I \rangle$ provides an interpretation function I mapping elements of individual constants to elements of E, and i-ary relation symbols to i-ary relations over E. A variable x_i denotes a function $[\![x_i]\!]_{\mathcal{M}} \in D_e$ from G to E, such that $[\![x_i]\!]_{\mathcal{M}}(g) = g_i$. We extend $[\![\cdot]\!]_{\mathcal{M}}$ to our other constants in the following way:

- for c an individual constant, $[\![c]\!]_{\mathcal{M}} \in D_e$

$$[\![c]\!]_{\mathcal{M}}(g) = I(c)$$

- for r^i an i-place predicate constant, $[\![r^i]\!]_{\mathcal{M}} \in D_{e^i t}$

$$g \in [\![r^i]\!]_{\mathcal{M}}(a_1) \dots (a_i) \text{ iff } \langle a_1(g), \dots, a_i(g) \rangle \in I(r^i)$$

  The primary difference between this system and the one presented in Montague (1974) is lies in the denotation of predicates, which in Montague's system are functions with domain $E^i \rightarrow G \rightarrow T$, and which are here functions with domain $\left(E^G\right)^i \rightarrow G \rightarrow T$.

- Finally, (sentential) conjunction and negation are given the obvious constant interpretations:

$$[\![\neg]\!]_{\mathcal{M}}(H) := \{g : g \notin H\}$$
$$[\![\&]\!]_{\mathcal{M}}(H)(K) := \{g : g \in H \wedge g \in K\}$$

Applied to well-formed formulae, the function $[\![\cdot]\!]_{\mathcal{M}}$ assigns to them sets of assignments, and is defined on them in the following manner.

1. $[\![r(v_1, \dots, v_n)]\!]_{\mathcal{M}} = [\![r]\!]_{\mathcal{M}}([\![v_1]\!]_{\mathcal{M}}) \dots ([\![v_n]\!]_{\mathcal{M}})$
2. $[\![\neg\phi]\!]_{\mathcal{M}} = [\![\neg]\!]_{\mathcal{M}}([\![\phi]\!]_{\mathcal{M}})$
3. $[\![(\phi \,\&\, \psi)]\!]_{\mathcal{M}} = [\![\&]\!]_{\mathcal{M}}([\![\phi]\!]_{\mathcal{M}})([\![\psi]\!]_{\mathcal{M}})$
4. $[\![\exists x_i.\phi]\!]_{\mathcal{M}} = \{g : \text{for some } h \in G \text{ such that } g \approx_i h,\ h \in [\![\phi]\!]_{\mathcal{M}}\}$
5. $[\![\forall x_i.\phi]\!]_{\mathcal{M}} = \{g : \text{for all } h \in G \text{ such that } g \approx_i h,\ h \in [\![\phi]\!]_{\mathcal{M}}\}$

We determine the interpretation of the formula $\exists x_1.\mathsf{giggle}(x_1)$ as follows.

$$\begin{aligned}
g \in \llbracket \exists x_1.\mathsf{giggle}(x_1) \rrbracket_{\mathcal{M}} &\text{ iff for some } h \in G,\ g \approx_1 h \wedge h \in \llbracket \mathsf{giggle}(x_1) \rrbracket_{\mathcal{M}} \\
&\text{ iff for some } h \in G,\ g \approx_1 h \wedge h \in \llbracket \mathsf{giggle} \rrbracket_{\mathcal{M}}(\llbracket x_1 \rrbracket_{\mathcal{M}}) \\
&\text{ iff for some } h \in G,\ g \approx_1 h \wedge \llbracket x_1 \rrbracket_{\mathcal{M}}(h) \in I(\mathsf{giggle}) \\
&\text{ iff for some } h \in G,\ g \approx_1 h \wedge h_1 \in I(\mathsf{giggle})
\end{aligned}$$

As can be seen from the above, the denotation of the formula $\mathsf{giggle}(x_1)$ with free variable x_1 is the set of all assignments g, such that g_1 giggled, $\{g : g_1 \in I(\mathsf{giggle})\}$. Intuitively, given a set H of assignments (a sentence denotation), the semantic effect of existential quantification over a variable x_i is to enlarge H with all g which are i-variants of some $h \in H$ ($g \approx_i h$). The effect of universal quantification over x_i is to remove from H all g of which an i-variant is missing from H. Thus, given an arbitrary sentence Φ, we have that for every x, $\llbracket \forall x.\Phi \rrbracket_{\mathcal{M}} \subseteq \llbracket \Phi \rrbracket_{\mathcal{M}} \subseteq \llbracket \exists x.\Phi \rrbracket_{\mathcal{M}}$. Note that formulae without free variables denote either all of G or none of it ($\emptyset$), depending on whether they are true or not. Thus, given Φ and x such that x is not free in Φ, $\llbracket \forall x.\Phi \rrbracket_{\mathcal{M}} = \llbracket \Phi \rrbracket_{\mathcal{M}} = \llbracket \exists x.\Phi \rrbracket_{\mathcal{M}}$.

2.3 Defining Lambda Abstraction

Above, the quantifiers were introduced syncategorimatically, and thus received no independent meaning. Following Church (1940), I dissociate quantification from variable binding, by revising the syntax and semantics of our language so that quantifiers combine with one-place predicates instead of with sentences. Our language is accordingly extended with a countably infinite set of symbols $\leftthreetimes_0, \leftthreetimes_1, \leftthreetimes_2, \ldots$ I revise clauses 4 and 5 of our definition of well-formed formulae, and add a clause 6, which allows us to derive one-place predicates from sentences.

4. if ρ is a one-place predicate, then $\exists\rho$ is a well-formed formula

5. if ρ is a one-place predicate, then $\forall\rho$ is a well-formed formula

6. if ϕ is a well-formed formula, then $\leftthreetimes_i(\phi)$ is a one-place predicate

We assign to the symbols $\exists$ and $\forall$ the following meanings in $D_{(et)t}$. For every $A \in D_{et}$,

$$\begin{aligned}
\llbracket \exists \rrbracket_{\mathcal{M}}(A) &:= \{g : \text{for some } a \in D_e,\ g \in A(a)\} \\
\llbracket \forall \rrbracket_{\mathcal{M}}(A) &:= \{g : \text{for all } a \in D_e,\ g \in A(a)\}
\end{aligned}$$

The symbols $\leftthreetimes_i$ also denote logical constants (which I will in later sections write as λ_i), and in the type D_{tet}.[7] For any $H \in D_t$, and $a \in D_e$,

$$[\![\leftthreetimes_i]\!]_{\mathcal{M}}(H)(a) = \{g : g^{[i:=a(g)]} \in H\}$$

To the revised and newly introduced syntactic possibilities are associated the following semantic interpretation rules.

4. $[\![\exists\rho]\!]_{\mathcal{M}} = [\![\exists]\!]_{\mathcal{M}}([\![\rho]\!]_{\mathcal{M}})$
5. $[\![\exists\rho]\!]_{\mathcal{M}} = [\![\forall]\!]_{\mathcal{M}}([\![\rho]\!]_{\mathcal{M}})$
6. $[\![\leftthreetimes_i(\phi)]\!]_{\mathcal{M}} = [\![\leftthreetimes_i]\!]_{\mathcal{M}}([\![\phi]\!]_{\mathcal{M}})$

With these definitions, it can be proven that, for example,

$$[\![\leftthreetimes_1(\mathsf{giggle}(x_1))(x_2)]\!]_{\mathcal{M}} = [\![\mathsf{giggle}(x_2)]\!]_{\mathcal{M}}$$

$$\begin{aligned}
[\![\leftthreetimes_1(\mathsf{giggle}(x_1))(x_2)]\!]_{\mathcal{M}} &= \{g : g \in [\![\leftthreetimes_1(\mathsf{giggle}(x_1))]\!]_{\mathcal{M}}([\![x_2]\!]_{\mathcal{M}})\} \\
&= \{g : g \in [\![\leftthreetimes_1]\!]_{\mathcal{M}}([\![\mathsf{giggle}(x_1)]\!]_{\mathcal{M}})([\![x_2]\!]_{\mathcal{M}})\} \\
&= \{g : g \in \{h : h^{[1:=[\![x_2]\!]_{\mathcal{M}}(g)]} \in [\![\mathsf{giggle}(x_1)]\!]_{\mathcal{M}}\}\} \\
&= \{g : g^{[1:=g_2]} \in [\![\mathsf{giggle}(x_1)]\!]_{\mathcal{M}}\} \\
&= \{g : g^{[1:=g_2]} \in [\![\mathsf{giggle}]\!]_{\mathcal{M}}([\![x_1]\!]_{\mathcal{M}})\} \\
&= \{g : [\![x_1]\!]_{\mathcal{M}}(g^{[1:=g_2]}) \in I(\mathsf{giggle})\} \\
&= \{g : (g^{[1:=g_2]})_1 \in I(\mathsf{giggle})\} \\
&= \{g : g_2 \in I(\mathsf{giggle})\} \\
&= \{g : [\![x_2]\!]_{\mathcal{M}}(g) \in I(\mathsf{giggle})\} \\
[\![\mathsf{giggle}(x_2)]\!]_{\mathcal{M}} &= \{g : g \in [\![\mathsf{giggle}]\!]_{\mathcal{M}}([\![x_2]\!]_{\mathcal{M}})\}
\end{aligned}$$

[7] It is possible to give a definition for general 'lambda-abstraction', i.e. a single function λ, which then combines with a variable denotation $[\![x_i]\!]_{\mathcal{M}}$ to give us the function $\lambda_i = [\![\leftthreetimes_i]\!]_{\mathcal{M}}$ defined here, as shown below

$$\begin{aligned}
\lambda(b)(H)(a) := \{g : &\text{ for some } h \in H,\ b(h) = a(g), \text{ and either } h = g \text{ or there is exactly} \\
&\text{one } j \in \mathbb{N} \text{ such that } g \approx_j h, \text{ and } g_j = b(g) \text{ and } h_j = b(h)\}
\end{aligned}$$

When b in the above definition is $[\![x_i]\!]_{\mathcal{M}}$, we have that $h = g^{[i:=a(g)]}$. The benefit of such a move becomes evident once we see that we can eliminate all but one variable name (see Kobele (2006)), leaving us with a semantic algebra with a finite signature.

2.4 Dynamic Predicate Logic

Groenendijk and Stokhof (1991) observe that the treatment of anaphora across sentence boundaries in Discourse Representation Theory (Kamp and Reyle, 1993) can be modeled in standard predicate logic if interpretation is done relative to *two* assignments, one which serves as a representation of the context prior to the utterance of a sentence, and one which represents the result that that sentence's utterance had on the first context. Thus, the denotation of a sentence is taken to be a set of *pairs* of assignment functions (representing the 'context change potential' of that sentence), and so we view G as a set of pairs of assignments $E^{\mathbb{N}} \times E^{\mathbb{N}}$.[8]

We accordingly need to modify our definitions of the denotations and interpretations of the expressions of our language. Variables x_i and constants c can be thought of as simply ignoring the second element of the pair of assignments they are given: $[\![x_i]\!]_{\mathcal{M}}(\langle g, h\rangle) = g_i$, and $[\![c]\!]_{\mathcal{M}}(\langle g, h\rangle) = I(c)$. Because of this lack of dependence on the second component of their argument, I will continue to write $[\![x_i]\!]_{\mathcal{M}}(g)$ and $[\![c]\!]_{\mathcal{M}}(g)$. I-place predicate constants r^i continue to distribute their context argument (now a pair of assignments) to their semantic arguments:

$$\langle g, h\rangle \in [\![r^i]\!]_{\mathcal{M}}(a_1)\dots(a_i) \text{ iff } g = h \text{ and } \langle a_1(\langle g, h\rangle), \dots, a_i(\langle g, h\rangle)\rangle \in I(r^i)$$

The denotations of the remaining expressions in our language are given as per the following.

- $[\![\rho(v_1, \dots, v_n)]\!]_{\mathcal{M}} = [\![\rho]\!]_{\mathcal{M}}([\![v_1]\!]_{\mathcal{M}})\dots([\![v_n]\!]_{\mathcal{M}})$
- $[\![\neg]\!]_{\mathcal{M}}(H) = \{\langle g, h\rangle : g = h \text{ and for all } k,\ \langle g, k\rangle \notin H\}$
- $[\![\&]\!]_{\mathcal{M}}(H)(K) = \{\langle g, h\rangle : \text{for some } k,\ \langle g, k\rangle \in H \text{ and } \langle k, h\rangle \in K\}$
- $[\![\exists]\!]_{\mathcal{M}}(A) = \{\langle g, h\rangle : \text{for some } a \in D_e,\ \langle g, h\rangle \in A(a)\}$
- $[\![\forall]\!]_{\mathcal{M}}(A) =$

  $$\{\langle g, h\rangle : g = h \text{ and for all } a \in D_e,\ \text{for some } k,\ \langle g, k\rangle \in A(a)\}$$
- $[\![\lambda_i]\!]_{\mathcal{M}}(H)(a) = \{\langle g, h\rangle : \langle g^{[i:=a(g)]}, h\rangle \in H\}$

Note that from this perspective, dynamicity is always and only introduced by variable binding. We can define a set of assignment pairs to be *static* just in case it contains only pairs of the form $\langle g, g\rangle$. A function f from pairs of

[8] Note that this really is just a perspective shift, as $E^{\mathbb{N}} \cong E^{\mathbb{N}} \times E^{\mathbb{N}}$.

assignments is static just in case for any two assignments h, k, $f(\langle g, h\rangle) = f(\langle g, k\rangle)$. Thus, variables and individual constants are static. A function $f : A \to B$ is static just in case for any static $a \in A$, $f(a)$ is also static. With this definition, we see that λ_i is not static, as $\lambda_i(\{\langle g, g\rangle\})(\mathbf{x_1}) = \{\langle h, g\rangle : h \approx_i g\}$, which is not static, although the unit set $\{\langle g, g\rangle\}$ and $\mathbf{x_1}$ are. The quantifiers and logical connectives are static, but differ amongst themselves regarding whether they are permeable to dynamic effects (i.e. on whether they map even dynamic arguments to static values).

With these definitions, and treating sentence juxtaposition as conjunction, we can compute the meaning (qua context change potential) of the text "*John tickled someone. She giggled.*", rendered into our logical language as the conjunction of the sentences $\exists(\leftthreetimes_1(\mathsf{tickle}(x_1)(\mathsf{john})))$ and $\mathsf{giggle}(x_1)$. The context change potential of the first sentence is calculated to be the following.

$$\begin{aligned}
[\![\exists(\leftthreetimes_1(\mathsf{tickle}(x_1)(\mathsf{john})))]\!]_{\mathcal{M}} &= \{\langle g, h\rangle : \text{for some } a \in D_e, \\
&\qquad \langle g, h\rangle \in [\![\leftthreetimes_1(\mathsf{tickle}(x_1)(\mathsf{john}))]\!]_{\mathcal{M}}(a)\} \\
&= \{\langle g, h\rangle : \text{for some } a \in D_e, \\
&\qquad \langle g^{[1:=a(g)]}, h\rangle \in [\![\mathsf{tickle}(x_1)(\mathsf{john})]\!]_{\mathcal{M}}\} \\
&= \{\langle g, h\rangle : \text{for some } a \in D_e, \\
&\qquad \langle g^{[1:=a(g)]}, h\rangle \in [\![\mathsf{tickle}]\!]_{\mathcal{M}}([\![x_1]\!]_{\mathcal{M}})([\![\mathsf{john}]\!]_{\mathcal{M}}) \\
&\qquad \text{ and } g^{[1:=a(g)]} = h\} \\
&= \{\langle g, g^{[1:=a(g)]}\rangle : \text{for some } a \in D_e, \\
&\qquad \langle a(g), I(\mathsf{john})\rangle \in I(\mathsf{tickle})\}
\end{aligned}$$

The context change potential of our second sentence, which is easily seen to be the below, is interpreted in the context of the first sentence.

$$\begin{aligned}
[\![\mathsf{giggle}(x_1)]\!]_{\mathcal{M}} &= \{\langle g, h\rangle : \langle g, h\rangle \in [\![\mathsf{giggle}(x_1)]\!]_{\mathcal{M}}\} \\
&= \{\langle g, h\rangle : g = h \text{ and } \langle g, h\rangle \in [\![\mathsf{giggle}]\!]_{\mathcal{M}}([\![x_1]\!]_{\mathcal{M}})\} \\
&= \{\langle g, g\rangle : g_1 \in I(\mathsf{giggle})\}
\end{aligned}$$

After processing the first sentence, the context is modified so as to assign $a(g)$ to x_1, for some $a \in D_e$ such that John tickled $a(g)$. The second sentence will be true in this modified context just in case that same $a(g)$ giggled.

3 Inverse linking via Function composition

Here I will illustrate the complex quantifier approach to inverse linking, as described in section §1, above. There, we saw that the intuitively desired meaning of such a complex quantifier qua lambda term (repeated below) suffered from the problem that in order for $\mathcal{Q}$ to bind variables external to the complex quantifier variable capture was required, which means that the thing we want isn't actually the lambda term we wrote.

$$\lambda A_{et}.\mathcal{Q}(\lambda x_e.D(N(x))(A))$$

The problem is that we want to combine the two generalized quantifiers $\mathcal{Q}$ and $D(N(x))$ in such a way as to feed the next property argument A directly to $D(N(x))$, and then to give the property $\lambda x_e.D(N(x))(A)$ as argument to $\mathcal{Q}$. In other words, the only reason we abstract over the A argument to $D(N(x))$ in the above term is because we need to abstract over the x argument in the resulting sentence $D(N(x))(A)$ before this can be given as an argument to the generalized quantifier $\mathcal{Q}$. Because A occurs in the scope of the lambda binding over x, when an argument is substituted for A into the term $\lambda x_e.D(N(x))(A)$, if that argument contains a free variable x, we α-convert our term so that x stays free in the result. What we want is a way to delay abstracting over x until *after* A has been given as an argument to $D(N(x))$, something along the lines below:

$$(\mathcal{Q} \circ \lambda_x) \circ (D(N(x)))$$

While this is not a well-formed lambda term, it *is* a well-formed model-theoretic object of the kind described in the previous section.

For the time being, I will adopt a Heim and Kratzer (1998)-style perspective on semantic interpretation, in the sense that the structures which serve as input to the denotation function $[\![\cdot]\!]_{\mathcal{M}}$ will be syntactic structures to which quantifier raising transformations have already applied. Aside from putting assignments into the model as in the previous section, a major difference in the system here is that indices will be represented on the moved expression (so NP_i is an NP which binds a trace t_i). Such an object will be interpreted as usual, except that its denotation will be composed with a binder for x_i, λ_i:

$$[\![XP_i]\!]_{\mathcal{M}} := [\![XP]\!]_{\mathcal{M}} \circ \lambda_i$$

Aside from working, this makes the LF-interpretation system in this section the syntactic equivalent of the 'cooper-storage' system used in the fragment in the next section. We will see that the parts $\mathcal{Q} \circ \lambda_x$ and $D(N(x))$ of the above semantic object are the kind of things we would expect to see as stored quantifiers on the one hand and as denotations of expressions containing that same quantifier in the store on the other.

As an example, and continuing with the store perspective, consider the denotation of the expression senators on every committee, which has as its main meaning $\mathbf{senator} \wedge \mathbf{on}(x)$,[9] and in the store is contained the meaning $\mathbf{every}(\mathbf{committee}) \circ \lambda_x$. After merger with two, the stored meaning, $\mathbf{every}(\mathbf{committee}) \circ \lambda_x$, needs to be combined with the main meaning, $\mathbf{two}(\mathbf{senator} \wedge \mathbf{on}(x))$. However, the main meaning is of type $(et)t$, whereas the stored meaning is of type tt, making function application impossible. In this case, however, function composition provides us with exactly the meaning that we want:

$$(\mathbf{every}(\mathbf{committee}) \circ \lambda_x) \circ (\mathbf{two}(\mathbf{senator} \wedge \mathbf{on}(x)))$$

Applying the above function to an argument of type et, we have the following equalities (by the definition of composition):

$$\begin{aligned}
&((\mathbf{every}(\mathbf{committee}) \circ \lambda_x) \circ (\mathbf{two}(\mathbf{senator} \wedge \mathbf{on}(x))))(A)\\
&= (\mathbf{every}(\mathbf{committee}) \circ \lambda_x)(\mathbf{two}(\mathbf{senator} \wedge \mathbf{on}(x))(A))\\
&= \mathbf{every}(\mathbf{committee})(\lambda_x(\mathbf{two}(\mathbf{senator} \wedge \mathbf{on}(x))(A)))
\end{aligned}$$

Crucially, A might be of the form $\lambda_y(\mathbf{despise}(x)(y))$, where x occurs free in A,[10] allowing the previously retrieved quantifier to bind into it.

To make this work, I adopt a type-driven approach to semantic combination (Klein and Sag, 1985), with basic combinatory operators not only forward and backward function application, but also forward and backward function composition. For α a binary branching node with daughters β and γ, I write $[\![\alpha]\!]_{\mathcal{M}} = \textsc{combine}([\![\beta]\!]_{\mathcal{M}}, [\![\gamma]\!]_{\mathcal{M}})$, where $\textsc{combine}$ is a catch-all for whichever of the above named combinatory operators which fits the bill. As mentioned above, a movement subscript contributes a semantic binder; so if the mother is α_i, and the daughters are β and γ, $[\![\alpha_i]\!]_{\mathcal{M}} = \textsc{combine}(\beta, \gamma) \circ \lambda_i$.

3.1 Inverse Scope

We can calculate the meaning of the inversely linked reading in 4 below (with syntactic structure as in figure 4) in the following manner.

(4) Some relative of every lawyer despises him.

[9] The operator $\wedge$ is boolean ‘and’, extended pointwise over functions; so on predicate denotations $(A \wedge B)(a) = A(a) \cap B(a)$, where $\cap$ is normal set-theoretic intersection.

[10] This statement is misleading in the extreme. A is a function of type et; a function from functions from assignments to entities to sets of assignments. It doesn’t *have* syntactic structure, and thus nothing ‘occurs’ in it, let alone freely. The correct way to phrase this is as follows. A might be such that for some a, $A(a)$ is not indifferent to x, where a set of assignments H is indifferent to a variable denotation x iff for all $g \in H$, if $g \equiv h$ **mod** x then $h \in H$, where assignments g and h are equivalent modulo a variable denotation x iff for all variable denotations y, if $y(g) \neq y(h)$ then $y = x$.

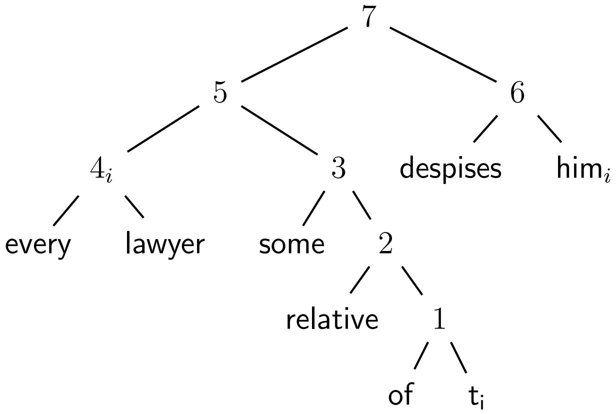


Figure 4: The LF-structure for the inverse scope reading of example 4

The lexical items (the leaves of figure 4) denote as described below.

$$
\begin{aligned}
&[\![\mathsf{some}]\!]_{\mathcal{M}} = \mathbf{some} \in D_{(et)(et)t}\\
&[\![\mathsf{every}]\!]_{\mathcal{M}} = \mathbf{every} \in D_{(et)(et)t}\\
&[\![\mathsf{of}]\!]_{\mathcal{M}} = \mathbf{of} \in D_{ee}\\
&[\![\mathsf{t_i}]\!]_{\mathcal{M}} = [\![\mathsf{him_i}]\!]_{\mathcal{M}} = \mathbf{x_i} \in D_e\\
&[\![\mathsf{lawyer}]\!]_{\mathcal{M}} = \mathbf{lawyer} \in D_{et}\\
&[\![\mathsf{relative}]\!]_{\mathcal{M}} = \mathbf{relative} \in D_{eet}\\
&[\![\mathsf{despises}]\!]_{\mathcal{M}} = \mathbf{despise} \in D_{eet}
\end{aligned}
$$

The denotations of the internal nodes (which are numbered in the figure), are given below.

1. $[\![1]\!]_{\mathcal{M}} = \textsc{combine}([\![\mathsf{of}]\!]_{\mathcal{M}}, [\![\mathsf{t_i}]\!]_{\mathcal{M}})$. As the type of $[\![\mathsf{of}]\!]_{\mathcal{M}}$ is ee, and the type of $[\![\mathsf{t_i}]\!]_{\mathcal{M}}$ is e, we apply the first argument to the second and obtain a value of type e:
$$[\![\mathsf{of}]\!]_{\mathcal{M}}([\![\mathsf{t_i}]\!]_{\mathcal{M}}) = \mathbf{of}(\mathbf{x_i})$$

2. $[\![2]\!]_{\mathcal{M}} = \textsc{combine}([\![\mathsf{relative}]\!]_{\mathcal{M}}, [\![1]\!]_{\mathcal{M}})$. As the type of $[\![\mathsf{relative}]\!]_{\mathcal{M}}$ is eet, and the type of $[\![1]\!]_{\mathcal{M}}$ is e, we again apply the first argument to the second and obtain a value of type et:
$$[\![\mathsf{relative}]\!]_{\mathcal{M}}([\![1]\!]_{\mathcal{M}}) = \mathbf{relative}(\mathbf{of}(\mathbf{x_i}))$$

3. $[\![3]\!]_{\mathcal{M}} = \textsc{combine}([\![\mathsf{some}]\!]_{\mathcal{M}}, [\![2]\!]_{\mathcal{M}})$. As $[\![\mathsf{some}]\!]_{\mathcal{M}}$ is of type $(et)(et)t$, and the type of $[\![2]\!]_{\mathcal{M}}$ is et, we apply once more the first argument to the second and obtain a value of type $(et)t$:
$$[\![\mathsf{some}]\!]_{\mathcal{M}}([\![2]\!]_{\mathcal{M}}) = \mathbf{some}(\mathbf{relative}(\mathbf{of}(\mathbf{x_i})))$$

4. $[\![4_i]\!]_{\mathcal{M}} = \textsc{combine}([\![\mathsf{every}]\!]_{\mathcal{M}}, [\![\mathsf{lawyer}]\!]_{\mathcal{M}}) \circ \lambda_i$. As the type of $[\![\mathsf{every}]\!]_{\mathcal{M}}$ is $(et)(et)t$, and the type of $[\![\mathsf{lawyer}]\!]_{\mathcal{M}}$ is et, we apply yet again the first argument to the second and obtain a value of type $(et)t$, which composes with λ_i of type tet to yield a value of type tt:

$$[\![\mathsf{every}]\!]_{\mathcal{M}}([\![\mathsf{lawyer}]\!]_{\mathcal{M}}) \circ \lambda_i = \mathbf{every}(\mathbf{lawyer}) \circ \lambda_i$$

5. $[\![5]\!]_{\mathcal{M}} = \textsc{combine}([\![4_i]\!]_{\mathcal{M}}, [\![3]\!]_{\mathcal{M}})$. As the type of $[\![4_i]\!]_{\mathcal{M}}$ is tt, and the type of $[\![3]\!]_{\mathcal{M}}$ is $(et)t$, we compose the first argument with the second and obtain a value of type $(et)t$:

$$(\mathbf{every}(\mathbf{lawyer}) \circ \lambda_i) \circ (\mathbf{some}(\mathbf{relative}(\mathbf{of}(\mathbf{x_i}))))$$

6. $[\![6]\!]_{\mathcal{M}} = \textsc{combine}([\![\mathsf{despises}]\!]_{\mathcal{M}}, [\![\mathsf{him}_i]\!]_{\mathcal{M}})$, as the type of $[\![\mathsf{despises}]\!]_{\mathcal{M}}$ is eet, and the type of $[\![\mathsf{him}_i]\!]_{\mathcal{M}}$ is e, we apply the first argument to the second and obtain a value of type et:

$$[\![\mathsf{despises}]\!]_{\mathcal{M}}([\![\mathsf{him}_i]\!]_{\mathcal{M}}) = \mathbf{despise}(\mathbf{x_i})$$

7. $[\![7]\!]_{\mathcal{M}} = \textsc{combine}([\![5]\!]_{\mathcal{M}}, [\![6]\!]_{\mathcal{M}})$. As the type of $[\![5]\!]_{\mathcal{M}}$ is $(et)t$, and the type of $[\![6]\!]_{\mathcal{M}}$ is et, we apply one last time the first argument to the second and obtain a value of type t:

$$((\mathbf{every}(\mathbf{lawyer}) \circ \lambda_i) \circ (\mathbf{some}(\mathbf{relative}(\mathbf{of}(\mathbf{x_i})))))(\mathbf{despise}(\mathbf{x_i}))$$

By the definition of function composition, this set of assignments is identical to the below:

$$\mathbf{every}(\mathbf{lawyer})(\lambda_i(\mathbf{some}(\mathbf{relative}(\mathbf{of}(\mathbf{x_i})))(\mathbf{despise}(\mathbf{x_i}))))$$

Although not much more can be said about the denotations of common nouns and verbs, the quantifiers and prepositions are intended to be logical constants in our models, and we can calculate on that basis a more refined description of the denotation of this reading.

The static denotations of our constants are given in figure 5. By the definition of **of**, this set of assignments is identical to the below.

$$\mathbf{every}(\mathbf{lawyer})(\lambda_i(\mathbf{some}(\mathbf{relative}(\mathbf{x_i}))(\mathbf{despise}(\mathbf{x_i}))))$$

Applying the definition of the function **some** in the above, we obtain the following.

$$\{g : \forall a.\ g \in \mathbf{lawyer}(a) \to g \in \lambda_i(\mathbf{some}(\mathbf{relative}(\mathbf{x_i}))(\mathbf{despise}(\mathbf{x_i})))(a)\}$$

$$\mathbf{some}(A)(B) := \{g : \text{for some } a \in D_e\ g \in A(a) \text{ and } g \in B(a)\}$$

$$\mathbf{of}(a) = a \qquad\qquad \mathbf{x_i}(g) = g_i$$

$$\mathbf{lawyer}(a) = \{g : a(g) \in \texttt{lawyer}\}$$

$$\mathbf{relative}(a)(b) = \{g : \langle a(g), b(g)\rangle \in \texttt{relative}\}$$

$$\mathbf{despise}(a)(b) = \{g : \langle a(g), b(g)\rangle \in \texttt{despise}\}$$

$$\mathbf{every}(A)(B) := \{g : \text{for every } a \in D_e \text{ if } g \in A(a) \text{ then } g \in B(a)\}$$

Figure 5: The static denotations of the constants

Unpacking the definition of λ_i from section §2.3, we arrive at the below.

$$\begin{aligned}\{g : \forall a.\ g \in \mathbf{lawyer}(a) \to \\ g \in \{h :\ h^{[i:=a(g)]} \in \mathbf{some}(\mathbf{relative}(\mathbf{x_i}))(\mathbf{despise}(\mathbf{x_i}))\}\}\end{aligned}$$

The equivalence $g \in \{h : \Phi(h)\} \leftrightarrow \Phi(g)$ nets us the following.

$$\begin{aligned}\{g : \forall a.\ g \in \mathbf{lawyer}(a) \to \\ g^{[i:=a(g)]} \in \mathbf{some}(\mathbf{relative}(\mathbf{x_i}))(\mathbf{despise}(\mathbf{x_i}))\}\end{aligned}$$

By the definition of **some**, this set is identical to the one below.

$$\begin{aligned}\{g : \forall a.\ g \in \mathbf{lawyer}(a) \to \\ g^{[i:=a(g)]} \in \{h : \exists b.\ h \in \mathbf{relative}(\mathbf{x_i})(b) \wedge \\ h \in \mathbf{despise}(\mathbf{x_i})(b)\}\}\end{aligned}$$

Again, from $g \in \{h : \Phi(h)\}$ we conclude that Φ holds of g.

$$\begin{aligned}\{g : \forall a.\ g \in \mathbf{lawyer}(a) \to \\ \exists b.\ g^{[i:=a(g)]} \in \mathbf{relative}(\mathbf{x_i})(b) \wedge g^{[i:=a(g)]} \in \mathbf{despise}(\mathbf{x_i})(b)\}\end{aligned}$$

Finally, unpacking the definitions of the non-logical constants, we arrive at the description of this set below.

$$\begin{aligned}\forall a.\ a(g) \in \texttt{lawyer} \to \\ \exists b.\ \langle b(g^{[i:=a(g)]}), a(g)\rangle \in \texttt{relative} \\ \wedge\ \langle b(g^{[i:=a(g)]}), a(g)\rangle \in \texttt{despise}\end{aligned}$$

Although there is no dynamicity involved in the ability of the inversely scoping quantifier to bind out of its syntactically containing DP, this fact is perhaps obscured by the assignment functionfullness of these denotations. I repeat the calculation done above, this time using the dynamic interpretations as given in figure 6.[11] As an abbreviatory convenience, I will write g^2 for $\langle g, g \rangle$. We begin with the description of the set of assignment functions as given below.

$$\begin{aligned} \mathbf{some}(A)(B) := \{ \langle g, h \rangle : & \text{ for some } a \in D_e \text{ and } k \in G \\ & \langle g, k \rangle \in A(a) \text{ and } \langle k, h \rangle \in B(a) \} \end{aligned}$$

$$\mathbf{of}(a) := a \qquad\qquad \mathbf{x_i}(\langle g, h \rangle) := g_i$$

$$\mathbf{lawyer}(a) := \{ \langle g, g \rangle : a(\langle g, g \rangle) \in \texttt{lawyer} \}$$

$$\mathbf{relative}(a)(b) := \{ \langle g, g \rangle : \langle b(\langle g, g \rangle), a(\langle g, g \rangle) \rangle \in \texttt{relative} \}$$

$$\mathbf{despise}(a)(b) := \{ \langle g, g \rangle : \langle b(\langle g, g \rangle), a(\langle g, g \rangle) \rangle \in \texttt{despise} \}$$

$$\begin{aligned} \mathbf{every}(A)(B) := \{ \langle g, g \rangle : & \text{ for every } a \in D_e \text{ and } k \in G \\ & \text{if } \langle g, k \rangle \in A(a) \text{ then for some } h \in G \\ & \langle k, h \rangle \in B(a) \} \end{aligned}$$

Figure 6: The dynamic denotations of the constants

$$\mathbf{every}(\mathbf{lawyer})(\lambda_i(\mathbf{some}(\mathbf{relative}(\mathbf{x_i}))(\mathbf{despise}(\mathbf{x_i}))))$$

Note that the pronoun in the VP is actually in the scope of the determiner **every**. Thus, it is in a canonical position to be bound in the normal way.

[11] These denotations seem perhaps unmotivated. That they are correct is best evidenced by the fact that they work to, for instance, derive the appropriate meaning for the sentence "every farmer who owns a donkey beats it", as modeled by the 'formula' below. (**and** is interpreted as dynamic predicate conjunction, where $\langle g, h \rangle \in \mathbf{and}(A)(B)(a)$ iff for some k, $\langle g, k \rangle \in A(a)$ and $\langle k, h \rangle \in B(a)$.)

$$\mathbf{every}(\mathbf{and}(\mathbf{farmer})(\lambda_1(\mathbf{some}(\mathbf{donkey})(\lambda_2(\mathbf{own}(\mathbf{x_2})(\mathbf{x_1}))))))(\mathbf{beat}(\mathbf{x_2})))$$

According to the denotations of the logical constants here assigned, the above can be shown identical to the set below.

$$\begin{gathered} \{ g^2 : \forall a, k.\ (g^2 \in \mathbf{farmer}(a)\ \&\ \exists b, h.\ h = g^{[1:=a(g^2)]}\ \&\ h^2 \in \mathbf{donkey}(b) \\ \&\ k = h^{[2:=b(h^2)]}\ \&\ k^2 \in \mathbf{own}(\mathbf{x_2})(\mathbf{x_1}) \to k^2 \in \mathbf{beat}(\mathbf{x_2})(a)) \} \end{gathered}$$

Note that the embedded existential quantifier ($\exists b$) is able to escape the scope of the antecedent of the conditional by the identification of k with $g^{[1:=a(g^2)][2:=b(g^{[1:=a(g^2)]2})]}$, and thus the pronoun $\mathbf{x_2}$ in the consequent is able to be assigned the value $b(g^{[1:=a(g^2)]2})$.

(In fact, it is semantically indistinguishable from the trace of the moved DP *every lawyer*.) If it were to be nonetheless bound dynamically, it would have to result from a context change originating in the restrictor argument. We will soon see that this is not the case. We next unpack the dynamic definition of **every** (writing g^2 for $\langle g, g\rangle$).

$$\begin{aligned}\{g^2 : \forall a, k.\ (\langle g, k\rangle \in \mathbf{lawyer}(a) \to \\ \exists h.\ \langle k, h\rangle \in \lambda_i(\mathbf{some}(\mathbf{relative}(\mathbf{x_i}))(\mathbf{despise}(\mathbf{x_i})))(a)\}\end{aligned}$$

By inspecting the definition of **lawyer**, we see that k must be identical to g, and thus that the pronoun (and the trace of *every lawyer*) must be being bound in the 'standard' way.

$$\begin{aligned}\{g^2 : \forall a.\ a(g^2) \in \texttt{lawyer} \to \\ \exists h.\ \langle g, h\rangle \in \lambda_i(\mathbf{some}(\mathbf{relative}(\mathbf{x_i}))(\mathbf{despise}(\mathbf{x_i})))(a)\}\end{aligned}$$

Using the dynamic version of λ_i from section §2.4, we obtain the description below.

$$\begin{aligned}\{g^2 : \forall a.\ a(g^2) \in \texttt{lawyer} \to \\ \exists h.\ \langle g, h\rangle \in \{\langle k, k'\rangle : \\ \langle k^{[i:=a(g^2)]}, k'\rangle \in \mathbf{some}(\mathbf{relative}(\mathbf{x_i}))(\mathbf{despise}(\mathbf{x_i}))\}\}\end{aligned}$$

Again, from $\langle g, h\rangle \in \{\langle k, k'\rangle : \Phi(\langle k, k'\rangle)\}$ we conclude that Φ holds of $\langle g, h\rangle$.

$$\begin{aligned}\{g^2 : \forall a.\ a(g^2) \in \texttt{lawyer} \to \\ \exists h.\ \langle g^{[i:=a(g^2)]}, h\rangle \in \mathbf{some}(\mathbf{relative}(\mathbf{x_i}))(\mathbf{despise}(\mathbf{x_i}))\}\end{aligned}$$

By the dynamic definition of **some**, this set is identical to the below.

$$\begin{aligned}\{g^2 : \forall a.\ a(g^2) \in \texttt{lawyer} \to \\ \exists h.\ \langle g^{[i:=a(g^2)]}, h\rangle \in \{\langle k', k''\rangle : \exists b, k.\ \langle k', k\rangle \in \mathbf{relative}(\mathbf{x_i})(b) \wedge \\ \langle k, k''\rangle \in \mathbf{despise}(\mathbf{x_i})(b)\}\}\end{aligned}$$

Once more, we apply basic notational conventions of set theory.

$$\begin{aligned}\{g^2 : \forall a.\ a(g^2) \in \texttt{lawyer} \to \\ \exists b, h, k.\ \langle g^{[i:=a(g^2)]}, k\rangle \in \mathbf{relative}(\mathbf{x_i})(b) \wedge \\ \langle k, h\rangle \in \mathbf{despise}(\mathbf{x_i})(b)\}\}\end{aligned}$$

The definition of **relative** admits only identical pairs of assignment functions, allowing us to conclude that $g^{[i:=a(g^2)]}$ and k are identical.

$$\begin{aligned}\{g^2 : \forall a.\ a(g^2) \in \texttt{lawyer} \to \\ \exists b, h.\ \langle b(g^{[i:=a(g^2)]2}), \mathbf{x_i}(g^{[i:=a(g^2)]2})\rangle \in \texttt{relative} \wedge \\ \langle g^{[i:=a(g^2)]}, h\rangle \in \mathbf{despise}(\mathbf{x_i})(b)\}\end{aligned}$$

By the definition of $\mathbf{x_i}$, the above can be written as the below.

$$\begin{aligned}\{g^2 : \forall a.\ a(g^2) \in \texttt{lawyer} \rightarrow &\\ &\exists b, h.\ \langle b(g^{[i:=a(g^2)]2}), a(g^2)\rangle \in \texttt{relative} \land \\ &\qquad \langle g^{[i:=a(g^2)]}, h\rangle \in \mathbf{despise}(\mathbf{x_i})(b)\}\end{aligned}$$

As **despise** imposes non-dynamic behaviour, we conclude that h and $g^{[i:=a(g^2)]}$ are identical.

$$\begin{aligned}\{g^2 : \forall a.\ a(g^2) \in \texttt{lawyer} \rightarrow &\\ &\exists b.\ \langle b(g^{[i:=a(g^2)]2}), a(g^2)\rangle \in \texttt{relative} \land \\ &\qquad \langle b(g^{[i:=a(g^2)]2}), \mathbf{x_i}(g^{[i:=a(g^2)]2})\rangle \in \texttt{despise}\}\end{aligned}$$

Finally, we use the definition of $\mathbf{x_i}$ to simplify the above description.

$$\begin{aligned}\{g^2 : \forall a.\ a(g^2) \in \texttt{lawyer} \rightarrow &\\ &\exists b.\ \langle b(g^{[i:=a(g^2)]2}), a(g^2)\rangle \in \texttt{relative} \land \\ &\qquad \langle b(g^{[i:=a(g^2)]2}), a(g^2)\rangle \in \texttt{despise}\}\end{aligned}$$

As a further confirmation that there is no dynamicity involved in the relation between *every lawyer* and its bound pronoun, note that the above meaning representation is identical (modulo the superscripts) to the representation of the non-dynamic meaning derived previously.

3.2 Direct Scope

Using function composition, we are also able to render the direct scope reading of example 4 without modifying the denotations assigned to our lexical items, simply by locating the landing site of QR beneath the determiner some, as in figure 7.

1. $[\![1]\!]_{\mathcal{M}} = \textsc{combine}([\![\mathsf{of}]\!]_{\mathcal{M}}, [\![\mathsf{t_i}]\!]_{\mathcal{M}})$, as the type of $[\![\mathsf{of}]\!]_{\mathcal{M}}$ is ee, and the type of $[\![\mathsf{t_i}]\!]_{\mathcal{M}}$ is e, we apply the first argument to the second and obtain a value of type e:

$$[\![\mathsf{of}]\!]_{\mathcal{M}}([\![\mathsf{t_i}]\!]_{\mathcal{M}}) = [\![\mathsf{t_i}]\!]_{\mathcal{M}} = \mathbf{x_i}$$

2. $[\![2]\!]_{\mathcal{M}} = \textsc{combine}([\![\mathsf{relative}]\!]_{\mathcal{M}}, [\![1]\!]_{\mathcal{M}})$. As the type of $[\![\mathsf{relative}]\!]_{\mathcal{M}}$ is eet, and the type of $[\![1]\!]_{\mathcal{M}}$ is e, we again apply the first argument to the second and obtain a value of type et:

$$[\![\mathsf{relative}]\!]_{\mathcal{M}}([\![1]\!]_{\mathcal{M}}) = \mathbf{relative}(\mathbf{x_i})$$

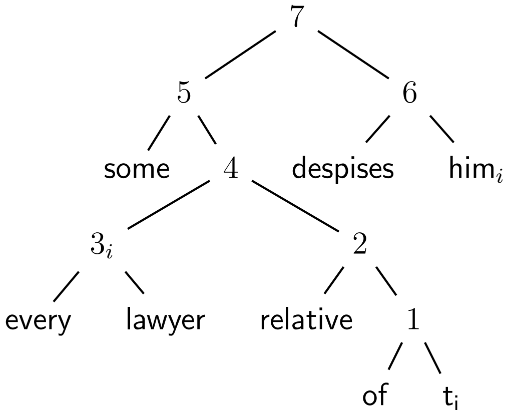


Figure 7: The LF-structure for the direct scope reading of example 4

3. $[\![3_i]\!]_{\mathcal{M}} = \textsc{combine}([\![\textsf{every}]\!]_{\mathcal{M}}, [\![\textsf{lawyer}]\!]_{\mathcal{M}}) \circ \lambda_i$. As the type of $[\![\textsf{every}]\!]_{\mathcal{M}}$ is $(et)(et)t$, and the type of $[\![\textsf{lawyer}]\!]_{\mathcal{M}}$ is et, we apply yet again the first argument to the second and obtain a value of type $(et)t$, which composes with λ_i of type tet to yield a value of type tt:

$$[\![\textsf{every}]\!]_{\mathcal{M}}([\![\textsf{lawyer}]\!]_{\mathcal{M}}) \circ \lambda_i = \mathbf{every}(\mathbf{lawyer}) \circ \lambda_i$$

4. $[\![4]\!]_{\mathcal{M}} = \textsc{combine}([\![3_i]\!]_{\mathcal{M}}, [\![2]\!]_{\mathcal{M}})$. As the type of $[\![3_i]\!]_{\mathcal{M}}$ is tt, and the type of $[\![2]\!]_{\mathcal{M}}$ is et, we compose the first argument with the second and obtain a value of type et:

$$(\mathbf{every}(\mathbf{lawyer}) \circ \lambda_i) \circ (\mathbf{relative}(\mathbf{x_i}))$$

5. $[\![5]\!]_{\mathcal{M}} = \textsc{combine}([\![\textsf{some}]\!]_{\mathcal{M}}, [\![4]\!]_{\mathcal{M}})$. As $[\![\textsf{some}]\!]_{\mathcal{M}}$ is of type $(et)(et)t$, and the type of $[\![4]\!]_{\mathcal{M}}$ is et, we apply once more the first argument to the second and obtain a value of type $(et)t$:

$$[\![\textsf{some}]\!]_{\mathcal{M}}([\![4]\!]_{\mathcal{M}}) = \mathbf{some}((\mathbf{every}(\mathbf{lawyer}) \circ \lambda_i) \circ (\mathbf{relative}(\mathbf{x_i})))$$

6. $[\![6]\!]_{\mathcal{M}} = \textsc{combine}([\![\textsf{despises}]\!]_{\mathcal{M}}, [\![\textsf{him}_i]\!]_{\mathcal{M}})$, as the type of $[\![\textsf{despises}]\!]_{\mathcal{M}}$ is eet, and the type of $[\![\textsf{him}_\mathsf{i}]\!]_{\mathcal{M}}$ is e, we apply the first argument to the second and obtain a value of type et:

$$[\![\textsf{despises}]\!]_{\mathcal{M}}([\![\textsf{him}_i]\!]_{\mathcal{M}}) = \mathbf{despise}(\mathbf{x_i})$$

7. $[\![7]\!]_{\mathcal{M}} = \textsc{combine}([\![5]\!]_{\mathcal{M}}, [\![6]\!]_{\mathcal{M}})$. As the type of $[\![5]\!]_{\mathcal{M}}$ is $(et)t$, and the type of $[\![6]\!]_{\mathcal{M}}$ is et, we apply one last time the first argument to the second and obtain a value of type t:

$$\mathbf{some}((\mathbf{every}(\mathbf{lawyer}) \circ \lambda_i) \circ (\mathbf{relative}(\mathbf{x_i}))))(\mathbf{despise}(\mathbf{x_i}))$$

Cashing out the function names in terms of their definitions, we arrive at a more transparent description of this set of assignments.

$$\begin{array}{l}
\mathbf{some}((\mathbf{every}(\mathbf{lawyer}) \circ \lambda_i) \circ (\mathbf{relative}(\mathbf{x_i})))(\mathbf{despise}(\mathbf{x_i})) \\
\hfill (\text{def } \mathbf{some}) \\
\{g : \exists b.\ g \in (\mathbf{every}(\mathbf{lawyer}) \circ \lambda_i) \circ (\mathbf{relative}(\mathbf{x_i}))(b) \\
\qquad \wedge\, g \in \mathbf{despise}(\mathbf{x_i})(b)\} \\
\hfill (\text{by composition}) \\
\{g : \exists b.\ g \in \mathbf{every}(\mathbf{lawyer})(\lambda_i(\mathbf{relative}(\mathbf{x_i})(b))) \\
\qquad \wedge\, g \in \mathbf{despise}(\mathbf{x_i})(b)\} \\
\hfill (\text{def } \mathbf{every}) \\
\{g : \exists b.\ g \in \{h : \forall a.\ h \in \mathbf{lawyer}(a) \rightarrow \\
\qquad\qquad\qquad h \in \lambda_i(\mathbf{relative}(\mathbf{x_i})(b))(a)\} \\
\qquad \wedge\, g \in \mathbf{despise}(\mathbf{x_i})(b)\} \\
\hfill (\text{set theory}) \\
\{g : \exists b.\ (\forall a.\ g \in \mathbf{lawyer}(a) \rightarrow g \in \lambda_i(\mathbf{relative}(\mathbf{x_i})(b))(a)) \\
\qquad \wedge\, g \in \mathbf{despise}(\mathbf{x_i})(b)\} \\
\hfill (\text{def } \lambda_i) \\
\{g : \exists b.\ (\forall a.\ g \in \mathbf{lawyer}(a) \rightarrow \\
\qquad\qquad g \in \{h : h^{[i:=a(g)]} \in (\mathbf{relative}(\mathbf{x_i})(b))\}) \\
\qquad \wedge\, g \in \mathbf{despise}(\mathbf{x_i})(b)\} \\
\hfill (\text{set theory}) \\
\{g : \exists b.\ (\forall a.\ g \in \mathbf{lawyer}(a) \rightarrow \\
\qquad\qquad g^{[i:=a(g)]} \in (\mathbf{relative}(\mathbf{x_i})(b))) \\
\qquad \wedge g \in \mathbf{despise}(\mathbf{x_i})(b)\}
\end{array}$$

As before, we can express this in more familiar terms: this reading is true with respect to an assignment g just in case there is an individual b of whom it is true that he despises g_i, and that for every lawyer a, b is a relative of $(g^{[i:=a]})_i = a$. Note that the QP every lawyer is *not* able to bind pronouns in the scope argument of the QP some relative of every lawyer in this reading.

4 The Syntax-Semantics Interface

The ideas outlined above in section §3 show how complex quantifier formation can be done using the functions λ_i and function composition. Of independent interest is that these very same functions and operations allow us to treat syntactic indices on moving expressions in the way syntacticians are used to, assigning a denotation directly to a DP with index i ($[\![DP]\!]_{\mathcal{M}} \circ \lambda_i$), instead of having to re-arrange structures so as to introduce the indicies in separate syntactic positions (as done by Heim and Kratzer (1998)). This is nice as it allows us to assign denotations directly to objects *as they are derived*, and not merely to the output of the derivational process. In the remainder of this paper I will show how to do this. I provide a grammar fragment written in the minimalist grammar formalism (Stabler, 1997), which is a formalization of some of the main ideas from Chomsky's (1995) Minimalist Program.

I begin with an introduction to the minimalist grammar formalism. For those readers already familiar with the syntactic literature in the minimalist rubric, this will provide a straightforward concretification of these ideas. Certain 'hot topics', such as the best treatment of morpho-syntactic feature bundles, will be glossed over or ignored, as they are irrelevant for the matter at hand, which is the integration of semantic interpretation with the syntactic derivation. After the formalism has been introduced, I present a rule-by-rule translation scheme, translating well-formed syntactic expressions into (sequences of) the model theoretic objects familiar from section §2. Finally, I give an interpreted (and infinite) fragment of English in terms of a minimalist grammar lexicon. The fragment prohibits QR out from within a DP, thus deriving Larson's Generalization, while at the same time permitting both direct and inverse scope readings, in the manner indicated in §3.

4.1 Minimalist Grammars

Minimalist grammars make use of two syntactic structure building operations; binary **merge** and unary **move**. **Merge** acts on its two arguments, viewed for the moment as having tree structure, by combining them together into a single tree (figure 8), Continuing for the moment to view expressions as tree-structured, the operation **move** rearranges the pieces of its single and syntactically complex argument in the manner shown in figure 9. The generating functions **merge** and **move** are not defined on all objects in their domain. Whether a generating function is defined on a particular object in

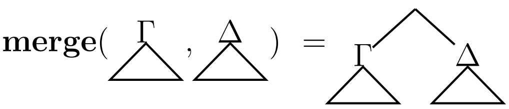


Figure 8: **merge**, schematically

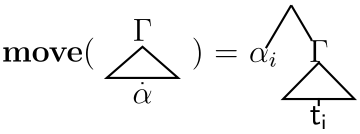


Figure 9: **move**, schematically

its domain (a pair of expressions in the case of **merge**, or a single expression in the case of **move**) is determined solely by the syntactic categories of these objects. In minimalist grammars, syntactic categories take the form of feature bundles. Consider the schematic instance of **merge** given in figure 8. In order for **merge** to apply to arguments Γ and Δ, the heads of both expressions must have matching features in their respective feature bundles. These features are eliminated ('checked', 'deleted') in the derived structure which results from their merger. In the case of **move**, as depicted in figure 9, the head of its argument Γ must have a feature matching a feature of the head of one of its subconstituents' α. In the result, both features are eliminated. Each feature type has an attractor and an attractee variant (i.e. each feature is either positive or negative), and for two features to match, one must be positive and the other negative. The kinds of features relevant for the **merge** and **move** operations are standardly taken for convenience to be different. For **merge**, the attractee feature is a simple categorial feature, written x. There are two kinds of attractor features, =x and x=, depending on whether the selected expression is to be merged on the right (=x) or on the left (x=). For the **move** operation, there is a single attractor feature, written +y, and two attractee features, -y and ⊖y, depending on whether the movement is overt (-y) or covert (⊖y). This is summarized in figure 10. Features are structured into feature bundles, which are ordered, so that some features can be available for checking only after others have been checked. I will represent feature bundles as lists, and the currently accessible feature is at the beginning (leftmost) position of the list.

	attractor	attractee
merge	=x, x=	x
move	+y	-y, ⊖y

Figure 10: Features

A lexical item is an atomic pairing of form and meaning, along with the syntactic information necessary to specify the distribution of these elements in more complex expressions. In the present context, I take lexical items to be pairings of abstract lexemes such as dog, cat, bank_1,. . . with feature bundles. I write lexical items using the notation $\langle\alpha, \delta\rangle$, where α is a lexeme (such as under), and δ is a feature bundle (such as '=d P'). An example lexical item is shown in figure 11. Its feature bundle '=d P' indicates that it first

$$\langle\mathsf{under}, \texttt{=d P}\rangle$$

Figure 11: A lexical entry for *under*

selects a DP argument, and then can be selected for as a PP.

Complex expressions will be written using Stabler's (1997) notation for the 'bare phrase structure' trees of Chomsky (1995). These trees are essentially X-bar trees without phrase and category information represented at internal nodes (see figure 12). Instead, internal nodes are labeled with 'arrows' > and <, which point to the head of their phrase. A tree of the form $[_< \alpha\ \beta]$ indicates that the head is to be found in the subtree α, and we say that α projects over β, while one of the form $[_> \alpha\ \beta]$ that its head is in β, and we say that β projects over α. Leaves are labeled with lexeme/feature pairs (and so a lexical item $\langle\alpha, \delta\rangle$ is a special case of a tree with only a single node). The head of a tree t is the leaf one arrives at from the root by following the arrows at the internal nodes. If t is a bare phrase structure tree with head H, then I will write $t[\text{H}]$ to indicate this. (This means we can write lexical items $\langle\alpha, \delta\rangle$ as $\langle\alpha, \delta\rangle[\langle\alpha, \delta\rangle]$.) The **merge** operation is defined on a pair of trees t_1, t_2 if and only if the head of t_1 has a feature bundle which begins with either =x or x=, and the head of t_2 has a feature bundle beginning with the matching x feature. The bare phrase structure tree which results from the merger of t_1 and t_2 has t_1 projecting over t_2, which is attached either to the right of t_1 (if the first feature of the head was =x) or to the left of t_1 (if the first feature of the head was x=). In either case, both selection features

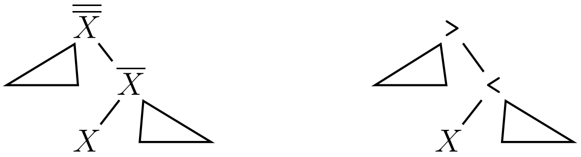
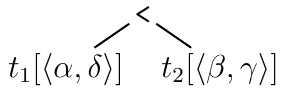


Figure 12: X-bar and Bare Phrase Structure Notation

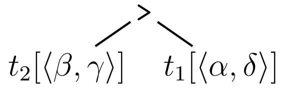
are checked in the result.

$\mathbf{merge}(t_1[\langle\alpha, \texttt{=x}\delta\rangle], t_2[\langle\beta, \texttt{x}\gamma\rangle]) =$



<
$t_1[\langle\alpha, \delta\rangle]$
$t_2[\langle\beta, \gamma\rangle]$

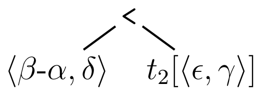
$\mathbf{merge}(t_1[\langle\alpha, \texttt{x=}\delta\rangle], t_2[\langle\beta, \texttt{x}\gamma\rangle]) =$



>
$t_2[\langle\beta, \gamma\rangle]$
$t_1[\langle\alpha, \delta\rangle]$

If the selecting tree is both a lexical item and an affix (which I notate by means of a hyphen preceding/following the lexeme in the case of a suffix/prefix), then phonological head movement is triggered from the head of the selected tree to the head of the selected tree to the head of the selecting tree.

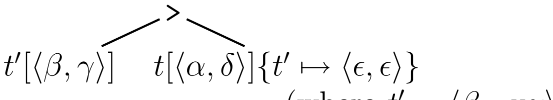
$\mathbf{merge}(\langle\text{-}\alpha, \texttt{=x}\delta\rangle, t_2[\langle\beta, \texttt{x}\gamma\rangle]) =$



<
$\langle\beta\text{-}\alpha, \delta\rangle$
$t_2[\langle\epsilon, \gamma\rangle]$

The operation **move** applies to a single tree $t[\langle\alpha, \texttt{+y}\delta]$ only if there is *exactly one* leaf ℓ in t with matching first feature $\texttt{-y}$ or $\ominus\text{y}$.[12] This is a radical version of the shortest move constraint (Chomsky, 1995), and will be called the SMC – it requires that an expression move to the first possible landing site. If there is competition for that landing site, the derivation crashes (because the losing expression will have to make a longer movement than absolutely necessary). If it applies, **move** moves the maximal projection of ℓ to a newly created specifier position in t (overtly, in the case of $\texttt{-y}$, and covertly, in the case of $\ominus\text{y}$), and deletes both licensing features. To make this precise, let $t\{t_1 \mapsto t_2\}$ denote the result of replacing all subtrees t_1 in t with t_2, for any tree t, and let ℓ_t^M denote the maximal projection of ℓ in t, for any leaf ℓ.

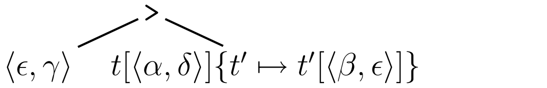
$\mathbf{move}(t[\langle\alpha, \texttt{+y}\delta]) =$



>
$t'[\langle\beta, \gamma\rangle]$
$t[\langle\alpha, \delta\rangle]\{t' \mapsto \langle\epsilon, \epsilon\rangle\}$

(where $t' = \langle\beta, \texttt{-y}\gamma\rangle_t^M$)

$\mathbf{move}(t[\langle\alpha, \texttt{+y}\delta]) =$



>
$\langle\epsilon, \gamma\rangle$
$t[\langle\alpha, \delta\rangle]\{t' \mapsto t'[\langle\beta, \epsilon\rangle]\}$

(where $t' = \langle\beta, \ominus\text{y}\gamma\rangle_t^M$)

[12] Other constraints have been explored in Gärtner and Michaelis (2007).

4.2 A Compositional Semantics for Minimalism

Here I present a rule-by-rule semantic interpretation scheme for minimalist grammars. Instead of an interpretation function, we have an interpretation relation; multiple meanings can be associated with a single syntactic object. The denotation of a syntactic object t is a pair of a model-theoretic object and a quantifier store. A quantifier store is a partial function from feature types to model-theoretic objects. The idea is that a syntactic object t with a moving subexpression $t' = \langle \beta, \texttt{-y}\gamma \rangle_t^M$ has a quantifier store $\mathcal{Q}$ such that $\mathcal{Q}(\textsc{y})$ is the stored meaning of t'. With $\emptyset$ I denote the empty quantifier store, such that for all feature types $\textsc{f}$, $\emptyset(\textsc{f})$ is undefined. Given two quantifier stores $\mathcal{Q}_1, \mathcal{Q}_2$, we form another using the $\vee$ operator, such that $\mathcal{Q}_1 \vee \mathcal{Q}_2$ is defined on an argument $\textsc{f}$ just in case either $\mathcal{Q}_1$ or $\mathcal{Q}_2$ is. If both $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are defined on an argument $\textsc{f}$, then $\mathcal{Q}_1$ takes precedence:

$$\mathcal{Q}_1 \vee \mathcal{Q}_2(\textsc{f}) := \begin{cases} \mathcal{Q}_1(\textsc{f}) & \text{if defined} \\ \mathcal{Q}_2(\textsc{f}) & \text{otherwise} \end{cases}$$

Let $\mathcal{Q}$ be a quantifier store, and $\textsc{f}$ a feature type. Then $\mathcal{Q}/\textsc{f}$ is the quantifier store just like $\mathcal{Q}$ except that it is undefined on $\textsc{f}$.[13]

For a lexical item ℓ, the denotation of ℓ is defined such that $[\![\ell]\!]_{\mathcal{M}} = \{\langle I(\ell), \emptyset \rangle\}$, where $I(\ell)$ is the interpretation of ℓ as specified in the lexicon.

An expression $\mathbf{merge}(t_1, t_2)$ will have $\langle \sigma, \mathcal{Q} \rangle$ as one of its meanings just in case one of the following is true:

1. $\sigma = \textsc{combine}(\sigma_1, \sigma_2)$ and $\mathcal{Q} = \mathcal{Q}_1 \vee \mathcal{Q}_2$, where $\langle \sigma_i, \mathcal{Q}_i \rangle \in [\![t_i]\!]_{\mathcal{M}}$.

   In this case, the meanings of t_1 and t_2 can combine in one of the 'standard' ways, via function application or function composition.

2. $\sigma = \sigma_1(\mathbf{x_k})$ and $\mathcal{Q} = \mathcal{Q}_1 \vee \mathcal{Q}_2 \vee \{\langle \textsc{f}, \sigma_2 \circ \lambda_k \rangle\}$, where $\langle \sigma_i, \mathcal{Q}_i \rangle \in [\![t_i]\!]_{\mathcal{M}}$ such that

   (a) $\sigma_1 \in D_{e\phi}$ and $\sigma_2 \in D_{(et)\psi}$

   (b) $t_2 = \langle \beta, \texttt{x}f\gamma \rangle_{t_2}^M$ such that $f \in \{\texttt{-f}, \ominus\texttt{f}\}$

   In this case, t_2 denotes a quantifier σ_2 which will take scope in a later position. The denoted quantifier is put into the store under the identifier $\textsc{f}$, which is the type of the next feature in the feature bundle of

[13] A partial function $f : A \to B$ is viewed as a total function $f : A \to B \cup \{\bullet\}$, where $\bullet$ is an object not contained in B. $\emptyset$ is the function which maps all objects in its domain to $\bullet$. $\mathcal{Q}/\textsc{f}$ is the function that maps $\textsc{g}$ to whatever $\mathcal{Q}$ maps it to, unless $\textsc{g} = \textsc{f}$, in which case it is mapped to $\bullet$. Thus, $\emptyset/\textsc{f} = \emptyset$ for all feature types $\textsc{f}$, and $\emptyset \vee \mathcal{Q} = \mathcal{Q}$ for all quantifier stores $\mathcal{Q}$.

the head of t_2. Note that this case overlaps with the previous one if σ_1 is of type *et*. Note also that some mechanism needs to be invoked to ensure that the variable $\mathbf{x_k}$ introduced in this step cannot be bound by any other quantifier on the store. See Kobele (2006) for details.

An expression $\mathbf{move}(t_1[\langle\alpha, \texttt{+y}\delta\rangle])$ will have $\langle\sigma, \mathcal{Q}\rangle$ as one of its meanings just in case one of the following is true:

1. $\sigma = \textsc{combine}(\sigma_1, (\mathcal{Q}_1 \vee \{\langle \textsc{y}, \texttt{id}_t\rangle\})(\textsc{y}))$, and $\mathcal{Q} = \mathcal{Q}_1/\textsc{y}$, where $[\![t_1]\!]_{\mathcal{M}}$ contains $\langle\sigma_1, \mathcal{Q}_1\rangle$

   In this case, the movement of the subexpression t_2 corresponds to retrieval of its meaning $\mathcal{Q}_1(\textsc{y})$ from the quantifier store. If $\mathcal{Q}_1$ is undefined on $\textsc{y}$, σ is either $\texttt{id}_t \circ \sigma_1 = \sigma_1$ or $\texttt{id}_t(\sigma_1) = \sigma_1$ (so long as σ_1 has a type ending in t).

2. $\mathcal{Q} = (\mathcal{Q}_1/\textsc{y}) \vee \{\langle \textsc{f}, \mathcal{Q}_1(\textsc{y})\rangle\}$, where $\langle\sigma, \mathcal{Q}_1\rangle \in [\![t_1]\!]_{\mathcal{M}}$, such that

   (a) $t_2 = \langle\beta, yf\gamma\rangle^M_{t_1}$ for $y \in \{\texttt{-y}, \ominus\texttt{y}\}$, such that $f \in \{\texttt{-f}, \ominus\texttt{f}\}$

   In this case, the meaning of t_2 is not retrieved from the store. However, it needs to be reassigned to the identifier $\textsc{f}$, so that it can be accessed again during the next movement of t_2.

Note that the semantic interpretation rules implement a reconstruction theory of quantifier scope, as proposed by Hornstein (1998), according to which the positions in which a quantified noun phrase can take scope are exactly those through which it has moved.

4.3 An Interpreted Fragment

In accord with the main thrust of this paper, I will take the meaning of prepositions to be invariant, and thus the inverse scope readings to be derived via (feature driven) movement of the prepositional object. This particular movement step will be assumed to be covert (another option would be to assume that the string identity of the direct/inverse scope readings is the result of a slew of conspiratorial movements), and, for purposes of uniformity, all movement driven by the need to check this feature ($\textsc{q}$) will be assumed covert. I will furthermore assume that all DPs must receive case ($\textsc{k}$), and that this comes about due to movement triggering feature checking, which, as it can sometimes have visible effects (passive, raising), is assumed always overt. Thus, DPs will uniformly have the features $\texttt{d}$ (a categorial feature), $\texttt{-k}$ (for overt case-driven movement), and $\ominus\texttt{q}$ (for covert 'QR'). These assumptions force us to a 'shell' structure (Larson, 1988) for PPs, as depicted in figure

13 - similar conclusions have been reached elsewhere (Jackendoff, 1983; van Riemsdijk and Huijbregts, 2007; Koopman, 2000). A P will then have the

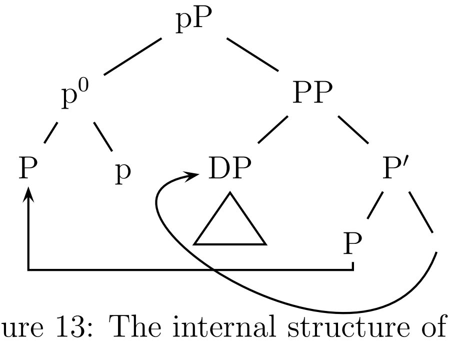


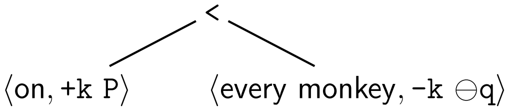
Figure 13: The internal structure of PPs

features `=d` (selecting a DP complement), `+k` (assigning case), `P` (a categorial feature), and will be taken to denote a relation between individuals (of type *eet*). The 'little-p' suffixal head will have the features `=P` (selecting a PP complement), `p` (a categorial feature). I will take the denotation of this higher, functional head to be conjunction (**and**) over PP denotations (one-place predicates). Note that under these assumptions, the PP is of the right type for a stored quantifier to compose with. I assume that QR may target the PP projection, and implement this idea by introducing a third layer to the P-system between the PP and the pP, which serves to introduce a `+q` feature. This suffixal head has the following features: `=P` (selecting a PP complement), `+q` (triggering QR), and `P` (a categorial feature), and will be taken to be semantically vacuous. Note that a PP of the form $[_{PP}$ ~~DP_i~~ $[_{P'}\ \mathrm{P}_j$ $[_{PP}\ \mathsf{DP}_i\ [_{P'}\ t_j\ t_i]]]]$ will have the meaning $(\mathbf{DP}\circ\lambda_i)\circ\mathbf{P}(\mathbf{x_i})$. These assumptions are implemented by means of the lexical items shown in figure 14. For a first

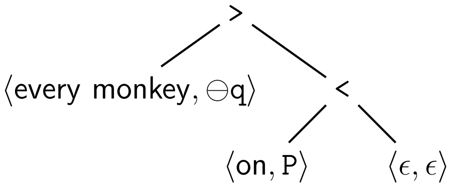
Phon	**Syn**	**Sem**
on	`=d +k P`	$\mathbf{on} \in D_{eet}$
-ϵ	`=P +q P`	$\mathtt{id}_{et}$
-ϵ	`=P p`	$\mathbf{and} \in D_{(et)(et)t}$

Figure 14: Lexical Entries for the P System

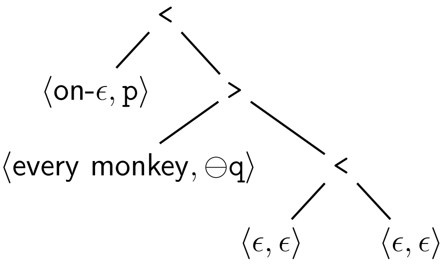
example, consider a derivation of the pP on every monkey. In addition to the lexical items in figure 14, I will make use of the expression *every monkey*, with features `d -k` $\ominus$`q`, which has as denotation the pair of the generalized quantifier $\mathbf{every}(\mathbf{monkey}) \in D_{(et)t}$ and the empty quantifier store $\emptyset$. For now I treat *every monkey* as a complex lexical item; we will see how to derive it later on. We derive the pP on every monkey as follows. First, we merge *on* with *every monkey*. This expression is interpreted as per the second case of the merge operation; $\langle \mathbf{on}(\mathbf{x_1}), \emptyset \vee \emptyset \vee \{\langle \textsc{k}, \mathbf{every}(\mathbf{monkey}) \circ \lambda_1\rangle\}\rangle = \langle \mathbf{on}(\mathbf{x_1}), \{\langle \textsc{k}, \mathbf{every}(\mathbf{monkey}) \circ \lambda_1\rangle\}\rangle$.



In the next step we apply the **move** operation to the expression above. Semantically, we apply the second case of the **move** interpretation rule: $\langle \mathbf{on}(\mathbf{x_1}), (\{\langle \textsc{k}, \mathbf{every}(\mathbf{monkey}) \circ \lambda_1\rangle\}/\textsc{k}) \vee \{\langle \textsc{q}, \{\langle \textsc{k}, \mathbf{every}(\mathbf{monkey}) \circ \lambda_1\rangle\}(\textsc{k})\rangle\}\rangle = \langle \mathbf{on}(\mathbf{x_1}), \emptyset \vee \{\langle \textsc{q}, \mathbf{every}(\mathbf{monkey}) \circ \lambda_1\rangle\}\rangle = \langle \mathbf{on}(\mathbf{x_1}), \{\langle \textsc{q}, \mathbf{every}(\mathbf{monkey}) \circ \lambda_1\rangle\}\rangle$.

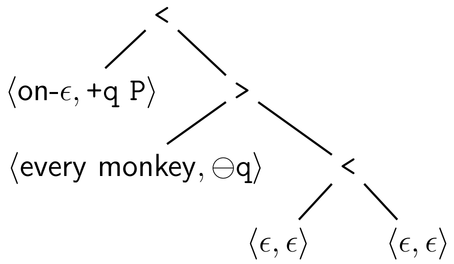


We now have a choice. We can either merge the second or the third expressions in figure 14, as both have as first feature =P. Both result in useful expressions; the former allows building a pP in which the prepositional object has checked its ⊖q feature (and thus cannot scope outside of the pP), the latter allows for QR of the prepositional object out from inside the pP. We begin with the latter lexical item (⟨-ϵ, =P p⟩), and then afterwards explore the other option. Because this expression is an affix, it triggers head movement. We use the first **merge** interpretation rule, which results in the following meaning: $\langle \textsc{combine}(\mathbf{and}, \mathbf{on}(\mathbf{x_1})), \emptyset \vee \{\langle \textsc{q}, \mathbf{every}(\mathbf{monkey}) \circ \lambda_1\rangle\}\rangle = \langle \mathbf{and}(\mathbf{on}(\mathbf{x_1})), \{\langle \textsc{q}, \mathbf{every}(\mathbf{monkey}) \circ \lambda_1\rangle\}\rangle$.

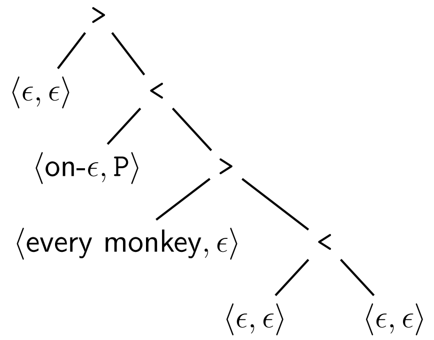


The syntactically relevant aspects of this expression are the following: its head has features p, and it contains one moving expression, with features ⊖q. Semantically speaking, it denotes a function of type $(et)et$, with a single expression (of type tt) in the quantifier store.

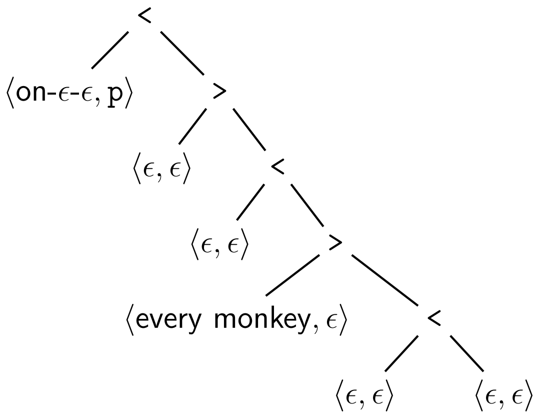
Revisiting our decision above, we this time choose to instead merge the second expression, $\langle$-ϵ, `=P +q P`$\rangle$. Because this expression is an affix, it triggers head movement from the head of its complement (on). We again use the first **merge** interpretation rule; resulting in $\langle \textsc{combine}(\texttt{id}_{et}, \mathbf{on}(\mathbf{x_1})), \emptyset \vee \{\langle \textsc{q}, \mathbf{every}(\mathbf{monkey}) \circ \lambda_1 \rangle\}\rangle = \langle \texttt{id}_{et}(\mathbf{on}(\mathbf{x_1})), \{\langle \textsc{q}, \mathbf{every}(\mathbf{monkey}) \circ \lambda_1 \rangle\}\rangle = \langle \mathbf{on}(\mathbf{x_1}), \{\langle \textsc{q}, \mathbf{every}(\mathbf{monkey}) \circ \lambda_1 \rangle\}\rangle$.



We apply again **move** to the above expression. As the movement is covert (*every monkey* has a $\ominus$`q` feature), the movement source position retains its phonological features. Semantically, we apply the first case of the **move** interpretation rule. Setting $\mathcal{Q} = \{\langle \textsc{q}, \mathbf{every}(\mathbf{monkey}) \circ \lambda_1 \rangle\}$, this yields $\langle \textsc{combine}(\mathbf{on}(\mathbf{x_1}), (\mathcal{Q} \vee \{\langle \textsc{q}, \texttt{id}_t \rangle\})(\textsc{q}), \mathcal{Q}/\textsc{q} \rangle = \langle \textsc{combine}(\mathbf{on}(\mathbf{x_1}), \mathbf{every}(\mathbf{monkey}) \circ$▮ $\lambda_1), \emptyset \rangle = \langle (\mathbf{every}(\mathbf{monkey}) \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}), \emptyset \rangle$.



The next step in the derivation is to merge the third expression in figure 14 with the expression above. Again, as this lexical item is marked as a suffix, it triggers head movement from its complement. Semantically, we apply the first case of the **merge** interpretation rule. The resulting meaning is computed to be the following: $\langle \textsc{combine}(\mathbf{and}, (\mathbf{every}(\mathbf{monkey}) \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1})), \emptyset \vee \emptyset \rangle = \langle \mathbf{and}((\mathbf{every}(\mathbf{monkey}) \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1})), \emptyset \rangle$.



The syntactically relevant aspects of this expression are the following: its head has features p, and it contains no moving expressions. Semantically speaking, it denotes a function of type $(et)et$, with an empty quantifier store.

I take PPs to be optional within the noun phrase. A functional projection of N (a non-pronounced head) selects first for an NP (=n), and then for a pP (=p) *also to the right*, to yield an N (n). This head is semantically vacuous.

A determiner selects an NP (=n) and returns a big-DP (D), of semantic type $(et)t$. Note that a quantified stored meaning ($\mathbf{DP} \circ \lambda_i$) can be semantically composed with something of this type. Accordingly, an opportunity for QR to apply at the DP level is created, and implemented by means of an suffixal head with the following features: =D (selecting a big-DP complement), +q (triggering QR), and D (a categorial feature), and will be taken to be semantically vacuous. To represent the lexical generalization that all DPs have the same feature bundle, I introduce a functional projection of D which selects for a big-DP (=D), and returns a DP (d) which must move for reasons of case (-k) and QR (-q). This functional projection is also taken to be semantically vacuous. These assumptions are implemented via the lexical items in figure 15. These structural assumptions are depicted in figure 16. The dotted movement arrows represent covert movements, of the PP-internal dP to its possible scope positions. As promised, I go through

Phon	**Syn**	**Sem**
monkey	n	$\mathbf{monkey} \in D_{et}$
ϵ	=n =p n	$\mathtt{id}_{et}$
every	=n D	$\mathbf{every} \in D_{(et)(et)t}$
-ϵ	=D +q D	$\mathtt{id}_{(et)t}$
ϵ	=D d -k ⊖q	$\mathtt{id}_{(et)t}$

Figure 15: Lexical Entries for the D System

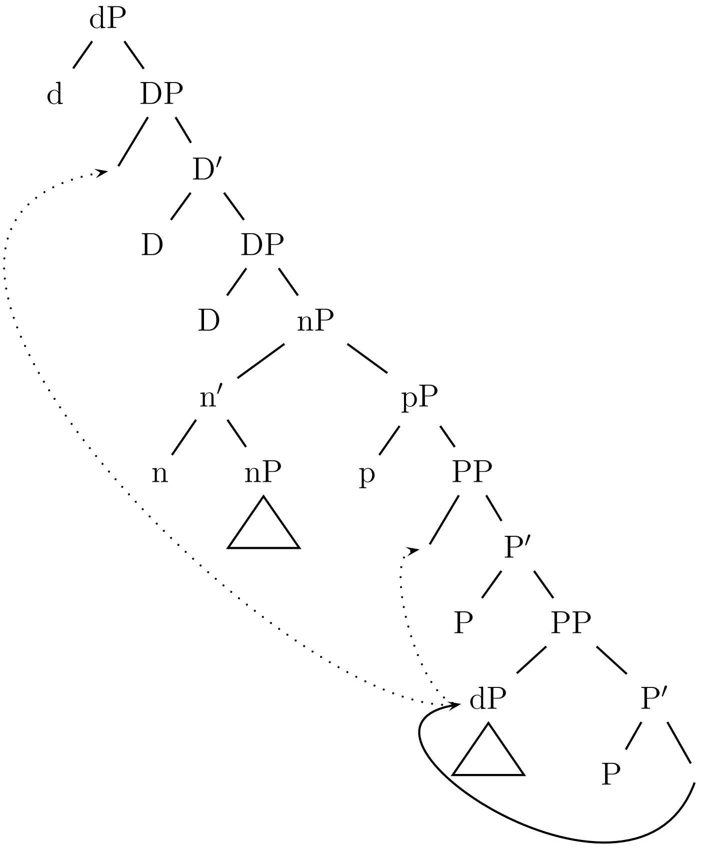
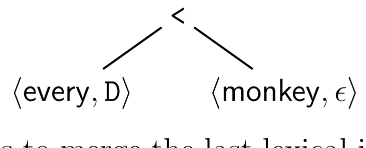
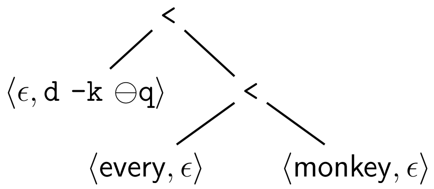


Figure 16: The structure of DP and PP

the derivation of the dP *every monkey* using these lexical items. We begin by merging the lexical items $\langle \mathsf{monkey}, \mathtt{n} \rangle$ and $\langle \mathsf{every}, \mathtt{=n\ D} \rangle$. Semantically, we employ the first case of the **merge** interpretation rule. This gives us $\langle \textsc{combine}(\mathbf{every}, \mathbf{monkey}), \emptyset \vee \emptyset \rangle = \langle \mathbf{every}(\mathbf{monkey}), \emptyset \rangle$.



The next and final step is to merge the last lexical item in figure 15 with the above expression. We use again the first case of the **merge** interpretation rule, resulting in $\langle \textsc{combine}(\mathtt{id}_{(et)t}, \mathbf{every}(\mathbf{monkey})), \emptyset \vee \emptyset \rangle$, which evaluates to $\langle \mathtt{id}_{(et)t}(\mathbf{every}(\mathbf{monkey})), \emptyset \rangle = \langle \mathbf{every}(\mathbf{monkey}), \emptyset \rangle$.

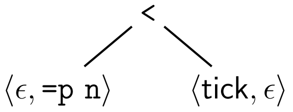


The syntactically relevant aspects of this expression are that its head has features `d -k` $\ominus$`q`, and that it has no moving sub-expressions. This allowed me to treat this expression as a lexical item with the same properties earlier on.

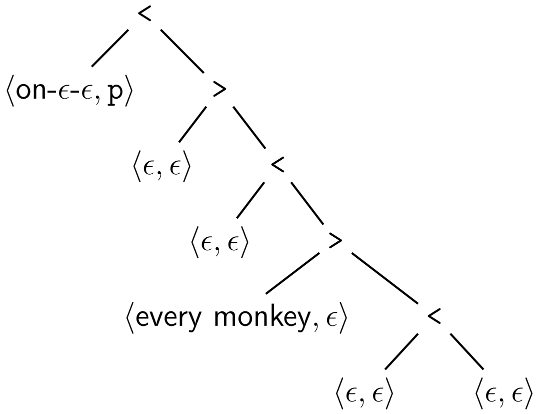
Now, I exhibit derivations of the dP *some tick on every monkey.* There are three such. The first two correspond to the direct and the indirect scope readings of this dP. The direct reading will make use of the second, and the indirect reading will make use of the first, of the pPs *on every monkey* derived previously. In addition to the lexical items in figures 14 and 15, I will need entries for *some* and *tick*, which have the same features as *every* and *monkey* respectively:

$$\langle \mathsf{some}, \texttt{=n D}\rangle \qquad \langle \mathsf{tick}, \texttt{n}\rangle$$

In this fragment, 'PP adjunction into the noun phrase' is dealt with by means of the lexical item $\langle \epsilon, \texttt{=n =p n}\rangle$, which selects to the right an NP, and then a pP, and then projects as an NP. The **merge** operation applies to this lexical item and the NP *tick*. Semantically, the first case of the **merge** interpretation rule applies, resulting in $\langle \textsc{combine}(\texttt{id}_{et}, \mathbf{tick}), \emptyset \vee \emptyset\rangle = \langle \texttt{id}_{et}(\mathbf{tick}), \emptyset\rangle = \langle \mathbf{tick}, \emptyset\rangle$.

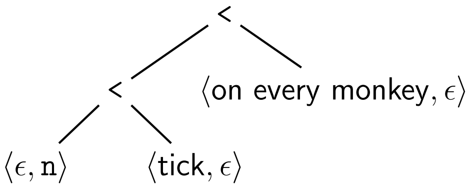


The next step in the derivation of the dP *some tick on every monkey* is to merge the above expression with a pP *on every monkey.* I begin with the direct scope reading, for which I recycle the second pP derived above, and repeated below, which denotes the singleton set whose only element is the pair $\langle \mathbf{and}((\mathbf{every}(\mathbf{monkey}) \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1})), \emptyset\rangle$.



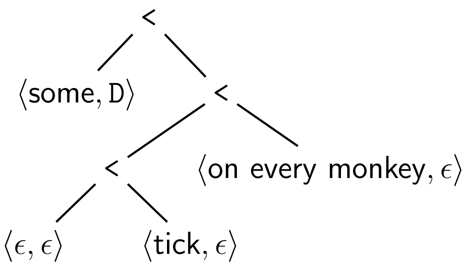
<
$\langle \textsf{on-}\epsilon\textsf{-}\epsilon, \texttt{p}\rangle$
>
$\langle \epsilon, \epsilon\rangle$
<
$\langle \epsilon, \epsilon\rangle$
>
$\langle \textsf{every monkey}, \epsilon\rangle$
<
$\langle \epsilon, \epsilon\rangle$
$\langle \epsilon, \epsilon\rangle$

To reduce clutter, I will write this complex expression as though it were a lexical item: $\langle \textsf{on every monkey}, \texttt{p}\rangle$. Syntactically speaking, this is inocuous; this lexical item has the same relevant syntactic properties as the more complex expression it is being substituted for: its head has features $\texttt{p}$, and it contains no moving expressions. The **merge** operation applies to the above two expressions. Semantically speaking, the first case of the **merge** interpretation rule applies, resulting in the pair (where I write **EM** as an abbreviation for $\mathbf{every}(\mathbf{monkey})$) $\langle \textsc{combine}(\mathbf{tick}, \mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))), \emptyset \vee \emptyset\rangle = \langle \mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick}), \emptyset\rangle$.



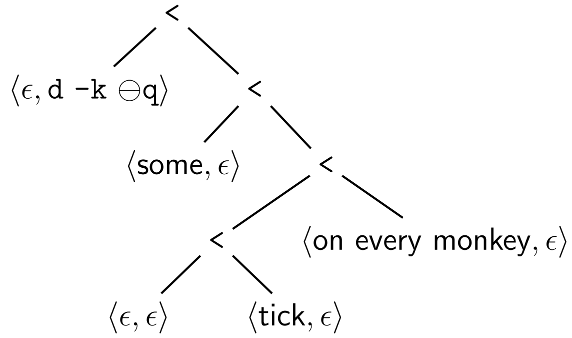
<
<
$\langle \textsf{on every monkey}, \epsilon\rangle$
$\langle \epsilon, \texttt{n}\rangle$
$\langle \textsf{tick}, \epsilon\rangle$

We next merge *some* with this expression. The first case of the **merge** interpretation rule applies again, yielding $\langle \textsc{combine}(\mathbf{some}, \mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick})), \emptyset \vee \emptyset\rangle = \langle \mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick})), \emptyset\rangle$.



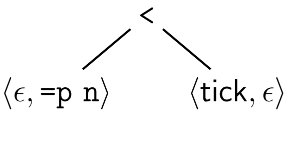
<
$\langle \textsf{some}, \texttt{D}\rangle$
<
<
$\langle \textsf{on every monkey}, \epsilon\rangle$
$\langle \epsilon, \epsilon\rangle$
$\langle \textsf{tick}, \epsilon\rangle$

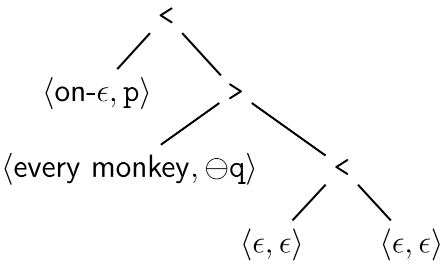
We conclude the derivation of the dP by merging the above expression with the lexical item $\langle \epsilon, \texttt{=D d -k} \ominus\texttt{q}\rangle$. The denotation of the resulting expression is again determined by the first case of the **merge** interpretation rule: $\langle \textsc{combine}(\mathtt{id}_{(et)t}, \mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick}))), \emptyset \vee \emptyset\rangle = \langle \mathtt{id}_{(et)t}(\mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick}))), \emptyset\rangle$, which is simply $\langle \mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick})), \emptyset\rangle$.



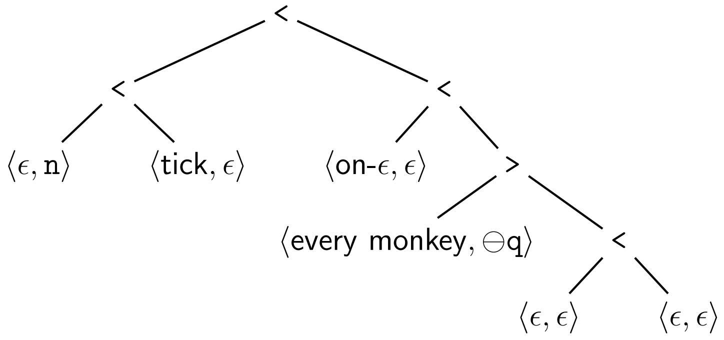
The relevant syntactic properties of the above expression are that its head has features `d -k` $\ominus$`q`, and that it has no moving subexpressions. Semantically, it is interpreted as the generalized quantifier $\mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick}))$ (which, intuitively, is true of a property just in case there is a tick which is on every monkey and which that property is true of), and has an empty quantifier store.

The inverse scope reading is obtained by merging the expressions below.



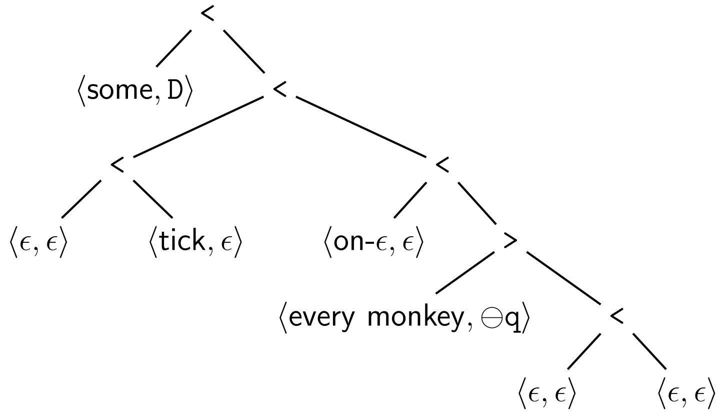


Doing this, we obtain the expression below, whose meaning is given by the first case of the **merge** interpretation rule, abbreviating the quantifier store $\{\langle \textsc{q}, \mathbf{EM} \circ \lambda_1\rangle\}$ as $\mathcal{Q}$: $\langle \textsc{combine}(\mathbf{tick}, \mathbf{and}(\mathbf{on}(\mathbf{x_1}))), \emptyset \vee \mathcal{Q}\rangle = \langle \mathbf{and}(\mathbf{on}(\mathbf{x_1}))(\mathbf{tick}), \mathcal{Q}\rangle$.



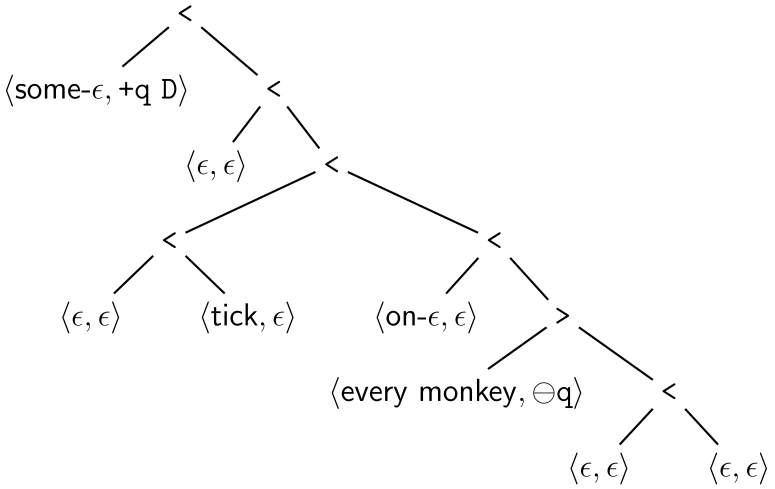
<
<
<
$\langle \epsilon, \mathtt{n} \rangle$
$\langle \mathsf{tick}, \epsilon \rangle$
$\langle \mathsf{on}\text{-}\epsilon, \epsilon \rangle$
>
$\langle \mathsf{every\ monkey}, \ominus\mathsf{q} \rangle$
<
$\langle \epsilon, \epsilon \rangle$
$\langle \epsilon, \epsilon \rangle$

We next merge *some* with the above expression. The meaning of the resulting expression we calculate by means of the first **merge** interpretation rule: $\langle \textsc{combine}(\mathbf{some}, \mathbf{and}(\mathbf{on}(\mathbf{x_1}))(\mathbf{tick})), \emptyset \vee \mathcal{Q} \rangle = \langle \mathbf{some}(\mathbf{and}(\mathbf{on}(\mathbf{x_1}))(\mathbf{tick})), \mathcal{Q} \rangle$.

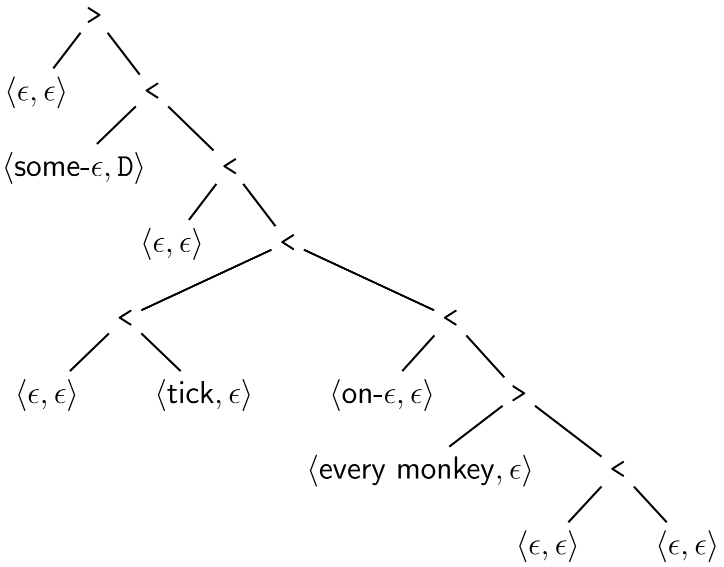


<
$\langle \mathsf{some}, \mathtt{D} \rangle$
<
<
<
$\langle \epsilon, \epsilon \rangle$
$\langle \mathsf{tick}, \epsilon \rangle$
$\langle \mathsf{on}\text{-}\epsilon, \epsilon \rangle$
>
$\langle \mathsf{every\ monkey}, \ominus\mathsf{q} \rangle$
<
$\langle \epsilon, \epsilon \rangle$
$\langle \epsilon, \epsilon \rangle$

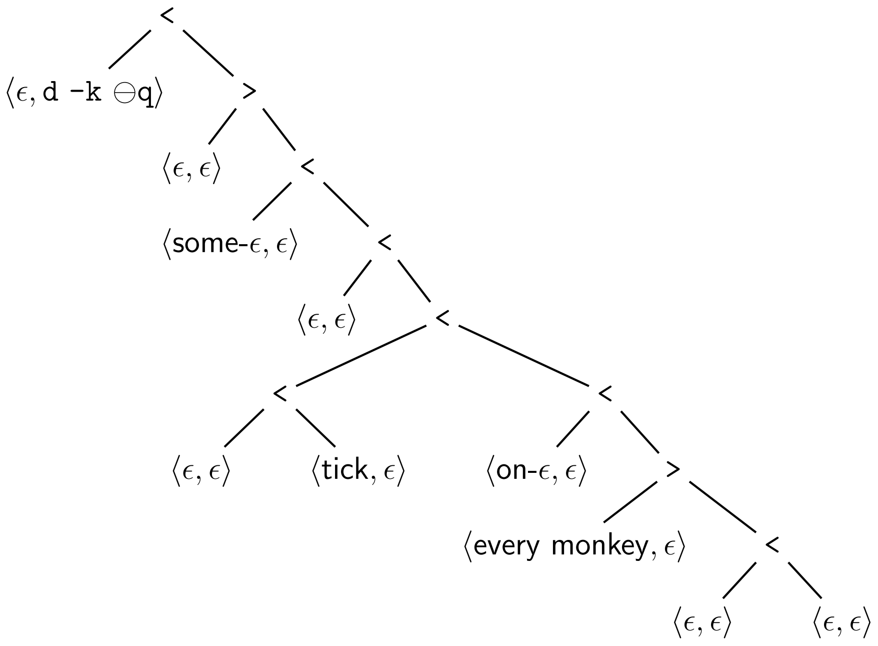
We have thus derived an expression (of type $(et)t$), with which the stored expression (of type tt) can be combined. We implement QR in the present system by means of the lexical item $\langle \text{-}\epsilon, \mathtt{=D\ +q\ D} \rangle$. The **merge** operation applies to this lexical item and the above expression. Semantically, the first case of the **merge** interpretation rule applies again, resulting in the following (where I write **ST** for $\mathbf{some}(\mathbf{and}(\mathbf{on}(\mathbf{x_1}))(\mathbf{tick}))$): $\langle \textsc{combine}(\mathtt{id}_{(et)t}, \mathbf{ST}), \emptyset \vee \mathcal{Q} \rangle = \langle \mathtt{id}_{(et)t}(\mathbf{ST}), \mathcal{Q} \rangle = \langle \mathbf{ST}, \mathcal{Q} \rangle$.



The **move** operation applies to the above expression. The movement is covert, as determined by the features of the moving expression *every monkey*. Semantically, the first case of the **move** interpretation rule applies, retrieving the stored quantifier and applying it to the meaning of the rest of the expression. We calculate $\langle \textsc{combine}(\mathbf{ST}, (\mathcal{Q} \vee \{\langle \textsc{q}, \mathtt{id}_t\rangle\})(\textsc{q})), \mathcal{Q}/\textsc{q}\rangle = \langle \textsc{combine}(\mathbf{ST}, \mathbf{EM} \circ \lambda_1), \emptyset\rangle = \langle (\mathbf{EM} \circ \lambda_1) \circ \mathbf{ST}, \emptyset\rangle$.

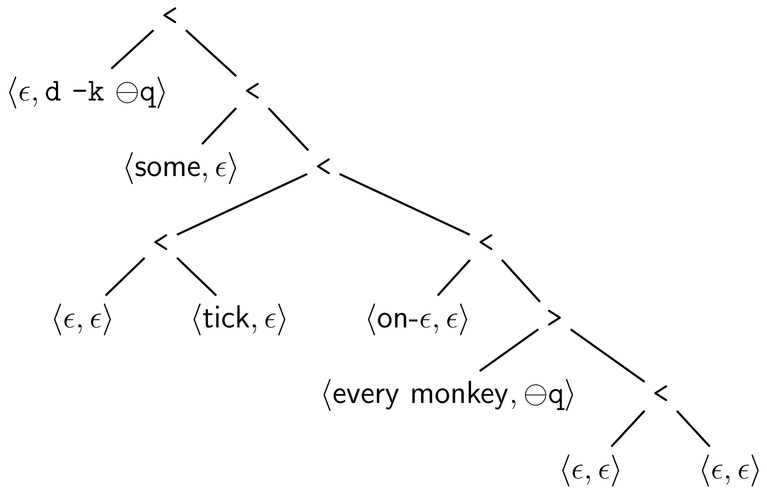


Finally, we merge the lexical item $\langle \epsilon, \texttt{=D d -k} \ominus\texttt{q}\rangle$ with the above. The first **merge** interpretation rule applies, yielding $\langle \textsc{combine}(\mathtt{id}_{(et)t}, (\mathbf{EM} \circ \lambda_1) \circ \mathbf{ST}), \emptyset \vee \emptyset\rangle = \langle \mathtt{id}_{(et)t}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{ST}), \emptyset\rangle = \langle (\mathbf{EM} \circ \lambda_1) \circ \mathbf{ST}, \emptyset\rangle$.



This dP is in all syntactically relevant respects indistinguishable from the previously derived one; its head has features `d -k` $\ominus$`q`, and it has no moving subexpressions. Semantically, it is type-theoretically indistinguishable as well, being of type $(et)t$, with an empty quantifier store.

Note however, that by implementing DP-internal QR in terms of the lexical item $\langle\text{-}\epsilon, \texttt{=D +q D}\rangle$, I have rendered it optional; no principles of the grammar formalism presented here make merger of one or another lexical item obligatory. If, instead of merging this lexical item during the previous derivation, we had skipped directly to the merger of the lexical item $\langle\epsilon, \texttt{=D d -k} \ominus\texttt{q}\rangle$, the following expression would have been obtained, with meaning $\langle\mathbf{ST}, \{\langle\text{Q}, \mathbf{EM}\rangle\}\rangle$.



Note that this dP is syntactically relevantly different from the other two: while its head has the same features, it contains a moving expression with features ⊖q. This expression is the kind of dP we would want if we wanted to allow for QR out from inside of a dP. As our grammar fragment derives it (we just saw explicitly how to), we need to ask whether this is desired, and, if not, how to block it from participating in derivations of complete sentences.

There are two reasons to want to anathematize this dP. The first is (the weak reason) that we already derive a dP which yields the inverse scope reading, rendering this one superfluous. The second reason is that, due to the SMC constraint on the **move** operation, the moving subpiece *every monkey* must move to its final landing site prior to the point at which this dP checks its -k feature.[14] This results in an unattested meaning whenever the dP *some tick on* t_1 is interpreted in any other than its base position. Assuming a predicate meaning of **die**, this meaning is roughly that there is a tick on some salient individual which has the property that if monkeys exist then the tick died, and is renderable as the following:

$$\mathbf{some}(\mathbf{and}(\mathbf{on}(\mathbf{x_1}))(\mathbf{tick})(\lambda_2(\mathbf{every}(\mathbf{monkey})(\lambda_1(\mathbf{die}(\mathbf{x_2}))))))$$

Furthermore, if one makes the common analytical assumption that the base position of subjects is above that of objects, then the only object-wide scope reading of the sentence *two fleas met some tick on every monkey* is the (unattested) one in which *some tick on* t_1 outscopes *two fleas* which outscopes *every monkey.*

[14] The SMC could of course be given up, but using the SMC as a constraint on movement has desirable computational effects (such as guaranteeing efficient recognizability – see (Harkema, 2001; Michaelis, 2001))

Assuming that this dP should indeed be blocked from participating in successful derivations of sentences, the question arises as to how this should be achieved. There are two basic strategies. One is to simply add a statement to the grammar that recognizes dP as a scope island. This could be done by adding a new variant of the attractee version of selection features, `*x`, on which **merge** is only defined if its second argument (the expression hosting this `*x` feature) contains no moving subexpressions. We then would replace the lexical item $\langle \epsilon, \texttt{=D d -k} \ominus\texttt{q}\rangle$ with the island-inducing $\langle \epsilon, \texttt{=D *d -k} \ominus\texttt{q}\rangle$ The other strategy is more conservative. Instead of modifying the underlying formal framework,[15] I will simply adopt an analysis of the VP in which there is no possibility for the inversely scoping DP to be moved before the case feature of the containing DP is checked.

I take, following a.o. Koopman and Sportiche (1991), subjects as well as objects to be base generated within the (extended projection of the) verb phrase. I take objects to be directly selected by their verbs, which then have features `=d V`, and subjects to be introduced in a higher verbal projection ('little-v'), which selects a verb phrase (`=V`), checks case of the object (`+k`), and selects for the subject (`=d`), thus implementing Burzio's generalization (Burzio, 1986): a verb which lacks an external argument fails to assign accusative case. I assume an empty lexical item which allows QR at the vP level for the object (`+q`). The subject's `-k` and $\ominus$`q` features are then checked in an IP, headed here by *will* (which I will assume to be semantically vacuous). This is shown in figure 17. I show now how to derive the sentence *some tick on every monkey will bite him* from the lexical items presented thus far. We lack only a lexical entry for the pronoun, *him*. I will simply postulate the existence of a family of pronouns, $\langle \textsf{him}_\textsf{i}, \texttt{d -k} \ominus\texttt{q}\rangle$, which are interpreted as variables with the same index: $[\![him_i]\!]_{\mathcal{M}} = \langle \mathtt{x_i}, \emptyset\rangle$, ignoring both the case restrictions on pronouns, as well as the principles of the binding theory, which constrain the distribution of indices on pronouns.[16] The

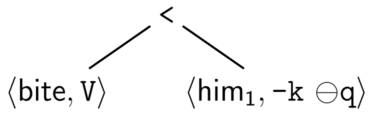
Phon	**Syn**	**Sem**
bite	`=d V`	**bite** $\in D_{eet}$
-ϵ	`=V +k d= v`	$\texttt{id}_{et}$
-ϵ	`=v +q v`	$\texttt{id}_{t}$
will	`=v +k +q i`	$\texttt{id}_{t}$

Figure 17: Lexical entries for the V and I Systems

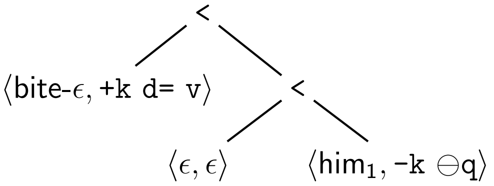
[15] In the context of the SMC, the introduction of the `*x` feature type turns out to be simulable by the introduction of (otherwise unmotivated) lexical items.

[16] While the distribution of indices on pronouns is a topic which seems to require more formal power than what I have here (Bonato, 2006), the distribution of the forms 'him'

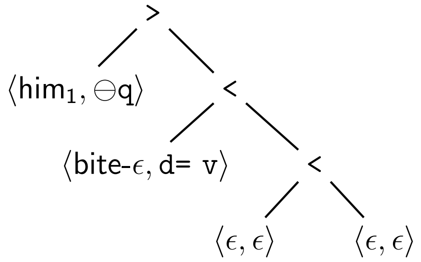
first three steps of the derivation are common to both inverse and direct scope readings. First *bite* and *him*$_i$ are merged. Semantically, we apply the first case of the **merge** interpretation rule. This results in the pair $\langle \textsc{combine}(\mathbf{bite}, \mathbf{x_1}), \emptyset \vee \emptyset \rangle = \langle \mathbf{bite}(\mathbf{x_1}), \emptyset \rangle$.



Next, we merge the active voice head $\langle\text{-}\epsilon, \texttt{=V +k d= v}\rangle$ with the expression above. The first case of the **merge** interpretation rule applies, yielding $\langle \textsc{combine}(\texttt{id}_{et}, \mathbf{bite}(\mathbf{x_1})), \emptyset \vee \emptyset \rangle = \langle \texttt{id}_{et}(\mathbf{bite}(\mathbf{x_1})), \emptyset \rangle = \langle \mathbf{bite}(\mathbf{x_1}), \emptyset \rangle$.



Now, we apply the **move** operation to the above tree. We apply the second case of the **move** interpretation rule, on which basis we obtain the pair $\langle \mathbf{bite}(\mathbf{x_1}), \emptyset/\textsc{k} \vee \{\langle \textsc{q}, \emptyset(\textsc{k})\rangle\}\rangle = \langle \mathbf{bite}(\mathbf{x_1}), \emptyset \rangle$.



There are now three possible next steps: we can merge either

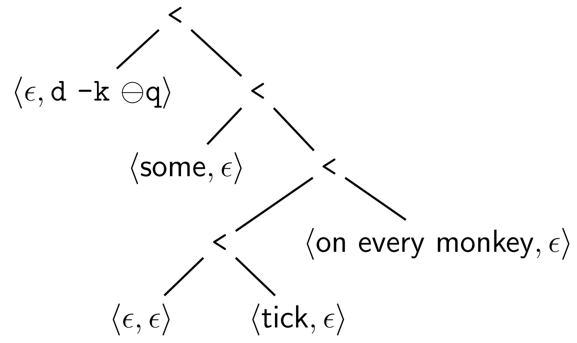
1. the directly linked version of *some tick on every monkey*
2. the inversely linked version of *some tick on every monkey*
3. the anathema version of *some tick on every monkey*

I go through each in turn. The latter ‘anathema’ version will cause the derivation to crash (as both *him*$_1$ and *every monkey* have as first feature in their feature bundles ⊖q).

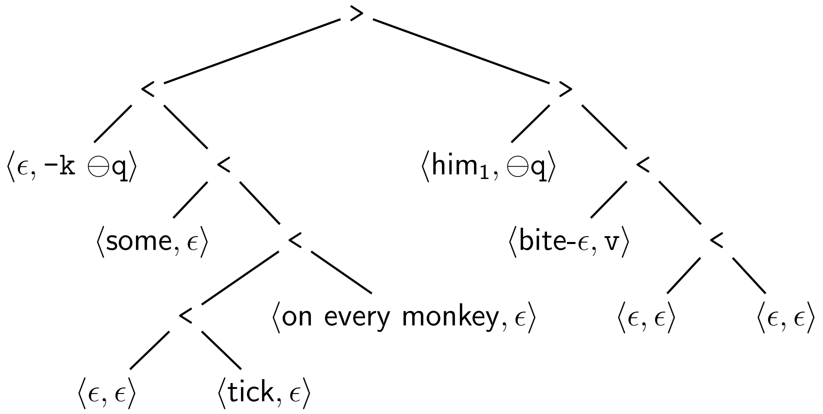
and ‘he’ seem straightforwardly modeled in terms of a more articulated feature structure, for example by means of partially ordered features (so `-k` can be checked by either `+acc` or `+nom`), which seems to be a formal variant of finite feature unification.

4.3.1 The Directly Linked Reading

We begin with the direct version of *some tick on every monkey*, repeated below, with meaning $\langle \mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick})), \emptyset\rangle$.



Merging the above dP with the derived vP *bite him* above yields the following semantically ambiguous structure.



As the second argument to **merge** (the predicate) is of semantic type *et*, and the first is of type $(et)t$, *both* cases (storage *and* function application) of the **merge** interpretation rule apply. These give rise to the following *two* meanings, both of which this derivation is associated with.

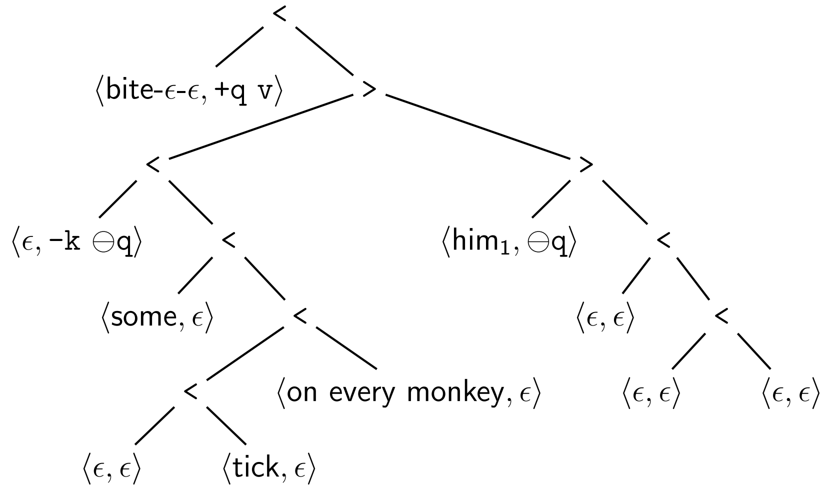
(case one)

$$\begin{aligned}&\langle \textsc{combine}(\mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick})), \mathbf{bite}(\mathbf{x_1})), \emptyset \vee \emptyset\rangle \\ &\quad = \langle \mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick}))(\mathbf{bite}(\mathbf{x_1})), \emptyset\rangle\end{aligned}$$

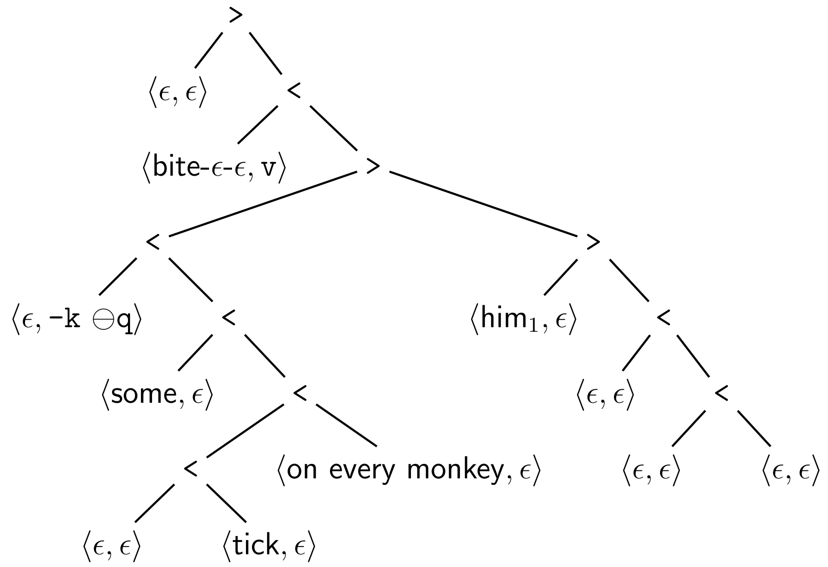
(case two)

$$\begin{aligned}&\langle \mathbf{bite}(\mathbf{x_1})(\mathbf{x_2}), \emptyset \vee \emptyset \vee \{\langle \textsc{k}, \mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick})) \circ \lambda_2\rangle\}\rangle \\ &\quad = \langle \mathbf{bite}(\mathbf{x_1})(\mathbf{x_2}), \{\langle \textsc{k}, \mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick})) \circ \lambda_2\rangle\}\rangle\end{aligned}$$

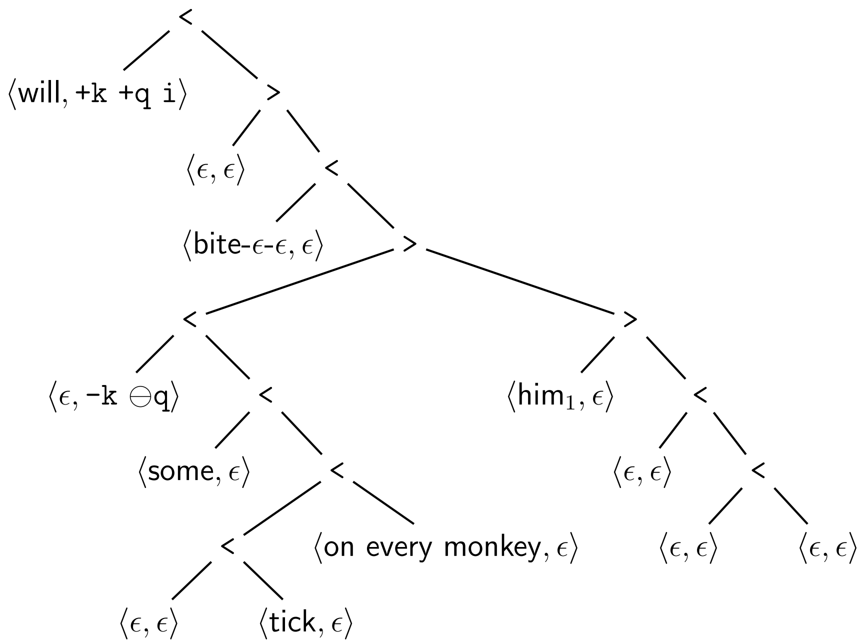
In the next derivational step, the optional 'QR at the vP level' lexical item $\langle$-ϵ, =v +q v$\rangle$ is merged with the above expression. As the meaning of this expression is simply the identity function over meanings of type t, the denotation of the resulting expression (via the first case of the **merge** interpretation rule) is unchanged from the above.



Now the **move** operation applies to the above expression, covertly moving *him* to adjoin to vP. The first case of the **move** interpretation rule applies, but, because there is no expression on the stack indexed by the feature type Q, this ends up simply applying the identity function to the current meanings, and thus the denotation of the result remains the same.



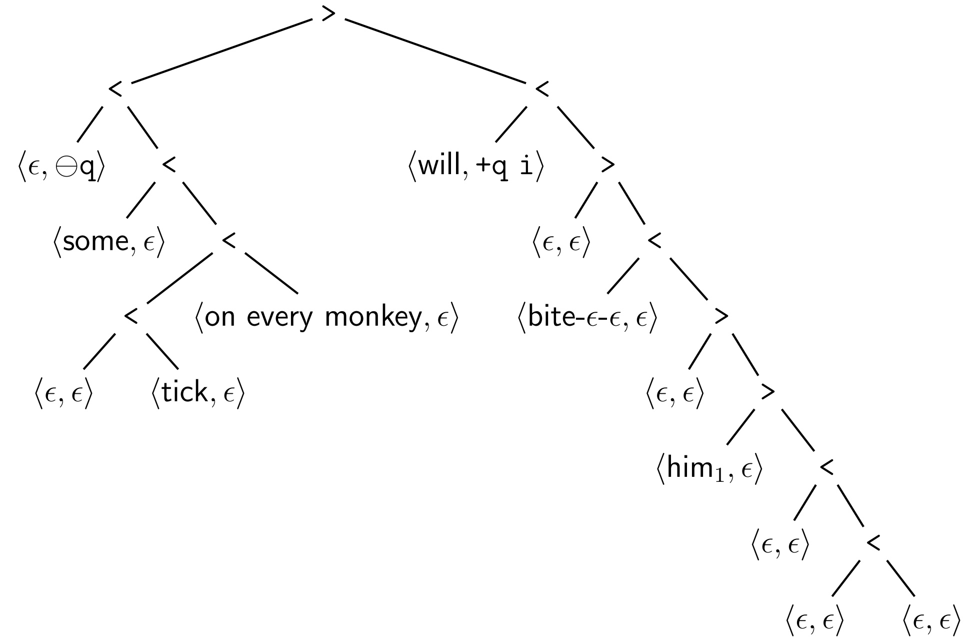
We next apply the **merge** operation to the pair consisting of the lexical item $\langle$will, =v +k +q i$\rangle$ and the expression pictured above. As I am ignoring all but the most basic of semantic properties, *will* is taken here to be semantically vacuous, and thus the denotation of the thus derived expression is identical to that of the expression depicted above.



The **move** operation applies to the above expression, checking both `+k` and `-k` features of the affected heads, and moving the subject dP to the left of *will*. The second case of the **move** interpretation rule applies, resulting (for one of the meanings of the expression thus derived) in a reindexing of the stored semantic object. Writing $\mathcal{Q}$ for $\{\langle \textsc{k}, \mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick})) \circ \lambda_2\rangle\}$, the two meanings of the expression below are as follows.

$$\begin{aligned}&\langle \mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick}))(\mathbf{bite}(\mathbf{x_1})), \emptyset/\textsc{k} \vee \{\langle \textsc{q}, \emptyset(\textsc{k})\rangle\}\rangle \\ &\qquad = \langle \mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick}))(\mathbf{bite}(\mathbf{x_1})), \emptyset\rangle\end{aligned}$$

$$\begin{aligned}&\langle \mathbf{bite}(\mathbf{x_1})(\mathbf{x_2}), \mathcal{Q}/\textsc{k} \vee \{\langle \textsc{q}, \mathcal{Q}(\textsc{k})\rangle\}\rangle \\ &\qquad = \langle \mathbf{bite}(\mathbf{x_1})(\mathbf{x_2}), \{\langle \textsc{q}, \mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick})) \circ \lambda_2\rangle\}\rangle\end{aligned}$$



The last step in the derivation of the direct scope reading of *some tick on every monkey will bite him* is the covert movement of the subject dP triggered by the $\textsc{q}$ feature type. We can only apply the first case of the **move** interpretation rule, resulting in the denotations shown below, writing again $\mathcal{Q}$ for $\{\langle\textsc{q}, \mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick})) \circ \lambda_2\rangle\}$:

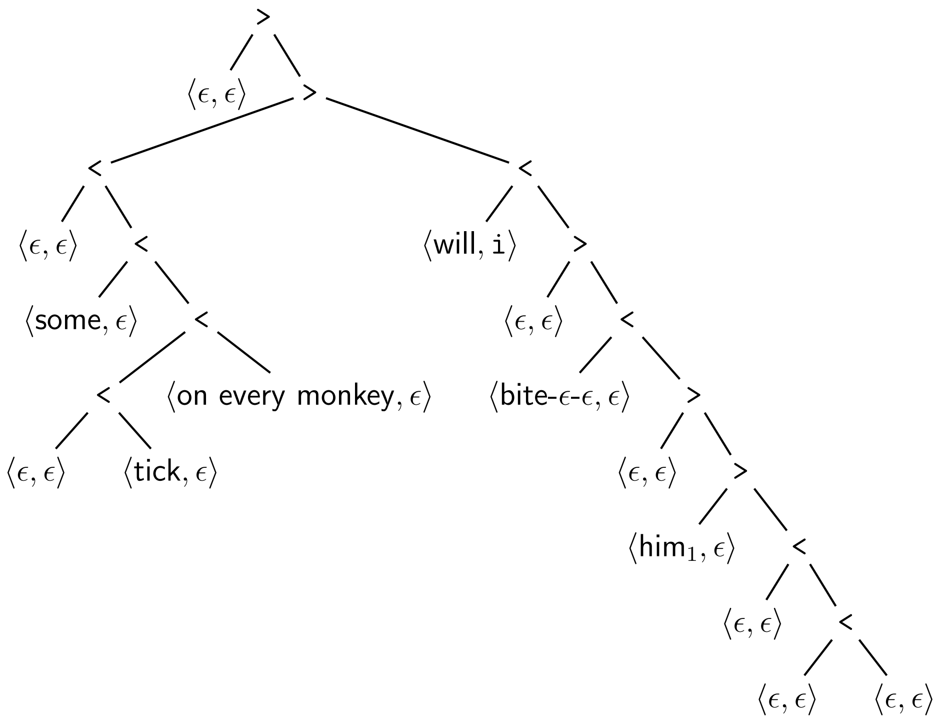
$$\begin{aligned}
&\langle\textsc{combine}(\mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick}))(\mathbf{bite}(\mathbf{x_1})),\\
&\qquad\qquad (\emptyset \vee \{\langle\textsc{q}, \mathtt{id}_t\rangle\})(\textsc{q})), \emptyset/\textsc{q}\rangle\\
&\qquad = \langle\mathtt{id}_t(\mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick}))(\mathbf{bite}(\mathbf{x_1}))), \emptyset\rangle\\
&\qquad = \langle\mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick}))(\mathbf{bite}(\mathbf{x_1})), \emptyset\rangle
\end{aligned}$$

$$\begin{aligned}
&\langle\textsc{combine}(\mathbf{bite}(\mathbf{x_1})(\mathbf{x_2}), (\mathcal{Q} \vee \{\langle\textsc{q}, \mathtt{id}_t\rangle\})(\textsc{q})), \mathcal{Q}/\textsc{q}\rangle\\
&\qquad = \langle\textsc{combine}(\mathbf{bite}(\mathbf{x_1})(\mathbf{x_2}),\\
&\qquad\qquad\qquad \{\langle\textsc{q}, \mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick})) \circ \lambda_2\rangle\}), \emptyset\rangle\\
&\qquad = \langle(\mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick})) \circ \lambda_2)(\mathbf{bite}(\mathbf{x_1})(\mathbf{x_2})), \emptyset\rangle\\
&\qquad = \langle\mathbf{some}(\mathbf{and}((\mathbf{EM} \circ \lambda_1) \circ \mathbf{on}(\mathbf{x_1}))(\mathbf{tick}))(\lambda_2(\mathbf{bite}(\mathbf{x_1})(\mathbf{x_2}))), \emptyset\rangle
\end{aligned}$$

As the reader can verify for himself, interpreted statically, both meanings above are identical.[17] The meanings differ when interpreted dynamically, as

[17] Although I am using turns of phrase such as 'interpreted statically', it is worth bearing in mind that the things written above denote model theoretic objects, and *not* expressions in a language which can be interpreted as model theoretic objects. In other words, the decision to 'interpret statically' is made before we derive a single expression, i.e. lexically.

in the second case, but not in the first, the scope argument of **some** is not static (in the depiction of this meaning above, it is prefixed by a λ_2 function, which introduces dynamicity).[18]



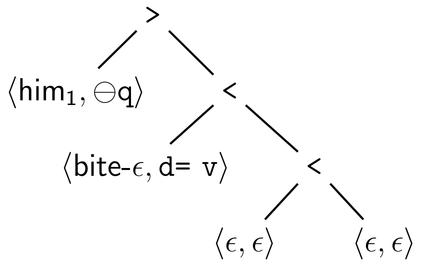
4.3.2 The Inversely Linked Reading

Modulo the identity of the dP, the inversely linked reading is derived in the same manner as the direct one. We begin with the vP *bite him* (repeated

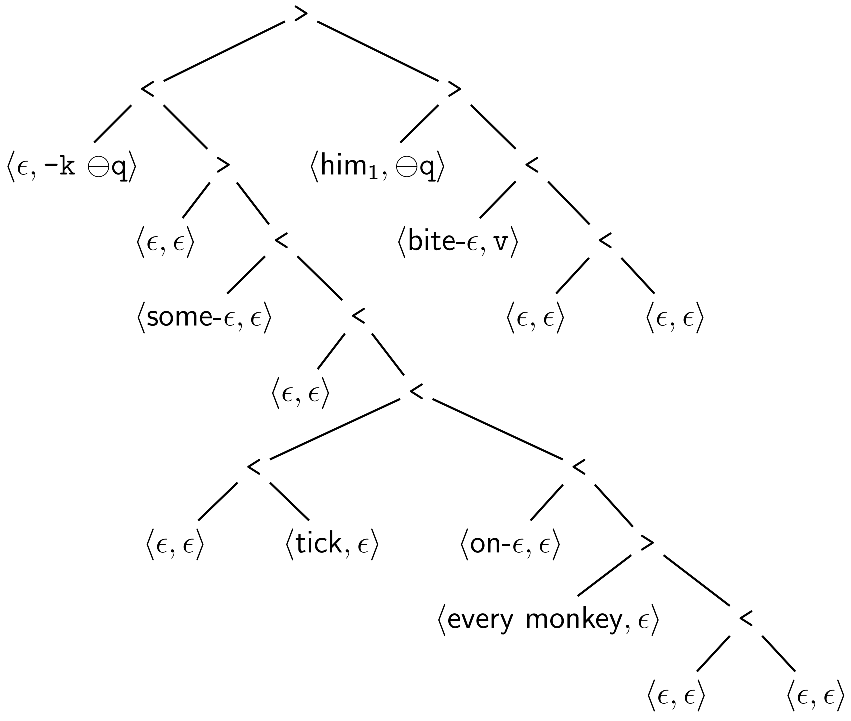
[18] What this means in empirical terms is that in the present system only quantifiers which are first stored and then retrieved are able to dynamically bind pronouns. As our current analysis of inverse scope (following Hornstein (1998)) is that it arises when the subject is interpreted in its base position (and the object in its highest (Q) position), we predict that in discourses a narrow (with respect to the object) scoping subject in a previous sentence cannot bind pronouns in other sentences. To evaluate this prediction, we need to find a pair of generalized quantifiers which are permeable to dynamic effects, but for which the inverse reading does not entail the direct reading. Thus, the first sentence in 1 gives us a pair of dynamic generalized quantifiers as desired, but the inverse scope reading entails the direct scope reading. And while the first sentence in 2 has a pair of generalized quantifiers for which the inverse scope reading does not entail the direct reading, the object dP is not permeable to dynamicity.

1. Some boy kissed some girl. He was on his bicycle.
2. Some boy kissed every girl. He was on his bicycle.

below), with meaning $\langle \mathbf{bite(x_1)}, \emptyset \rangle$.



Merging with the above vP the inversely linked dP *some tick on every monkey* with meaning $\langle (\mathbf{EM} \circ \lambda_1) \circ \mathbf{ST}, \emptyset \rangle$ nets the expression below.



In this **merge** step, as in the corresponding one for the direct sentence, both case of the **merge** interpretation rule apply; being of type $(et)t$, the subject dP can be semantically combined with the denotation of the predicate resulting in the following meaning:

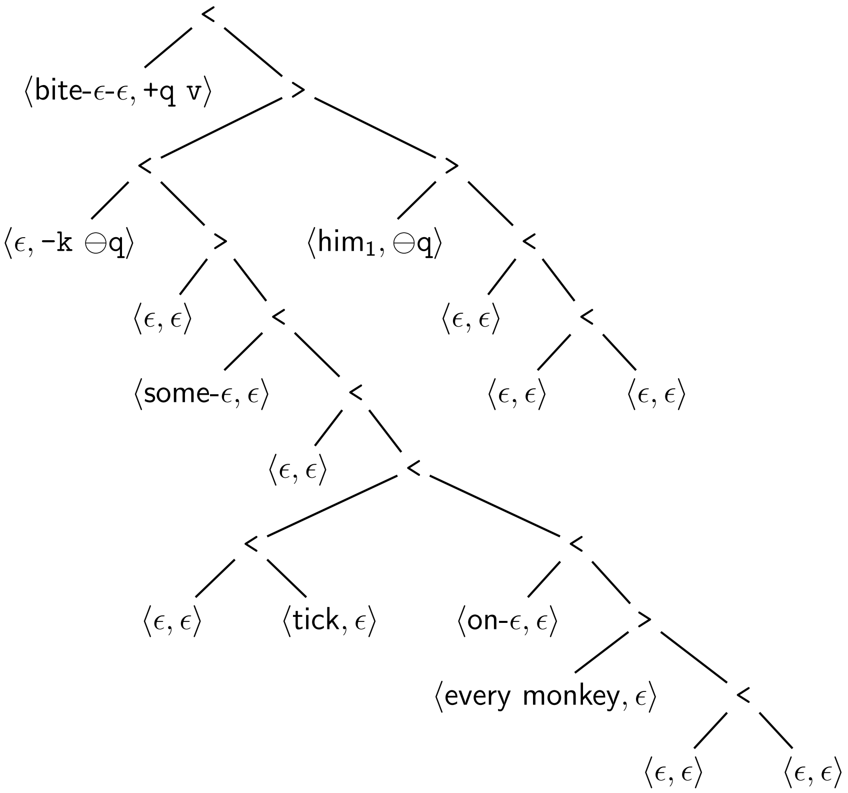
$$\begin{aligned} &\langle \textsc{combine}(\mathbf{EM} \circ \lambda_1) \circ \mathbf{ST}, \mathbf{bite(x_1)}), \emptyset \vee \emptyset \rangle \\ &= \langle ((\mathbf{EM} \circ \lambda_1) \circ \mathbf{ST})(\mathbf{bite(x_1)}), \emptyset \rangle \\ &= \langle \mathbf{EM}(\lambda_1(\mathbf{ST}(\mathbf{bite(x_1)}))), \emptyset \rangle \end{aligned}$$

The subject dP meaning can also be put into storage under the index $\textsc{k}$.

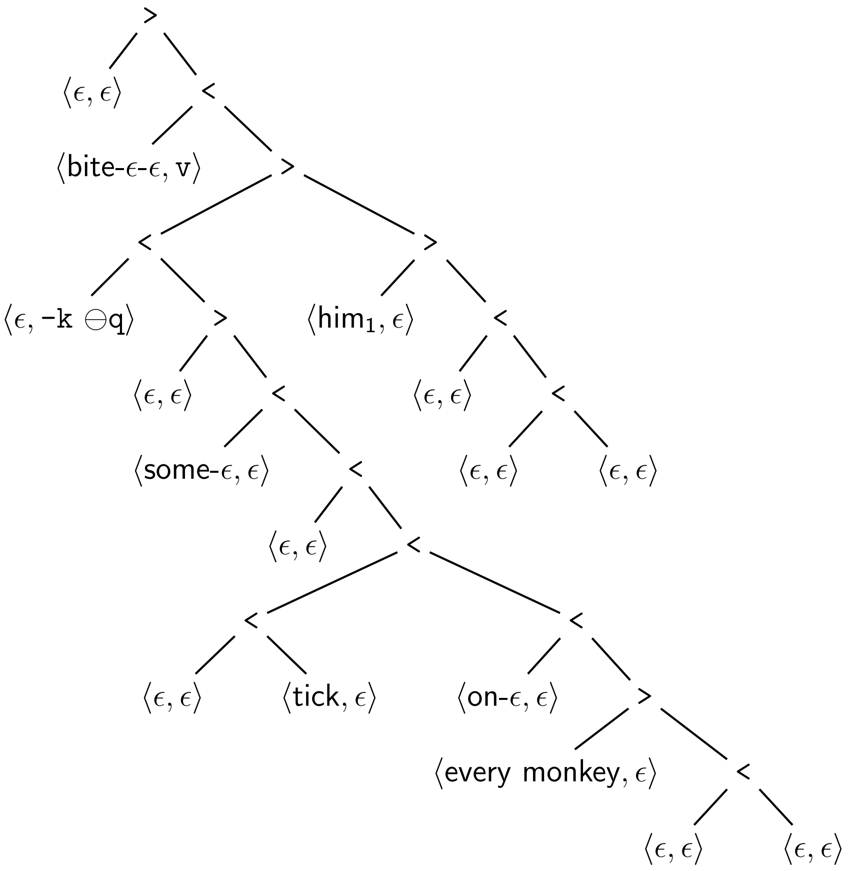
$$\begin{aligned}&\langle \mathbf{bite(x_1)(x_2)}, \emptyset \vee \emptyset \vee \{\langle \textsc{k}, ((\mathbf{EM} \circ \lambda_1) \circ \mathbf{ST}) \circ \lambda_2\rangle\}\rangle \\ &\quad = \langle \mathbf{bite(x_1)(x_2)}, \{\langle \textsc{k}, ((\mathbf{EM} \circ \lambda_1) \circ \mathbf{ST}) \circ \lambda_2\rangle\}\rangle\end{aligned}$$

As before, the expression above denotes *both* these meanings.

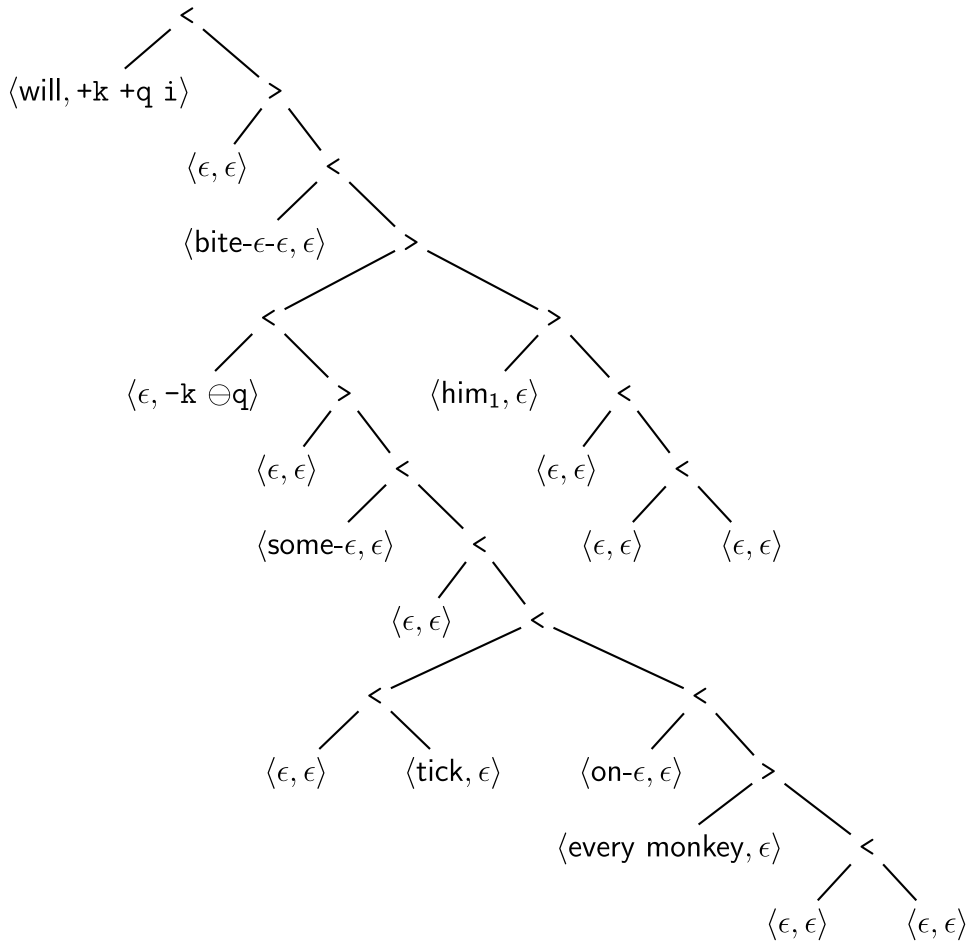
The remainder of the derivation of this reading of this sentence sentence is identical to that of the direct one. We next merge the 'QR at the vP level' lexical item, which triggers head movement of the verb; the meaning of the resulting expression is identical to that of the one above.



Next, the **move** function applies to the above. The subtree $\langle \mathsf{him_1}, \ominus\mathsf{q}\rangle$ is moved covertly to the specifier of the head of the above expression, leaving its phonetic features in situ. The first case of the **move** interpretation rule applies, but, as there was nothing in the store indexed to the $\textsc{q}$ feature, this results in no semantic contribution of the movement step; the meaning of the below expression is identical to that of the above.



Next, the above expression is merged together with the lexical item *will*. As the semantic contribution of tense is being ignored in the present discussion, the meaning of the thus derived expression is identical to that of the previous one.



The next two derivational steps move the subject dP *some tick on every monkey* to successively higher specifier positions of *will*; once overtly to check the K feature type, and once covertly to check the final ⊖q of the subject. The semantic effect of this first, K-driven movement, is to reassign the index under which the semantic object assigned to the feature type K is stored to the feature type Q. For one of the meanings denoted by this expression, this results in no change.

$$\begin{aligned}&\langle \mathbf{EM}(\lambda_1(\mathbf{ST}(\mathbf{bite}(\mathbf{x_1})))), \emptyset/\textsc{k} \vee \{\langle \textsc{q}, \emptyset(\textsc{k})\rangle\}\rangle\\ &\quad = \langle \mathbf{EM}(\lambda_1(\mathbf{ST}(\mathbf{bite}(\mathbf{x_1})))), \emptyset\rangle\end{aligned}$$

For the other of this expression's two meanings, this serves to synchronize the stored element with the syntactic phrase to which it corresponds.

$$\begin{aligned}&\langle \mathbf{bite}(\mathbf{x_1})(\mathbf{x_2}),\\ &\quad \{\langle \textsc{k}, ((\mathbf{EM} \circ \lambda_1) \circ \mathbf{ST}) \circ \lambda_2\rangle\}/\textsc{k} \vee \{\langle \textsc{q}, \{\langle \textsc{k}, ((\mathbf{EM} \circ \lambda_1) \circ \mathbf{ST}) \circ \lambda_2\rangle\}(\textsc{k})\rangle\}\rangle\\ &\qquad = \langle \mathbf{bite}(\mathbf{x_1})(\mathbf{x_2}), \{\langle \textsc{q}, ((\mathbf{EM} \circ \lambda_1) \circ \mathbf{ST}) \circ \lambda_2\rangle\}\rangle\end{aligned}$$

As before, the expression above denotes *both* these meanings. The semantic impact of the final **move** step is to retrieve the meaning stored under the index Q, reintegrating it into the meaning of the expression as a whole. For the first meaning denoted by this expression, this results in no change.

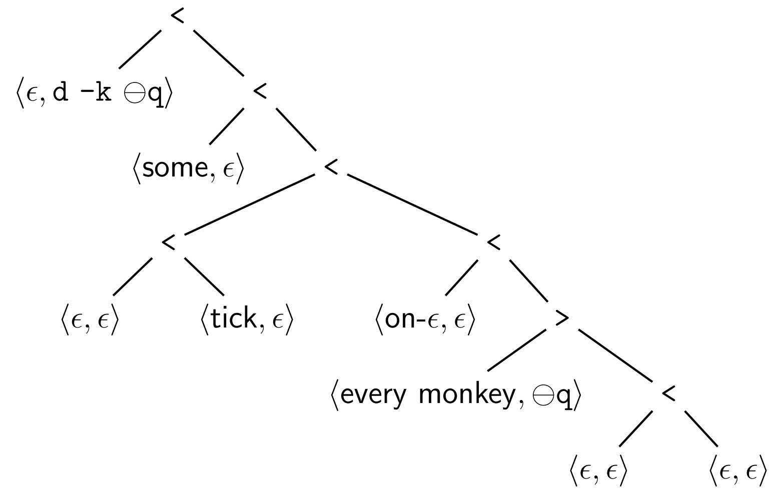
$$\begin{aligned}
&\langle \textsc{combine}(\mathbf{EM}(\lambda_1(\mathbf{ST}(\mathbf{bite}(\mathbf{x_1})))), (\emptyset \vee \{\langle \textsc{q}, \texttt{id}_t\rangle\})(\textsc{q})), \emptyset/\textsc{q}\rangle \\
&\qquad = \langle \textsc{combine}(\mathbf{EM}(\lambda_1(\mathbf{ST}(\mathbf{bite}(\mathbf{x_1})))), \texttt{id}_t), \emptyset\rangle \\
&\qquad = \langle \texttt{id}_t(\mathbf{EM}(\lambda_1(\mathbf{ST}(\mathbf{bite}(\mathbf{x_1}))))), \emptyset\rangle \\
&\qquad = \langle \mathbf{EM}(\lambda_1(\mathbf{ST}(\mathbf{bite}(\mathbf{x_1})))), \emptyset\rangle
\end{aligned}$$

The second meaning of this expression is however significantly altered by this interpretation rule.

$$\begin{aligned}
&\langle \textsc{combine}(\mathbf{bite}(\mathbf{x_1})(\mathbf{x_2}), (\{\langle \textsc{q}, ((\mathbf{EM} \circ \lambda_1) \circ \mathbf{ST}) \circ \lambda_2\rangle\} \vee \{\langle \textsc{q}, \texttt{id}_t\rangle\})(\textsc{q})), \\
&\qquad\qquad\qquad\qquad\qquad\qquad \{\langle \textsc{q}, ((\mathbf{EM} \circ \lambda_1) \circ \mathbf{ST}) \circ \lambda_2\rangle\}/\textsc{q}\rangle \\
&\qquad = \langle \textsc{combine}(\mathbf{bite}(\mathbf{x_1})(\mathbf{x_2}), ((\mathbf{EM} \circ \lambda_1) \circ \mathbf{ST}) \circ \lambda_2), \emptyset\rangle \\
&\qquad = \langle (((\mathbf{EM} \circ \lambda_1) \circ \mathbf{ST}) \circ \lambda_2)(\mathbf{bite}(\mathbf{x_1})(\mathbf{x_2})), \emptyset\rangle \\
&\qquad = \langle \mathbf{EM}(\lambda_1(\mathbf{ST}(\lambda_2(\mathbf{bite}(\mathbf{x_1})(\mathbf{x_2}))))), \emptyset\rangle
\end{aligned}$$

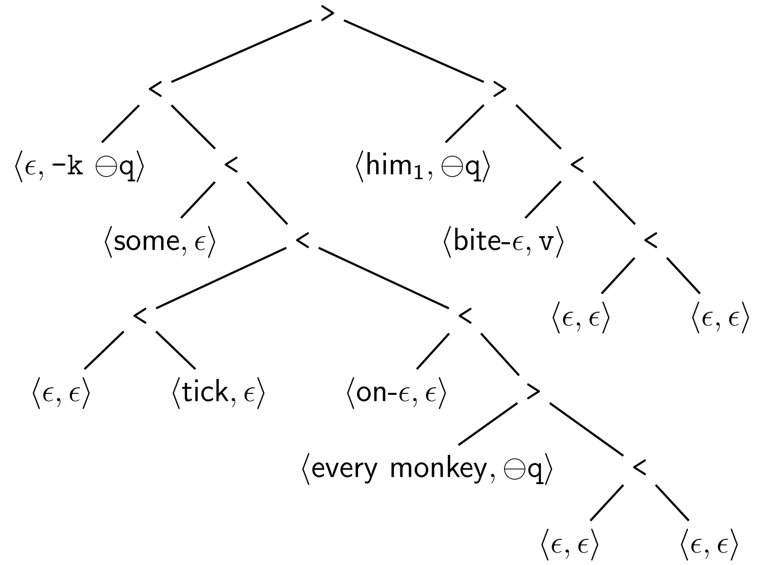
4.3.3 The Anathema Reading

Finally, we come to the question of how to block the grammatical 'overgeneration' of dPs, such as the below, in which the dP *every monkey* has not checked its ⊖q feature dP-internally.

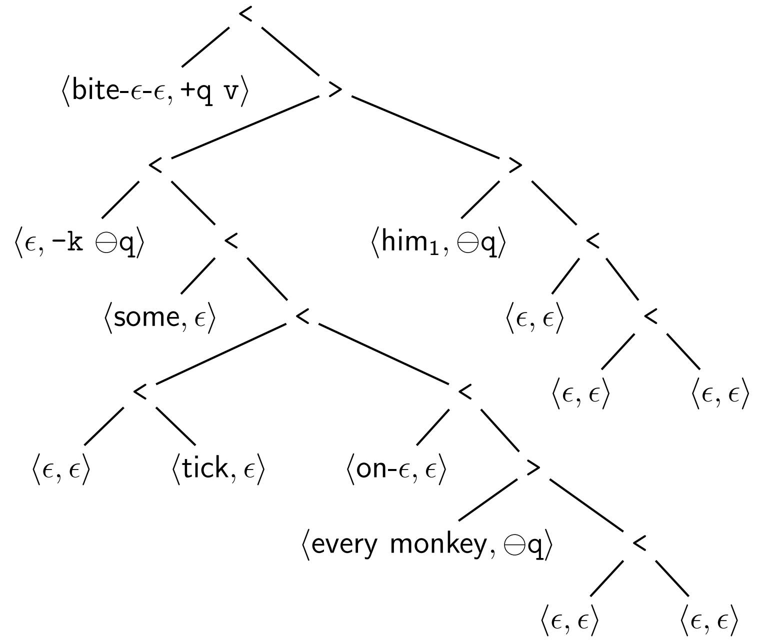


I mentioned the two basic strategies previously: either to stop the unwanted structures from being generated at all (for example, by designating dP as an island for movement), or to allow them to be generated, but to block them from being part of a successful derivation. The analysis of the verbal domain in this fragment ends up causing any attempted derivation of *some tick on every monkey will bite him* involving this dP to crash.

As before, we merge this dP with the vP *bite him*, obtaining the (well-formed) expression below.



Next, we merge the vP-level QR lexical item, triggering head-movement of the verb.



Now, however, we are stuck: the **move** operation is defined *only* on trees which have exactly one element whose first feature matches the `+y` feature of the root, but here we have two: *every monkey*, and *him*. Thus, although this expression is well-formed, it is a grammatical 'dead-end'.

5 Conclusion

We have seen that making assignment functions first class denizens of our models allows for straightforward implementation of obvious ideas about the interpretation of inverse linking constructions. In so doing, we have seen that not even inverse linking poses a challenge to the direct interpretation of minimalist grammars. It might be claimed that the incorporation of assignment functions into our models is too high a price to pay, as it makes the denotations of things ugly. Important to keep in mind when evaluating this claim is that the assignment functions were there all along, like dirt swept under the rug. By making explicit the role assignments are playing in our semantics, not only are we able to see simple solutions to otherwise perplexing problems (cf. the discussion on pages 230–235 in Heim and Kratzer (1998)), but more importantly, we are able to eliminate spurious justifications for the very serious claim that the derivation tree does not provide the right kind of structure on which to base compositional semantic interpretation (i.e. the claim that we need LF.)

References

Barker C (2002) Continuations and the nature of quantification. Natural Language Semantics 10:211–242

Bonato R (2006) An integrated compuational approach to binding theory. PhD thesis, Università degli Studi di Verona/Université Bordeaux I

Burzio L (1986) Italian syntax: A Government-Binding approach. D. Reidel, Dordrecht

Chomsky N (1995) The Minimalist Program. MIT Press, Cambridge, Massachusetts

Church A (1940) A formulation of the simple theory of types. Journal of Symbolic Logic 5(2):56–68

Gärtner HM, Michaelis J (2007) Some remarks on locality conditions and minimalist grammars. In: Sauerland U, Gärtner HM (eds) Interfaces + Recursion = Language?, Studies in Generative Grammar, vol 89, Mouton de Gruyter, Berlin, pp 161–195

Groenendijk J, Stokhof M (1991) Dynamic predicate logic. Linguistics and Philosophy 14:39–100

Harkema H (2001) Parsing minimalist languages. PhD thesis, University of California, Los Angeles

Heim I, Kratzer A (1998) Semantics in Generative Grammar. Blackwell Publishers

Hornstein N (1998) Movement and chains. Syntax 1(2):99–127

Jackendoff R (1983) Semantics and Cognition, Current Studies in Linguistics, vol 8. MIT Press

Kamp H, Reyle U (1993) From Discourse to Logic: Introduction to Model-theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory. Kluwer, Dordrecht

Keenan EL, Faltz LM (1985) Boolean Semantics for Natural Language. D. Reidel, Dordrecht

Klein E, Sag IA (1985) Type-driven translation. Linguistics and Philosophy 8:163–201

Kobele GM (2006) Generating copies: An investigation into structural identity in language and grammar. PhD thesis, University of California, Los Angeles

Koopman H (2000) Prepositions, postpositions, circumpositions, and particles: The structure of Dutch PPs. In: The Syntax of Specifiers and Heads, Routledge Leading Linguists, Routledge, London, chap 8, pp 198–255

Koopman H, Sportiche D (1991) The position of subjects. Lingua 85:211–258

Larson RK (1985) Quantifying into NP. unpublished ms.

Larson RK (1988) On the double object construction. Linguistic Inquiry 19(3):335–391

May R (1977) The grammar of quantification. PhD thesis, MIT, Cambridge, Massachusetts

May R (1985) Logical Form: Its Structure and Derivation. MIT Press, Cambridge, Massachusetts

May R, Bale A (2005) Inverse linking. In: Everaert M, van Riemsdijk H (eds) The Blackwell Companion to Syntax, vol 2, Blackwell, Oxford, chap 36, pp 639–667

Michaelis J (2001) On formal properties of minimalist grammars. PhD thesis, Universität Potsdam

Montague R (1970) Universal grammar. Theoria 36(3):373–398

Montague R (1973) The proper treatment of quantification in ordinary english. In: Hintikka J, Moravcsik J, Suppes P (eds) Approaches to Natural Language, D. Reidel, Dordrecht, pp 221–242

Montague R (1974) English as a formal language. In: Formal Philosophy: Selected Papers of Richard Montague, Yale University Press, New Haven, chap 6, pp 188–221, edited and with an introduction by R. H. Thomason

van Riemsdijk H, Huijbregts R (2007) Location and locality. In: Karimi S, Samiian V, Wilkins WK, Emonds JE (eds) Phrasal and Clausal Architecture: Syntactic Derivation and Interpretation, Linguistik Aktuell/Linguistics Today, vol 101, John Benjamins, pp 339–364

Sauerland U (2005) DP is not a scope island. Linguistic Inquiry 36(2):303–314

Stabler EP (1997) Derivational minimalism. In: Retoré C (ed) Logical Aspects of Computational Linguistics, Lecture Notes in Computer Science, vol 1328, Springer-Verlag, Berlin, pp 68–95

Sternefeld W (1997) The semantics of reconstruction and connectivity. Arbeitspapiere des SFB 340 97:1–58, tübingen